Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

APLICACIÓN DE PATRONES DE DISEÑO EN EL DISEÑO DE ARQUITECTURAS ORIENTADAS A SERVICIOS

Cristian Giovanni Martínez Rodríguez
Asesorado por el Ing. Sergio José Rodríguez Méndez

Guatemala, marzo de 2012

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

APLICACIÓN DE PATRONES DE DISEÑO EN EL DISEÑO DE ARQUITECTURAS ORIENTADAS A SERVICIOS

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

CRISTIAN GIOVANNI MARTÍNEZ RODRÍGUEZ
ASESORADO POR EL ING. SERGIO JOSÉ RODRÍGUEZ MÉNDEZ

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, MARZO DE 2012

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Murphy Olympo Paiz Recinos
VOCAL I	Ing. Alfredo Enrique Beber Aceituno
VOCAL II	Ing. Pedro Antonio Aguilar Polanco
VOCAL III	Ing. Miguel Ángel Dávila Calderón
VOCAL IV	Br. Juan Carlos Molina Jiménez
VOCAL V	Br. Mario Maldonado Muralles
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Murphy Olympo Paiz Recinos
EXAMINADOR	Ing. César Augusto Fernández Cáceres
EXAMINADOR	Ing. Juan Alvaro Díaz Ardavín
EXAMINADOR	Ing. Pedro Pablo Hernández Ramírez
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

APLICACIÓN DE PATRONES DE DISEÑO EN EL DISEÑO DE ARQUITECTURAS ORIENTADAS A SERVICIOS

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 28 de febrero de 2011.

Cristian Giovanni Martínez Rodríguez

ACTO QUE DEDICO A:

Dios	Porque me ha regalado la vida y llenado de bendiciones en cada momento.
Mis padres	Por el magnífico ejemplo que me han dado, por ser la luz que ha iluminado mi camino.
Mi patria	Los Altos por darme identidad y hacerme sentir orgulloso de mis raíces.
Mi esposa	Por estar conmigo en los momentos difíciles y convertirse en cómplice de mis logros.
Mi hijo	Por ser mi inspiración para convertirme en la persona que creé que soy.

AGRADECIMIENTOS A:

Dios	Por brindarme la oportunidad de existir.
Mis padres	Por confiar siempre y ser pilares fundamentales en el logro de mis metas.
Mis hermanos	Fernando, Erick y Pamela, por su apoyo y amor incondicional.
Mis amigos	A la memoria de Juan Rodrigo Sac (q.e.p.d.), Roberto de León, Juan Carlos Herrera, Byron Javier, Evanidia Quiñonez, Melvin Miculax, Fernando Espinoza, Mauricio Yaxcal, Juan José Hernández y Manuel Samayoa por su amistad, apoyo, paciencia y conocimientos compartidos.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

GLOSARIO

Base de datos Almacenamiento colectivo de las bibliotecas de datos que son requeridas y organizadas para cubrir los requisitos de procesos y recuperación de información.

Capa Se refiere a un segmento lógico en el diseño en el que se cumple un conjunto de reglas para todos los objetos que la componen y que ejecutan una función particular.

Caso de uso Diagrama utilizado para representar como luce para el usuario un sistema.

Cortafuegos Dispositivos de *hardware* o *software* cuyo objetivo es bloquear los accesos no autorizados a la red, permitiendo al mismo tiempo las comunicaciones autorizadas.

Enrutadores Dispositivos de *hardware* que permiten direccionar el tráfico de una red.

Infraestructura	Se refiere al conjunto de equipos o hardware destinado a ser la base sobre la que se construya una solución.
Patrón de diseño	Un patrón de diseño es una solución a un problema de diseño, cuya efectividad ha sido probada.
Protocolo	Conjunto de reglas utilizadas por computadoras para comunicarse.
SOA	*Service Oriented Architecture*. Arquitectura Orientada a servicios, estilo de arquitectura que define la utilización de servicios para dar soporte a los requisitos del negocio.
XML	*Extensible Markup Language.* Lenguaje extensible de marcas, es un estándar para intercambio de información.

RESUMEN

La evolución de la tecnología y la incorporación de nuevas tecnologías dan a las industrias nuevas posibilidades, que a su vez generan nuevas necesidades, una de estas necesidades es la de abandonar el esquema de operación en la que el *software* funciona aislado.

La necesidad es entonces de plantear un nuevo escenario en donde el *software* conoce su entorno, se integra a él y responde a necesidades tanto de un usuario como de otro componente de *software*, sin embargo, esto presenta el inconveniente de *software* desarrollado con distintas tecnologías y lograr la interacción de diferentes tecnologías con el menor costo y esfuerzo posible.

Es entonces necesario conocer la alternativa de la utilización de una arquitectura orientada a servicios para facilitar la interacción entre los distintos sistemas que forman el entorno empresarial, que puede estar compuesto por sistemas desarrollados con diferentes tecnologías y épocas; y conseguir aún la integración de los mismos en un entorno que permita formar un sistema mayor.

Por otro lado, el desarrollo de soluciones a problemas recurrentes ha permitido la definición de patrones, mismos que pueden ser aplicados con el fin de aplicar a un problema conocido una solución de antemano conocida y validada para facilitar así el desarrollo de soluciones efectivas en menor tiempo y con menor esfuerzo.

Resulta entonces posible y útil combinar los patrones de diseño y la arquitectura orientada a servicios.

OBJETIVOS

General

Mostrar de manera práctica la aplicación de los diferentes tipos de patrones en el diseño de una arquitectura orientada a servicios.

Específicos

1. Construir un marco teórico en el que se conjugue la teoría de patrones de arquitectura y diseño; y el diseño de una arquitectura orientada a servicios y las relaciones entre estos conceptos.

2. Mostrar a través del diseño de una arquitectura orientada a servicios la implementación de patrones.

INTRODUCCIÓN

A lo largo de este trabajo se genera un marco de referencia para la implementación de una arquitectura orientada a servicios con aplicación de patrones de diseño, para optimizar la implementación del modelo de arquitectura.

El desarrollo expondrá los aspectos fundamentales de la arquitectura orientada a servicios desde sus fundamentos, para posteriormente pasar a la conceptualización de uno de los modelos de arquitectura más versátiles para facilitar la interconexión entre distintas tecnologías y que cada día es adoptada por más organizaciones como es SOA.

Es importante conocer los conceptos básicos detrás de la definición de los patrones, por lo que se hace una presentación de la definición, ventajas y desventajas de los patrones de diseño.

Se presenta también una forma de clasificar los patrones para ubicarles en categorías, de acuerdo con distintos criterios, que van desde funcionalidad hasta su complejidad.

Finalmente, se presenta un caso práctico de aplicación de patrones de diseño a una arquitectura de servicios, mostrando los aspectos clave y necesarios en el diseño y conceptualización del planteamiento de la solución.

1. ARQUITECTURA Y SERVICIOS

1.1. Vista general de SOA

La Arquitectura Orientada a Servicios (SOA por sus siglas en inglés), es una arquitectura cuyo objetivo es crear una red de servicios disponibles a nivel mundial y establecer guías para integrar diferentes sistemas informáticos. Es un paradigma de arquitectura de sistemas que consiste en identificar, racionalizar y exponer servicios existentes en una unidad organizacional para su posterior reutilización; así como coordinar los servicios publicados para realizar funcionalidades más complejas.

1.2. Conceptos básicos de arquitectura

El concepto de arquitectura es un concepto que la tecnología de la información tomo como analogía de la construcción y diseño de edificios, analogía que permite distinguir claramente entre el diseño conceptual y la construcción.

La arquitectura es entonces básicamente el nivel más alto de abstracción bajo el que se define un sistema.

1.2.1. Tecnologías de arquitectura

Las tecnologías de arquitectura pueden variar, de acuerdo con lo que se está diseñando.

- Arquitectura de componentes
- Arquitectura de aplicación
- Arquitectura de integración
- Arquitectura empresarial

1.2.2. Tecnologías de infraestructura

En una empresa típica, la infraestructura representa el ambiente en el que el *software* es desarrollado, algunos componentes de infraestructura que se incluyen cuando se habla de arquitectura son:

- Servidores
- Estaciones de trabajo
- Enrutadores
- Cortafuegos

También existen algunas aplicaciones que típicamente se entienden como parte de la arquitectura por ejemplo:

- Sistemas Operativos
- Agentes de servicio
- Bases de datos y directorios
- Programas de administración de colas de mensajes
- Administradores de cuentas y aplicaciones de seguridad

La característica que poseen todos estos componentes que hacen que se consideren parte de la infraestructura, es que fueron hechos para estar disponibles para varias aplicaciones y así existir como un recurso de la empresa, un ejemplo de esto se puede ver en la base de datos.

La importancia de la arquitectura de una empresa generalmente determina el potencial de procesamiento de las tecnologías que se construyan sobre esta, por lo tanto, un programa de *software* tiene como límite inherente los límites de la infraestructura y arquitectura que lo soporta.

1.2.3. *Software*

Un programa de *software* en relación a tecnología de arquitectura puede ser entendido como una implementación del diseño documentado en una especificación de arquitectura en el que se imprime la lógica y se ejecuta con la ayuda del ambiente definido por la tecnología de infraestructura.

Existen diferentes estilos de arquitectura que se utilizan para soportar el *software*. Un estilo de arquitectura es un conjunto de principios, que proveen un marco de trabajo abstracto para una familia de sistemas, cada estilo define un conjunto de reglas que especifican que clase de componentes se pueden utilizar y los supuestos sobre cómo poner a trabajar juntos estos componentes. Un estilo de arquitectura promueve la reutilización del diseño, brindando soluciones a problemas frecuentemente recurrentes.

Los estilos de arquitectura pueden ser organizados por su área de enfoque, en la tabla siguiente se muestra, de acuerdo al área el estilo de arquitectura correspondiente.

Tabla I. **Categoría y estilos de arquitectura**

Categoría	Estilo de arquitectura
Desarrollo	Cliente/servidor, 3 capas, N-capas
Estructura	Basada en componentes, orientada a objetos, arquitectura de capas, modelo vista controlador (MVC).
Dominio	Modelo de dominio, Gateway
Comunicación	Orientada a Servicios (SOA), bus de mensajes, tuberías y filtros.

Fuente: elaboración propia.

Estos son solo algunos de los estilos, incluso hay situaciones en las que se pueden mezclar algunos estilos para ajustarse a los requerimientos. Sin embargo, hay algunas que a esta fecha son claves y tienen un mayor auge.

Tabla II. **Estilos de arquitectura compuestos**

Estilo de arquitectura	Descripción
Cliente servidor	Divide el sistema en dos aplicaciones, donde el cliente hace una petición de servicio al servidor.
Arquitectura basada en componentes	La ampliación se diseña en componentes funcionales o lógicos reutilizables que tienen interfaces de comunicación bien definidas.
Arquitectura de capas	Divide los objetivos de la aplicación en grupos apilados (capas)
Bus de mensajes	Un *software* que puede recibir y enviar mensajes que están basados en un conjunto de formatos conocidos con antelación de manera que los sistemas pueden comunicarse unos con otros sin necesidad de conocer al destinatario actual.
Modelo vista controlador	Separa la lógica del manejo e interacción del usuario
N-*tiers*/3-*tiers*	Separa la funcionalidad en segmentos muy parecido a la arquitectura de capas, sin embargo, cada segmento aquí se separa a una computadora diferente.
Orientada a objetos	Es una arquitectura basada en la división de tareas en objetos individuales y reutilizables que contienen la información y el comportamiento más relevante del objeto
Arquitectura Orientada a Servicios(SOA)	Se refiere a las aplicaciones que exponen y consumen funcionalidad por medio de servicios y con el uso de contratos y mensajes.

Fuente: elaboración propia.

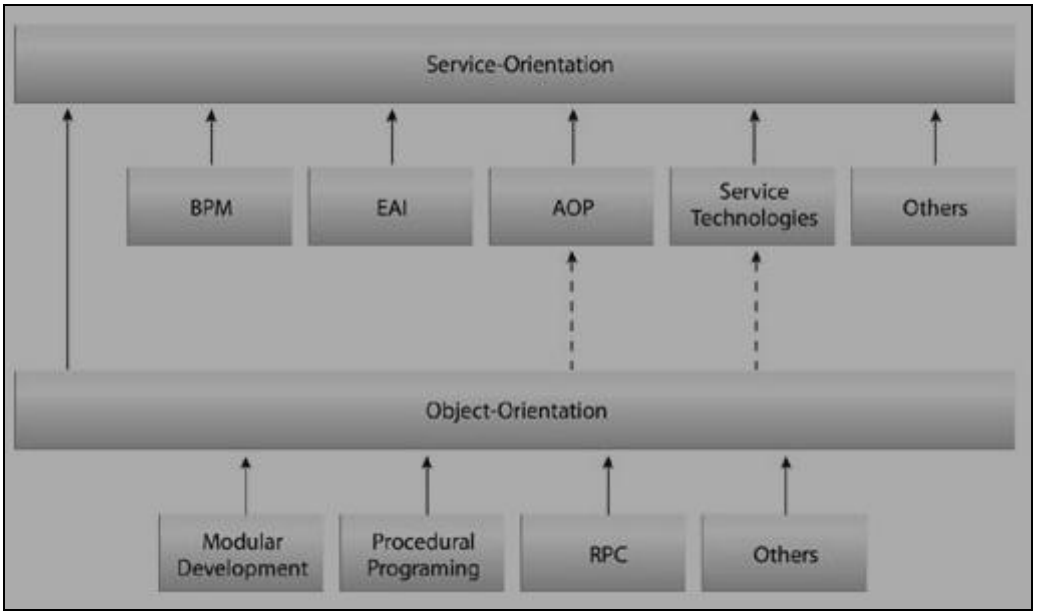
1.3. Computación orientada a servicios

La computación orientada a servicios, es un término que se aplica a una nueva generación de plataformas de computación distribuida que incluyen muchas cosas como su propio paradigma y principios de diseño, patrones de diseño y un modelo de arquitectura diferente.

