VALIDACION DE LA COMPUTACION ORIENTADA A SERVICIOS COMO NUEVO PARADIGMA PARA LA CONSTRUCCION DE SOFTWARE EMPRESARIAL

Elias Francisco Castro Camargo
Nayib Zamir Alarcón Ahumada

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA DE SISTEMAS
CARTAGENA JUNIO 2011

VALIDACION DE LA COMPUTACION ORIENTADA A SERVICIOS COMO NUEVO PARADIGMA PARA LA CONSTRUCCION DE SOFTWARE EMPRESARIAL

Autor

Elias Francisco Castro Camargo
Nayib Zamir Alarcon Ahumada

Trabajo presentado para cumplir requisito al titulo
Ingeniero de Sistemas

Director

MI(c). Edwin Puertas Del Castillo

UNIVERSIDAD TECNOLÓGICA DE BOLÍVAR
FACULTAD DE INGENIERÍA DE SISTEMAS
CARTAGENA JUNIO 2011

Cartagena de Indias, D.T. y C. 20 de Junio de 2011.

Msc. Moisés Ramon Quintana Alvarez
Director de Programa de Ingeniería de Sistemas
UNIVERSIDAD TECNOLOGICA DE BOLIVAR

Cordial saludo.

Por medio de la presente me permito presentar a usted para que sea puesto a consideración y evaluación el trabajo de grado titulado **"VALIDACION DE LA COMPUTACION ORIENTADA A SERVICIOS COMO NUEVO PARADIGMA PARA LA CONSTRUCCION DE SOFTWARE EMPRESARIAL"**, realizado por mi persona como requisito de grado para optar por el título de **INGENIERO DE SISTEMAS**.

Agradeciendo de antemano la atención prestada.

Atentamente,

_____

ELIAS FRANCISCO CASTRO CAMARGO

Cartagena de Indias, D.T. y C. 20 de Junio de 2011.

Msc. Moisés Ramon Quintana Alvarez
Director de Programa de Ingeniería de Sistemas
UNIVERSIDAD TECNOLOGICA DE BOLIVAR

Cordial saludo.

Por medio de la presente me permito presentar a usted para que sea puesto a consideración y evaluación el trabajo de grado titulado **"VALIDACION DE LA COMPUTACION ORIENTADA A SERVICIOS COMO NUEVO PARADIGMA PARA LA CONSTRUCCION DE SOFTWARE EMPRESARIAL"**, realizado por mi persona como requisito de grado para optar por el título de **INGENIERO DE SISTEMAS**.

Agradeciendo de antemano la atención prestada.

Atentamente,

_____

NAYIB ZAMIR ALARCON AHUMADA

AGRADECMIENTOS

Gracias a Dios por darme la oportunidad de crecer en una familia tan generosa, afectuosa y sobre todo llena de amor. Siempre han sido y serán el motor que impulsa mi vida.

A mi padre Ernesto José, por nunca dejar de creer en mí,

A mi madre Liz María por siempre tener una voz de aliento para mí siempre que la necesite,

A mis hermanos Liz María y Hernando José por permitir estar en sus vidas y ofrecerme su amor.

Este trabajo no hubiese sido posible sin el apoyo de Edwin, gracias por apostar en mí,

A Yuliana y a Vanesita por permitir compartir y motivar a seguir trabajando.

Al profesor Giovanny, por su sabiduría y orientación.

Y por ultimo y no menos importante a mi compañero en este gran reto, Nayib, gracias por hacer este trabajo posible.

Elías Francisco Castro Camargo

Agradezco infinitamente a Dios,

Por haber me dado la disciplina y la voluntad para alcanzar este escalón mas en mi vida,

Doy gracias a mis padres José Joaquín y Martha Cecilia,

Quienes siempre me brindaron su apoyo y afecto,

Y cuyo esfuerzo y dedicación fueron una gran inspiración,

Para finalizar mis estudios profesionales,

A mis hermanos José Alberto y Jaifa María acompañantes en este viaje,

A mi director de Tesis Edwin Puertas y al profe Giovanny Vásquez por sus sabios consejos,

A Yuliana amiga y compañera de gran parte de la travesía,

y su hija Vanesita por regalarnos tanto tiempo de su madre,

Y a todos aquellos que hicieron posible este trabajo.

Nayib Zamir Alarcon Ahumada

TABLA DE CONTENIDO

1. PLANTEAMIENTO DEL PROBLEMA 11
   1.1. Descripción del problema 11
   1.2. Formulación del problema 13
2. JUSTIFICACIÓN 14
3. OBJETIVOS 16
   3.1. Objetivo general 16
   3.2. Objetivos Específicos 16
4. INTRODUCCIÓN Y ANTECEDENTES 17
   4.1. Propósito 17
   4.2. Importancia 17
   4.3. Introducción a la Computación orientada a servicio 17
      4.3.1. Ad Hoc Network Programming 18
      4.3.2. Comunicación Estructurada 19
      4.3.3. Middleware 20
      4.3.4. Arquitectura Orientada a Servicios y Servicios Web 22
5. REVISIÓN LITERARIA 24
   5.1. Computación Orientada a Servicios 24
      5.1.1. Servicio 25
      5.1.2. Modelo para la distribución de Software Orientado a Servicios 26
      5.1.3. Modelo para el desarrollo de Software Orientado a Servicios 27
      5.1.4. Modelo de distribución de Software Orientado a Servicios Vs. Modelo de construcción de Software Orientado a Servicios 29
      5.1.5. Retos de investigación en la computación orientada a servicios 31
   5.2. Modelo de Desarrollo de Software 32
      5.2.1. Orígenes e influencias de la COS 32
      5.2.2. Metas de diseño del desarrollo de software orientado a servicios 38
      5.2.3. Principios de diseño del desarrollo de software orientado a servicio 41
   5.3. Arquitectura Orientada a Servicios (SOA) 64
      5.3.1. Arquitectura 65
      5.3.2. Servicios Web 67
      5.3.3. Estándares básicos de los Servicios Web 69

6. CASO ESTUDIO 74
6.1. Alcance 74
6.2. Metodología 74
6.3. Desarrollo del sistema 75
6.3.1. Diseño 75
6.3.2. Implementación 87
6.3.3. Despliegue 89
6.3.4. Pruebas 90
7. VERIFICACIÓN DE LOS PRINCIPIOS DE DISEÑO DE LA COS 91
7.1. Estandarización De Contratos 91
7.2. Bajo Acoplamiento. 92
7.3. Reusabilidad. 93
7.4. Autonomía. 93
7.5. Manejo de Estado. 94
7.6. Composición 94
7.7. Descubrimiento 95
7.8. Abstracción 95
8. EXPERIENCIAS 97
9. CONCLUSIONES 100
10. REFERENCIAS 104

TABLA DE FIGURAS

Figura 1 - Computación Orientada a Servicio 24
Figura 2 - Relación de Conceptos COS 29
Figura 3 - Niveles de la COS. Las capas arquitecturales ofrecen una separación lógica de funcionalidad, mientras que el eje perpendicular indica características de los servicios transversales a los tres planos [26]. 32
Figura 4 - Influencias del Modelos de desarrollo COS 33
Figura 5 - Programación Orientada a Aspectos 34
Figura 6 - Procesos Organizacionales Internos. Tomado de A Computer Scientist's Introductory Guide to Business Process Management (BPM) 35
Figura 7 - Modelos de capas del software empresarial 36
Figura 8 - Servicios con operaciones orientadas a mensajes 41
Figura 9 - Composición de un Servicio de Contrato 42
Figura 10 - Intercambio de mensajes con estructuras de datos homogenizadas 43
Figura 11 - Relaciones de dependencia entre artefactos de software 45
Figura 12 - Dependencia en tiempo de ejecución entre servicios 45
Figura 13 - Acoplamiento del contrato con la tecnología 47
Figura 14 - Acoplamiento a nivel interno de la arquitectura de un servicio y su relación con su contrato -Tomado de [7] 49
Figura 15 - Acoplamiento entre el servicio y sus consumidores 50
Figura 16 - Acoplamiento del consumidor con el contrato del servicio 51
Figura 17 - Abstracción de la información publicada de un servicio - Tomada de [7] 52
Figura 18 - Consecuencias de la encapsulación sobre la abstracción [7] 54
Figura 19 - Autonomía sobre los recursos encapsulados [7] 58
Figura 20 - Tipos de estados 61
Figura 21 - Modelo de orquestación 67
Figura 22 - Arquitectura SOA basada en Servicios Web 72
Figura 23 - Modelo del dominio 77
Figura 24 - Diagrama de clases 78
Figura 25 - Diagrama de secuencia: Selección de Proveedores 79
Figura 26 - Diagrama de secuencia: Evaluación de servicios 80
Figura 27 - Diagrama de entidad relación 81
Figura 28 - Diagrama de componentes 89

LISTADO DE TABLAS

Tabla 1 - Comparativa Modelo de Distribución SaaS Vs. Legado 27
Tabla 2 - Comparativa Zachman Modelo de Construcción COS Vs. Modelo de Distribución COS 31
Tabla 3 - Servicios Candidatos 76
Tabla 4- Aplicación principio de abstracción 96

1. PLANTEAMIENTO DEL PROBLEMA

1.1. Descripción del problema

La llegada de diferentes paradigmas y tecnologías que han surgido en las distintas etapas de la computación, como es el caso de la programación estructurada, la programación orientada a objetos, la programación orientada a aspectos, las bases de datos relacionales, la construcción de software distribuido, entre otros, han posibilitado la construcción de software empresarial de gran complejidad y alta funcionalidad, que en algún momento de la historia llegaron a ser inconcebibles.

Esta relación, negocio – software con el tiempo se ha venido estrechando, convirtiendo al software en un factor clave para las empresas que deseen automatizar, monitorear, gestionar, optimizar e integrar sus procesos de negocio. En la actualidad, el desarrollo de software empresarial enfrenta nuevas necesidades que dan lugar a nuevos retos para la ingeniería de software, los que a su vez han propiciado el surgimiento de nuevos enfoques, paradigmas, filosofías y/o tecnologías para la construcción de software.

Algunos de los principales desafíos que afronta la construcción de software empresarial en nuestros tiempos son:

- Los negocios están en constante evolución, en pro de responder y sobrevivir a los modelos económicos actuales, por tanto, el software debe ser flexible para adaptarse rápidamente a los cambios.
- El software debe responder a las necesidades del negocio, no al contrario. Los verdaderos directores de los cambios que tomará el negocio, son los directores administrativos, ejecutivos y el personal ajeno a la tecnología, con alto conocimiento de los procesos de negocio, pero con bajo conocimiento del funcionamiento de las tecnologías que lo soportan.

- Las organizaciones se conforman por diferentes procesos de negocios, los cuales generalmente son automatizados con el uso de aplicaciones específicas que podrían ser independientes y estar aisladas, como es el caso de las aplicaciones para la captura de solicitudes, para la gestión de recursos humanos, para la gestión de los clientes, entre otras. Es este un ambiente poco propicio para el mantenimiento y evolución de las aplicaciones, así como para la integración de procesos de negocios y por ende la integración de aplicaciones.

Entre las posibles soluciones a estas necesidades, nace el paradigma de la Computación Orientada a Servicios o por sus siglas en ingles SOC (Service-Oriented Computing). Los autores Michael P. Papazoglou y Dimitrios Georgakopoulos lo definen como “un paradigma que utiliza los servicios como elemento fundamental para soportar el desarrollo rápido y de bajo costo, de aplicaciones distribuidas en ambientes heterogéneos”.

La COS está regida por un conjunto de principios, conceptos y objetivos, que definen un conjunto de elementos necesarios para la construcción de soluciones lógicas orientadas a servicios, que comúnmente son diseñadas para ser desplegadas bajo el estilo arquitectónico SOA (Service-Oriented Architecture) e implementadas utilizando Servicios Web.

En la actualidad este paradigma se encuentra en una etapa de maduración, en la que aún hay muchos retos de investigación, así como su consolidación en los campos académico y empresarial, lo que ha impulsado múltiples estudios en esta área del conocimiento. Para el caso específico de la Universidad Tecnológica de Bolívar, no se evidencian trabajos de investigación relacionados con la COS, sin embargo, en la actualidad se dispone de recursos bibliográficos que permiten dar los primeros pasos en el conocimiento y apropiación de este nuevo paradigma.

Esta carencia de conocimiento reduce la capacidad de construir software orientado a las necesidades de las Pymes locales, que comprenda un conjunto soluciones lógicas que además de ser agnósticas y transversales a los diferentes

procesos de negocio, ya sean individuales o colaborativos, también sean altamente reusables y con un alto retorno de inversión, y que además reflejen un comportamiento más acorde a los requerimiento económicos y de escalabilidad de dichas organizaciones. De aquí la importancia del estudio del modelo de construcción y distribución (SaaS), propuestos por la computación orientada a servicios.

1.2. Formulación del problema

¿Es adecuado desarrollar software empresarial a través del uso de los modelos de desarrollo y distribución propuestos por la computación orientada a servicio, para dar solución a problemática de construcción de software actual?

2. JUSTIFICACIÓN

El ambiente de operación de un software empresarial está regido por la heterogeneidad. Al mirar cómo están conformadas las organizaciones, se observa que de ellas hacen parte un conjunto de sub-organizaciones, departamentos, secciones, grupos, dependencias, etc., cada una de las cuales realizan diferentes actividades bajo criterios que pueden ser propios o dirigidos por los propósitos, objetivos, políticas y recursos propios de la organización.

Desde una perspectiva más técnica para este ambiente de operación, es posible encontrar las siguientes condiciones:

- La no existencia de un software en las organizaciones que cubra todos los procesos de negocio, por el contrario existen diferentes aplicaciones, que soportan uno o más procesos de negocio, las cuales pueden divergir en: sistema operativo bajo el cual corren, plataforma de ejecución, entorno de red, propietario, estilos arquitecturales y paradigmas de programación bajo los cuales fueron desarrollados, gestores y fuentes de datos a los que acceden, Middlewares de comunicación, modelos de transmisión de información, políticas de seguridad, entre otras.
- Divergencia en modelo de datos.
- Problemas en la colaboración y negociación, debido a las políticas que rigen cada organismo.
- Redundancia de aplicativos.

La constante interacción de estos elementos, a través del uso de los recursos y/o actividades que cada una de ellos brinda, es necesaria para la conformación de procesos de negocios, interacciones que en términos informáticos se traducen en intercambios de información.

Este es el panorama que debe ser afrontado por los ingenieros de software de la actualidad. En [11] se plantea la importancia de la enseñanza de la COS como primer paradigma, sin olvidar que este se encuentra en una etapa de maduración,

en la que aún hay muchos retos de investigación, así como su consolidación en los campos académico y empresarial, lo que ha impulsado múltiples estudios en esta área del conocimiento.

Para el caso específico de la Universidad Tecnológica de Bolívar, no se evidencian trabajos de investigación relacionados con la COS, sin embargo, en la actualidad se dispone de recursos bibliográficos que permiten dar los primeros pasos en el conocimiento y apropiación de este nuevo paradigma.

Un trabajo que consolide el marco teórico y brinde acceso a una experiencia de desarrollo bajo el modelo de la COS, permitirá a la comunidad académica de la UTB e incluso de la Región, tener un marco de referencia para futuras investigaciones e inclusión de la temática en las mallas curriculares de los programas académicos afines.

3. OBJETIVOS

3.1. Objetivo general

Validar el paradigma de la Computación Orientada Servicio a través del estudio de sus conceptos y principios, y la aplicación de los mismos en el diseño de una solución para la selección de proveedores de la Universidad Tecnológica de Bolívar.

3.2. Objetivos Específicos

- Describir el estado del arte de los modelos, enfoques y estrategias utilizados para la comunicación de aplicaciones.
- Describir una revisión bibliográfica referente a la computación Orientada a servicios.
- Desarrollar una solución para la selección de proveedores de la Universidad Tecnológica de Bolívar basada en el paradigma de construcción de software propuesto por la computación orientada a servicios.
- Verificar y validar el cumplimiento de los principios de diseño de la computación orientada a servicio en la solución desarrollada para la selección de proveedores de la Universidad Tecnológica de Bolívar.

4. INTRODUCCIÓN Y ANTECEDENTES

Esta sección se proporciona una introducción a este informe y una contextualización a la computación orientada a servicio y entorno donde se desarrolla.

4.1. Propósito

Verificar y validar los conceptos, principios e importancia de la computación orientada a servicios, como una solución a la problemática actual en el desarrollo de software empresarial, a través de su aplicación a un caso de estudio real.

4.2. Importancia

La computación orientada a servicios propone modelos de desarrollo y distribución de software que aplicados de forma correcta reducen los costos de construcción, infraestructura, tiempo de desarrollo y despliegue de los sistemas de información empresariales.

4.3. Introducción a la Computación orientada a servicio

Los sistemas distribuidos son comúnmente piezas complejas de software, cuyos componentes, por definición se encuentran dispersos en múltiples maquinas, la forma de coordinar y controlar esta complejidad es crucial para la correcta organización del sistema [1].

La organización de un sistema distribuido se concentra mayormente en los componentes de software que lo constituyen. Las arquitecturas de software describen como estos componentes están organizados y cómo interactúan, y los estilos arquitecturales definen la forma como estos componentes se conectan, como intercambian información y como son configurados en conjunto con el sistema.

A continuación se describe una línea de tiempo de los diferentes estilos arquitecturales, ahondando en el contexto en que se originaron, es decir las

problemáticas que abordaron, la forma como se dio respuesta a esta y las tecnologías que los sustentaron.

4.3.1. Ad Hoc Network Programming

A mediados de los años 80's, el surgimiento de los microprocesadores y las redes de datos, abrió la posibilidad de comunicar a las computadoras y sus componentes de software, entre sí, lo cual llevo a la creación de los mecanismos de comunicación de interprocesos (IPC), que permiten la cooperación de procesos con diferentes espacios de memoria aun cuando estos se encuentran en diferentes maquinas.

Los principales mecanismos IPC son:

- Sockets: Es un concepto abstracto que hace referencia a un punto de comunicación bidireccional, a través del cual dos programas pueden intercambiar flujos de datos de manera confiable y ordenada.
- Memoria Compartida: Los procesos que se ejecutan en una misma máquina tienen espacios de direcciones diferentes, por lo tanto, el espacio de memoria utilizado por un proceso no podrá ser accedido por ningún otro. La memoria compartida permite la cooperación entre procesos, al crear un espacio en memoria en el cual se pueda compartir información. Esta área puede ser creada por un proceso y posteriormente escrita y leída por cualquier número de procesos.
- Tuberías (pipes): consiste en una cadena de procesos conectados de forma tal que la salida de cada elemento de la cadena es la entrada del próximo.

Los sistemas diseñados actualmente con base en este estilo arquitectural, son escasos debido al gran esfuerzo requerido para su implementación y mantenimiento, su poca flexibilidad y su gran dependencia del ambiente de operación.

4.3.2. Comunicación Estructurada

La comunicación estructurada nace como respuesta a la necesidad de desacoplar los aplicativos de los mecanismos IPC, creando mecanismos de comunicación de alto nivel a través del encapsulado de detalles de bajo nivel, tales como la lectura y escritura de bits, bytes, etc. De esta forma el estilo de comunicación estructurada se asemeja mucho a los estilos de programación de aplicaciones, implicando el uso de tipos y estructura de datos.

En la historia de los sistemas distribuidos, el ejemplo más significativo de este tipo de arquitectura son los RPC o Llamadas a Procedimientos Remotos, como es el caso de Sun RPC, DCOM, y DCE. Estas plataformas permiten la colaboración en aplicaciones distribuidas a través de la invocación de funciones entre ellas, las cuales reciben parámetros y retornan el resultado a quien la invoca, lo cual permitió a los desarrolladores familiarizarse con este tipo de comunicación debido a sus semejanzas con la programación orientada a objetos.

A pesar de la capa de abstracción lograda con la comunicación estructurada una gran variedad de problemas inherentes a los sistemas distribuidos, aun no pudieron ser resueltas, dichos problemas son listadas por [2] y se describen a continuación:

- Transparencia en la ubicación de los componentes: Dentro de un sistema distribuido, idealmente, la comunicación con recursos locales o remotos debería realizarse con el mismo modelo de programación, lograr esta independencia en la ubicación de los recursos requiere que el código sea separado de detalles específicos de las ubicaciones de la implementación.
- Despliegue y redespliegue flexible de componentes: En muchas situaciones el despliegue original de un componente puede dejar de ser la mejor manera de solucionar la problemática para la que fue creada, tales situaciones como el mejoramiento de hardware, la incorporación de nuevos nodos al sistema o la aparición de nuevos requerimientos, hacen necesario el redespliegue del

componente, que en un sistema distribuido debe realizarse sin romper la continuidad del sistema ni mucho menos el apagado del mismo.

- Integración concódigo heredado: Los sistemas distribuidos por diferentes razones, de costo y tiempos, deben ser capaces de integrar elementos y aplicaciones que no fueron desarrolladas para trabajar en un ambiente distribuido.
- Componentes heterogéneos: Una de las características de un sistema distribuido ideal es que sus componentes puedan ser desarrolladas en diferentes lenguajes y ambientes, permitiendo ser integrados en un solo y coherente sistema distribuido.

4.3.3. Middleware

Debido a la gran variedad de dispositivos de cómputo y software que existen, la comunicación y cooperación entre estos, se ha convertido en una necesidad que día a día toma mayor relevancia. Como respuesta a esta necesidad aparece el concepto de middleware.

Un middleware desarrolla una infraestructura distribuida de software dedicado, el cual reside entre las aplicaciones, y el sistema operativo, las bases de datos, la red, etc., que permite construir sistemas distribuidos con características que con arquitecturas tradicionales no sería posible. La forma cómo se despliega esta capa o software dedicado ha dado lugar a estilos arquitecturales que se detallan a continuación.

4.3.3.1. Middleware de Componentes

Un componente de software puede definirse como una pieza no trivial de software, un módulo, un paquete o un subsistema que completa una función clara, tiene un límite claro y puede ser integrado en una arquitectura bien definida [3].

La aplicación de este concepto en sistemas distribuidos es lo que permite el nacimiento de este estilo arquitectural, el cual es una extensión de los middleware de objetos distribuidos y es una respuesta a las limitaciones que contienen estos,

aborda las limitaciones funcionales permitiendo a objetos agruparse de manera coherente en componentes, a través del uso de múltiples interfaces, y la definición de mecanismos estandarizados necesarios para la ejecución de dichos componentes servidores de aplicaciones genéricas. Las limitaciones en el despliegue de software y las configuraciones son superadas con la especificación de infraestructuras para encapsulamiento, personalización, ensamble y distribución de componentes sobre sistemas distribuidos.

Algunos ejemplos de estas tecnologías son los Enterprise JavaBeans y CORBA Component Model.

