UNIVERSIDAD AUTÓNOMA DE SINALOA

FACULTAD DE INFORMÁTICA CULIACÁN
MAESTRÍA EN INFORMÁTICA APLICADA

Integración de unidades de software a través de un modelo canónico de datos dinámico basado en mensajes agnósticos dirigido al acoplamiento externo y de datos

TESIS

Que como requisito para obtener el grado de
Maestría en Informática Aplicada

PRESENTA
Juan Antonio Ruiz Ceniceros

DIRECTORES DE TESIS
Dr. José Alfonso Aguilar Calderón
Dra. Carolina Tripp Barba

Culiacán de Rosales, Sinaloa a 11 de Mayo de 2023

Índice General

Índice de Figuras

Índice de Tablas

Resumen

Actualmente, se ha hecho de gran necesidad en las empresas la integración de sistemas adquiridos a terceros y legados. Esto se debe principalmente a la necesidad de intercambio de información entre entidades, tales como bancos, proveedores, clientes, aliados, etc. Por lo que, es indispensable garantizar la integridad de los datos, así como mantener al día las integraciones con los diferentes cambios que se pueden presentar, debido a lo cual es necesario reducir el riesgo en transacciones y evitar perder información. En esta tesis se presenta una extensión de una arquitectura denominada Modelo Canónico de Datos Dinámico mediante mensajes agnósticos (MCDD). El aporte consiste en el tratamiento del bajo acoplamiento en la integración de unidades de software, llamado también integraciones de unidades de software débilmente acopladas. Particularmente, se enfoca en el acoplamiento de datos y el acoplamiento externo. Con esta propuesta se disminuye el tiempo y costo de la integración y de su mantenimiento, asimismo se maximiza su escalabilidad y se impulsa su reutilización.

Palabras clave: Integración de Sistemas, Integración de Sistemas Legados, Unidades de Software, Escalabilidad de Software, Bajo Acoplamiento.

Abstract

Integrating systems acquired from third parties and legacies has become of great necessity in companies. This is mainly due to the need to exchange information between entities, such as banks, suppliers, customers, allies, etc. Therefore, it is essential to guarantee the integrity of the data and keep the integrations up to date with the different changes that can occur. Furthermore, it is necessary to reduce the risk in transactions and avoid losing information. This thesis presents an extension of the Canonical Model of Dynamic Data architecture through agnostic messages. The contribution involves treating the low link in the integration of software units, also called integrations of loosely coupled software units. In particular, it focuses on data sugar and external sugar. This proposal reduces the time and cost of their integration and maintenance, maximizing their scalability and promoting their reuse.

Keywords: Systems Integration, Legacy Systems Integration, Software Units, Software Scalability, Loose Coupling.

Agradecimientos

Agradezco profundamente a mi familia, amigos y sobre todo a Dios por permitirme concretar este trabajo.

A mis directores de tesis, al Dr. José Alfonso Aguilar Calderón y a la Dra. Carolina Tripp Barba, agradezco profundamente su apoyo, guía y consejo, sobre todo su paciencia, mil gracias.

Dedicatoria

Dedico este trabajo a G.·. A.·. D.·. U.·. por permitirme arrancar un destello de luz de su inmenso e infinito conocimiento, a mi esposa amada Olga Lilia, y mis hijos queridos Isaac, José y Caleb por su amor incondicional, a mis padres por ser un pilar en mi formación, también al Dr. José Alfonso Aguilar Calderón y a la Dra. Carolina Tripp Barba por sus consejos y ayuda en esta empresa tan noble como lo es el conocimiento y la investigación.

Juan Antonio Ruiz Ceniceros

Capítulo I
Introducción

Actualmente, la industria del software está presente en todas las áreas del conocimiento a través de los sistemas computacionales, por lo que resulta indispensable garantizar la confiabilidad de los datos y de la información producida debido a que los usuarios toman decisiones que afectan directamente a la empresa. Lo complejo en la industria del software se debe en parte a la problemática en el desarrollo de sistemas de información para ambientes heterogéneos, distribuidos, abiertos y dinámicos. En este sentido, aparecen nuevos retos tecnológicos y oportunidades comerciales por la necesidad de integrar sistemas que interactúen superando las fronteras de las organizaciones además de operar efectivamente integrando sistemas de información de terceros. La integración es probablemente la característica más importante en estos sistemas debido al intercambio de información con otras entidades como las bancarias, por ello son más difíciles de construir debido a que se tiene que integrar diferente tecnología. Para paliar esta problemática actualmente se necesita la mejora de los modelos de computación tradicional e incluso de la creación de nuevos modelos y paradigmas. En este contexto se ha ido desarrollando un área de la Ingeniería de Software que se conoce como Integración de Aplicaciones Empresariales (EAI, por sus siglas en inglés) [1] como una metáfora de diseño y como una tecnología concreta para la integración de sistemas de terceros en las empresas.

1.1 Motivación

Durante la década de 1990, las pequeñas y medianas empresas (PyMES) y distintas corporaciones compraron soluciones de software de aplicación para mejora de sus procesos a empresas como SAP [2], Oracle ERP [3], *PeopleSoft* [4], JDEdwards [5], Siebel [6], etc. Aunque estas soluciones funcionan bien individualmente, no funcionaron bien en conjunto, por lo que crearon islas de información, es decir, datos dispersos en la organización, creando redundancia de estos. Como resultado, los empleados actualizan manualmente los datos asociados en cada sistema, un proceso que rápidamente se volvió tedioso, a razón de esto se optó por encontrar formas de integrar los sistemas. Estas situaciones surgen de fenómenos naturales en la empresa como lo son: aplicaciones que nacieron y crecieron aisladas, aplicaciones de diversos fabricantes, fusiones y adquisiciones entre otras. Por lo que, en la empresa actual, la cual cuenta con departamento de Tecnología de la Información (TI), es obligatoria la integración de aplicaciones conforme a las necesidades del negocio. Lamentablemente, en las diferentes arquitecturas y métodos de integración de sistemas existe en mayor o menor grado una dependencia entre las unidades de software, es decir un estrecho

acoplamiento. Esto ocasiona un riesgo en la integración debido al incremento de tiempo en el desarrollo lo que se deriva en un alto costo en mantenimiento en función al dinamismo del negocio y sus necesidades de cambios en las aplicaciones durante toda su existencia. Ante esta situación, surge la necesidad de nuevas arquitecturas para la integración de aplicaciones empresariales que permitan mejorar aún más el bajo acoplamiento en la integración de unidades de software, para que cuanto menos dependientes fueran las partes que constituyen un sistema su mantenimiento representa un menor costo para la empresa.

1.2 Problemática

Hoy en día es indispensable la confiabilidad de los datos e información producida por los sistemas de información gracias a que los usuarios toman decisiones y acciones que afectan directamente los beneficios económicos de su negocio. En este contexto los errores en el software pueden causar pérdida de la información y esto trasciende en las decisiones que se tomen en la organización. Una problemática asociada a esto es la integración con sistemas adquiridos a terceros, es decir, la integración de aplicaciones empresariales [1]. La EAI fundamentalmente consiste en el intercambio de información y procesos de negocio entre diferentes aplicaciones y fuentes de datos en la empresa.

En la actualidad existen importantes desafíos en lo que respecta a la EAI, entre ellos se encuentra la heterogeneidad entre las aplicaciones de software que deben interactuar, por ejemplo: una compañía petrolera, en la que sus procesos de negocio van desde el descubrimiento de nuevas fuentes de hidrocarburos hasta su explotación, pasando por varios procesos intermedios. Los procesos son soportados por diferentes aplicaciones, en un ambiente heterogéneo con distintos proveedores, plataformas y productos. La problemática en este ejemplo se presenta cuando se necesita un reporte completo de un pozo petrolero en una ubicación geográfica particular, la información necesaria podría estar diseminada entre una decena de aplicaciones distintas, ¿cómo se puede obtener la información integrada? Un ejemplo más complejo es cuando se presenta una fusión bancaria, esto es cuando un banco internacional pasa por una fusión con otro banco de envergadura similar, por lo que algunos productos financieros son manejados por una aplicación perteneciente al banco adquirido y otros por aquella que tenía ya el banco comprador, sin embargo, se debe de consolidar ambas fuentes para poder calcular ganancias y pérdidas, ¿Se deberá volcar los reportes de ambas aplicaciones y consolidarlos manualmente? ¿Qué costo tendría eso? y ¿Cuál es la posibilidad e impacto de un error?

De tal forma que los avances en la tecnología de integración prometen nuevas formas de diseñar arquitecturas empresariales más ágiles y con mayor capacidad de respuesta que brindan el tipo de valor que el negocio ha estado buscando con lo cual el departamento de TI puede desarrollar

capacidades comerciales de manera rápida, económica y con un proceso estandarizado, es decir, en un vocabulario que la empresa pueda entender. Esto ha permitido el surgimiento de arquitecturas como la Arquitectura Orientadas a Servicios (SOA, por sus siglas en inglés), en la cual la tecnología debe expresarse como una parte del negocio y no como una aplicación arcana. Los Directores Ejecutivos de la Información (CIO, por sus siglas en inglés) estiman que pueden tener alrededor de 200 servicios en un repositorio en sus redes computacionales internas comúnmente conocidas como INTRANETS, que van desde operaciones como la verificación de crédito hasta el registro de clientes. Los empresarios pueden solicitar un servicio en un idioma que puedan entender y el departamento de TI puede vincularlo rápidamente con otros servicios para formar un flujo de trabajo o, si es necesario, crear una nueva aplicación. En este sentido, tecnologías como capas medias (*Middleware*) la cuál establece una lógica de intercambio de información entre aplicaciones, la computación distribuida (*Grid Computing)* que permite compartir recursos, servicios web (web *services*), gestión de procesos empresariales (*Business Process Management*) y herramientas de arquitectura empresarial están madurando hasta el punto en que los arquitectos de software pueden tener un impacto estratégico real en el negocio y en el departamento de TI [7].

En este trabajo, se propuso como meta la definición de una arquitectura escalable que permita una mejor adaptación en las integraciones de unidades de software con fácil mantenimiento, de tal forma que propicie la reducción de costo y tiempo en la integración mediante el efecto del bajo acoplamiento. Para lograrlo, se analizó lo que es un Modelo Canónico de Datos (CDM, por sus siglas en inglés), el cual según la Corporación Internacional de Máquinas Comerciales (IBM, por sus siglas en inglés) es un modelo bien definido y estructura la información en una organización, el objetivo no solo es limitarse a modelar los datos en base de datos, sino que sirve como referencia para todas las entidades y sus relaciones mediante todas las bases de datos que existan en la empresa y todas las aplicaciones heredadas que aporten a la iniciativa [8]. Este modelo se extendió con Mensajes Agnósticos, que son estructuras digitales representando una entidad o proceso no conocido independientemente si son utilizados en tiempo de diseño o de ejecución. La utilización de estos mensajes permite modelar los datos dentro de una sola base de datos y servir de referencia para todas las entidades y sus relaciones, así como representar la información común producida y consumida por aplicaciones [8] de una forma simplificada usando estructuras de tipo mapa de colecciones *(collection map)*, las cuales según Oracle son objetos que mapean claves (*key)* a valores (*values)* que no puede contener claves duplicadas [9]. De esta forma, en combinación con el patrón *Composite*, el cual consiste en crear una estructura del mensaje a partir de estructuras sencillas (*envelope*, *payload* y *entity fields*) para las colecciones Map, que contienen los datos de las entidades que representan los esquemas de base de datos, lo cual permitió crear el efecto agnóstico. Asimismo, se creó un componente de ayuda para la interpretación del mensaje y la transformación

de los datos usando el patrón *Strategy*, utilizado para indicar cómo se debe realizar el intercambio de mensajes entre diferentes objetos para resolver una tarea.

1.3 Objetivo de la tesis

El trabajo de investigación que presenta esta tesis tuvo como objetivo realizar una contribución a la ingeniería de software, particularmente en el área de la EAI a través de la conceptualización de una arquitectura escalable basada en un modelo canónico de datos que permita: 1) mejor adaptación en las integraciones de unidades de software (plataformas, aplicaciones y todo sistema software que puede ser integrado con otro) con fácil mantenimiento, 2) posibilitar la reducción de costo y tiempo en la integración mediante el efecto del bajo acoplamiento y 3) utilizar mensajes agnósticos para modelar los datos dentro de una sola base de datos, y servir de referencia para todas las entidades y sus relaciones para representar la información común producida y consumida por aplicaciones de una forma simplificada.

Los usuarios directos de esta propuesta son arquitectos y desarrolladores de software quienes se encargan de analizar, proponer e implementar soluciones de integración. De esta forma obtendrán un beneficio al permitirles el uso de una arquitectura que les brinde, el tan buscado, bajo acoplamiento entre arquitectos tecnológicos al proponer soluciones de integración.

1.4 Estructura del documento

En el capítulo II titulado Marco Teórico, se abordan los conceptos comprendidos dentro del área de integración de aplicaciones empresariales necesarios para el desarrollo de la investigación presentada en esta tesis.

El capítulo III titulado Estado del Arte, presenta la revisión bibliográfica de la integración de aplicaciones empresariales realizada mediante un mapeo sistemático de la literatura. El cual describe una serie de investigaciones de casos reales donde se abordaron temas de integración y las propuestas de solución a las mismas, con el propósito de conocer las tecnologías implementadas en cada una de ellas y sus deficiencias al momento de llevar a cabo la investigación descrita en la tesis.

El capítulo IV titulado Arquitectura Modelo Canónico de Datos Dinámico mediante Mensajes Agnósticos, presenta la propuesta de la arquitectura, así como su funcionamiento e implementación de los principales componentes.

El capítulo V presenta la conclusión y consideraciones finales de esta investigación. La tesis se complementa con un glosario de términos, referencias bibliográficas y una serie de anexos entre los cuales se especifica la arquitectura de integración definida en esta tesis.

Capítulo II
Marco Teórico

Este capítulo comprende el marco conceptual utilizado en el área de Integración de Aplicaciones Empresariales. Se presentan los conceptos de unidades de software, modelo de datos y su clasificación, arquitectura orientada a servicios,integración de aplicaciones empresariales, bajo acoplamiento y tipos de acoplamiento, modelo canónico de datos, tecnología agnóstica, mensajes digitales, patrones de diseño, patrón arquitectónico *Composite*, patrón arquitectónico *Strategy* y el componente MAP, que sirven de base para la tesis. Este capítulo concluye con la motivación que dio origen a la arquitectura propuesta.

2.1 Unidades de software

Las empresas están típicamente compuestas de cientos de aplicaciones que son hechas en casa (desarrollo *in house*), adquiridas a terceros, sistemas legados o una combinación de todos ellos, operando en múltiples capas en diferentes sistemas operativos. Los desafíos en la integración no son un trabajo fácil, los verdaderos desafíos están compuestos de una gran cantidad de asuntos comerciales y técnicos [10]. De acuerdo con la definición presentada en [11], una unidad de software es un componente modular con interfaces que nos ayuda a publicar y/o solicitar un grupo de servicios o procedimientos. Incluso puede ser una pieza que realiza algún cometido, una función, un método, una clase, una librería de funciones, una aplicación, un componente, entre otros. Estas suelen interactuar con colecciones de datos para guardar, actualizar, eliminar y presentar información.

2.2 Modelo de datos

Es una colección de herramientas conceptuales para describir a estos, las relaciones, la semántica y las restricciones de consistencia. En este sentido, se encuentra la definición presentada en [12], la cual explica que es un conjunto de conceptos y reglas que permiten estructurar los datos resultantes de la observación de la realidad, de forma que pueden ser representadas todas sus propiedades, tanto estáticas como dinámicas. Esencialmente, es una notación para describir información acerca de ellos [13]. La definición más aceptada señala que es una representación,

usualmente gráfica. Se dividen en dos tipos, los modelos de alto nivel, o conceptuales, los cuales utilizan conceptos muy cercanos a la forma en que los usuarios perciben los datos y los de bajo nivel o físicos, que describen la forma en cómo se almacenan en los dispositivos físicamente [14].

2.2.1 Clasificación de los modelos de datos

Los modelos de datos, acorde con [14] se clasifican en tres grupos:

a) Modelos lógicos basados en objetos

Son usados para describir datos a nivel conceptual. Se caracterizan porque proporcionan capacidad de estructuración flexible y permiten especificar restricciones en los datos explícitamente. Los más conocidos son:

- Entidad-Relación, consiste en un modelado de datos de un sistema de información, donde se expresan sus entidades relevantes, así como sus interrelaciones y propiedades [14].

- Orientado a objetos, este representa los datos mediante objetos, que contiene variables y métodos. La manipulación se lleva a efecto por medio de mensajes, al igual que el modelo E-R (Entidad-Relación), está basado en una colección de estos, sus valores están almacenados en instancias dentro de sí mismos, en un nivel de anidamiento de profundidad arbitraria [14].

- Binario, consiste en el uso conceptual de relaciones binarias entre conjuntos de datos, es importante para el desarrollo de bases de datos que utilizan el modelo de red [14].

- Semántico de datos, está basado en modelos de redes semánticas con bases en la inteligencia artificial, permiten captar mejor el significado de estos. Un objeto semántico representa cosas que son identificables en el ambiente de trabajo de los usuarios, de manera formal, este es un conjunto de atributos que describen adecuadamente una identidad bien determinada [14].

- Funcional de datos, son usados bajo una función matemática como el elemento de modelado, una solicitud de información se puede acceder y visualizar mediante una llamada bajo ciertos argumentos que devuelve la información solicitada [14].

b) Modelos lógicos basados en registros

Usualmente describen los datos bajo los niveles conceptuales y físicos, además permiten especificar la estructura lógica general y nos dan una descripción a un nivel más alto en la implementación. Es llamado de esta forma debido a que la base de datos está estructurada en registros en un formato fijo con diferentes tipos. Cada registro tiene un conjunto de campos fijos, también son de tamaño bien definido. El uso de este tipo de registros nos ayuda a la implementación del nivel físico. Los más conocidos son:

- Modelo relacional, tiene su base en la lógica de predicado y en la teoría de conjuntos, es el más usado en la actualidad para modelar problemas reales y administrar datos de forma dinámica, la manera y el lugar en que se almacenan estos no tienen relevancia (respecto a otros modelos como el jerárquico y el de red). Teniendo una gran ventaja de que se vuelve más fácil de entender y de ser usados por los usuarios, los datos pueden ser recuperados o almacenados mediante "consultas" que nos brindan una amplia flexibilidad y poder para administrar la información, el lenguaje más común para construir las consultas a bases de datos relacionales es *SQL* (*Structured Query Language*) o lenguaje estructurado de consultas, implementado como un estándar por los motores más importantes de sistemas de gestión de bases de datos relacionales [14].

- Modelo de red, basado en el modelo relacional. Actualmente es usado muy poco debido a que hace compleja la tarea del modelado de datos, sin embargo, el código de este se sigue implementado en bases de datos antiguas. La manera de representar los datos es mediante colecciones de registros y sus relaciones se representan por medio de ligas o enlaces, que pueden verse como apuntadores de registros, y se organizan en un conjunto de gráficas arbitrarias [14].

- Modelo jerárquico, de la misma forma que el modelo de red, no es muy usado para estos propósitos, es muy parecido a los modelos de entidad relación en cuanto a las relaciones y datos por lo que es un antecedente al relacional, son representados mediante registros y sus ligas, la diferencia entre el de red radica en que están organizados por conjuntos de árboles en lugar de gráficas arbitrarias, el principal problema de los sistemas de bases de datos jerárquicos es la poca independencia de los programas respecto a cómo están almacenados los datos, lo que dificulta además la programación de software de acceso a estos sistemas [14].

c) Modelos físicos de datos

Son usados para describir datos en el nivel más bajo, generalmente no son muy utilizados en la actualidad, siendo estos el modelo unificador y el de memoria de elementos. Muchos autores lo describen como modelos de datos primitivos que son de interés principalmente para los fabricantes de Sistemas Manejadores de Bases de Datos (DBMS, por sus siglas en inglés).

2.3 Arquitectura orientada a servicios

La arquitectura orientada a servicios (SOA, por sus siglas en inglés) es una arquitectura bajo un modelo de diseño de software con un concepto basado en la de aplicaciones de encapsulación lógica dentro de los servicios que interactúan mediante un protocolo de comunicación común. Se considera un paradigma tecnológico ampliamente difundido, que a través de los años ha apoyado a múltiples organizaciones en el logro de objetivos estratégicos. Está conformada por componentes disponibles a través de interfaces genéricas y protocolos estandarizados y preferentemente libres de licenciamiento, diseñados con el menor nivel de dependencia posible con las unidades de software que los consume y del contexto técnico del desarrollo, impulsando su reutilización y aprovechamiento. Básicamente, se trata de un conjunto de servicios cohesivos, donde cada uno es relativamente fácil de construir o reemplazar si es necesario. Al ser independientes, nos permite adaptarlos a los cambios de formas más sencillas que las arquitecturas tradicionales. En este sentido, SOA es un patrón arquitectónico en el diseño de software en el que los componentes de aplicaciones proporcionan funcionalidad a otros componentes a través del protocolo de comunicación. Cuando estos protocolos son utilizados para establecer esta forma de comunicación, representan una implementación basada en este paradigma. La arquitectura resultante establece un diseño dentro del cual los servicios web son un elemento clave, esto significa que, al migrar su aplicación a esta solución, se está comprometiendo con sus principios de diseño y las tecnologías que lo acompañan como partes centrales de su entorno tecnológico. Esta plataforma de integración se funda en capas de tecnología XML (*eXtensible Markup Language*) establecidas, con el fin de exponer la lógica de aplicación existente, también promueve el uso de un mecanismo de descubrimiento a través de un agente de este [15]. Separa las funciones en unidades o pedazos de software distintos, a los que los desarrolladores hacen accesibles a través de una red para permitir que los usuarios los combinen y reutilicen en la producción de aplicaciones. Se encuentra dentro de los principales estilos de arquitecturas de software conocidos hasta el momento. El principal aporte es la abstracción de los procesos, por la que estos se desacoplan de las soluciones y se exponen directamente al negocio. Cabe destacar que no hay estándares en relación con la composición exacta, aunque varias fuentes de la industria han publicado sus propios principios [16]. Algunos de estos principios son los siguientes:

- Contrato de servicios estandarizados. En este tipo de contratos los servicios están inscritos a un acuerdo de comunicación, el cual está definido en conjunto con uno o más documentos de lenguaje descripción de servicios web (WSDL, por sus siglas en inglés).

- Acoplamiento débil de sistemas: los servicios mantienen una relación de bajo acoplamiento que minimiza las dependencias y sólo se requiere que se mantenga un conocimiento de uno al otro.

- Abstracción: más allá de las descripciones del contrato, los servicios encapsulan y ocultan la lógica de negocio de los demás aplicativos.

- Reutilización de servicios: las funciones son divididas en pequeños bloques o servicios con el objetivo de promover la reutilización.
- Autonomía: los servicios mantienen el control sobre la lógica que es encapsulada, desde el punto de diseño y ejecución.

- Servicios sin-estado: permite la disminución del uso de recursos para la gestión de la información de estado cuando sea necesario.

- Descubrimiento de servicios: esta arquitectura es complementada con el uso de metadatos con el fin de descubrir en una red las interfaces publicadas además de ser interpretadas.

- Composición: los servicios deben de estar constituidos por partes eficazmente diseñadas, optimizadas, independientemente del tamaño y la complejidad de la composición.

- Granularidad: una consideración de diseño de los servicios proporciona un ámbito óptimo y un correcto nivel granular, es decir, tamaño de cada tarea que refleja la funcionalidad del negocio en una operación de servicio y su independencia con las demás tareas.

- La normalización: mediante la descomposición de servicios a un nivel de forma normal la normalización ayuda a minimizar la redundancia. En algunos casos, se desnormalizan para fines específicos, como la optimización del rendimiento, el acceso y agregación.

- Relevancia: la funcionalidad se presenta en un nivel de granularidad reconocido por el usuario como un servicio significativo.

- Encapsulación: muchos de los servicios están unificados o destinados para ser usados mediante SOA. Generalmente, debido a plataformas legadas estos no fueron planificados para estar en esta condición.

- Transparencia de ubicación: indica la capacidad de un aplicativo consumidor para llamar a un servicio independientemente de su ubicación en la red. Esto también reconoce la propiedad de descubrimiento, la cual es uno de los principios fundamentales de la arquitectura orientada a servicios y el derecho de un consumidor para acceder a estos. Generalmente, la idea de la virtualización también se refiere a la transparencia de ubicación. Sin importar la invocación del servicio lógico, por lo que SOA habilita la ejecución del componente de la infraestructura, normalmente se implementa mediante bus de datos, que lo mapea y llama al ente físico.

SOA otorga la posibilidad de convertir las tecnologías en auténticos habilitadores de negocio. Esto se debe a que permite alinear y acercar las áreas de tecnología y negocio (los objetivos de negocio o metas organizacionales) mediante el desarrollo de aplicaciones manejables y más seguras, gracias a que proporciona una infraestructura y documentación común para desarrollar servicios con la posibilidad de añadir nuevas funcionalidades. Una gran ventaja es que debido a la seguridad que brinda permite minimizar la pérdida de datos. En el aspecto de desarrollo, se pueden llevar a cabo las soluciones en menor tiempo y más económicas gracias a la integración de todos los datos de manera flexible, con lo cual mejora significativamente la agilidad y flexibilidad de las empresas. La principal ventaja es que comunica sistemas y plataformas antiguos con tecnología moderna, exaltando al máximo el tiempo de vida de los componentes, lo que permite optimizar recursos y acceder a la información actualizada con una rapidez considerable. A continuación, en la Figura 1 se muestran los elementos de una arquitectura SOA, la cual se divide en cuatro componentes principales. El primer componente, llamado Aplicación *Frontend*, sirve para que los servicios puedan ser consumidos por los clientes o por otras aplicaciones propietarios del proceso de negocio. Si bien el *frontend* de la aplicación es el propietario del proceso del negocio, los servicios proporcionan la funcionalidad del negocio que los *frontends* de las aplicaciones y otros servicios pueden utilizar. Un *frontend* de una aplicación puede ser una interfaz gráfica de usuario, como por ejemplo una aplicación Web que interactúa directamente con los usuarios. Los programas por lotes o los procesos de larga duración que invocan a las funcionalidades de forma periódica o como resultado de eventos específicos son también ejemplos válidos de *frontends*.

En última instancia, siempre es un *frontend* de una aplicación quién inicia un proceso de negocio y recibe los resultados. Los *frontends* de aplicaciones son similares a las capas de nivel más alto en las arquitecturas multi-capa tradicionales.El segundo componente denominado Servicios, es un componente software con un significado funcional que lo distinque de otros, y que típicamente encapsula un concepto de negocio de alto nivel. Además, puede ser una implementación que provee la lógica de negocio y de datos mediante un contrato de servicio WSDL o ruta donde son localizados en una red. El repositorio de servicios permite encontrar servicios y brinda la información necesaria para utilizarlos. Si bien mucha de la información requerida forma parte del contrato del servicio, el repositorio de servicio puede proporcionar información extra, por ejemplo la localización física e información sobre el proveedor y cuestiones de seguridad [17]. El cuarto componente es el llamado bus de servicios (ESB por sus siglas en inglés). El ESB conecta a los componentes mencionados (*frontend*, servicios y repositorio de servicios). El bus de servicios no necesariamente debe estar formado por una única tecnología, sino que puede comprender varios productos y conceptos.

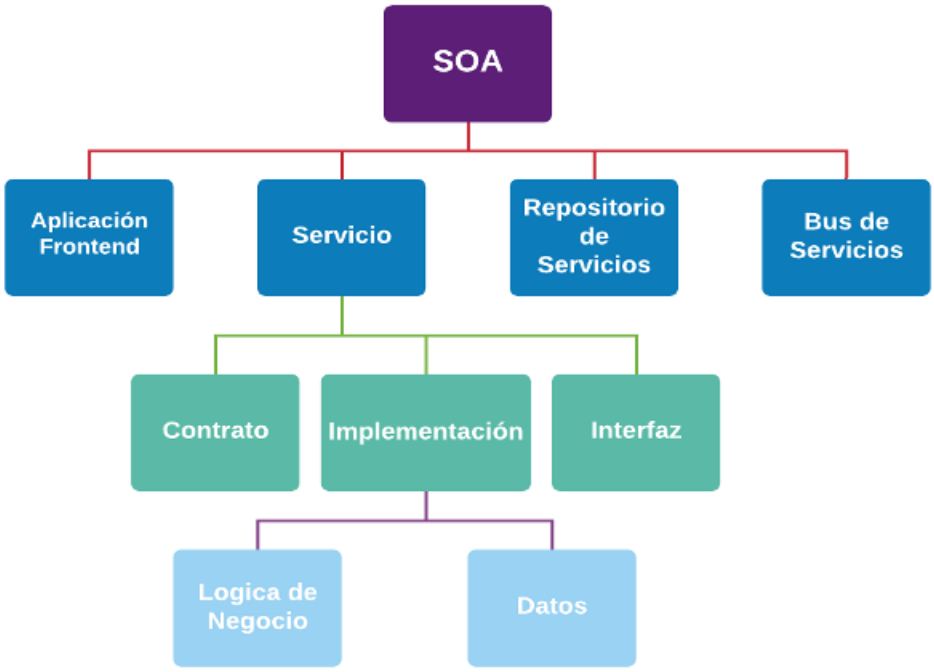


Figura 1. Elementos de una arquitectura SOA.

En la Figura 2 se observan los componentes de servicio de funciones de SOA. Se dividen en componentes para funcionalidad y componentes para calidad. De lado izquierdo se encuentran los

componentes de funcionalidad, estos son transporte, protocolo, descripción del servicio, servicio, proceso de negocio y registro, a continuación, se describen cada uno de ellos.

1. Transporte: es el mecanismo usado para trasladar las peticiones desde el cliente, hasta el proveedor del servicio y viceversa.
2. Protocolo de comunicación: es el medio estandarizado de comunicación entre las unidades de software.
3. Descripción del servicio: representado por un esquema usado para describir a este, indicando de cómo se le puede invocar y cuáles son los datos necesarios para realizar su invocación.
4. Servicio: se define la implementación de las interfaces contenidas en este.
5. Proceso de negocio: son los bloques o conjunto de servicios, invocados en una determinada secuencia, con un grupo particular de reglas para satisfacer un requisito de negocio.
6. Registro: el medio físico o repositorio usado por los proveedores, para publicar y que los clientes tengan donde buscarlos.

También, en la Figura 2 se encuentran los componentes de calidad de servicio de SOA, estos son: las políticas, la seguridad, transacción y gestión. Estos se detallan a continuación.

1. Políticas: representado por las reglas bajo las cuales, un proveedor hace que estos, estén disponible para los clientes.
2. Seguridad: es el conjunto de reglas que podrían ser aplicadas en la identificación, autorización y control de acceso a los servicios.
3. Transacción: atributos que generalmente son aplicados sobre un grupo de estos para devolver datos consistentes.
4. Gestión: conjunto de atributos que son aplicados al grupo de servicios para ser administrados.