La computación orientada a servicios utiliza como base plataformas existentes y agrega nuevas capas de diseño, conceptos de administración y un completo *set* de tecnologías de implementación muchas de las cuales están basadas en marcos de trabajo de servicios *web*.

Este paradigma de orientación a servicios tiene como finalidad la construcción de una solución a partir de componentes unitarios, que pueden ser en conjunto o individualmente utilizados para dar soporte a los objetivos y beneficios asociados con la arquitectura orientada a servicios, una solución lógica diseñada con orientación a servicios puede ser calificada como orientada a servicios y una unidad de esta orientación a servicios es nombrada servicio. Algunas veces el paradigma de orientación a servicios se compara con el de orientación a objetos, al respecto el paradigma de orientación a servicios es una evolución del paradigma de orientación a objetos.

Figura 1. **Evolución del paradigma orientado a objetos orientado a servicios**



Fuente: elaboración propia.

1.4. Medios de implementación de servicios

SOA es un modelo de arquitectura, independiente de la plataforma tecnológica que se utilice para su implementación, esto da a la industria la libertad de perseguir constantemente sus objetivos estratégicos asociados con SOA y la orientación a servicios y aprovechar los avances tecnológicos, actualmente, un servicio puede ser implementado de una de las siguientes maneras:

- Componente
- Servicio *web*
- Servicio REST

Básicamente, cualquier forma de implementación puede utilizarse para crear un sistema distribuido que puede ser orientado a servicios.

1.4.1. Implementación de servicios como componente

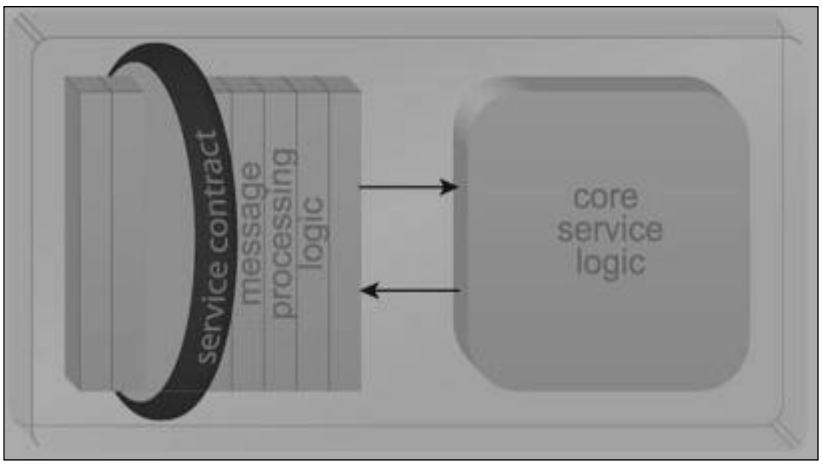
Un componente se define como un programa de *software* que es diseñado para formar parte de un sistema distribuido. Este componente brinda una interface comparable con una interface tradicional de una aplicación, por medio de la cual expone sus funcionalidades como métodos, que podrán ser llamados por otros programas

1.4.2. Implementación de servicios cómo servicios *web*

Un servicio *web* es una unidad lógica que proporciona un contrato físicamente desacoplado para quien lo contrata, el contrato está definido en WSDL y puede tener uno o varios esquemas definidos en XML.

El contrato del servicio *web* muestra las capacidades de operación como operaciones, lo que se logra con esto es una interfaz independiente de los marcos de trabajo o *frameworks.*

Figura 2. **Arquitectura de un servicio *web***



Fuente: elaboración propia.

1.4.3. Servicios REST

Estos servicios permiten construir sistemas distribuidos basados en recursos, al ser programas ligeros que se diseñan haciendo hincapié en la simplicidad, escalabilidad y usabilidad.

1.5. Arquitectura global para servicios *web* XML (GXA)

GXA define un marco de trabajo para los servicios *web*, es un conjunto de tecnologías cuyo objetivo es hacer servicios *web* que faciliten la integración de distintas plataformas sobre Internet, estos componentes han sido desarrollados con base en estándares brindados por Microsoft, IBM y otras compañías.

El objetivo de GXA espera por medio de la estandarización, generar servicios *web* con la capacidad de vencer los obstáculos que suponen diferentes Sistemas Operativos, infraestructuras y arquitecturas de *software*.

GXA nació a partir de un artículo desarrollado conjuntamente entre IBM y Microsoft, liberado en abril de 2001 en el taller sobre servicios web organizado por el consorcio de la *World Wide Web* o W3C, el artículo nombrado *"Web Services Framework"*, definió funciones específicas que deben ser consideradas si la integración de aplicaciones estará basada en servicios web.

En dicho artículo se incluyeron consideraciones sobre:

Seguridad: garantizar el origen y contenido del mensaje

Transacciones: la capacidad de asegurar la integridad de las bases de datos subyacentes incluso cuando hay concurrencia de los servicios *web*.

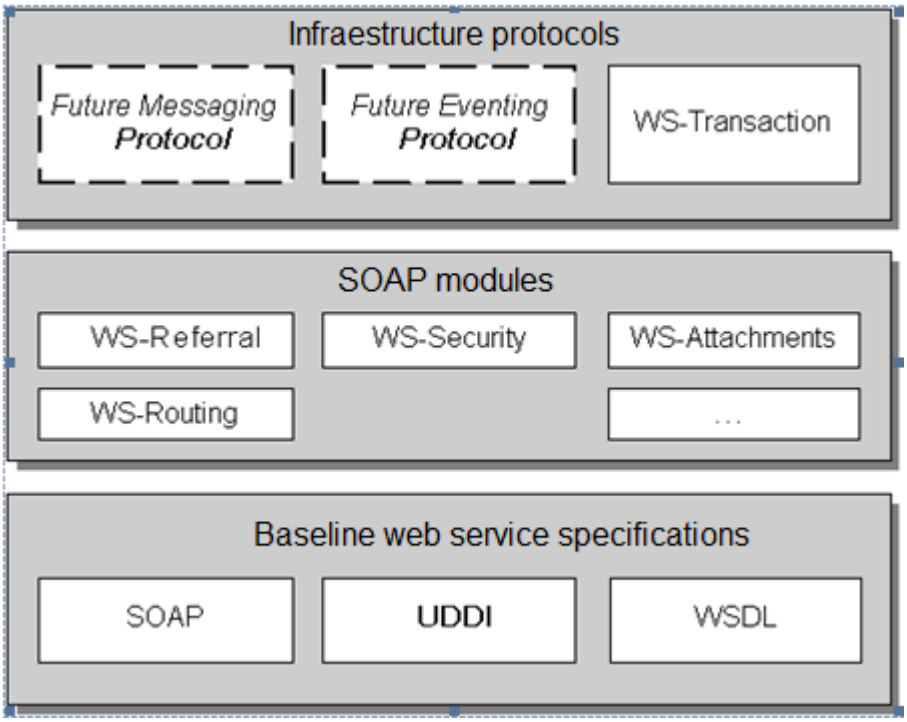
Coordinación: en referencia a la capacidad de describir y definir el flujo de datos entre múltiples servicios *web*.

Confiabilidad de la mensajería: en referencia a la garantía de que un mensaje es actualmente enviado al destinatario correcto.

Ruteo y referencia: en este bloque hacían referencia a la capacidad de especificar la ruta para el mensaje.

Adjuntos: es la capacidad de incluir datos binarios adicionalmente en el mensaje enviado.

Figura 3. **Diagrama de arquitectura global de servicios *web* (GXA)**



Fuente: elaboración propia.

1.6. Especificaciones de servicios *web*

Para definir servicios *web* existe una gran variedad de especificaciones, en general estas especificaciones se refieren a un aspecto particular del servicio *web*, generalmente se complementan aunque hay algunas que incluso compiten con otras.

Entre los aspectos para los que se han definido estándares están:

1.6.1. XML

El lenguaje extensible de marcas, consiste en un texto simple basado en un formato para representar información de manera estructurada, actualmente existen las siguientes especificaciones.

- XML Canónico
- Intercambio eficiente de XML
- Internacionalización de XML
- XML relacionado con otros formatos

1.6.2. Protocolos

Un protocolo es un conjunto de reglas que definen la manera en la que se establecerá la comunicación, es decir, un protocolo ya representa un estándar en sí mismo, algunas de las especificaciones más importantes son:

- HTTP
- SOAP
- Direccionamiento a servicios *web* 1.0
- Direccionamiento a servicios *web* 1.0 SOAP
- Arquitectura de servicios *web*

1.6.3. Descripción del servicio

Una parte importante de un servicio *web* es poder ser consumido y para esto es imprescindible poder describir las operaciones y la semántica del uso de los parámetros, las especificaciones existentes en relación a la descripción son:

- WSDL
- Coreografía de servicios *web*
- Políticas de servicios *web*
- Anotaciones semánticas para WSLD y esquema XML
- Lenguaje de modelado de servicio
- Acceso a los recursos de los servicios *web*

1.6.4. Seguridad

Bajo ciertas condiciones es necesaria la creación de un ambiente seguro para el manejo de la información en el servicio *web*. Para asegurar el XML se pueden utilizar firmas digitales, estas brindan servicios de integridad o autenticación para cualquier tipo de dato que se incluya en el XML que está firmado.

La encriptación es otra manera de asegurar que la información enviada por el remitente solo podrá ser interpretada o entendida por el destinatario.

Existen especificaciones disponibles para definir:

- Encriptación XML
- Firmas XML
- XKMS

1.6.5. Internacionalización

Cuando el servicio debe ser utilizado en distintas regiones del mundo con distintas culturas o lenguajes, es necesario asegurar que el servicio funcionará de la manera esperada, para esto existen especificaciones tales como:

- *Legacy extended IRIs for XML resource identification*
- *Working with Time Zones*
- *Web services Internationalization Usage Scenarios*

1.7. Categorías de servicios *web*

Los servicios *web* se clasifican de acuerdo a su función en siete categorías.

Integración de aplicaciones empresariales

- Mapeo *web*

- Especificación de servicios *web*
- Proveedores de servicios *web*
- Mensajería instantánea
- Búsqueda instantánea
- Sistemas Operativos *web*

2. ARQUITECTURA ORIENTADA A SERVICIOS

2.1. El método de orientación a servicios

Cuando se diseña una arquitectura orientada a servicios, se debe buscar que cada componente de la solución sea diseñado consistentemente de manera que se adapte al ambiente en el que se implementa.

Los principios de la orientación a servicios sirven como directrices en el diseño de la solución.

El paradigma de orientación a servicios tiene ocho principios cada uno dirigido a un aspecto singular del diseño.

- Contratos de servicio estandarizado, es decir, servicios con parámetros de diseño similares, deben cumplir con un contrato estándar.

- Bajo acoplamiento de servicio: los contratos de servicios deben imponer al consumidor un nivel bajo de requerimientos para su acoplamiento y ellos mismos ser desacoplados de su entorno.

- Servicio de abstracción: los contratos de servicios únicamente deben contener información limitada a lo que se publica en los contratos de servicio.

- Servicio de reusabilidad: los servicios deben contener y expresar la lógica agnóstica de su funcionalidad y así poder ser un componente reutilizable.

- Servicio de autonomía: los servicios deben ejercer un alto nivel de control en tiempo de ejecución del ambiente en el que se ejecuta.

- Servicio Statelessness: los servicios deben minimizar el consumo de recursos aplazando la gestión de la información de estado.

- Servicio de detectabilidad: los servicios deben ser dotados con la metadata que le permita a cada uno ser efectivamente detectado e interpretado.

- Servicio de componibilidad: los servicios deben poder ser utilizados efectivamente para composición sin importar el tamaño o la complejidad de la composición.

2.2. Características de SOA

Una vez definido el fundamento de la orientación a servicios, conviene definir las características que convierten los elementos del método en una arquitectura, hay cuatro características que se deben mantener en cualquier implementación de SOA.

- Orientada al negocio: esto quiere decir que la tecnología de la arquitectura debe estar alineada con la arquitectura actual del negocio, lo que implica que la arquitectura debe evolucionar en el tiempo tal y como lo hace el negocio.

- Proveedor neutral: el modelo de arquitectura no debe estar basado en una plataforma de un vendedor específico, debe permitir que tecnologías de diferentes vendedores se combinen.

- Centrado en la empresa: al estar centrado en la empresa la arquitectura alcanza la representación significativa de un segmento de la empresa, permitiendo la reutilización y composición de servicios de esta manera se permite tener soluciones orientadas a servicios.

- Centrado en composición: una arquitectura orientada a servicios es inherente su capacidad de soportar los mecanismos de agregación, facilitando así el ensamblaje de nuevos componentes por medio de la composición de servicios.

2.3. Tipos de arquitectura orientada a servicios

Cada programa se diseña con una arquitectura base, en el caso de la arquitectura orientada a servicios existen al menos cuatro variantes ó tipos más comunes de implementaciones.

- Servicios simple

- Composición de servicios

- Inventario de servicios

- Orientada a servicios empresarial

2.3.1. Arquitectura de servicio

La arquitectura de servicios generalmente tiende a ser amplia, porque un servicio puede pertenecer a otras cosas, incluso formar parte de múltiples componentes.

