SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Tecnológico Nacional de México

Centro Nacional de Investigación y Desarrollo Tecnológico

Tesis de Maestría

Método para migrar servicios web bajo el protocolo SOAP hacia microservicios basados en REST

presentada por

L.I. Omar Hernández López

como requisito para la obtención del grado de

Maestro en Ciencias de la Computación

Director de tesis

Dr. René Santaolaya Salgado

Cuernavaca, Morelos, México. Septiembre de 2018.

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Centro Nacional de Investigación y Desarrollo Tecnológico

Cuernavaca, Morelos a 01 de agosto del 2018
OFICIO No. DCC/217/2018

Asunto: Aceptación de documento de tesis

DR. GERARDO V. GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO
PRESENTE

Por este conducto, los integrantes de Comité Tutorial del **Lic. Omar Hernández López**, con número de control M16CE079, de la Maestría en Ciencias de la Computación, le informamos que hemos revisado el trabajo de tesis profesional titulado "**Método para migrar servicios web bajo protocolo SOAP hacia microservicios basados en REST**" y hemos encontrado que se han realizado todas las correcciones y observaciones que se le indicaron, por lo que hemos acordado aceptar el documento de tesis y le solicitamos la autorización de impresión definitiva.

DIRECTOR DE TESIS

Dr. René Santaolaya Salgado
Doctor en Ciencias de la Computación
4454821

REVISOR 1

Dra. Olivia Graciela Fragoso Díaz
Doctora en Ciencias en Ciencias de la Computación
7420199

REVISOR 2

M.C. Humberto Hernández García
Maestro en Ciencias con Especialidad
en Sistemas Computacionales
7573641

C.p. M.T.I. María Elena Gómez Torres - Jefa del Departamento de Servicios Escolares.
Estudiante
Expediente

NACS/lmz

Cuernavaca, Mor., 24 de agosto de 2018
OFICIO No. SAC/356/2018

Asunto: Autorización de impresión de tesis

LIC. OMAR HERNÁNDEZ LÓPEZ
CANDIDATO AL GRADO DE MAESTRO EN CIENCIAS
DE LA COMPUTACIÓN
P R E S E N T E

Por este conducto, tengo el agrado de comunicarle que el Comité Tutorial asignado a su trabajo de tesis titulado **"Método para migrar servicios web bajo protocolo SOAP hacia microservicios basados en REST"**, ha informado a esta Subdirección Académica, que están de acuerdo con el trabajo presentado. Por lo anterior, se le autoriza a que proceda con la impresión definitiva de su trabajo de tesis.

Esperando que el logro del mismo sea acorde con sus aspiraciones profesionales, reciba un cordial saludo.

A T E N T A M E N T E
EXCELENCIA EN EDUCACIÓN TECNOLÓGICA®
"CONOCIMIENTO Y TECNOLOGÍA AL SERVICIO DE MÉXICO"

SEP TecNM
CENTRO NACIONAL
DE INVESTIGACIÓN
Y DESARROLLO
TECNOLÓGICO
SUBDIRECCIÓN
ACADÉMICA

DR. GERARDO VICENTE GUERRERO RAMÍREZ
SUBDIRECTOR ACADÉMICO

C.p. M.T.I. María Elena Gómez Torres .- Jefa del Departamento de Servicios Escolares.
Expediente

GVGR/mcr

Dedicatorias

Con todo mi amor y cariño a mi esposa Diana e hijo Mateo, quienes han sido mi inspiración, motivación y apoyo para lograr cada meta que me he propuesto. Gracias por su paciencia, amor y comprensión en estos dos años.

Con mucho cariño a mis padres Francisco y Oralia quienes han sido un pilar en mi vida y un ejemplo a seguir. Gracias por su apoyo incondicional.

Agradecimientos

A Dios por permitirme llegar a este momento tan importante de mi formación profesional y por darme la fortaleza y salud para seguir adelante.

Al Tecnológico Nacional de México/Centro Nacional de Investigación y Desarrollo Tecnológico por darme el tiempo y espacio para realizar la Maestría en Ciencias de la Computación.

Al Instituto Nacional de Electricidad y Energías Limpias, por todo el apoyo para realizar este posgrado. Al director de la División de Tecnologías Habilitadoras, Dr. Salvador González Castro y al Gerente de Tecnologías de la Información, Dr. Gustavo Arroyo Figueroa por la confianza puesta en mí y por la oportunidad otorgada para seguir creciendo en mi formación profesional y laboral.

A mi director de tesis, Dr. René Santaolaya Salgado por todo su apoyo, motivación y seguimiento a este trabajo de investigación. Agradezco también cada una de sus charlas y enseñanzas que me permitieron ser una mejor persona tanto en lo profesional como en lo personal.

A la Dra. Olivia Graciela Fragoso Díaz, y al Mtro. Humberto Hernández García por su apoyo en la revisión, seguimiento de este trabajo y por cada comentario realizado con el objetivo de mejorar esta investigación. A mis profesores de maestría por todo su conocimiento y experiencias trasmitidas, que fueron base para el desarrollo de esta tesis.

A mis compañeros y amigos de la maestría por todos los momentos compartidos y por todo su apoyo.

Por último, pero no menos importante, agradezco a mis hermanas Adriana y Mónica, por estar siempre al pendiente de mi formación profesional y por todos sus ánimos.

Resumen

Uno de los desafíos en el área de la informática es el desarrollo de software de calidad, escalable y que favorezca el reuso. Los sistemas de software se hacen más grandes a lo largo del tiempo, se desvían de su arquitectura original así como de sus patrones con los cuales fueron implementados y se vuelven difíciles, arriesgados y costosos.

Los microservicios son una nueva tendencia de la ingeniería de software, que enfatiza el diseño y desarrollo de software escalable y altamente mantenible. Los microservicios administran la creciente complejidad al descomponer funcionalmente los grandes sistemas monolíticos en un conjunto de servicios independientes que atienden una única responsabilidad o meta de valor y se comunican por medio de protocolos ligeros por ejemplo HTTP bajo el estilo REST. Al hacer que los servicios sean completamente independientes en el desarrollo y la implementación, se favorece la reusabilidad.