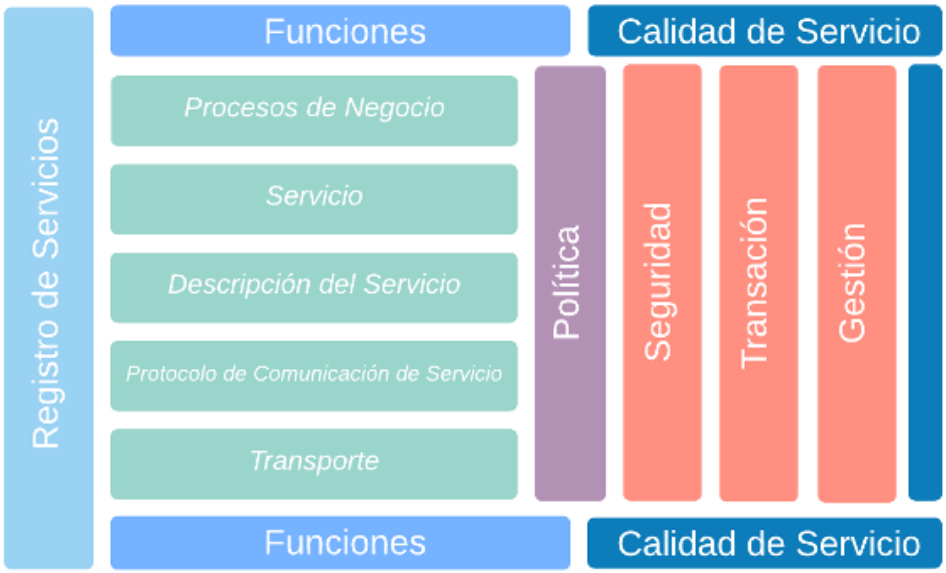


Figura 2. Componentes de SOA.

Finalmente, es importante no confundir servicios web con SOA. Los primeros son una colección de tecnologías que incluyen XML, SOAP (*Simple Object Access Protocol*), WSDL *y* UDDI (*Universal Description, Discovery and Integratio*n); los cuales permiten construir soluciones de programación para mensajes específicos y para problemas de integración de aplicaciones. SOA es una arquitectura, es más que un conjunto particular de tecnologías, dentro de la cual todas las funciones están definidas como servicios independientes con interfaces invocables bien definidos, las cuales pueden ser llamados en secuencias para formar los procesos de negocio. Las funciones son definidas como servicios, que incluye funciones de negocio, transacciones de negocio compuestas de funciones de bajo nivel y de sistema. Además, los servicios son independientes, lo cual quiere decir que actúan como cajas negras, en este sentido, se entiende que no saben cómo realizan la tarea, pero devuelven el resultado esperado. Incluso las interfaces que sean invocables representan que a nivel arquitectura es irrelevante, si son locales o remotos, no importa el sistema de conexión o el protocolo que se utilice para la invocación, o la estructura de componentes que usan para la conexión, en SOA la clave está en el interfaz, la cual define los parámetros requeridos y la naturaleza del resultado, lo que significa que define la naturaleza del servicio y no la tecnología utilizada. Gracias a esto, permite realizar dos puntos críticos: ser independientes y ser manejados libremente y adaptados [18].

2.4 Integración de aplicaciones empresariales

La EAI implica la unión de plataformas y fuentes de datos empresariales para que puedan compartir fácilmente procesos y datos de negocios. La integración debe implementarse sin requerir cambios significativos en las soluciones existentes. Antes de EAI, en un entorno corporativo, integrar aplicaciones de distintos proveedores era una propuesta costosa y que podría ocasionar el surgimiento de islas de información (consultar Capítulo 1.1). Las empresas intentaban combinar software que a menudo se ejecutaban en diferentes plataformas de *hardware* y no tenían protocolos para comunicarse con otras aplicaciones fuera de su propio ámbito o dominio estrechamente definido, en cierto sentido, las empresas tenían islas de funciones y datos comerciales, cada una de ellas existía en su propio contexto separado del problema. Lo cual representa un enfoque diferente a este desafío, define la semántica para la aplicación y la integración de datos, es decir, una metodología o enfoque estándar para que estos se comuniquen. Al soportar este estándar, las plataformas pueden comunicarse fácilmente entre sí. Los módulos de aplicaciones de la integración son como un DBMS y pueden cambiar, pero debido a esta metodología común, un módulo de repuesto se puede conectar y la comunicación puede continuar ininterrumpidamente [19].

EAI es el uso de unidades de software para conectar un conjunto de aplicaciones empresariales, normalmente de distinto proveedor. En este sentido, se considera cualquier mecanismo que permita realizar tal conectividad, desde archivos planos hasta servicios de mensajería. Por otro lado, una aplicación empresarial es cualquier solución que brinda servicios a la empresa, sea una propia o del mercado en el cual opera (petróleo, banca, etc.) hasta aquellas aplicaciones administrativas típicas tales como los sistemas para la administración de relaciones con el cliente (CRM, por sus siglas en inglés) o los sistemas de planificación de recursos empresariales (ERP, por sus siglas en inglés). La forma básica de llevar a cabo la conectividad es el intercambio de datos entre dos plataformas a través de algún medio o herramienta. Es común en una empresa, por ejemplo, la existencia de interfaces consistentes en archivos planos depositados temporalmente por un proceso en un directorio compartido, que luego son tomados por otro proceso, otra forma de realizar la conectividad es a través de bases de datos accedidas por diferentes sistemas. Finalmente, hoy en día se hace uso de servicios de mensajería, desde *ad-hoc* hasta basados en estándares de la industria.

Cabe destacar que el concepto de EAI no es para nada nuevo, prácticamente existe desde el momento en que se comenzó a construir sistemas software. Su fundamento radica en el hecho de que una empresa es un ecosistema de aplicaciones de diferente naturaleza las cuales necesitan forzosamente comunicarse entre sí, pero que no fueron creadas para ser escalables en cuanto a comunicación se refiere. La necesidad de integración es estratégica, de tal forma que siempre estará presente donde se requiera la intercomunicación de dos o más plataformas. En este sentido, no se

debe considerar como un problema actual debido a que es heredado de las plataformas legadas (*legacy*) que no tuvieron en cuenta el operar en conjunto, por lo que se considera una realidad de siempre. Es decir, siempre existirán desarrollos software diferentes entre sí en las empresas, cuyas funciones siempre serán distintas porque muchos de ellos fueron diseñados para ser utilizados en forma aislada, comúnmente para un departamento o pequeño grupo de usuarios. Generalmente, están preparados para soportar la seguridad y escalabilidad requeridas para hacerlos disponibles hacia otros usuarios y sistemas.

Existen problemas del estilo organizacional o de administración, entre los cuáles resaltan la integración no planificada, la cual es el resultado de crear interfaces (código fuente en programación orientada a objetos con el cual especificamos qué métodos deben ser implementados por una clase, sin tener que definir cómo son manipulados) para resolver la necesidad inmediata. Esto ocasiona acoplamientos indeseables, que se suman a la pila de sistemas legados (*legacy)* poco mantenibles. Es decir, se resuelve un problema y se crea otro. Además, se agrega la carencia de un análisis de impacto en la forma de trabajar de los usuarios, por ejemplo: *una vez que se llevó a cabo la integración de A y B, entendiendo que ambos pueden ser sistemas o módulos de sistemas... ¿cómo se ve afectado el trabajo de los usuarios de A y B?, ¿cuáles módulos o funciones del sistema A se ven afectadas con la integración del sistema B y viceversa?, es decir, ¿cuál es el impacto de la integración?*

Lo anterior son manifiestos de los siguientes problemas de fondo, aún más graves:

- No disponer de una administración de aplicaciones integradas. Lo que dificulta el poder detectar si una nueva integración no está causando un acoplamiento indeseable, o si se está integrando aplicaciones en forma consistente en diferentes sectores de la empresa.

- Relación entre aplicaciones y procesos de negocio no definida, poco clara o desordenada. Esto impide discernir o conocer qué desarrollos son responsables de cada proceso o flujo de información y, por lo tanto, impide discernir la forma de integración más adecuada para el negocio.

Mientras que los problemas técnicos son generalmente visibles, los problemas organizacionales no son tan evidentes, por lo tanto, son mucho más riesgosos, difíciles y a su vez costosos de solucionar. Por otro lado, la mayoría de los proyectos de EAI son conducidos por expertos para resolver el aspecto técnico de la integración, pero normalmente carecen de la experiencia en resolución de los conflictos que se puedan presentar con los objetivos organizacionales. Finalmente, es importante priorizar dichas dificultades en términos de los riesgos que las mismas pueden representar para la

organización. Un desafío técnico con normalidad puede solucionarse, regularmente con un valor adicional. Sin embargo, una integración (posiblemente con un costo elevado) que no fue pensada claramente desde el punto de vista del negocio, puede no acarrear beneficios para el mismo, redundando en un gasto necesario con algunos problemas adicionales. Dejando a un lado tales problemas, es claro que un proyecto de EAI es tanto informático como organizacional, debido a que tanto las necesidades como las dificultades están de ambos lados.

2.4.1. Clasificación de la EAI

Es importante destacar que no existe una forma única de integración adecuada a todas las situaciones, por el contrario, a lo largo de los años se han desarrollado diferentes formas de estas, adecuadas a diferentes necesidades. Si bien no hay una taxonomía estándar, los siguientes son tipos generalmente aceptados en la industria hoy en día:

1. Integración orientada a la información, consiste en el pasaje de información de un sistema a otro. En la Figura 3 se observa cómo viaja la información entre los repositorios o bases de datos en para sun integración. Posteriormente, esto permite que puedan ser accedidos por las diferentes aplicaciones del ecosistema (administración de clientes, fuerza de venta y ERP) que intervienen en su consumo. Casos típicos: el envío de transacciones comerciales en las cuales las bases de datos están integradas y sincronizadas.

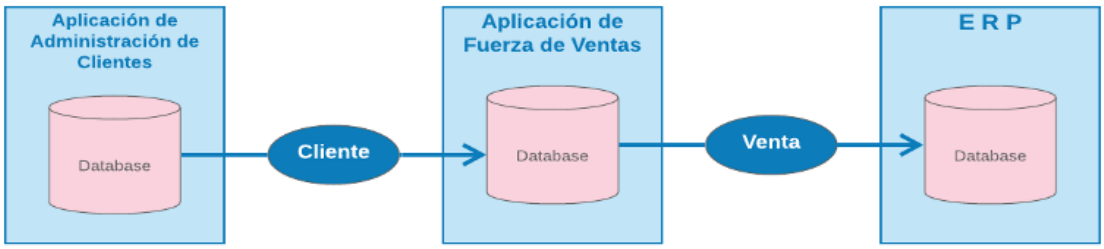


Figura 3. Integración orientada a la información.

Lo anterior, generalmente no requiere modificaciones en las aplicaciones integradas, sino solamente la implementación del mecanismo de envío de información entre los repositorios de datos de las plataformas participantes, esto hace que sea un mecanismo de sistemas más simples y de menor impacto. Dicha integración consiste en la transmisión de datos entre los repositorios de las plataformas, no necesariamente esto implica que esta se realice pura y exclusivamente usando tecnología de base de datos. También podría realizarse mediante archivos planos, interfaces de programación de aplicaciones, (API 's, por sus siglas en inglés) o incluso servicios de mensajería.

La clave de esta actividad no está en el medio técnico, sino en el hecho de que lo que se integra es información y no procesos o servicios.

2. Integración orientada a procesos, llegando a un paso o nivel de abstracción más elevado, nos encontramos con la integración orientada a los procesos de negocio. En la Figura 4 se muestra un flujo de trabajo (proceso de venta) en un ecosistema formado por diversas plataformas. Se aprecia la orquestación de la integración y el consumo de los datos. La integración se lleva a cabo mediante la automatización de los diferentes pasos de un proceso de negocio a través de una o más aplicaciones. A menudo esto implica un intercambio de información entre sistemas para lograr el objetivo en cuestión. Además de información, también se integra el control. Generalmente, esto se lleva a cabo mediante un flujo de trabajo (*workflow*).

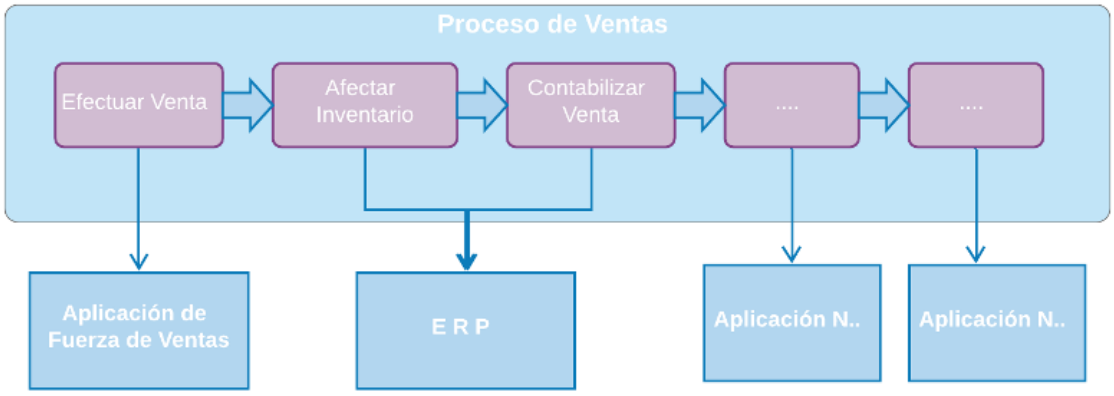


Figura 4. Integración orientada a los procesos de negocio.

3. Integración Orientada a Servicios, en este paradigma de integración una aplicación expone una serie de servicios de negocio que pueden ser usados por otras plataformas. Se busca no solamente el reúso, sino también el hacer que cierta lógica de negocio sea implementada una única vez en la empresa y sea reutilizada por otras aplicaciones. También permite la creación de las llamadas aplicaciones compuestas, las cuales surgen a partir de la unificación de diferentes servicios preexistentes en la organización (ver Figura 5), la solución de *Trading* (compra venta) podría ser una de ellas, si toda su lógica partiera de invocación a componentes de otras aplicaciones. Una aplicación de *Trading* es una aplicación de compra venta de activos. Es importante mencionar que la orientación a servicios y la orientación a procesos no son excluyentes, sino complementarias. Esto se debe a que la aplicación (arquitectura) está orientada a servicios, pero, como se observa en la Figura 5, los servicios son orientados a los procesos de compra-venta. La integración se realiza con servicios. Incluso, una integración implementada correctamente es aquella en la cual los procesos de negocio pueden integrarse a partir del llamado con un orden preestablecido de las

diferentes aplicaciones software que conforman la integración, aplicaciones que satisfacen las necesidades de los distintos subprocesos.

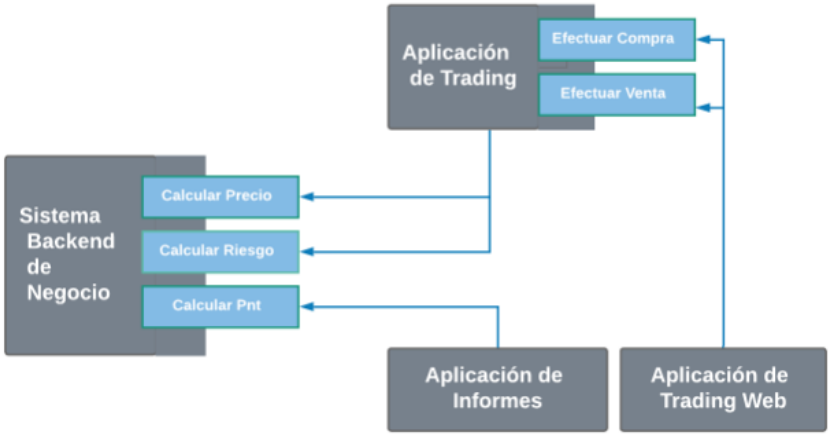


Figura 5. Integración orientada a los servicios.

4. Integración orientada a los portales, varios autores separan a esta integración como un tipo diferente. El aspecto y la forma distintiva es la agrupación de varias aplicaciones bajo una interfaz visual común, normalmente un portal web o *portlet* (es un componente modular de la interfaz de usuario de un portal web, particularmente un *portlet* puede integrar y personalizar contenido de diferentes fuentes dentro de una página web y generar contenido dinámico), un ejemplo se puede ver en la Figura 6 que muestra cómo un portal principal contiene una serie de accesos a las diferentes plataformas CRM, ERP, Aplicación N, etc. integrando en un solo punto todas las soluciones mediante un servidor de portal, para ser consumidas por los usuarios.

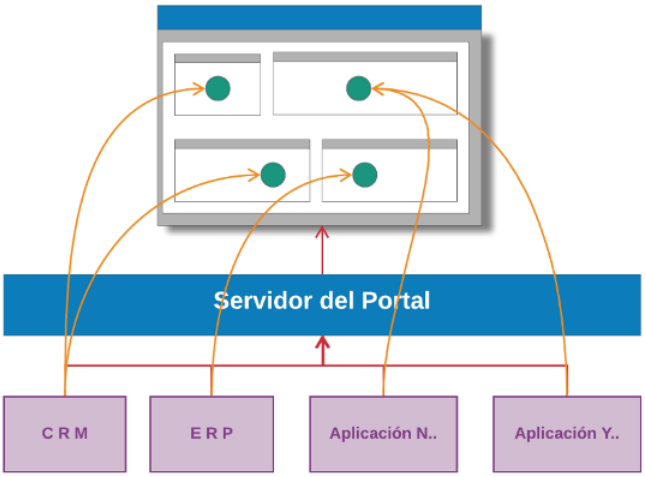


Figura 6. Integración orientada a portales.

Por otro lado, las aplicaciones no son excluyentes de los otros tipos de integración. El portal puede alimentarse a través de servicios y el mismo puede también soportar la participación de actores humanos en procesos de negocio.

2.4.2. Metodología para la integración de aplicaciones empresariales

Para llevar a efecto una metodología para la integración de aplicaciones empresariales es necesario conocer la situación y realidad de la necesidad de la empresa, esto incluye conocer los planes estratégicos de la organización. Además, es necesario esto para resolver correctamente los desafíos que enfrenta en la perspectiva de integración, sobre todo en la automatización de procesos que se requiere. Un enfoque cuidadoso y estructurado ayudará a evitar que la respuesta a necesidades tácticas de hoy entorpezca la integración estratégica que busquemos mañana. Para lograrlo, es necesario definir un mapa de ruta para la integración, utilizado para guiar los proyectos de EAI. Se divide en dos rutas a seguir, una ruta táctica y una estratégica [20].

La ruta táctica es aquella que es seguida para dar solución a las necesidades actuales del negocio, pero sin ir en detrimento de aquellas que podamos tener en el futuro a mediano y largo plazo. Se puede recurrir a dicha ruta cuando se tiene una necesidad de integración (integrar información, funcionalidad y/o procesos) a corto plazo que se debe suplir. A continuación, se presentan los pasos recomendados para tal propósito:

1. Identificar la necesidad a cubrir. Es poder discernir o conocer si debemos integrar información, funcionalidad y/o procesos. Para el caso de un proceso, deberá ser un proceso aislado, es decir que no dependa de otros procesos para su ejecución; por otro

lado, si fuera un caso contrario, raramente podrá ser tratado a nivel táctico y requerirá un enfoque estratégico. En caso de la integración orientada a portales, no muy frecuentemente puede ser tratado a nivel táctico, a no ser que se trate de un prototipo inicial restringido a un grupo limitado de usuarios [20].

2. Identificar el alcance e impacto organizacional. Es determinar las unidades organizacionales, procesos, aplicaciones y usuarios afectados. Una ruta táctica no debería ser, por lo general, muy extendida a nivel organizacional. En caso contrario, se recomienda nuevamente revisar si el camino no debería ser una ruta estratégica [20].

3. Identificar concesiones para en el momento y acciones para el futuro. Debido a que la naturaleza táctica de la solución que se requiere puede llevarnos a contemplar diversas concesiones, como ejemplo, el realizar una integración que como resultado en el futuro se derive en un acoplamiento estrecho no del todo deseable, esto ocurre a nivel de código fuente, cuando una clase dependiente contiene un puntero directamente a una clase concreta que proporciona el comportamiento requerido. La dependencia no puede ser sustituida sin requerir un cambio en la clase dependiente. Para tal caso, es muy importante identificar y conocer esas concesiones y establecer algunas reglas que permitan evitar que los riesgos subyacentes se incrementen con el tiempo. Un ejemplo de esto es poder acordar que un acoplamiento estrecho no deseable no será usado e implementado más allá del proyecto actual particular. Adicionalmente, se pueden definir acciones a futuro para eliminar dicho acoplamiento [20].

4. Definir el tipo de integración a utilizar. Es necesario definir el tipo de integración a utilizar en una ruta táctica, esto permitirá orientar el enfoque de la ruta a seguir. Generalmente, se realiza una integración de información o una de servicios [20].

5. Seleccionar la tecnología de integración. Es la forma en que se va a implementar el tipo de integración previamente seleccionado. Entonces si la ruta es táctica, es conveniente que sea resuelta con alguna tecnología ya probada en la organización [20] para evitar contratiempos.

6. Planear la integración. Se deben de planificar todas las actividades a seguir durante la integración, es de suma importancia y de mucho cuidado planificar las integraciones, en especial aquellas de alto impacto considerando un buen análisis de riesgos [20].

Para la planificación de actividades con el uso de un diagrama de Gantt suele ser suficiente.

7. Planear la transición. ¿Qué se debe hacer para que la empresa continúe funcionando adecuadamente una vez implementadas las integraciones? Generalmente se establece un plan de capacitación para los usuarios involucrados en el uso de las aplicaciones afectadas. Además, se comienza a establecer un proceso a seguir para eventualmente dejar de utilizar las diversas plataformas empleadas en la empresa antes de la integración [20].

8. Implementar la integración. Llevar a cabo la implementación, es decir accionar cada actividad conforme a lo establecido en la planificación de la integración [20].

La ruta estratégica es aquella presentada por la organización que va más allá del planteamiento de un proyecto particular. Esta se deriva no sólo de las necesidades actuales, sino que va en función también de las necesidades a mediano y largo plazo. Es utilizada cuando la ruta está guiada para la optimización de procesos de negocio en forma integral y holística. A continuación de describen las acciones recomendadas:

1. Conocer y discernir la necesidad a cubrir. Un proyecto con una ruta estratégica generalmente estará asociado en la manera en que uno o más procesos de negocio son soportados mediante las aplicaciones, y el poder optimizar los mecanismos en que tales aplicaciones se relacionan para ser soportados [20].

2. Identificar el impacto organizacional. Conocer y determinar las unidades organizacionales, procesos, aplicaciones y usuarios afectados en la integración [20].

3. Identificar el mapa actual de procesos. Conocer e identificar los procesos y aplicaciones con el fin de establecer un gobierno y tener el control e inventario de componente tecnológicos, es muy probable que, aunque se conozca qué procesos se desea automatizar, los detalles de estos no sean completamente conocidos, al no tener esta parte establecida no se tendrá tampoco la forma en que se mapean a los sistemas. Es importante, antes de iniciar una implementación de integración estratégica, documentar los procesos afectados, las unidades a las que pertenecen, los roles que participan y las aplicaciones requeridas para llevarlos a cabo. De otra manera, se dejarán de lado importantes integraciones con diversas unidades de software que impedirán trazar el mapa futuro [20].

4. Definir el mapa futuro de procesos y aplicaciones. Una vez detectado el mapa actual, es imperante determinar cómo este se verá afectado tras la integración. Puntos importantes para considerar son los procesos, las unidades de software que sufrirán modificaciones o serán retiradas, además de cómo los diferentes roles de la organización deberán cambiar su forma de trabajar en un aspecto u otro [20].

5. Establecer el plan de transición a seguir. Un proyecto estratégico raramente se ejecuta de una sola vez, dado su alcance y magnitud. El mapa futuro se alcanzará siguiendo una ruta consistente a través de varios proyectos que acercarán paulatinamente la situación actual a la futura [20].

6. Implementar el plan de transición. De acuerdo con lo referido en el punto 5, la implementación se logrará a través de varios proyectos de índole táctica [20].

La integración de aplicaciones empresariales va dirigida a toda la organización que se encuentre en fase de evolución y que presente desafíos con cualquiera de las situaciones siguientes:

- Fusiones y adquisiciones. Cuando se presenta este desafío en las organizaciones y que pueda ser exitosa, y se en este caso que se requiera la integración de procesos de negocio de dos o más compañías de tal modo que puedan trabajar como una sola corporación. EAI es la solución que nos permitirá una integración rápida y bajo estándares ya conocidos [20].

- Automatización de procesos de negocio. Por lo general muchas tareas manuales o la necesidad de nuevos productos y servicios que deben integrarse con las aplicaciones existentes, con lo cual se busca mejorar la eficiencia, los costos de operación y los servicios al cliente en toda la organización. La EAI proporciona el medio eficiente para llevar a cabo esta tarea [20].

- Comercio electrónico, para su implementación es importante el cumplir con las necesidades del B2B (*Business to Business*, por sus siglas en inglés) y B2C (*Business to Customer*, por sus siglas en inglés) como parte del trabajo de comercio electrónico, esto requiere la conexión de clientes, proveedores y socios en todo el mundo, de manera que formen una cadena de suministro y de valor integrada a través del Internet, donde el apalancamiento tecnológico deriva en la EAI [20].

Se concluye en torno a la integración bajo EAI que es de gran valor en la reducción del ciclo de vida de los procesos de negocio en un ecosistema heterogéneo con diferentes plataformas y tecnologías. Actualmente, en el entorno empresarial competitivo, la necesidad de alinear los sistemas con los objetivos de estos es una realidad. Es inevitable la evolución continua de la organización y requieren nuevos métodos y datos que a su vez necesitan la integración con los sistemas existentes, los cuales deben comenzar a operar rápidamente. La implementación de soluciones EAI hacen esto posible debido a que ayudan a integrar distintas aplicaciones permitiendo el cambio en las reglas de negocio en un tiempo menor [20]. Con lo cual es posible reducir el ciclo de vida de los procesos de negocio porque se llevarán a cabo entre aplicaciones integradas, no en un ambiente heterogéneo.

2.5 Bajo acoplamiento

El bajo acoplamiento se define como el nivel de interdependencia entre las unidades de software, se toma en cuenta la cercanía entre ellas y la fuerza que existe en la relación. Se considera de bajo acoplamiento el estado ideal en su relación, cuantas menos dependencias existan entre éstas, mejor será el diseño de la solución. Por ello, el objetivo final del diseño de aplicaciones es reducir al máximo el acoplamiento [21].

2.5.1 Tipos de acoplamiento

El nivel de acoplamiento se clasifica en los siguientes tipos:

1) **Unidades completamente desacopladas.** Dos unidades de software están completamente desacopladas cuando una no depende de la otra para hacer cada una su trabajo. Esto permite mover una de ellas y utilizarla sin necesidad de la otra [21].

2) **Acoplamiento común.** También conocido como acoplamiento global. Generalmente dos unidades de software comparten procedimientos o variables globales. La implicación que ocasiona al presentarse este acoplamiento es hacer un cambio en todos los componentes que lo usan [21].

3) **Acoplamiento de datos.** Este tipo de acoplamiento sucede cuando una unidad de software está acoplada a otra cuando entre ellos comparten datos en común (ejemplo, algún parámetro) que usan para funcionar [21].

4) **Acoplamiento de control.** Para este acoplamiento, un método está acoplado a otro por control cuando de alguna manera un método controla el flujo de otro, en general, esto sucede cuando se pasa algún parámetro en función de este, el segundo se comporta de una u otra manera de acorde al tipo de parámetro [21].

5) **Acoplamiento de mensajes.** Es la forma más eficaz de un bajo acoplamiento, se puede lograr por la descentralización de estados de los mensajes (como en los objetos). La forma de comunicarse de las aplicaciones es mediante el paso de mensajes entre los componentes [21].

6) **Acoplamiento de contenido.** Es conocido también como acoplamiento patológico, se produce cuando un módulo está en función de otro al momento de apoyarse en el funcionamiento interno de uno respecto al otro, podría ser cuando se accede a datos locales de otro módulo esto conducirá a cambiar el módulo dependiente [21].

7) **Acoplamiento externo.** El acoplamiento externo se presenta cuando dos módulos comparten un formato de datos impuesto externamente, también el protocolo de comunicación y/o interfaz de la aplicación. Está relacionado con la comunicación a herramientas externas y dispositivos [21].

2.6 Modelo canónico de datos

Un modelo canónico de datos (MCD, por sus siglas en inglés) surge para solucionar la integración estrechamente acoplada (consultar sección 2.4.1 de este capítulo), en una invocación a un servicio web, el consumidor y el proveedor deben acordar los formatos del mensaje. Cuando los equipos de desarrollo independientemente diseñan las dos partes es posible que se presente la dificultad de encontrar un acuerdo sobre los esquemas comunes a utilizar. Si se repite esto por docenas de aplicaciones con un servicio y un componente de software típico que utilice todos ellos se podrá ver cómo los formatos de mensajes de negociación simple pueden convertirse en una tarea de tiempo completo con un costo de mantenimiento muy alto. Un enfoque común para evitar situaciones similares a la descrita consiste en utilizar un MCD. El cual es un conjunto de esquemas que es independiente de cualquier unidad de software y lo comparten todos los demás. De esta manera las aplicaciones no tienen que acordar en los formatos, simplemente pueden utilizar los existentes [8]. Para lograr implementar este modelo se necesita un acuerdo en torno al resto de las firmas de los esquemas que son independientes desde cualquier componente específico. Esto permitirá que todas las aplicaciones puedan comunicarse entre sí en una forma en común. Si el

contrato interno de una unidad cambia, solo el traductor del mensaje y el canal entrante deben cambiar, mientras que el resto de las aplicaciones y los traductores no se verán afectados [10].

Concretamente, el objetivo del MCD es proporcionar los formatos de intercambio de datos del negocio, de tal forma que todos los componentes (servicios, aplicaciones, etc.), sólo necesitan saber su formato y el predeterminado, el cual es el que comparte dentro del bus de servicio. El MCD también representa el contrato estándar utilizado para intercambiar información de negocios. Por lo general este patrón suele utilizarse en organizaciones que poseen múltiples sistemas legados y que además tienen diferentes bases de datos con diversos esquemas y modelado de las entidades de negocio. Por esta razón es el escenario ideal para su implementación cuando se requiera alguna solución de integración.

2.7 Tecnología agnóstica

Es la capacidad que tiene un componente de soportar la interoperabilidad y compatibilidad entre diversas tecnologías, plataformas, sistemas y ambientes, sin requerir una adaptación especial [22]. Este término es utilizado o aplicado no solo a software y hardware sino también a procesos y tareas. Un ejemplo utilizado para distinguir este tipo de tecnología son las *device-agnostic app*, que cuentan con propiedades para ser compatibles con la mayoría de los sistemas operativos, así también el de funcionar en diversos dispositivos o tecnologías incluidos portátiles, tabletas y teléfonos inteligentes. En el desarrollo de nuevas tecnologías se han propuesto técnicas como la de páginas web responsivas (*responsive web design*) que permiten introducir este tipo de soluciones. Además, con la filosofía *device-agnostic* se pretende lograr un alcance mayor en la utilización de tecnología agnóstica en diferentes contextos [23].