La aparición de este tipo de middleware permitió la automatización de ciertos aspectos en el ciclo de vida de las aplicaciones de software, tales como: el empaquetamiento, el ensamblaje y el despliegue, lo que ha facilitado construcción de aplicaciones, de forma más rápida y robusta que con el middleware de objetos distribuidos.

4.3.3.2. Middleware orientados al mensaje

Las arquitecturas que se han estudiado en este capítulo, poseen un modelo común de comunicación de tipo cliente - servidor, en el cual el cliente realiza peticiones que son respondidas por el servidor, lo que implica que la comunicación solo puede ser iniciada por el cliente, por lo cual este modelo, resulta insuficiente para cierto tipo de aplicaciones distribuidas, que deben reaccionar a estímulos externos o eventos, y en los cuales el trabajo colaborativo entre maquinas es importante, como es el caso de los sistemas de control o sistemas de negociación de bolsa de valores.

Los middleware orientados al mensaje, nacen como una solución a este tipo de sistemas, ofreciendo una pasarela de mensajes, que permite a las aplicaciones intercambiar información asíncronamente y de forma desacoplada, evitando el bloqueo durante la espera de la respuesta. Además proveen frecuentemente propiedades transaccionales como colas de confiabilidad y persistencia hasta que el consumidor pueda recoger los mensajes [2].

Hay dos subtipos de middleware orientados a mensajes, los de mensajería punto a punto y los de publicación/suscripción. En los primeros un sistema de mensajes redirige los mensajes desde el emisor hasta la cola de mensajes del receptor, donde el middleware proporciona un depósito de mensajes, permitiendo el desacoplamiento del envío y la recepción, también proporcionan abstracciones que facilitan la extracción de los mensajes y el manejo asíncrono y paralelo de las operaciones necesarias para el procesamiento de los mismos.

Para el caso de los middleware de publicación/suscripción, están basados en la comunicación anónima, donde hay un débil acoplamiento entre el publicador y el consumidor, y por lo tanto no conocen la existencia del otro, lo cual permite que múltiples subscriptores reciban datos de un publicador. Este tipo de middleware permite a las aplicaciones correr en nodos distintos y realizar eventos de escritura-lectura sobre un espacio global de datos en un sistema distribuido. Las aplicaciones usan este espacio global para compartir información con otras aplicaciones, declarando su intención de producir eventos, lo cual frecuentemente es categorizado en tópicos de interés, así si otra aplicación que dese consumir debe declarar su intención en el espacio global.

Los elementos del middleware de publicación/suscripción son separados en los siguientes roles:

- Publicadores: son las fuentes de eventos, producen eventos en tópicos específicos que son propagados en el sistema.
- Suscriptores: son quienes reciben los eventos, es decir reciben información de tópicos que les interesan.
- Canales de eventos: Son los componentes del sistema que propagan los eventos del publicador al suscriptor.

4.3.4. Arquitectura Orientada a Servicios y Servicios Web

Una Arquitectura orientada a servicios (SOA) es un estilo de organización y utilización de las capacidades distribuidas que pueden ser controlados por las

diferentes organizaciones o propietarios. Por lo tanto, proporciona un modo uniforme para ofrecer, descubrir, interactuar y utilizar estas capacidades débilmente acopladas y servicios de software interoperables para soportar las necesidades de los procesos de negocios y los usuarios de la aplicación [4].

En una arquitectura orientada a servicios, se diferencia claramente tres elementos básicos:

- Servicio: conjunto de actividades que son ofrecidas por una o más entidades, y que responden a necesidades específicas de un grupo de clientes, los cuales son los encargados de consumirlo.
- Proveedor: Ente encargado de implementar, poner en marcha y presentar el servicio.
- Consumidor: Ente que consume los servicios.

SOA, propone un conjunto de directivas para la elaboración de software en ambientes distribuidos, pero como tal no específica que tecnologías y estándares se usaran en la implementación específica de un sistema, es en este punto que los servicios web han tomado importancia, como la principal variante de SOA.

Los servicios web son una tecnología que permite encapsular lógica de aplicación en servicios, a través del uso de estándares dictados por el World Wide Web Consortium (W3C), que define los principales aspectos de dichos servicios, entre los más importante tenemos el lenguaje de descripción de servicios web o WSDL.

5. REVISIÓN LITERARIA

5.1. Computación Orientada a Servicios

La computación orientada a servicios (COS) es una novedosa área de la computación que se fundamenta en la filosofía de servicios aplicados a la tecnología, la cual debe ser vista desde una perspectiva alejada de slogans comerciales y apreciaciones individuales que promueven ideas difusas acerca de lo que realmente representa.

Esta área involucra principalmente las siguientes disciplinas de la computación: sistemas distribuidos, arquitecturas de software, grid computing, ingeniería de software, lenguajes de programación, sistemas multi- agentes, sistemas de bases de datos, seguridad, calidad de servicio y sistemas basados en conocimiento.

En la computación orientada a servicio, es posible encontrar dos modelos para la representación del concepto de servicio, el **"modelo para la distribución de software"** y el **"modelo para la construcción de software"** [6]. Como se indica en la figura 4.1.

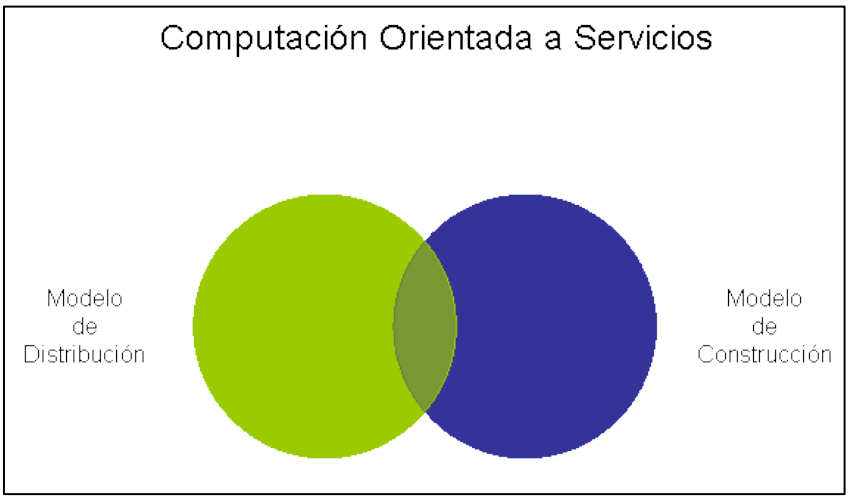

Figura 1 - Computación Orientada a Servicio

En la actualidad la diferentes empresas, organismos gubernamentales, universidades e institutos de investigación han centrado su atención en la COS, lo cual se evidencia en una serie de estudios realizados en [7], a partir de los cuales, se concluye que las mayores compañías de computación están involucradas con la COS, empresas tales como: BEA, IBM, Microsoft, Oracle, HP, SAP, Intel, Cisco, Juniper, Sun Microsystem.

5.1.1. Servicio

En este apartado se pretende abrir un panorama general acerca del concepto de servicio como eje fundamental de la COS. Hay que resaltar que este marco conceptual es bastante árido e incluso no es posible establecer una definición general de servicio. IBM lidero la iniciativa de estudiar y abordar el marco de los servicios como una ciencia, dando como resultado la constitución de la "Service Science, Management and Engineering" como un área de estudio formal.

En el ciclo de vida de un servicio se reconocen dos principales entes que interactúan con él, proveedores y consumidores. Los primeros se refieren a una entidad (persona, negocio o institución) que hace preparativos para satisfacer una necesidad, es decir esta presta a servir a otras entidades. Y los consumidores, también llamados clientes, son entidades (persona, negocio o institución) que consumen un servicio.

En [8] un servicio en términos generales tiene las siguientes características básicas:

- Son valores que pueden ser rentados.
- Poseen autonomía.
- Son independientes de plataforma.
- Representan funciones de negocios.
- Son heterogéneos, desde la perspectiva de los clientes existes variedad de ofertas de servicios.

- La percepción de la calidad oscila de un cliente a otro.
- Son efímeros, cada vez que no se usan a su máxima capacidad el proveedor pierde oportunidades.
- No son tangibles.
- Son complejos al compararlos y evaluarlos antes de comprarlos.
- Su producción y consumo ocurren de forma simultánea.

En [9] se categorizan los servicios como:

- Servicios intelectuales y espirituales: en esta categoría se brindan ideas, principios y conceptos, así como bases para actividades de negocio e interacción con clientes.
- Servicios de comportamiento: estos se refieren a expresiones, gestos y creación de atmosferas para la interacción del cliente.
- Servicios operacionales y de funcionalidad del negocio: se refieren a la provisión de bienes intangibles y creación directa de valor económico

En el ámbito de la computación también es posible evidenciar entes con las características de un servicio, tales como: servicios de almacenamiento, de procesamiento, de hosting, de seguridad, de red, de sistema, lógicos y operativos.

5.1.2. Modelo para la distribución de Software Orientado a Servicios

SaaS (Software as a Service), es un modelo para la distribución de software orientado a servicio, que cambia la forma de ofrecer el software, no como producto, si no como servicio. [10] plantea que la visión principal de SaaS es separar la posesión y propiedad del software, de su uso.

Un producto se define como una entidad canjeable, que puede ser adquirido, y distribuido a diferentes localidades y en diferentes momentos, preservando su identidad, desde la perspectiva de los consumidores, su ciclo de vida cumple las siguientes actividades:

- Determinar los requerimientos y justificar la compra del producto.
- Encontrar un proveedor del producto.
- Financiar el producto.
- Instalar el producto.
- Modificar otros productos o procesos para trabajar con el producto.
- Mantener el producto y remplazar partes.
- Entrenar al personal para la utilización del producto.
- Actualizar el producto.
- Eliminar los residuos del producto.
- Eliminar el producto.

Existen diferencias notables entre el modelo de distribución de software legado y el SaaS, las cuales se detallan a continuación en la tabla 4.1.

Característica	Modelo de Distribución	
	Saas	Legado
Entregable	Colección de aplicaciones estandarizadas y servicios.	Aplicaciones personalizadas para los usuarios.
Forma de Pago	Es rentado, se debe pagar el uso de sus servicios.	Es adquirido o comprado.
Métodos de Distribución	Se trata de aplicaciones que principalmente están alojadas en servidores centralizados o distribuidos en la nube.	Se rige bajo la premisa de ser instalado.

Tabla 1 - Comparativa Modelo de Distribución SaaS Vs. Legado

5.1.3. Modelo para el desarrollo de Software Orientado a Servicios

Es un paradigma computacional que utiliza servicios como los elementos fundamentales para el desarrollo rápido y de bajo costo, de aplicaciones

distribuidas en ambientes heterogéneos [1]. En [11] lo define como un paradigma que se refiere a un conjunto de conceptos, principios y métodos en los cuales el software es construido con base en la composición de servicios con interfaces estándares.

El modelo para el desarrollo de software propuesto por la COS encapsula diferentes elementos y conceptos, que incluyen paradigmas, principios y catálogos de patrones de diseño, distintos modelos arquitecturales, así como tecnologías y Frameworks relacionados.

La aplicación del conjunto de principios de diseño que rigen a la computación orientada a servicios, da como resultado una solución lógica orientada a servicios, constituida por su más fundamental unidad de representación lógica, los servicios.

En [7] se describen los principales elementos que abarca el modelo para el desarrollo de Software Orientado a Servicios, de la siguiente manera:

Los servicios existen como artefactos de software físicamente independientes con diferentes características de diseño que soportan la realización de las metas estratégicas asociadas con la computación orientada a servicios. Cada servicio es asignado a su propio contexto funcional y este a su vez abarca un conjunto de funcionalidades relacionadas con él. Estas capacidades están disponibles para ser invocadas por programas externos, que actúan como consumidores, capacidades que son comúnmente expresadas a través de contratos de servicios que han sido publicados, muy parecidos a una API tradicional.

La computación orientada a servicios representa una plataforma de computación distribuida de última generación, construida sobre plataformas de cómputo distribuido anterior, agregándoles nuevas capas de diseño, consideraciones de gobierno, y un vasto conjunto de tecnologías predilectas para su implementación.

La arquitectura orientada a servicios (SOA), se refiere a un estilo arquitectural, basado en este modelo de desarrollo, el cual define un conjunto de elementos y

sus relaciones, para la implementación y despliegue de servicios, que apunta a mejorar la eficiencia, agilidad y productividad de una empresa, a través del posicionamiento de los servicios como el medio principal sobre el cual las soluciones lógicas son representadas en soporte de la realización de las metas estratégicas asociadas a la computación orientada a servicios.

Cabe resaltar que el término SOA (Service Oriented Architecture) es muchas veces usado de forma inadecuada como sinónimo de computación orientada a servicios, como se evidencia en la figura 2.

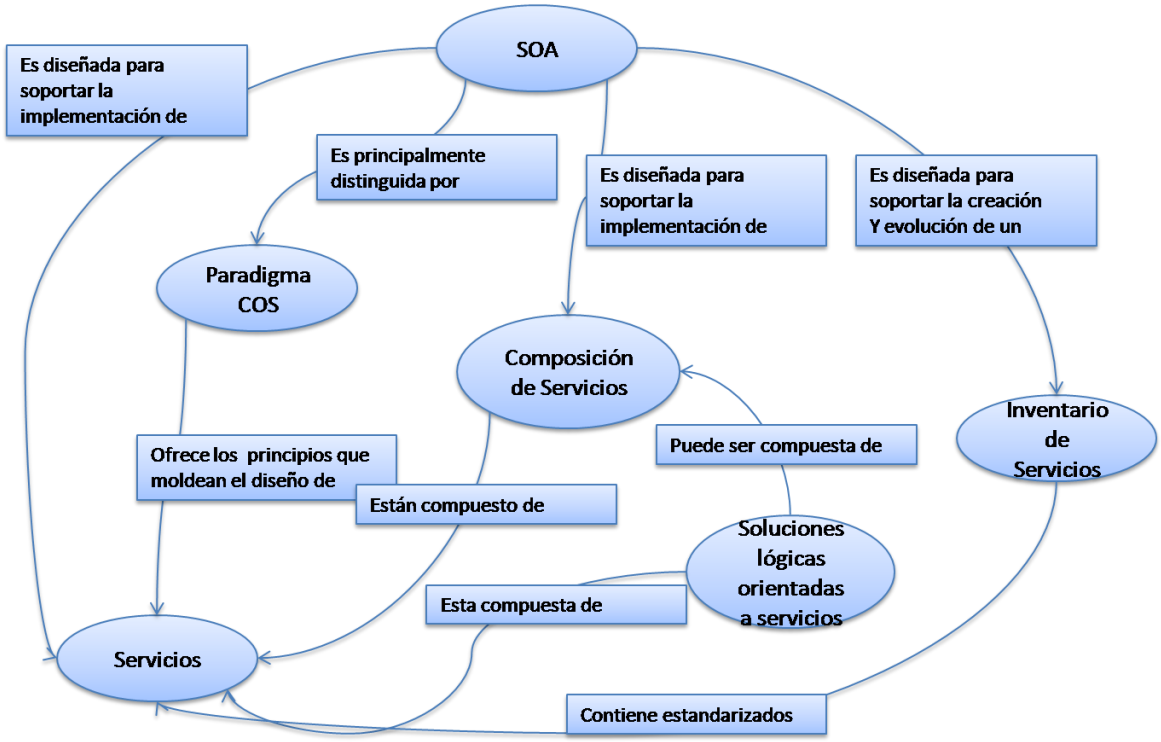

Figura 2 - Relación de Conceptos COS

5.1.4. Modelo de distribución de Software Orientado a Servicios Vs. Modelo de construcción de Software Orientado a Servicios

Es importante, hacer una comparativa entre el modelo de distribución y el modelo de construcción de software, resaltando que estos modelos no son excluyentes, y

por el contrario se complementan el uno con el otro, lo cual se evidencia en que un software orientado a servicios, permite una rápida distribución a través del modelo SaaS.

En [6] se ofrece una comparativa desde las diferentes perspectivas de los sistemas de información desarrollados bajo los dos modelos, utilizando el framework para la descripción de arquitecturas de Zachman, la cual se muestra en la tabla 2.

Perspectiva de los Stakeholders	**Red (Modelo de Construcción)**	**Red (Modelo de Distribución)**
Objetivo/Alcance	Lista de posibles servicios para usar	Lista de posibles servicios para entregar
Modelo del negocio	Lista de posibles servicios de negocio para usar	Lista de posibles servicios de negocio para proveer
Modelo del sistema de información	Modelo de interacción de componentes de servicios	Modelo de interacción de componentes
Modelo tecnológico	Modelo de interacción de componentes de servicios dependiente de tecnología y de plataforma	Modelo de interacción de componentes dependiente de tecnología y plataforma
Representación detallada	Lista de lenguajes dependientes de tecnología, protocolos usados y servicios actualmente usados	Arquitectura de publicación/suscripción y facilidades de notificación; lista de tecnologías dependientes de lenguaje, protocolos, y servicios usados (si hay alguno)
Funcionalidad del sistema	Comunicación inter-servicio, coordinación y	Comunicación inter-componentes,

	colaboración	coordinación y colaboración

Tabla 2 - Comparativa Zachman Modelo de Construcción COS Vs. Modelo de Distribución COS

5.1.5. Retos de investigación en la computación orientada a servicios

La interrelación entre los niveles de la COS y las diferentes áreas del conocimiento de las ciencias computacionales, implican nuevos retos de investigación de la COS, en [25] se identifican desafíos y retos como: arquitecturas reconfigurables dinámicamente en tiempo de ejecución, soluciones de seguridad, infraestructuras para el soporte de integración de datos y procesos, mejoras en el descubrimiento de servicios con el uso de semántica, en cuanto a la construcción de servicios, entre otros.

En cuanto al proceso de composición de servicios existen desafíos en temáticas como procesos dinámicos y adaptativos, composición de servicios dinámica, composiciones que consideren elementos semánticos, de calidad de servicio y otros requerimientos no funcionales, así como la composición automatizada orientada por los procesos de negocio.

A nivel administrativo y de monitoreo surgen necesidades en cuanto a la administración capaz de automatizar la configuración, adaptación, recuperación, optimización y protección de servicios.

A nivel de diseño y desarrollo los retos están el crecimiento la ingeniería de aplicaciones de servicios, control de versiones, adaptación y gobierno de los mismos.

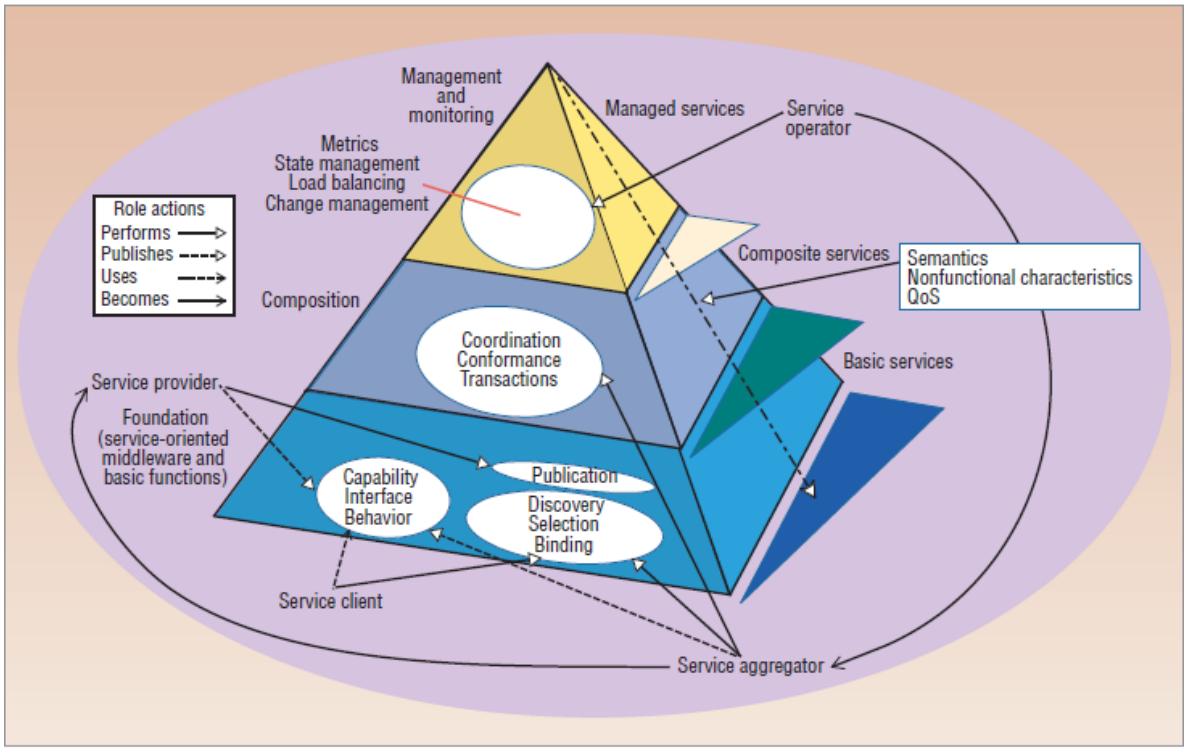

Figura 3 - Niveles de la COS. Las capas arquitecturales ofrecen una separación lógica de funcionalidad, mientras que el eje perpendicular indica características de los servicios transversales a los tres planos [26].

5.2. Modelo de Desarrollo de Software

En la descripción del modelo de desarrollo de software contenido en la COS, se tomó como referente base los contenidos desarrollados por Thomas Erl, tales como [7], [26], [27], [28]; debido a su gran aceptación en la comunidad académica, y el gran número de publicaciones y autores que soportan su producción bibliográfica.

5.2.1. Orígenes e influencias de la COS

La computación orientada a servicios no es un paradigma que surge de la nada, es la recopilación, adaptación y evolución de diferentes filosofías y tecnologías para la construcción de software distribuido, las cuales dan solución a problemáticas que surgieron en diferentes momentos de la historia de la computación. La COS tiene sus raíces en la **programación orientada a objetos**, la **programación orientada a aspectos**, en algunos sistemas **middlewares**,

constituidos para dar solución a algunas necesidades de la **EAI (Integración de aplicaciones empresariales)**; en la apropiación de los beneficios y metas de **BPM**, y finalmente su implementación se ve facilitada por tecnologías como los **Servicio Web**.