En esta investigación se propone un método para migrar Servicios Web bajo el protocolo SOAP hacia Microservicios basados en REST, mediante la generación de servicios envueltos en la capa REST que atienden una única responsabilidad y que cumplan con características balanceadas de coherencia, cohesión, factor de acoplamiento y tiempo de respuesta para lograr un equilibrio en estos atributos de calidad. El propósito que se persigue es desplegar los microservicios mediante el estilo arquitectónico REST, así como medir el rendimiento de los Servicios Web y Microservicios enfocándose en el tiempo de respuesta con base a los protocolos de comunicación de cada uno para poder determinar cuál tiene un mejor rendimiento.

El método desarrollado en esta tesis se implementó en una extensión a la herramienta SR2-Transforming (Sistema de Reingeniería para Transformación) para dar soporte a la Reingeniería de Marcos de Aplicaciones Orientados a Objetos hacia Marcos de Microservicios equivalentes.

Por último, se establecieron pruebas estadísticas para probar que se cumplen los objetivos planteados en esta tesis.

Palabras clave: microservicios, rest, servicios web, soap, reingeniería de software, métricas, calidad.

Abstract

One of the challenges in the information technology area is the development of quality, scalable software that encourages reuse. Software systems become larger over time, deviating from their original architecture as well as their patterns with which they were implemented and become difficult, risky and expensive.

Microservices are a new trend in software engineering, which emphasizes the design and development of scalable and highly maintainable software. Microservices manage the increasing complexity by functionally decomposing large monolithic systems into a set of independent services that serve a single responsibility or value goal and communicate through light protocols such as HTTP under REST style. Reusability is favored, by making services completely independent in development and implementation.

In this research we propose a method to migrate Web Services under the SOAP protocol towards REST-based Microservices, by generating services wrapped in the REST layer that serve a single responsibility and that comply with balanced characteristics of coherence, cohesion, coupling factor and response time to achieve a balance in these quality attributes. The purpose is to deploy the microservices through the REST architectural style, as well as to measure the performance of Web Services and Microservices focusing on the response time based on the communication protocols of each one in order to determine which has a better performance.

The method developed in this thesis was implemented in an extension to the tool SR2-Transforming (Reengineering System for Transformation) to support the Reengineering of Object-Oriented Applications Frameworks towards equivalent Microservices Frameworks.

Finally, statistical tests were applied to prove that the objectives set this thesis were met.

Keywords: microservices, rest, web services, soap, software reengineering, metrics, quality

Índice

Índice ..... **i**

Lista de figuras ..... **iv**

Lista de tablas ..... **v**

Tabla de abreviaturas ..... **vi**

Glosario de términos ..... **vii**

Capítulo 1: Introducción **10**

1.1 Presentación 10

1.2 Organización del documento de tesis 12

Capítulo 2: Marco de Referencia **13**

2.1 Planteamiento del problema 13

2.2 Objetivos de la tesis 13

2.3 Justificación 14

2.4 Alcances y limitaciones de la investigación 15

2.5 Situación actual de los microservicios 16

2.6 Trabajos relacionados 18

2.7 Trabajos antecedentes realizados en el CENIDET 21

2.8 Comparativa de los trabajos relacionados 23

Capítulo 3: Marco Conceptual **27**

3.1 Aplicación Monolítica 27

3.2 Modularidad 27

3.3 Servicio 28

3.4 Marco de Servicios Web 28

3.5 Servicios Web 28

3.6 SOA (Arquitectura Orientada a Servicios) 29

3.7 Microservicios 30

3.7.1 Características de los microservicios 31

3.7.2 Contextos delimitados 32

3.8 REST 32

3.8.1 Principios de REST 33

3.9 SOAP 34

3.10 Ventajas y desventajas de REST vs SOAP 34

3.11 Orquestación y Coreografía 35

3.11.1 Orquestación 35
3.11.2 Coreografía 36
3.12 Throughput 36

Capítulo 4: Descripción y Análisis del Método MSWeb2MMServ 37
4.1 Descripción del Método MSWeb2MMServ 37
4.1.2 Definir los límites del servicio 38
4.1.3 Re-factorizar el código de la aplicación 41
4.1.4 Re-factorizar los datos 42
4.2 Análisis de casos de uso del método MSWeb2MMServ 44
4.2.1 Diagrama principal de casos de uso del Método MSWEB2MMServ. 45
4.2.2 CU_01 Analizar el marco orientado a objetos 45
4.2.3 CU_02 Identificar el código para cada caso de uso 46
4.2.4 CU_03 Generar Microservicios 47

Capítulo 5: Diseño e Implementación del Método en la Herramienta SR2-Transforming 49
5.1 Diagrama de secuencias 49
5.1.1 Diagrama de secuencias del proceso principal 50
5.1.2 Diagrama de secuencias para analizar el MOO 51
5.1.3 Diagrama de secuencia para definir el código para cada caso de uso 52
5.1.4 Diagrama de secuencias para generar los microservicios 53

Capítulo 6: Diseño Experimental y Plan de Pruebas 57
6.1 Diseño experimental 57
6.1.1 Hipótesis general 57
6.1.2 Hipótesis específica 57
6.1.3 Hipótesis estadísticas 57
6.1.4 Formulación del modelo experimental 57
6.1.5 Variables dependientes 58
6.1.6 Variables independientes 58
6.1.7 Factores de ruido 58

Capítulo 7: Ejecución de las Pruebas y del Experimento 63
7.1 Especificación de entrada 63
7.2 Especificación de los casos de prueba 63
7.2.1 CP- MSWeb2MMServ-01 : generación correcta de código de los microservicios, así como los archivos correspondientes para su despliegue. 63
7.2.2 CP- MSWeb2MMServ-02: funcionalidad de los microservicios a través del método MOOInMS “statistic” contra el MOO estadístico original. 64