Finalmente, se puede definir la tecnología agnóstica como aquella en la que las herramientas no son dependientes de ninguna otra. Esto es, que son alternativas universales que pueden funcionar con una cantidad diversa de plataformas e interactuar con cualquier tipo de información [24].

2.8 Estructura de mensajes SOAP

Los mensajes SOAP se definen como una envoltura digital de datos bajo una estructura donde se encuentra un encabezado (*header*) y un cuerpo (*body*). El cuerpo contiene los datos útiles denominados *payloads* y en su conjunto se le denomina *Envelope*. Cada mensaje tiene una sección principal, que puede contener subsecciones de encabezado y cuerpo. El encabezado contiene información de acciones, entidades y el flujo del mensaje, el cuerpo contiene los datos necesarios

que serán depositados en un modelo de datos, este dominio es conocido en tiempo de diseño en el contrato de los servicios WSDL. Además, pueden contener reglas de codificación, que expresan instancias de tipos de datos definidos por la aplicación. En la Figura 7 se muestra un documento de una representación de un mensaje, compuesto de la estructura o esqueleto XSD y la información que en su conjunto forma el XML, el cual viaja debido a a la ejecución de los servicios web por medio del protocolo SOAP. En la parte superior muestra un encabezado con información que ayudará a saber al cliente receptor de qué forma se va a gestionar el mensaje una vez que llegue a su destino, esto lo lleva a cabo con las propiedades POST, *Host*, *Content-Type* y *Content-Length*. En la etiqueta <*?xml version*> indica la versión del mensaje, también se muestra la envoltura de este mediante la etiqueta <*soap:Envelope*>. Además, se puede ver el cuerpo en la sección formada por la etiqueta <*soap:Body*>. La información viaja en las etiquetas <*m:GetStockPrice*> y <*m:StockName*> [13]. El *payload* contiene datos en formato WSDL que se pasan hacia o desde una función. Las cargas útiles de solicitud contienen todo lo necesario para ejecutar una función, incluidos datos y argumentos pasados como parámetros. Las *payloads* de respuesta contienen los valores que se devuelven desde una función. Las llamadas y respuestas a procedimientos remotos también se describen en un mensaje SOAP. Existen dos tipos de mensajes:

- Los mensajes de solicitud (*request*) solicitan a un proceso remoto que realice algún tipo de procesamiento.
- Los mensajes de respuesta (*response*), los cuales son respuestas de un proceso remoto que devuelven datos o un mensaje de error que indica por qué no fue posible procesar la solicitud.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding"

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

Figura 7. Mensaje SOAP de solicitud.

2.9 Patrones de diseño arquitectónico

Los patrones se han convertido en una herramienta esencial para desarrollar sistemas a gran escala. Estos permiten documentar una solución a un problema recurrente proporcionando un contexto problemático y una solución. Los patrones de diseño se pueden aplicar en el diseño de sistemas, la comunicación, la recopilación de ideas y la revisión de estos [25]. Buschmann [26] define un patrón arquitectónico como "la expresión de un esquema de organización estructural fundamental para los sistemas de software. Proporciona un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para organizar las relaciones entre ellos" [26]. Un patrón de diseño es mucho más que un modelo, es una solución a un problema. Deben explicar por qué lo resuelve y además explicar cuales son las circunstancias bajo las que funciona y cuáles no. Además, son la siguiente etapa tras el dominio de los elementos básicos de un lenguaje o técnica para modelar. Finalmente, ofrecen una serie de soluciones y enseñan lo que constituye un buen modelo y cómo es que se construye [27].

2.9.1 Clasificación de patrones de diseño

Los patrones de diseño se clasifican según su propósito y su alcance. Se dividen en tres grupos importantes, los creacionales, de comportamiento y estructurales. A continuación, se introduce cada uno de ellos.

- **Creacionales**. El propósito de este tipo de patrones es abstraer el proceso de instanciación ocultando los detalles de cómo los objetos son creados [28]. Enseguida se listan algunos ejemplos de ello:
  - ***Factory***. Sirve para la creación de objetos de un tipo determinado y son elegidos por el usuario al momento de crear un objeto. El fundamento del uso de este patrón es que las subclases indican la clase a implementar. Se recomienda cuando: i) una clase no puede saber la clase de objetos que tiene que crear, ii) cuando las subclases tienen que especificar qué tipos de objetos se tienen que crear y iii) cuando se necesite que las clases deleguen la responsabilidad entre varias clases auxiliares.
  - ***Builder***. Ayuda a construir una diversidad de objetos complejos desde un solo punto reutilizando la base del objeto principal. Esto lo logra heredando características y funciones específicas a través de un conjunto de llamadas a interfaces comunes de su clase constructor. Se debe usar este patrón cuando se necesite que la forma de creación de objetos complejos sea independiente en un algoritmo principal de las partes que constituyen a este y cuando el proceso de generación (objeto) pueda tener una variedad de representaciones para el que está construido.
  - ***Prototype***. Ayuda a duplicar objetos de una instancia creada previamente por medio de la clonación. Estos objetos son creados y destruidos constantemente manteniendo

un registro de los que se encuentran activos. Aunque este patrón es implementado por un cliente se puede preguntar al registro si existe antes de clonarlo, a esta gestión se le conoce como *Prototype Manager* y se lleva a cabo mediante un almacén asociativo el cual devuelve el elemento asociado en referencia a una clave, de esta manera se permite al cliente manejar y extender sus prototipos evitando el escribir código. Es recomendable usar este patrón cuando se requiere especificar instancias en tiempos de ejecución y cuando se necesita reducir el número de clases.

- ***Singlenton*.** Este patrón es usado cuando se tiene la necesidad de controlar y restringir la creación de objetos pertenecientes a una clase como un valor único. Se aplica cuando en las plataformas se busca poder garantizar que solo existe una instancia de una clase.

- **De comportamiento**. Permiten definir la comunicación entre los objetos del sistema y el flujo de la información entre los mismos [28]. Ejemplo:

  - ***Iterator.*** Este patrón ayuda a acceder secuencialmente a un grupo de objetos por medio de una interfaz, donde se declaran los métodos necesarios para recuperar la colección de objetos. Existen dos formas de llevar a cabo esta tarea, una externa mediante el cliente, o de forma interna, por medio del *Iterador* [28]. Este patrón se debe de utilizar en tres posibles escenarios: i) cuando se requiera acceder a la información de una colección sin exponer su estructura interna, ii) cuando se necesite usar varios tipos de recorrido sobre una colección y iii) cuando se requiere una interfaz para acceder diferentes tipos de agregados.
  - ***Mediator*.** Este patrón de diseño ayuda a coordinar la interacción entre sus asociados, los objetos se comunican con el cuándo se presenta un evento. Una vez que existe un cambio de estado es notificado por el *Mediator,* el cual responde propagando los efectos de los eventos a los componentes que son afectados, el acoplamiento abstracto de la clase ayuda que los objetos trabajen con diferentes subclases de esta [28]. Se utiliza cuando un conjunto de objetos interactúa entre sí de forma compleja y cuando la comunicación de los objetos está distribuida en diversas clases, estas pueden ser resumidas en una o varias por subclasificación.
  - ***Observer*.** Este patrón ayuda a comunicar un cambio de estado al resto de los componentes relacionados. El patrón *Observer* se encarga de notificar dicho cambio al resto de los componentes dependientes, es importante sea extendida la interfaz de actualización al resto de los integrantes, para que este sepa qué sujeto manifestó el cambio. La actualización por medio de la llamada a la notificación se puede efectuar de dos formas: el sujeto puede realizarlo desde los métodos que cambian de estado, lo que ayuda a desacoplarse del cliente, pero no es eficiente si existen diversos cambios de

estados consecutivos, por otro lado, se puede invocar a la notificación desde el cliente, delegando la responsabilidad de la invocación para notificar [28]. Se recomienda el uso de este patrón en casos en donde: i) una abstracción tiene dos o más atributos dependientes el uno del otro, ii) cuando el cambiar un objeto nos lleva a modificar a otros, sin conocer el número exacto a cambiar y iii) cuando un objeto tiene que ser capaz de notificar a otros sin conocer los otros objetos de una forma precisa, es decir, cuando se quiere bajo acoplamiento.

- ***State*.** Se utiliza cuando el comportamiento de un objeto tiene un cambio, el cual está en función del estado de este mismo. Tiene una implicación sobre la definición de transiciones la cual no indica exactamente en dónde ejecutarse. Para abordar este tema se presentan dos formas de solucionar esto: la primera es definiendo estas dentro de la clase contexto, la segunda es cuando pueden ser definidas en las subclases de *State*, dependiendo la situación conviene utilizar la primera solución cuando el criterio a considerar es fijo, es decir, no cambiará, La segunda resulta adecuada cuando este criterio es dinámico, el problema aquí se presenta en la dependencia por código entre la subclase [28]. Se recomienda utilizar este patrón en casos en los que se necesita que el comportamiento de un objeto esté en función del estado en el que se encuentra y tiene que ser modificado bajo los criterios de las mismas transiciones. También en situaciones en las que es necesario separar una lógica repetitiva y se tiene que separar en una clase por separado.
- ***Interpreter*.** Patrón utilizado cuando se interpreta un lenguaje en una definición como representación de una gramática dada. Además, se especifica el mecanismo de interpretación (intérprete) para su uso [28]. Se debe utilizar este patrón cuando se necesita implementar gramáticas simples, de lo contrario es mejor utilizar *parsers*.
- ***Strategy*.** Es un patrón de comportamiento que ayuda a resolver un problema mediante un grupo de algoritmos. El cliente que lo implementa puede elegir aquel que le conviene e intercambiarlo en tiempo de ejecución según sus necesidades, es implementado mediante un contexto [28]. Se recomienda usar este patrón: i) cuando se necesite una serie de clases relacionadas donde difiere en su comportamiento, ii) en casos en los que se necesite implementar variantes de un mismo algoritmo, iii) si es necesario implementar una jerarquía de clases y iv) si una clase define diversas formas de comportarse basados en condiciones.

- **Estructurales.** Permiten crear grupos de objetos para ayudarnos a realizar tareas complejas [28]. Ejemplo:

  - ***Adapter*.** Este patrón ayuda a convertir la interfaz de una clase en otra que puede ser adaptada a las necesidades de un nuevo desarrollo en particular. Para llevar a cabo su implementación adecuada se recomienda generar una nueva clase que será el *Adapter*,

que es extendida del componente existente y pueda implementar la interfaz obligatoria, de esta forma tenemos la funcionalidad deseada con lo cual se cumple la condición de implementar la interfaz [28]. Se puede usar este patrón en casos en los que se quiere reusar una clase, sin embargo, su interfaz no se adapta a nuestras necesidades y cuando quiere generar una clase reusable que interactúa con otras no relacionadas, es decir, que no tienen necesariamente interfaces compatibles. También se aplica cuando se busca un componente oculto en el código a los clientes.

- ***Bridge*.** El objetivo de este patrón es desacoplar su abstracción de la implementación, de tal forma que las dos puedan ser modificadas sin la necesidad de que una afecta a la otra y viceversa [28]. Las situaciones en las que puede ser usado son: i) cuando se quiere evitar una relación permanente entre la abstracción y su implementación, ii) cuando se requiere extender por medio de subclases, en este caso, que el patrón pueda mezclar abstracciones e implementaciones diferentes, además de extenderlas independientemente de cada una de ellas y iii) si se desea ocultar la implementación de una abstracción a los clientes.
- ***Composite*.** Ayuda a crear objetos complejos (estructuras de árboles) basados de estructuras más simples (hojas) y similares entre sí, con ayuda de la composición recursiva y a una estructura en forma de árbol se trabaja cada componente con la misma interfaz [28]. Se recomienda su uso cuando se necesita generar jerarquías de objetos del tipo "todo-parte" y en caso en los que es necesario ignorar la diferencia entre los componentes y composiciones de estos**.**
- ***Proxy*.** Este patrón ayuda como un intermediario (interfaz) para acceder a cualquier recurso como puede ser una conexión a red, un objeto en memoria, un archivo, etc., actúa como un contenedor que el cliente invoca para acceder al objeto del servicio real [28]. Su uso es recomendado en casos en los que se necesita retrasar la instancia de un objeto hasta que sea realmente necesario usarlo. También cuando se necesite de manera local acceder a un objeto situado en otro espacio de direcciones. Por último, en situaciones en las que se requiere controlar el acceso de un objeto en sistemas concurrentes mediante cerrojo [28].
- ***Facade*.** Patrón que ayuda a tener un solo punto de acceso mediante una interfaz unificada sencilla, donde un cliente gestiona por este el uso de servicios de un grupo de interfaces más complejas [28]. Se recomienda el uso de este patrón cuando se necesita proporcionar una interfaz para un subsistema complejo. Puede usarse también en casos en los que se quiere desacoplar un subsistema de otros subsistemas haciéndose más independiente y portable. Finalmente, cuando se necesita dividir el sistema en niveles de accesos, actuando las fachadas como entradas.

2.10 Patrón arquitectónico *composite*

El patrón *composite* es una estructura conformada por una serie de nodos, es decir, un componente que contiene otros componentes. Está clasificado como estructural por el hecho que constituye una composición compleja (una estructura de árbol) partiendo de otros más simples (hojas o listas de objetos) que son similares entre sí, estas implementan y comparten la misma interfaz donde se definen los métodos que serán implementados en todo el árbol. Con ayuda de la composición recursiva simplifica el tratamiento de los métodos y son gestionados todos los nodos de la misma manera [28].

En la implementación, al cliente no le es necesario saber si se trata de la composición de objetos o si son objetos individuales, el patrón brinda una solución elegante a un problema de gestión entre sus nodos basado en una implementación sencilla. [28].

2.11 Patrón arquitectónico *strategy*

El patrón *strategy* es parte del grupo de patrones de comportamiento que brindan a las aplicaciones diversos métodos de resolución. Las estrategias están conformadas por un grupo de algoritmos que son autónomos, es decir se pueden intercambiar. Incluyen algunas reglas y soporte a los desarrolladores. Por ejemplo, indican como crear u organizar un conjunto de clases y construir objetos, una de las cualidades más sobresaliente de este, es que los objetos y la forma de comportarse se pueden realizar en tiempo de ejecución en un software. Bajo este patrón las piezas básicas asumen funciones especiales, estos se encapsulan, este puede elegir el que prefiera de entre los disponibles o puede ser él mismo que escoja el mejor según la situación que se presente. Cualquier aplicación que proporcione un servicio o función, que pueda ser ejecutada de varias formas, es un posible candidato para utilizar dicho patrón [28].

2.12 Interfaz MAP

La Interfaz MAP (*java.io.Map*) permite representar una estructura para almacenar datos con la estructura por pares de tipo clave/valor (*key*/*value*). Para una clave solo existe un valor relacionado. También es conocida en otros lenguajes de programación como “Diccionarios”, aunque en cada lenguaje tiene sus matices. También, MAP, tiene implementada por debajo toda la teoría de las estructuras de datos de los árboles (AVL, B, B+, B*) por tanto permite agregar, eliminar y actualizar elementos de forma transparente para el desarrollador [29].

MAP es un objeto que asigna claves a valores, no puede contener claves duplicadas, cada una de ellas puede mapearse a un valor como máximo, también modela la abstracción de la función matemática, esta incluye métodos para operaciones básicas (como poner, obtener, eliminar, contiene clave, contiene valor, tamaño y vacío), operaciones masivas (como poner y borrar) y vistas de colección como *keySet*, *entrySet* [9].

2.13 Model-Driven Architecture

La arquitectura dirigida por modelos (MDA, por sus siglas en inglés) es un procedimiento específico para implementar el desarrollo de software dirigido por modelos, el cual fue propuesto por el *Object Management Group* (OMG, por sus siglas en inglés). MDA es un marco de trabajo que proporciona las pautas que deben seguir los desarrolladores durante todo el proceso de para construir software a partir de modelos conceptuales que representen de una forma abstracta la funcionalidad del sistema a construir. Según Liu *et al.* (2011) MDA es un cambio fundamental de un diseño orientado por objetos hacia uno dirigido por modelos el cual fue propuesto por el OMG además señala que la idea central de esta metodología consiste en separar el modelo independiente de la plataforma el cual no está relacionado con la tecnología de implementación y solo contiene la especificación de la lógica del negocio, posteriormente se definen las reglas de conversión de acuerdo a las diferentes tecnologías de implementación con el objetivo de poder convertir el modelo independiente de plataforma a un modelo específico de la plataforma para finalmente convertir dicho modelo en código. Esta es una de las ideas en las que se fundamenta MDA, separar el desarrollo en 3 modelos: el Modelo Independiente de Computación (CIM), el Modelo Independiente de la Plataforma (PIM) y los Modelos Específicos de la Plataforma (PSM). Los modelos conforman todo el proceso de desarrollo antes de la generación de código la cual se hace a partir del PSM.

2.14 Comentarios finales

En el presente capítulo se desarrolló un resumen del marco conceptual enfocado a los aspectos teóricos necesarios para el desarrollo de la arquitectura que presenta esta tesis y se explicó la motivación que dio origen a este trabajo de investigación de forma implícita.

La integración de aplicaciones empresariales es un tema de gran importancia. Sin embargo, los conceptos centrales se pierden en la complejidad de la oferta tecnológica y las promesas de los proveedores. EAI es un concepto que debe entenderse desde las necesidades empresariales, de la

arquitectura de las aplicaciones y el valor que estas últimas proveen al negocio y no solamente desde la perspectiva puramente tecnológica.

Capítulo III
Estado del Arte

Este capítulo presenta los temas de estudio que sirvieron como fundamento a esta tesis. Principalmente, se enfoca en la revisión bibliográfica y el estado del arte. Cabe destacar que este capítulo es el producto de un mapeo sistemático de la literatura (SMS, por sus siglas en inglés) propuesto por Petersen, Feldt, Mujtaba y Mattsson [30] y Kitchenham, Budgen y Brereton [31]. El mapeo sistemático es un método utilizado para conocer y contextualizar un determinado tema. Dicho método tiene como fin identificar, evaluar y sintetizar información de diversas investigaciones con respecto a una temática y a unas preguntas previamente establecidas. El mapeo ofrece un resumen visual y global de la manera en que un tema ha sido abordado y estudiado.

A continuación, se explica el proceso que se realizó para obtener el estado del arte mediante un SMS.

3.1 Estudio de mapeo sistemático

Para conocer las diferentes arquitecturas, iniciativas y propuestas para mejorar el bajo acoplamiento en las integraciones de unidades de software que se encuentran en torno a este tema, se realizó un mapeo sistemático de la literatura. Un SMS es una metodología utilizada ampliamente en investigaciones del área médica y recientemente es aplicada al campo de la Ingeniería de Software. Su objetivo es el de clasificar y estructurar los resultados de investigación que han sido publicados sobre un tema en particular por un periodo de tiempo [31]. Gracias a eso permite conocer los avances en un tema e identificar vacíos (*gaps*) en la investigación. Existe también la metodología conocida como revisión sistemática de la literatura (SLR, por sus siglas en inglés), en donde buscando identificar las mejores prácticas (basándose en la evidencia empírica) se realiza una exploración a fondo de los estudios describiendo su métodos y resultados, la diferencia con el mapeo sistemático radica en que éste último busca proporcionar una perspectiva más general [31].

El objetivo primordial del mapeo sistemático de la literatura es proporcionar una visión más amplia del área de investigación, también identificar la cantidad y el tipo de los resultados disponibles. Asimismo, es importante mapear las frecuencias de publicación a lo largo del tiempo para conocer las tendencias e identificar los foros en los que se ha publicado la investigación en el área [31].

En este trabajo se utilizó la metodología para revisión sistemática de la literatura descrita en [31] pero se aplica la adaptación que se propone en [30] donde es aplicada para realizar un mapeo sistemático de la literatura. El procedimiento de revisión a seguir está compuesto por cinco etapas: 1) Definición de preguntas de investigación, 2) Realizar búsqueda de estudios primarios, 3) Selección de documentos para inclusión y exclusión, 4) Palabras claves de los resúmenes y 5) Extracción de datos y mapeo de estudios.

A continuación, la Figura 8 muestra el proceso del mapeo sistemático, en primera instancia se define las preguntas de investigación que determinan el alcance de la revisión a realizar. Posteriormente, se inicia con la búsqueda de documentos a considerar en el mapeo, se van recopilando cada uno de ellos con ayuda de un repositorio o software para su gestión. Una vez obtenido todos los documentos, se hace una selección de ellos donde se busca sean relevantes al tema que se está considerando, es decir, los denominados estudios primarios. El siguiente paso consiste en crear un esquema de clasificación para los estudios. A continuación, se analizan y se obtiene la información de cada estudio primario seleccionado y es clasificado para dar paso al mapeo sistemático.

Figura 8. Proceso del mapeo sistemático de la literatura.

3.1.1 Definición de preguntas de investigación

Para la realización del mapeo sistemático de la literatura se diseñaron un total de tres preguntas de investigación que permiten estructurar el conocimiento acumulado en un periodo de tiempo delimitado. En la investigación realizada se delimitó para el periodo de tiempo comprendido del año 2008 al 2020, considerando antes de ese tiempo a los autores y trabajos destacados que el mismo mapeo permite obtener. Las preguntas se describen a continuación.

PI-1. ¿Cuáles son las principales propuestas para mejorar el bajo acoplamiento en la integración de unidades de software? El objetivo era conocer las propuestas presentadas para mejora del bajo acoplamiento en las integraciones de software en el periodo de 2008-2020.

PI-2. ¿Qué arquitecturas tecnológicas se han considerado para mejorar el bajo acoplamiento en las integraciones de unidades de software? Se planeó la pregunta para analizar las arquitecturas tecnológicas propuestas para mejora del bajo acoplamiento en las integraciones de software.

PI-3. ¿Qué tecnología y/o marcos de trabajo se han desarrollado para el bajo acoplamiento en las integraciones de software? Esta pregunta se propuso para determinar si existe la falta de herramientas que asistan a los desarrolladores en el bajo acoplamiento en las integraciones de unidades de software.

Una vez definido el periodo de tiempo y las preguntas de investigación, el siguiente paso consistió en hacer una búsqueda de estudios primarios. Estos son, la producción científica, es decir, son los documentos que se pueden obtener después de buscar información sobre un tema acorde a una definición de una cadena de búsqueda con palabras clave. Pueden ser artículos científicos, de revista, congresos, talleres, foros, seminarios, tesis de pre y posgrado.

3.1.2 Realizar búsqueda de estudios primarios

Los estudios primarios se identificaron utilizando cadenas de búsqueda en bases de datos científicas. Para seleccionar los estudios primarios en este trabajo se utilizaron las bases de datos *Springer*, *IEEE*, CONRICYT, *World Wide Web*: *Google Scholar*, *arXiv* y DOAJ. Cabe destacar que una buena forma de crear la cadena de búsqueda es estructurar en términos de población, intervención, comparación y resultado [32].

El periodo de búsqueda seleccionado, como se mencionó anteriormente, es de los años 2008 a 2020.

A continuación, con el fin de realizar la búsqueda de la producción científica asociada al concepto Bajo Acoplamiento (*Loose Coupling*) e Integración de Aplicaciones Empresariales (*Integración de Aplicaciones Empresariales*, IAE), se definieron las palabras clave para formar las cadenas de búsqueda a consultar, se consideraron expresiones de búsqueda genéricas tomando la estructura de cada pregunta de investigación, los términos importantes y los sinónimos identificados. Las expresiones se construyeron usando operadores lógicos para búsquedas (*AND* y *OR*), en la Tabla 1 se muestra la forma en que se generó la cadena de búsqueda respecto a las preguntas de

investigación, para cada pregunta PI-1, PI-2 y PI-3 se define tres columnas en la primera (Términos más importantes), segunda (Sinónimos, términos y temas) y tercera (Expresión de búsqueda).

Tabla 1. Estructuración de las cadenas de búsqueda.

	PI-1	
Términos más importantes	Sinónimos, términos y temas	Expresión de búsqueda
"Loose Coupling", "Enterprise Application Integration", "Software Integration Proposal", "Coupling", "Integration"	"EAI", "Software Unit Integration"	Loose Coupling OR Enterprise Application Integration OR Software Integration Proposal OR (Coupling AND Integration)
	PI-2	
Términos más importantes	Sinónimos, términos y temas	Expresión de búsqueda
"Loose Coupling", "Enterprise Application Integration", "Software Integration Proposal", "Coupling", "Integration", "Technological architectures"	"EAI", "Software Unit Integration"	(Loose Coupling AND Technological architectures) OR (Enterprise Application Integration AND Technological architectures) OR (Software Integration Proposal AND Technological architectures) OR ((Coupling) AND (Integration) AND (Technological architectures))
	PI-3	
Términos más importantes	Sinónimos, términos y temas	Expresión de búsqueda
"Loose Coupling", "Enterprise Application Integration", "Software Integration Proposal", "Coupling", "Integration", "Framework"	"EAI", "Software Unit Integration", "Library"	(Loose Coupling AND Framework) OR (Enterprise Application Integration AND Framework) OR (Software Integration Proposal AND Framework) OR ((Coupling) AND (Integration) AND (Framework))

3.1.3 Selección de documentos para inclusión y exclusión

Para el proceso de selección de estudios primarios se establecieron los criterios de inclusión y exclusión para determinar su relevancia. Los criterios de inclusión definidos se presentan en la Tabla 2 que muestra tres criterios CI-1, CI-2, CI-3 y su respectiva regla de inclusión, de igual forma los criterios de exclusión en la Tabla 3 muestra tres CE-1, CE-2, CE-3 de igual manera con las reglas de exclusión.

Tabla 2. Criterios de inclusión de estudios primarios.

ID	Criterios
CI-1	Los términos relevantes aparecen en el título del documento o resumen.
CI-2	La publicación del artículo es de fecha posterior a 2007.
CI-3	El resumen del estudio primario hace referencia al problema que abarca la pregunta de investigación correspondiente.

Tabla 3. Criterios de exclusión en estudios primarios.

ID	Criterios
CE-1	El estudio primario candidato no está escrito en idioma inglés o español.
CE-2	El estudio primario no tiene relación con el problema de la pregunta de investigación correspondiente.
CE-3	Documentos duplicados del mismo estudio.

3.1.4 Palabras claves de los resúmenes

Para este estudio, se siguió un proceso sistemático (*Keywording*), ver Figura 9, el cual consiste en una forma de reducir el tiempo necesario para desarrollar el esquema de clasificación y garantizar que tenga en cuenta los estudios existentes, este se llevó a cabo en dos pasos: i) se leyeron los resúmenes de los estudios encontrados, ii) se buscaron palabras clave y conceptos que reflejan la contribución del documento para identificar el contexto de la investigación. Esto originó que el conjunto de palabras clave de diferentes documentos se combinarán para desarrollar una

comprensión de alto nivel sobre la naturaleza y la contribución de la investigación, lo que permitió encontrar un conjunto de categorías representativas de la población subyacente.

Por otra parte, cuando los resúmenes son de mala calidad para permitir que se elijan palabras clave significativas, se puede optar por estudiar también las secciones de introducción o conclusión del documento. Cuando se ha elegido un conjunto final de palabras clave, se pueden agrupar y usar para formar las categorías del mapa [31].

A continuación, se muestra en la Figura 9 la secuencia en el proceso de selección y clasificación de los documentos relevantes. En primera instancia se hace la búsqueda con el filtro de las palabras claves, después se crea el esquema de clasificación de acuerdo con el tema y las preguntas de investigación. Si es necesario se actualiza el esquema de clasificación y se clasifica el artículo en el esquema. Con esta información se lleva a cabo el mapeo sistemático.

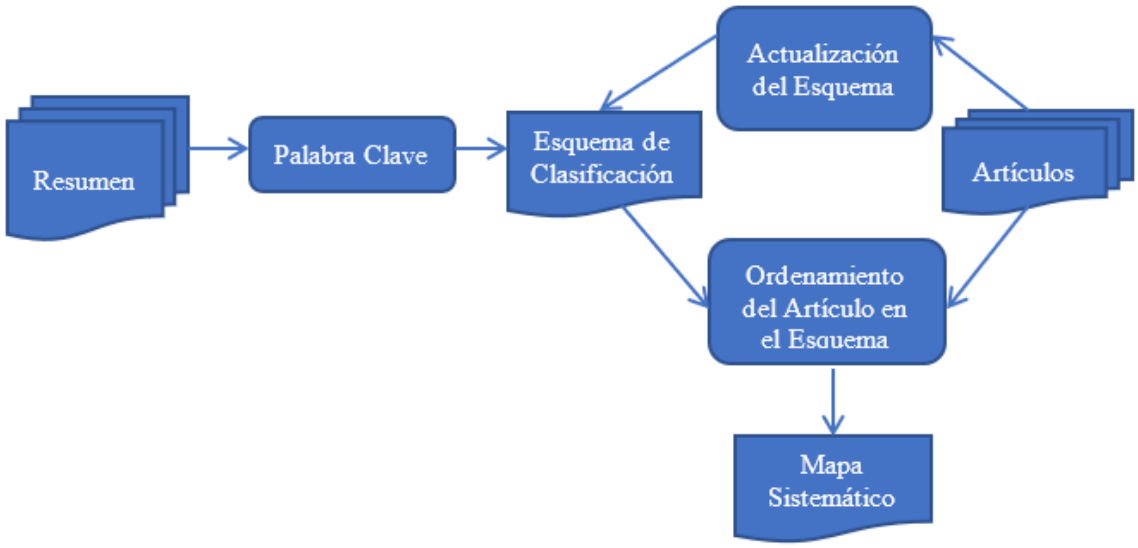


Figura 9. Construcción del esquema de clasificación.

En la Tabla 4 se muestra cómo se clasificó esta investigación de estudios, en principio por medio de categorías, las cuales son *Por pregunta de investigación*, *Propuesta de solución al bajo acoplamiento en las integraciones de unidades de* software y *Por año de publicación*.

Tabla 4. Clasificación de estudios en esquemas.

Categoría	Descripción
Por pregunta de investigación.	Nos brinda el número de estudios relacionados con cada pregunta de investigación, se clasifica (SI o NO).

Propuestas de solución para el bajo acoplamiento en las integraciones de unidades de software.	Se clasifica (SI o NO) donde se propone una solución, la cual debe ser novedosa o por lo menos una extensión significativa de una técnica ya existente.
Por año de publicación	Nos permite conocer sobre el interés que se ha tenido sobre el tema en el tiempo.

3.1.5 Extracción de datos y mapeo de estudios (mapa sistemático)

Cuando se tiene el esquema de clasificación finalizado, los artículos relevantes se clasifican en él, es decir, se realiza la extracción de datos. Como se muestra en la Figura 9, este funciona mientras se realiza la extracción de datos, como el agregar nuevas categorías o fusionar y dividir categorías existentes. En este paso, se utilizó una hoja de cálculo de un paquete informático para documentar el proceso de extracción de datos. El esquema de clasificación [32], está publicado para su consulta en el siguiente enlace: DOI 10.6084/m9.figshare.14888217.v2.