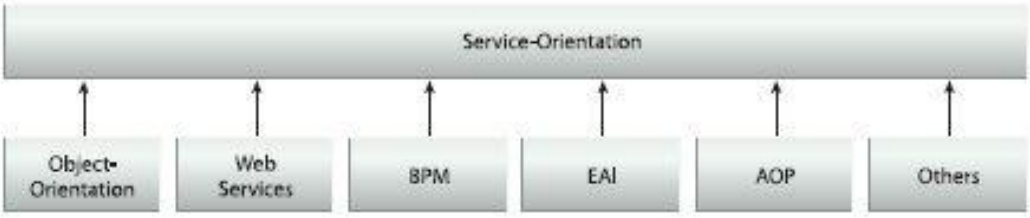

Figura 4 - Influencias del Modelos de desarrollo COS

De la programación orientada a objetos se retoman los principios de abstracción, composición y reusabilidad; paradigma que indudablemente es de gran aceptación y de mayor utilización en la construcción de software para nuestros tiempos, permitiendo la construcción de aplicaciones de gran complejidad y ofreciendo beneficios en su mantenibilidad, desarrollo y prueba.

La tecnología de objetos tiene dificultades para ubicar problemáticas que involucran restricciones globales y comportamientos pandémicos, dificultades para una segregación apropiada de la problemática, y aplicación de conocimiento específico de un dominio. Como respuesta a estas necesidades, surgen tecnologías y paradigmas pos-programación orientada a objetos, una de ellas la programación orientada a aspectos, la cual está basada en la idea que los sistemas computacionales son mejor programados, a través de la especificación independiente de las diferentes propiedades o áreas de interés del sistema y con una cierta descripción de sus relaciones, y luego apoyarse en mecanismos subyacentes al ambiente de POA para entrelazarlos y componerlos en un programa coherente. La problemática puede ir desde nociones de alto nivel como calidad de servicio y seguridad, hasta nociones de bajo nivel como caches y buffers. También pueden ser funcionales, como características o reglas de negocio, o no funcionales, como sincronización y administración de transacciones. Mientras que la tendencia en la POO es buscar similitudes entre clases y luego

utilizarlas en un árbol de herencia, la POA busca convertir necesidades generales en elementos de primera clase, y ubicarlos horizontalmente sobre la estructura de objetos [12].

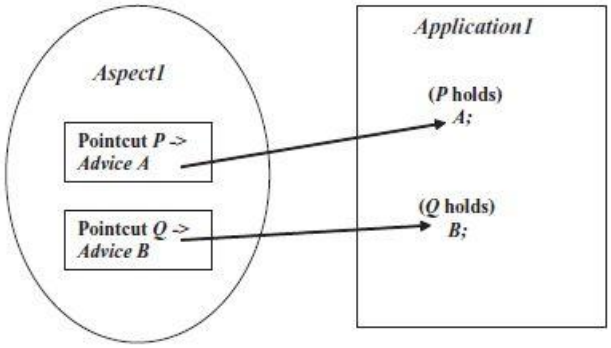

Figura 5 - Programación Orientada a Aspectos

La COS así como la POA, tienen una meta en común, la importancia de construir soluciones lógicas que son agnósticas y transversales a los procesos de negocios y a las aplicaciones, y por lo tanto altamente reusables; Además el desarrollo basado en roles, permitiendo la colaboración entre desarrolladores y los expertos en diferentes áreas.

Otra disciplina con mucha influencia para la COS es BPM (*Business Process Management)*, a tal punto que es posible encontrar que se puede llegar a confundir con SOA o que existe una relación inherente con la COS.

BPM es un término que abarca técnicas para la identificación, documentación y completa administración de los procesos que una organización utiliza para ejecutar tareas individuales, especialmente cuando estas involucran la cooperación de múltiples individuos, departamentos u organizaciones independientes. Las docenas o cientos de procesos dentro de una organización típicamente pueden contar con componentes humanos y de TI, y muchas de las actividades individuales dentro de cualquier proceso también son usadas en conjunto con otras actividades en otros procesos [13].

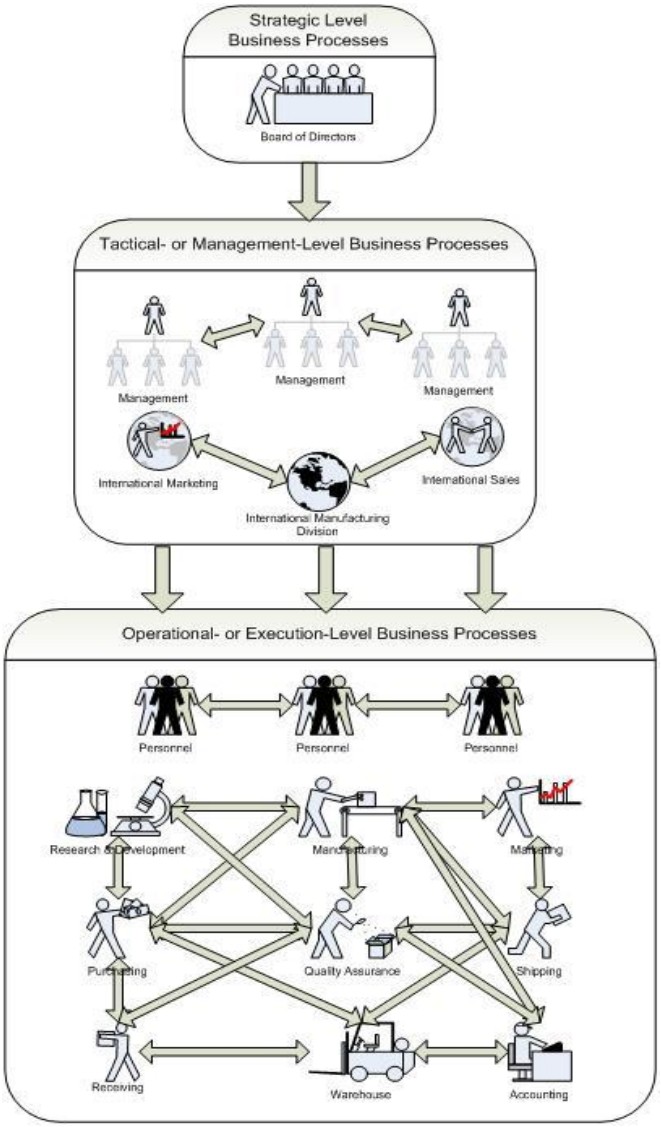

Figura 6 - Procesos Organizacionales Internos. Tomado de A Computer Scientist's Introductory Guide to Business Process Management (BPM)

Los procesos de negocio pueden ser catalogados en procesos privados o públicos. Los privados pueden ser a nivel estratégico, de administración o de operación, y se encuentran bajo el completo dominio de la organización; mientras que los públicos o también denominados procesos de negocios colaborativos involucran organizaciones externas y algunos de sus procesos privados. Estos implican un gran esfuerzo humano para el llenado manual de información e intercambio de documentos, procesos como la compra de suministros, peticiones

de servicios, despacho de mercancía, etc. que indudablemente requieren ser automatizados.

BPM permitió la introducción de una nueva capa de abstracción (workflow layer) en la construcción de software empresarial, que contiene modelos de ejecución de procesos.

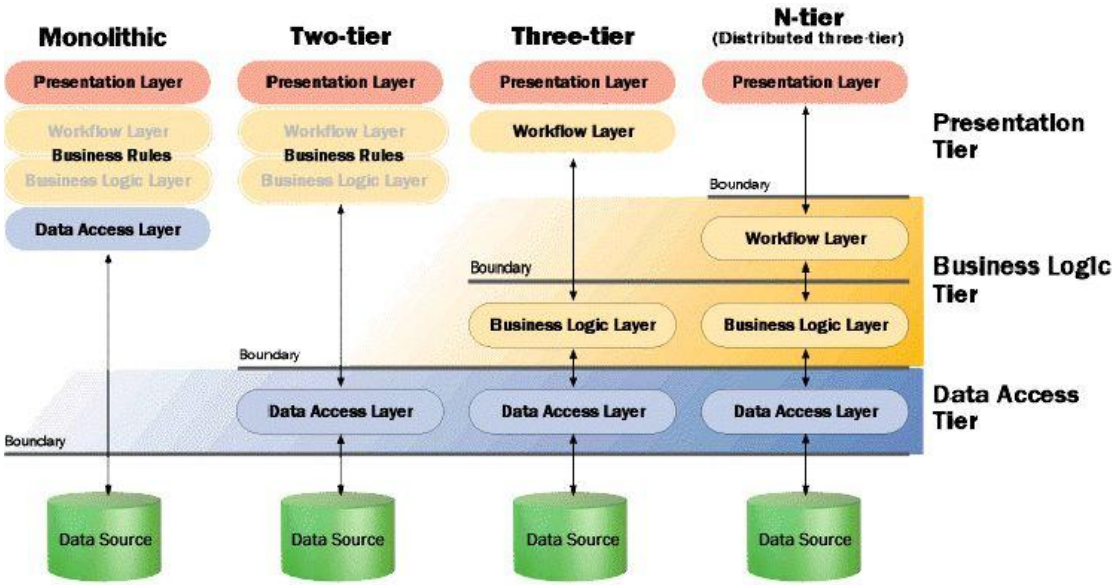

Figura 7 - Modelos de capas del software empresarial

Esta capa es extendida por la COS, a través de la composición de servicios, en la cual a través de la orquestación de servicios se conforman procesos de negocios ya sean públicos o privados, que además cuentan con los beneficios de la COS.

En paralelo a los esfuerzos de BPM se encuentra la Integración de Aplicaciones Empresariales, en su búsqueda de integrar las aplicaciones que soportan los diferentes procesos negocios de las organizaciones. En el contexto de operación de las organizaciones, existe un ambiente heterogéneo, de políticas, estrategias, tecnologías, léxicos y de aplicaciones, un ambiente que posee las siguientes características:

- No existencia de un solo software en las organizaciones que cubra todos los procesos de negocios, por el contrario existen diferentes aplicaciones que soportan uno o más procesos de negocios.
- Cada software puede divergir en:
    - El Sistema Operativo, bajo el cual corren.
    - Plataforma de ejecución, Frameworks.
    - Entorno de red.
    - Propietario, difícil o nula adaptabilidad o extensión.
    - Diferentes estilos arquitecturales.
    - Desarrollados bajo diferentes paradigmas de programación.
    - Acceden a diferentes gestores de bases de datos, que a su vez pueden diferir en modelo y propietario.
    - Corriendo bajo diferentes Middlewares de comunicación, que pueden ser orientadas a mensajes, ORB (Object Request Brokers), a invocaciones de procedimientos remotos, etc.
    - Divergencia en los modelos de transmisión de información, como publish/subscribe, request/reply y conversacionales.
    - Diferentes Middlewares para la administración de transacciones y seguridad.
    - Su modelo de datos, la semántica y estructura de la información varía.
- Las políticas que rigen los organismos o departamentos de una organización son transitivos a los procesos y aplicaciones que estos comprenden, generando problemas de colaboración y negociación.
- Puede existir redundancia de aplicativos.

Para este contexto de operación surgieron diferentes soluciones y perspectivas que atacan diferentes niveles de integración (de datos, de presentación, de aplicaciones, de procesos de negocio, de negocio a negocio o de infraestructura), dando origen a tecnologías y estilos arquitecturales, que con el pasar del tiempo han evolucionado y arrojando resultados tan importantes como los Servicios Web.

5.2.2. Metas de diseño del desarrollo de software orientado a servicios

5.2.2.1. Interoperabilidad inherente

La interoperabilidad es una característica y un factor de calidad de software, que hace referencia a la capacidad que tiene este para intercambiar información con otros.

Un software con baja capacidad de interoperabilidad, es un candidato a participar en un proceso de integración, acarreando esfuerzos y costos a futuro para cualquier organización.

Un servicio debe tener una alta capacidad de inter-operar, disminuyendo la posibilidad de requerir futuros desarrollos de integración para generar interoperabilidad.

5.2.2.2. Incrementar la Federación

Cada una de las soluciones lógicas o aplicaciones que soportan los procesos de negocios de una organización deben concebirse, ejecutarse y evolucionar bajo las directrices de los departamentos de TI de las organizaciones. Por lo tanto un ambiente de TI federado, es aquel en donde los recursos y aplicaciones están unificados de tal forma que mantienen su propia autonomía y gobierno. Sin importar en qué dirección las aplicaciones dentro de las organizaciones crezcan, la federación debe prevalecer de igual manera.

El desarrollo de servicios estandarizados y con capacidad de composición, garantizan un alto grado de federación; estos encapsulan segmentos de la empresa y son expresados de forma consistente, siendo la estandarización un factor clave, fomentando un ambiente de operación donde las soluciones lógicas de la empresa, se encuentren armonizadas independientemente de la naturaleza de su implementación.

5.2.2.3. Diversidad de Tecnologías

El objetivo de la computación no es promover intrínsecamente la diversidad de tecnologías, ya que esto implica expertos en cada una de las áreas, costos en licenciamiento, entre otros. Pero si promueve la capacidad de diversificarse cuando sea necesario.

Se persigue un ambiente en donde las soluciones lógicas, no distingan entre las tecnologías que otros servicios utilizan para soportar la lógica que encapsulan, permitiendo así operar en una arquitectura tecnológica independiente de plataforma, ofreciendo a las organizaciones capacidad de cambiar, extenderse e incluso remplazar soluciones implementadas o recursos tecnológicos sin interrumpir la federación.

5.2.2.4. Mejorar la alineación entre el negocio y la tecnología

Las soluciones lógicas albergadas en las organizaciones son una respuesta directa a las necesidades del negocio, es decir el apalancamiento de sus procesos de negocio a través del uso de las tecnologías de la información. Bajo los paradigmas para el desarrollo de software tradicional como la programación estructurada u orientada a objetos, las aplicaciones son diseñadas para satisfacer requerimientos inmediatos o tácticos; la transición en entre la adquisición de los requerimientos, el análisis y el diseño, implica una traducción entre el experto en el negocio y el experto en la tecnología para la implementación.

El grado en que los requerimientos de negocio de TI son alcanzados, esta muchas veces asociado a la precisión con la que la lógica de negocio sea expresada y automatizada a través de soluciones lógicas.

Se persigue una disminución de la brecha existente entre el experto del negocio y el experto en la tecnología, a través de la construcción de soluciones lógicas

mucho más agnósticas, pero que expresen claramente las necesidades del negocio.

5.2.2.5. Incrementar el Retorno de la inversión

Debido a que la naturaleza de la lógica requerida en las aplicaciones se ha incrementado en complejidad, además de la creciente integración de arquitecturas no federadas que son difíciles de mantener y evolucionar, los departamentos de TI en promedio representan una cantidad significante en el total del presupuesto operacional para una organización.

De tal forma que el retorno de la inversión de las soluciones automatizadas es un factor clave para determinar qué tan rentable es una aplicación o un sistema, siempre en busca de aumentar el uso que se le pueda dar a este. Es por esto que la COS promueve la creación de soluciones lógicas transversales y agnósticas, que incrementen su potencial de reusabilidad, a través del ensamble repetitivo en diferentes composiciones. Cualquier servicio agnóstico puede por lo tanto encontrarse a sí mismo siendo utilizado numeras ocasiones para automatizar diferentes procesos de negocios como parte de diferentes soluciones orientadas a servicios.

5.2.2.6. Agilidad Organizacional

Un departamento de TI algunas veces puede ser percibido como un cuello de botella, dificultando los tiempos de respuestas deseados a la víspera de cambios. En ocasiones le toma mucho tiempo o recursos para completar requerimientos de negocios nuevos o modificaciones. Es por esto que la eficiencia con la que una organización puede responder a los cambios es un valor de alto grado.

La COS a través del uso de servicios altamente estandarizados y agnósticos a procesos de negocios padres y a ambientes de aplicación específicos; permite la

reducción del tiempo y el esfuerzo requerido para la creación y/o modificación nuevos procesos de negocios automatizados.

5.2.3. Principios de diseño del desarrollo de software orientado a servicio

5.2.3.1. Estandarización de contratos

Un contrato es la información publicada por el propietario de un servicio, en la que se exponen las condiciones técnicas y generales para el uso del mismo. Su diseño está directamente relacionado con la tecnología que se usará para su implementación, políticas de uso y el modelo de datos involucrado en su invocación y/o respuesta.

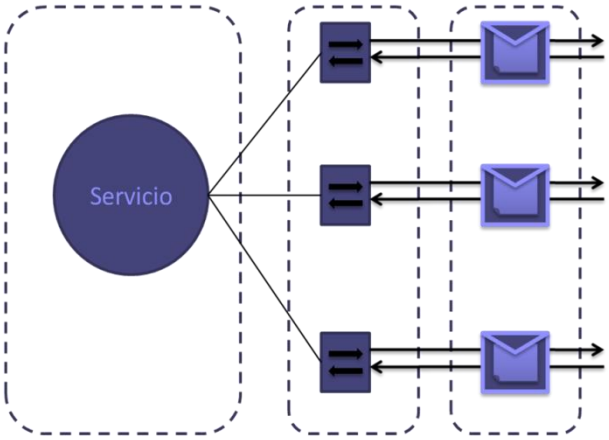

Figura 8 - Servicios con operaciones orientadas a mensajes

Los contratos aparecen desde hace mucho en diferentes tecnologías tales como CORBA con su IDL(*Interface Description Language*) o la utilización de proxies (contratos) para la comunicación de objetos distribuidos basados en RPC (Remote Procedure Call). Para el caso de los servicios web existe WSDL (Web Service Description Language) el cual es un dialecto XML (*eXtensible Markup Language*) estandarizado y auspiciado por la W3C para la descripción de Servicios Web.

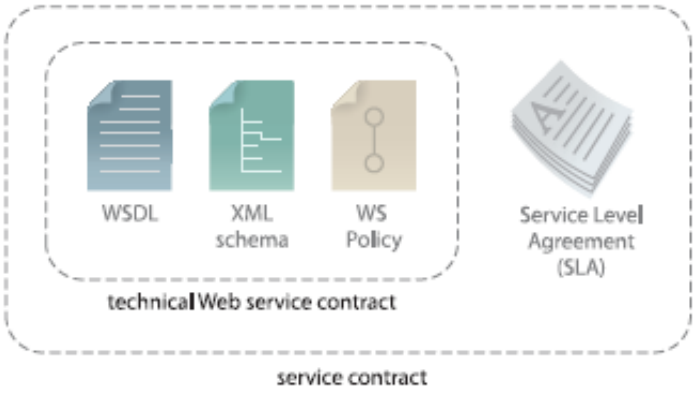

Figura 9 - Composición de un Servicio de Contrato

El contrato técnico de un servicio describe las condiciones para la invocación del servicio. Siempre que dos programas o soluciones lógicas deseen comunicarse, una cierta de forma de contrato técnico es necesario; donde los datos necesarios para la invocación e intercambio de información es predefinido y basado en especificaciones técnicas y formales.

La estandarización de contratos se reduce la necesidad de transformar de datos entre diferentes modelos, a través del uso de modelos de datos consistentes para el intercambio de información descrita en los contratos técnicos. Además permite que las capacidades y propósito de los servicios sean más fáciles de entender.

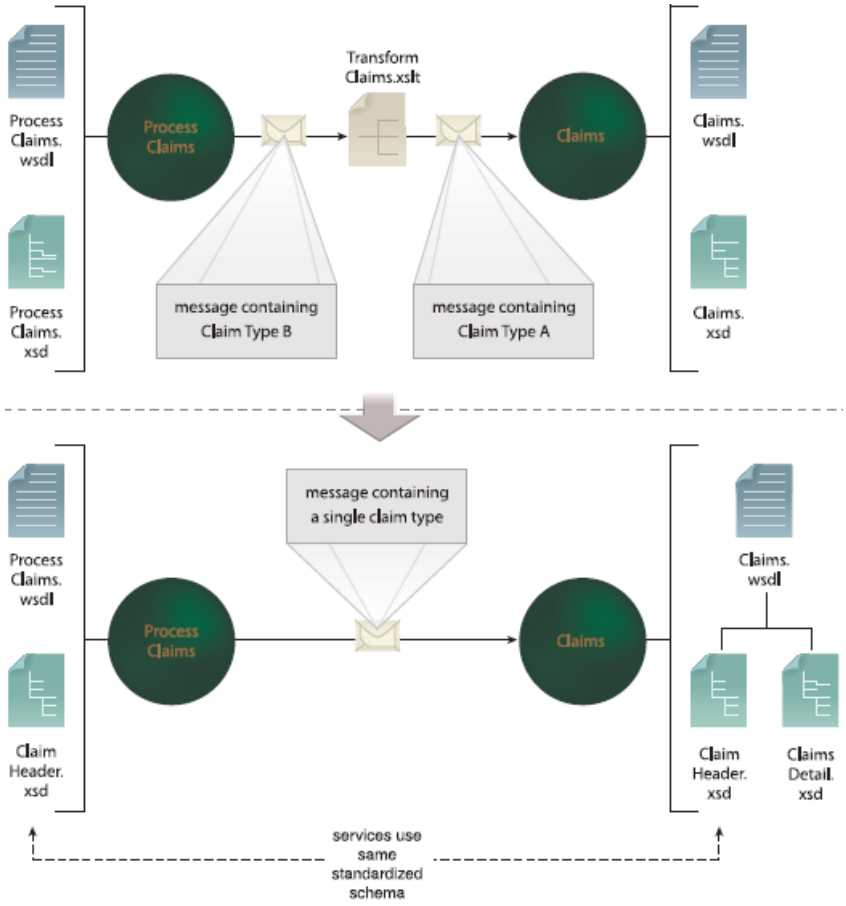

Figura 10 - Intercambio de mensajes con estructuras de datos homogenizadas

El contrato de un servicio debe estar comprendido de una interface técnica o, una o más documentos de descripción del servicio, el cual debe ser provisto con el servicio.

Los estándares de diseño y las convenciones necesitan ser impuestas a priori de cualquier despliegue de un servicio, en función de garantizar el alcance de una adecuada estandarización.

Procesos formales necesitan ser introducidos para garantizar que los servicios sean modelados y diseñados consistentemente, incorporando principios de diseño, convenciones y estándares aceptados.

Es importante el uso de herramientas que sean capaces de importar contratos de servicios personalizados sin imponerles cambio, lo cual ocurre con algunas herramientas de desarrollo actuales.

5.2.3.2. Bajo acoplamiento

El acoplamiento es un tema gran importancia en el desarrollo de software, pero además es un término que puede llegar a ser confuso o ambiguo, en el contexto del software hace referencia al nivel de conexión o relación entre dos o más, componentes o programas.

El bajo acoplamiento promueve la reducción de dependencia que se crea al momento de asociar un servicio proveedor con sus consumidores, o las asociaciones internas de un servicio con relación a su funcionalidad, tecnología u otros componentes o servicios.

Cualquier parte de un ambiente automatizado que sea separable tiene el potencial y usualmente la necesidad de estar acoplada con alguna otra por el bien de impartir su valor. Esto implica que el acoplamiento es inevitable, pero al momento de tomar decisiones de diseño lo importante es determinar y/o comprender que tan cerrada son o deben ser estas relaciones; relaciones que pueden ser unidireccionales o bidireccionales.