No es común que se documente por separado la arquitectura de cada componente en las aplicaciones distribuidas tradicionales. Sin embargo, la importancia de que los servicios sean independientes y altamente autosuficientes y autocontenidos requieren que cada cual sea diseñado por separado.

- Ocultar información: para ajustarse al principio de diseño de abstracción de servicio, estos deben ser diseñados de manera que encapsulen la información que poseen y únicamente expongan la información requerida por terceros.

- Contrato de servicios: una parte medular de la arquitectura orientada a servicios es el contrato técnico, usualmente este es la primera parte de un servicio antes de ser entregado. Las capacidades expresadas en el contrato deben dictar el alcance y la naturaleza de la lógica subyacente y los requerimientos de los procesos para lograr su implementación.

- Agentes de servicios: un aspecto importante en la infraestructura relacionada con el diseño de servicios son las dependencias que pueden haber con los agentes de servicio, por ejemplo en el manejo por eventos los programas son capaces de interceptar y procesar mensajes enviados hacia y desde el servicio de manera trasparente. Los agentes de servicio pueden ser desarrollados a medida o proveídos por el ambiente en el que se consumen los servicios. Dentro de la arquitectura los agentes pueden ser identificados por medio de la información de ejecución. Los agentes de servicio pueden tener su propia especificación de arquitectura que puede hacer referencia a la arquitectura de servicios.

- Habilidades del servicio: una consideración clave con la arquitectura de servicio, es el hecho de que la funcionalidad ofrecida por un servicio se encuentra en una o más capacidades individuales. Cada habilidad del servicio debe encapsular su propia pieza de lógica, aunque la lógica subyacente puede estar organizada por sí misma y modularizada de manera diferente que permita a otras habilidades compartir rutinas básicas o podrían algunas de estas habilidades expuestas representar o necesitar acceso a uno o más recursos incluyendo otros servicios.

Entonces las habilidades individuales pueden ser tan complejas que podrían necesitar un diseño individual que debe ser diseñado por separado.

2.3.2. Arquitectura composición de servicios

El propósito fundamental de publicar una serie de servicios independientes es que estos puedan ser combinados para conformar servicios más complejos, que resuelvan problemas igual de complejos para el negocio, en donde cada servicio simple tiene su propia arquitectura. En este modelo fundamentalmente sincroniza las arquitecturas de los servicios que se involucran en la solución. En este modelo existen dos roles básicos que son el controlador de composición y los servicios compuestos.

Una arquitectura de composición puede compararse con una arquitectura tradicional, sin embargo, esta comparación podría ser válida únicamente en el marco de la composición y orquestación de los servicios para formar la nueva solución debido a que de los servicios involucrados únicamente se conoce el contrato.

Otra característica singular de la arquitectura de composición de servicios es que una composición podría estar formada por composiciones anidadas.

- Composición e infraestructura: una arquitectura de composición es fuertemente dependiente de las características de administración subyacentes en el ambiente de ejecución que aloja a los servicios involucrados en la solución. La seguridad, administración de transacciones, confiabilidad de mensajería así como el soporte para el ruteo de mensaje podrían tener un lugar en la especificación de la arquitectura.

2.3.3. Arquitectura inventario de servicios

Un inventario de servicios es una colección de servicios independientemente estandarizados y controlados, servicios que son expuestos con un límite arquitectónico predefinido. Esta colección representa significativamente el alcance excede el límite de proceso de cada proceso de negocio individualmente.

El alcance y el límite de una arquitectura de este tipo puede variar. Idealmente el inventario de servicios es primero modelada conceptualmente, dirigiéndola a la creación de un borrador de la arquitectura, es incluso este borrador el que muchas veces termina definiendo el alcance mismo de la arquitectura.

Desde la perspectiva de una empresa, el inventario de servicios puede verse como un límite concreto para la implementación de una arquitectura estandarizada. Esto es porque los servicios incluidos en un inventario están estandarizados, entonces también lo están las tecnologías y extensiones proporcionadas por las arquitecturas subyacentes.

2.3.4. Arquitectura orientada a servicio empresarial

Este tipo de arquitectura básicamente representa todos los servicios, la composición de servicios y el inventario de servicios, todos los tipos de arquitecturas. Este tipo se puede comparar con una arquitectura tradicional solo cuando la mayoría de los ambientes son orientados a servicios, de otra manera esto sería simplemente documentación de las partes de una empresa que ha adoptado SOA.

La arquitectura orientada a servicios empresariales puede ir más allá de establecer estándares y convenciones para todos los servicios y puede también requerir ser referenciada en las especificaciones de arquitectura correspondiente. Un punto relevante de adoptar SOA es que deben estar sincronizados la tecnología y la arquitectura del negocio de una empresa.

- Arquitectura tipos y herencia: el ambiente y las convenciones establecidas por una empresa son llevados en la implementación de cada servicio, que puede residir en un ambiente empresarial simple. Estos introducen nuevos y específicos elementos de arquitectura tal es el caso de plataformas en donde se ejecutan. El resultado final de esto es la formación de un modelo de arquitectura que se hereda.

- Otros tipos y formas de arquitectura orientada a servicios: la arquitectura entre negocios es una arquitectura que extienden las empresas más allá de sus límites permitiendo orquestar distintos ambientes con convenciones de diseño incompatibles.

- Otra forma sería la arquitectura de comunidad orientada a servicios: con la aparición de la estandarización para el intercambio de datos y el crecimiento del mercado de servicios de terceros, la opción es definir una arquitectura orientada a servicios dedicada a colaboración a través de miembros de la comunidad.

2.4. El resultado final de la orientación a servicios

La orientación a servicios está basada en cuatro puntos básicos, que son:

- Los servicios deben tener fronteras claras: cuando una solución está compuesta por servicios distribuidos geográficamente en distintas ubicaciones, algo que no debe tomarse a la ligera es el costo de desplazarse de un lugar a otro, por lo que este hecho debe reducirse cuanto sea posible, como resultado de esta situación, SOA se basa en un modelo de intercambio de mensajes explícito que es útil para mantener el área del servicio tan pequeña como sea posible, por eso se puede concluir que la simplicidad y generalización en los servicios no es un lujo sino una característica crítica del servicio.

- Servicios autosuficientes: este punto tiene que ver con dos características básicas de los servicios la ubicuidad, que es la capacidad de cada servicio de ser encontrado y explorado y el manejo de sus propias funciones como en el caso de las excepciones y errores que debe manejar el servicio.

- Los servicios no comparten clases y tipos sino esquemas y contratos: a diferencia de la programación orientada a objetos SOA requiere que la exposición de la funcionalidad y la interacción con los servicios se haga a través de esquemas que definen lo requerido y proporcionado por los mismos así como contratos que delimiten el campo de acción de cada servicio, para facilitar la interacción entre servicios. Por regla general resulta imposible propagar cambios en un esquema o contrato a todas las partes que han consumido alguna vez un servicio, por esto generalmente se extienden en vez de cambiarse las interfaces existentes.

- La compatibilidad de los servicios está basada en políticas: cada servicio debe advertir de sus capacidades y requerimientos de forma clara y entendible para el sistema que le consume, las políticas establecen condiciones y garantías, estas deben habilitarse para soportar el funcionamiento normal del servicio, generalmente se definen de forma declarativa por medio de algún estándar.

Por otro lado las comunidades de negocios automatizados y la industria de tecnología de información tienen una relación de negocios en la que se influyen mutuamente, la comunidad de negocios provee requerimientos y la comunidad de TI genera nuevas ideas para aportar soluciones a dichos requerimientos. La orientación a objetos y la integración de aplicaciones empresariales son muestras de esta relación, cada solución de estas finalmente incide en la formación de distintos requerimientos de tecnologías de arquitectura. La arquitectura orientada a servicios finalmente acontece de la misma forma, la plataforma establece la posibilidad de conseguir importantes objetivos estratégicos.

La arquitectura orientada a servicios finalmente tendrá como resultado componentes y soluciones con las siguientes características:

- Aumento de la interoperabilidad
- Aumento de la federación
- Aumento de la posibilidad de diversificación de proveedores
- Incremento de la alineación entre el negocio y el dominio de la tecnología
- Aumento del retorno de inversión ROI por sus siglas en inglés
- Una organización más ágil
- Reducción de la carga para IT

Es este el estado final que se busca conseguir al momento de adoptar la orientación a servicios.

.

3. PATRONES

3.1. Vista general de los patrones

Los patrones se clasifican de acuerdo a la naturaleza de su diseño, ya sea para cumplir con una funcionalidad o para conformar una arquitectura con alguna orientación, a partir de su incorporación en el diseño se han desarrollado mejores prácticas y elementos claves en el diseño.

3.1.1. Patrones

La Real Academia de la Lengua Española define patrón como "modelo de muestra que sirve para sacar otra cosa igual". Y los patrones de diseño de *software* son soluciones probadas a una serie de circunstancias particulares que definen un escenario en el que se puede aplicar esa solución.

Los patrones de diseño son útiles porque:

- Representan soluciones probadas a ciertos problemas, es decir, evitan la búsqueda de soluciones a problemas recurrentes.
- Promueven la estandarización en el diseño de soluciones
- Pueden usarse para garantizar la consistencia en la forma en la que los sistemas son diseñados y construidos

Por otro lado, el uso de los patrones de diseño no es obligatorio en el diseño de una solución, solo son una herramienta útil que puede facilitar el diseño, es importante remarcar que en el uso de los patrones de diseño como en todo diseño en el que se involucra la abstracción, el nivel en el que se dejen de aplicar los patrones depende de la experiencia del arquitecto, esto debido a que el abuso en el uso de los patrones de diseño puede tener un efecto contrario al deseo convirtiendo una solución en algo difícil o impráctico de implementar.

3.1.2. Reseña histórica

En 1979 el arquitecto Christopher Alexander, publicó un libro en el que presentaba una serie de patrones cuya finalidad era construir edificios de una mayor calidad en un menor tiempo el título de aquel libro era *The Timeless Way of Building*.

Esta como muchas otras veces fue un enfoque tomado por la tecnología de la información y adaptado al desarrollo de *software* en 1987 Ward Cunningham y Kent Beck usaron alguna de las ideas de Alexander y publicaron un artículo con cinco patrones hombre-ordenador, sin embargo, fue hasta en los noventas cuando los patrones de diseño tuvieron éxito en la informática luego de la publicación del libro *Design Patterns* escrito por el grupo autodenominado Gang Of Four (GoF) formado por Richard Helm, Erick Gamma, Ralph Johnson y John Visides, en el que se mostraban veintitrés patrones de diseño.

3.1.3. Clasificación de patrones

Los patrones pueden clasificarse según el nivel de abstracción en el que se ubiquen.

- Patrones de arquitectura

- Patrones de diseño de *software*
  - Patrones creacionales
  - Patrones estructurales
  - Patrones de comportamiento
  - Patrones de interacción

Patrones de arquitectura

Estos patrones aportan soluciones en la definición de la organización estructural para soportar el *software* y sus operaciones.

Patrones de diseño de *software:* estos patrones definen soluciones a problemas de la construcción del *software* y a su vez se clasifican, de acuerdo con su finalidad.

3.2. Notación de patrones

A lo largo de este trabajo los diagramas se expresarán por convención en lenguaje unificado de modelado UML por sus siglas en inglés, el detalle y explicación de la representación de cada gráfico de UML escapa al alcance de este trabajo, dejando únicamente los conceptos e ideas básicas detrás de UML.

UML: por sus siglas en inglés cuya traducción al español es Lenguaje Unificado de Modelado, es una familia de notaciones gráficas que facilitan la descripción y diseño de los sistemas de *software.*

A inicios de los ochenta, cuando algunos empezaron a pensar en la programación orientada a objetos también empezaron a pensar en un lenguaje gráfico de diseño orientado a objetos.

Grady Booch, Peter Coad, Ivar Jacobson, Jim Odell, Jim Rumbaugh son algunos de los nombres de los principales directores de proyectos enfocados en proyectos de este tipo que se desarrollaron entre 1988 y 1992. Todos los métodos eran similares y los mismos conceptos aparecían en diferentes notaciones, durante este período de tiempo no había un estándar. Cuando Jim Rumbaugh se unió a Grady Booch de Rational, parte de IBM, esta alianza parecía que podría tomar una porción importante del mercado, lo que propició algunas alianzas anti Rumbaugh-Booch en 1995 se les unió Ivar Jacobson, a quienes se les otorga el crédito de la creación de UML.

Finalmente UML es una especificación del *Object Managment Group*, la especificación más actualizada puede descargarse en el sitio www.uml.org.

3.3. Perfiles

Cuando se habla de la definición de un patrón es conveniente hacerlo en un formato que si bien no es estándar en la industria permita entender rápidamente el patrón en cuestión, para esto al definir cada patrón se debe tomar en consideración la inclusión mínima de la siguiente información:

- Requerimiento que atiende: el requerimiento consiste en una sentencia simple que muestre el origen del patrón en forma de pregunta.

- Sumario: consiste en una tabla en la que se muestra un grupo de sentencias que proporcionan una referencia rápida del propósito del patrón, en esta parte se muestra una sinopsis del problema, solución, aplicación e impacto.

- Problema: en esta sección se describe la cuestión que genera el problema, se describen sus efectos, este es el caso para el que el patrón aportará una solución.

- Solución: representa la solución de diseño que se propone para resolver el problema y cumplir con el requerimiento, generalmente consiste en una sentencia corta y seguida por un diagrama que muestra la solución final, sin embargo, en este no se muestra el detalle del ¿cómo?

- Aplicación: esta parte contiene el ¿cómo? en referencia a la aplicación del patrón, puede incluir líneas guía, detalles de implementación e incluso sugerir procesos y mejores prácticas.