3.2 Principales propuestas para mejorar el bajo acoplamiento en la integración

Las arquitecturas que se han propuesto en los últimos años para EAI, obtenidas de los estudios primarios extraídos de la búsqueda de información en el mapeo sistemático son: i) SOA (*Service Oriented Architecture*) encontrada en [33, 34, 37, 36, 37, 38, 39, 40, 41, 42], ii) Servicios web mencionados en [43, 44, 45, 46, 47, 48], iii) *Microservices* implementados en [49, 50, 51, 52, 53], iv) *Middleware* utilizados en [54, 55, 56], v) *Models of Metadata* estudiados en [57, 58] y vi) *Library and Framework* propuestos en [59, 60]. Los trabajos de investigación mencionados implementaron distintas formas de llevar a cabo el bajo acoplamiento entre las unidades de software, para lograrlo utilizaron un conjunto de técnicas basadas en mensajes asíncronos mediante *middleware queues* y *topics* (el cual es un sistema que coloca un mensaje en una cola y no requiere una respuesta inmediata para continuar el procesamiento). Otras formas de llevar a cabo esta tarea son mediante los llamados contratos de servicio estandarizado WSDL de servicios web y

Microservices. Estos implementan los protocolos como SOAP (*Simple Object Access Protocol*), HTTP (*Hypertext Transfer Protocol*), REST (*REpresentational State Transfer*) con la ayuda del estándar de esquemas de datos como lo es XSD (XML *Schema Definition*), en el cual los datos viajan en un formato XML (*eXtensible Markup Language*). En el caso de REST el formato de esquema utilizado es JSON (*JavaScrip Object Notation*). Otra alternativa que se plantea es llevar a cabo este acoplamiento mediante la construcción y desarrollo de *frameworks*, reglas, *constrains* y modelos (*Models of Metadata*). También se ha encontrado que el uso de las bases de datos federadas es otra forma de establecer el bajo acoplamiento a nivel de datos mediante la creación de modelos canónicos donde se intercambia la información basada en esquemas predefinidos conocidos como Modelos de Datos Canónicos (MDC). Por su parte SOA implementó una orquestación de servicios web, *Microservices*, *Queues*, etc., para el bajo acoplamiento entre las unidades de software. En la Tabla 5 se describen las principales propuestas en esta área como son SOA, *Web Services*, *Microservices*, *Middleware*, *Model of Metadata* y *Library and Framework*.

Tabla 5. Principales propuestas para la mejora del bajo acoplamiento.

Propuesta	**Estudios Primarios**
SOA	10
Web Services	7
Microservices	5
Middleware	3
Model of Metadata	1
Library and Framework	2

Como se observa en la Figura 10, de las propuestas encontradas en los estudios primarios, el 36% fueron implementadas con SOA, el 25% mediante servicios web y el 18% *Microservices*, representando el 80/20 de las propuestas. El resto se desarrollaron con otras tecnologías como *Middleware*, Base de Datos Federadas y Marcos de trabajo (*Framework*).

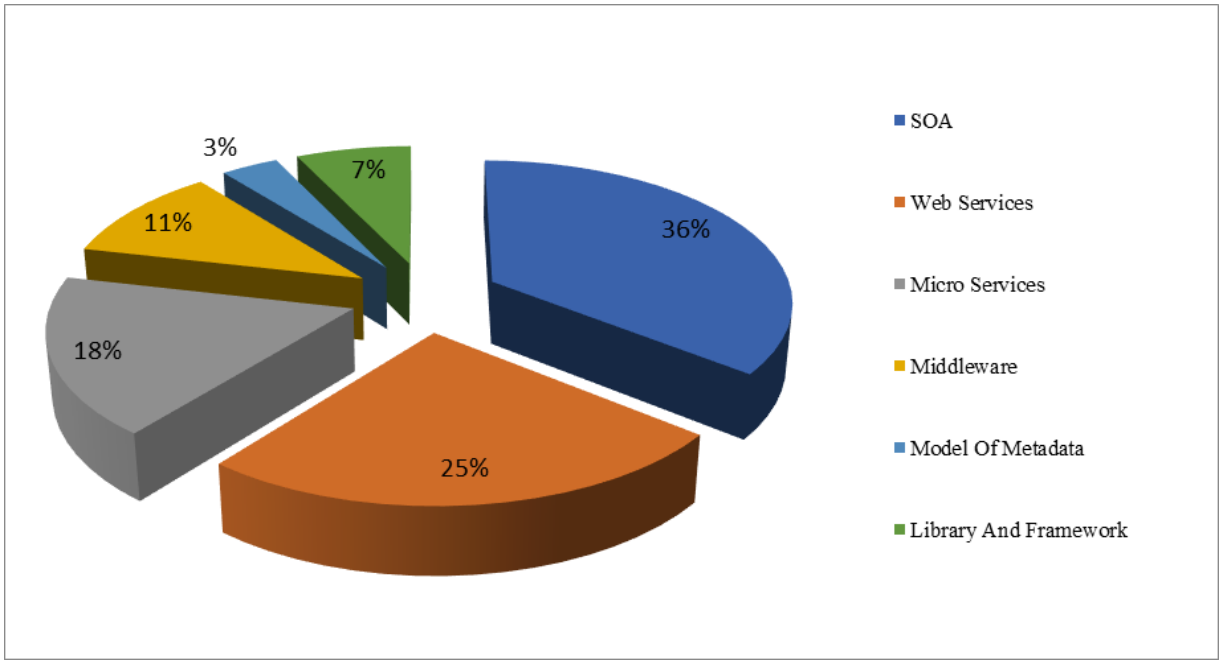


Figura 10. Propuestas para el bajo acoplamiento.

3.3 Arquitecturas consideradas en mejora del bajo acoplamiento en la integración

De la revisión realizada a los estudios primarios, se encontraron una serie de arquitecturas aplicadas en la integración de unidades de software con el objetivo de mejorar el bajo acoplamiento en dichas integraciones. Las más usadas para dicho fin son SOA (*Service Oriented Architecture*) en los trabajos de [34, 35, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 61, 62, 63]. Dicha integración corresponde a una orquestación de tecnologías apoyada con protocolos de comunicación existentes como lo son SOAP (*Simple Object Access Protocol*), HTTP (*Hypertext Transfer Protocol*), entre otros. También, para este fin, surge la arquitectura de *Microservices REST* (*REpresentational State Transfer*) aplicada en las propuestas [49, 50, 51, 52, 53] cómo la segunda arquitectura más utilizada para tal propósito (ver Figura 4). Además, cabe destacar que existen otras arquitecturas que han sido implementadas tales como fueron *Publish/Subcribe* encontrada en [33, 54], *Hub and Spoke* presentada en [64], *Camel Apache* utilizada en [55], *Multi Tier Reference* propuesta en [56], MOM, *Message Oriented Middleware* descrita en [65], *Federated Database* [58], BDI, *Belief Desire Intention* Software *Model* [59], *Intermediate Layer* [66], SCA, *Service Component Architecture* [60] y *Grid Computing* [57], las cuales en su totalidad representan en menor proporción su uso. A continuación, en la Tabla 6 se muestra el resumen de las arquitecturas mencionadas como son SOA *Architecture*, *Microservices Architecture*, *Pub-Sub Architecture*, etc.

Tabla 6. Principales arquitecturas usadas para mejorar el bajo acoplamiento.

Arquitecturas	Estudios Primarios
SOA Architecture	16
Microservices Architecture	5
Pub-Sub Architecture	2
Hub and Spoke	1
Camel Apache Architecture	1
Reference Architecture	1
Message Oriented Middleware	1
Federated Database Architecture	1
BDI Architecture	1
SCA Architecture	1
Grid Computing Architecture	1

En el caso de las arquitecturas implementadas, en la Figura 11 se puede observar que SOA representa el 50% de las implementaciones encontradas en los estudios primarios, el 16% para los *Microservices*, el 7% *Publish/Subcribe* y el 27% restante está integrado por un conjunto de arquitecturas diversas.

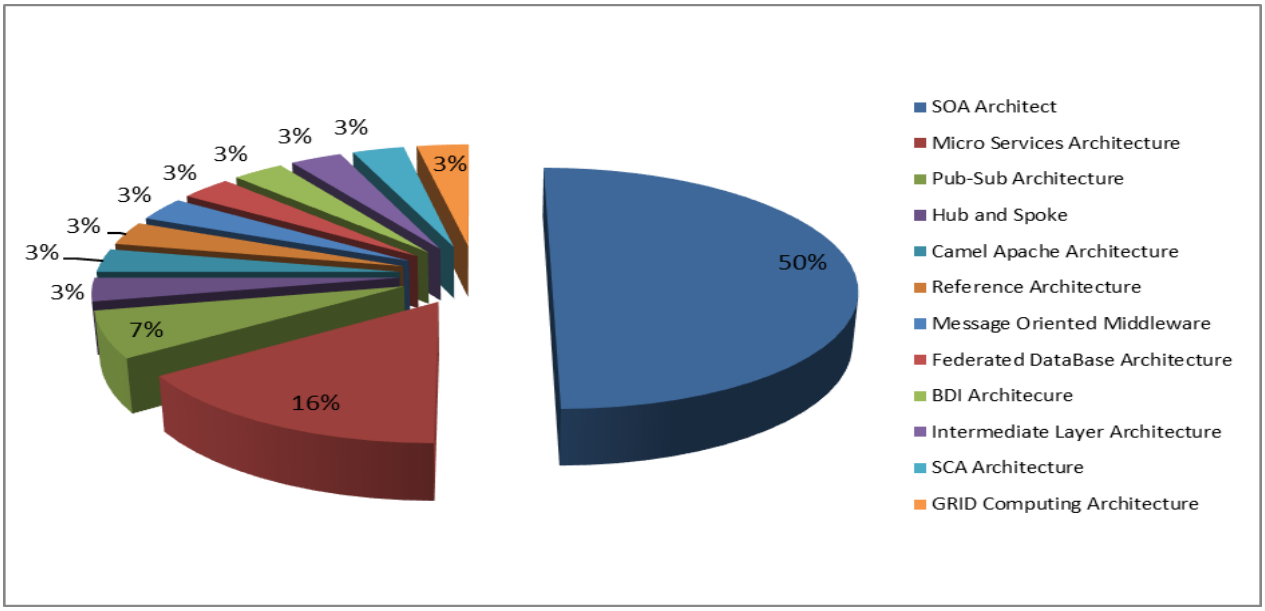


Figura 11. Arquitecturas implementadas.

3.4 Tecnologías y/o marcos de trabajo para el bajo acoplamiento en la integración

Las tecnologías y/o *frameworks* utilizados para la integración de unidades de software aplicando el bajo acoplamiento encontradas en las publicaciones fueron *SOA* (*Service Oriented Architecture*), SOAP (*Simple Object Access Protocol*), WSDL, XSLT (*eXtensible Stylesheet Language for Transformations*), ESB (*Enterprise Service Bus*), XML (*eXtensible Markup Language*), XSD (*XML Schema Definition*), BPEL (*Business Process Execution Language*), JMS (*Java Message Service*). Dichas tecnologías se encontraron en los trabajos publicados por [33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 44, 46, 63, 67, 68, 69]. También, un conjunto de tecnologías para estos propósitos fueron los *Microservices* REST (*REpresentational State Transfer*), JSON (*JavaScrip Object Notation*) [49, 50, 51, 52, 53, 70], junto con ello los servicios web, WSDL, SOAP (*Simple Object Access Protocol*), XML (*eXtensible Markup Language*), XSD (*XML Schema Definition*) [43, 44, 47, 48]. Asimismo, se encontró que se han aplicado las tecnologías *Publish/Subscribe*, JMS (*Java Message Service*), *Queue*, *Topics*, MDB (*Message Driver Bean*) [54, 71] y por último una serie de tecnologías con menos implementaciones como lo fueron *Apache Camel Java Framework* [59], HL7 (*Health Level Seven*), Dublin Core [57], CORBA *(Common Object Request Broker Architecture*), RMI (*Java Remote Method Invocation*), *Canonical Model*, ODBC (*Open Database Connectivity*) / JDBC (*Java Database Connectivity*) [58], *Self Adaptive*, AI (*Artificial Intelligence*), KQML (*Knowledge Query Manipulation Language*), *Neuronal Network* [59] y la tecnología SCA (*Service Component Architecture*) [60].

Es importante mencionar que a pesar de que se encontraron una gran cantidad de diferentes tecnologías enfocadas en tratar el bajo acoplamiento en las integraciones de unidades de software, lamentablemente la mayoría de las arquitecturas ya probadas llamadas patrones arquitectónicos o arquetipos, utilizan estructuras definidas XSD en tiempo de diseño llamados contratos de servicio representadas por el lenguaje de descripción para servicios WSDL o interfaz abstracta que crea la dependencia a nivel de datos en el elemento <*message*>, esto crea el estrecho acoplamiento entre las plataformas, sistemas y/o aplicaciones. A continuación, en la Tabla 7 se muestra el resumen de tecnologías y/o *frameworks* encontrados en los estudios primarios, como son SOA, *Microservices*, *Web Services*, *Public-Subcribe*, etc.

Tabla 7. Principales tecnologías y/o *frameworks* para mejorar el bajo acoplamiento.

Tecnologías y/o *Frameworks*	**Estudios Primarios**
SOA	17
Microservices	6
Web Services	4
Public-Subcribe	2
Camel Apache	1
HL7-Dublin Core	1
CORBA-Canonical Model	1
Self-Adaptive Technology	1
SCA Technology	1

Para un mejor entendimiento de las tecnologías y/o *frameworks* aplicadas durante el periodo de tiempo comprendido de 2008 a 2020 a continuación se presenta la Figura 12, en donde podemos ver que el 80/20 se clasifica SOA *Tech* con un 50%, *Microservices Tech* con 17%, y *Web Services Tech* con un 12% de las implementaciones tecnológicas. El 21% restante representa una diversidad de tecnologías entre estándares, marcos de trabajo (*frameworks*), librerías, patrones tecnológicos y protocolos de comunicación.

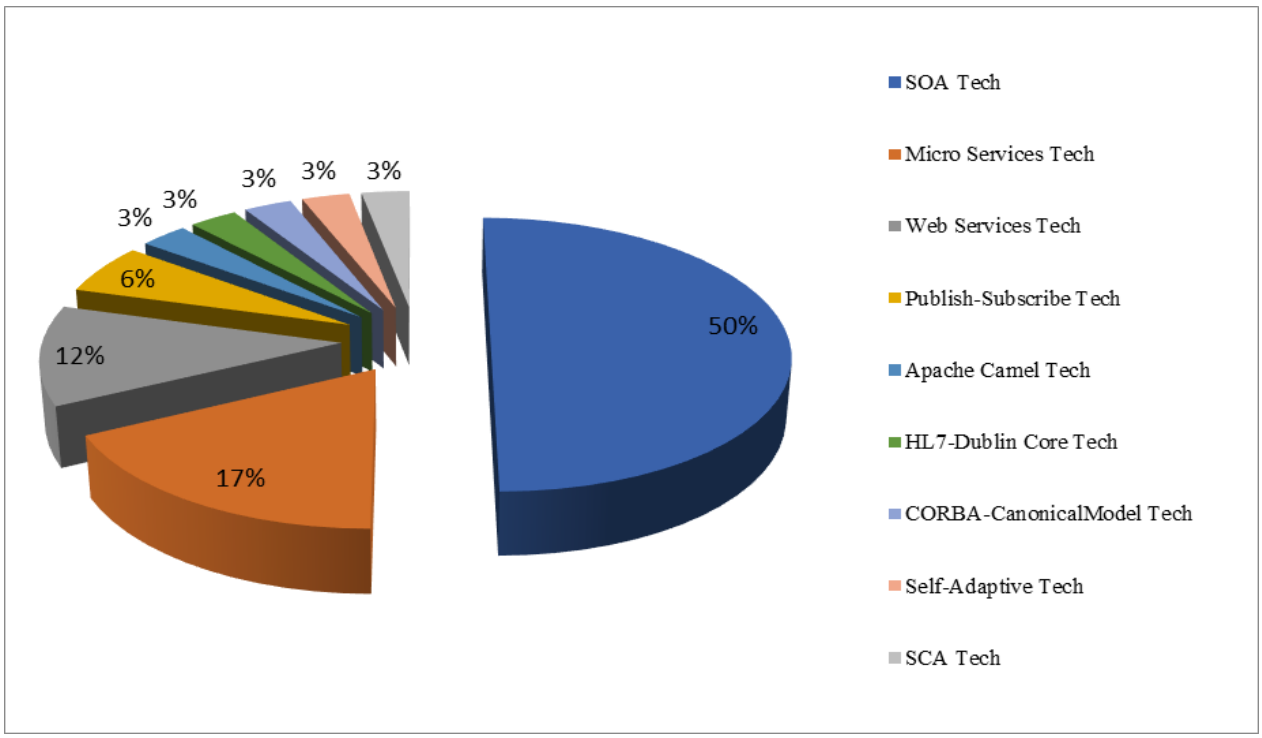


Figura 12. Tecnologías y/o *frameworks* implementadas.

3.5 Análisis

En esta sección se presenta un análisis de las principales propuestas, arquitecturas y tecnologías para mejorar el bajo acoplamiento en las integraciones de unidades de software, el cual ha sido fundamentado en la revisión que se efectuó en los estudios primarios durante el periodo comprendido de 2008 a 2020.

Se encontró como solución al bajo acoplamiento de unidades de software las arquitecturas o arquetipos SOA con un 36%, servicios web con 25% y *Microservices* con el 18%, en conjunto 79% de los estudios primarios que dan una solución en el bajo acoplamiento en integraciones de unidades de software.

En la Figura 13 se muestra las diversas soluciones encontradas para el bajo acoplamiento de unidades de software las arquitecturas SOA con un 36%, *Web Services* con 25% y *Microservices* con el 18%, en conjunto 79% de los estudios primarios que dan una solución en el bajo acoplamiento en integraciones de unidades de software.

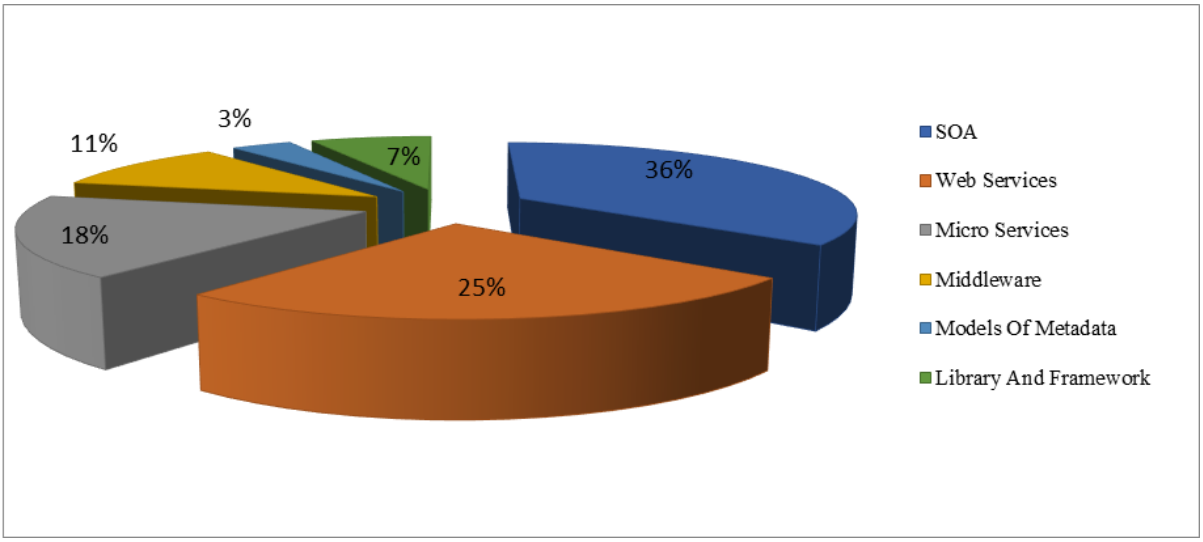


Figura 13. Propuesta de solución en el bajo acoplamiento.

En un ambiente SOA, servicios web y *Microservices*, los nodos de la red hacen disponibles sus recursos a otros participantes en la red como servicios independientes a los que tienen acceso de un modo estandarizado. La mayoría de las definiciones identifican la utilización de los servicios web (empleando SOAP y WSDL) en su implementación, no obstante, se puede usar utilizando cualquier tecnología basada en servicios [15]. En este sentido, un documento WSDL define los servicios como colecciones de puntos finales de red o puertos, la definición de estos y los mensajes

se separa de su despliegue de red concreto o enlaces de formato de datos, esto permite la reutilización de definiciones abstractas como mensajes, que son descripciones de los datos que se intercambian, y tipos de puertos que son colecciones abstractas de operaciones. Asimismo, el protocolo concreto y las especificaciones de formato de datos para un tipo de puerto particular constituyen un enlace reutilizable. En este contexto, un puerto se define asociando a una dirección de red con un enlace reutilizable y una colección de puertos define un servicio, por lo tanto, un documento WSDL utiliza los siguientes elementos en la definición de servicios de red [15]:

- *Types*: Contenedor para definiciones de tipos de datos que utilizan algún tipo de sistema como *XSD* (*XML Schema Definition*).
- *Message*: Definición abstracta de intercambio de los datos que se comunican.
- *Operation*: Es una descripción abstracta de una acción soportada por el servicio.
- *Port Type*: Conjunto abstracto de operaciones admitidas por uno o más puntos finales.
- *Binding*: Especificación concreta de protocolo y formato de datos para un tipo de puerto en particular.
- *Port*: Único punto final definido como una combinación en un enlace y una dirección de red
- *Service*: Una colección de puntos relacionados.

Los elementos mencionados se definen en dos conjuntos, una es la parte concreta que indica el "cómo" y "dónde" y la sección abstracta define "qué" hace el servicio a través de los mensajes que envía y recibe [15].

Debido a este último punto encontramos una deficiencia en el alto acoplamiento a nivel externo y de datos en la integración de aplicaciones o unidades de software. Es decir, se detectó que el elemento <*message*> genera este acoplamiento, debido a que se crea bajo una estructura fija predefinida (XSD, por sus siglas en inglés), antes de ser usada, a continuación, se muestra un fragmento de código XML de un documento WSDL, para un mejor entendimiento la Figura 14 ejemplifica como se definen los tipos de datos en tiempo de diseño quedando una estructura inamovible en tiempo de ejecución, a su vez hace referencia al detalle de los datos bajo un espacio de nombre <xsd1> referenciando al objeto "*TradePrice*".

A continuación, la Figura 14 muestra un fragmento de código del XML, donde podemos ver los elementos o *tag* que conforman la estructura del contrato WSDL, como cabecera se encuentra la versión del XML "*?xml versión*", después en la etiqueta *targetNamespace* se encuentra localizados la definiciones de los diferentes documentos que conforman el servicio web. Además de otros

documentos importados en la etiqueta *import namespace*. Se continúa con la sección de los mensajes de entrada y salida representada con las etiquetas <*Message name = "GetLastTradePriceInput"*> y <*Message name = "GetLastTradePriceOutput"*>, finalizando con la sección de las operaciones que los clientes consumidores pueden usar bajo la etiqueta <*portType*>, de esta manera se constituye el documento.

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote/definitions"
          xmlns:tns="http://example.com/stockquote/definitions"
          xmlns:xsd1="http://example.com/stockquote/schemas"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

   <import namespace="http://example.com/stockquote/schemas"
           location="http://example.com/stockquote/stockquote.xsd"/>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>
</definitions>
```

Figura 14. Fragmento de código *XML* definición del elemento <*message*>.

En la Figura 15, por su parte, se observa el detalle del elemento "*TradePrice*" en donde lo señalado en amarillo en el código representa el dato y/o las propiedades de la estructura llamada "*price*" y el tipo de esta propiedad "*float*" marcando la forma estricta de recibir el intercambio de información entre unidades de software.

```
<?xml version="1.0"?>
<schema targetNamespace="http://example.com/stockquote/schemas"
       xmlns="http://www.w3.org/2000/10/XMLSchema">