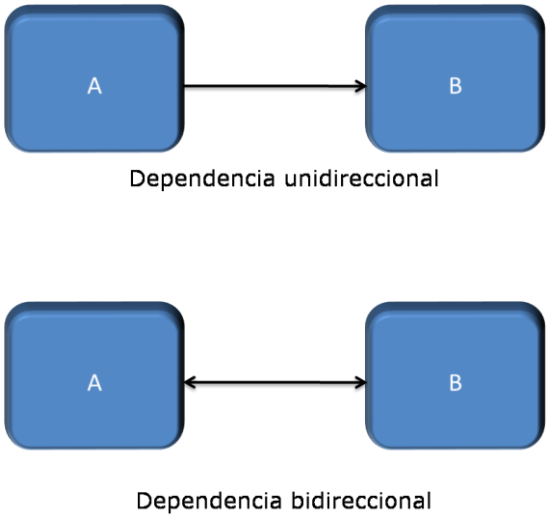

Figura 11 - Relaciones de dependencia entre artefactos de software

Los servicios como unidades lógicas, pueden presentar acoplamiento por su naturalidad de dos formas, acoplamiento concerniente a su arquitectura interna y/o en sus relaciones con otros servicios.

Es muy importante conocer los diferentes tipos de acoplamiento que puede sufrir un servicio, pues estas determinarán la forma en que este podrá evolucionar en el tiempo y responder a los cambios.

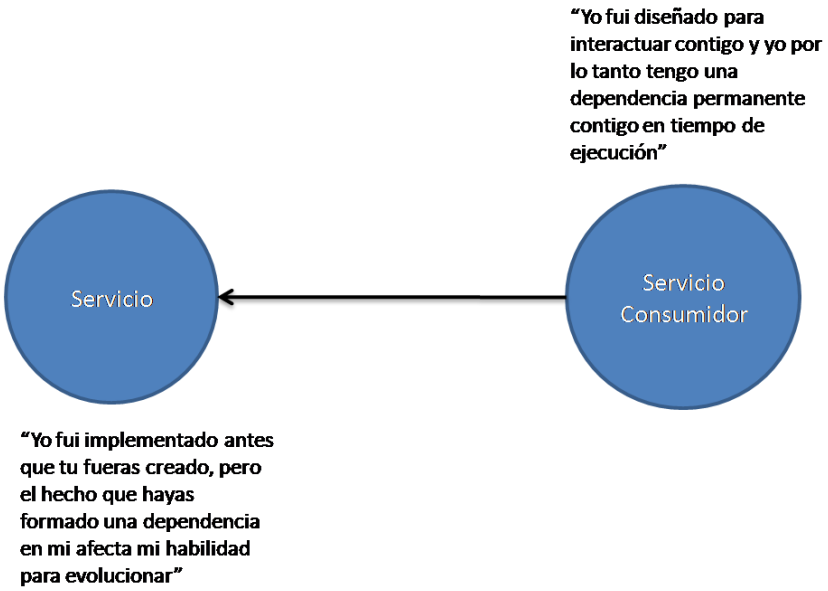

Figura 12 - Dependencia en tiempo de ejecución entre servicios

El bajo acoplamiento fomenta un ambiente en el cual los servicios y sus consumidores pueden evolucionar con el mínimo impacto. En el cual los servicios están idealmente desacoplados de los detalles tecnológicos y de implementación, en un contexto funcional del servicio que no depende lógica externa y con mínimos requerimientos de acoplamiento para el consumidor.

Para alcanzar el balance adecuado de acoplamiento es necesario un excelente diseño de los contratos. Lo cual típicamente requiere de más procesamiento en tiempo de ejecución que los altamente acoplados.

Acoplamiento interno de un servicio

La arquitectura interna de un servicio contiene diferentes connotaciones a nivel tecnológico referente a su plataforma de ejecución, sus gestores de bases de datos, la relación entre sus componente que soportan toda la lógica que estos encapsulan, entre otros.

Los contratos pueden estar acoplados de las siguientes formas:

- **Acoplamiento de la lógica al contrato:** este es un tipo de acoplamiento positivo que es resultado del enfoque de construir el contrato inicialmente y luego la lógica para soportar al servicio. Es un acoplamiento unidireccional que facilita la construcción de contratos estandarizados y permite incluso afinar el rendimiento y confiabilidad en tiempo de ejecución de la lógica subyacente; pero con limitaciones en ambientes en los que se necesite auto-generar los contratos o donde estos sean provistos por adaptadores para la extensión de servicios (Service Wrappers). Este tipo de acoplamiento permite que la lógica subyacente pueda ser remplazada en

un futuro sin afectar a los consumidores que han formado dependencias en el contrato.

- **Acoplamiento del contrato con la lógica:** Este tipo de acoplamiento se presenta cunado la funcionalidad expuesta por el contrato depende fuertemente de cómo esta implementado el servicio, un caso en que frecuente este tipo de acoplamiento resulta, se da al utilizar herramientas para la auto-generación de contratos, como por ejemplo los WSDs creados por las IDE's en el caso de los Servicios Web, para encapsular lógicas existente. Cada vez que la lógica cambie y un nuevo contrato se genere, una nueva versión del contrato debe ser publicada, lo que genera diferentes problemas con los consumidores.

- **Acoplamiento del contrato con la tecnología:** Un servicio desarrollado con un componente propietario puede requerir que el contrato del servicio exista como una extensión propietaria del servicio. Esto acopla el contrato a la tecnología de implementación lo que impone el requerimiento que todos los servicios consumidores soporten la misma tecnología de implementación y/o el mismo protocolo de comunicación propietario.

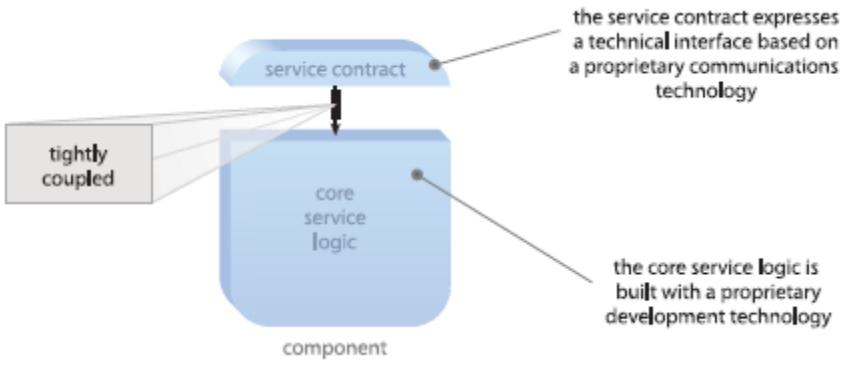

Figura 13 - Acoplamiento del contrato con la tecnología

- **Acoplamiento del contrato con la implementación:** Detrás de la lógica subyacente encapsulada en un servicio existe un conjunto de elementos que soportan su ejecución. Estos elementos pueden ser bases de datos con sus respectivos modelos de datos, APIs legacy, cuentas y grupos de usuarios, asociaciones a estructuras físicas de directorios, ambientes físicos de servidores y sus dominios asociados, nombres de archivos, direcciones de red, y muchos más elementos. El grado con el que un servicio se base en recursos externos determina el nivel de la forma de acoplamiento con su ambiente de implementación circundante.

- **Acoplamiento del contrato con la funcionalidad:** este tipo de acoplamiento se presenta cuando el servicio está encapsulando funcionalidad que se encuentra fuera de sus fronteras.
    - Acoplamiento al proceso padre, cuando un servicio ha sido diseñado específicamente para soportar un proceso de negocio particular, por lo tanto su lógica y por lo tanto su contrato deben estar bien acoplados con la lógica del proceso.
    - Acoplamiento con el consumidor, en un ambiente colaborativo o una arquitectura B2B los consumidores pueden dictar unos estándares de diseño que correspondan a su organización o ambiente, por lo tanto el diseño del servicio debe estar acoplado a estos requerimientos.

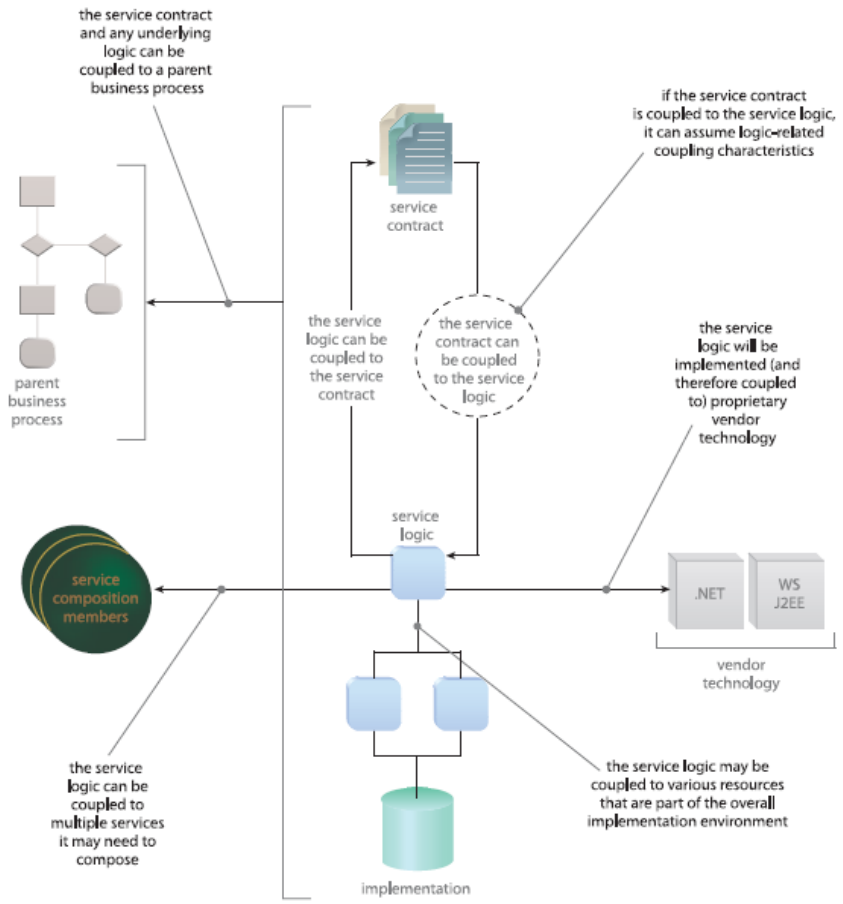

Figura 14 - Acoplamiento a nivel interno de la arquitectura de un servicio y su relación con su contrato -Tomado de [7]

Acoplamiento entre el servicio y sus consumidores

El acoplamiento que se presente entre un servicio y sus consumidores afectará directamente su capacidad de evolucionar, cualquier modificación en sus elementos impactará a sus consumidores.

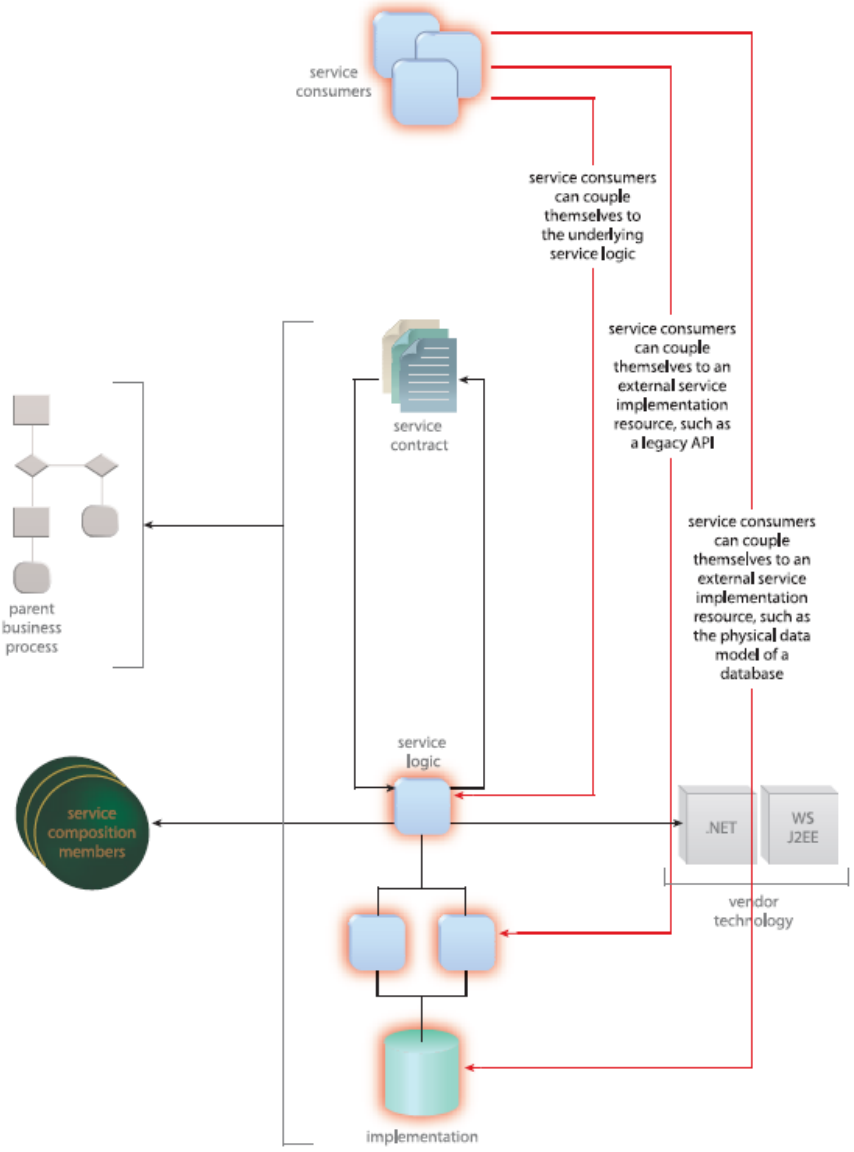

Figura 15 - Acoplamiento entre el servicio y sus consumidores

Es posible identificar dos tipos de acoplamiento entre los servicios y sus consumidores:

- **Acoplamiento del consumidor con la lógica del servicio:** este tipo de acoplamiento se presenta cuando los consumidores acceden no a la lógica que encapsula el servicio, sino a los recursos que este contiene. Esto es muy común en los tipos de integración, pero genera un alto acoplamiento que se debe evitar. Una posible estrategia para abolir este tipo de

acoplamiento es el uso de contratos estandarizados y la centralización de contratos, que obliguen a formar un acoplamiento con el contrato.

- **Acoplamiento del consumidor con el contrato del servicio:** este tipo de acoplamiento es deseable, ofreciendo una mayor independencia entre el servicio y sus consumidores. Es vital que el acoplamiento interno del servicio este en forma, pues este acoplamiento entre el contrato y los demás elementos será inherente al consumidor cuando este se acople al contrato.

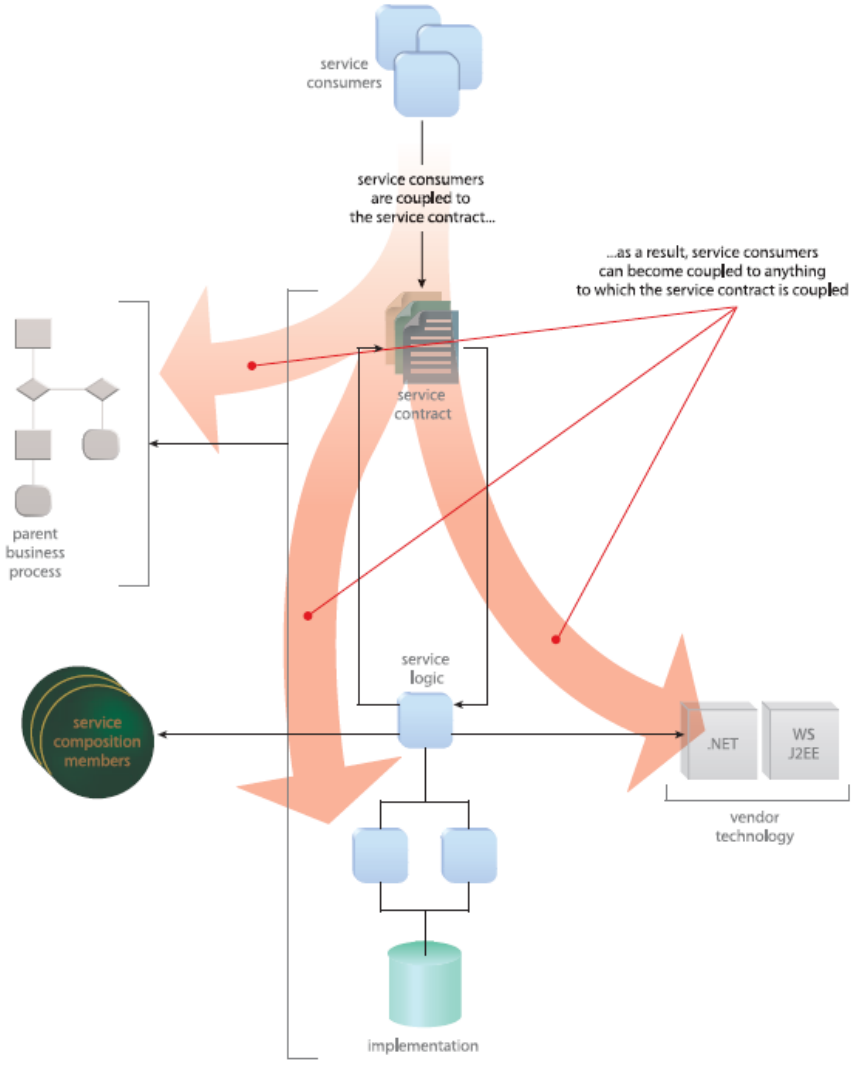

Figura 16 - Acoplamiento del consumidor con el contrato del servicio

5.2.3.3. Abstracción

La abstracción determina el nivel de detalle de la información funcional, tecnológica, programática o descriptiva que se publica sobre una solución lógica.

Este principio de diseño promueve ocultar información a los consumidores, donde los contratos de los servicios solo contienen información esencial y la información acerca de los servicios está limitada a lo que sea publicado en el contrato. Lo que se publique acerca de un servicio comunica sus propósitos y capacidades, y ofrece detalles a sus potenciales consumidores sobre como invocar y engranar programáticamente el servicio.

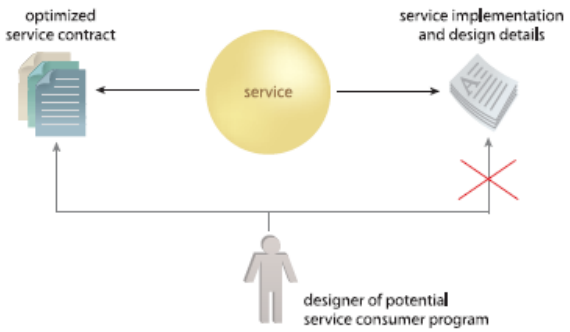

Figura 17 - Abstracción de la información publicada de un servicio - Tomada de [7]

La información que no sea publicada acerca de un servicio protege la integridad del acoplamiento formado entre este y sus futuros consumidores. Ocultando detalles específicos, permite que la lógica del servicio y su implementación pueda evolucionar en el tiempo y seguir cumpliendo con las obligaciones adquiridas en relación a lo que fue publicado originalmente en el contrato.

Los contratos de los servicios definen consistentemente los requerimientos y restricciones de interacción y otra meta información requerida del servicio.

Fuera de lo que es documentado en el contrato de un servicio, la información acerca del servicio es controlada o del mismo modo ocultada dentro de un ambiente particular.

El principal prerrequisito para alcanzar un apropiado nivel de abstracción para cada servicio es la aplicación apropiada de abstracción en el contrato del servicio.

Abstracción Vs. Encapsulación

La encapsulación se refiere a la lógica, recursos y la información contenida dentro de los límites del servicio. Un Servicio en su totalidad está comprendido por un contrato y lo que este encapsula.

La abstracción hace referencia a las partes encapsuladas por el servicio y, que son expuestas y ocultadas para los programas consumidores que se encuentran fuera de los límites del servicio. Entre menos información sea compartida acerca de lo que el servicio encapsula, más cambios se podrán aplicar a lo que esta encapsulado sin afectar a los programas consumidores que ya se encuentren usando el servicio.

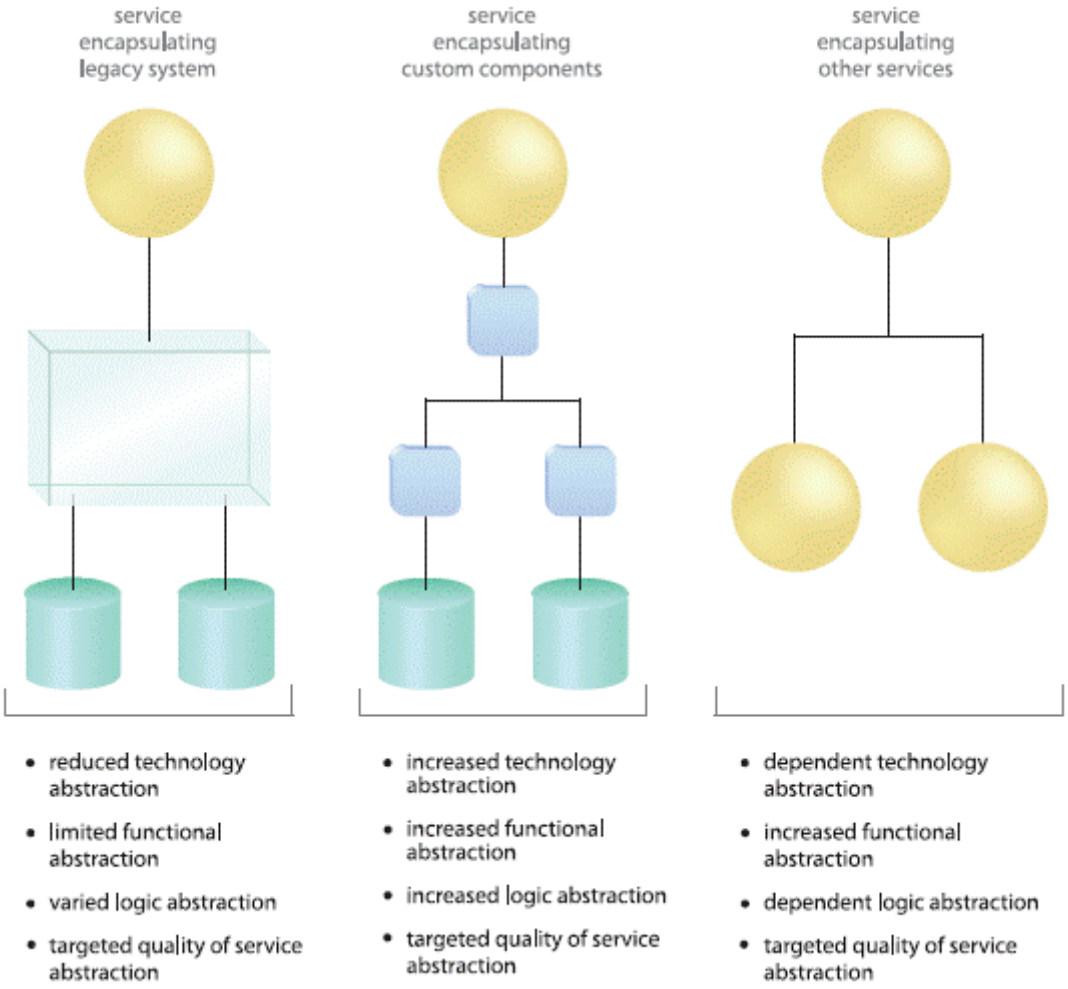

Figura 18 - Consecuencias de la encapsulación sobre la abstracción [7]

Tipos de meta-información

Un servicio abarca información tecnológica, funcional, programática y de calidad de servicio principalmente. Esta información compromete la capacidad de evolución de un servicio y por lo tanto debe ser abstraída según sea necesario.