7.2.3 CP- MSWeb2MMServ-03: funcionalidad de los Microservicios generados con la herramienta SR2-Transforming contra la funcionalidad del MOO de entrada "*PSPCenidet"* 65
7.2.4 CP- MSWeb2MMServ-04: Comparativa entre el rendimiento de los Microservicios obtenidos en esta investigación contra el rendimiento obtenido con los servicios web obtenidos en las pruebas de la tesis de (Gallardo, 2018) sobre el MOO statistic. 65
7.3 Instrumentos de medición aplicados 66
7.3.1 Instrumento de medición para el rendimiento con base en el tiempo de respuesta 66
7.3.2 Instrumento para el análisis estadístico. 66
7.3.3 Eliminación del ruido experimental. 66
Capítulo 8: Análisis Estadístico e Interpretación de Resultados 67
8.1 Pruebas de normalidad 67
8.1.1 Análisis estadístico 67
8.2 Comparación de tiempos de respuesta de los casos de prueba 69
Capítulo 9: Conclusiones 70
9.1 Conclusiones generales 70
9.2 Problemas presentados 71
9.3 Trabajos futuros 72
Referencias Bibliográficas 73

Lista de figuras

Figura 1 Ejemplo de la arquitectura monolítica 27
Figura 2 Ejemplo de la arquitectura monolítica vs microservicios 32
Figura 3 Diagrama de procesos del Método MSWeb2MMServ para refactorizar a microservicios. 38
Figura 4 Diagrama de procesos para definir los límites del servicio 39
Figura 5 Ejemplo de una tabla compartida por dos servicios diferentes 43
Figura 6 Ejemplo de copias de tablas replicadas en cada servicio 44
Figura 7 Diagrama principal de casos de uso del método MSWEB2MMServ 45
Figura 8 Diagrama de secuencias del flujo principal del método MSWeb2MMServ 49
Figura 9 Diagrama de secuencias del flujo EjecutaProceso () para obtener microservicios 50
Figura 10 Diagrama de secuencias del flujo para leer el marco orientado a objetos 51
Figura 11 Diagrama de secuencias para definir el código para cada caso de uso 52
Figura 12 Diagrama de secuencias para generar microservicios 53
Figura 13 Código de la clase controlador para el llamado al microservicio. 55
Figura 14 Código de la clase que hace al microservicio ejecutable. 56
Figura 15 Código de ejecución de tareas del microservicio 56

Lista de tablas

Tabla 1 Tabla comparativa de trabajos relacionados y trabajos de antecedente vs esta investigación .. 25
Tabla 2 Ventajas y desventajas de REST vs SOAP 34
Tabla 3 Descripción del caso de uso Analizar el marco orientado a objetos..... 46
Tabla 4 Caso de uso Identificar el código para cada caso de uso 47
Tabla 5 Caso de uso Generar Microservicios 48
Tabla 6 Resultados de funcionalidad de los microservicios 64
Tabla 7 Funcionalidad de los Microservicios contra el Marco de entrada original *PSPCenidet*..... 65
Tabla 8 Elementos necesarios para poder realizar las pruebas con la herramienta JMeter 66
Tabla 9 Especificaciones técnicas del equipo donde está instalado el cliente y servidor..... 66
Tabla 10 Resultados de la prueba paramétrica U de Mann Whitney 68
Tabla 11 Comparativa entre los tiempos de respuesta 69

Tabla de abreviaturas

API	Interfaz de Programación de Aplicaciones.
DDD	Diseño Dirigido por Dominios.
HTTP	Protocolo de Transferencia de Hipertexto.
MOO	Marco Orientado a Objetos.
MOOinMS	Acrónimo del método de transformación de Marcos Orientados a Objetos con arquitectura de clases internas a Marcos de Servicios Web.
MSWeb2MMServ	Acrónimo del "Método para migrar Servicios Web bajo el protocolo SOAP hacia Microservicios basados en REST" .
REST	Representational State Transfer (Transferencia de estado representacional
SOAP	SOAP (Service Oriented Access Protocol).
URI	Identificador de recursos uniforme (Uniform Resource Identifier) .
WSDL	Web Services Description Language (Lenguaje de descripción de Servicios Web).
XML	eXtensible Markup Language (Lenguaje de marcado extensible) .

Glosario de términos

ACOPLAMIENTO	El acoplamiento es una medida del grado de relación de un módulo contra los demás. Se refiere al número de variables que intervienen en la comunicación entre módulos, más que al número de interfaces de comunicación entre módulos. Si dos módulos se comunican, deben de intercambiar tan poca información como sea posible (Brito, Iseg, Goulão, Esteves, & Ist, 1995) .
COHERENCIA	La coherencia se define como el grado de relación funcional de una responsabilidad en una unidad de programa. La coherencia se basa en el principio de una única responsabilidad Single Responsibility Principle (SRP) el cual indica que: "una clase o módulo debe tener uno y sólo un motivo para cambiar" (Martin, 2002) . En términos generales una clase está compuesta por elementos que pueden ser métodos y atributos. En el contexto de la coherencia, una responsabilidad se refiere a una secuencia interactiva de métodos implicados para cumplir o alcanzar una meta u objetivo.
MARCO ORIENTADO A OBJETOS (MOO)	Es un conjunto semi-completo de clases en colaboración, que incorpora un diseño genérico el cual puede ser adaptado a una serie de problemas específicos para producir nuevas aplicaciones hechas a la medida (Mattsson, Bosch, & Ronneby, 1997).
META DE VALOR	En el contexto de esta investigación una meta u objetivo de valor a nivel de función, como unidad de programa, es realizar una única operación funcional. A nivel de módulo o subsistema, una meta de valor u objetivo es el de realizar un caso de uso o requerimiento especifico. A nivel de sistema una meta de valor u objetivo es resolver una aplicación completa o proceso de negocio planteada por un cliente o usuario.