    <element name="TradePriceRequest">
        <complexType>
            <all>
                <element name="tickerSymbol" type="string"/>
            </all>
        </complexType>
    </element>
    <element name="TradePrice">
        <complexType>
            <all>
                <element name="price" type="float"/>
            </all>
        </complexType>
    </element>
</schema>
```

Figura 15. Fragmento de código *XML* definición del elemento <*xsd1*: *TradePrice*>.

Por lo que, haciendo referencia al ejemplo descrito en las Figuras 14 y 15, si se considera un cambio en el elemento (*TradePrice*) este contrato impactaría a todos las aplicaciones o unidades de software que están integrados a esta estructura por ejemplo si se considera que este servicio es consumido por un centenar de clientes y cada uno de ellos utiliza el mismo contrato de servicio o documento WSDL para acceder a realizar alguna operación, y debido a algún cambio en el tipo de dato que solicite algunos de esos clientes, o bien se quiera integrar uno nuevo haciendo necesario enviar un dato más, se tendrá que crear otro contrato nuevo o documento WSDL por cada cliente que ha cambiado, esto conlleva tener un control y mantenimiento por cada contrato creado, en el peor de los casos se tendrían cien contratos que mantener y esto representaría un alto costo en recursos como el tiempo, esfuerzo y dinero.

Por otro lado, en la clasificación que se realizó para dar una solución en el bajo acoplamiento en las integraciones de software, se encontró en los estudios primarios (ver Figura 16) que en el periodo del año 2008 al 2011 la comunidad científica dedicó un esfuerzo considerable para definir e implementar soluciones a la problemática. Ese énfasis tuvo un decremento en el periodo 2012 al 2017 y retomó un impulso en el periodo de tiempo de los años 2018 al 2020. Se observa que, en ese periodo, del 2008 al 2020, se definieron una serie de propuestas basadas en las arquitecturas SOA, servicios web y *Microservices* con lo cual se determinó que fueron las tecnologías mayormente implementadas para tal propósito.

En la Figura 16 se observa una distribución de los estudios primarios durante el periodo que comprende desde el año 2008 al año 2020 con respecto a las preguntas de investigación. En la gráfica, las siglas PI-1, PI-2 y PI-3 corresponden con las preguntas de investigación presentadas en la sección 3.1.1. de este capítulo.

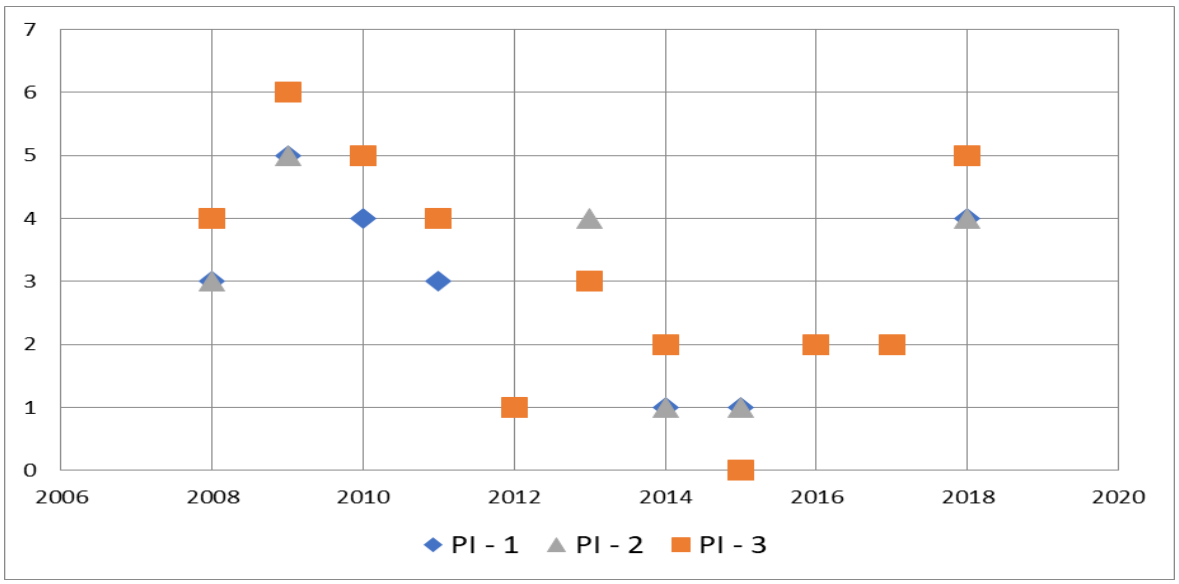


Figura 16. Año de publicación propuestas, arquitecturas y tecnologías.

3.6 Discusión

El SMS ha traído varias percepciones sobre las tendencias de investigación en EAI a nivel de unidades de software. A lo largo de los años, el principal problema de EAI ha sido la comunicación e intercambio de datos entre sistemas heterogéneos. Recientemente, en el período cubierto por el mapeo sistemático, han surgido nuevas tecnologías para desarrollar sistemas de software, algunas tecnologías existentes se han utilizado en combinación con las nuevas para proporcionar marcos de trabajo robustos para el desarrollo de aplicaciones que satisfagan las necesidades de empresas en lo que respecta a la integración.

Un tema que ha quedado atrás con el paso de los años es el considerar un proyecto de EAI como un proyecto independiente que refleja sus características únicas. No se encontró ninguna investigación en los estudios primarios con respecto a propuestas que presenten una guía paso a paso para implementar un proyecto de EAI. Este es un tema crítico que debe de ser estudiado debido a que un proyecto de desarrollo de software con una arquitectura distribuida optimizada para el intercambio de datos no es lo mismo que un proyecto de EAI. Las directrices de la SE permiten obtener un producto que satisfaga las necesidades del cliente en un proyecto de desarrollo de software, pero este no es el caso en proyectos de EAI. Además, no existe un enfoque metodológico genérico para que las empresas implementen un proyecto de integración de aplicaciones empresariales. Este tipo de proyecto se considera como un componente más en un proceso de desarrollo de software tradicional que implica un retraso en su finalización. Por lo tanto, no se considera como un proyecto independiente que debe definirse dentro de un enfoque metodológico solo para EAI. En este sentido, las empresas no priorizan cuestiones como la forma de medir el valor que un proyecto de EAI brinda en un futuro cercano y cómo podría ayudar significativamente a reducir los costos de mantenimiento de sus sistemas. Cuando se deciden por un proyecto de EAI, carecen de una planificación ad-hoc guiada a través de una metodología para ello.

A través de la constante evolución de plataformas tecnológicas para el desarrollo de sistemas software, han surgido propuestas para EAI, como se observa en los estudios primarios analizados en el mapeo sistemático. La mayoría de ellos están basados en SOA y microservicios. La evolución de la tecnología ha sido tal que ha superado a las propuestas que deberían existir para solucionar problemas que la EAI ha arrastrado desde los primeros años. Las tecnologías aparecen para desarrollar nuevos sistemas software y como resultado una nueva integración esencial nace con él. En este sentido, la investigación en el campo de integración de aplicaciones empresariales ha dedicado mucha atención en lograr un simple intercambio de datos entre unidades de software. Pero, la privacidad de los datos y las preocupaciones de seguridad han aumentado. Aunque esta

área está fuera del alcance de este SMS, la investigación en EAI hasta la fecha ha pasado por alto la seguridad y privacidad de los datos. Estos problemas surgieron debido a las nuevas plataformas tecnológicas y no han sido adecuadamente estudiados.

La investigación en EAI no proporciona hasta el día de hoy un marco general para proyectos de integración de aplicaciones empresariales, ni a nivel de intercambio de datos, bases de datos o interfaces. En este sentido, se han realizado algunos esfuerzos, por ejemplo, se han industrializado propuestas de integración y se han publicado artículos científicos presentando estas soluciones. Las propuestas ayudan a la integración de proyectos de EAI porque están basados en MDA (consultar sección 2.13). La ventaja propuesta por los proyectos de integración basados en MDA para resolver el problema EAI es que los mismos modelos creados se pueden convertir en el código fuente del lenguaje de programación que se utilice en la integración. Esto proporciona una ventaja a los proyectos de EAI porque al reutilizar los modelos, es posible generar la solución exacta para diferentes plataformas tecnológicas y construir el código necesario para la integración. La ventaja de esta idea consiste en mejorar una deficiencia en la integración de aplicaciones empresariales, la cual es la reconfiguración de los sistemas empresariales y no sólo la integración. Independientemente de la importancia de un enfoque como MDA, no ha habido suficiente investigación en este campo como en otros temas acerca de EAI.

La literatura científica destaca que las soluciones EAI actuales se enfrentan a un problema de heterogeneidad. Por lo tanto, las soluciones de EAI carecen de un enfoque de integración robusto y consistente apoyado en una metodología diseñada para ese trabajo. Particularmente dedicada a la integración de aplicaciones empresariales heterogéneas que consideren las actividades de la Ingeniería de Requisitos (RE, por sus siglas en inglés). Dichas actividades se pueden modelar de acuerdo con las necesidades de la integración teniendo en consideración importantes atributos de calidad del software como la seguridad y privacidad. Incluso esto permitiría que los requisitos pudieran ser modelados explícitamente según la necesidad de la integración con lo cual podría mejorarse su especificación en la ejecución de la integración.

Como se mencionó anteriormente, las aplicaciones empresariales están creciendo en número en diferentes sectores de la sociedad. Los sistemas software utilizados en las empresas suelen tener implementada su funcionalidad en diferentes plataformas y tecnologías recientes, por lo que se necesitan soluciones EAI dinámicas para resolver los problemas de integración que se presenten. Las soluciones de integración EAI se pueden lograr utilizando servicios web, SOA y con tecnología reciente como los microservicios REST. Una ventaja de estas nuevas tecnologías es la facilidad de integración a bajo nivel que mejora el intercambio de datos entre aplicaciones. Esto permite la interoperabilidad a través de un flujo controlado de datos.

A medida que evolucionan las nuevas tecnologías, es importante que las soluciones de EAI se adapten a ellas, pero es necesario aumentar la investigación en esta área porque la evolución es tan rápida que los problemas que existen hoy seguirán existiendo mañana junto con otros más como resultado de: (i) nuevas tecnologías de implementación, (ii) la creciente demanda del mercado de las telecomunicaciones, y (iii) un rápido cambio en la empresa en el departamento de TI y su entorno tecnológico. Además, mientras tanto, la principal aplicación de la EAI es la colaboración entre sistemas de diferentes empresas, por ejemplo, bancos para pagos en ventas, servicios de envío, así como proveedores de productos. Dichas empresas dedican una cantidad considerable de su presupuesto y de recursos humanos para mantener el intercambio de datos entre las empresas con las que interactúan en sus procesos de negocio. Por lo tanto, al crear interfaces a nivel de unidad de software que puedan integrar sus aplicaciones se podrán reducir los recursos dedicados al mantenimiento. Según los estudios primarios obtenidos de la revisión bibliográfica realizada en este SMS, SOA es descrito por varios autores como una arquitectura que puede ayudar a resolver no solo el problema de integración sino también optimizar técnicas de integración.

Hoy en día, la EAI enfrenta dos importantes desafíos: la integración sintáctica y semántica entre aplicaciones empresariales a nivel de unidades de software o de datos. Esto se debe a que cada departamento/área en su momento construyó sus sistemas sin considerar a los demás, o la empresa fue creciendo con el paso del tiempo, con lo cual la interoperabilidad se volvió difícil entre esos sistemas software. Los estudios primarios muestran que es esencial tener un enfoque para la integración que sea semánticamente coherente derivado de un análisis previo a la integración. Para ello, las propuestas siguen utilizando una definición declarativa de los tipos de datos y formatos de los campos de las estructuras de datos utilizadas por los sistemas software para facilitar el intercambio de información en la configuración de la integración.

3.7 Comentarios finales

En este capítulo se presentó el estado del arte en la temática referente al bajo acoplamiento de unidades de software en el contexto de la EAI. Los resultados analizados son producto de la definición y ejecución de un Estudio de Mapeo Sistemático, SMS por sus siglas en inglés. Para ello se consideró un total de 1,104 artículos publicados en la literatura científica y extraídos de las bases de datos bibliográficas-científicas tales como *Springer, IEEE* y CONRICYT. Se consideró la *grey literature* para buscar en la *World Wide Web*, *Google Scholar* por tesis, *drafts*, *white papers* y reportes técnicos, y se consideraron revistas científicas abiertas tales como *arXiv* y *DOAJ*. De los estudios obtenidos, 42 dieron respuesta a las preguntas de investigación y fueron analizados.

El análisis de los estudios primarios ha demostrado que, en un periodo de 12 años, considerando aquel comprendido del año 2008 al 2020, se siguen implementando las mismas arquitecturas, patrones arquetipos y tecnologías en el bajo acoplamiento de unidades de software en la EAI. De ellos, el más utilizado en literatura científica e industria es SOA en combinación con servicios web. Existe una variación que se está aplicando en los últimos años (2018 al 2020), *Microservices*. Ambas soluciones han permitido obtener resultados satisfactorios y de cierta forma cumplen con el propósito del bajo acoplamiento en las integraciones, sin embargo, al final siguen utilizando un contrato de datos o de servicio WSDL, lo que ocasiona que el acoplamiento se considere estrecho, es decir una dependencia más fuerte entre las unidades de software. En este sentido, se recordará que un contrato de datos o de servicio WSDL es un acuerdo formal entre un servicio y un cliente que abstractamente describe los datos que se van a intercambiar. Es decir, para comunicarse, el cliente y el servicio no tienen que compartir los mismos tipos, solo los mismos contratos de datos. Un contrato de datos define con precisión, para cada parámetro o tipo de valor devuelto, qué datos se serializan (se convierten en XML) para su intercambio. Por lo que el acoplamiento a nivel de datos se vuelve un acoplamiento estrecho.

A razón de lo expuesto en el párrafo anterior, es necesario proporcionar soluciones dentro de las arquitecturas que brindan la facilidad de desacoplar las unidades de software al ser integradas. Esto debido a los problemas que se presentan, tales como el alto costo en la integración de plataformas y el de mantenimiento. Esa problemática se intentó solucionar en los trabajos expuestos en los estudios primarios de este SMS tales como *SOA* (*Service Oriented Architecture*), SOAP (*Simple Object Access Protocol*), WSDL, XSLT (*eXtensible Stylesheet Language for Transformations*), ESB (*Enterprise Service Bus*), XML (*eXtensible Markup Language*), XSD (*XML Schema Definition*), BPEL (*Business Process Execution Language*), JMS (*Java Message Service*) [33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 46, 63, 67, 68, 69]. Otro conjunto de tecnologías para estos propósitos fueron los microservicios REST (*REpresentational State Transfer*), JSON (*JavaScript Object Notation*) [49, 50, 51, 52, 53, 70], así como los servicios web, WSDL, SOAP, XML, XSD [43, 45, 47, 48], pero todos ellos con la problemática en común: presentando la falta del bajo acoplamiento a nivel externo y de datos en sus contratos WSDL ya expuesta anteriormente en la subsección 3.5 de este capítulo de la tesis.

Capítulo IV

Arquitectura modelo canónico de datos dinámico mediante mensajes agnóstico

En este capítulo se describe el producto derivado de la investigación realizada en esta tesis. Esta propuesta de integración está conformada por dos componentes arquitectónicos llamados *composite* y *strategy*, ambos conocidos en la ingeniería de software como patrones de diseño que ayudaron a la solución. Además, se utilizó una estructura representativa de los campos y valores de una entidad de una base de datos, que de manera dinámica puede adaptarse a cualquiera de estas en tiempo de ejecución, se llama arquitectura modelo canónico de datos dinámico (MCDD) mediante mensajes agnósticos. Su objetivo consiste en mejorar el bajo acoplamiento a nivel de datos y externo en la integración de unidades de software, permitiendo la reducción de costos de implementación, así como de mantenimiento en las integraciones de plataformas empresariales. Esto es posible debido a su escalabilidad, reutilización y flexibilidad. Se concluye el capítulo con un ejemplo de aplicación en un entorno empresarial real para demostrar el funcionamiento y beneficios de la arquitectura propuesta.

4.1 Introducción

La arquitectura modelo canónico de datos dinámico (MCDD) mediante el uso de mensajes agnósticos, de aquí en adelante se le llamará *Agnostic Message*, es una propuesta dirigida a mejorar el bajo acoplamiento externo y de datos en las integraciones de unidades de software, como parte de las estrategias de integración de aplicaciones empresariales. La arquitectura está constituida por dos patrones de diseño, además del uso de estructuras de datos MAP (consultar Capítulo II) que describen en conjunto la solución del bajo acoplamiento en las integraciones de aplicaciones y plataformas digitales mediante el intercambio de mensajes diseñados con tal propósito.

4.2 Estructura de la arquitectura MCDD

La solución sobre integración mediante MCDD, se divide en tres componentes, el primero es el mensaje con el uso de la estructura de datos MAP, el segundo, por su parte, es el patrón *Composite* que es parte del mensaje y permite encapsular los datos en un compuesto universal, es decir, en una estructura única y estandarizada que no cambia en el tiempo, por otro lado el tercer componente es el patrón *Strategy* con el cual se define la mejor estrategia a tomar para la integración mediante el uso de un *Contexto* según convenga (ver Figura 17). Los componentes tienen por objetivo mejorar el bajo acoplamiento. A continuación, en la sección 4.3 se explica el componente *Agnostic Message* que implementa la estructura del mensaje al cliente, en la sección 4.4 se detalla el componente *Envelope* como parte del patrón *Composite* y en la sección 4.5 se explica la interfaz *StretegyCDDM* que expone las estrategias para llevar a cabo el manejo de las acciones en la integración. La sección 4.6 detalla la funcionalidad de MCDD y finalmente en la 4.7 se presenta un ejemplo de aplicación en un escenario real.

La Figura 17 presenta los componentes de la solución MCDD, la primera parte, de izquierda a derecha indica el *Client* que puede ser cualquier plataforma, unidad de software, sistema, etc. que hace uso de la arquitectura. El segundo componente llamado *AgnosticMessage* representa el envoltorio o mensaje principal donde se envían las instrucciones al receptor para que este conozca cómo será gestionado el mensaje. Ahí mismo, también viaja la información representada como MAP y la acción *Action Type* a ejecutar en las entidades *Entity Name*. El tercer componente *Composite* representa la estructura donde se configura y almacena la carga útil *Payload* para ser integrada. Por último, el componente *Strategy* es donde viven las estrategias o algoritmos a implementar según convenga las instrucciones del mensaje.

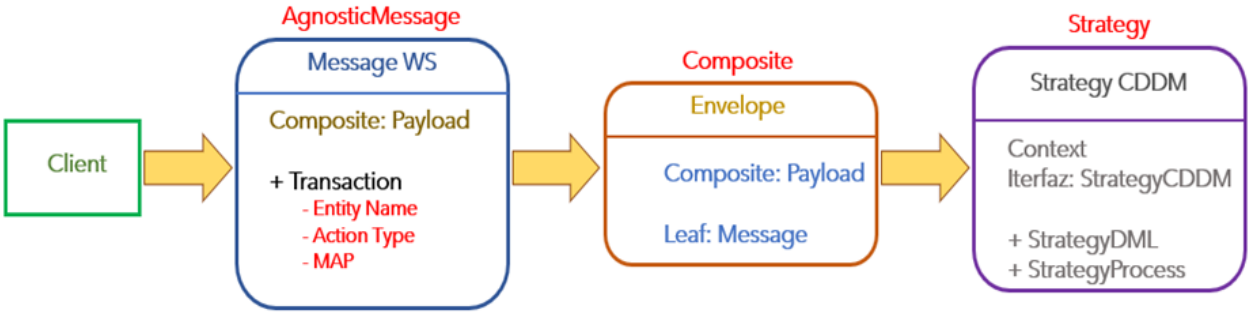


Figura 17. Vista de arquitectura MCDD mediante *Agnostic Message*.

4.3 Agnostic message

En una integración de unidades de software la interacción entre ellas se realiza mediante el intercambio de información, en esta solución se presenta el *Agnostic Message* que es la representación de las transacciones y propiedades de una entidad existente en un repositorio o base de datos. La particularidad de este mensaje radica en el contrato o firma universal que se expone a las aplicaciones para envío de información estandarizada dentro de una estructura dinámica que usa el componente MAP para este propósito. La representación gráfica de la estructura se muestra en la Figura 18, que expone los componentes que conforman el mensaje.

En la Figura 18 se muestra *Agnostic Message* y las partes que conforman el mensaje a detalle. En primer lugar, *Envelope* el cual es el objeto que almacena las secciones de *Payload*, es decir la carga útil, que a su vez contiene *Properties Entity* o propiedades de la entidad y la sección *Transaction* que constituye la información representada como un MAP <*key*, *value*> la cual representa la información que será usada según convenga al algoritmo de gestión.

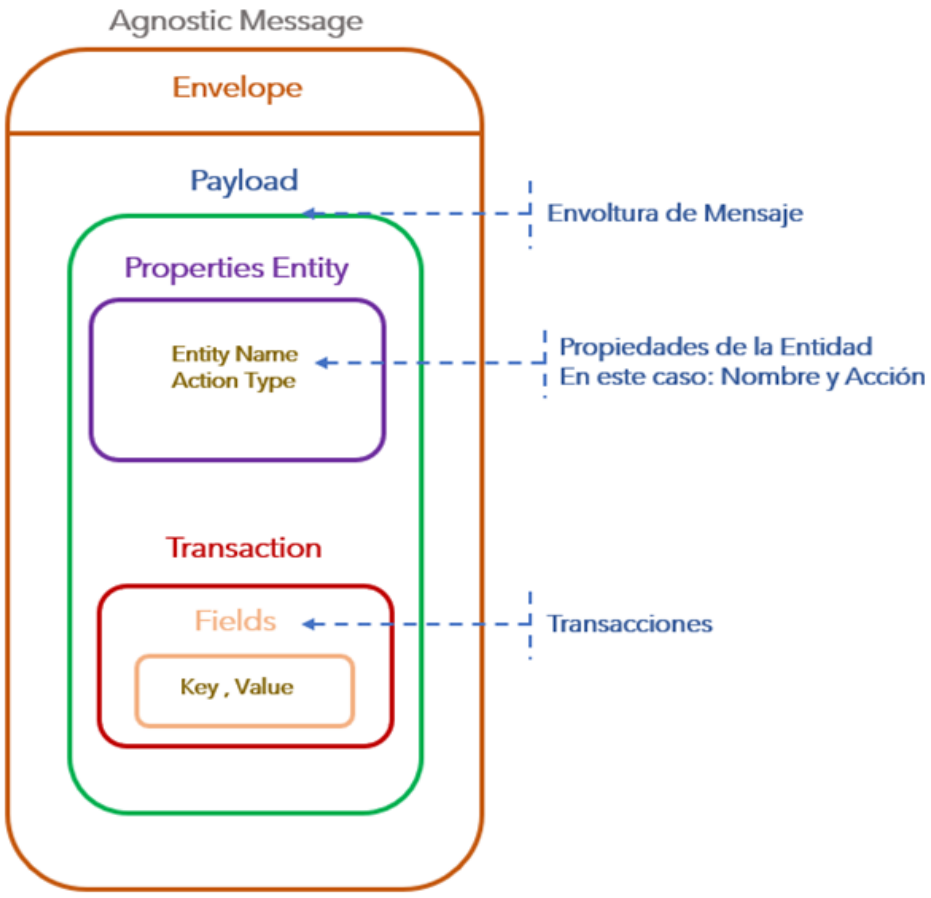


Figura 18. Representación gráfica de *Agnostic Message*.

El mensaje se utiliza de manera universal, es decir, puede ser usado para cualquier propósito al momento de agregar, modificar o eliminar información de alguna entidad en cualquier base de datos, también el poder ejecutar alguna función, procedimiento que se encuentre en las bases de datos, utilizando el mismo mensaje o envoltorio con la definición propuesta, por otro lado la traducción de este diseño se llevó a cabo en su forma de estructura de esquemas XSD (XML *Schema Definition*) para representación del contrato hacia el cliente dentro del WSDL (*Web Services Description Language*), en la Figura 19 se muestra dicha representación gráfica, la cual se detalla a continuación, en la sección de localización del servicio llamada *targetNamespace* que puede ser consumido desde la dirección web *http://message.cddm.mx/* y es en donde se encuentra el espacio de nombre. La sección *sendMessage* contiene todos los objetos que conforman el *Agnostic Message*, y utiliza la fachada *messageFacade* para acceder al esquema de cada componente que está bajo *transactionScheme*. En esta se encuentra alojado el *payload* o carga útil (*actionType, entityName, fields, key, value*). El *request* representado por *sendMessage* y *response* por *sendMessageResponse* del servicio quedan expuestos públicamente para ser usados por las diversas unidades de software, también está definida las posibles acciones de *actionType* (*SAVE, UPDATE, DELETE* y *PROCESS*).

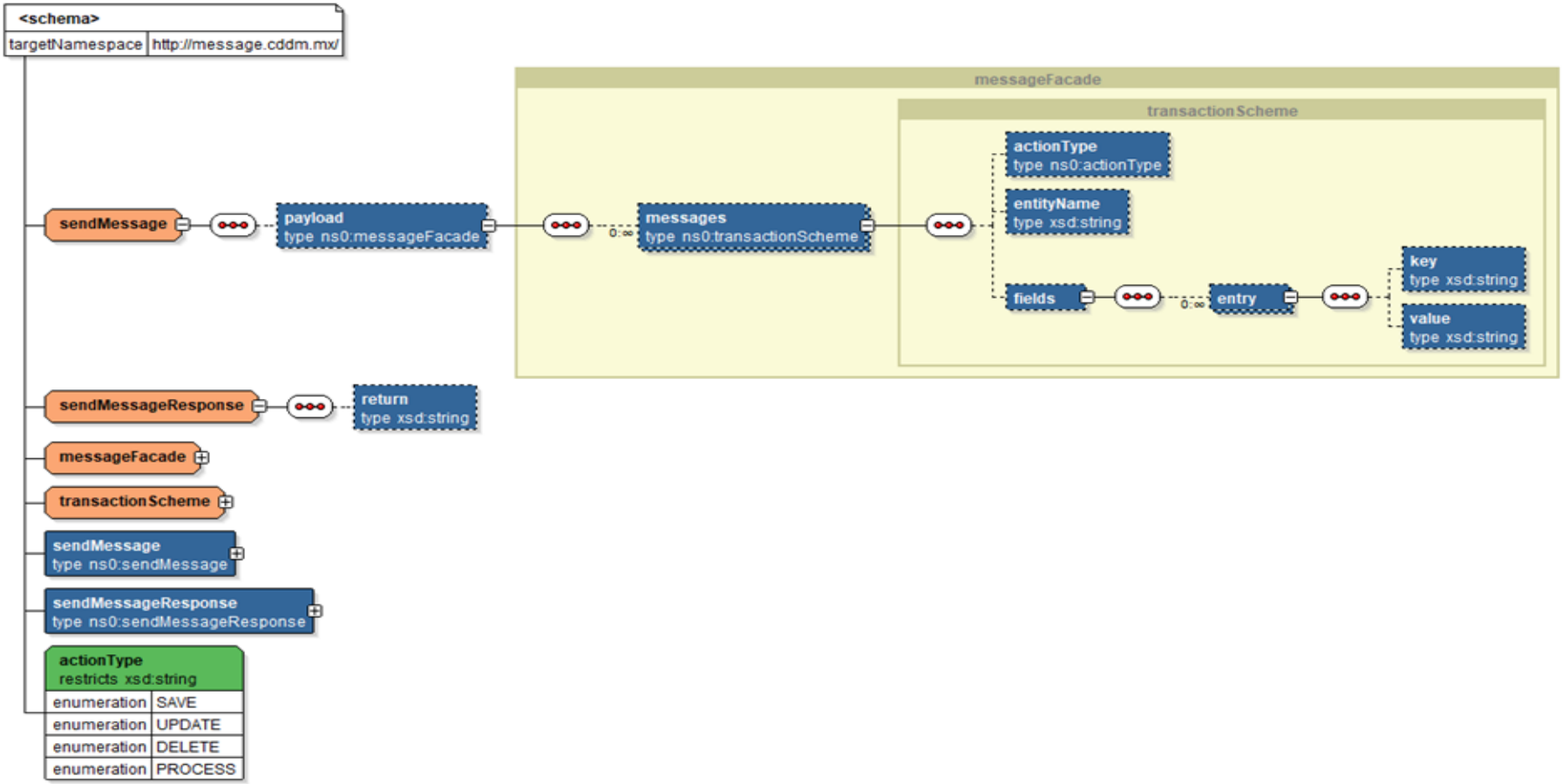


Figura 19. Representación *XSD* del *Agnostic Message*.

Por otro lado, la representación del *Agnostic Message* en el esquema de servicio hacia el cliente se muestra en la Figura 20, el cual está contenido en el WSDL expuesto para su consumo. Como se observa, el mensaje se crea en un archivo XSD que contiene etiquetas o *tags* predefinidas para cada elemento del documento. La etiqueta *types* muestra los tipos de datos a manejar y ayuda a dar forma al mensaje. La etiqueta *Message* sirve para contener el *request* y *response* donde viaja el *Agnostic Message*. La etiqueta *portType* o interface que contiene la indicación de las operaciones del documento *sendMessage* correspondiente al *request* y el *sendMessageResponse*. Finalmente, la etiqueta *binding* especifica que el protocolo de comunicación usado es SOAP.