- **Información tecnológica:** describe los detalles técnicos de la implementación de la lógica subyacente de un servicio.

- **Información funcional:** describe de que es capaz un servicio. Es necesario determinar cuáles de las capacidades de un servicio deben ser publicadas y cuáles no, en el contrato técnico de un servicio.

- **Información programática:** describe como un servicio lleva a cabo su funcionalidad. Típicamente representa detalles de diseño de bajo nivel como algoritmos, manejo de excepciones, bitácoras de errores y otra información asociada a como la lógica subyacente de un servicio es construida.

- **Información de calidad de servicio:** describe el comportamiento, limitaciones y requerimientos para la interacción de un servicio. Existe información sobre la calidad del servicio que los diseñadores de los servicios consumidores necesitan conocer y entender para saber que esperar del servicio, como también otra que no debe ser de su conocimiento tal como las acciones tomadas para administrar la calidad del servicio.

5.2.3.4. Reusabilidad

Este principio persigue el posicionamiento de los servicios como recursos empresariales reusables, a través del encapsulamiento de lógica agnóstica y multipropósito.

La reusabilidad no es siempre una característica de diseño deseable, esto depende como siempre del fin último del software que se esté desarrollando. Para una aplicación multipropósito con finalidad comercial que busca satisfacer diferentes necesidades la reusabilidad si es una característica deseable, mientras que para una solución interna muy puntual para una organización esto acarrearía desafíos innecesarios. La reusabilidad incrementa la complejidad, costos, esfuerzos y tiempo para construir software.

La reusabilidad no es una característica nueva para el desarrollo de software, un ejemplo de esto es como el desarrollo de software orientada a objetos a través de la distribución de componentes, ha alcanzado cierto nivel de éxito en el reúso. La problemática de la reusabilidad en la POO se debe a algunos de las siguientes condiciones:

- La reusabilidad de un componente está limitada por ambientes de ejecución propietarios y/o programas consumidores propietarios.

- La reusabilidad es alcanzada principalmente a través de estructuras de herencia y otro tipo de enfoques de diseño altamente acoplados, creando como resultado una alta dependencia en otros componentes.

La reusabilidad permite que la lógica de los servicios pueda ser utilizada repetidamente en el tiempo para alcanzar un alto retorno de inversión, incrementando la agilidad de negocio a nivel organizacional, gracias al cumplimiento de futuros requerimientos de automatización en el negocio usando la composición de servicios a gran escala. Permitiendo la creación de inventarios de servicios con un alto porcentaje de servicios agnósticos.

El diseño de los servicios debe ser definido dentro de un contexto funcional agnóstico, que permita su uso en cualquier tipo de escenario de uso, para incrementar sus posibilidades de reusabilidad.

Una lógica altamente genérica, permite que pueda ser usado por diferentes tipos de servicios consumidores en diferentes escenarios de uso, además de contar con un contrato genérico y extensible, que sea lo suficientemente flexible para procesar un rango de mensajes de entrada y salida; garantizando que este pueda ser accedido de forma concurrente, para facilitar el acceso simultaneo por múltiples programas consumidores.

Es necesario considerar, que los servicios reusables deben contemplar los siguientes ambientes:

- Un ambiente de ejecución escalable, capaz de ofrecer al servicio una concurrencia alta a extrema.
- Un sistema de versión sólido para manejar de forma apropiada la evolución de los contratos de los servicios.
- Analistas y diseñadores de servicios con alto grado de experiencia en la problemática, que puedan garantizar que los limites del servicio y su contrato este representado de forma precisa el contexto funcional del servicio.
- Experiencia en el desarrollo de software comercial y servicios de alto nivel, con el fin de estructurar la lógica subyacente en rutinas y funcionalidad genérica.

5.2.3.5. Autonomía

Entre más independiente es un sistema de influencias externas impredecibles, mayor es su confiabilidad. La previsibilidad y la confiabilidad son los dos factores principales que hacen a la autonomía una característica de diseño a considerar.

El nivel de autonomía es representada por el grado en el que un sistema pueda actuar independientemente. Por lo tanto si un programa existe en un estado de ejecución autónomo, este será capaz de llevar a cabo su lógica independiente de influencias externas, como lo son recursos, datos o funcionalidad que están fueran del domino del programa. A su vez este tendrá la capacidad de gobernarse a sí mismo en tiempo de ejecución, por lo tanto entre más control tenga sobre su ambiente de ejecución, más autonomía podrá reclamar.

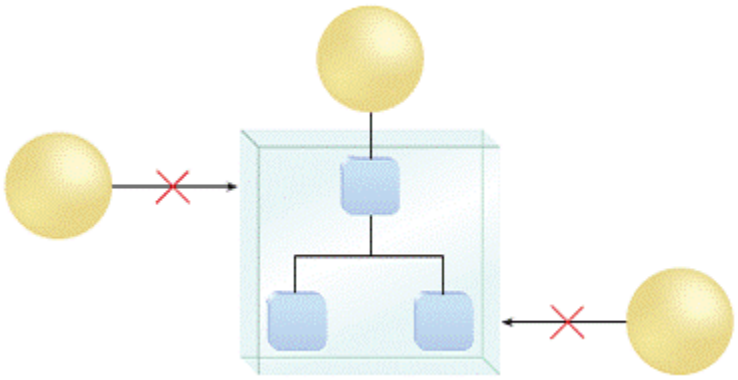

Figura 19 - Autonomía sobre los recursos encapsulados [7]

La autonomía ha sido tradicionalmente una cualidad más asociada con plataformas o ambientes de ejecución como un todo, en lugar que con componentes individuales de soluciones lógicas. Como consecuencia esta muchas veces es solamente perseguida a través del despliegue de aplicaciones en servidores dedicados, por lo que es importante entender que esta es una característica de diseño de alto impacto en las soluciones lógicas.

Existen estrategias de diseño que promueven la autonomía, tales como:

- Definir los contratos para que expresen de forma bien definida sus límites funcionales, los cuales no deben ser solapados por otros servicios.
- Desplegar los servicios en ambientes sobre los cuales estos ejerzan un gran control.
- Albergar las instancias de servicios en ambientes que se acomodan a altos niveles de concurrencia para propósitos de escalabilidad.

Es importante considerar los siguientes requerimientos a la hora de implementar altos grados de autonomía en los servicios:

- Un alto nivel de control sobre como la lógica del servicio es diseñada y desarrollada. Dependiendo del nivel de autonomía que se busque, también se debe involucrar control sobre el modelo de datos soportado.

- Un ambiente de despliegue distribuido, para que el servicio pudiera ser movido, aislado o compuesto cuando sea requerido.
- Una infraestructura capaz de soportar los niveles de autonomía deseados.

Tipos de autonomía en los servicios

Autonomía en tiempo de ejecución

El grado de control que un servicio tenga sobre su lógica de procesamiento en el momento en el que el servicio es invocado y ejecutado es llamado autonomía en tiempo de ejecución. Su principal objetivo es garantizar niveles aceptables de rendimiento en tiempo de ejecución, además de un grado de confiabilidad mínimo sobre el rendimiento. Proporcionar además la opción de poder ser aislado en respuesta a requerimientos específicos de seguridad, confiablidad o rendimiento.

Cada servicio que encapsule y este compuesto por lógica de otros servicios, forma una dependencia en la lógica externa y por lo tanto fuera de su control. En consecuencia la autonomía de un servicio que encapsula una composición de servicios es determinada por la autonomía colectiva de cada uno de los servicios pertenecientes a las composiciones, sobre todo cuando hay composiciones presentes en grandes inventarios de servicios.

Unos de los aspectos por los que la autonomía en tiempo de ejecución es importante son:

- La habilidad para escalar un servicio en respuesta a altos niveles de demanda en la usabilidad.
- La opción para mejorar el ambiente de hosting de un servicio.
- La libertad para aumentar, actualizar, o remplazar la tecnología de un servicio, en respuesta a nuevos requerimientos o al deseo de liberar nuevas innovaciones.

Los cuales pueden ser alcanzados a través de la aplicación del principio del bajo acoplamiento de los servicios, persiguiendo un acoplamiento positivo en el contrato.

Autonomía en tiempo de diseño

El grado de libertad que un servicio tiene para cambiar sobre su ciclo de vida, representa su autonomía en tiempo de diseño. Una vez que los programas consumidores se enlazan programáticamente al contrato de un servicio, este no podrá escapar sus obligaciones con el contrato.

5.2.3.6. Bajo manejo de estados

En pro de garantizar alta reusabilidad y alta concurrencia en servicios de alta usabilidad, es necesario garantizar un buen uso de los recursos. El bajo manejo de estados se logra, diseñando estrategias para delegar y diferir el manejo de los mismos. Entiéndase por datos de estado, información principalmente asociada con una actividad en el contexto de ejecución de un servicio, y el manejo de estados el procesamiento de esta información.

Tecnologías anteriores han movido la responsabilidad el manejo de estados a nivel de clientes y servidores, buscando aliviar temporalmente esta carga y permitir a las aplicaciones mejorar su escalabilidad en forma general.

Una buena estrategia para el manejo de estados en los servicios, incrementa las posibilidades de estos para escalar y de ser reusados. Dependiendo de la estrategia usada y de los servicios, diferentes características de diseño pueden ser implementadas. Por ejemplo, servicios con lógica de procesos de negocio altamente agnóstica no deberían ser diseñados para retener información de estados de ningún proceso de negocio padre; tener bajas restricciones en los contratos para permitir la recepción y transmisión de un alto rango de datos de estado en tiempo de ejecución; así como incrementar las rutinas para la

conversión y serialización de información de estados en el intercambio de mensajes.

Con el fin de llevar a cabo este principio se considere los siguientes requerimientos de implementación:

- Un entorno de ejecución que permita realizar de forma eficiente a los servicios la transición de un estado activo a inactivo.
- Parsers XML de alto rendimiento o de nivel empresarial, además del uso de apoyo en hardware para permitir a los servicios implementados como servicios web procesar altas cargas de mensajes largos con menos restricciones de procesamiento.
- Servicios web capaces de adjuntar información que no requiera ser validada en la capa de interfaces ni convertida a formatos locales, para facilitar una mejor transmisión de datos de estado.

Tipos de Estados

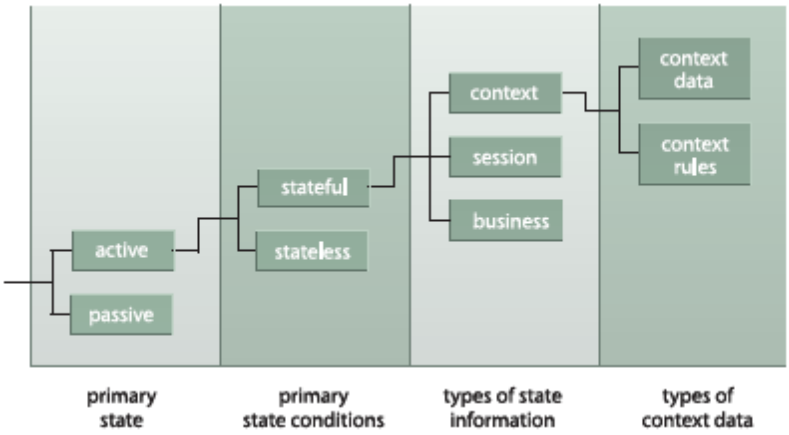

Figura 20 - Tipos de estados

Un estado activo representa un servicio siendo invocado o ejecutado. Un estado pasivo, representa el periodo durante el cual el servicio no es usado. Un servicio activo no comprometido con el procesamiento de datos de estado, es considerado un servicio sin manejo de estados (stateless); por el contrario uno que si está

comprometido con el procesamiento de estados es considerado un servicio con manejo de estados (stateful). Las diferentes categorías de información que representan estados son los datos de sesión, negocio y de contexto. Los datos de sesión típicamente representan información asociada con la retención de una conexión hecha entre un programa y su cliente. Los datos de negocio son datos relevantes a la tarea de negocio en ejecución, típicamente datos transitorios entre la aplicación y los repositorios de datos. Los datos de contexto hacen alusión a las reglas y datos que hacen parte de la lógica de ejecución de un proceso.

Es importante tener cuidado en la administración de flujos de trabajos conformados por múltiples servicios y en las estrategias utilizadas para delegar y diferir a repositorios de datos este manejo de estados de datos de contexto.

5.2.3.7. Capacidad de descubrimiento

Desde una perspectiva arquitectural, la fácil localización de las soluciones lógicas individuales es una característica deseable. El proceso de búsqueda y localización de dichas soluciones lógicas dentro de un ambiente específico se conoce como capacidad de descubrimiento. Esta fomenta el uso de adecuado de soluciones lógicas existentes, posibilitando la correcta demarcación de los esfuerzos necesario para construir, usar y evolucionar las mismas.

La implementación de este principio se ha venido incorporando de manera informal dentro las organizaciones, los proyectos para la entrega de objetos compartidos o componentes han centrado sus mecanismos de descubrimiento en el uso de medios centralizados que comunican la disponibilidad de dichos recursos a otras partes de las organizaciones, tales como hojas de Excel, páginas Web, entre otros. Un esfuerzo para estandarizar estos mecanismos es UDDI, siendo parte de la primera generación de Servicios Web, sin embargo este no ha sido ampliamente adoptado.

Se persigue el diseño de servicios posicionados como recursos con alta capacidad de descubrimiento dentro de las organizaciones. Con propósitos y capacidades claramente expresadas, de tal forma que sean fácilmente interpretadas por humanos y por programas de software. Para alcanzarlo los servicios deben estar equipados con contratos con meta información correctamente referenciada en los resultados de búsquedas de descubrimiento y que contengan información clara de sus funcionalidades, a su vez interpretables por humanos.

Para la implementación de este principio es necesario:

- La creación de estándares que gobiernen la información usada la creación de contratos de servicios, haciendo de estos los más fáciles de descubrir e interpretables posibles; así como guías para cómo y cuándo realizar anotaciones suplementarias a dichos contratos.
- La constitución de estándares que establezcan el significado de la meta información que esta por fuera de los contratos, lo cual debe estar contenida en un documento suplementario o en un registro de repositorio de servicio, si existiese.

Meta información de descubrimiento

La meta información relevante para el descubrimiento es la funcional y de calidad de servicio. Se detalla así:

- Funcional, describe las capacidades y propósitos de los servicios. A través de la estandarización de contratos se garantiza que los servicios puedan ser localizados y entendidos.

- Calidad de Servicio, hace referencia a factores de comportamiento como tiempos de espera, políticas de uso, entre otras, asociados al entorno de ejecución de los servicios. Esto mejora el detalle de los criterios de búsqueda que posibilita enlaces en tiempo de ejecución a servicios según las necesidades de QoS de sus potenciales consumidores.

5.3. Arquitectura Orientada a Servicios (SOA)

El termino SOA (*Service Oriented Architecture*), fue acuñado originalmente por Alexander Pasik un ex-analista de Gartner, en el 1994, para describir una clase de middleware el cual estaba enseñando, debido a que el término "cliente - servidor" había perdido su significado clásico, a razón que en gran parte de la industria, era y es usado para referirse a sistemas distribuidos que involucran dos computadoras, donde la primera se refiere a un PC que representa al cliente y se encargaba de ejecutar la lógica de presentación y gran parte de la lógica de negocio, y la segunda hace referencia al servidor, sobre el cual corría el sistema de base de datos, se almacenaban los datos y alguna veces se ejecutaba parte de la lógica del negocio.

Dentro de este tipo de sistemas distribuidos los término "cliente" y "servidor" se refieren únicamente al hardware, por lo cual, el software que corre tanto en el cliente, como en el servidor, ha perdido la relevancia que tenía en el uso clásico del término "cliente - servidor", para evitar ambigüedades, Pasik destacó "la orientación al servicio" e instó a los desarrolladores a diseñar aplicaciones empresariales de tipo SOA.

En [14] y [15] se publica la primera descripción formal del termino SOA, definiéndolo como un estilo arquitectural para aplicaciones de negocios que son modulares, distribuidas, compartibles y débilmente acoplados.

Y es solo hasta el año 2000, cuando SOA adquiere su mayor importancia, cuando aparece la tecnología de los servicios web, y aunque una arquitectura SOA no

implica necesariamente una implementación con servicios web y no toda implementación con servicios web es de tipo SOA, los servicios web permiten una implementación adecuada de este tipo de arquitecturas, de hecho se han convertido en la tecnología dominante para la implementación de aplicaciones SOA.

Lo anterior atrajo la atención de otras compañías como IBM, SAP, SUN, etc., los cuales han invertido muchos recursos para el desarrollo de esta tecnología y con ella el concepto organizacional que implica la arquitectura, lo cual ha hecho que muchos consideren actualmente a SOA como el concepto clave para el futuro del software.

5.3.1. Arquitectura

SOA es un estilo arquitectural enmarcado dentro del modelo de construcción de software propuesto por la computación orientada a servicios, el cual utiliza a los servicios como unidad de construcción, cuyo objetivo es definir diferentes formas de interacción y comunicación entre los mismos, preservando los principios y metas propuestas por el paradigma de la COS.

La arquitectura fue creada para dar solución a problemáticas empresariales, la principal de ellas la integración de aplicaciones, razón por la que está fue diseñada, con características que le permitieran ser desplegada fácilmente sobre ambientes heterogéneos e infraestructuras distribuidas como es común en el ámbito empresarial.

Las arquitecturas basadas en SOA, como es de esperar por su fundamento en la Computación Orientada a Servicio, tienden a originar soluciones débilmente acopladas, cuyos sistemas son flexibles y tolerantes a fallas, debido a la reducción de dependencias que está trae consigo, aportando a la escalabilidad al evitar la generación de cuellos de botella y promoviendo el uso de formas lo más descentralizadas posible para la implementación, aunque desafortunadamente

SOA requiere un mínimo de centralización para formar las bases comunes de la arquitectura.

5.3.1.1. Interacciones Básicas

Una solución de tipo SOA, define tres interacciones básicas que modelan su comportamiento, dichas interacciones reflejan las relaciones existentes entre los componentes definidos en el modelo de la Computación Orientada a Servicios (servicio, consumidor y servidor).

Estas interacciones son descritas a continuación:

- Publicación: Toma efecto cuando la descripción de un servicio se expone de forma que se accesible por sus posibles consumidores permitiéndoles descubrirlo e invocarlo.
- Descubrimiento: Es la acción a través de la que un consumidor servicios localiza un servicio que cumpla con un cierto criterio consultando el registro de servicios.
- Conexión e Invocación: Una vez obtenida la descripción de un servicio por parte de un consumidor, éste lo invoca haciendo uso de la información presente en la descripción del servicio.

5.3.1.1.1. Composición

Una vez definido dentro de un sistema las formas básicas de comunicación entre componentes, se pueden definir procesos más sofisticados de interacción y colaboración, a esta nueva capa de abstracción, pertenece el proceso conocido como composición, que consiste en la determinación de la secuencia de operaciones que debe acontecer en la interacción entre clientes y servidores. La secuencia debe respetar el orden establecido para que pueda ser válida y, con el objeto de que esto resulte viable, se define un protocolo de coordinación que será, precisamente, el encargado de describir el conjunto de secuencias válidas.

Existen dos modelos de composición o colaboración entre servicios:

El **modelo de orquestación** se fundamenta en la existencia de un mecanismo de control centralizado que es el encargado de dirigir las actividades, siendo cada una de ellas una interacción entre servicios. Permite la definición de un modelo de interacción pero sólo desde el punto de vista del controlador. La orquestación define el comportamiento y la forma de llevarlo a cabo, y todos los eventos se supervisan de forma centralizada.

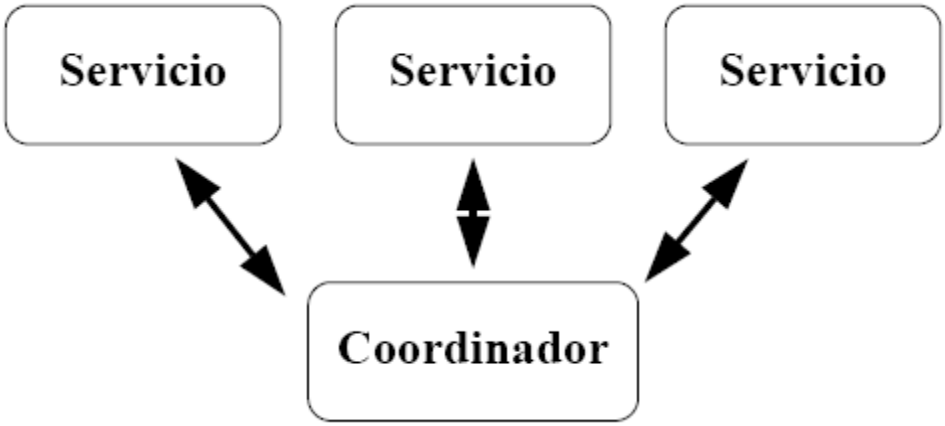

Figura 21 - Modelo de orquestación

El **modelo de coreografía** describe el comportamiento que debe observarse entre las partes que interactúan. Cada una de las organizaciones implicadas en este modelo desarrolla independientemente el papel que desea jugar en la colaboración, la única condición es que respeten el "contrato global" que supone la coreografía descrita. La ejecución y el control es responsabilidad de los propios participantes.

5.3.2. Servicios Web

Un servicio web es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre

cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet.

Desde el punto de vista de los negocios, los servicios web permiten que las organizaciones integren sus diferentes aplicaciones de una manera eficiente, sin preocuparse por cómo fueron construidas, dónde residen, sobre qué sistema operativo se ejecutan o cómo acceder a ellas. Precisamente por esta razón es que los servicios web se consideran integradores, porque permiten crear una interfaz de acceso a las aplicaciones, sin importar las características de implementación de éstas [16].