- Impacto: generalmente un patrón tendrá ventajas y desventajas, por lo que será importante poder remarcar las consecuencias más comunes de su implementación, costo y requerimientos asociados si los hubiera.

- Relaciones: el uso del diseño de patrones puede conllevar interacción entre los aspectos del diseño y la arquitectura, es importante entender los requerimientos que tiene un patrón, aunque no es posible diagramar todas las relaciones que surgen de manera directa o indirecta.

3.4. Consideraciones clave en el diseño

Es importante que el diseño de patrón tenga un fundamento sólido es decir que la implementación de un patrón debe estar basada en los principios de diseño elementales, a continuación se encuentran algunas definiciones importantes de la relación entre los patrones y los principios de diseño.

- Los principios de diseño se aplican a la solución lógica con el objetivo de que el modelado de la solución fomente las características de diseño claves que soportan los objetivos estratégicos asociados con la orientación a servicios.

- El diseño de patrones provee soluciones a problemas comunes encontrados cuando se aplican los principios de diseño y cuando se establece un ambiente adecuado para su implementación y de acuerdo con los principios de la orientación a servicios.

Finalmente la mejor manera de ver la relación entre los patrones y los principios de diseño será entender que los patrones de diseño brindan el soporte para que los objetivos de los principios de diseño se cumplan.

3.4.1. Principios de diseño

Los principios de diseño para soluciones de arquitectura orientada a servicios.

- Reusabilidad: todos los servicios deben ser reutilizables, cada servicio debe planearse para que pueda ser reutilizado en la aplicación.

- Formalidad contractual: los servicios deben definir contratos formales que indiquen claramente la funcionalidad que ofrecen y la manera en la que interactúan.

- Bajo acoplamiento: los servicios deben ser bajamente acoplados es decir, deben diseñarse siendo independientes de otros servicios.

- Composición: siempre debe diseñarse favoreciendo la composición sobre la herencia, esto quiere decir que los servicios deben diseñarse de manera que puedan formar parte de servicios más grandes.

- Autonomía: cada servicio debe ser diseñado considerando su propio ambiente de ejecución.

- Sin estados: los servicios no deben almacenar estados para favorecer la composición y evitar inconsistencias en su funcionamiento.

- Descubiertos: cada servicio debe descubrirse para ser utilizado con esto se evitará duplicar servicios con la misma funcionalidad.

3.5. Patrones de diseño

Los patrones de diseño se clasifican de acuerdo a criterios relacionados con su funcionalidad y diseño, tal es el caso del alcance, disciplina, propósito, granularidad y dominio.

3.5.1. Clasificación de patrones

El aumento del número de patrones disponibles hace necesario clasificarlos de alguna manera, incluso existen distintas maneras de clasificarlos, la clasificación se hace a partir de las propiedades que comparten, que tampoco es un estándar, sin embargo, algunos criterios de agrupación pueden ser: estructura, función o aplicación, dependiendo de ese criterio se puede definir un esquema de clasificación.

Los esquemas de clasificación pueden tener varias dimensiones, un mismo patrón puede tener diferentes clases de atributos que le permitan ser incluido en más de una categoría. Para tener una clasificación más eficiente, se creé que un patrón no debería pertenecer a más de dos categorías y resulta clave enfocarse en los atributos claves del patrón para lograr una mejor clasificación.

Propiedades de los esquemas de clasificación: el objetivo principal de clasificar los patrones debe ser facilitar la elección del patrón correcto.

Un esquema de clasificación adecuado debe tener las siguientes características:

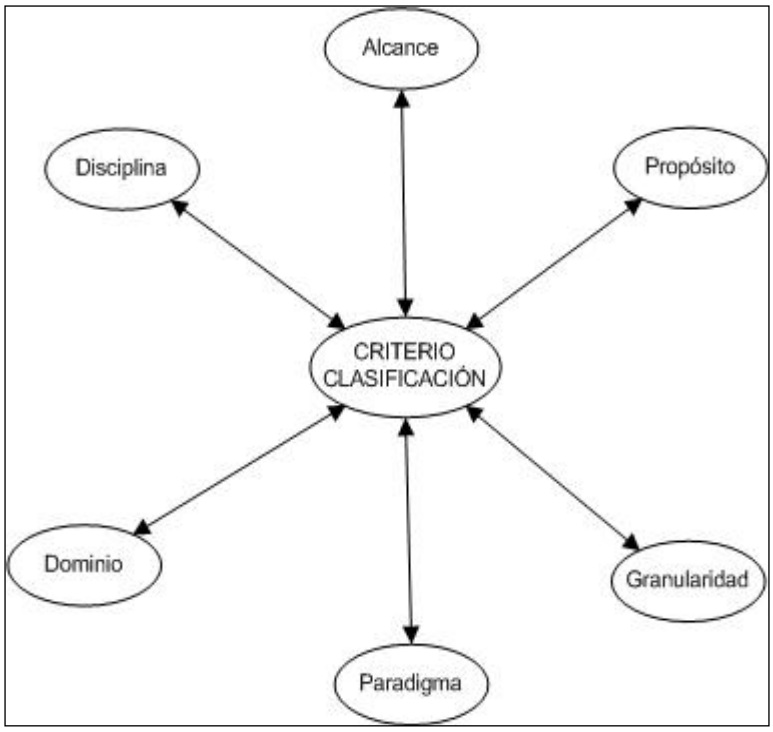
- Simple y fácil de aprender
- Cantidad moderada de criterios de agrupación, para evitar la ambigüedad y también la complejidad.

- Reflejar las propiedades de los patrones

- Apertura a la incorporación de nuevos patrones

Representación gráfica de los criterios de agrupación:

Figura 4. **Representación de criterios para clasificación de patrones**



Fuente: elaboración propia.

A partir de esto se pueden entender 3 clasificaciones referidas de autores reconocidos que se describen a continuación:

3.5.1.1. Clasificación de Gamma

Gamma clasifica sus patrones a partir de dos criterios:

- Propósito
- Alcance

El criterio de propósito refleja lo que un patrón hace, de acuerdo con este criterio define tres categorías:

- Creacional
- Estructural
- Comportamiento

Los patrones que pertenecen a la categoría creacional les concierne el proceso de creación de objetos. A los patrones de la categoría estructural les compete la composición de clases y objetos. Y los patrones de la categoría de comportamiento se caracterizan por definir la manera en que los objetos interactúan y distribuyen las responsabilidades.

El criterio de alcance especifica si un patrón se aplica básicamente a una clase o a los objetos, hay dos categorías, en este criterio:

- Clase
- Objeto

Los patrones que pertenecen a la categoría de objeto lidian con las relaciones entre ellos, que pueden cambiar en tiempo de ejecución y ser más dinámicos. Los patrones que pertenecen a la categoría Clase les atañen las relaciones entre clases y sus subclases. Estas relaciones son establecidas a través de herencia.

En esta clasificación un patrón puede únicamente pertenecer a una categoría por criterio, combinando sus dos criterios principales:

- Patrones creaciones-clase: esta combinación define patrones que trasladan la creación de objetos hacia otro objeto.

- Patrones estructurales-clase: usan herencia para componer interfaces o implementaciones.

- Patrones estructural-objeto: describen maneras de componer objetos para realizar nuevas funcionalidades.

- Patrones comportamiento-clase: describen como un grupo de objetos cooperan para realizar una tarea que un objeto simple no puede realizar.

Tabla III. **Clasificación de patrones Gamma**

Clasificación de patrones de diseño			
Proposito			
Alcance	*Creacional*	*Structural*	*Behavioral*
CLASE	*Factory Method*	*Adapter (class)*	*Interpreter*
			Template method
OBJETO	*Abstract Factory*	*Adapter (object)*	*Command.*
	Builder	*Bridge*	*Iterator*
	Prototype	*Composite*	*Mediator*
	Singleton	*Decorator*	*Memento*
		Facade	*Observer*
		Flyweight	*State*
		Proxy	*Strategy*
			Visitor
			Chain Of resp.

Fuente: elaboración propia.

3.5.1.2. Clasificación de patrones de diseño de Buschman

Buschman presenta los patrones de una forma similar a la clasificación que hace Gamma dividiendo los patrones a partir de tres criterios:

- Funcionalidad
- Principios estructurales
- Granularidad

El criterio de funcionalidad refleja para que función en particular sirve el patrón como una plantilla.

A partir de la funcionalidad distingue cuatro categorías que son:

- Creación
- Comunicación
- Acceso
- Organización de tareas complejas

Patrones por criterio de funcionalidad

Los patrones que pertenecen a la categoría de creación, especifican cómo crear instancias particulares de complejas estructuras recursivas o agregadas.

Los patrones de la categoría de comunicación, describen cómo organizar la comunicación entre un conjunto de objetos que colaboran entre sí.

Los patrones que se ubican en la categoría de acceso describen cómo acceder a los servicios y estado de objetos compartidos o remotos de una manera segura sin violar la encapsulación de estado y comportamiento.

Los patrones de tareas complejas especifican cómo distribuir responsabilidades a través de objetos que cooperan y el orden para resolver requerimientos funcionales complejos.

Patrones por criterio de principios estructurales

Estos patrones reflejan algunos de los principios fundamentales de diseño y Buschman los distingue en cuatro categorías:

- Abstracción
- Encapsulación
- Separación de intereses
- Acoplamiento y cohesión

Los patrones que pertenecen a la categoría de abstracción proveen una vista abstracta generalizada de una entidad particular o tarea en el sistema.

Los patrones de la categoría de encapsulación encapsulan detalles de un objeto o componente en particular.

Los patrones de la categoría de separación de intereses agrupan a aquellos patrones que separan los objetos o componentes para resolver una tarea en particular o servicio. Finalmente los patrones de la categoría de acoplamiento y cohesión remueven o suavizan las relaciones de estructura, comunicación y dependencias que en otro caso estarían fuertemente acoplados.

Al contrario de la clasificación que hace Gamma, un patrón puede ser clasificado en más de una categoría a la vez.

3.5.2. Patrones implementados

Los patrones que han sido implementados, fueron elegidos de acuerdo a criterios tales como funcionalidad, incidencia en la arquitectura final, reducción del tiempo de la implementación final, para poder asegurar un diseño escalable.

3.5.2.1. *Enterprise Inventory*

Este patrón optimiza la implementación de los patrones para facilitar la recomposición.

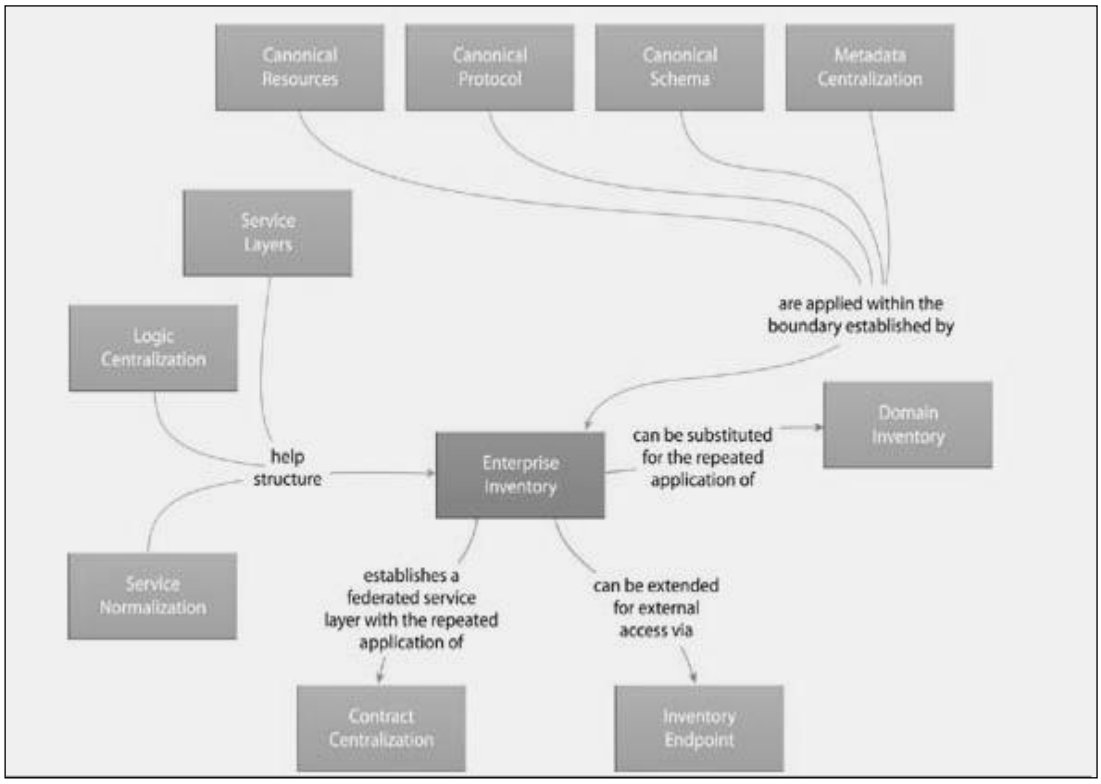
Problema: implementar servicios independientemente por medio en diferentes proyectos en la empresa, impone algunos riesgos relacionados con la constante posibilidad de producir servicios y arquitecturas con las que se compromete la reutilización de los mismos.

Solución: los servicios para múltiples soluciones pueden ser diseñados para implementarse en una arquitectura de inventario estandarizada en donde puedan ser libremente y repetidamente recompuestos.

Impacto: un análisis inicial significativo debe hacerse para definir un bosquejo inicial del proyecto para evaluar el impacto organizacional de los subsecuentes requerimientos para gobernar el inventario.