```
<!--  Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.3.0-b170407.2038 svn-revision#2eaca54d17a59d265c6fe886b7fd0027836c766c.  -->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns/ws-policy"
xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tns="http://message.cddm.mx/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://message.cddm.mx/"
name="AgnosticMessageCDDMService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://message.cddm.mx/" schemaLocation="http://172.26.208.1:7101/cddm/AgnosticMessageCDDMPort?xsd=1"/>
    </xsd:schema>
  </types>
  <message name="sendMessage">
    <part name="parameters" element="tns:sendMessage"/>
  </message>
  <message name="sendMessageResponse">
    <part name="parameters" element="tns:sendMessageResponse"/>
  </message>
  <portType name="AgnosticMessageCDDM">
    <operation name="sendMessage">
      <input wsam:Action="http://message.cddm.mx/AgnosticMessageCDDM/sendMessageRequest" message="tns:sendMessage"/>
      <output wsam:Action="http://message.cddm.mx/AgnosticMessageCDDM/sendMessageResponse" message="tns:sendMessageResponse"/>
    </operation>
  </portType>
  <binding name="AgnosticMessageCDDMPortBinding" type="tns:AgnosticMessageCDDM">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="sendMessage">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service name="AgnosticMessageCDDMService">
    <port name="AgnosticMessageCDDMPort" binding="tns:AgnosticMessageCDDMPortBinding">
      <soap:address location="http://172.26.208.1:7101/cddm/AgnosticMessageCDDMPort"/>
    </port>
  </service>
</definitions>
```

Figura 20. Representación del *Agnostic Message* contrato WSDL.

Por último, se representa el *Agnostic Message* en la implementación y caso de estudio en la herramienta *SoapUI* en su versión 5.4.0 que se utilizó como ayuda para la ejecución de pruebas (ver Figura 21), donde se consume el contrato del documento *WSDL*.

La Figura 21 muestra una ventana de la herramienta SoapUI desarrollada en el lenguaje de programación Java. Se utilizó para realizar pruebas a los servicios web. En la imagen se observa la estructura del mensaje tal cual lo interpreta un ser humano pues contiene información capturada la cual se observa en las etiquetas *actionType*, *entityName*, *fields*, *key*, *value*. En la sección de ejecución, en la parte superior de la imagen se encuentra la URL o *endpoint* en donde se aloja el servicio web *AgnosticMessageCDDMPort* y desde ahí es ejecutada la prueba.

Figura 21. Representación del consumo de *Agnostic Message* usando *SoapUI*.

4.4 Composite Envelope

Una vez definido el *Agnostic Message*, se diseñó y se creó el compuesto *Payload* (*árbol*) a partir del componente *Envelope*, este patrón permite construir objetos complejos, es decir, mediante estructuras de arreglos básicos y recursividad se crean estructuras de árboles compuestos a partir de componentes más simples representados como hojas del árbol que heredan funciones y propiedades de la primer estructura básica y es extendida a todos sus nodos, lo cual ayuda a simplificar el tratamiento de los objetos creados mediante una sola interfaz en común, de esta manera todos los objetos se gestionan de igual forma para dar soporte al mensaje enviado por el cliente. El componente *Envelope* es una clase abstracta que contiene todas las propiedades de la *Entidad* (*Entity*), incluye los procedimientos y métodos que fueron usados por el compuesto *Payload* y también el compuesto *Message* (*hoja*), que son gestionados de la misma forma por el patrón *Strategy* que se explicará en las subsecciones siguientes de este capítulo.

En la Figura 22, se muestra la representación del patrón *Composite* implementado en el *Agnostic Message*. Este patrón está conformado por la clase abstracta *Envelope* y cuenta con dos propiedades *tipoNodo* que indica si es una estructura árbol o una hoja de esa estructura. El segundo es *Transaction* el cual está constituido por *actionType* que indica la acción o transacción que se ejecutará en la entidad, para este caso puede ser *SAVE, UPDATE, DELETE* y *PROCESS*. En este sentido, si el valor de *actionType* es *SAVE* entonces se agregará un nuevo registro en la entidad *entityName*. En este orden de ideas, la propiedad *entityName* representa el nombre de la entidad a ser afectada, también contiene un conjunto de instancias denominadas *Fields* mediante el objeto MAP que almacena los registros campos-valor que serán agregados, eliminados, actualizados o consultados de una entidad determinada. Este componente cuenta con la clase *Payload* que representa el contenido útil del mensaje, la función de esta clase es generar una estructura de transacciones que pueden afectar a una o varias entidades. Además, contiene seis métodos que pueden ser utilizado para administrar la estructura, una de ellas es *addEnvelope* y como parámetro de entrada recibe un objeto de tipo *Envelope*, el cual sirve para agregar un nodo u hoja al árbol con sus propiedades. El método *removeEvelope* remueve o elimina una hoja (nodo) del árbol y *getEnvelope* recupera una hoja o conjunto de hojas (consultar Capítulo II subsección 2.10). Por último, el método *invokeStrategy* es el más importante debido a que ayuda a invocar al componente *Strategy* por medio del contexto *Context* que crea el acceso a todos los algoritmos que serán usados para el tratamiento de los mensajes enviados por los clientes.

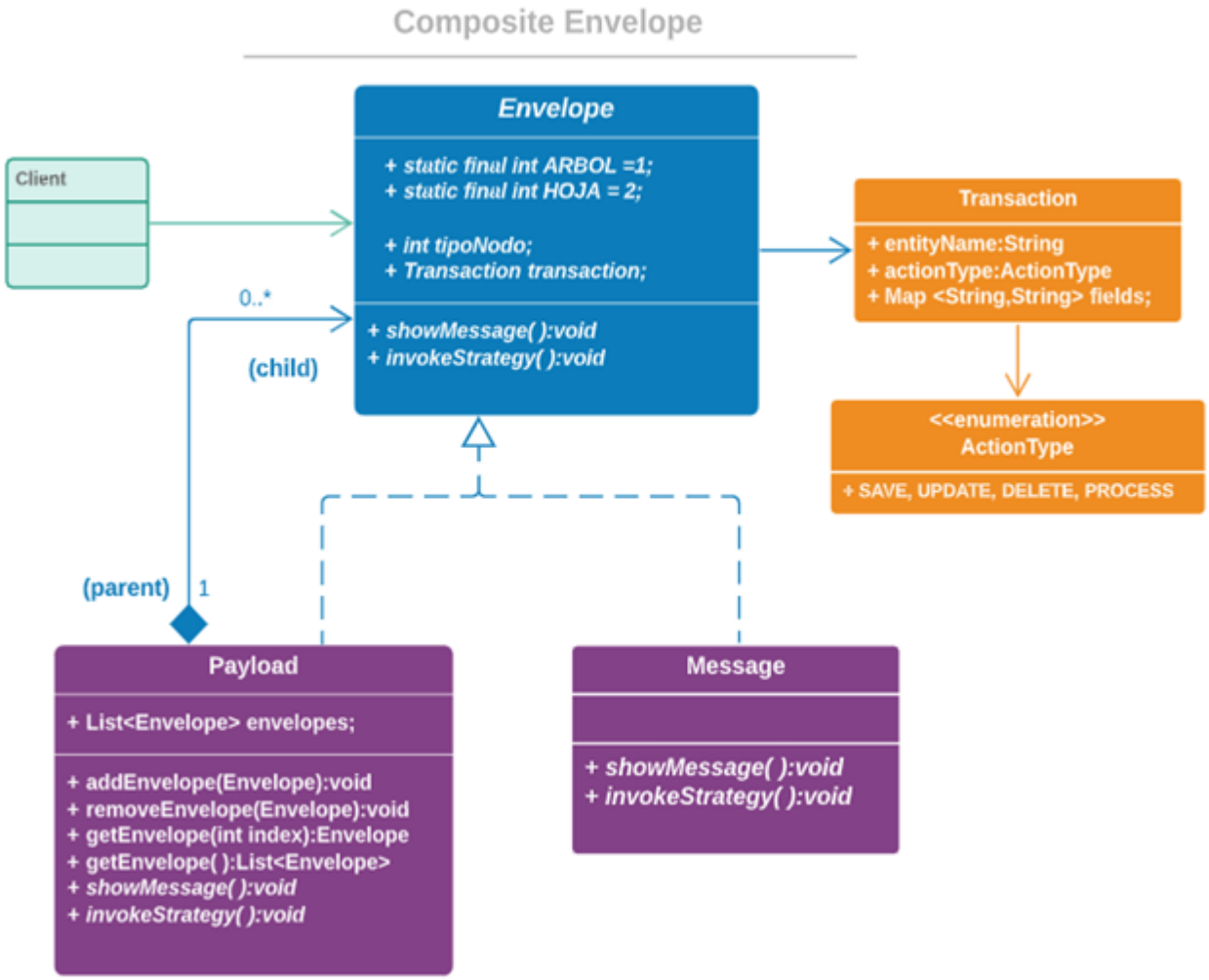


Figura 22. Representación de la implementación del compuesto *Agnostic Message*.

4.5 StrategyCDDM

Un componente fundamental en la arquitectura propuesta en esta tesis es el patrón *Strategy* (*StrategyCDDM*), el cual consiste en una interfaz hacia los clientes que utilizan los algoritmos diseñados para la gestión del mensaje agnóstico, el uso de este patrón se lleva a cabo mediante un contexto representado por el elemento *Context* como interfaz hacia el cliente *Composite Payload*.

Los algoritmos creados para esta solución *StrategyDML* y *StrategyProcess* representan un ejemplo de la diversidad de algoritmos que pueden ser definidos e implementados como estrategia en cada integración. Esto permitirá, como ventaja, que la arquitectura sea capaz de tomar decisiones según sea el mensaje entrante.

También se crearon componentes para acceder a los metadatos de la base de datos como parte de *StrategyCDDM*, con el propósito de crear las entidades dinámicamente según el mensaje entrante y generar el recurso para su persistencia. En este sentido, se definió otro método para implementar la persistencia del mensaje según la entidad examinada, representada y persistida en los almacenes de datos. De esta forma se concluye con el objetivo de implementar y gestionar los mensajes *Agnostic Message* representados en el patrón *Composite*. A continuación, se muestra la Figura 23, donde se implementa el patrón *Strategy* en la arquitectura propuesta en esta tesis.

La Figura 23 describe las clases que componen e implementa el patrón *Strategy*, una de ellas es la interfaz *StrategyCDDM* que declara el método *manageEnvelope*. La clase es implementada en los diversos algoritmos, en este caso *StragyDML* y *StrategyProcess*, para el caso del primero agrega los métodos *generateEntity*, *executeProcess*, *createQueryInsert* y *formatField* como métodos privados y auxiliares. Ambos se encargan de crear y validar la instrucción DML (*Data Manipulation Language*), para después ser ejecutada con el método *executeProcess*. Por otro lado, el algoritmo *StragyProcess* no tiene ninguna estrategia en este momento, se creó con el fin de que más adelante se pueda construir la ejecución de consultas y ejecución de *store procedures* y/o *pl-sql* según sea el motor de base de datos donde se implemente. Los clientes pueden acceder a los algoritmos por medio de la clase *Context* que tiene como función generar el acceso al componente *composite* descrito en este capítulo en la subsección 4.4 que habla del funcionamiento y gestión de los mensajes.

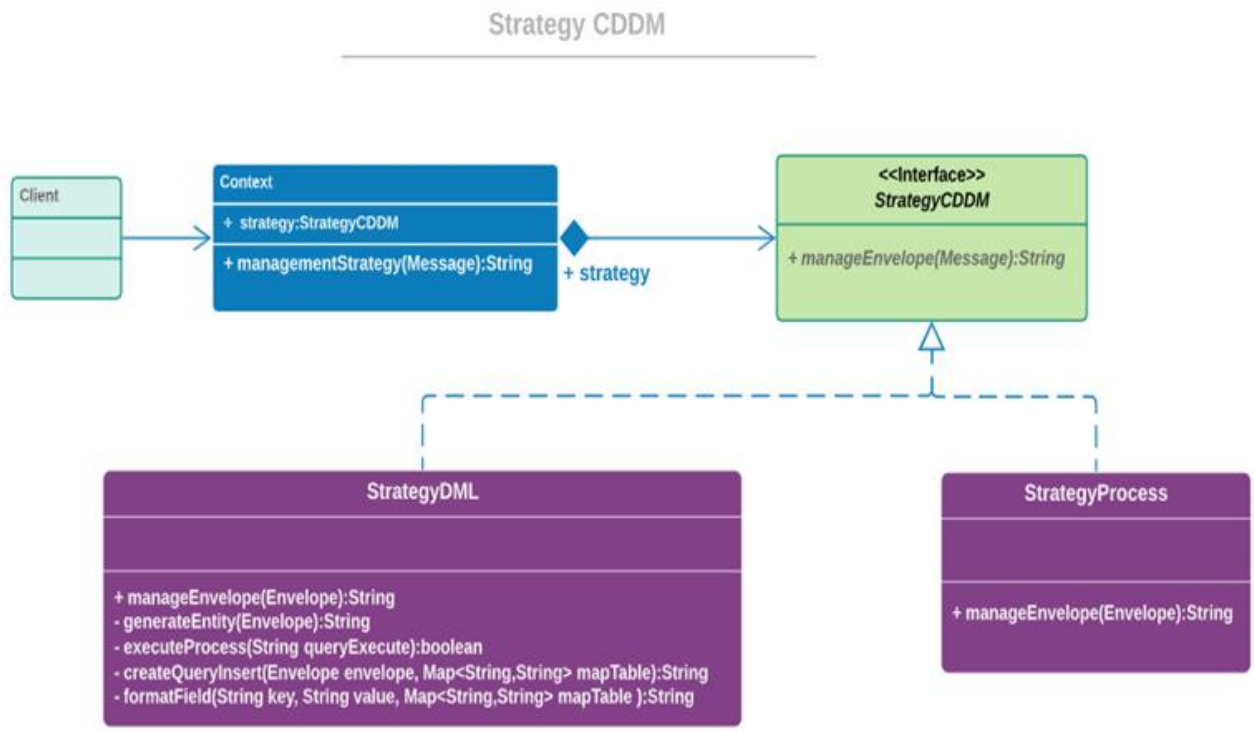


Figura 23. Representación de la implementación de estrategia *StrategyCDDM*.

4.6 Funcionalidad

Para llevar a cabo la integración de unidades de software mediante *Agnostic Message* con el fin de mejorar el bajo acoplamiento, a continuación, se explica la funcionalidad de la propuesta. Se iniciará analizando la Figura 24, en la cual se observa el diagrama de secuencia donde se comunican los tres componentes que conforman la arquitectura.

En la Figura 24 se muestra como un cliente inicia la integración mediante la arquitectura propuesta en esta tesis. El proceso inicia cuando el cliente envía un mensaje dirigido al componente *Agnostic Message*, lo cual se lleva a cabo mediante la operación *sendMessage*. Como parámetro de entrada se envía un objeto de tipo *MessageFacade* de nombre *Payload* que representa el compuesto (*Composite*) del componente *Envelope* donde se aloja el mensaje. El mensaje de tipo *Composite Payload* se divide en dos partes importantes, la primera es la sección *Properties Entity* y *Transaction Entity* como se mostró en la Figura 18 de la sección 4.3 de este capítulo. En la primera

sección del mensaje se aloja el nombre de la entidad que se buscará en la *metadata* del repositorio o base de datos, este será el punto de partida para que se cree la instancia o la secuencia de lenguaje *DML* (*Data Manipulation Language*). Esto permitirá ejecutar la orden de acuerdo con la propiedad *Action Type*, la cual indica la acción a realizar en el repositorio de la base de datos. En la segunda parte, se encuentran las transacciones que son representadas por una estructura de datos MAP. Estas estructuras permiten gestionar los campos y sus respectivos valores, lo que posibilita que sean almacenados en pares de "*key*/*value*", en donde *key* es el campo de la entidad y *value* es el valor del campo. Una vez enviado el mensaje por el cliente, se accede desde el compuesto *Payload* al patrón *Strategy* con la implementación *StrategyCDDM* en donde se encuentran alojados los algoritmos que darán solución a la petición del mensaje. Este acceso se proporciona mediante una interfaz llamada *Context*, la estrategia que se tomará de acuerdo con la propiedad *Action Type*, ahí es donde se crea una nueva instancia del algoritmo a implementar. *StrategyDML* y *StrategyProcess* atenderán el mensaje y darán solución a la integración con la operación *managementStrategy*, que tiene como parámetro de entrada el *composite Message*. Una vez generada la entidad y persistido el mensaje se da la respuesta al cliente indicando si su ejecución fue correcta o existe algún problema.

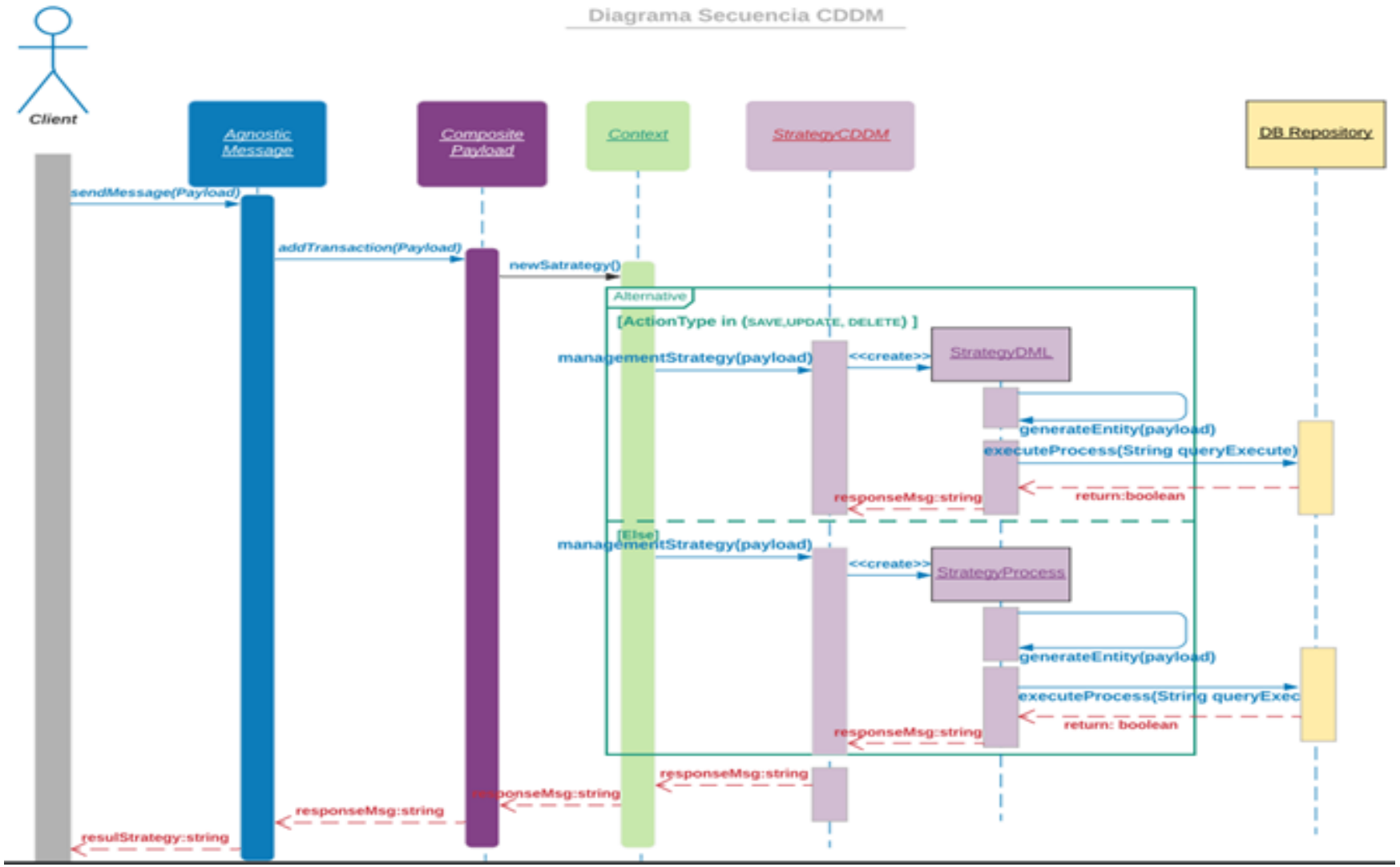


Figura 24. Diagrama de secuencia de la arquitectura CDDM.

De acuerdo con la Figura 24, cuando un cliente inicia la integración mediante la arquitectura propuesta en esta tesis, primero envía un mensaje dirigido al componente *Agnostic Message*, esto se lleva a cabo mediante la operación *sendMessage*, como parámetro de entrada se envía un objeto de tipo *MessageFacade* de nombre *Payload* que representa el compuesto (*Composite*) del componente *Envelope* donde se aloja el mensaje. El mensaje de tipo *Composite Payload* se divide en dos partes importantes, la primera es la sección *Properties Entity* y *Transaction Entity* como se mostró en la Figura 18 de la sección 4.3 de este capítulo. En la primera sección del mensaje se aloja el nombre de la entidad que se buscará en la *metadata* del repositorio o base de datos, este será el punto de partida para que se cree la instancia o la secuencia de lenguaje *DML* (*Data Manipulation Language*). Esto permitirá ejecutar la orden de acuerdo con la propiedad *Action Type*, la cual indica la acción a realizar en el repositorio de la base de datos. En la segunda parte, se encuentran las transacciones que son representadas por una estructura de datos MAP, las cuales permiten gestionar los campos y sus respectivos valores, lo que posibilita que sean almacenados en pares de "*key*/*value*", en donde *key* es el campo de la entidad y *value* es el valor del campo. Una vez enviado el mensaje por el cliente, se accede desde el compuesto *Payload* al patrón *Strategy* con la implementación *StrategyCDDM* en donde se encuentran alojados los algoritmos que darán solución a la petición del mensaje. Este acceso se proporciona mediante una interfaz llamada *Context*, la estrategia que se tomará de acuerdo con la propiedad *Action Type*, ahí es donde se crea una nueva instancia del algoritmo a implementar, para fines ilustrativos, se encuentran *StrategyDML* y *StrategyProcess* que atenderán el mensaje y darán solución a la integración con la operación *managementStrategy*, que tiene como parámetro de entrada el *composite Message*. Una vez generada la entidad y persistido el mensaje se da la respuesta al cliente indicando si su ejecución fue correcta o existe algún problema.

4.7 Ejemplo de Aplicación

En este apartado se detalla un ejemplo de aplicación de la arquitectura MCDD en un entorno real, se explica paso a paso cómo se realizó la implementación y se valida su resultado (ver subsección 4.8 de este capítulo).

La empresa seleccionada para la implementación y prueba de la arquitectura MCDD en un entorno real fue Paquetexpress (https://www.paquetexpress.com.mx). Esto se debió a las facilidades otorgadas a razón de la relación laboral existente. La empresa Paquetexpress fue fundada en el año de 1986 y cuenta con más de 8,000 empleados en 20 departamentos, el giro principal es la logística y el núcleo de este es la recolección, documentación, envío y entrega de paquetes principalmente en México y también alrededor del mundo. La oficina central de administración se encuentra en la ciudad de Los Mochis, en el municipio de Ahome, Sinaloa.

En el área Comercial, que es de gran importancia para la empresa, se presentó el siguiente problema: cómo controlar y registrar los convenios y acuerdos con clientes (descuentos, promociones y tarifas) al momento de que solicitan los servicios ya mencionados, así como seguimiento a clientes, ventas y comisiones de los ejecutivos. Así mismo, se presentaba un problema asociado con respecto a cómo afecta la cartera de crédito y contabilización de los ingresos de cada uno de ellos. Ante esta situación, se entabló comunicación con el cuerpo directivo de las áreas de dirección general, dirección comercial, dirección de administración, dirección de operaciones y la dirección de TI con la finalidad de proponer una solución a través de la arquitectura MCDD. También se analizó el tema ante los arquitectos de tecnología para ver la viabilidad de la implementación de la arquitectura de integración presentada en esta tesis para paliar la problemática encontrada. De tal forma que, se obtuvo la aprobación.

Posteriormente, se definió el plan de trabajo para el proceso de integración quedando establecido como lenguaje de programación para implementar la arquitectura propuesta el lenguaje *Java*. Este es uno de los más utilizados en la industria de aplicaciones empresariales fuera de la academia. Después, se procedió con la definición de los sistemas software a integrar, entre los cuales destacaron las plataformas CRM *Salesforce Cloud* y las financieras cómo es el ERP *Oracle Cloud*, así como de todos los puntos de venta que abarca la documentación en línea, piso y servicios web de integración B2B (*business to business*) de los clientes. La solución a la problemática de la empresa Paquetexpress mediante la arquitectura MCDD se describe a continuación:

La implementación de la arquitectura se divide en cinco pasos, cada uno de ellos se listan a continuación:

i) Creación de los componentes *Envelope*, *Payload y Messages*.
ii) Generación de la estructura *Transaction* (MAP).
iii) Generación del mensaje *Agnostic Message* (*MessageFacade*).
    a. Carga del componente *Payload* con hojas *Message*.
iv) Generación del componente *strategy* (*StrategyCDDM*) algoritmos.
    a. Generación del algoritmo *StrategyDML*.
    b. Generación del algoritmo *StrategyProcess*.
v) Generación y publicación del servicio *Agnostic Message* mediante la clase *AgnosticMessageCDDM*.

Para consultar el detalle de cada uno de los cinco puntos establecidos para la implementación de la arquitectura MCDD se puede consultar la sección 4.2 (Estructura de la arquitectura MCDD) de

esta tesis. En la siguiente subsección se presenta la descripción y el código de cada paso en la implementación de la arquitectura antes citada.

4.7.1 Creación de los componentes *Envelope, Payload y Message*

Para crear la envoltura del mensaje del elemento *composite* (*Envelope*) se usó una clase abstracta que representa la base de las propiedades y funciones que fueron extendidas a los elementos *Payload y Messages* requeridos para generar el núcleo del mensaje conteniendo el grupo de transacciones a ejecutar a través de la estructura MAP y las propiedades de las entidades (ver Figura 18). En el código fuente mostrado en la Figura 25 se observa la clase abstracta *Envelope* que fue la base para crear el resto de componentes, por otro lado, la Figura 26 muestra el componente *Payload* extendiendo las propiedades y funciones de dicho compuesto. Finalmente, en la Figura 27 se aprecia que también es creado el compuesto *Message* el cual se extiende del mismo componente, de esta manera se crea un árbol con sus hojas (nodos) preparadas para ser cargadas con las instrucciones a ejecutar en los algoritmos gestionados por el patrón *strategy*.

```java
package mx.cddm.composite.component;

import ...;

/**
 * Clase Abstracta Envelope sirve como base extendida para el componente Composite Payload y Message:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

public abstract class Envelope {

    public static final int ARBOL =1;
    public static final int HOJA = 2;

    protected int tipoNodo;
    protected Transaction transaction;

    public abstract void showMessage();
    public abstract String invokeStrategy();


    public void setTransaction(Transaction transaction) {
        this.transaction = transaction;
    }

    public Transaction getTransaction() {
        return transaction;
    }

    public void setTipoNodo(int tipoNodo) {
        this.tipoNodo = tipoNodo;
    }
}
```

Figura 25. Clase abstracta componente *Envelope*.

```java
package mx.cddm.composite.composite;

import ...;

/**
 * Clase Payload Arbol extendido de Envelope gestion de Message class:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

public class Payload extends Envelope{

    List<Envelope> envelopes = new ArrayList<Envelope>();

    /**
     * Constructor para carga de transacciones Transaction
     * @param transaction El parámetro transaction hace la carga de campo-valor
     */
    public Payload(Transaction transaction){
        this.setTransaction(transaction);
        this.setTipoNodo(ARBOL);
    }

    public void addEnvelope( Envelope envelope )
    {
           envelopes.add( envelope );
    }

    public void removeEnvelope( Envelope envelope )
    {
            envelopes.remove( envelope );
    }
```

Figura 26. Clase *Payload* extendida del componente *Envelope*.

```java
package mx.cddm.composite.composite;

import ...;

/**
 * Clase Message hoja extendido de Envelope gestion de Agnostic Message:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

public class Message extends Envelope {

    public Message(Transaction transaction){
        this.setTransaction(transaction);
        this.setTipoNodo(HOJA);
    }

    /**
     * Método showMessage que imprime los valores del mensage
     * @return void
     */
    @Override
    public void showMessage() {
        System.out.println( "---------------------- Message --------------------------" );
        System.out.println( "Entity: [" + this.getTransaction().getEntityName() + "]" );
        System.out.println( "Action Type: [" + this.getTransaction().getActionType() + "]" );
        System.out.println( "---------------------- Transactions --------------------------" );

        Iterator it =  this.getTransaction().getFields().keySet().iterator();
        while(it.hasNext()){
            String key = (String) it.next();
            System.out.println("Field: " + key + " -> Value: " + this.getTransaction().getFields().get(key));
        }

    }
```

```java
    /**
     * Método invokeStrategy que interactua con Clase Strategy para aplicar algoritmo
     * @return retorna "Ok" si fue exitoso, caso contrario el error que se produce, ademas de la impresión del error del codigo
     */
    @Override
    public String invokeStrategy() {
        String resulStrategy = "Ok";
        Context context = null;

        switch ( this.getTransaction().getActionType()){

                        case SAVE:
                        case UPDATE:
                        case DELETE:
                            context = new Context(new StrategyDML());
                            resulStrategy = context.managementStrategy(this);
                            break;
                        case PROCESS:
                            context = new Context(new StrategyProcess());
                            resulStrategy = context.managementStrategy(this);
                            break;
                        default:
                            resulStrategy = "Acción inexistente revise Action Type ...";
                            break;
        }

        return resulStrategy;
    }


}
```

Figura 27. Clase *Message* extendida del componente *Envelope*.

4.7.2 Generación de la estructura Transaction (MAP)

Después de haber construido los componentes que almacenan las propiedades y transacciones a ejecutar, se creó la clase *Transaction* como parte del mensaje. En la Figura 28 se detalla la clase que contiene las propiedades *ActionType* y *entityName* como parte importante del mensaje, para el caso de *fields* contiene los campos y valores en una estructura MAP que son usados en la implementación del *Agnostic Message.* Estos campos fueron consumidos por los clientes que integraron información a las diversas plataformas existentes.

```
/**
 * Clase Transaction nos ayuda a recolectar la información para procesar:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

public class Transaction {

     private Map <String,String> fields;
     private ActionType actionType;
     private String entityName;

     public void setFields(Map<String, String> fields) {
          this.fields = fields;
     }

     public Map<String, String> getFields() {
          return fields;
     }

    public void setActionType(ActionType actionType) {
        this.actionType = actionType;
    }

    public ActionType getActionType() {
        return actionType;
    }

    public void setEntityName(String entityName) {
        this.entityName = entityName;
    }

    public String getEntityName() {
        return entityName;
    }
}
```

Figura 28. Clase *Transaction* y estructura MAP.

4.7.3 Generación del mensaje *Agnostic Message* (*MessageFacade*)

En esta sección se describe la generación del mensaje mediante la clase *MessageFacade* y se estructura cada una de sus partes antes generadas (ver sección 4.7.1 de este capítulo), como lo son los componentes *Payload* y *Messages*. Por otro lado, con la ayuda de la clase *TransactionScheme* se expone una estructura fachada hacia el cliente que dará entrada a los datos MAP y la propiedad de *EntityName* y *ActionType* como parte de la carga útil (*payload*), en la Figura 29 se muestra la clase *MessageFacade* y su estructura.

```java
package mx.cddm.message;

import ...;
/**
 * Clase MessageFacade clase para generar el objeto de entrada de los datos de las Unidades de Software:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

public class MessageFacade {

    private List<TransactionScheme> messages;


    public void setMessages(List<TransactionScheme> messages) {
        this.messages = messages;
    }

    public List<TransactionScheme> getMessages() {
        return messages;
    }

}
```

```java
/**
 * Clase TransactionScheme Nos ayuda a generar el esquema para ser usado de fachada y recolectar los datos:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

public class TransactionScheme {
    private Map <String,String> fields;
    private ActionType actionType;
    private String entityName;

    public void setFields(Map<String, String> fields) {
        this.fields = fields;
    }

    public Map<String, String> getFields() {
        return fields;
    }

    public void setActionType(ActionType actionType) {
        this.actionType = actionType;
    }

    public ActionType getActionType() {
        return actionType;
    }

    public void setEntityName(String entityName) {
        this.entityName = entityName;
    }

    public String getEntityName() {
        return entityName;
    }

}
```

Figura 29. Clase *MessageFacade* parte de propiedades del *Agnostic Message*.

4.7.4 Generación del componente *strategy* (*StrategyCDDM*) algoritmos

En esta sección se expone a detalle el patrón *strategy* junto con sus algoritmos. Para continuar con la integración, se procedió con la implementación de la *interfaz StrategyCDDM*, la Figura 30 muestra las operaciones necesarias para dar gestión al mensaje *Agnostic Message* según sea la estrategia adoptada por *Action Type*.

```
package mx.cddm.strategy.context;

import ...;
/**
 * Interface StrategyCDDM contiene los metodos a ser implementados en los algoritmos:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

public interface StrategyCDDM {

    public String manageEnvelope(Message msg);

}
```

Figura 30. Interfaz *StrategyCDDM*.

Para acceder a los algoritmos que dan gestión a los mensajes se usa el componente *Context* que crea el contexto según sea la estrategia seleccionada en tiempo de ejecución mediante la operación *managementStrategy*. Esta operación tiene como parámetro de entrada un objeto *Message*, ver componente *Context* (Fig. 31). El código a implementar se muestra en la Figura 31, la explicación sobre cada uno de estos componentes se encuentra en la subsección 4.5 *StrategyCDDM* de este capítulo de la tesis.

```java
package mx.cddm.strategy.context;

import ...;

/**
 * Clase Context clase que ayuda a crear el contexto en la estrategia del algoritmo a ejecutar:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

public class Context {
    StrategyCDDM strategy;

    public Context(StrategyCDDM strategy){
        this.strategy = strategy;
    }

    public void setStrategy(StrategyCDDM strategy){
        this.strategy = strategy;
    }

    public String managementStrategy(Message msg){
        return strategy.manageEnvelope(msg);
    }

}
```

Figura 31. *Context* acceso hacia algoritmos de *StrategyCDDM*.

Los algoritmos que son accedidos por *Context* y están dando gestión a los mensajes entrantes para esta implementación son *StrategyDML* y *StrategyProcess*, ambos son extendidos por la interfaz *StrategyCDDM* mediante la implementación de sus operaciones, en la Figura 32 podemos ver la estrategia implementada para el mensaje *Agnostic Message*. Dicha estrategia fue generar consultas a las entidades y ejecución de una acción de manipulación de datos DML (*Data Manipulation Language*, por sus siglas en inglés).

```java
package mx.cddm.strategy.algorithm;

import ...;

/**
 * Clase StrategyDML contiene el algoritmo DML para el trabajo con la Base de Datos que implementa la interface StrategyCDDM:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

public class StrategyDML implements StrategyCDDM {

    Message message;
    Connection con = null;

    public StrategyDML() {
        super();
    }

    /**
     * Método manageEnvelope nos ayuda a gestionar el mensaje ante la base de datos de Postgress
     * @param msg Este parametro del tipo Message que contiene los datos para ser gestionados en la BD de Postgress
     * @return responseMsg retorna "Ok" si fue exitoso, caso contrario el error que se produce, ademas de la impresión del error del codigo
     */

    @Override
    public String manageEnvelope(Message msg) {
        String responseMsg = "Ok";
        System.out.println("-> Entering a DML Strategy ...");
        responseMsg = this.generateEntity(msg);
        return responseMsg;
    }
```

```
    * Método generateEntity nos ayuda a gestionar el mensaje segun sea Action Type e invoca la ejecución de la cadena DML
    * @param msg Este parametro del tipo Message que contiene los datos para ser gestionados en la BD de Postgress
    * @return query retorna "Ok" si fue exitoso, caso contrario el error que se produce, ademas de la impresión del error del codigo
    */

   private String generateEntity(Message msg) {
       String query = "Ok";
       AccessMetadata md = new AccessMetadata();
       try{
           Map<String,String> mapTable = md.accessData(msg.getTransaction().getEntityName());
           if ( mapTable.size()>0){
             switch ( msg.getTransaction().getActionType()){
               case SAVE:
                 query = this.createQueryInsert(msg, mapTable);
                 System.out.println(query);

                 if (this.executeProcess(query)){
                      query = "!! Fields were inserted ...";
                 }else{
                      query = "!! Some error occurred ...";
                 }

               break;
               case UPDATE:
               break;
               case DELETE:
               break;
             }
           }else{
               query = "!! Entity does not exist ...";
           }
       }catch(Exception e){
           query = "Transaction Error ...";
           e.printStackTrace();
       }
       return query;
   }
```

```java
    /**
     * Método executeProcess invoca la ejecución de proceso ante la base de datos, haciendo la conexion y despues ejecutando
     * @param queryExecute Este parametro es la cadena DML a ejecutar
     * @return resulExecute retorna True si fue exitoso, caso contrario False, ademas de la impresión del error del codigo
     */

    private boolean executeProcess(String queryExecute) {
        DataSourceConexion dataSource = DataSourceConexion.getConexion();
        ResultSet tablas = null;
        PreparedStatement pstm = null;
        boolean resulExecute = false;
        try {
            con = dataSource.Conecta("jdbc/postgress");
            pstm = con.prepareCall(queryExecute);
            resulExecute = pstm.execute();
            resulExecute = true;
        }catch(Exception e){
            e.printStackTrace();
        }finally{
            try{
            if (pstm != null){ pstm.close(); pstm = null;}
            if (tablas != null){ tablas.close(); tablas = null;}
            if (con != null){ con.close(); con = null;}
            }catch(Exception ex){
                ex.printStackTrace();
            }
        }
        return resulExecute;
    }
```

```java
/**
 * Método createQueryInsert genera el query a ejecutar con los datos que viajan en el Message
 * @param msg Este parametro contiene los datos a ser ejecutados ante la BD
 * @param mapTable datos de la metadata para revisión de los campos del MAP key-value o campo-valor
 * @return queryReturn retorna cadena ya construida para ser ejecutada por el metodo executeProcess
 */
private String createQueryInsert(Message msg, Map<String,String> mapTable){
    String queryValues = null;
    String queryFields = null;
    String fieldInsert = "(";
    String valueInsert = "(";
    String key = "";
    String new_value = "";
    String queryReturn = null;
    Iterator it =  msg.getTransaction().getFields().keySet().iterator();
    Map<String,String> fields = msg.getTransaction().getFields();
        while(it.hasNext()){
            key = (String) it.next();
            new_value = this.formatField(key,fields.get(key), mapTable);
            fieldInsert = fieldInsert + key + ",";
            valueInsert = valueInsert + new_value + ",";
       }
        StringBuilder cadFinalfi = new StringBuilder(fieldInsert);
        cadFinalfi.setCharAt(fieldInsert.length()-1, '%');
        queryFields = cadFinalfi.toString();
        queryFields = queryFields.replace("%", "");
        queryFields = queryFields + ")";
        StringBuilder cadFinalvi = new StringBuilder(valueInsert);
        cadFinalvi.setCharAt(valueInsert.length()-1, '%');
        queryValues = cadFinalvi.toString();
        queryValues = queryValues.replace("%", "");
        queryValues = queryValues + ")";
        queryReturn = "insert into public."+'"' + msg.getTransaction().getEntityName()+'"'+ " "+queryFields+ " values " + queryValues + ";";
    return queryReturn;
}
```

```java
/**
 * Método formatField nos ayuda a validar los valores si el tipo es numerico para su conversion de cadena a numerico
 * @param key Este parametro contiene el dato llave o campo
 * @param value Este parametro contiene el dato valor del campo
 * @param mapTable datos de la metadata para revisión de los campos del MAP key-value o campo-valor
 * @return result retorna el valor tipo numerico o de cadena
 */

private String formatField(String key, String value, Map<String,String> mapTable ){
    String result = "";
    boolean exit = true;
    Iterator it = mapTable.entrySet().iterator();
    if(mapTable.containsKey(key)){
    while (it.hasNext()&&exit) {
        Map.Entry e = (Map.Entry)it.next();
        if(e.getKey().toString().equals(key)){
            if (e.getValue().toString().equals("numeric")){
                result = value;
                exit = false;
            }else{
                result = "'" + value + "'";
                exit = false;
            }
        }
    }
    }
    return result;
}
```

Figura 32. Algoritmo *StrategyDML* implementando interfaz *StrategyCDDM*.

La Figura 33 propone una estrategia de ejecución de procedimientos o funciones de la base de datos, estas son diseñadas de acuerdo con la necesidad de quien implementa la arquitectura.