Los servicios web se caracterizan por: estar basados en XML, poseer una comunicación basada en mensajes, ser independientes del lenguaje de programación, ser dinámicamente localizables, ensamblables y agregables, accesibles por Internet, débilmente acoplados y basados en estándares industriales, lo que les permite utilizarse para entregar de forma eficiente información como noticias, inventarios y reportes de clima a los sitios web o ser usados para funcionalidades complejas que pueden involucrar transacciones más elaboradas entre varias partes, involucrando socios de negocios o proveedores.

Lo anterior nos permite llegar a una definición más exacta de los servicios web, como la brindada por la W3C, la cual define *a* un servicios web como, una aplicación de software identificada por una URI (Uniform Resource Identifier), cuyas interfaces y vinculaciones son capaces de ser definidas, descritas y descubiertas, como artefactos XML las cuales soportan interacción directa con otras aplicaciones de software a través del intercambio de mensajes basados en XML vía protocolos basados en Internet.

Los servicios web nacen en el año de 1999, en respuesta a la necesidad de estandarizar la comunicación entre distintas plataformas (PC, Mainframe, Mac, etc.) y lenguajes de programación (PHP, C#, Java, etc.), y desde entonces han

aparecido diversos factores que han impulsado la tecnología, los cuales se definen a continuación [17]:

- El contenido se está volviendo más dinámico, Los sitios web actuales proporcionan contenido “instantáneo”. Un servicio web debe ser capaz de combinar contenido proveniente de fuentes muy diferentes.

- El ancho de banda es menos costoso, Actualmente un servicio web puede entregar tipos variables de contenido, como video o audio. A medida que crezca el ancho de banda, los servicios web deben adaptarse a nuevos tipos de contenido.

- El almacenamiento es más barato, Un servicio web debe ser capaz de manejar cantidades masivas de datos, y debe poder hacerlo de forma inteligente.

- La computación extendida se está volviendo más importante: Con cientos de millones de dispositivos como teléfonos móviles, beepers, y agendas computarizadas existentes actualmente, estamos llegando a un momento en el cual el PC está dejando de ser el dispositivo más común en internet. A medida que las plataformas se hacen más diversas, tecnologías como XML se volverán más importantes. Un servicio web no puede exigir que los usuarios ejecuten, por ejemplo, un navegador web tradicional en alguna versión de Windows; por el contrario, los servicios web deben servir a todo tipo de dispositivos, plataformas y navegadores, entregando contenido sobre una amplia variedad de tipos de conexión.

5.3.3. Estándares básicos de los Servicios Web

Los servicios web contemplan una cantidad de estándares desarrollados para diferentes aspectos abarcados por la tecnología, pero los estándares básicos bajo

los cuales se desarrolla la infraestructura propuesta por la tecnología son, XML, SOAP, WSDL y UDDI, los cuales serán descritos a continuación.

5.3.3.1 Extensible Markup Language (XML*)*

Es un formato simple basado en texto para representar información estructurada: documentos, datos, configuración, los libros, las transacciones, facturas, y mucho más, desarrollada por W3C, y cuyo principal objetivo es separar los datos, de la presentación. Permitiendo a los desarrolladores crear sus propios tags o etiquetas, que les posibilita habilitar definiciones, transmisiones, validaciones, e interpretación de los datos entre aplicaciones y entre organizaciones.

La W3C público la primera especificación de XML en el año de 1996, con los siguientes objetivos de diseño:

- XML debe ser utilizable directamente sobre internet.
- XML debe soportar una amplia variedad de aplicaciones.
- XML debe ser compatible con SGML.
- Debe ser fácil escribir programas que procesen documentos XML.
- El número de características opcionales en XML debe ser mantenido en un mínimo, idealmente cero.
- Los documentos XML deben ser legibles por un humano y razonablemente claros.
- El diseño de XML debe ser preparado rápidamente
- El diseño de XML debe ser formal y conciso.
- Los documentos XML deben ser fáciles de crear.

5.3.3.2 Simple Object Access Protocol (SOAP)

Es un protocolo ligero destinado al intercambio de información estructurado en un entorno descentralizado, distribuido. Utiliza tecnologías XML para definir un framework extensible de mensajería proveyendo un constructor de mensajes que pueden ser intercambiados a través de una variedad de protocolos

subyacentes. El framework ha sido diseñado para ser independiente de cualquier modelo particular de programación y otras implementaciones de semánticas.

Los principales objetivos de diseño de SOAP son la simplicidad y extensibilidad, para alcanzar estos objetivo la especificación omite de su arquitectura aquellos aspectos que se encuentran a menudo en sistemas distribuidos. Estas características incluyen, pero no se limitan a fiabilidad, seguridad, correlación, ruteo, y *MEPs (*Patrones de intercambio de mensajes*)*. Si bien se prevé que muchas de estas características se definirán, la especificación proporciona detalles sólo para dos MEPS. Otras características pueden ser agregadas como extensiones de la especificación [20].

SOAP ha recibido un gran apoyo por parte de la industria, siendo el primer protocolo de su tipo que ha sido aceptado prácticamente por todas las grandes compañías de software del mundo. Compañías que en raras ocasiones cooperan entre sí están ofreciendo su apoyo a este protocolo. Algunas de las mayores Compañías que soportan SOAP son Microsoft, IBM, SUN Microsystems, SAP y Arib.

5.3.3.3 WSDL

Web Services Description Language, es un lenguaje especificado en XML que se ocupa de describir los requisitos funcionales necesarios para establecer una comunicación con los servicios Web, para lo cual define un modelo que separa las funcionalidad ofrecidas por un servicio, de los detalles concretos, de *como* y *donde* esta funcionalidad es ofrecida. Las definiciones del Servicio WSDL proveen documentación destinada a los sistemas distribuidos y hace las veces de especificación para automatizar los detalles involucrados en las aplicaciones de comunicación.

5.3.3.4 Universal Description, Discovery and Integration (UDDI)

Es un estándar de facto basado en XML, SOAP y HTTP, que provee un método para la publicación y el descubrimiento de información sobre los Servicio Web. Esta iniciativa parte de la industria, que pretendía crear una plataforma independiente, un marco de trabajo abierto para la descripción de servicios, descubrimiento de organizaciones y la integración de servicios de negocio. UDDI se centra en el proceso del descubrimiento dentro de la arquitectura orientada a servicios.

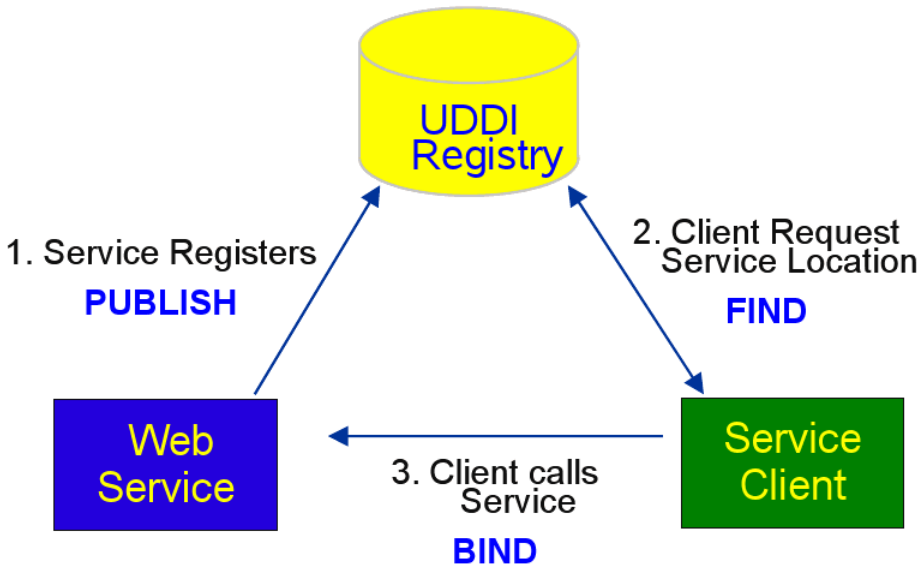

Figura 22 - Arquitectura SOA basada en Servicios Web

UDDI es un único registro conceptual distribuido a lo largo de multitud de nodos que replican la información unos de otros, cuyo objetivo es solucionar la problemática concerniente al descubrimiento y la integración de los servicios web.

UDDI constituye un registro en el que las empresas *suscriben* los servicios web que desarrollan, permitiendo a los posibles usuarios interesados en hacer uso de esos servicios para realizar determinadas acciones en sus negocios, puedan encontrarlos sin demasiada dificultad.

Un registro UDDI permite a las organizaciones ingresar tres tipos de información dentro de él:

- **Páginas blancas,** constituida por información básica de contacto e identificadores de la empresa, incluyendo el nombre del empresa, dirección, información de contacto. Esta información permite a otros descubrir los SW de una empresa basándose en la identificación del negocio.

- **Páginas amarillas,** consiste en información que describe al SW mediante diferentes categorizaciones o taxonomías. Esta información permite descubrir SW basándose en esta categorización.
- **Páginas Verdes,** consiste en información técnica que describe el comportamiento y las funciones soportadas por un SW. Esta información incluye punteros que, entre otras cosas, indican dónde están ubicados los SW.

6. CASO ESTUDIO

La selección del paradigma de desarrollo de software a para llevar a cabo una solución lógica que dé respuesta a los requerimientos para la construcción de un sistema, es una decisión que debe ser evaluada con base a como estos se adaptan y dan respuesta a la visión, contexto operativo y tecnológico del sistema.

En la investigación de [29], se realizó un estudio donde se identificó la metodología AHP para la selección de proveedores de la UTB, permitiéndole evaluar, analizar y seleccionar proveedores previamente a la realización de compras, obteniendo mayor información para una toma de decisiones cada vez formal y objetiva.

El contexto operacional tecnológico de la UTB actualmente alberga la lógica de negocio necesaria para llevar a cabo la funcionalidad mínima para la gestión de los procesos de negocio en el sistema legado SIFAD, que por sus características tiene poca capacidad de escalabilidad, usabilidad e interoperabilidad.

El paradigma del desarrollo de software COS, ofrece una solución acertada para el desarrollo de la solución requerida por [29].

6.1. Alcance

El alcance del caso está delimitado a la aplicación del modelo de desarrollo software COS, al diseño e implementación del sistema de “Gestión y Evaluación de Proveedores SSP”, que responda a los requerimientos descritos anteriormente.

6.2. Metodología

Para el diseño e implementación de la solución lógica del caso de estudio, se utilizó como base la metodología RUP, agregándole artefactos propios del modelo de desarrollo de software COS.

6.3. Desarrollo del sistema

En este apartado se describen las fases de diseño, implementación, pruebas y despliegues desarrolladas en la solución del caso de estudio. La descripción de la problemática y fase de análisis es descrita en el anexo A.

6.3.1. Diseño

Para la utilización de una metodología de desarrollo tipo RUP y el diseño de una solución lógica bajo el modelo de desarrollo COS, es necesario agregar un conjunto de actividades, que garanticen el cumplimiento de los principios de diseños estipulados por el modelo a la solución resultante.

El desarrollo de la solución conceptual estuvo comprendido por las siguientes actividades y artefactos:

- Identificación servicios
- Modelo del dominio
- Diagramas de Clases
- Diseño de contratos
- Diagramas de entidad-relación

De estas actividades y artefactos, la selección de servicios candidatos y el diseño de contratos son una extensión de las actividades y artefactos tradicionales propuestos por la metodología RUP.

6.3.1.1. Identificación de Servicios

En esta actividad se busca identificar funcionalidad contenida en software legado o procesos de negocios, para ser encapsulada en forma de servicio.

Para realizar esta actividad se puede empezar por responder interrogantes como:

- ¿La funcionalidad a encapsular participa o puede llegar a ser parte en la composición de otros procesos de negocio?

- ¿Es la funcionalidad un proceso agnóstico?
- ¿Es la funcionalidad lo suficientemente autónoma que le permita tener el suficiente control sobre sus recursos que permitan su correcto funcionamiento?

Si por lo menos una de estas preguntas tiene una respuesta afirmativa es correcto pensar en convertir la funcionalidad en un servicio, sin olvidar los requerimientos implícitos (seguridad, rendimiento, etc.) en la implementación de servicios.

Normalmente los servicios se categorizan de acuerdo a la zona de negocio en la cual la funcionalidad encaja principalmente, pudiendo hacer parte en mayor o menor medida de una u otra zona.

Los servicios identificados para el caso de estudio fueron los siguientes:

ZONA DE SERVICIOS EN NEGOCIO	SERVICIOS
NEGOCIO	Evaluación de Proveedores Selección de Proveedores
TAREA	Autenticación
INFRAESTRUCTURA	Sincronización de Datos Log
ENTTIDAD	Notificación de Solicitudes a Proveedores

Tabla 3 - Servicios Candidatos

6.3.1.2. Modelo del dominio

El modelo del dominio que responde a la solución desarrollada es el siguiente:

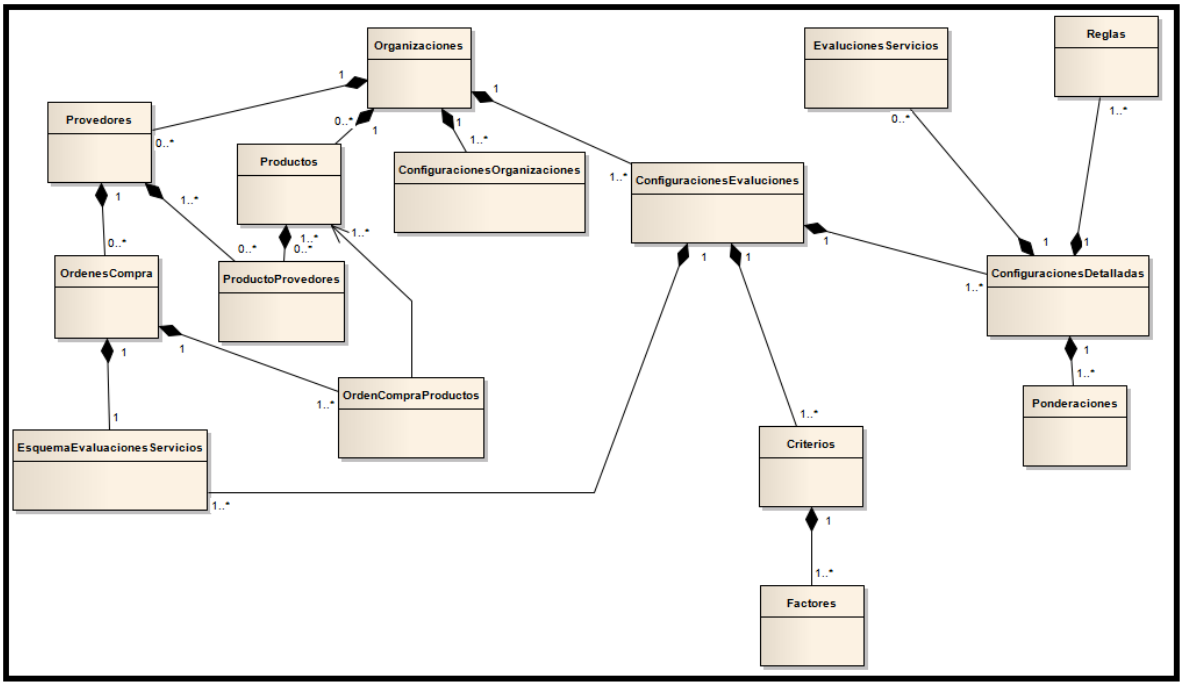

Figura 23 - Modelo del dominio

6.3.1.3. Diagramas de Clases

Este diagrama responde a la decisión de implementar la lógica subyacente de los servicios bajo un paradigma orientado a objetos, así como la solución general. En todo caso la implementación de la lógica subyacente puede ser realizada bajo cualquier tipo de paradigma o tecnología tradicional, requiriendo este artefacto u otros equivalentes, según sea el caso.

Las clases que hacen parte de la solución conceptual son las siguientes:

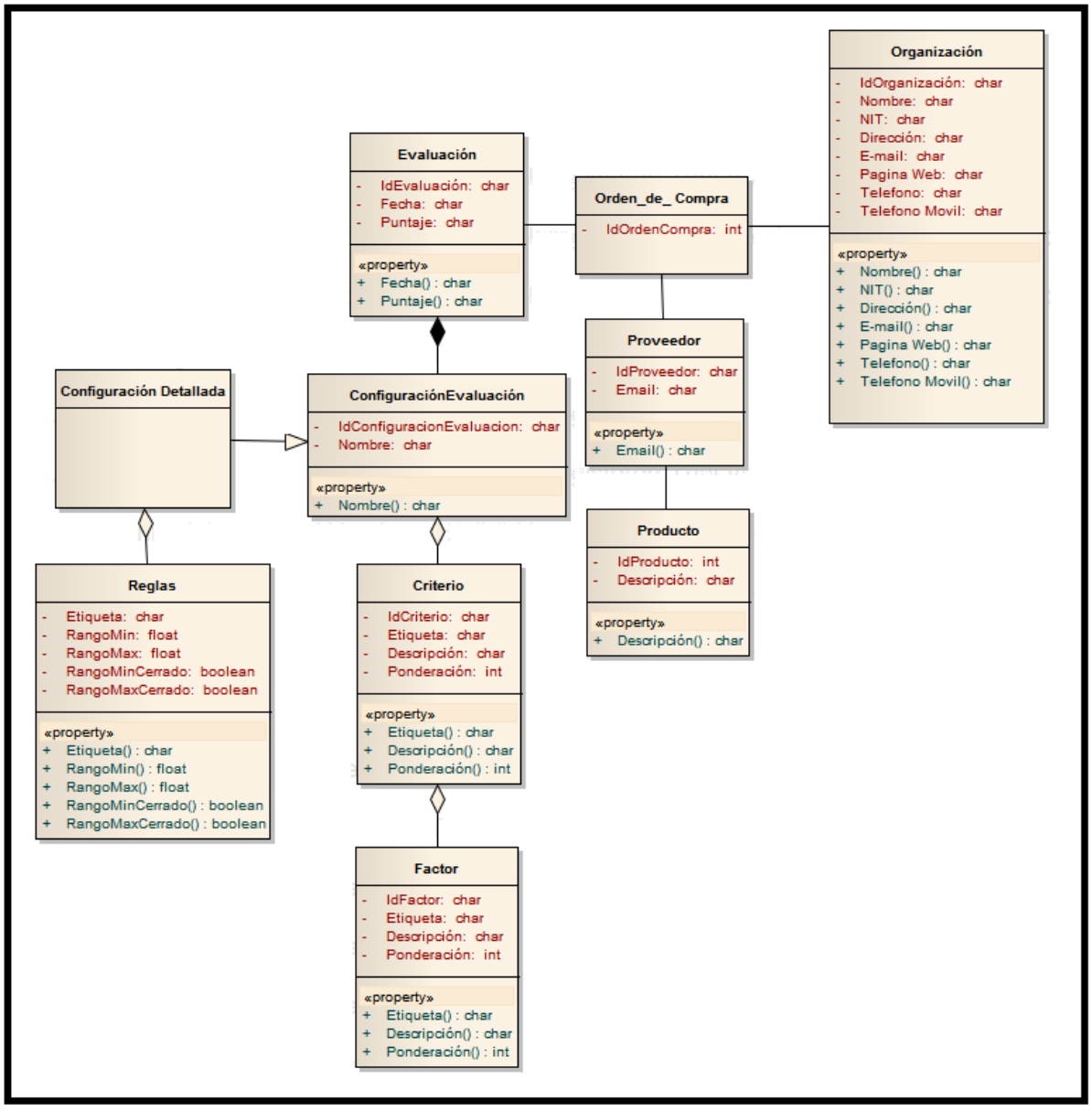

Figura 24 - Diagrama de clases

6.3.1.4. Diagramas de secuencia

Se realizo diagrama de secuencia de las dos principales actividades, la Selección de proveedores y la evaluación de servicios.