MICROLITO	Un Microlito (Microlith) es un servicio de instancia única en el que las llamadas a métodos sincrónicos se han convertido en llamadas REST sincrónicas y el acceso a la base de datos permanece bloqueado (Bonér, 2017).
MICROSERVICIO	Son servicios pequeños y autónomos, cada uno con sus propios datos y aislados de forma independiente, escalables y resistentes a fallas (Eisele, 2016). Los microservicios ahora son una nueva tendencia en la arquitectura de software, que enfatiza el diseño y desarrollo de software escalable y altamente mantenible. Los microservicios administran la creciente complejidad al descomponer funcionalmente los grandes sistemas en un conjunto de servicios independientes (Dragoni et al., 2017)
MODULARIDAD	Es una medida de la capacidad de reuso de una unidad de software que puede ser una función, una clase, un paquete, un subsistema o un sistema, y que se distingue principalmente por ser reusable, fácil de mantener y no requiere de otros módulos para obtener un resultado o una meta de valor para el usuario final (Erdemir & Buzluca, 2014).
MONOLITO	La aplicación de software monolítico es una aplicación de software compuesta de módulos que no son independientes de la aplicación a la que pertenecen (Dragoni et al., 2017).
REFACTORIZACIÓN	La refactorización es aquel proceso cuyo objetivo es modificar la arquitectura de clases de un sistema, más no modificar la funcionalidad del mismo. Para asegurarse que esto no ocurra, una refactorización siempre cuenta con precondiciones y pos-condiciones que se deben de cumplir antes y después de aplicar una transformación. Por tanto, la refactorización no ayuda a un sistema actual sino a sistemas futuros que se extiendan del sistema actual (Fowler, 2002).

SECUENCIA INTERACTIVA	En el contexto de esta investigación una secuencia interactiva es: "El conjunto de métodos que responden a una secuencia sucesiva de mensajes que son enviados desde los objetos. La secuencia inicia con un evento que activa al método iniciador, considerándose como tal todos aquellos métodos públicos de las clases públicas de alto nivel no abstractas de un Marco Orientado a Objetos a través de una interacción de un actor, y termina con la meta de valor que se espera del caso de uso". Este conjunto de mensajes puede extraerse de un diagrama de secuencias del UML (Gallardo, 2018).
SERVICIO WEB	Es una unidad lógica para el intercambio de datos con otras unidades lógicas a través de una red usando un grupo de protocolos y estándares tales como XML, WDSL y SOAP (Vinay & Jain, 2012).
SOA	La arquitectura orientada a servicios (SOA) es un enfoque de diseño en el que múltiples servicios colaboran para proporcionar un conjunto final de capacidades. En SOA un servicio generalmente significa un proceso de sistema operativo completamente separado. La comunicación entre estos servicios se produce a través de llamadas por medio de una red en lugar de llamadas de método dentro de un límite de proceso (Newman, 2015) .
TIEMPO DE RESPUESTA	Es el tiempo transcurrido desde que se envía una petición de servicio y se recibe su respuesta (Kalepu, Krishnaswamy, & Loke, 2004).

Capítulo 1: Introducción

1.1 Presentación

La arquitectura orientada a servicios SOA y el estilo arquitectónico de Microservicios, se basan en los servicios como el principal componente para implementar funcionalidades de negocios. Las arquitecturas basadas en servicios, tienen en común que son distribuidas; esto significa que los componentes de servicio se acceden de forma remota a través de algún tipo de protocolo de acceso remoto (por ejemplo, HTTP sobre Representacional State Transfer (REST), Simple Object Access Protocol (SOAP), Java Message Service (JMS), Microsoft Message Queue Server (MSMQ), Invocación de Método Remoto (RMI) etc.

La arquitectura de microservicios apareció recientemente como un nuevo paradigma para la programación de aplicaciones mediante la composición de sistemas monolíticos en pequeños servicios, cada uno de los cuales ejecuta sus propios procesos y se comunica a través de mecanismos ligeros. Este enfoque se ha basado en los conceptos de SOA (Newman, 2015) a partir de identificar los procesos y flujos de trabajo de la organización que atienden a una única responsabilidad o meta de valor.

Uno de los objetivos de los sistemas distribuidos basados en servicios, es el reuso de componentes así como obtener calidad de cada uno de ellos. Los Microservicios ayudan a cumplir con esos objetivos debido a que son una tendencia en la arquitectura de software cuyo enfoque es el software escalable y altamente mantenible. Descomponen grandes sistemas en servicios independientes que atienden una única responsabilidad o meta de valor permitiendo un acoplamiento flexible y una alta cohesión mejorando la modularidad de los servicios.

Uno de los beneficios de la arquitectura basada en Microservicios es la velocidad de desarrollo, el tiempo de lanzamiento de los sistemas y la entrega continua. De esta forma, pueden proporcionar una base para construir sistemas elásticos y resilientes.

En el CENIDET se han realizado trabajos de investigación que se encuentran relacionados con el presente tema de tesis. Los trabajos de investigación: “Generación de Servicios Web desde Marcos Orientados a Objetos a partir de Clases Colaborativas en Casos de Uso (MOOaSW)”(León, 2009), “Transformación de Marcos de Aplicaciones Orientados a Objetos hacia Marcos con Arquitectura Orientada a Servicios” (Legorreta, 2017) y “Generación de Servicios Web desde Marcos Orientados a Objetos con el Enfoque de Clases Internas”

(Gallardo, 2018) utilizan un método de análisis estático, con casi nula intervención humana. Siguen las reglas gramaticales del lenguaje java, el análisis rastrea totalmente el código del Marco Orientado a Objetos (MOO) original para identificar los casos de uso que lo integran. Cada caso de uso es considerado un componente reusable ya que satisface una capacidad o requerimiento del dominio o negocio. De este modo, cada componente reusable identificado, es implementado en un caso de uso del MOO original y envuelto en una capa SOA para implementar y desplegar los Servicios Web que integran al Marco de Servicios Web de salida.

Para cada caso de uso identificado en el MOO de entrada, el trabajo de (León, 2009) respeta las relaciones arquitecturales entre clases como se definen en el MOO original. Así la arquitectura interna de clases resultante para cada servicio web se organiza con el conjunto de clases relacionadas y sus funciones y datos que son utilizados en la secuencia interactiva del Caso de Uso como en el MOO original. Mientras que el trabajo de (N. Legorreta, 2017) integra la totalidad del código de cada servicio web que participa en la secuencia interactiva de los casos de uso correspondientes, en una única clase envolvente, la cual contiene la totalidad de las funciones y datos que participan en la secuencia interactiva del caso de uso correspondiente. Este trabajo toma ventaja en su rendimiento y factor de acoplamiento contra el trabajo de (León, 2009), ofreciendo mejores tiempos de respuesta y nulo acoplamiento entre clases, sin embargo, pierde en los factores de calidad coherencia y cohesión.