Principios: contrato de servicios estandarizado, abstracción de servicios y composición.

Figura 5. **Patrón *Service Inventory***



Fuente: elaboración propia.

3.5.2.2. *Domain Inventory*

Este patrón sirve para maximizar la recomposición cuando la estandarización a todo nivel no es posible.

Problema: al establecer un inventario de servicios puede este volverse inmanejable para algunas empresas y los intentos pueden poner en peligro el éxito de una adopción de SOA.

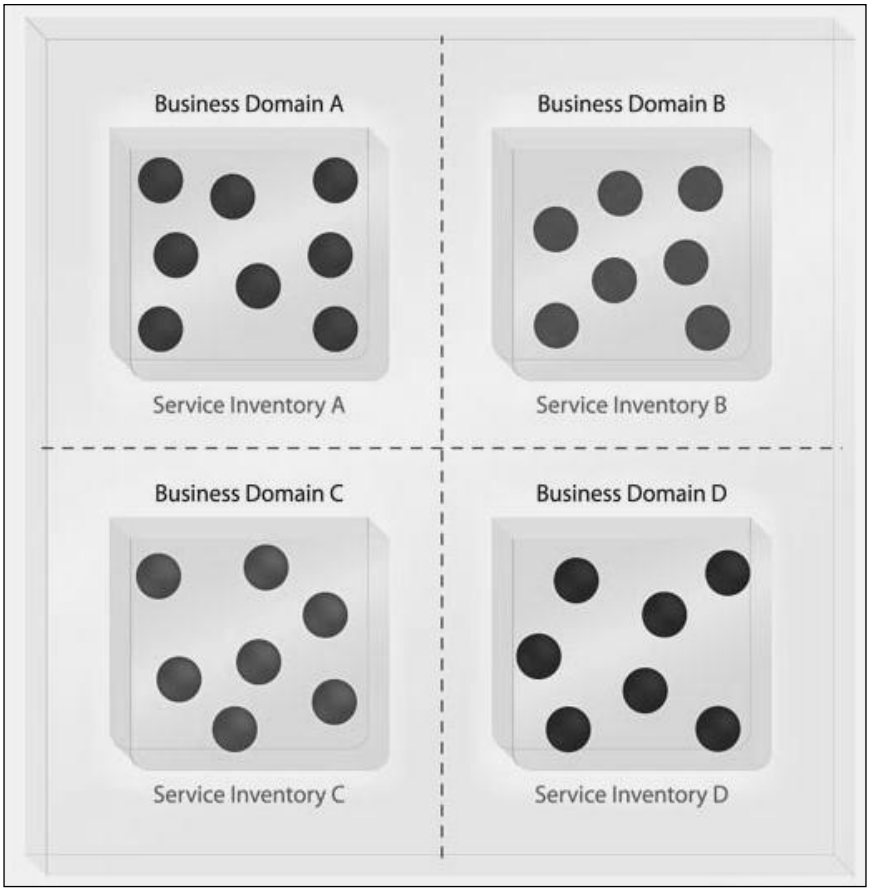
Solución: los servicios pueden ser agrupados en inventarios clasificados por dominio, de los que cada uno puede ser estandarizado y gobernado con mayor facilidad.

Aplicación: este patrón es aplicable cuando interactúan distintos dominios lógicos y se requieren establecer cuidadosamente las fronteras entre los dominios.

Impacto: la disparidad en la normalización entre los dominios de los inventarios de servicios, impone la transformación de los requerimientos y reduce el total de los beneficios potenciales de la implementación de SOA.

Principios: contrato de servicios estandarizado, abstracción de servicios y composición.

Figura 6. **Modelo de patrón *Domain Inventory***



Fuente: elaboración propia.

4. DESCRIPCIÓN DEL SISTEMA

Una empresa cuenta con dos centros de operaciones en ubicaciones geográficas distintas, en cada centro de operaciones cuenta con un equipo de desarrollo. El *software* desarrollado por ambos equipos se utiliza indistintamente en los dos centros de operaciones, surge ahora la necesidad de diseñar una solución que permita interconectar varias aplicaciones que la empresa ya posee, los sistemas involucrados son:

- Sistema de gasolinera
- Sistema de compras
- Sistema de contabilidad
- Sistema de inventario
- Sistema de báscula

La solución deberá cumplir cada uno de los siguientes requerimientos:

- Cada ingreso de combustible inicia con una orden de compra en el sistema de gasolinera, que deberá estar disponible para consulta posteriormente.

- Una vez generada la orden de compra se recibe el combustible enviado por la petrolera, al llegar el combustible a la gasolinera el camión es pasado por una báscula antes y después de descargar la gasolina, lo que genera una nota de peso.

- El ingreso del combustible debe ser ingresado en el sistema de inventario, lo que como resultado actualiza los saldos de cada tipo de combustible.

- Al momento de registrar el combustible debe notificarse al sistema de gasolinera sobre la recepción para que este actualice los saldos en los tanques respectivos.

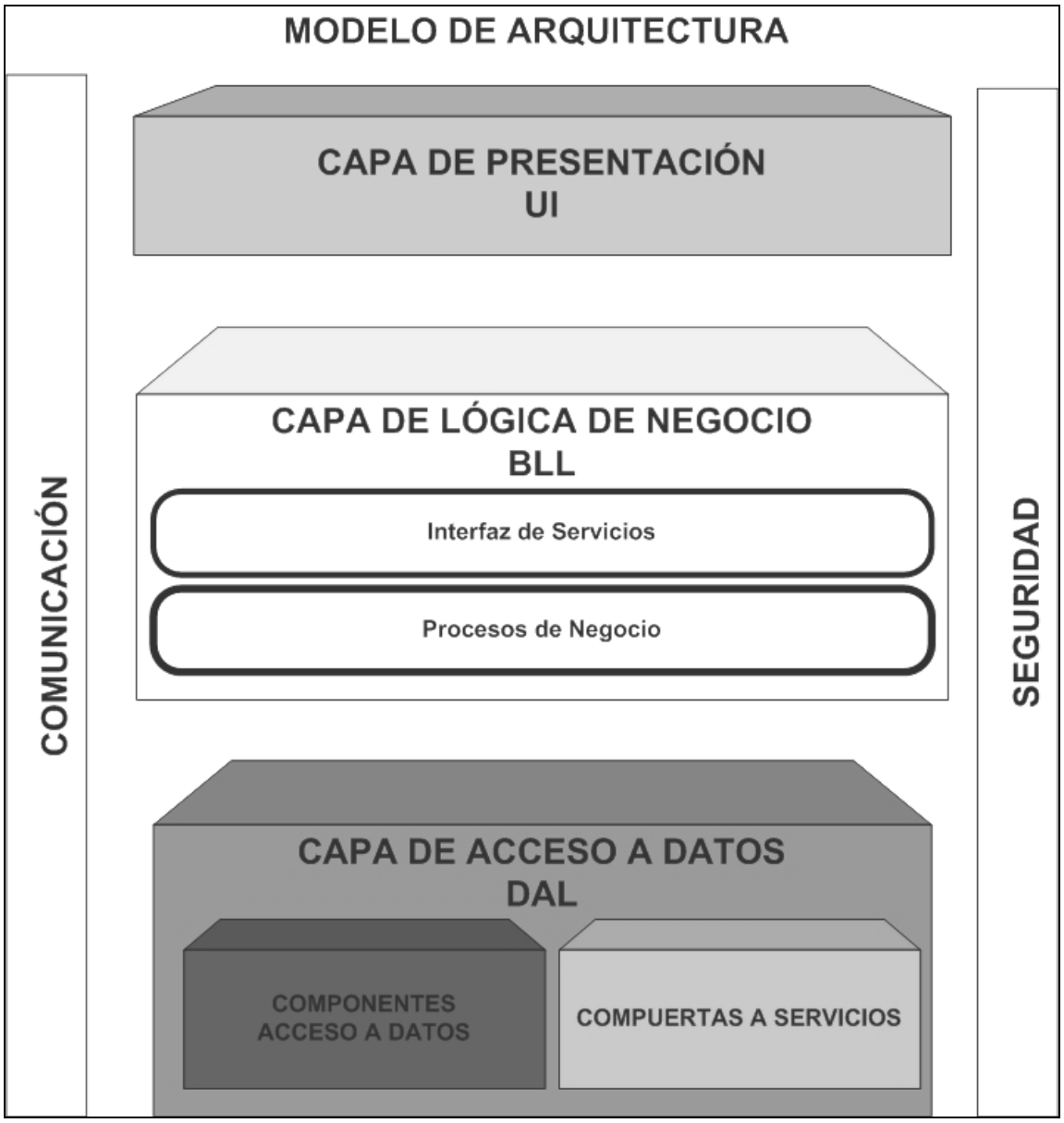
- El proceso de recepción debe generar el registro contable correspondiente en el sistema de contabilidad.

- Una vez registrado el ingreso del combustible debe actualizarse el estado de la orden de compra que originó el proceso.

4.1. Arquitectura del sistema

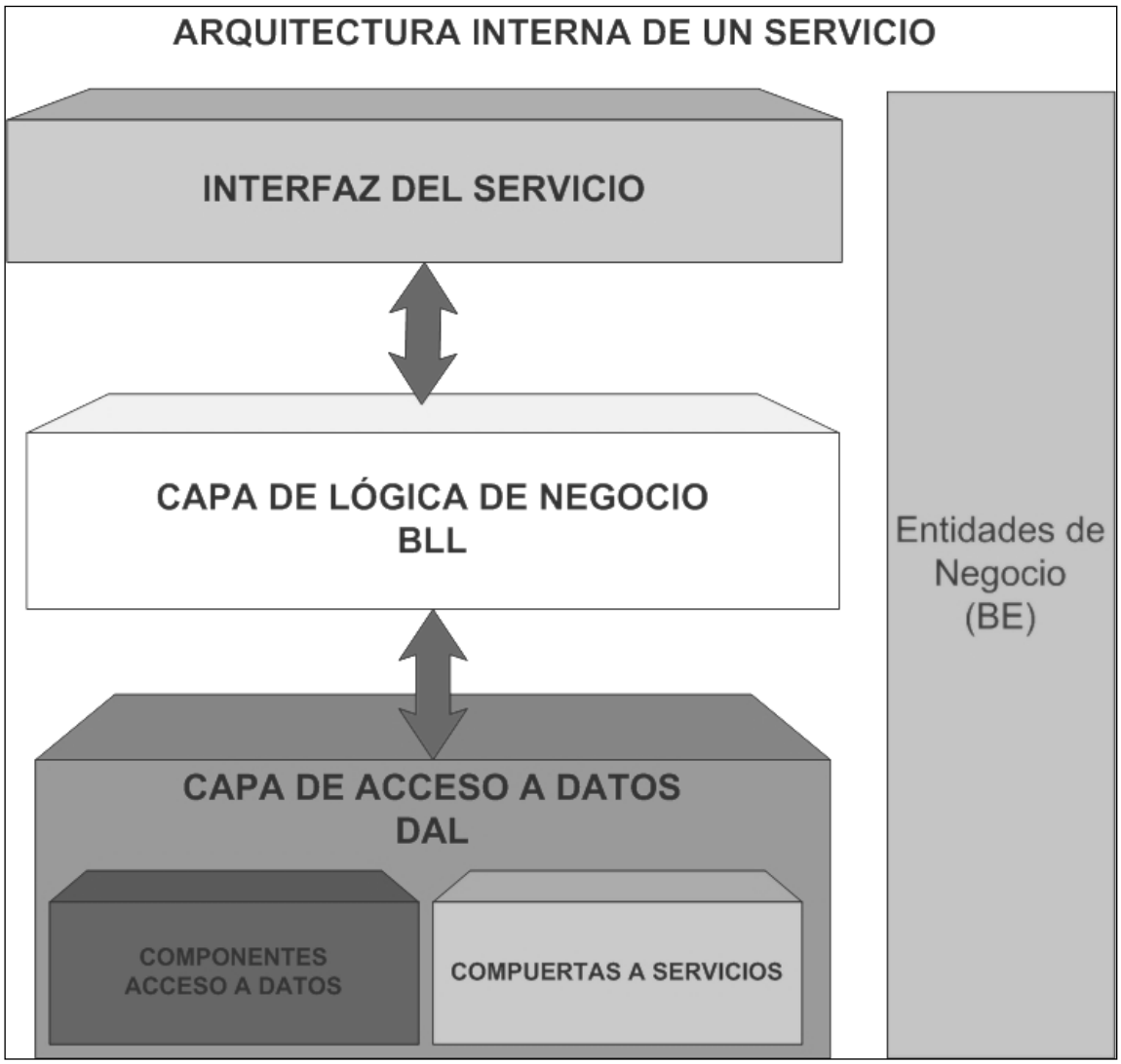
La arquitectura del sistema estará definida tanto de manera global como de manera interna para cada componente, siguiendo el patrón de arquitectura de n-capas o multicapas.

Figura 7. **Diagrama de arquitectura general**



Fuente: elaboración propia.

Figura 8. **Arquitectura singular**



Fuente: elaboración propia.

4.2. Las capas con las que se estructurará el sistema

Las capas con las que se estructura el sistema están definidas de acuerdo al modelo de arquitectura elegido y representan la encapsulación de funcionalidades relacionadas, que solo tienen comunicación con la capa inmediata superior o inferior.