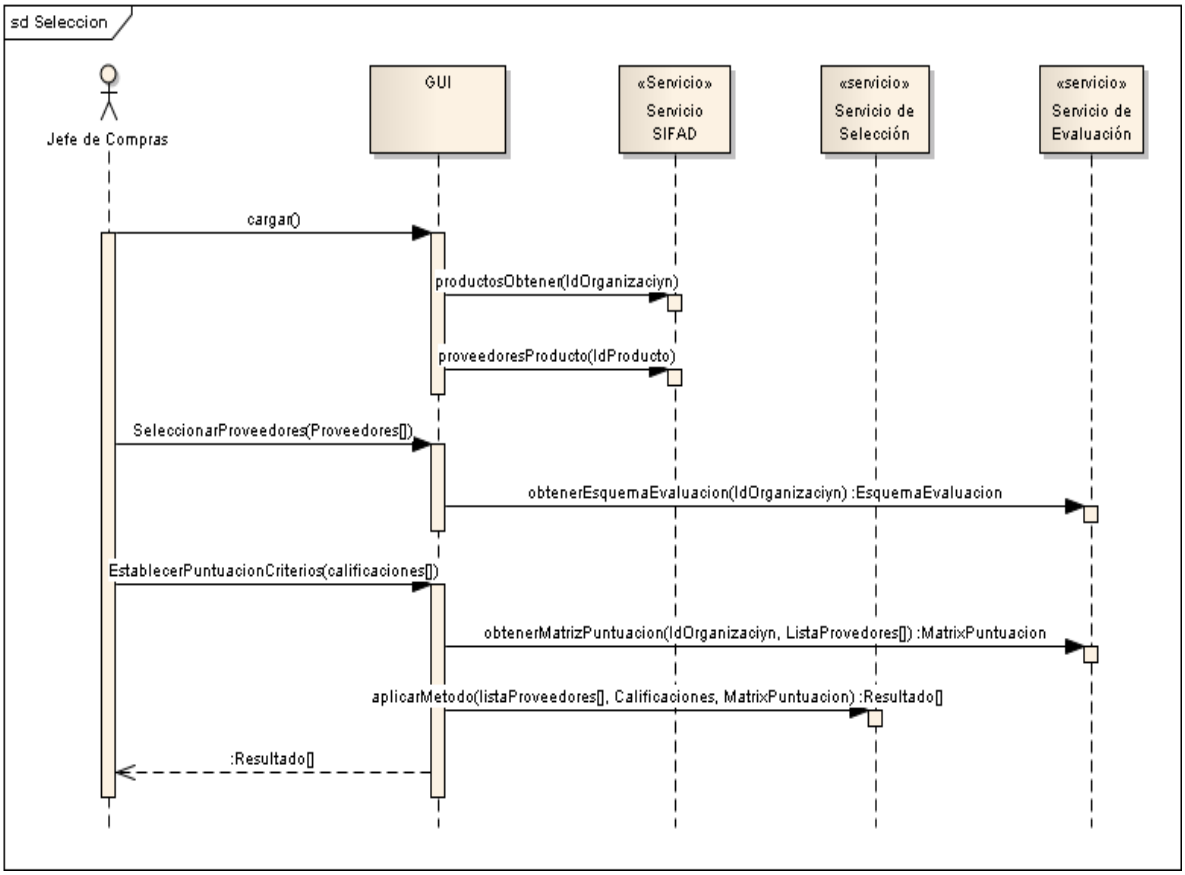

Figura 25 - Diagrama de secuencia: Selección de Proveedores

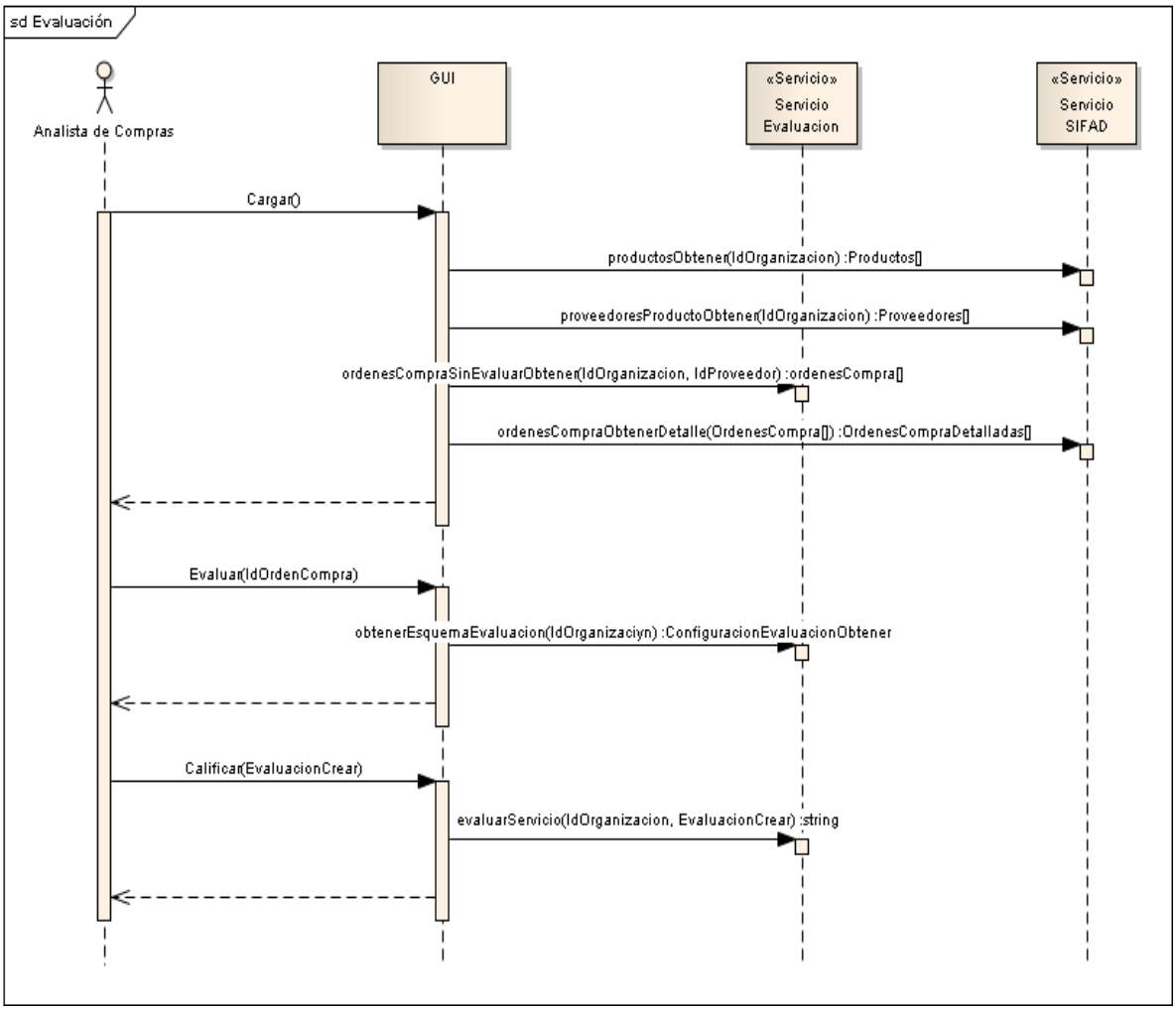

Figura 26 - Diagrama de secuencia: Evaluación de servicios

6.3.1.5. Diagramas de entidad-relación

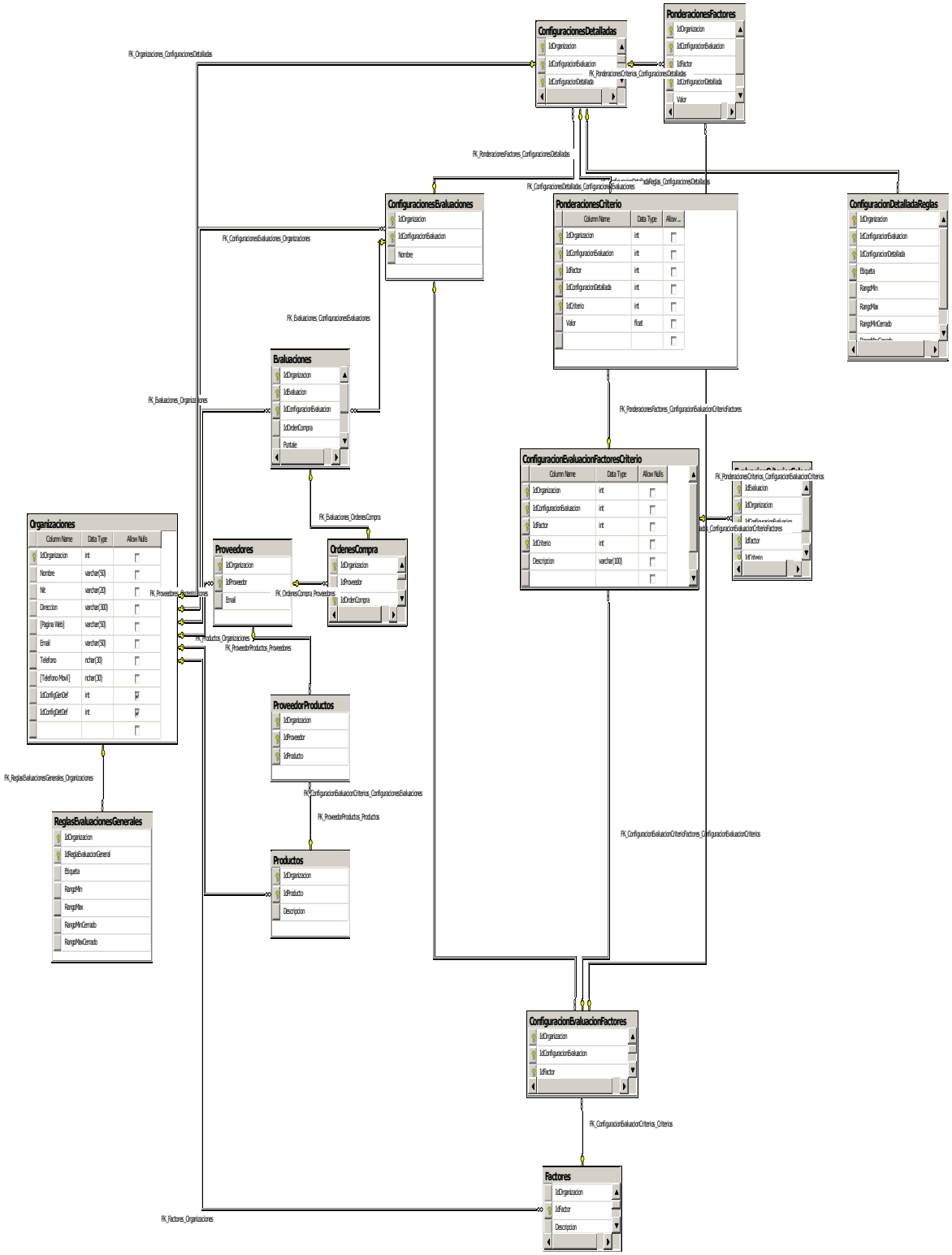

Figura 27 - Diagrama de entidad relación

6.3.1.6. Diseño de contratos

Esta actividad contempla la descripción de las operaciones y mensajes que conforman la interfaz publica del servicio. Este es el punto de partida para una correcta aplicación de los principios de diseños, haciendo hincapié en una diferenciación de un estilo RPC y un estilo orientado a mensajes. En este punto también es posible identificar patrones de intercambio de mensajes requeridos para llevar a cabo la funcionalidad del sistema. Además de la creación de esquemas centralizados para el intercambio de mensajes, en el caso de servicios web, la creación de esquemas XSD.

El diseño de los servicios para el caso de estudio es el siguiente:

Servicio de Evaluación de Servicios

ID	**S1**
Nombre	Evaluación de Servicios
Tipo	Tarea
Taxonomía	Sistema de Información
Requerimientos Funcionales	R.1.1 - R.1.5
Requerimientos No Funcionales	
Método de Descubrimiento	Top Down
Justificación	El servicio encapsula la lógica de evaluación del servicio prestado por un proveedor.
Operaciones	**1. Crear_Evaluación** Entrada: Nombre, Esquema_Configuración. Salida: IdOrganizacion, IdConfiguracionEvaluacion, IdConfiguracionDetallada. Excepciones:
	2. Actualizar_Evaluación Entrada:IdOrganizacion, IdConfiguracionEvaluacion, IdConfiguracionDetallada, Esquema_Configuración_Actualizado. Salida: Esquema_Configuración. Excepciones:

	3.Obtener_Esquema_ de_ Evaluación Entrada: IdOrganizacion, IdConfiguracionEvaluacion, IdConfiguracionDetallada. Salida: Esquema_Configuración. Excepciones:
	4. Evaluar_Servicio_ de_ Proveedor: Entrada: IdOrganizacion,IdOrdenCompra, IdUsuario. Salida: Puntaje. Excepciones:
	5. Crear Criterios Entrada: Descripción, IdOrganización. Salida: Criterio. Excepciones:
	6. Obtener_Criterios Entrada: IdOrganización Salida: []Criterios Excepciones: 1. Que los criterios existan
	7. Crear Factores Entrada: Descripción, IdOrganización. Salida: Factor. Excepciones:
	8. Obtener_Factores Entrada: IdOrganización. Salida: []Factores Excepciones:
	9. Crear Reglas Entrada: RangoMin, RangoMax, RangoMinCerrado, RangoMaxCerrado, Etiqueta. Salida: Etiqueta. Excepciones:
	10. Obtener_Reglas Entrada: IdOrganización. Salida: []Reglas Excepciones:
	11. Obtener_Evaluación_General Entrada: Fecha_Inicial, Fecha Final, IdOrganizacion, IdProveedor Salida: Puntaje Excepciones:

	12. ObtenerProvedores Entrada: IdOrganización, IdProducto. Salida: []Proveedores Excepciones:
	13. Recuperar Evaluación Entrada: []Proveedores Salida: ConglomeradoEvaluaciones Excepciones:

Selección de Proveedores

ID	**S2**
Nombre	Selección de Proveedores
Tipo	Tarea
Taxonomía	Sistema de Información
Requerimientos Funcionales	
Requerimientos No Funcionales	
Método de Descubrimiento	Top Down
Justificación	El servicio encapsula la lógica del método AHP aplicado para seleccionar a los mejores proveedores.
Operaciones	**1. Aplicar_Método_AHP** Entrada: Conglomerado Salida: Alternativas Excepciones:

Notificación de Solicitudes a Proveedores

ID	**S3**
Nombre	Notificación de Solicitudes a Proveedores
Tipo	Entidad
Taxonomía	Sistema de Información
Requerimientos Funcionales	
Requerimientos No Funcionales	
Método de Descubrimiento	Top Down
Justificación	Este servicio se encara de enviar las notificaciones de solicitudes de cotización a los proveedores seleccionados.
Operaciones	**1. Configuración de Cuenta de Correo** Entrada: Servidor de Correo, Puerto, Cuenta de Correo, Password. Salida: Null

	Excepciones:
	2. Configuración del Mensaje Entrada: (String) Estructura_del_Mensaje Salida: Null Excepciones:
	3. Envío de Solicitud Entrada: Para, Asunto. Salida: Void Excepciones:

Sincronización de Datos

ID	S4
Nombre	Sincronización de Datos
Tipo	Infraestructura
Taxonomía	Sistema de Información
Requerimientos Funcionales	
Requerimientos No Funcionales	
Método de Descubrimiento	Top Down
Justificación	Este servicio se encarga de mantener actualizada la información entre la base de datos y los demás servicios.
Operaciones	**1. Configuración de Periodicidad** Entrada: [ ]Periodo(Día, Hora). Salida: Bool. Excepciones: **2. Configuración del Servicio** Entrada: [ ]EsquemaSincronización(TablaCliente, ColumnaCliente, TablaServidor, ColumnaServidor, ConexiónCliente, ConexiónServidor). Salida: Void Excepciones:

Log

ID	**S5**
Nombre	Log
Tipo	Infraestructura
Taxonomía	Sistema de Información
Requerimientos Funcionales	
Requerimientos No Funcionales	
Método de Descubrimiento	Top Down

Justificación	Este servicio se encarga de mantener un registro de los diferentes eventos y errores del sistema.
Operaciones	

Autenticación

ID	**S6**
Nombre	Autenticación
Tipo	Tarea
Taxonomía	Sistema de Información
Requerimientos Funcionales	
Requerimientos No Funcionales	
Método de Descubrimiento	Top Down
Justificación	Este servicio se encarga de alojar los usuario y verificar el acceso a los diferentes recursos de la aplicación.
Operaciones	**1. Crear Recursos** Entrada: []Operaciones, IdOrganización. Salida: Excepciones:
	2. Crear Rol Entrada: Nombre, []Recursos_Asociados, IdOrganización. Salida: Rol Excepciones:
	3. Listar Roles Entrada: IdOrganización. Salida: []Rol. Excepciones:
	4. Modificar Rol Entrada: Rol Salida: void Excepciones:
	5. Crear Perfil Entrada: Nombre, []Roles_Asociados Salida: Perfil Excepciones:
	6. Listar Perfiles Entrada: IdOrganización. Salida: []Perfiles. Excepciones:
	7. Modificar Perfil Entrada: Perfil Salida: void Excepciones:
	8. Crear Usuarios Entrada: Nombre, Password, Rol Salida: Usuario

	Excepciones:
	9. Cambiar Contraseña Entrada: PasswordActual, PasswordNuevo Salida: Bool Excepciones:
	10. Listar Usuarios Entrada: IdOrganización. Salida: []Usuarios. Excepciones:
	11. Modificar Usuario Entrada: Usuario Salida: void Excepciones: 1. Que el usuario no exista
	12. Ingreso de Usuario Entrada: Nombre, Password Salida: Bool Excepciones:
	13. Obtener Recursos Asociados Entrada: Nombre Salida: []Recursos Excepciones:

6.3.2. Implementación

La implementación de un sistema basado en el enfoque COS, tiene mayores retos y requerimientos que el desarrollo bajo un esquema tradicional. Para el desarrollo del caso de estudio se evidenciaron retos a nivel de:

- Compatibilidad, en el caso del proceso de serialización entre .Net y Java. Demostrando que aunque el uso de los Servicios Web ofrece una alta interoperabilidad, la misma está sujeta a como cada una de las tecnologías aplican el estándar.
- Rendimiento, garantizando la mínima cantidad de transformación de tipos.
- Manejo de fallas, por la condición del tipo de enlace utilizado para el transporte de mensajes (fallos de red, de verificación, de protocolo, etc.), las cuales deben ser capturadas y recuperadas.

Además de estas consideraciones, se realizó una generación automatizada de procedimientos almacenados, objetos de la capa de datos y objetos para el intercambio de datos para agilizar el desarrollo.

Puntualmente se utilizó un script personalizado para la herramienta MyGeneration, que genera una clase con base a una tabla de la base de datos y cada campo en un miembro de la clase, agregando los atributos [DataMember]. Todo esto para crear una correspondencia entre los objetos de datos y los esquemas de intercambio de mensajes, además de una correcta generación de los contratos de los servicios.

En el transcurso de esta etapa se generó el siguiente artefacto, el cual es un diagrama de componentes que incluye que describe un Patron MVC y un modelo que incluye los servicios relacionados con los controladores y su interacción con algunos servicios de la solución.

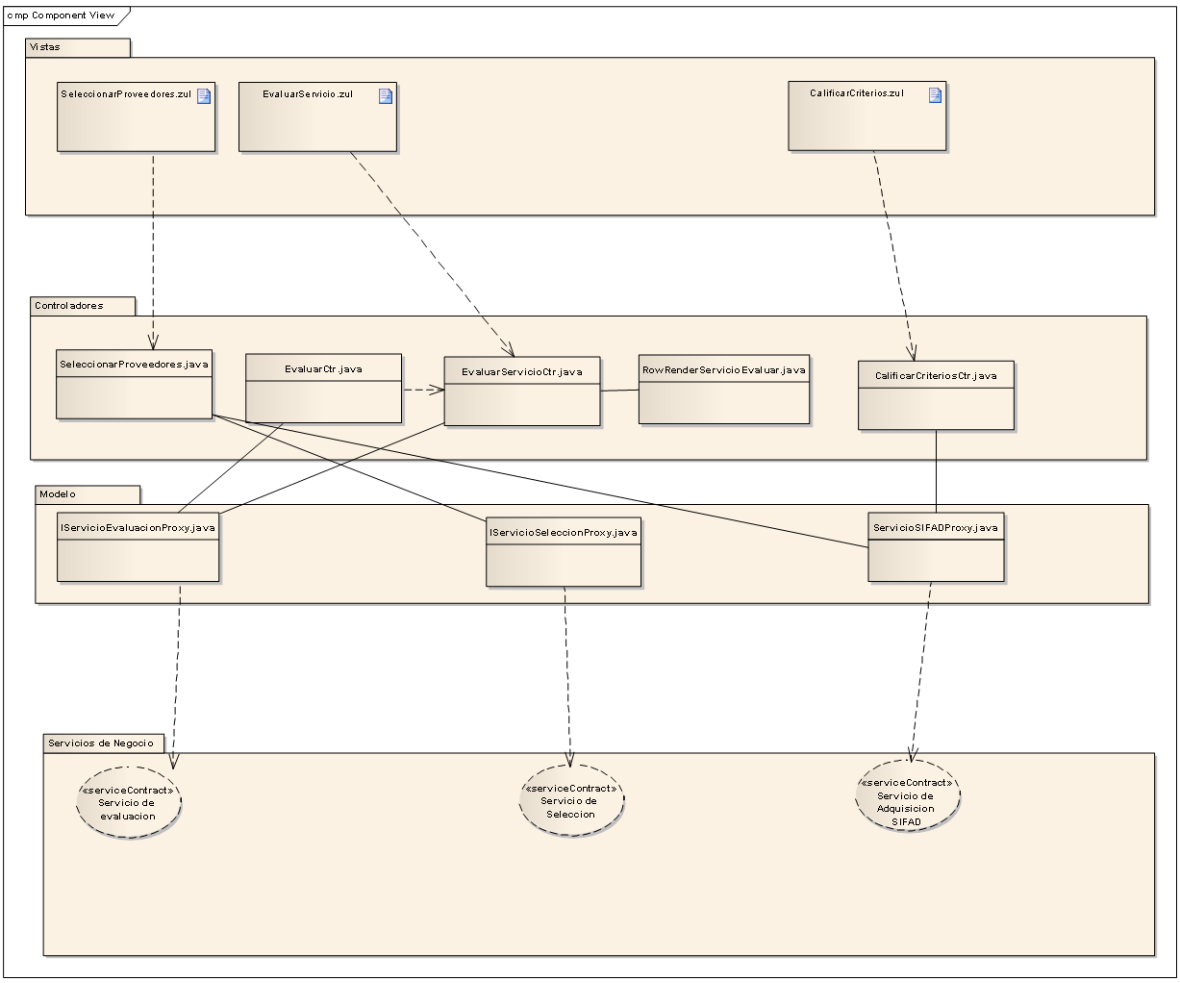

Figura 28 - Diagrama de componentes

6.3.3. Despliegue

En el despliegue de la solución se realizaron las siguientes actividades:

- Configuración del entorno de ejecución
    - IIS 7, se desplegaron los servicios sobre este servidor web, para lo cual fue necesario crear los directorios virtuales y convertir estos aplicaciones web.

    - GlassFish, En este servidor web se desplego la interfaz de usuario, para el consumo de los servicios es necesario instalar un framework a este, se eligió el axis.
- Configuración de orígenes de datos
    - SqlServer 2008, en este motor se crearon la base de datos usados por el servicio de evaluación.
    - Informix, Se identificó y configuro el driver que permite la comunicación entre la base de datos de SIFAD y el servicio de facade de datos.
- Optimización de empaquetado de Interfaz de Usuario Web a entorno de ejecución.

6.3.4. Pruebas

Con el ánimo de verificar el correcto funcionamiento del sistema y sus componentes se realizaron las siguientes pruebas

- Pruebas funcionales
- Pruebas transaccionales
- Pruebas de fallas en los canales de comunicación
- Validación de acceso a meta información de servicios Web (WSDL's)
- Pruebas unitarias de aplicación del método AHP
- Pruebas de usabilidad

7. VERIFICACIÓN DE LOS PRINCIPIOS DE DISEÑO DE LA COS

En este apartado se busca revisar el cumplimiento de los principios de la computación orientada a servicios a nivel del diseño y la implementación de la herramienta computacional de selección de proveedores.

En el capítulo 5 se estudiaron en detalle los principios de diseño de la COS, cuáles eran las implicaciones de utilizarlas, verificando cuales eran las metas de la filosofía. A continuación se revisa en detalle cada principio:

7.1. Estandarización De Contratos

A nivel del lenguaje funcional se le dio nombre a los servicios de entidades teniendo en cuenta su correspondencia con las entidades de negocios desde las cuales se derivaban, el nombre de los servicios de tarea se dio con base en los procesos de negocios que se automatizaron, para lo cual se utilizó la siguiente notación, servicio+ verbo sujeto. Además ninguna operación fue nombrada igual a un servicio.

En cuanto a la representación de los Datos se utilizaron los atributos [***DataMember***] para los campos que se requieren serializar y [***DataContract***] para las clases, ambos pertenecientes al framework de .NET, los cuales permiten regular la serialización de los objetos asegurando la interoperabilidad y la estandarización de los datos.