El último de los trabajos de este tipo desarrollados en el laboratorio de Ingeniería de Software del CENIDET fue el de (Gallardo, 2018). Este último fue inspirado en las ventajas y desventajas de los dos primeros trabajos de antecedente, en busca de mejorar la modularidad de los Servicios Web que integran los Marcos de Servicios Web con arquitectura SOA de salida. Para lograr este objetivo, el enfoque arquitectural para cada servicio web generado fue el agrupamiento racional de clases internas, cada una integrada con sus funciones y datos que participan en las secuencias interactivas correspondientes a cada caso de uso. De esta forma la modularidad y el rendimiento se ven favorecidos contra los dos trabajos de antecedente por sus niveles balanceados de coherencia, cohesión, factor de acoplamiento y el tiempo de respuesta en su arquitectura interna de cada servicio web generado.

En este trabajo, se propone un ***método o procedimiento para obtener un marco de Microservicios basados en el estilo arquitectónico REST, traduciendo el código de servicios web escritos en lenguaje Java y que fueron obtenidos al identificar la secuencia de sus casos de uso***. El método obtiene microservicios por cada uno de los enfoques realizados en los trabajos de antecedentes y cumplen con características balanceadas de coherencia, cohesión,

factor de acoplamiento y tiempo de respuesta para lograr un equilibrio en estos atributos de calidad. El propósito que se persigue es demostrar el rendimiento de los Servicios Web y Microservicios enfocándose en el tiempo de respuesta con base en los protocolos de comunicación para determinar cuál tiene un mejor rendimiento. Para esto, es necesario desplegar los microservicios mediante el estilo arquitectural REST.

El método desarrollado en esta tesis se implementó en una extensión a la herramienta SR2-Transforming para dar soporte a la Reingeniería de Marcos de Aplicaciones Orientados a Objetos hacia Marcos de Microservicios.

1.2 Organización del documento de tesis

El contenido del presente documento de tesis está organizado de la siguiente manera: en el capítulo dos se presenta la descripción de los objetivos, el planteamiento del problema, los alcances y las limitaciones del presente documento de tesis. Se describe la situación actual en temas de microservicios, seguido por los trabajos relacionados y trabajos antecedentes realizados en el CENIDET.

El capítulo tres incluye el marco teórico y un acercamiento a lo que son los microservicios, sus características y ventajas.

El cuarto capítulo contiene la descripción del método para migrar a Microservicios así como el Análisis y Diseño del mismo.

El quinto capítulo incluye la Implementación del Método para migrar a Microservicios integrado a la herramienta "SR2-Transforming".

En el sexto capítulo se realiza el diseño experimental, el planteamiento de las hipótesis y el plan de pruebas.

En el séptimo y octavo capítulos se incluye el proceso de desarrollo del caso de prueba, los resultados obtenidos y el análisis de los mismos.

Por último en el noveno capítulo, se concluyen los resultados, los problemas presentados y se proponen los trabajos futuros a desarrollar.

Capítulo 2: Marco de Referencia

2.1 Planteamiento del problema

La arquitectura orientada a servicios (SOA) es un paradigma para organizar y utilizar capacidades distribuidas, que pueden estar bajo el control de diferentes propietarios del dominio. Los servicios web son la implementación de una arquitectura orientada a servicios y describen parte de un sistema de aplicación que puede ser consumido por separado por varias entidades. Los servicios web intercambian información a través de protocolos, por ejemplo SOAP y HTTP basado en REST independientemente del lenguaje y plataforma en la que fueron construidos a través de redes de computadoras como el internet.

Debido a que la concisión y la eficacia de los servicios web no son preponderantes en la arquitectura SOA y esas características se ven afectadas por el protocolo de comunicación, se presenta el problema de bajo rendimiento. Por lo que en este proyecto de tesis, se pretende mejorar la modularidad de los servicios por medio de la transformación de servicios web hacia microservicios, así como mejorar el rendimiento de los microservicios con base en el tiempo de respuesta por medio del protocolo de comunicación HTTP basado en REST.

2.2 Objetivos de la tesis

El objetivo general del presente trabajo de tesis es el siguiente:

- El objetivo de este proyecto de tesis es desplegar Microservicios de software bajo el estilo arquitectónico REST, mediante código en lenguaje java anotado que opere bajo el protocolo HTTP basado en REST.

Los objetivos específicos de este trabajo de tesis son:

- Específicamente, se requiere mejorar la modularidad de los Servicios que conforman Marcos de Microservicios (MM) basados en REST. Mediante el agrupamiento racional de funciones y atributos de clase, que conserven ventajas de modularidad tales como encapsulamiento, ocultamiento, coherencia, cohesión e independencia que ofrecen mayores ventajas de rendimiento con base en el tiempo de respuesta mediante el uso del protocolo de comunicación HTTP basado en REST.

Para este objetivo, en los antecedentes a esta investigación de tesis ya se ha trabajado en mejorar la modularidad de los Servicios Web por lo tanto se pretende atender los mismos atributos para el caso de los Microservicios.

- Extender la funcionalidad del "SR2-Transforming", para dar soporte a la generación de Microservicios desde marcos de Aplicaciones Orientados a Objetos.

2.3 Justificación

Los Servicios Web bajo arquitectura SOA y los Microservicios basados en REST, ambos enfoques tienen ventajas y desventajas comparativas de una contra la otra. A diferencia de las arquitecturas monolíticas ambos enfoques, con un correcto diseño, tienen una única responsabilidad es decir que atienden a un único requerimiento.

El estilo arquitectónico de Microservicios es un patrón en el que aplicaciones complejas se componen de procesos pequeños e independientes, que representan una única función empresarial y que se comunican utilizando protocolos ligeros, por ejemplo HTTP basado en REST, el cual se piensa que tiene un mejor rendimiento con base en el tiempo de respuesta comparado con el protocolo SOAP para el caso de los Servicios Web.