4.2.1. Capa de presentación

Esta capa se define como el conjunto de objetos que permiten controlar la interacción entre el usuario y el *software,* puede ser tan simple como una línea de comando, un menú, pero últimamente existe una tendencia a hacer interfaces graficas en el cliente que faciliten la operación o incluso interfaces basadas en HTML para navegadores, también con un alto contenido gráfico. La responsabilidad básica de esta capa es mostrar la información al usuario e interpretar las acciones que este realiza para finalmente transferirlas a la capa siguiente:

La capa de presentación debe cumplir con los principios básicos de usabilidad.

- Usabilidad: la Organización Internacional para la Estandarización (ISO/IEC 9126) define la usabilidad de la siguiente manera: la usabilidad se refiere a la capacidad de un *software* de ser comprendido, aprendido, usado y ser atractivo para el usuario en condiciones específicas de uso

Los principios básicos en que se basa la usabilidad son:

- Facilidad de aprendizaje: es la facilidad con la que los nuevos usuarios desarrollan una interacción efectiva con el sistema o producto.

- Flexibilidad: se refiere a las maneras en las que el usuario puede intercambiar información con el sistema.

- Robustez: esta es el nivel de soporte que el sistema provee al usuario para el cumplimiento de sus objetivos.

4.2.2. Capa de lógica del negocio

En esta capa es el corazón de la arquitectura aquí se concentra el código que realiza todas las funciones que el *software* debe cumplir. En esta capa se incluyen los cálculos basados en entradas o información almacenada y validación de datos.

Esta capa permite desacoplar la presentación del acceso a los datos, puede tener interacción ya sea con una capa de presentación ya sea una interfaz de usuario como también puede contener una subcapa de interfaz de servicios, que exhibirá la funcionalidad contenida en cada objeto incluido en esta capa. En el caso de tener una subcapa de servicios *web*, cada servicio define un contrato que permite a la capa interactuar con componentes externos o de otras capas.

4.2.3. Capa de acceso a datos

La capa de acceso a datos es una capa lógica que suele ser encarga de gestionar la información que el sistema requiera para operar. La información puede provenir de sistemas de gestión de bases de datos, archivos, incluso de otros sistemas, por ejemplo de servicios *web*.

Cuando los datos se gestionan con un sistema de administración de base de datos, esta capa contiene el código que permite interactuar con la base de datos por medio de objetos que deben proveer al menos estas funcionalidades:

- Gestionar una conexión

- Soportar las operaciones básicas de cada entidad
  - Consulta
  - Inserción
  - Eliminación
  - Modificación

Cuando los datos se gestionan de otros sistemas o de servicios la capa de acceso a datos debe incluir objetos que provean las siguientes funcionalidades:

- Gestión de la comunicación

- Aceptación o rechazo de contratos

- Manejo de excepciones
- Modos de conexión
- Envío y recepción de información hacia y desde el servicio contratado

Una de las características de esta capa es que al desacoplar el acceso a los datos de la lógica del negocio, permite que la aplicación sea flexible y que en caso de que exista un cambio en el proveedor del servicio o la base de datos el impacto en el sistema en general se minimice debiendo hacerse los ajustes necesario solo sobre esta capa.

4.2.4. Elementos de interacción

Los elementos de interacción no comprenden una capa como tal más bien son el canal de comunicación que permite la interacción entre cada capa.

Estos elementos son también conocidos como entidades de negocio y corresponden a la abstracción de un objeto que cumple una función dentro del sistema, como puede ser un cliente, una factura, etcétera.

Las entidades del negocio generalmente se pueden mapear contra una tabla en la definición del modelo de base de datos o alguno de los componentes que permiten interactuar con servicios según sea el caso.

La definición de estas puede ir desde una clase que defina un objeto hasta un archivo en XML con una estructura que permita envolver un conjunto de características. El objetivo final de los elementos de interacción será permitir que las capas interactúen intercambiando entidades permitiendo así, hacer cambios en el estado o en las propiedades mismas de un objeto y trasladar esos cambios a la capa correspondiente.

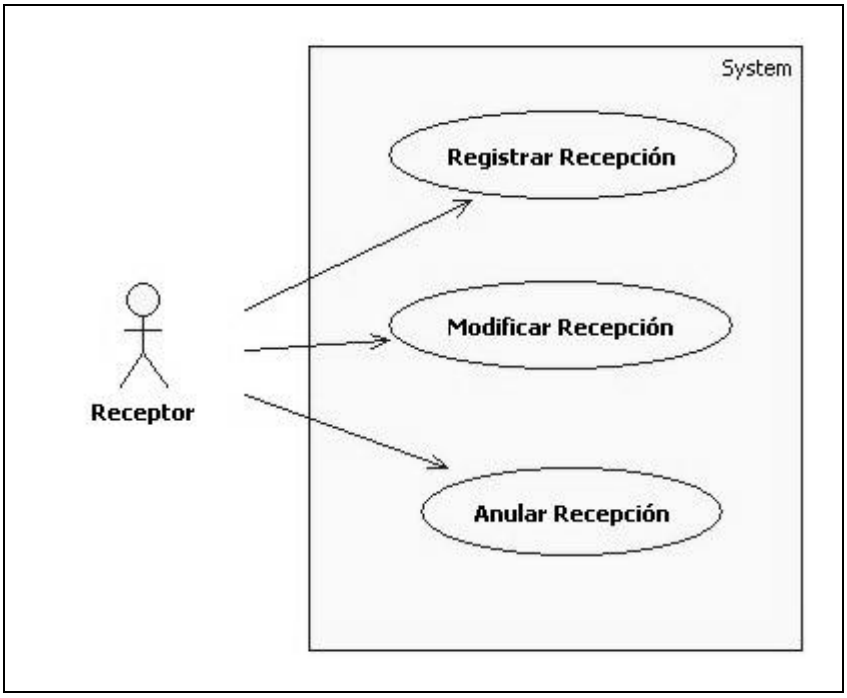
5. PROTOTIPO DE ARQUITECTURA

En el desarrollo del prototipo de la arquitectura se implementan los patrones cuya definición es posible encontrar en el capítulo 3 y a continuación solo se hace un sumario de los mismos y se muestra su implementación en el planteamiento de la solución:

5.1. Casos de uso

Los requerimientos funcionales del sistema se modelan a partir de la definición de casos de uso incluidos en los siguientes diagramas:

Figura 9. **Casos de uso básico**



Fuente: elaboración propia.

Caso de uso: registro de recepción

Resumen: es el caso en el que se realiza el proceso de recepción de combustible.

Actores:

Receptor

Flujo básico:

- El receptor selecciona en el sistema registrar nueva recepción;

- Se consulta la orden de compra que origina la recepción;
- Se registra en el sistema de inventario el ingreso;
- Se consulta la nota de peso en el sistema de báscula;
- Se traslada la información de precios al sistema de gasolinera;
- Se registra la póliza contable en el sistema de contabilidad; y
- Se actualiza el estado de la orden de compra que dio origen a la operación.

Precondiciones:

- Debe existir una orden de compra que dé origen a la entrega del producto.

Excepciones:

- Cuando no existe una orden de compra previa, se realiza la recepción sin ejecutar el paso 7, en ese punto se crea la orden de compra a partir de la información de la entrega (factura, valor).

Caso de uso: modificación de recepción

Resumen: este escenario surge cuando se registra una recepción de combustible y alguno de los datos es registrado de manera incorrecta.

Actores:

Receptor

Flujo básico:

- Ingresar el número de recepción registrada,
- Se consultan los datos de la recepción en el sistema de combustible,
- Se validan los datos ingresados,
- Se solicita confirmación de la operación,
- Se actualiza la información en el sistema de gasolinera,
- Se actualiza información en el inventario y
- Se notifica cuando la operación se ha completado.

Precondiciones:

- Se requiere una recepción registrada previamente

Excepciones:

- Cuando la recepción se realiza en el sistema de gasolinera y se requiere modificar la información básica de la recepción debe ejecutarse el caso de uso anulación de recepción.

Caso de uso: anulación de recepción.

Resumen: este escenario surge cuando se han registrado errores en una recepción de combustible y dicha información corresponde a la información básica de una recepción.

Actores:

Receptor

Flujo básico:

- Brinda el número de recepción registrada;
- Se solicita confirmación de la operación;
- Se anula la recepción en el sistema de gasolinera;
- Se anula la recepción en el sistema de inventario; y
- Se externa el registro contable.

Precondiciones:

- Se requiere una recepción registrada previamente

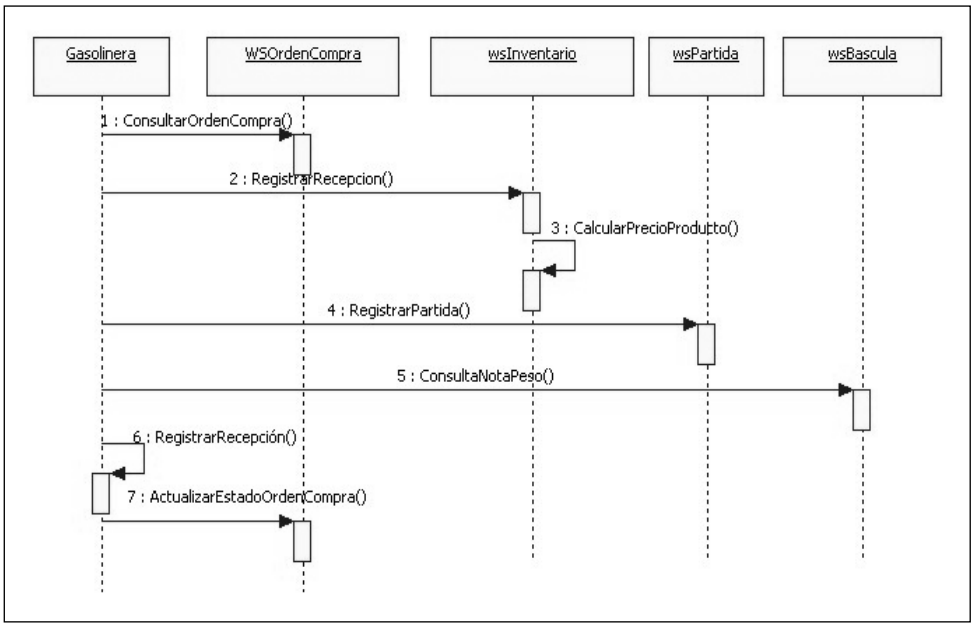
Excepciones:

- Cuando el período contable ya ha sido cerrado se requiere de un nivel de autorización adicional.

5.2. Diagrama de secuencia

Crear recepción de combustible

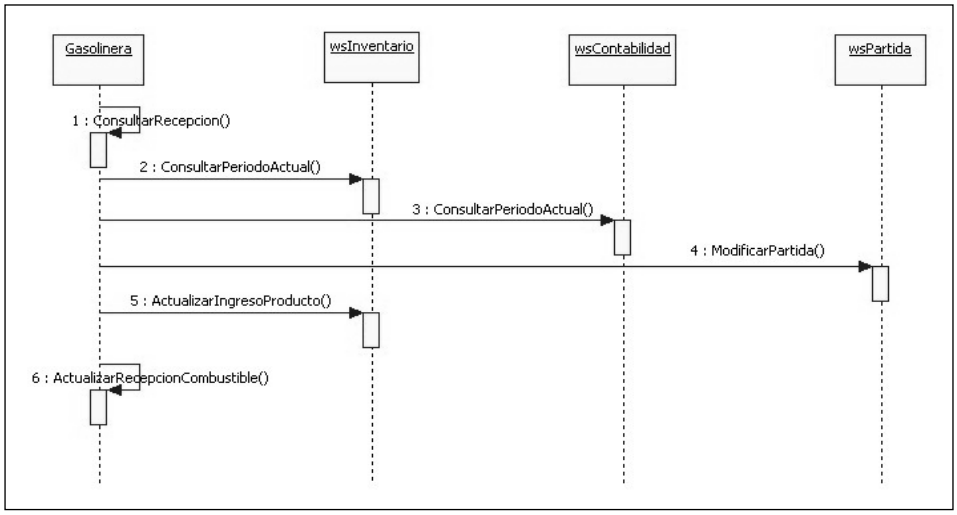
Figura 10. **Diagrama de secuencia, creación de recepción de combustible**



Fuente: elaboración propia.

Modificar recepción de combustible

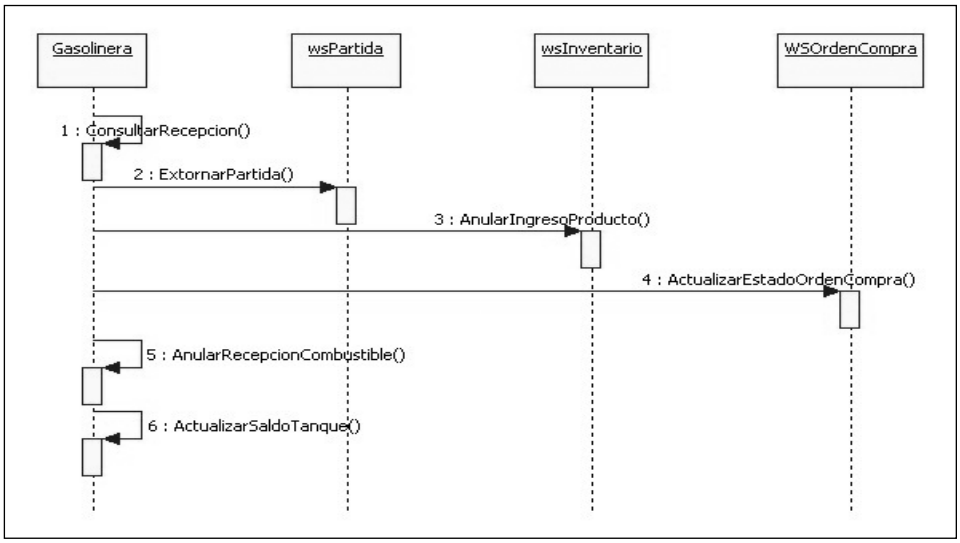
Figura 11. **Diagrama de secuencia: modificación de recepción de combustible**



Fuente: elaboración propia.