```java
package mx.cddm.strategy.algorithm;

import ...;

/**
 * Clase StrategyProcess contiene el algoritmo Process no implementado en este caso:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

public class StrategyProcess implements StrategyCDDM{
    public StrategyProcess() {
        super();
    }

    @Override
    public String manageEnvelope(Message msg) {
        String responseMsg = "Ok";
        System.out.println("-> Entering a Strategy Process ...");
        msg.showMessage();
        return responseMsg;
    }

}
```

Figura 33. Algoritmo *StrategyProcess* implementando interfaz *StrategyCDDM*.

4.7.5 Publicación del servicio *AgnosticMessageCDDM*

Continuando con la implementación de la arquitectura MCDD en el entorno empresarial de Paquetexpress, se construyó el servicio web en el formato SOAP. En la Figura 34 se observa una representación gráfica del WSDL que describe la interfaz del servicio. En la sección de *Port Types* se muestra las operaciones *sendMessage* y *sendMessageResponse* como parte del *request* y *response* de este. En la sección *Bindings* nos indica el uso del protocolo SOAP como medio de comunicación para *AgnosticMessageCDDM* y por último la sección *Services* hace referencia a las secciones anteriores donde expone los puertos y direcciones que localizan al servicio.

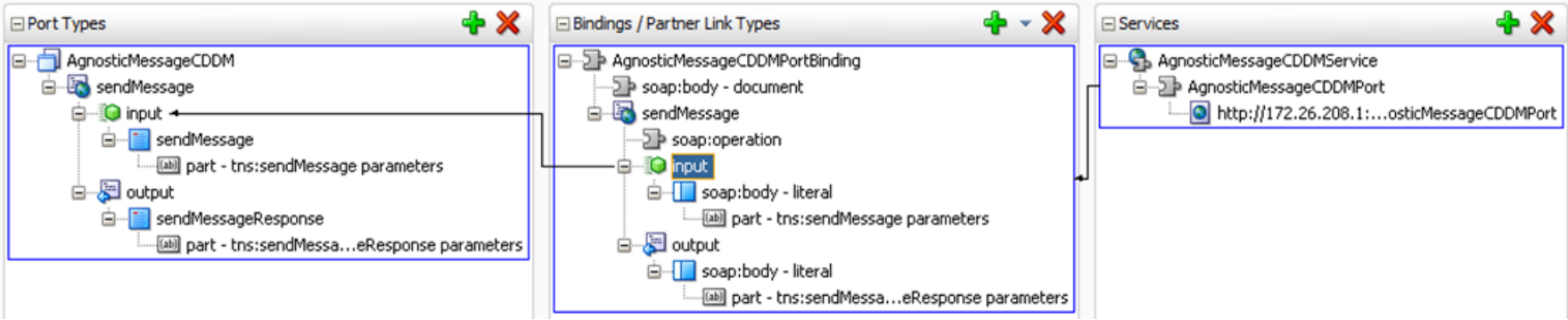


Figura 34. Servicio web SOAP *AgnosticMessageCDDM*.

En la Figura 35 se puede observar la clase codificada en el lenguaje Java que implementa el servicio web *AgnosticMessageCDDM* mediante el método *sendMessage* que recibe el mensaje para ser gestionado. En la Figura 36 se muestra el proyecto finalizado con todos los componentes y patrones a implementar bajo el paquete principal *mx.cddm*, sobresaliendo de ellos *mx.cddm.component* y *mx.cddm.composite* que contienen la estructura donde viaja, las instrucciones de la gestión del mensaje, así como la información. El paquete *mx.cddm.message* contiene el punto de entrada del mensaje para los aplicativos que lo utilizarán, y *mx.cddm.strategy* contiene los algoritmos que se implementarán para dar gestión al mensaje junto con ello la clase de acceso a estos.

```java
package mx.cddm.message;

import ...;

/**
 * Clase AgnosticMessageCDDM clase principal que es expuesto como servicio web para ser consumido:.
 * @author: Juan Antonio Ruiz Ceniceros:.
 * @version: 06/04/2021/Beta-A:.
 * @see <a href = "https://ruce.wordpress.com/about/" /> Arquitectura TECH </a>
 */

@WebService
public class AgnosticMessageCDDM {

    /**
     * Método sendMessage Es la puerta de entrada de los datos de los clientes o Unidades de Software que estan conectados
     * @param payload Este parametro del tipo MessageFacade nos ayuda a generar el esquema y entrada de datos del servicio web AgnosticMessageCDDM
     * @return retorna "Ok" si fue exitoso, caso contrario el error que se produce, ademas de la impresión del error del codigo
     */

    @WebMethod
    public String sendMessage(@WebParam(name = "payload") MessageFacade payload){
    String successful = "Ok";
    try{
        List<TransactionScheme> messages = payload.getMessages();
        successful = this.callCompositePayload(messages);
    }catch(Exception e){
        successful = "Transaction Error ...";
        e.printStackTrace();
    }
    return successful;
    }
```

```java
    /**
     * Método callCompositePayload nos ayuda a invocar la estructura Composite Payload y Message para almacenar información para ser gestionada
     * @param messages Este parametro representa una lista del tipo TransactionScheme que representa los mensajes que son suministrados por los clientes
     * @return retorna "Ok" si fue exitoso, caso contrario el error que se produce, ademas de la impresión del error del codigo
     */

    private String callCompositePayload(List<TransactionScheme> messages){
      String successful = "Ok";
      Payload payload = new Payload(null);
      Message msg = null;
      TransactionScheme txSche = null;
      Transaction tx = null;
      try{
          Iterator iterTx = messages.iterator();
          while(iterTx.hasNext()){

              txSche = (TransactionScheme) iterTx.next();
              tx = new Transaction();
              tx.setEntityName(txSche.getEntityName());
              tx.setActionType(txSche.getActionType());
              tx.setFields(txSche.getFields());

              msg = new Message(tx);

              payload.addEnvelope(msg);
          }
              successful = payload.invokeStrategy();
        }catch(Exception e){
          successful = "Transaction Error ...";
          e.printStackTrace();
        }
      return successful;
    }