Los microservicios pueden ser desplegados independientemente, ser desconectados cuando no sean requeridos en un sistema y no tienen impacto en cualquier otro microservicio, entre sus principales características se pueden mencionar las siguientes: cuentan con un dominio de problema pequeño, son construidos y desplegados por sí mismos, se ejecutan en su propio proceso, se integran mediante interfaces bien conocidas, son dueños de sus base de datos o almacén de datos.

Mientras que los microservicios utilizan colas de mensajes, en SOA el bus de servicios empresariales (ESB) puede llegar a ser un único punto de falla. El ESB es un estilo de integración de arquitecturas que permite la comunicación por medio del bus de comunicaciones, el cual consiste de una variedad de conexiones punto-a-punto entre proveedores y consumidores. Si uno de los servicios responde con mayor lentitud podría obstruir nuevas peticiones de servicio en el ESB. En la plataforma de microservicios es mucho mejor la tolerancia a fallas, afectando únicamente los microservicios con falla, los demás microservicios continuarán manejando peticiones de servicio(Xiao & Qiang, 2016).

En SOA, los servicios comparten el almacenamiento de datos, mientras que en microservicios cada uno puede tener su propio almacenamiento de datos independientemente. Compartir el almacenamiento de datos tiene ventajas y desventajas. Por ejemplo, una ventaja es que los datos pueden ser reutilizados por todos los servicios, mientras que una desventaja es que compartir datos ocasiona fuerza de acoplamiento dentro de los servicios.

2.4 Alcances y limitaciones de la investigación

Los alcances obtenidos durante el presente desarrollo de investigación son los siguientes:

- El método desarrollado en este trabajo de tesis se implementó en la herramienta SR2-Transforming.
- El código de los Microservicios se genera de forma automática y se envuelve en la capa REST.
- El despliegue de los microservicios se realiza de forma independiente.
- Se determinó la influencia del tiempo de respuesta al consumir los Microservicios contra los Servicios Web utilizando el mismo enfoque.
- Los resultados fueron interpretados mediante un análisis estadístico.

Las limitaciones de la investigación de desarrollo tecnológico son los siguientes:

- El método propuesto no contempla el pre-procesamiento al MOO de origen.
- El alcance de los microservicios se limita al despliegue de los mismos, el método no contempla la gestión: el registro, balanceo de carga, la seguridad y el uso de contenedores.
- El método no incluye la verificación de errores lógicos ni sintácticos que pudiera tener el código del marco de aplicaciones orientado a objetos de origen.
- El método al implementarlo en la herramienta sólo realiza transformaciones de marcos de aplicaciones orientadas a objetos desarrollados en el lenguaje de programación *Java* versión 1.5.

- Las pruebas de rendimiento medido en términos de tiempo de respuesta, se realizaron únicamente con los casos de pruebas del trabajo de antecedente de (Gallardo, 2018).

- Las pruebas solo fueron realizadas bajo dos Marcos Orientados a Objetos de origen, el MOO *Statistic* y el MOO que contiene acceso a base de datos *PSPCenidet*.

- Este es un trabajo exploratorio encaminado a mostrar un método inicial para alcanzar el objetivo que apertura alguna futura línea de investigación.

2.5 Situación actual de los microservicios

La arquitectura de software es lo que permite a los sistemas evolucionar y proporcionar un cierto nivel de servicio hacia otros sistemas. En ingeniería de software, la arquitectura se preocupa de proporcionar un puente entre la funcionalidad del sistema y los requisitos de calidad que el sistema debe cumplir. En las últimas décadas, la arquitectura de software se ha estudiado a fondo, y como resultado, los ingenieros de software han ideado diferentes formas de realizar sistemas que proporcionen una amplia funcionalidad y logren satisfacer una amplia gama de requisitos (Dragoni et al., 2017).

Los problemas asociados con el desarrollo de software a gran escala se experimentaron por primera vez alrededor de la década de 1960 (Brooks, 1975). Las referencias al concepto de arquitectura de software también comenzaron a aparecer alrededor de la década de 1980. Sin embargo, una base sólida sobre el tema fue establecida en 1992 por Perry y Wolf (Perry & Wolf, 1992). Su definición de arquitectura de software era distinta a la del diseño de software, y desde entonces se ha generado una gran comunidad de investigadores que estudian la noción y las aplicaciones prácticas de la arquitectura de software, permitiendo que los nuevos conceptos sean ampliamente adoptados por la industria y la academia.

El advenimiento y la difusión de la Orientación a Objetos, a partir de la década de 1980 y, en particular, en la década de 1990, trajo su propia contribución al campo de la arquitectura de software. El clásico de Gamma (Gamma, Helm, Johnson, & Vlissides, 2002) cubre el diseño de software Orientado a Objetos y plantea cómo traducirlo en código presentando una colección de soluciones recurrentes, llamadas patrones.

La atención a la separación de las responsabilidades en un sistema, ha llevado recientemente a la aparición de la llamada Ingeniería de Software basada en componentes (CBSE) (Szyperski,

Bosch, & Weck, 1999), que ha otorgado un mejor control sobre el diseño, la implementación y la evolución de los sistemas de software. La última década ha visto un nuevo cambio hacia el concepto de servicio (Booth et al., 2004) y posteriormente hacia la evolución a microservicios.

El computo orientado al servicio (SOC) es un paradigma emergente para la computación distribuida y el procesamiento del comercio electrónico que tiene su origen en la informática orientada a objetos y componentes. Se ha introducido para aprovechar la complejidad de los sistemas distribuidos e integrar diferentes aplicaciones de software (MacKenzie, Laskey, McCabe, Brown, & Metz, 2006).

La primera generación de Arquitecturas Orientadas a Servicios (SOA) estableció requisitos desalentadores para los servicios (por ejemplo, descubrimiento y contratos de servicio), y esto limitó la adopción del modelo SOA. Los Microservicios son la segunda generación sobre el concepto de SOA y SOC. El objetivo es eliminar niveles innecesarios de complejidad para centrarse en la programación de servicios simples que implementan efectivamente una funcionalidad única. Al igual que la programación orientada a objetos, el estilo arquitectónico de microservicios necesita herramientas ad-hoc para apoyar a los desarrolladores y, de manera natural, conduce a la aparición de patrones de diseño específicos (Safina, 2016).