Anulación de recepción de combustible.

Figura 12. **Diagrama de secuencia: Anulación de recepción de combustible**



Fuente: elaboración propia.

5.3. Diagrama de clases

Se presenta el modelo de clases diagramado de la solución final, que corresponde al mapeo de los objetos de negocio que intervienen en la solución que se plantea en cada módulo que toma parte de la arquitectura final.

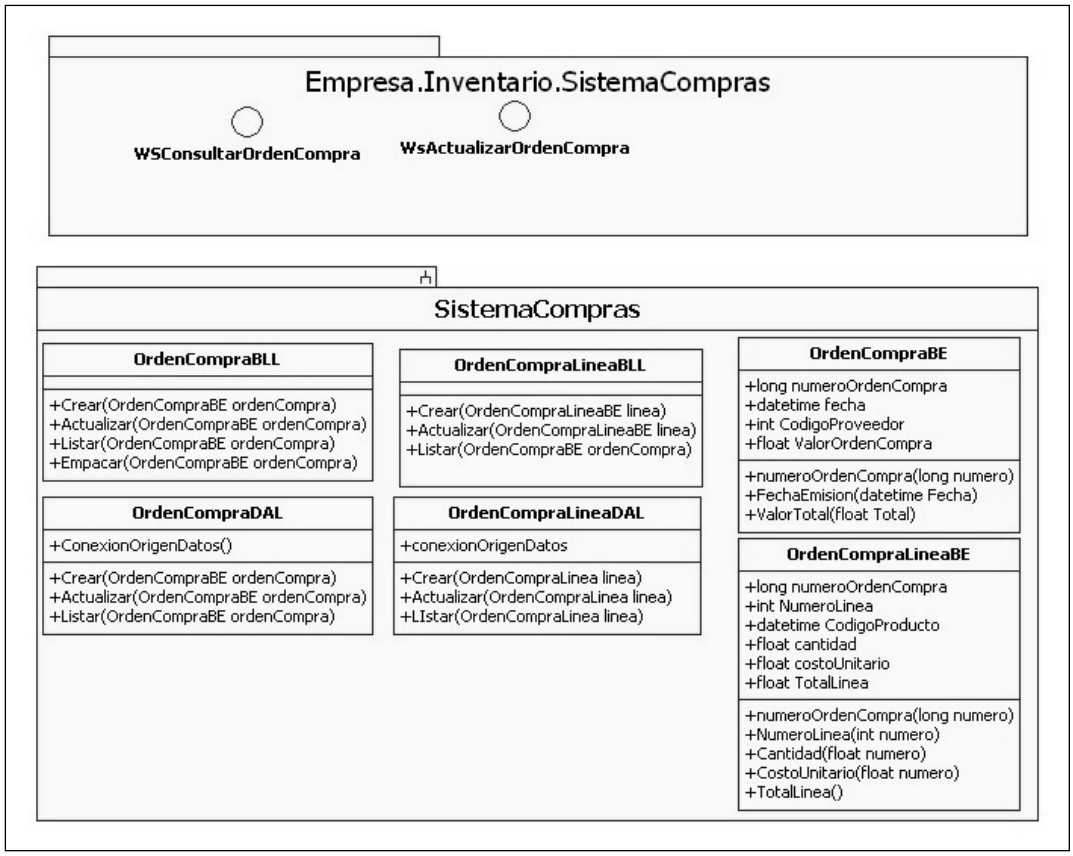
5.3.1. Diagrama de clases sistema de compras

Descripción: el sistema de compra aplica del patrón de arquitectura de capas, definiendo las capas:

- Lógica del negocio denotada por BLL
- Acceso a datos denotada por DAL
- Y para fines de modelado de esta solución la capa de presentación está compuesta por servicios *web*.

Diagrama:

Figura 13. **Diagrama de clases sistema de compras**

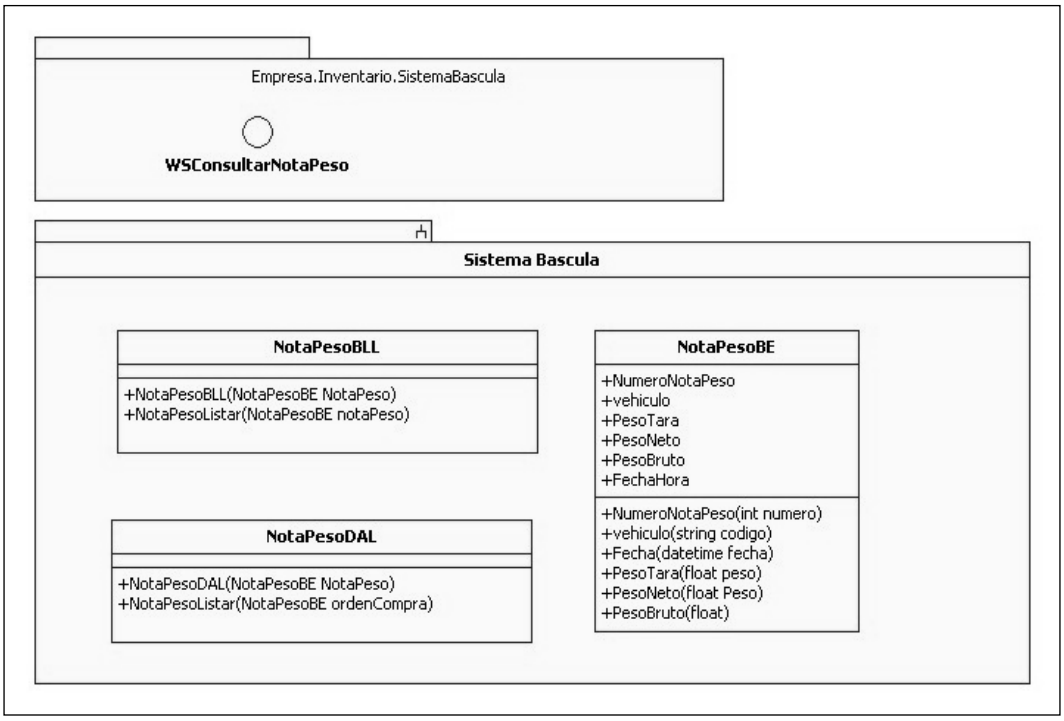


Fuente: elaboración propia.

5.3.2. Diagrama de clases sistema de báscula

El diagrama de clases del sistema de báscula es la representación conceptual de los objetos de negocio que intervienen en el diseño del módulo.

Figura 14. **Diagrama de Clases sistema de báscula**

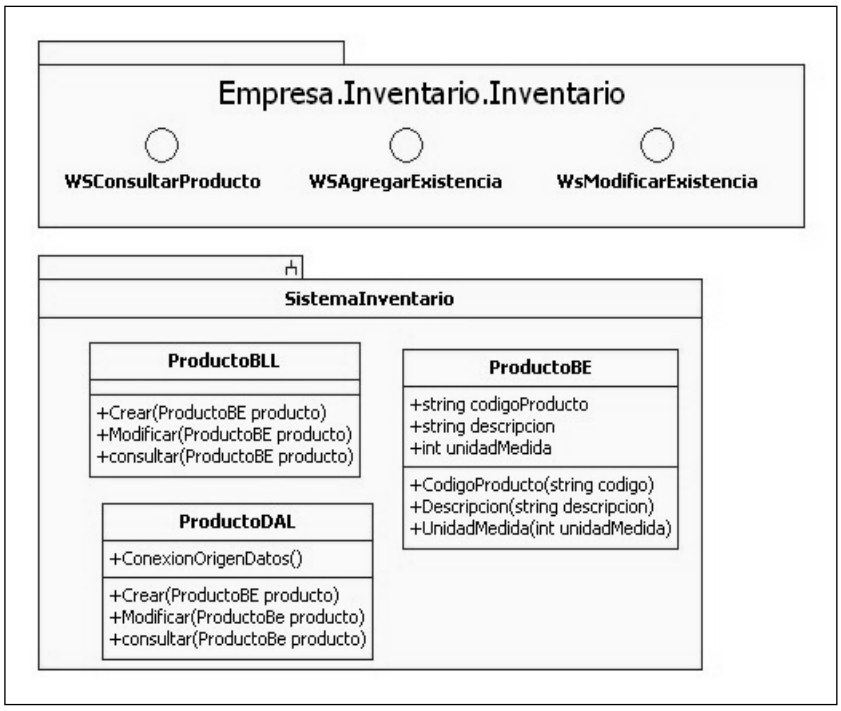


Fuente: elaboración propia.

5.3.3. Diagrama de clases sistema de inventario

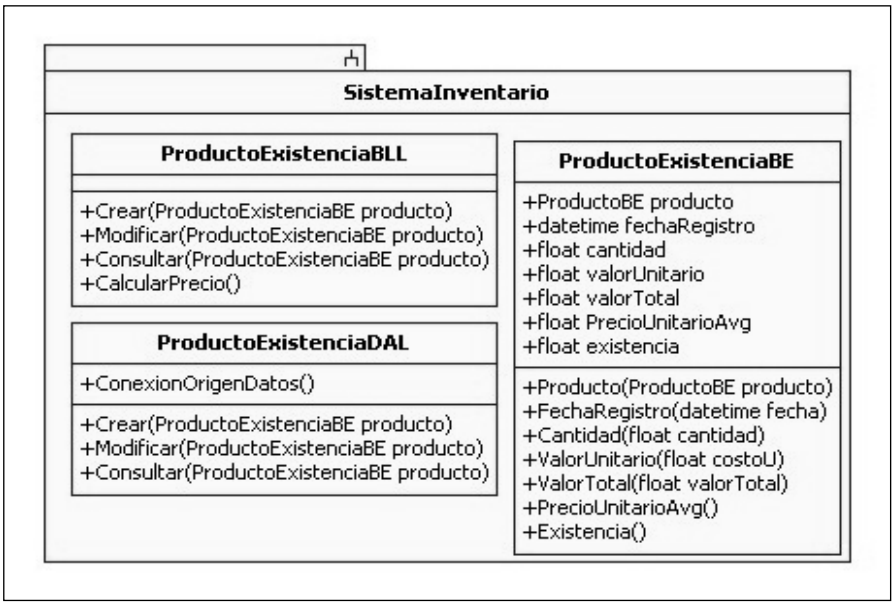
Descripción: se incluyen las clases más importantes con la finalidad de mostrar el funcionamiento del sistema de inventario en la integración a la solución final, también se ha desarrollado aplicando el patrón de arquitectura de capas.

Figura 15. **Diagrama de clases sistema de inventario**



Fuente: elaboración propia.

Figura 16. **Diagrama de clases sistema de inventario**



Fuente: elaboración propia.

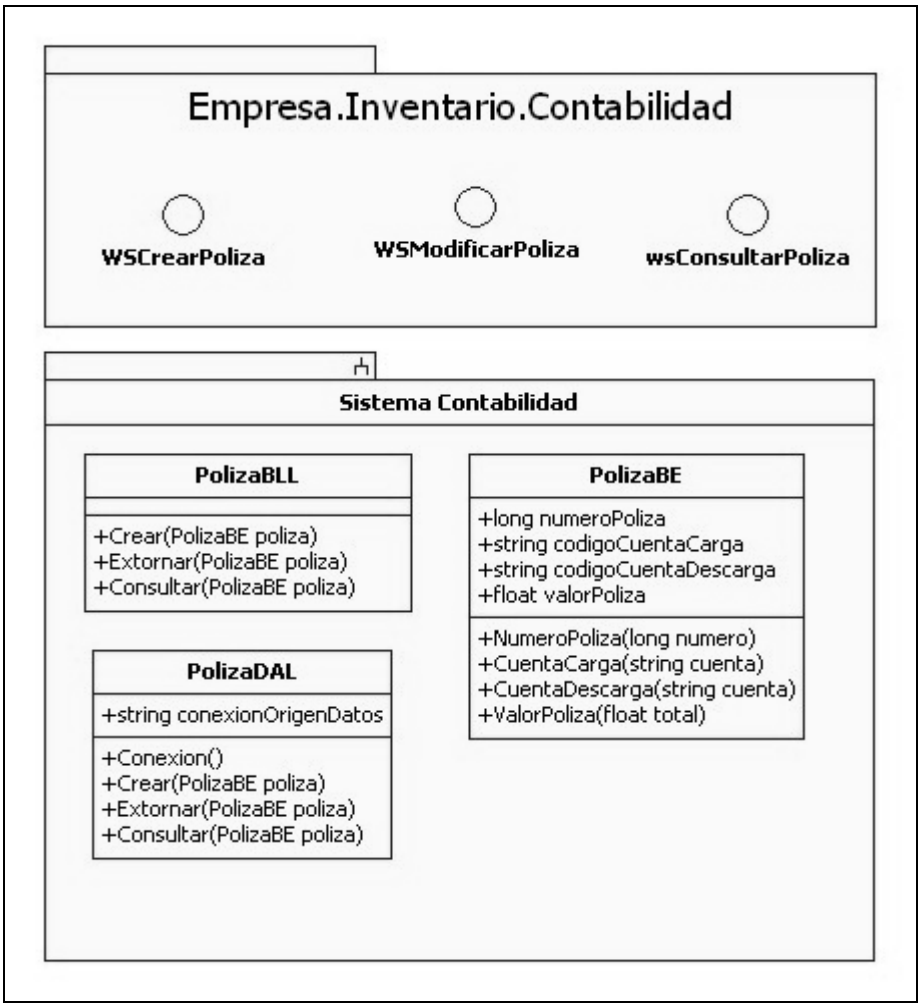
5.3.4. Diagrama de clases sistema de contabilidad

Descripción: el sistema de contabilidad crea una póliza por cada ingreso que se registra en el sistema de inventario a partir de los datos del ingreso.