Se utilizaron únicamente tipos de datos primitivos y agrupaciones de los mismos, persiguiendo altos niveles de interoperabilidad. En especial para el manejo de arreglos, donde se especificaron los arreglos como listas ***List<E>***, para garantizar la máxima interoperabilidad, lo cual impactó en la transformación de listas a arreglos y viceversa.

Por último, la creación de esquemas XML para la descripción de mensajes y tipos de datos complejos usados fue delegado a la plataforma tecnológica. El trabajo se centró en la creación de clases, las cuales indicaban la forma explícita en que serían serializadas por el servidor, lo cual es suficiente para generar una representación única del esquema XML, para asegurar que las definiciones de los mensajes de entrada y salida sean modeladas en concordancia con la lógica del modelo de datos existentes.

Las clases pertenecientes al ***namespace*** se listan en el DAL.***DataContracts***. Para el caso puntual de .Net, es importante definir un mismo valor del parámetro ***Namespace*** en el atributo ***DataContract***, para garantizar que la descripción de los mensajes y tipos de datos complejos sea una sola.

7.2. Bajo Acoplamiento.

La lógica es dependiente del contrato, sin embargo, el contrato no es dependiente de la lógica.

Se revisó el nivel de acoplamiento en cada uno de los servicios descritos en el apartado anterior. Estos exhiben alto acoplamiento en cuanto a la lógica implementada y su contrato, y por el contrario, los contratos están débilmente acoplados a la lógica implementada. Por ejemplo el servicio de *Selección de Proveedores,* el cual fácilmente podría llegar a implementar un nuevo método de selección diferente al usado actualmente (AHP), sin realizar ningún tipo de cambios al contrato.

El servicio de evaluación y selección, y la interfaz reflejan independencia entre sí. Un ejemplo que demuestra lo anterior podría ser: “En el caso de requerir modificar la lógica del negocio plasmada en alguno de los servicios (selección, evaluación), tal y como esta implementado no afecta los demás servicios”

El servicio de Evaluación posee muchas operaciones de tipo entidad, lo cual disminuye el posible acoplamiento funcional con la lógica de otros procesos de negocios padres o externos.

Se diseñó un servicio de fachada para la integración con el software legado SIFAD que se utiliza actualmente en la institución. Este servicio fue diseñado para crear una interfaz estandarizada para el consumo de datos, que permite separar la lógica funcional de la capa de acceso a datos, facilitando la integración con diferentes fuentes de datos y el despliegue sobre diferentes ambientes.

La matriz de puntuaciones calculada y enviada por el servicio de evaluación es enviada al cliente (Interfaz de usuario) y de esta enviada al servicio de selección. Esto representa desacoplamiento, sacrificando rendimiento ya que es necesario el empaquetamiento y la serialización y envío sobre la red de los datos.

7.3. Reusabilidad.

Los servicios creados presentan diferentes niveles de reusabilidad, el servicio de selección por no estar atado a una lógica de negocios especifica es altamente agnóstico, lo cual se evidencia en el diseño de su contrato, que no está atado a ninguna de las entidades presentes en los otros servicios.

Por el contrario, el servicio de evaluación tiene mayor dependencia por ser un servicio de negocio, cuya lógica está asociada a entidades de datos externas, que hacen parte del contexto funcional del proceso de negocio de compras, como son: órdenes de compra, proveedores y productos.

En cuanto al servicio de SIFAD, aunque está atado directamente al sistema de legado, por ser un servicio de entidad podría ser utilizado por cualquier sistema que requiera utilizar las entidades de datos expuestas.

7.4. Autonomía.

Los servicios se diseñaron considerando su despliegue sobre sistema operativo Windows XP ó superior, y el entorno de programación .Net 4.0 o superior, lo cual implica que los niveles de autonomía están acotados por las anteriores características.

Para el acceso a los servicios, en el contrato técnico se utilizó el perfil ***BasicHttpBinding*** aceptado por la gran mayoría de plataformas de desarrollo y ejecución de aplicaciones de software. Esta decisión permite alcanzar altos niveles de autonomía, al brindar libertad para ampliar, actualizar o sustituir la tecnología de un servicio en respuesta a nuevas necesidades ó el deseo de aprovechar las innovaciones tecnológicas, sacrificando seguridad al no permitir implementarla en la capa de transporte.

El servicio de evaluación dispone de su propia base de datos y es el único que accede a ella, brindándole mayor control sobre los datos utilizados.

El servicio de selección es el que más recursos de procesamiento consume, por lo anterior se recomienda que más adelante y en función a los niveles de concurrencia se considere correrlo bajo una máquina virtual o maquina independiente con gran capacidad de procesamiento y memoria.

7.5. Manejo de Estado.

El escenario favoreció al no manejo de estado en los servicios debido a la baja composición y el número reducido de servicios, y por ello no hubo manejo de datos de sesión, de contexto y de entidades de negocio en memoria.

7.6. Composición

En el diseño del sistema, se consideró realizar diferentes composiciones tales como, la composición del servicio de evaluación y el servicio de autenticación y notificación. Lamentablemente dichos servicios no fueron implementados y se estipula como trabajo futuro.

Vale la pena denotar que se delegó lógica de composición a la interfaz gráfica, lo cual pudo haberse realizado de una manera diferente a través del uso de un servicio controlador principal, que encapsulara al servicio de evaluación, selección, notificación y autenticación. Solo con lógica de orquestamiento, lo cual probablemente sea una solución más acorde a este principio y que debe ser tomado en cuenta para trabajo futuro.

A nivel de diseño se buscó aplicar este principio, inicialmente se diseñaron servicios de autenticación y de notificación que dependerían del servicio de evaluación, pero hasta el momento, en la implementación solo se logró la construcción del servicio de evaluación y de selección, los cuales no requieren de este principio para su funcionalidad.

7.7. Descubrimiento

Desarrollar una aplicación orientada bajo este principio no era prioridad en el desarrollo de este trabajo, de lo contrario implicaría la necesidad de implementar contratos altamente interpretables, ricos en semántica y en documentación para agentes humanos, expuestos en repositorios de servicios públicos, entre otras características que benefician la distribución de aplicaciones bajo el modelo SaaS.

7.8. Abstracción

La tabla 6.16 describe la aplicación de este principio.

Tipo de Abstracción	Servicio de Evaluación		Servicio de Selección		Servicio SIFAD	
	Publicado	**No Publicado**	**Publicado**	**No Publicado**	**Publicado**	**No Publicado**
Abstracción Tecnológica.	Ninguna	• Se implementó en tecnología .NET, utilizando C# framework 4.0 • Para la capa de datos se usó MyGeneration y Doodads, Motor de BD SQL 2008	Ninguna	• Se implementó en tecnología .NET, utilizando C# framework 4.0. • Se utiliza la librería TPL de programación paralela.	Ninguna	Se implementó en tecnología .NET, utilizando C# framework 4.0.
Abstracción Funcional	Las órdenes de compra deben estar registradas en el servicio de evaluación como no evaluadas.	Ninguna	• En la parte no técnica del contrato del servicio de Selección se especifica que el método de selección usado es AHP. • El orden en que se envía la matriz de puntuación coincide con el orden de los proveedores enviados.	Ninguna	Ninguna	Ninguna
Abstracción Lógica.	Todas las operaciones expuestas	Lógica de validación de datos y manejo de concurrencia.	Escala de Saaty	Validaciones de división entre cero.	Ninguna	Ninguna
Abstracción Calidad del Servicios.	Tamaño máximo del mensaje	• Tiempos de respuesta. • Concurrencia máxima. • Disponibilidad.	Tamaño máximo del mensaje	• Tiempos de respuesta. • Concurrencia máxima. • Disponibilidad	Tamaño máximo del mensaje.	• Tiempos de respuesta. • Concurrencia máxima. • Disponibilidad.

Tabla 4- Aplicación principio de abstracción

8. EXPERIENCIAS

La sed de conocimiento y de innovación fue una constante de aquellos jóvenes integrantes de la generación de la Celula.Net del 2009. En búsqueda de un horizonte investigativo tanto para estudiantes de pregrado como de maestría, la palabra SOA comenzó a retumbar en el día a día del grupo. El pionero y principal impulsador del estudio de esta nueva área del conocimiento, es nuestro mentor, el Ingeniero Edwin Puertas. En el inicio todos aportábamos ideas y divagamos en posibles proyectos, que pensábamos podían ser viables. Se plantearon proyectos como, usos de ontologías en el proceso de descubrimiento de servicios, la creación de un framework para el desarrollo COS, consumidores inteligentes (generación automática en tiempo de ejecución de Stubs), agentes de búsqueda de mejores servicios y consumo automático, entre muchas otras ideas.

Pero hubo una constante al tratar de concretar cada uno de estos proyectos, la falta de experiencia y poco conocimiento en la temática. Si quiera pensar en apuntar a temáticas de mediana complejidad referente a SOA era difícil. No existían referentes bibliográficos, ni autores representativos locales que permitieran la apropiación de las bases conceptuales necesarias para la realización de los proyectos anteriormente mencionados.

La principal motivación desde entonces fue lograr que estas bases conceptuales que no existían, fuesen consolidadas, validadas a través de una extensa revisión bibliografía y su aplicación en un caso real de desarrollo, para ponerla a disposición de la comunidad académica de la Tecnológica de Bolívar, reducción esta brecha y abriendo camino a trabajos que en un futuro pudieran culminar los proyectos que en un principio se plantearon o enfrentar cualquiera de las necesidades investigativas actuales de la COS.

El trabajo inicial giro entorno a SOA, como estilo arquitectural. Nuestra atención se centro en este término gracias al boom por el cual este pasaba. No obstante nuestros deseos de formar una buenas bases conceptuales, nos impulsaron a

ahondar en la temática, permitiéndonos comprender que SOA solo era la punta del Iceberg y que bajo esta punta visible se encontraba un paradigma cuyos principios, metas y demás elementos eran las bases sobre las cuales esta arquitectura era posible. Fue en este punto que decidimos que debido a la falta de conocimiento, nuestro principal objetivo debía ser el estudio de la computación orientada a servicio, el cual es un término menos conocido que SOA, pero igual de relevante.

Todos estos esfuerzos dan los primeros frutos, sirviendo de apoyo para el trabajo de maestría **Integración empresarial orientada a Servicio para las PyMES** del Ingeniero Edwin Puertas, y creando también la oportunidad para la validación de toda esta revisión conceptual, a través de el diseño e implementación de un sistema para soportar el proceso de selección y evaluación de proveedores para la UTB, a través del trabajo conjunto con la tesis de maestría de la Ingeniera Yuliana Puerta. El contexto operativo y tecnológico del sistema, así como la necesidad de integración con un sistema legado como SIFAD, y altas probabilidades de reusabilidad; hicieron de esta problemática un caso de estudio atractivo para la aplicación de la filosofía propuesta por COS.

Nuestra experiencia dentro del diseño e implementación nos permitió ver que aunque las bondades de la computación orientada a servicio son muchas, los requerimientos y retos para llevarlas a cabo también lo son. Estos van más allá de la temática tecnológica, cambiar de un paradigma tradicional a uno orientado a servicios, como todo proceso de cambio es difícil, exige abrir nuestros esquemas mentales a nuevas formas de hacer software, abandonar practicas que se daban como inmutables y hacer nuevas que respondan a requerimientos que se daban como solucionados.

En nuestra experiencia se vivieron situaciones como tratar de diseñar la solución con base al modelo de datos y no a los proceso de negocio. Este puede ser un error que haga fracasar la orientación a servicios, donde se exige un pensamiento Top Down, donde se definan primero las entidades de intercambio de datos entre los diferentes procesos de negocio y luego las entidades subyacentes que

soporten su implementación. Otra situación que debe tenerse en cuenta es la necesidad de extender las metodologías tradicionales, incluyendo nuevas actividades y artefactos que permitan apuntar al cumplimiento de los principios propuestos por la COS, tal puede la identificación de servicios candidatos y la definición de contratos.

Por último cabe destacar que el desarrollo de una tesis, es un proceso arduo y extenso, que permite adquirir habilidades investigativas, profesionales y humanas, dejan la satisfacción de realizar un buen trabajo e innumerables conocimientos y experiencias.

9. CONCLUSIONES

La bibliografía existente sobre computación orientada a servicios aunque abundante, también es bastante joven, autores y grupos de investigación referentes se han consolidado recientemente, a consecuencia de lo cual una cantidad significativa de esta tiene tintes comerciales y publicitarios. Dicha situación obligo a centrar los primeros esfuerzos del trabajo en la identificación y asimilación de la información académica y científica referente al tema.

El proceso de recopilación bibliográfica implico, numerosas horas de lectura que incluyeron un rango de textos dirigidos desde principiantes a expertos consolidados en el tema, además del estudio del origen de la filosofía de servicios en términos generales necesario para entender que impulso el origen, y cuáles son las metas principales de la COS.

La realización de la revisión bibliográfica evidencio la importancia de la creación de un documento que pudiera ser usado como referente de la temática en el ámbito regional y local, y particularmente dentro de la Universidad Tecnológica de Bolívar, en el cual no solo se consoliden y catalogue de manera coherente los conceptos básicos de la Computación Orientada a Servicios, sino en el que se muestre que es posible la aplicación de dichos conceptos en la solución de problemáticas empresariales reales de nuestra localidad.

El modelo de construcción de software propuesto por la Computación Orientada a Servicios se elaboró como una respuesta para dar soluciones problemáticas empresariales, fundamentándose en una serie de principios que tienen como meta lograr características que frecuentemente son deseadas en este tipo de soluciones. Lo anterior obligo a realizar una descripción completa de cada principio en la cual se muestre el marco conceptual del principio, la finalidad que persigue y los retos que implica su implementación.

El software empresarial frecuentemente es desplegado en ambientes distribuidos, en todo caso un sistema monolítico se puede considerar un caso especial de un sistema distribuido, por lo cual la Computación Orientada a Servicios fue concebida para responder a este requerimiento desde su creación, es decir que los sistemas construidos bajo el paradigma podrá estar compuestos de elementos como servicios, recursos, nodos, clusters, grillas, etc., que trabajan para lograr un objetivo común, pero que además es importante considerar que todos estos elementos se pueden encontrar dispersos geográficamente, empotrados en ambientes diferentes, operando en diversos entornos de red, sobre diferentes sistemas operativos y ejecutándose incluso bajo diferentes plataformas de desarrollo.

Desde la arquitectura interior de un servicio hasta el más alto grado de composición se debe garantizar:

- **Integridad**, ya que una vez iniciada una operación es necesario garantizar que esta sea indivisible, consistente, que no interfiera con otras y persista en el tiempo. Recuerde estas siglas ACID (Atomic, Consistent, Isolated and Durable). El uso de máquinas de flujos de trabajo como Microsoft Workflow Foundation o Java Workflow Tooling, son de gran ayuda para llevar a cabo este principio en un nivel de orquestación en composiciones de servicios.

- **Transparencia**, tanto para usuarios como para aplicaciones. En acceso, ocultando detalles de cómo es representada la información y como se accede a ella. En localización, haciendo transparente la ubicación real de los elementos, como su dirección física o su posición geográfica. En migración, donde la reubicación de un elemento no debe alterar el funcionamiento del sistema. En redundancia, debe ser transparente el hecho de agregar y correlacionar elementos que cumplan la misma funcionalidad. En concurrencia, para el acceso simultaneo a un elemento. En fallas, ocultando la falla y recuperación de un elemento.

- **Tolerancia a Fallas**, garantizando la continuidad de operaciones. Las medidas que se tomen para garantizar diferentes grados de la continuidad de las operaciones depende, de la importancia del sistema y van más allá de considerar la infraestructura. No basta con tener fuentes de poder, conexiones de red, dispositivos de almacenamiento e incluso nodos redundantes; es de vital importancia garantizar que la falla de un elemento, no conlleve a la caída completa del sistema. Los canales de comunicación siempre estarán expuestos a fallas, y a nivel de programación es importante la captura de estos errores, que no afecten el correcto funcionamiento de la aplicación y con mecanismos de recuperación transparentes al usuario.

- **Escalabilidad**, un sistema distribuido puede crecer respecto a tres dimensiones:
    - En tamaño, que se traduce en agregar más usuarios o recursos al sistema,
    - Geográficamente, refiriéndose a la distribución de los recursos en diferentes espacios geográficos, dispersos inclusos a distancias continentales.
    - En la administración, permitiendo la confederación de diferentes políticas y reglas que ejerzan sobre un conjunto de elementos, y aun así conservando una administración global.

  El crecimiento de un sistema distribuido en cualquiera de sus dimensiones trae como consecuencia directa perdida en el rendimiento del mismo, es por esto que este es un tema crítico y sobre el cual se deben emplear técnicas sofisticadas que permitan minimizar el impacto en el rendimiento al momento de escalar el sistema. Técnicas tales como algoritmos descentralizados, programación paralela, utilización de hilos, distribución de los procesos por capas, etc.

- **Latencia**, siendo esta la que determina el tiempo de respuesta a una solicitud realizada por un elemento del sistema sobre otro. Incluso en

medios de comunicación realizados a través de medios de alto rendimientos como conexiones ópticas, existen procesos intermedios como la serialización, enrutamientos, codificaciones, autentificaciones, etc., que deben ser considerados al momento del diseño y seleccionar muy bien la infraestructura necesaria que los va a soportar, en búsqueda de la minimización de latencia. También es importante la utilización de mecanismos visuales que permitan al usuario percibir el rendimiento de forma diferente y no pantallas congeladas que pueden generar una sensación de mal funcionamiento.

10. REFERENCIAS

[1] GEORGAKOPOULOS, Dimtrios y Papazoglou, Michael (2009): *Service-Oriented Computing*. MIT Press. ISBN 978-0-262-07296-0

[2] BUSCHMANN, Frank; Henney, Kevlin; Schmidt, Douglas C. (2007): *Pattern-Oriented Software Architecture A Pattern Language for Distributed Computing*. Vol. 4. John Wiley & Sons Ltd. ISBN-13: 978-0-470-05902-9

[3] KRUCHTEN, PHILIPPE (2004): *The Rational Unified Process: An Introduction (3rd Ed.).* ISBN 0-321-19770-4.

[4] Organization for the Advancement of Structured Information Standards: Reference Model for Service-Oriented Architecture, Version 1.0, Committee Specification, July 2006, http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

[5] M. L. LIU (2003): *Distributed Computing: Principles and Applications*. ISBN 978-0201796445

[6] PHILLIP A., Laplante; Zhang, Jia; Jeffrey Voas (2008). "*What's in a Name? Distinguishing between SaaS and SOA"*. IT Pro, Mayo / Junio 2008.

[7] ERL, Thomas (2007): *SOA: Principles of Service Design.* Prentice Hall. ISBN 0-13-234482-3

[8] FITZSIMMONS, James; Fitzsimmons, Mona (2007): *Service Management: Operations, Strategy, Information Technology*. McGraw-Hill/Irwin. 6° Edición. ISBN 0077228499.

[9] ABE, Tadahiko (2005): *"What is Service Science?".* Economic Research Center, The Fujitsu Research Institute. Research Report No. 246 December 2005.

[10] TURNER, Mark; Budgen, David; Brereton, Pearl (2003). *"Turning Software into a Service"*. Keele University. http://www.shsu.edu/~csc_ghg/classes/Summer2003/CS470/webservices/IEEEPaper-TurningSoftwareIntoService.pdf

[11] TSAI, W.T.; CHEN, Yinong (2006). *"Introduction to Service-Oriented Computing"*. ASU SOA Workshop. http://www.public.asu.edu/~ychen10/activities/SOAWorkshop/Background.pdf

[12] ELRAD, Tzilla; Filman, Robert E.; Bader, Atef (2001): "*Aspect – Oriented Programming: Introduction"*. Communications of the ACM. October 2001/Vol. 44, No. 10.

[13] WATSON, Andrew (2008): *"Achieving software agility with BPM, SOA and MDA®"*. 19th Australian Conference on Software Engineering.

[14] GARTNER (1996). *"Service Oriented Architectures, Part 1"*. SSA Research Note SPA-401-068.

[15] GARTNER (1996). *"Service Oriented Architectures, Part 2"*. SSA Research Note SPA-401-069.

[16] ARBOLEDA, Liliana M. C. (2004). *"Servicios WEB: Distribución e integración"*. Universidad Icesi. http://www.icesi.edu.co/biblioteca_digital/bitstream/item/403/1/larboled_servicios-web.pdf

[17] REYNOSO, Billy. *"Arquitectura Orientada a Servicios (SOA)"* [en línea]. http://download.microsoft.com/download/4/F/F/4FF88340-43CC-4C5B-8E50-09002969D0DD/20051129-ARC-BA.ppt

[18] W3C. "*Extensible Markup Language (XML) 1.0 (Fifth Edition)"* [en línea]. http://www.w3.org/TR/2008/REC-xml-20081126/

[19] BENZ, Brian; Durant, John R. (2003): XML Programming Bible: Wiley Publishing, Inc. ISBN: 0-7645-3829-2.

[20] W3C*. "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)"* [en línea]. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[21] GONZÁLEZ C., Benjamín*." SOAP (Simple Object Access Protocol)"* [en línea]. http://www.desarrolloweb.com/articulos/1557.php . DesarrolloWeb.com. 7 de Julio de 2004.

[22] BOTELLO CASTILLO, Alejandro. *Construcción de Servicios Web con SOAP.* [En línea]. Revista digital universitaria. 31 de marzo del 2002 Vol.3 No.1. http://www.revista.unam.mx/vol.3/num1/art3/

[23] W3C. *"Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language".* http://www.w3.org/TR/2007/REC-wsdl20-20070626

[24] IGNACIO GARCÍA, Macario Polo; Francisco Ruiz, Mario Piattini(2005): *"Servicios Web".* Departamento de Informática, Universidad de Castilla-La Mancha. Informe Técnico UCLM DIAB- 05 - 01 – 1. Enero 2005.

[25] PAPAZOGLOU, Michael; TRAVERSO, Paolo; DUSTDAR, Schahram; LEYMANN, Frank (2007). "Service-Oriented Computing: State of the Art and Research Challenges". IEEE Noviembre 2007.

[26] CHOU, David; deVadoss, John; Erl, Thomas. [Et al.] (2010). *SOA with .NET & Windows Azure: Realizing Service-Orientation with the Microsoft Platform.* Prentice Hall/PearsonPTR .ISBN: 0131582313.

[27] ERL, Thomas. *Service-Oriented Architecture: Concepts, Technology & Design.* Prentice Hall/PearsonPTR. ISBN: 0131858580.

[28] ERL, Thomas.*SOA Design Patterns.* Prentice Hall/PearsonPTR. ISBN: 0136135161.

[29] PUERTAS CRUZ, Yuliana (2011). Diseño de una herramienta computacional orientada a servicios para la toma de decisiones en la selección de proveedores. Programa de ingeniería de Sistemas, Universidad Tecnológica de Bolívar