La arquitectura de microservicios apareció últimamente como un nuevo paradigma para la programación de aplicaciones mediante la composición de pequeños servicios, cada uno de los cuales ejecuta sus propios procesos y se comunica a través de mecanismos ligeros. Este enfoque se ha basado en los conceptos de SOA (MacKenzie et al., 2006).

El término "Microservicios" se introdujo por primera vez en 2011 en una presentación de arquitectura como una forma de describir las ideas comunes de los participantes en los patrones de arquitectura de software (Fowler, 2006). Hasta entonces, este enfoque también se conocía con diferentes nombres. Por ejemplo, Netflix utilizó una arquitectura muy similar bajo el nombre de SOA de grano fino (A Wang, 2013).

La arquitectura de Microservicios ganó popularidad hace poco tiempo y puede considerarse apenas en sus inicios, ya que aún no hay consenso sobre qué son realmente los Microservicios. M. Fowler y J. Lewis proporcionan un punto de partida al definir las principales características de los microservicios (Fowler, 2006). Sam Newman (Newman, 2015) se basa en el artículo de M. Fowler y presenta consejos y mejores prácticas con respecto a algunos aspectos de la

arquitectura antes mencionada. L. Krause en su trabajo (L. Krause, 2016) discute patrones y aplicaciones de microservicios.

La arquitectura de microservicios es un estilo arquitectónico que ha ido ganando popularidad en los últimos años, tanto en lo académico como en lo empresarial y de negocio. En particular, el cambio hacia los microservicios es delicado y se debe ir paso a paso, con cuidado para realizar una correcta refactorización de los sistemas.
Existen trabajos de investigación que están relacionados de cierta forma con los temas estudiados en la presente tesis, los cuales se describen a continuación:

2.6 Trabajos relacionados

Para esta investigación se analizaron siete trabajos relacionados con el tema propuesto, el análisis se realizó con artículos de los últimos 8 años tomados de una muestra heterogénea de artículos que han sido publicados y que tienen una similitud al trabajo desarrollado en esta investigación. Estos trabajos se describen a continuación:

En el estudio *"Service Identification in Interorganizational Process Design"* (Bianchini, Cappiello, De Antonellis, & Pernici, 2014) se presenta P2S (Process-to-Services), una metodología asistida por computadora para permitir la identificación de servicios que componen un proceso de negocio colaborativo. La metodología se basa en métricas definidas para configurar la granularidad, cohesión, acoplamiento y reutilización del servicio. También se describe un prototipo de herramienta basado en la metodología con referencia a un caso real.

La metodología P2S tiene como objetivo proporcionar un enfoque semiautomático para apoyar a los diseñadores a analizar un proceso de negocio e identificar un subconjunto de funcionalidades que se pueden exportar como servicios. La metodología está diseñada para ser aplicada en los primeros pasos de un ciclo de vida SOA, en un enfoque top-down, o en cualquier caso en el que no existe una cartera de servicios disponibles y el objetivo es identificar subprocesos adecuados que se ajusten propiedades bien definidas, tales como: bajo acoplamiento y alta cohesión. La metodología P2S se usa para abordar los problemas relacionados con los enfoques "meet-in-the middle", es decir, en situaciones en las que el servicio describe un proyecto de trabajo en progreso para la identificación del servicio intermedio, donde se recuperan los servicios disponibles dejando la identificación obtenida a través de la metodología P2S moviéndose sobre un árbol de agregación. Este proyecto muestra

cómo los resultados de la metodología pueden utilizarse como punto de partida para diseñar una herramienta que sea capaz de apoyar al diseñador a través de todas las actividades del ciclo de vida SOA.

En el trabajo de investigación *"Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud"* (Villamizar et al., 2015) se analizó y probó el patrón de arquitectura de Microservicios, utilizado durante los últimos años por grandes empresas de Internet como Amazon, Netflix y LinkedIn para desplegar grandes aplicaciones en la nube como un conjunto de pequeños servicios que se pueden desarrollar, probar, desplegar, escalar, operar y actualizar de forma independiente. Se utilizó el enfoque top-down para el análisis del negocio e identificar a través de un proceso de ingeniería directa los servicios de negocio.

En el estudio *"Extracting SOA Candidate Software Services from an Organization's Object Oriented Models"* (Yousef, Adwan, & Abushariah, 2014) se propone una metodología para identificar servicios a partir de un conjunto de diagramas de clases y modelos de casos de uso para generar un modelo orientado a servicios. Una extensa evaluación de los servicios generados ha demostrado que estos servicios se ajustan a los principios de Arquitectura Orientada a Servicios (SOA) y proporcionan una metodología sencilla que puede reutilizar la lógica empresarial que reside en aplicaciones legadas para migrar a sistemas basados en SOA.

Se propone un nuevo enfoque para identificar los servicios a partir de un conjunto de diagramas de clases y modelos de casos de uso de la organización. Estos modelos proporcionan la información necesaria para construir una matriz CRUD que, después de realizar un análisis de afinidad, se pueden agrupar funciones y entidades que comparten operaciones de creación y actualización. Esto se ajusta a los principios SOA tales como acoplamiento e interoperabilidad. Además, el uso de clases como entidades garantiza la reutilización y la abstracción de la lógica de negocio.

En el estudio *"Use Case-Based Service-Oriented Analysis and Modeling"* (Liu, Fu, & Luo, 2011), se presenta un nuevo método Top-Down de análisis y diseño orientado a servicio SOAD basado en el enfoque de Casos de Uso (UC), por lo que un modelo de servicio está diseñado para facilitar la identificación del servicio a partir del script de caso de uso. Además, se introduce un proceso para lograr una identificación de servicio de alta calidad.

El método tiene algunas ventajas para identificar servicios, tales como bajo acoplamiento y alta cohesión. El método SOAD, se basa en la tecnología de adquisición de requisitos de casos de uso.