Este sistema implementa el patrón de arquitectura de capas, en el diagrama se muestran 3 capas básicas que son:

- Capa de presentación por medio de servicios *web* (WS)
- Capa de lógica de negocio (BLL)
- Capa de acceso a datos (DAL)

Figura 17. **Diagrama de clases de sistema de contabilidad**



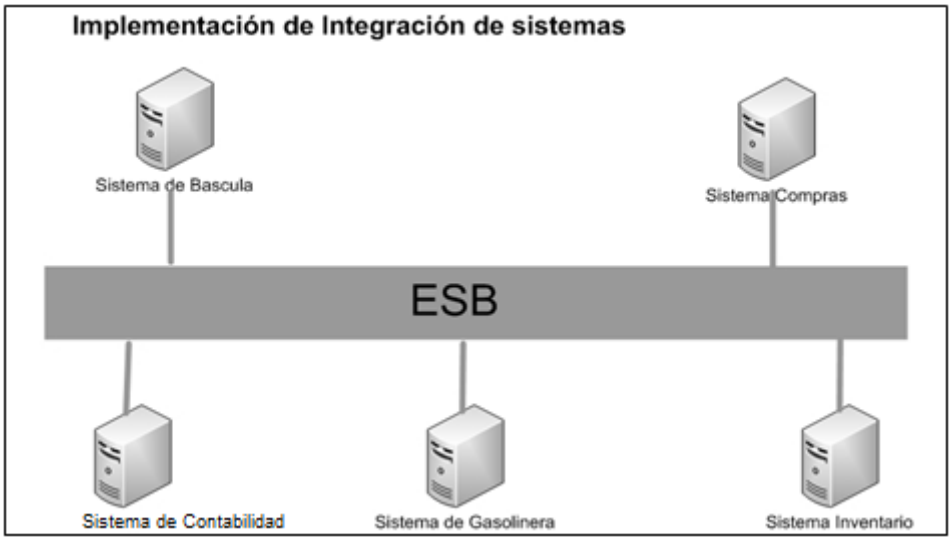
Fuente: elaboración propia.

5.4. Diagrama de implementación

Puesto que los patrones se aplican directamente al diseño de la solución, la implementación de la solución no tiene requerimientos adicionales a los de una arquitectura orientada a servicios, o dicho de otra manera es posible implementar una solución de este tipo incluso cuando ya se posee la arquitectura la aplicación de los patrones favorece a la optimización de la administración e incorporación de nuevos servicios al modelo.

La implementación lógica final de la solución se muestra a continuación en el diagrama.

Figura 18. **Diagrama de implementación**



Fuente: elaboración propia.

5.5. Implementación de Enterprise *Inventory*

El patrón Enterprise *Inventory,* se aplica con el objetivo de conseguir organizar los servicios *web* de manera que se facilite la recomposición y se evite duplicar funcionalidad y con ello ahorrar tiempo.

La forma de implementar este patrón a lo largo de toda la arquitectura de la empresa requiere:

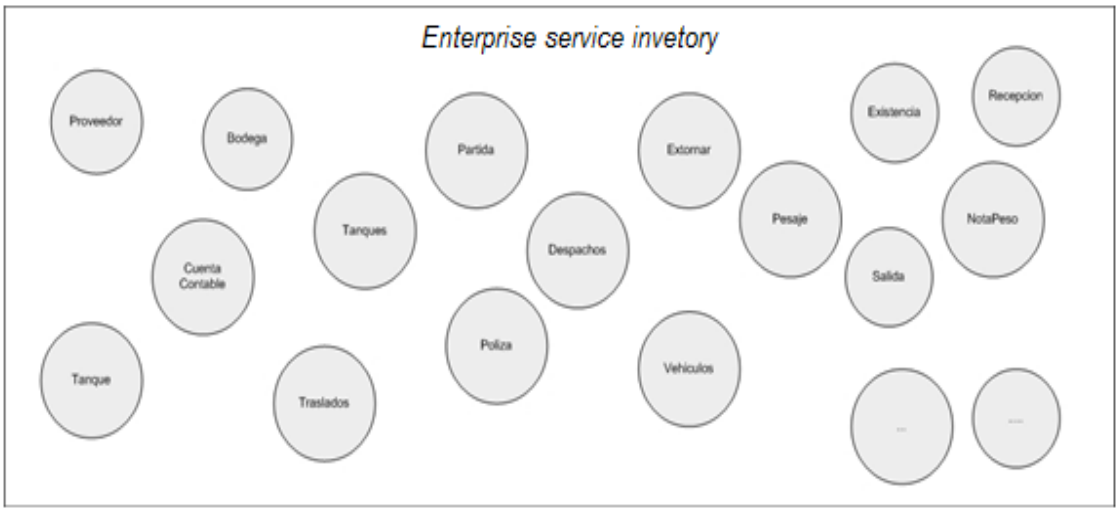
- Evaluación y planeación del alcance que requiere tenga el inventario

- Analizar y planear adecuadamente la descomposición de la funcionalidad que se requiere a cada servicio.

- Realizar el paso 2 de manera iterativa hasta obtener el nivel de abstracción que permita articular adecuadamente cada funcionalidad en módulos más grandes.

- Para ir organizando los servicios desde su desarrollo se define un nombre de espacios de servicios en el que se agrega cada servicio nuevo.

Figura 19. **Implementación de patrón Enterprise *Service Inventory***



Fuente: elaboración propia.

5.6. Implementación de *Domain Inventory*

El patrón Enterprise *Service Inventory*, ayudará en una implementación empresarial a manejar de manera más eficiente los servicios de los que se dispone, pero con el transcurrir del tiempo se plantea la situación en la que el inventario de servicio se hace tan grande que hay simultáneamente mayor cantidad de desarrolladores y mayor número de servicios lo que puede dificultar la administración del inventario.

Con la aplicación del patrón *Domain Inventory*, se optimiza la aplicación del patrón Enterprise *Service Inventory* la estrategia para su implementación en la empresa es:

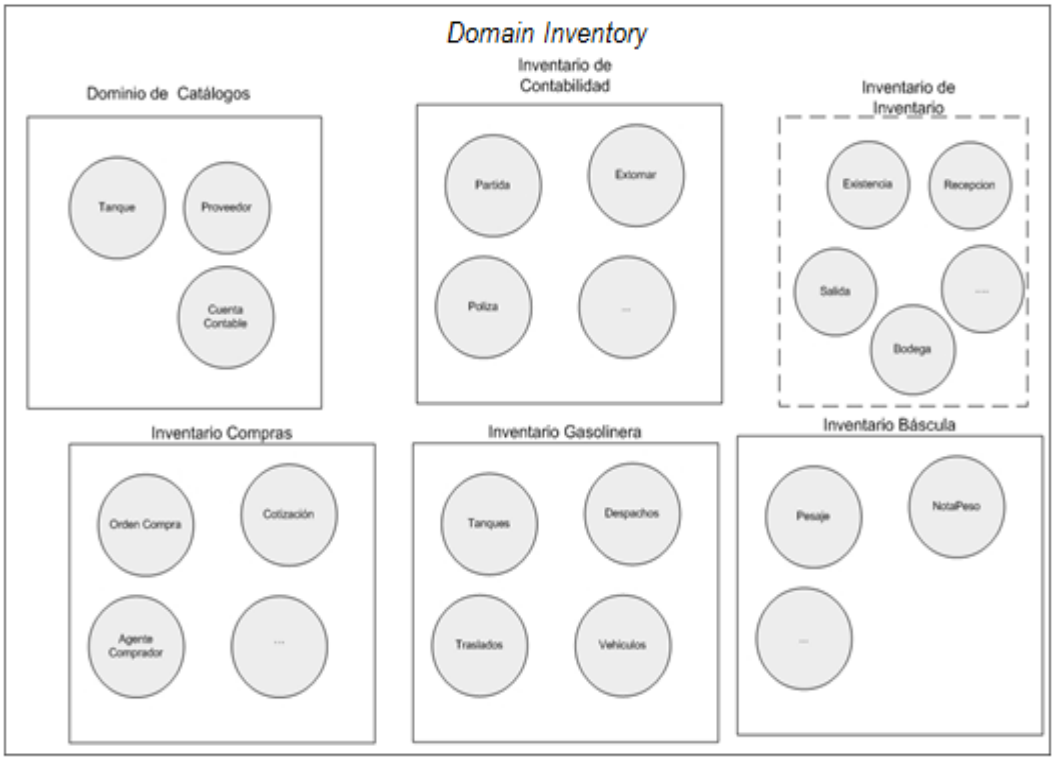
- Definir adecuadamente los dominios por venir para la clasificación de los servicios.

- No definir dominios diferentes para distintos equipos o departamentos de TI.

- En el caso de los equipos de trabajo con ubicaciones geográficas diferentes, pueden resultar un dominio basado en geografía más no en el equipo de trabajo, de acuerdo al punto anterior.

- Para aplicar el patrón desde el desarrollo mismo de los servicios, se agrega un nivel al espacio de nombres que corresponde al dominio en el que se clasifica el servicio.

Figura 20. **Implementación de patrón *Domain Inventory***



Fuente: elaboración propia.

CONCLUSIONES

1. Cuando se habla de arquitectura orientada a servicios no se trata de sistemas interconectados por medio de servicios *web*, que se comunican por medio de protocolos estándar, para llevar a cabo una función. Se trata de sistemas que desde su concepción son diseñados y desarrollados con orientación a servicios, que se organizan para facilitar la composición, escalabilidad y que a su vez sean íntegros y robustos.

2. Los patrones de diseño son una solución efectiva y eficiente a un problema conocido y aplicada a la implementación de la arquitectura orientada a servicios, permite un ordenamiento efectivo que incide en la agilidad de una empresa para ajustar y adoptar sus recursos a sus necesidades.

3. La implementación de una arquitectura orientada a servicios facilitará y promoverá la reutilización de código, reduciendo por consiguiente costos y tiempo en el desarrollo de un proyecto.

4. La aplicación de patrones y la implementación de una arquitectura orientada a servicios, puede implementarse con la reutilización de los mismos recursos que se utilizan en el desarrollo de una arquitectura tradicional de aplicaciones, cliente servidor.

5. No existe un número de patrones mínimos ni máximos para aplicar en el diseño de una solución, el número adecuado es el que facilita la implementación de la solución sin incrementar la dificultad en su desarrollo y administración del diseño.

RECOMENDACIONES

1. Cuando se considere implementar una arquitectura orientada a servicios es necesario planificar el crecimiento de los participantes, tanto de servicios como de consumidores y definir la forma en la que se administrará este crecimiento.

2. Al momento de elegir una arquitectura orientada a servicios, es necesario validar que los sistemas que interactúan en la misma, tengan también una orientación a servicios.

3. Cuando aplique patrones en el diseño de una arquitectura tome en cuenta que la aplicación excesiva de patrones podría resultar en una mayor dificultad para manejar la solución completa.

BIBLIOGRAFÍA

1. DE LA TORRE LLORENTE, Cesar. *Guía de Arquitectura N-Capas Orientada al Dominio con .Net 4.0*, Estados Unidos de América: Karasis Consulting, 2010. ISBN: 978-84-936696-3-8.

2. ERL, Thomas. *SOA Desing Patterns*. Estados Unidos de América: Prentice Hall. ISBN: 0-13-613516-1.

3. FERRARA, Alex. *.Net Web Services.* Estados Unidos de América: Oreilly, 2002. 396 p. ISBN: 0-596-00250-5.

4. FOWLER, Martin. *UML Distilled*. 3a ed. Estados Unidos de América: Addison-Wesley. ISBN: 0-321-19368-7.

5. FREEMAN, Erick. *Design patterns*. Estados Unidos de América: O'reilly, 2004. 638 p. ISBN: 978-0-596-00712-6.

6. ISO. *ISO/IEC 9126-1:2001. Usabilidad.* [en línea]. Estados Unidos de América: Disponible en Web:<http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=22749>. [Consulta: en mayo 2011].

7. *Microsoft Aplication Architecture For .Net.* 2008. Fascículo Designing Aplications and Services. Estados Unidos de América: Microsoft, 2008. 158 p. ISBN: 0-7356-1837-2.

8. MICROSOFT*. Arquitectura Global para Servicios Web XML GXA*, [en línea]. Estados Unidos de América: Disponible en *Web*: <http://www.directionsonmicrosoft.com/sample/DOMIS/update/2002/10oct/1002gdffws.htm>. [Consulta: en febrero 2011].

9. W3C. *Servicios Web,*[en línea]. Estados Unidos de América: Disponible en *Web*: <http://www.w3.org/standards/webofservices/>. [Consulta en febrero 2011].

10. _____*. XML.* [en línea]. Estados Unidos de América: Disponible en *Web*:<http://www.w3.org/standards/xml/>. [Consulta: en febrero 2011].

11. WIKIPEDIA. *Clasificación de Servicios Web* [en línea]. Estados Unidos de América: Disponible en Web: <http://en.wikipedia.org/wiki/Category:Web_services>. [Consulta: en marzo 2011].

12. _____. *Patrón de Diseño.* [en línea]. Estados Unidos de América: Disponible en Web: <http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o>. [Consulta: en marzo 2011].