}
```

Figura 35. Mensaje *Agnostic Message* implementando *Strategy*.

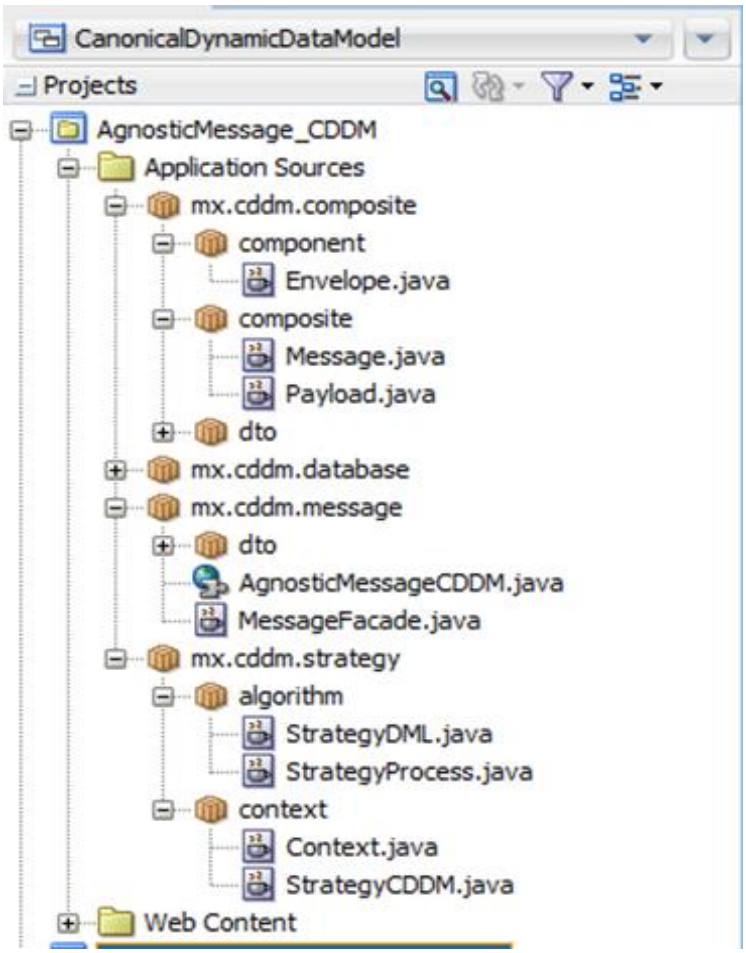


Figura 36. Proyecto CDDM *Agnostic Message*.

4.8 Validación de ejemplo de aplicación

En esta subsección se describe cómo se llevó a cabo la validación de la arquitectura MCDD en el entorno empresarial de Paquetexpress con el fin de mitigar la problemática planteada en la subsección 4.1 de este capítulo de la tesis. Para realizar la validación del ejemplo de aplicación se diseñó un plan de pruebas, debido a las prácticas de mejora de procesos que se realizan en la empresa, que consiste en tener cuatro clientes implementados bajo la herramienta gráfica S*oapUI* en su versión 5.4.0 (ver Figura 39), después, estos fueron conectados al servidor que expone la solución arquitectónica MCDD (Modelo Canónico de Datos Dinámico) para proceder a integrar información a la misma entidad (*Items*) de la base de datos de *PostgresSQL*, sin embargo, dos de ellos necesitarán enviar un dato (campos nuevos) más a esa entidad debido a la estructura de datos que opera el software que utilizan.

Para iniciar con la implementación y ejecución del caso de prueba, fueron conectados al contrato WSDL desplegado en el *Web Server* en este caso *WebLogic Server 12c* en la versión 12.2.1.3.0. Una vez hecha la conexión, se capturaron los campos y valores que viajaron dentro del mensaje universal agnóstico el cual fue gestionado por el servicio web llamado *AgnosticMessageCDDMService* que contiene la implementación de la arquitectura.

En la Tabla 8 se muestran todos los recursos tanto de *hardware* como de software utilizado para establecer el ambiente de pruebas para ejecutar el servicio, se consideran las características por grupo.

Tabla 8. Recursos de hardware y software para ambiente de caso de estudio.

Recursos de Hardware	
Recurso	**Descripción**
Equipo	HP EliteBook
Memoria	8 GB
Procesador	Core i5 1.80 GHz
Recursos de Software	
Recurso	**Descripción**
Sistema Operativo	Windows 10
Web Server	WebLogic 12c
Base de Datos	PostgresSQL 9.5
Herramienta para consumo WSDL (Cliente)	SoapUI 5.4.0
Plataforma	Java 8

A continuación, se muestra la Tabla 9 en donde se describe el objetivo del caso de prueba, sul identificador, nombre del caso, el cual es "*Desacoplamiento Entidad Items*", también las precondiciones o criterios para su funcionamiento y por último 8 pasos o secuencias de ejecución con sus resultados para comprobar el correcto funcionamiento.

Tabla 9. Caso de estudio para arquitectura MCDD.

Caso de estudio bajo acoplamiento MCDD	
Objetivo del caso de prueba	Validar el desacoplamiento a nivel de datos y externo en la integración de plataformas, usando cuatro clientes conectados al servicio de integración, dos de ellos tendrán que enviar dos datos más que se agregaron a la entidad *Items* de la base de datos de *PostgresSQL*, no tendrá que afectar este cambio a los tres Clientes conectados.
Identificador	TC_Desacople_*Item*
Nombre del caso	Desacoplamiento Entidad *Items*
Precondiciones	a). - Los cuatro Clientes estén conectados al servicio con la arquitectura propuesta.
Paso	**Resultados**
1) Conectar cada cliente al servicio.	Crear conexión con el contrato WSDL del servicio con la arquitectura propuesta.
2) Enviar datos de integración por cada uno.	Validar que la información se integre de los cuatro Clientes a la Entidad *Items*.
3) Insertar dos campos más a la entidad *Items*.	En la base de datos agregar dos campos (*sku*, *kop*) más a la entidad *Items*.
4) En uno de los clientes envía el campo y valor de los campos nuevos.	En uno de los clientes agregar (*sku*) con valor 100234.

5) En otro de los clientes enviar el segundo campo nuevo y el respectivo valor.	En el segundo cliente agregar (*kop*) con valor 24.
6) Enviar de nuevo datos de los dos clientes sin cambios.	Validar que se integre la nueva información sku 100234 a la entidad *Items*.
7) Enviar datos del tercer cliente al servicio	Validar que se integre la nueva información *kop* 24 a la entidad *Items*.
8) Enviar datos del cuarto cliente al servicio.	En los dos clientes anteriores enviar de nuevo información que siempre se ha enviado, estos clientes no tendrán que presentar ningún error.

En la Figura 37 se detalla la entidad *Items*, incluyendo da uno de sus campos (*id*, *description*, *amount* y *price*) representada de forma gráfica en la base de datos *PostgreSQL*, en donde se realizó la integración de datos.

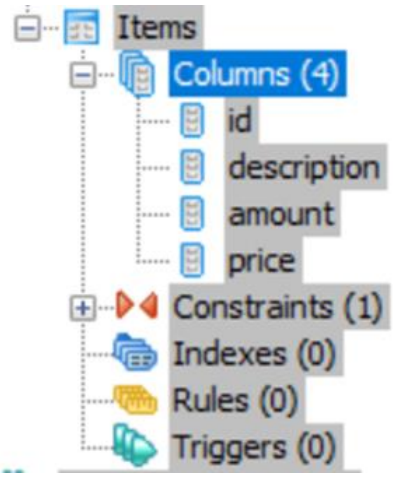


Figura 37. Entidad *Items* en *PostgreSQL*.

Posteriormente, se procedió con el despliegue del servicio (*AgnosticMessageCDDM*) en el servidor de aplicación *WebLogic Server 12c*. En la Figura 38 podemos ver de forma gráfica el despliegue y activación del servicio en una instancia de servidor *Weblogic*. En la Figura 39 se muestra la herramienta *SoapUI* que se utilzó para configurar cinco clientes, de esta forma llevar a cabo los casos de estudio de la Tabla 9 de esta sección.

Despliegues

Instalar | Actualizar | Suprimir

Nombre	Estado	Estado	Tipo
AgnosticMessageCDDM	Activo	Ok	Aplicación Web
Servicios Web			
AgnosticMessageCDDMService			Servicio Web

Figura 38. Despliegue de AgnosticMessageCDDM en WebLogic Server 12c.

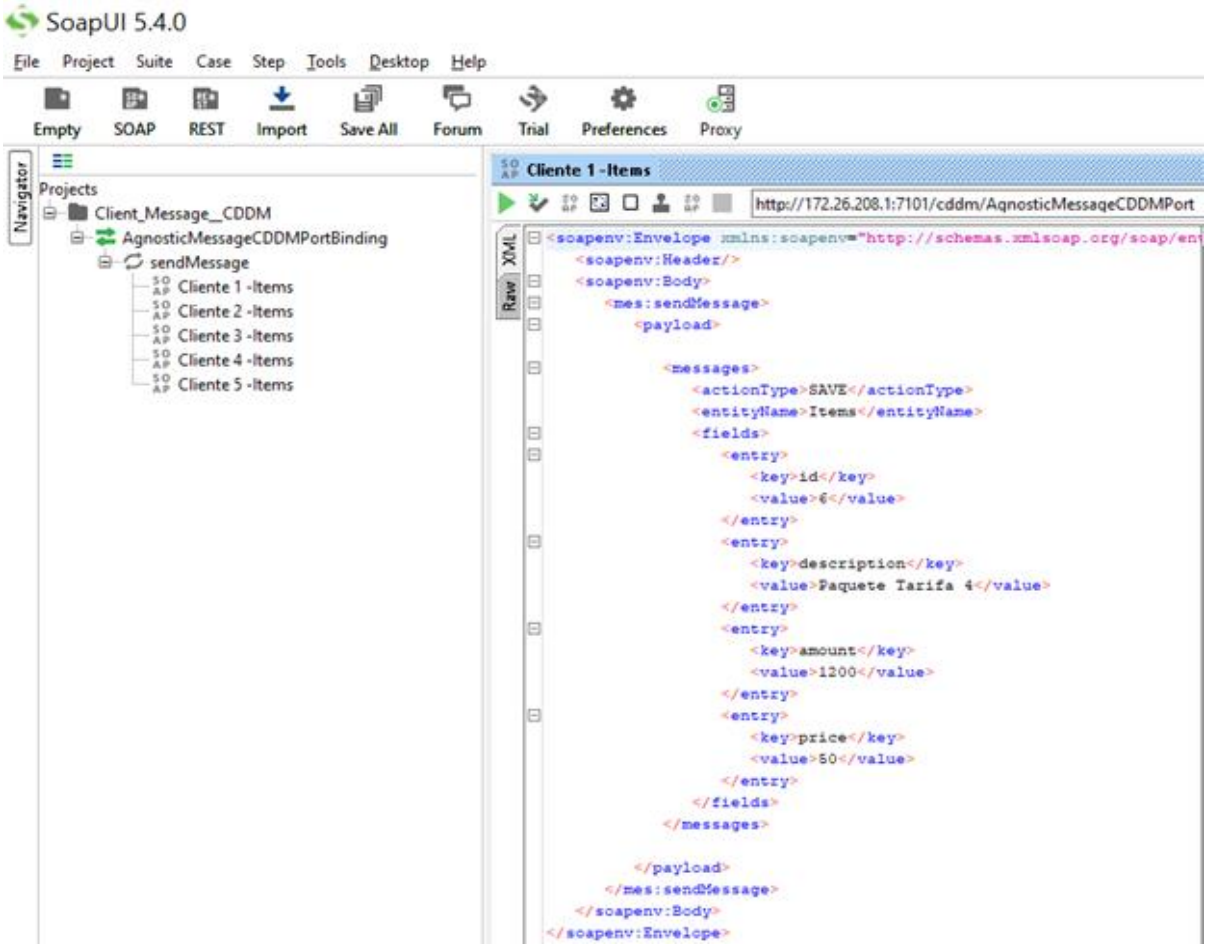
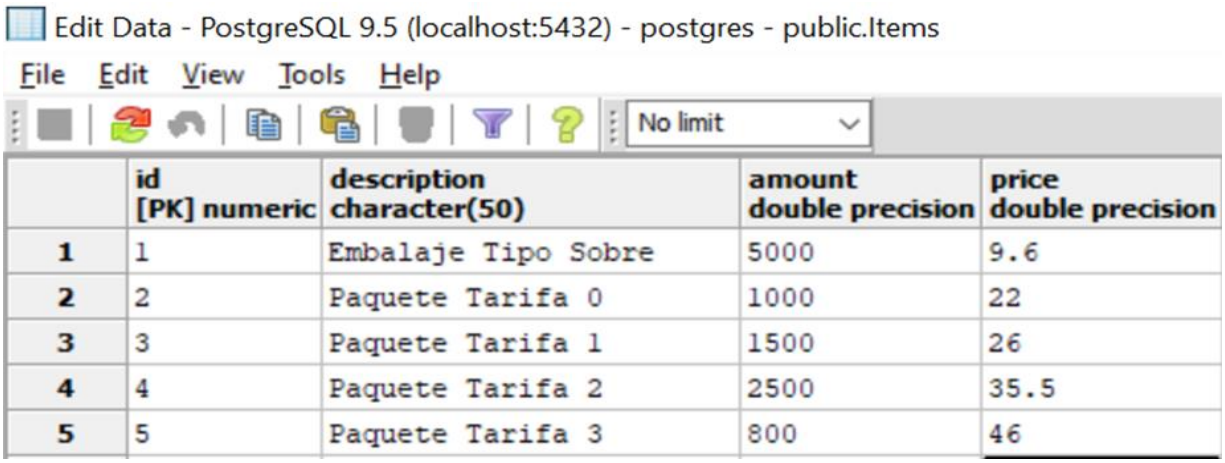


Figura 39. Clientes en *SoapUI* que enviaran datos a integrar.

Por otra parte, la Tabla 10 y Figura 40 muestran algunos registros que fueron enviados a la base de datos de PostgreSQL de acuerdo con los puntos 1 y 2 del caso de estudio de la Tabla 9 en donde se representa el mensaje enviado con la información de las columnas (*id, description, amount, price*) que fueron gestionados por los algoritmos para la integración.

Tabla 10. Integración de datos a entidad *Items* mediante Arquitectura MCDD.

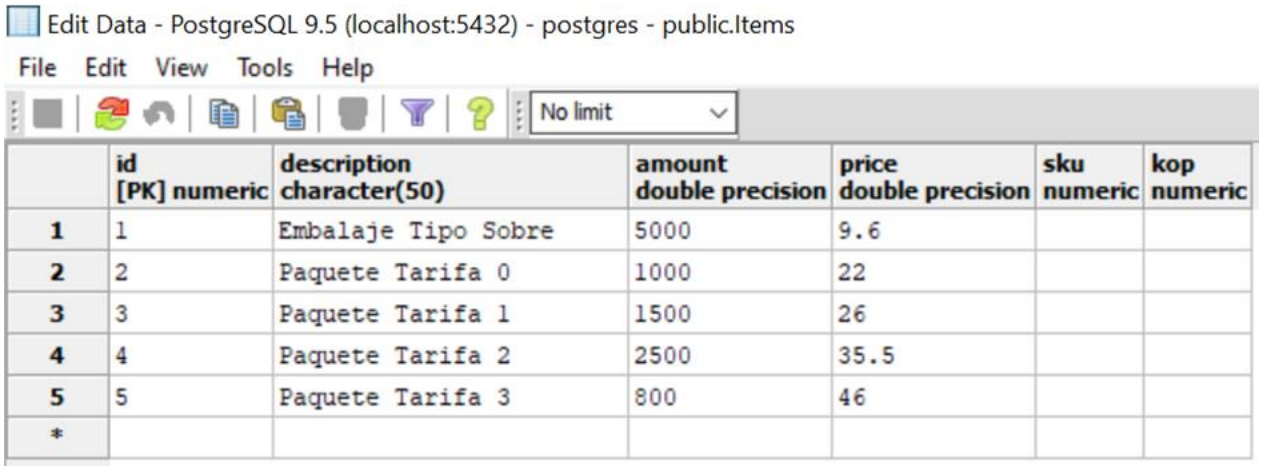
Clientes	***id***	***description***	***amount***	***price***
Cliente 1	1	Embalaje Tipo Sobre	5000	9.6
Cliente 2	2	Paquete Tarifa 0	1000	22
Cliente 3	3	Paquete Tarifa 1	1500	26
Cliente 4	4	Paquete Tarifa 2	2500	35.5
Cliente 1	5	Paquete Tarifa 3	800	46



	id [PK] numeric	description character(50)	amount double precision	price double precision
1	1	Embalaje Tipo Sobre	5000	9.6
2	2	Paquete Tarifa 0	1000	22
3	3	Paquete Tarifa 1	1500	26
4	4	Paquete Tarifa 2	2500	35.5
5	5	Paquete Tarifa 3	800	46

Figura 40. Datos integrados del cliente.

Para continuar con el caso se ejecutaron los puntos 3, 4, 5, 6, 7 y 8 de la Tabla 9, enseguida se muestran los atributos y/o campos "*sku*" y "*kop*" agregados a la entidad *Items* como parte del punto 3. La Figura 41 muestra este agregado en la base de datos de PostgreSQL.



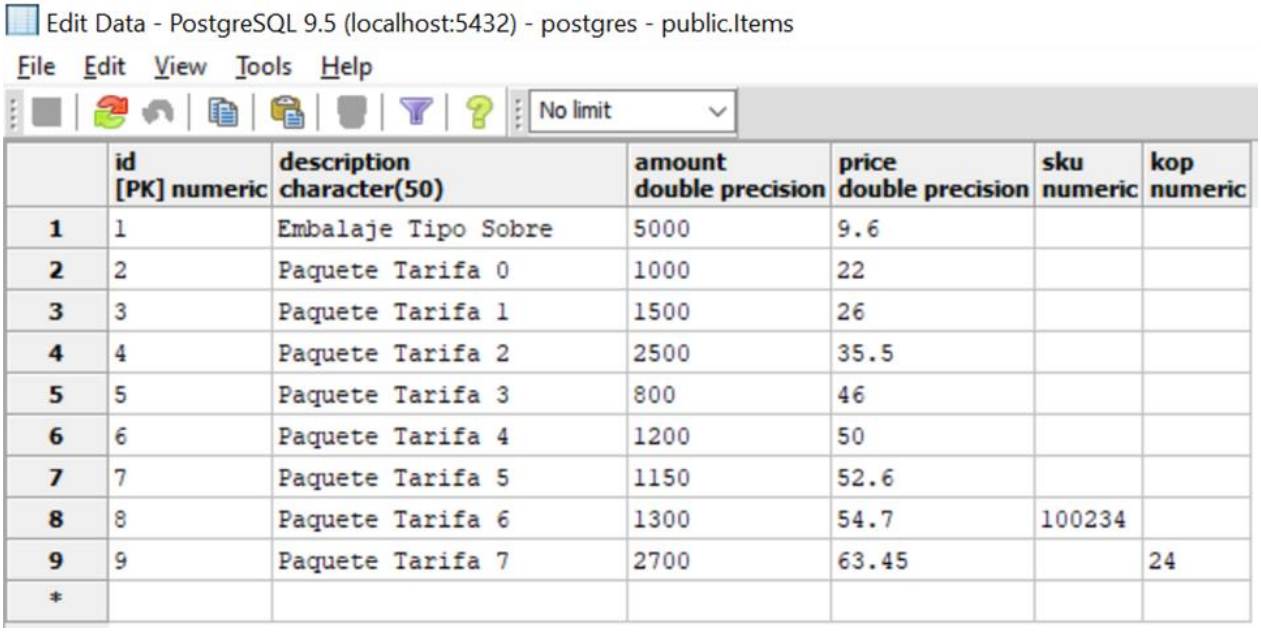
	id [PK] numeric	description character(50)	amount double precision	price double precision	sku numeric	kop numeric
1	1	Embalaje Tipo Sobre	5000	9.6		
2	2	Paquete Tarifa 0	1000	22		
3	3	Paquete Tarifa 1	1500	26		
4	4	Paquete Tarifa 2	2500	35.5		
5	5	Paquete Tarifa 3	800	46		
*						

Figura 41. Entidad *Items* con los dos nuevos campos "*sku*" y "*kop*".

En la Figura 42 y Tabla 11 se muestra la integración de los nuevos registros de acuerdo con los puntos 4 y 5 de la Tabla 9 del caso de estudio evaluado y para los campos *sku* y *kop* en la base de datos de PostgreSQL. Para lograr esto, se continuó utilizando el mismo contrato WSDL donde se encuentra la implementación de la arquitectura MCDD la cual no fue cambiada, ni fue hecho un nuevo despliegue del servicio web. Es importante mencionar que no se realizó mantenimiento alguno en ninguno de los clientes conectados, según los pasos 6, 7 y 8 como se muestra en la Tabla 9 caso de estudio.

Tabla 11. Integración en *Items* agregando dos campos, mediante Arquitectura MCDD.

Clientes	***id***	***description***	***amount***	***price***	***sku***	***kop***
Cliente 1	6	Paquete Tarifa 4	1200	50	-	-
Cliente 2	7	Paquete Tarifa 5	1150	52.6	-	-
Cliente 3	8	Paquete Tarifa 6	1300	54.7	100234	-
Cliente 4	9	Paquete Tarifa 7	2700	63.45	-	24



	id [PK] numeric	description character(50)	amount double precision	price double precision	sku numeric	kop numeric
1	1	Embalaje Tipo Sobre	5000	9.6		
2	2	Paquete Tarifa 0	1000	22		
3	3	Paquete Tarifa 1	1500	26		
4	4	Paquete Tarifa 2	2500	35.5		
5	5	Paquete Tarifa 3	800	46		
6	6	Paquete Tarifa 4	1200	50		
7	7	Paquete Tarifa 5	1150	52.6		
8	8	Paquete Tarifa 6	1300	54.7	100234	
9	9	Paquete Tarifa 7	2700	63.45		24
*						

Figura 42. Entidad *Items* con los datos integrados en nuevos campos "*sku*" y "*kop*".

Con esto se concluyó el caso de estudio de la Tabla 9, "*Caso de estudio bajo acoplamiento MCDD*", donde los resultados son satisfactorios para el objetivo del Bajo Acoplamiento a nivel de Datos y Externo. Entre los resultados obtenidos destaca el hecho de que el contrato que se expuso para consumo de los clientes no sufrió cambios, de igual forma los clientes conectados al servicio web no necesitaron recibir mantenimiento, con lo cual se ahorra tiempo y esfuerzo. Además, se demostró que la propuesta es escalable y reutilizable, cumpliendo con todos los objetivos especificados en la tesis.

Capítulo V
Conclusiones

5.1 Conclusiones

Actualmente, no solo los grandes corporativos, sino también las pequeñas y medianas empresas están adoptando EAI como una medida para integrar la información creciente de sus sistemas independientes con la finalidad de mejorar sus procesos y evitar error de captura, trabajo duplicado e inconsistencia de la información de sus clientes, ventas, recursos humanos, nóminas, etc.

En el mercado de herramientas de EAI existen opciones de suites de software listas con interfaces de integración. Sin embargo, las soluciones siempre van a depender de la arquitectura que se instaure con apoyo de los consultores y expertos.

Finalmente, el desarrollo constante de la tecnología y su aplicación en los procesos de la empresa llevará a la introducción de nuevas plataformas y, por tanto, será necesario llevar a cabo soluciones de integración de aplicaciones para garantizar el intercambio entre los nuevos y antiguos sistemas, a razón de que ninguna empresa puede soportar la reprogramación desde cero de todas las aplicaciones que utilizan en su actividad operacional.

Unos de los objetivos de este trabajo fue definir, diseñar e implementar una propuesta arquitectónica mediante una representación de modelos canónicos de datos dinámica a través mensajes agnósticos para mejorar el bajo acoplamiento dirigido a los niveles de datos y externo en la integración de unidades de software. Este objetivo fue cubierto con el diseño propuesto e implementado mediante el uso de patrones de diseño que fueron apoyo a la propuesta presentada, de esta forma se crea una capa intermedia entre las diversas plataformas existentes. El objeto de esa capa intermedia es que a través de ella viaje la información a integrar, así como un fragmento de la representación del modelo de datos (estructura), sus entidades y relaciones representadas en un mensaje universal. El cual fue gestionado e interpretado por algoritmos diseñados para tal propósito, en este caso se propusieron dos de los algoritmos, sin embargo, se pueden crear otros algoritmos de acuerdo con la necesidad de la integración. Con esto, se evitó rediseñar o reconstruir

el contrato WSDL que ya estaba implementado en las diversas plataformas integradas de tal forma que, cuando exista un cambio en alguna de las plataformas, la integración pueda ser escalable, de fácil mantenimiento y por lo tanto de bajo costo en el desarrollo y mantenimiento al reutilizar el componente.

Una de las ventajas de la propuesta presentada en esta tesis radica en los escenarios que tienen una gran cantidad de clientes conectados al contrato de servicio WSDL y por alguna necesidad alguno de ellos necesita agregar nueva información (un nuevo dato), el cambio no impactaría en el resto de los demás sistemas ya conectados debido a que solo uno de ellos requiere el nuevo dato, el cual se agregaría en la estructura del mensaje con su información. De esta forma todos seguirán trabajando normalmente sin ningún inconveniente.

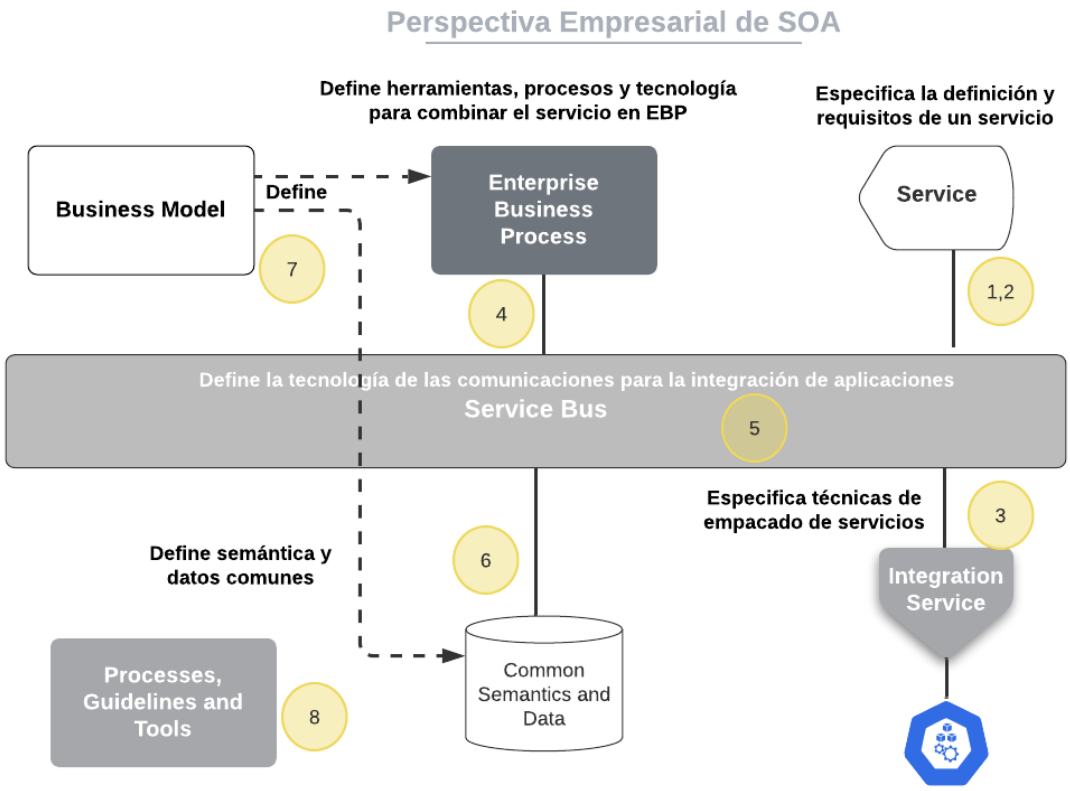
Una desventaja podría ser al momento de generar el armado del mensaje y declarar la sección de transacción, debido a que además de la información, se especifica los campos pertenecientes a las entidades, sin embargo, se podría construir como parte de un *framework* el armado de estas secciones y que pueda ser más sencillo a la hora de implementar esta parte de la propuesta.

5.2 Trabajo futuro

En la propuesta antes mencionada, no se incluyó una parte importante en la estructura de árbol sobre las relaciones que existen entre las entidades, que son parte esencial de la canonización de datos de forma dinámica, donde se representan las llaves primarias y llaves foráneas junto con sus restricciones. Esta mejora se considera como trabajo futuro para un mejor funcionamiento de la arquitectura. Además, se integrará en un *framework* con interfaz de usuario para gestionar de manera sencilla y rápida.

Finalmente, se contempla extender las opciones de representación de datos con JSON sobre arquitecturas de servicios como REST que se transfieren por el protocolo HTTP.

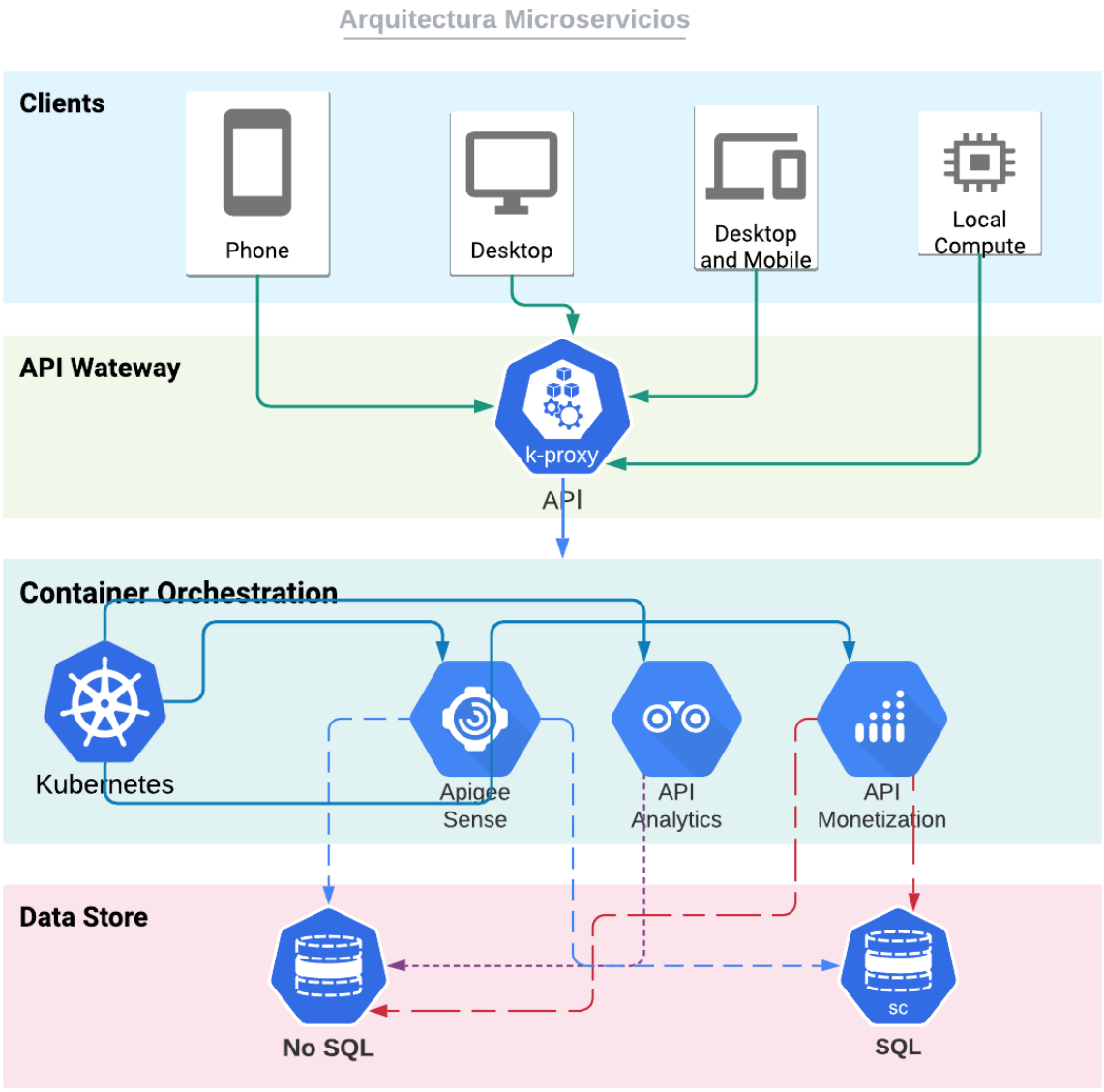
Anexo A. Perspectiva Empresarial SOA



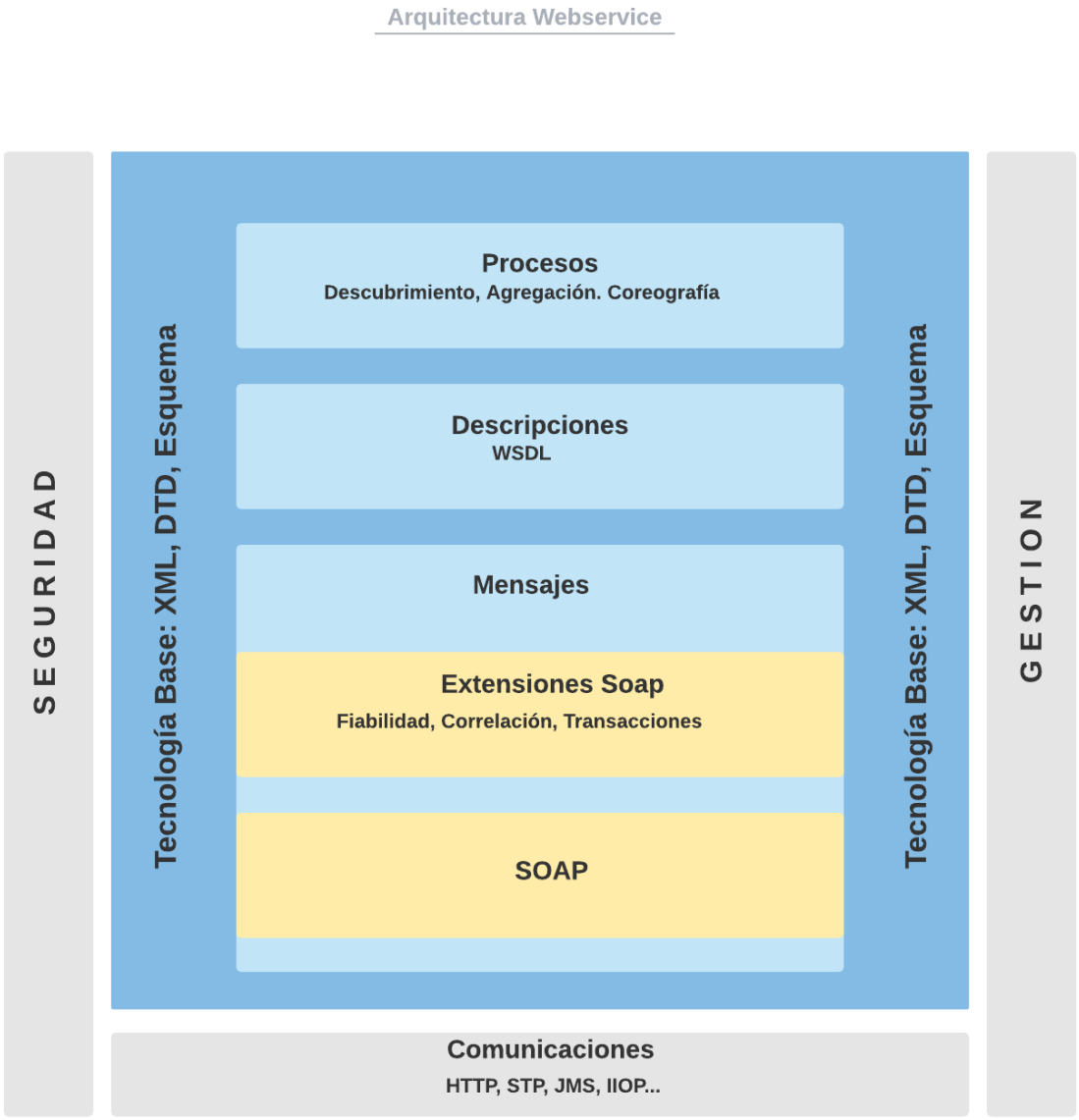
SOA necesita describir los siguientes aspectos de los servicios dentro de una empresa:

1. Una definición de servicios, la granularidad y los tipos de servicios.
2. Cómo se construyen y utilizan los servicios
3. Cómo se integran los sistemas existentes empaquetados y heredados en el entorno de servicio
4. Cómo se combinan los servicios en procesos
5. Cómo se comunican los servicios a nivel técnico (es decir, cómo se conectan unos a otros y pasar información)
6. Cómo interoperan los servicios a nivel semántico (es decir, cómo comparten significados de esa información)
7. Cómo se alinean los servicios con la estrategia y los objetivos comerciales
8. Cómo utilizar la arquitectura

Anexo B. Arquitectura *Microservice*



Anexo C. Arquitectura *Web Service*



Glosario

El presente glosario tiene como objetivo principal ayudar a la comprensión del significado de palabras utilizadas en MCDD (Modelo Canónico de Datos Dinámico), para su fácil comprensión en el área de las Ciencias Informáticas en la rama de Ingeniería de Software para la Integración de Aplicaciones Empresariales.

API	*Application Programming Interface*
B2B	*Business To Business*
B2C	*Business To Client*
BPM	*Business Process Management*
CDM	*Canonical Data Model*
CIO	*Chief Information Officer*
CORBA	*Common Object Request Broker Architecture*
CRM	*Customer Relationship Management*
DBMS	*DataBase Management System*
DML	*Data Manipulation Language*
EAI	*Enterprise Application Integration*
ENTITY	*Entidad*
E-R	*Entidad-Relación*
ERP	*Enterprise Resource Planning*
ESB	*Enterprise Service Bus*
FRAMEWORK	*Marco De Trabajo*
GC	*Grid Computing*
HL7	*Health Level Seven*

HTTP	*Hypertext Transfer Protocol*
IBM	*International Business Machines*
JDBC	*Java Data Base Connectivity*
JSON	*Java Scrip Object Notation*
KQML	*Knowledge Query and Manipulation Language*
MDB	*Message Driven Bean*
MOM	*Message Oriented Middleware*
ODBC	*Open Data Base Connectivity*
PyMES	*Pequeñas y Medianas Empresas*
QUEUES	*Patrón Arquitectónico Cola*
REST	*Representational State Transfer*
RMI	*Java Remote Method Invocation*
SAP	*System Application Products in Data Processing*
SCA	*Service Component Architecture*
SLR	*Systematic Literature Review*
SMS	*Systematic Mapping Study*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SQL	*Structure Query Language*
IT	*Information Technology*
WORKFLOW	*Flujo de Trabajo*
WSDL	*Web Services Description Language*
XML	*Extensible Markup Language*

XSD	*XML Schema Definition*
XSLT	*eXtensible Stylesheet Language for Transformation*

Referencias

[1] Zahir, I., Marinos, T., & Love, P. E.D. (2003). The impact of enterprise application integration on information system lifecycles. *Information & Management*, 41 (2), 177-187. https://doi.org/10.1016/S0378-7206(03)00046-6

[2] SAP. (s.f.). *Software Para Gestión De Pequeñas y Medianas Empresas SAP SME*. Consultado el 17 de febrero de 2019. https://www.sap.com/latinamerica/products/sme-business-software.html?url_id=ctabutton-lao-icon-sme

[3] Chander, S. (2021). *The Arrival of Java 16*: *Oracle Java Platform Group, Product Management Blog*. Consultado el 7 de mayo de 2021. https://blogs.oracle.com/java-platform-group/the-arrival-of-java-16

[4] Oracle. (s.f.). *Aplicaciones De Oracle PeopleSoft | Oracle América Latina*. Consultado el 17 de febrero de 2019. https://www.oracle.com/lad/applications/peoplesoft/

[5] Oralce. (s.f.). *JD Edwards EnterpriseOne*. Consultado el 17 de febrero de 2019. https://www.oracle.com/applications/jd-edwards-enterpriseone/

[6] Oralce. (s.f.). *Oracle Siebel CRM Applications*. Consultado el 17 de febrero de 2019. https://www.oracle.com/cx/siebel/

[7] Minoli, D. (2008). *Enterprise architecture A to Z: frameworks, business process modeling, SOA, and infrastructure technology*. Boca Raton, FL: CRC Press.

[8] Muñoz, A., Aguilar, J. & Martínez, R. (2005). Modelos inteligentes para base de datos distribuida. *Gerencia Tecnológica Informática*, 4 (10), 91-114. http://revistas.uis.edu.co/index.php/revistagti/article/view/1562

[9] Oracle. (s.f.). *The Map Interface*. Consultado el 7 de marzo de 2020. https://docs.oracle.com/javase/tutorial/collections/interfaces/map.html

[10] Hohpe, G., & Woolf, B. (2015). *Enterprise integration patterns: designing, building and deploying messaging solutions*. Boston: Addison-Wesley.

[11] Rosen, M., Lubblinsky, B., Smith, K. & Balcer, M. (2012). *Applied SOA: Service-Oriented Architecture and Design Strategies*. Indianapolis: Wiley Publishing, Inc.

[12] Celma Giménez, M., Casamayor Ródenas J.C. & Mota Herranz, L. (2003). *Bases De Datos Relacionales*. Madrid: Pearson Educación.

[13] Garcia-Molina, H., Ullman, J. D. & Widom, J. (2009). *Database systems: the complete book* (2nd Ed.). Upper Saddle River, New Jersey: Pearson Prentice Hall.

[14] Gomez Fuentes, M. C. (2013). *Notas del curso: Bases de datos*. Cuajimalpa: Universidad Autónoma Metropolitana. http://ilitia.cua.uam.mx:8080/jspui/handle/123456789/177

[15] Erl, T. (2004). *Service-Oriented Architecture: A field guide to integrating xml and web services*. Prentice Hall.

[16] González Quiroga, M. (2011). *Estudio de arquitecturas de redes orientadas a servicio*. [Tesis de Licenciatura]. Universidad Politécnica de Cataluña. https://upcommons.upc.edu/bitstream/handle/2099.1/12312/ESTUDIO_DE_ARQUITECTURAS_DE_REDES_ORIENTADAS_A_SERVICIO.pdf

[17] Krafzig, D., Banke, K., & Slama, D. (2005). *Enterprise Soa: service-oriented architecture best practices*. Prentice Hall.

[18] Channabasavaiah, K., Holley, K., Tuggle, E. M. (2004). On demand operating environment solutions: Migrating to a service-oriented architecture. white paper. ftp://ftp.boulder.ibm.com/s390/audio/pdfs/G224-7298-00_FinalMigratetoSOA.pdf

[19] Sharma, R., Stearns, B., & Ng, T. (2003). *J2EE connector architecture and enterprise application integration*. Addison-Wesley.

[20] Enterprise Application Integration (2012). *Mucho más que productos de moda*. Consultado el 18 de noviembre de 2019. https://sg.com.mx/content/view/340

[21] Pressman, R., (2010). *Ingeniería del software: Un enfoque practico*. México: McGraw-Hill.

[22] García Alonso, J. (2014). *Arquitectura multicapa: acercando el diseño a la implementación.* [Tesis Doctoral] Universidad de Extremadura España. http://hdl.handle.net/10662/2221

[23] Martín-González, E., Moya-Sáez, E., Menchón-Lara, R.-M., Royuela-del-Val, J., Palencia-de-Lara, C., Rodríguez-Cayetano, M., Simmross-Wattenberg, F. & Alberola-López, C. (2021). A clinically viable vendor-independent and device-agnostic solution

for accelerated cardiac MRI reconstruction. *Computer Methods and Programs in Biomedicine*, 207, 1-13. https://doi.org/10.1016/j.cmpb.2021.106143

[24] Trad, A. & Kalpić, D. (2017*). A neural networks portable and Agnostic Implementation Environment for Business Transformation Projects the framework.* IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), Annecy, France. https://doi.org/10.1109/CIVEMSA.2017.7995319

[25] Garland, J., & Anthony, R. (2003). *Large-scale software architecture: a practical guide using UML*. England: Wiley.

[26] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. (1996). *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley.

[27] Fowler, M., & Scott, K. (1999). *UML gota a gota*. Pearson Educación. http://www.face.ubiobio.cl/~cvidal/modelamiento/libros/UMLgotaagota.pdf

[28] Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design Pattern: Elements of reusable object-oriented software*. Addison-Wesley.

[29] Sharan, K. (2017). *Java 9 Revealed: For early adoption and migration*. Apress.

[30] Petersen, K., Feldt, R., Mujtaba, S. & Mattsson, M. (2008). Systematic mapping studies in software engineering. En G. Visaggio (Ed.), *12th international conference on evaluation and assessment in software engineering* (pp. 68-77). BCS Learning & Development Ltd. https://dl.acm.org/doi/10.5555/2227115.2227123

[31] Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O. P., Turner, M., Niazi, M. & Linkman, S. (2010). Systematic literature reviews in software engineering--a tertiary study. Information and Software Technology, 52 (8) 792-805. https://doi.org/10.1016/j.infsof.2010.03.006

[32] Kitchenham, B. & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering: Technical Report EBSE-2007-01*. UK: School of Computer Science and Mathematics, Keele University. https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf

[33] Green, S. J. (2013). An evaluation of four patterns of interaction for integrating disparate ESBs effectively and easily. *Journal of Systems Integration*, 4 (3), 3-19.

[34] Voican, C. (2012). *Service orientation in distributed automation and control service*. 4th International Conference Advanced Composite Materials Engineering, Brasov, Romania. http://aspeckt.unitbv.ro/jspui/bitstream/123456789/1535/1/Paper_23_2012_vol_2.pdf

[35] Cuadrado Latasa, F., García Gutiérrez, B., Dueñas López, J. C. & Parada Gélvez, H. A. (2008). *A Case Study on Software Evolution towards Service-Oriented Architecture*. 22nd International Conference on Advanced Information Networking and Applications, GinoWan, Japón. https://doi.org/10.1109/WAINA.2008.296

[36] Herrera-Quintero, L. F., Maciá-Pérez, F., Marcos-Jorquera, D. & Gilart-Iglesias, V. (2010). *SOA-based Model for the IT Integration into the Intelligent Transportation Systems*. Workshop on Emergent Cooperative Technologies in Intelligent Transportation Systems. Madeira Island.

[37] Punitha Devi, C., Prasanna Venkatesan, V., Diwahar, S. & Shanmugasundaram, G. (2014). A Model for Information Integration Using Service Oriented Architecture. *International Journal of Information Engineering and Electronic Business*, 6 (3), 34-42. https://doi.org/10.5815/ijieeb.2014.03.06

[38] Kim, J. (2009). *Mini-SOA/ESB design guidelines and simulation for wireless sensor networks: ProQuest Dissertations Publishing*. [Master Thesis]. Korea National Open University. https://hdl.handle.net/11244/8182

[39] Hong, C. & Wen-yue, G. (2010). *Study on enterprise Order Processing System based on SOA*. IEEE International Conference On Computer Design and Applications, Qinhuangdao, China. https://doi.org/10.1109/ICCDA.2010.5541067

[40] Chen, M. (2009). *Research and Implementation on Enterprise Application Integration Platform*. IEEE International Forum on Information Technology and Applications, Chengdu, China. https://doi.org/10.1109/IFITA.2009.332

[41] Coronado-García, L. C., González-Fuentes, J. A., Hernández-Torres, P. J. & Pérez Leguízamo, C. (2011). *An Autonomous Decentralized Service Oriented Architecture for High Reliable Service Provision*. IEEE Tenth International Symposium on Autonomous Decentralized Systems, Tokyo, Japan. https://doi.org/10.1109/ISADS.2011.49

[42] Deng, W., Yang., X., Zhao, H., Lei, D. & Li, H. (2008). *Study on EAI Based on Web Services and SOA*. IEEE International Symposium on Electronic Commerce and Security, Guangzhou City, China. https://doi.org/10.1109/ISECS.2008.121

[43] Dejan, R. (2007). An integration strategy for large enterprises. *Yugoslav Journal of Operations Research*, 17 (2), 209-222. https://doi.org/10.2298/YJOR0702209R

[44] Beer, M. I. & Hassan, M. F. (2018). Adaptive security architecture for protecting RESTful web services in enterprise computing environment. *Service Oriented Computing and Applications*, 12, 111-121. https://doi.org/10.1007/s11761-017-0221-1

[45] González, L. O. & Guadalupe (2013). An ESB-Based Infrastructure for Event-Driven Context-Aware Web Services. En C. Canal, M. Villari (Eds.) *Advances in Service-Oriented and Cloud Computing* (pp. 360-369). Springer. https://doi.org/10.1007/978-3-642-45364-9_29

[46] Monfort, V., & Slimane, H. (2009). Towards Adaptable SOA: Model Driven Development, Context and Aspect. En L. Baresi, C. H. Chi & J. Suzuki (Eds.), *Service-Oriented Computing. 7th International Joint Conference, ICSOC-ServiceWave* (pp. 175-189). Springer. http://dx.doi.org/10.1007/978-3-642-10383-4_12

[47] Huang, D. & Zhang, W. (2010). *Study on Enterprise Informationization Models*. IEEE International Conference on E-Business and E-Government, Guangzhou, China. https://doi.org/10.1109/ICEE.2010.645

[48] Ji, X. (2009). *A Web-based Enterprise Application Integration solution*. 2nd IEEE International Conference on Computer Science and Information Technology, Beijing, China. https://doi.org/10.1109/ICCSIT.2009.5234730

[49] Mazzara, M., Dragoni, N., Bucchiarone, A., Giaretta, A. Larsen, S. T. & Dustdar, S. (2017). Microservices: Migration of a Mission Critical System. IEEE Transactions on Services Computing, 14 (5), 1464-1477. https://doi.org/10.1109/TSC.2018.2889087

[50] Dragoni, N., Giallorenzo, S., Lluch Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R. & Safina, L. (2016). Microservices: yesterday, today, and tomorrow. En M. Mazzara & M. Meyer (Eds.) *Present and Ulterior Software Engineering* (pp. 195-216). Springer. https://doi.org/10.1007/978-3-319-67425-4_12

[51] Parizi, R. M. (2018). *Microservices as an Evolutionary Architecture of Component-Based Development: A Think-aloud Study*. https://doi.org/10.48550/arXiv.1805.11757

[52] Shadija, D., Mo., R. & Hill, R. (2017). *Towards an Understanding of Microservices*. IEEE 23 rd International Conference on Automation and Computing, Huddersfield, UK. https://doi.org/10.23919/IConAC.2017.8082018

[53] Albertos Gómez, E. (2018). *Arquitecturas software para microservicios: una revisión sistemática de la literatura.* [Trabajo de Fin de Máster]. Universidad Politécnica de Madrid. https://oa.upm.es/51460/

[54] Antipov, V., Antipov, O. & Pylkin A. (2016). *Mobility support in publish/subscribe systems*. *ITM Web of Conferences* (6), 1-4. https://doi.org/10.1051/itmconf/20160603001

[55] Cranefield, S. & Surangika, R. (2013). *Embedding agents in business applications using enterprise integration patterns*. https://doi.org/10.48550/arXiv.1302.1937

[56] Acosta Cano de los Ríos, J. E. (2015). *Esquema de Referencia para Acoplamiento Débil entre Sistema Informático y Equipo de Producción, Industriales* [Tesis Doctoral]. Universidad Politécnica de Madrid. https://oa.upm.es/39360/

[57] Montoya Múnera, E. & García Loaiza, B. A. (2011). Integración de Repositorios Digitales en salud. *Renata*, 1 (2), 118-145.

[58] Muñoz, A. & Aguilar, J. (2009). Modelo ontológico para bases de datos multimedia. *Journal of Ciencia e Ingeniería*, 30 (2), 149–159.

[59] Weyns, D. & Georgeff, M. (2010). Self-Adaptation Using Multiagent Systems. *IEEE Software*, 27 (1), 86-91. https://doi.org/10.1109/MS.2010.18

[60] Ma, S., Tang, J. & Wang, D. (2009). *Process Based Application Level Architecture for RFID System*. IEEE 5th International Conference on Wireless Communications, Networking and Mobile Computing, Beijing, China. https://doi.org/10.1109/WICOM.2009.5301380

[61] Martins, A., Carrilho, P., Mira da Silva, M. & Alves, C. (2008). Using a SOA Paradigm to Integrate with ERP Systems. En G. Magyar, G. Knapp, Wojtkowski, W., Wojtkowski, W. G. & Zupančič, J. (Eds.) *Advances in Information Systems Development* (pp. 179-190). Boston, MA: Springer. https://doi.org/10.1007/978-0-387-70761-7_15

[62] Berná Martínez, J. V. & Maciá Pérez. F. (2010). *Model of integration and management for robotic functional components inspired by the human neuroregulatory system*. IEEE Conference on Emerging Technologies and Factory Automation (ETFA), Bilbao, Spain. http://dx.doi.org/10.1109/ETFA.2010.5641077

[63] Qu, L., Chen, Y. & Yang, M. (2009). *The Coordination and Integration of Agile Supply Chain Based on Service-oriented Technology*. IEEE Third International Symposium on

Intelligent Information Technology Application, Shanghai, China. https://doi.org/10.1109/IITA.2009.94

[64] Krishna Mohan, K., Verma, A. K., Srividya, A. S. & Ravi Kumar, G. (2010). *A* Practical Perspective on the Design and Implementation of Enterprise Integration Solution to improve QoS using SAP NetWeaver Platform. *Journal of Systemics, Cybernetics and Informatics*, 8 (3), 29-35. http://www.iiisci.org/journal/sci/FullText.asp?var=&id=GM042MC

[65] Esteban-Gutiérrez, M., García-Castro, R. & Mihindukulasooriya, N. (2013). A Coreference Service for Enterprise Application Integration using Linked Data. En M. Horbach (Ed.) *Informatik 2013 - Informatik angepasst an Mensch, Organisation und Umwelt* (pp. 1910-1924). Bonn: Gesellschaft für Informatik e.V. https://dl.gi.de/bitstream/handle/20.500.12116/20623/1910.pdf?sequence=1&isAllowed=y

[66] Lehsten, P. Gladisch, A. & Tavangarian, D. (2011). *Context-Aware integration of smart environments in legacy applications*. In Proceedings of the Second International Conference on Ambient Intelligence (AmI'11), Amsterdam, The Netherlands. https://doi.org/10.1007/978-3-642-25167-2_14

[67] Sánchez, M., Aguilar, J. & Exposito, E. (2018). Integración SOA-MAS en Ambientes Inteligentes. DYNA, 85 (206), 268-282. https://doi.org/10.15446/dyna.v85n206.68671

[68] Nazih, M. & Alaa, H. (2011). *Generic service patterns for web enabled public healthcare systems*. IEEE 7th International Conference on Next Generation Web Services Practices, Salamanca, Spain. https://doi.org/10.1109/NWeSP.2011.6088190

[69] Ruiz, J. L., Dueñas, J. C. & Cuadrado, F. (2008). *A Service Component Deployment Architecture for e-Banking*. 22nd International Conference on Advanced Information Networking and Applications - Workshops, Okinawa, Japan. https://oa.upm.es/3118/1/INVE_MEM_2008_53265.pdf

[70] Lendak, I., Varga, E., Erdeljan, A. & Gavrić, M. (2010). *RESTful web services and the Common Information Model (CIM)*. IEEE International Energy Conference, Manama, Bahrain. https://doi.org/10.1109/ENERGYCON.2010.5771774

[71] Patri, O. P., Panangada, A. V., Sorothia, V. S. & Prasanna, V. K. (2014). *Semantic management of Enterprise Integration Patterns: A use case in Smart Grids*. IEEE 30th

International Conference on Data Engineering Workshops, Chicago, IL, USA. https://doi.org/10.1109/ICDEW.2014.6818302

[72] Liu, Y. & Wang Y. (2011). *A study of metamodeling based on MDA. IEEE* 3rd International Conference on Computer Research and Development, Shanghai. https://doi.org/10.1109/ICCRD.2011.5764107