En el estudio "*The ENTICE Approach to Decompose Monolithic Services into Microservices*" (Kecskemeti, Marosi, & Kertesz, 2016), se presenta una metodología para descomponer los servicios monolíticos en varios microservicios. La metodología propuesta aplica varios resultados del proyecto ENTICE (es decir, su síntesis de imágenes y herramientas de optimización). Por último, el artículo proporciona información sobre cómo estos resultados ayudan a revitalizar los servicios monolíticos pasados y qué técnicas se aplican para ayudar a futuros desarrolladores de microservicios.

El objetivo de esta investigación es proponer una metodología que pueda utilizarse para dividir un servicio monolítico en pequeños microservicios que posteriormente puedan ser utilizados para aumentar la elasticidad de aplicaciones a gran escala o para permitir composiciones más flexibles con otros servicios.

En el estudio *"Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems"* (Levcovitz, Terra, & Tulio Valente, 2016), se describe una técnica para identificar microservicios en sistemas monolíticos. Se considera un sistema monolítico a una aplicación de software compuesta de módulos que no son independientes de la aplicación a la que pertenecen. Y se aplicó con éxito en un sistema monolítico bancario de 750 KLOC, que gestiona transacciones desde 3,5 millones de cuentas bancarias y realiza casi 2 millones de autorizaciones por día. La evaluación realizada muestra que la técnica propuesta es capaz de identificar microservicios en el sistema monolítico y se concluye que los microservicios pueden ser una alternativa prometedora para modernizar el sistema monolítico empresarial heredado.

La técnica comienza evaluando y clasificando las tablas de la base de datos en subsistemas comerciales, lo que solo exige acceso al código fuente y al modelo de la base de datos. Esta técnica se basa en una serie de refactorizaciones que tiene como objetivo modular la recuperación de código través del aislamiento de sus fragmentos. Esta técnica no aplica para los fragmentos de código a nivel de operación atómica.

Se obtiene un grafo para cada subsistema que ayudó considerablemente a evaluar las funciones, identificar y describir microservicios. Cabe hacer mención que este trabajo no fragmenta la base de datos y los asocia a los microservicios.

En la investigación *"Extraction of Microservices from Monolithic Software Architectures"* de (Mazlami, Cito, & Leitner, 2017), se presenta un modelo formal de extracción de microservicio para permitir la recomendación algorítmica de candidatos de microservicio en un escenario de refactorización y migración. El modelo formal se implementa en un prototipo basado en web. Una evaluación de desempeño demuestra que el enfoque presentado proporciona un rendimiento adecuado. La calidad de la recomendación se evalúa cuantitativamente mediante métricas personalizadas específicas de microservicio.

El objetivo de este trabajo es abordar algorítmicamente el problema de extracción de bases de código monolíticas. Para ello, se presentan tres estrategias formales de acoplamiento y las incorporan en un algoritmo de agrupamiento basado en gráficos. Las estrategias de acoplamiento se basan en información de bases de código monolíticas para construir representaciones gráficas de los monolitos que a su vez son procesadas por el algoritmo de agrupamiento para generar recomendaciones de candidatos potenciales de microservicios en un escenario de refactorización.

2.7 Trabajos antecedentes realizados en el CENIDET

Como antecedentes a esta investigación se han realizado en el CENIDET trabajos de tesis que tratan el mismo problema y comparten objetivos similares pero que tienen diferentes enfoques de solución. Estos son: *"Generación de Servicios Web desde Marcos Orientados a Objetos a partir de Clases Colaborativas en Casos de Uso (MOOaSW)" , "Transformación de Marcos de Aplicaciones Orientados a Objetos hacia Marcos con Arquitectura Orientada a Servicios (INLineCLASS)" y "Generación de Servicios Web desde Marcos Orientados a Objetos con el Enfoque de Clases Internas (MOOinMS)".*

Los tres trabajos plantean una estrategia alternativa para generar servicios web desde marcos de aplicaciones orientadas a objetos. Consisten en la construcción de Marcos de Servicios Web (MSOA´s) desde los casos de uso descubiertos en el código de Marcos Orientados a Objetos (MOO´s) de dominios, con la premisa que un Caso de Uso (CU) representa una capacidad como la unidad adecuada para producir un resultado de valor para el actor.

En el trabajo de tesis *"Generación de Servicios Web desde Marcos Orientados a Objetos a partir de Clases Colaborativas en Casos de Uso (MOOaSW)"*(León, 2009), se utiliza un método de análisis estático con casi nula intervención humana que rastrea totalmente el código del MOO siguiendo las reglas gramaticales del lenguaje java, considerando que éste es el lenguaje en el que el código nativo del MOO original está implementado. Cada componente reusable definido en un Servicio Web implementa un caso de uso del MOO original. Otro rasgo de este enfoque, radica en que la metodología usada permite descubrir los CU´s utilizados por actores externos que saben de las capacidades que deben soportar los procesos de negocios, así como por actores lógicos internos, unidades de programa u otros servicios web que interactúan e intercambian información para producir una aplicación de software completa. La arquitectura de clases resultante para cada servicio web es, únicamente, el conjunto de clases relacionadas y las funciones y datos que son utilizados en la secuencia interactiva del Caso de Uso, respeta las relaciones arquitecturales entre éstas como se definen en el MOO original.

En el trabajo de tesis *"Transformación de Marcos de Aplicaciones Orientados a Objetos hacia Marcos con Arquitectura Orientada a Servicios"* (Legorreta, 2017)**,** el enfoque es similar al trabajo descrito anteriormente sólo que la arquitectura de cada servicio web está integrada en una única clase envolvente, la cual contiene la totalidad de las funciones y datos que participan en la secuencia interactiva de un caso de uso.

El enfoque del trabajo de tesis *"Generación de Servicios Web desde Marcos Orientados a Objetos con el Enfoque de Clases Internas"* (Gallardo, 2018), consiste en mejorar la modularidad de los Servicios Web que conforman los Marcos de Servicios Web con arquitectura SOA obtenidos desde Marcos Orientados a Objetos, mediante la composición de clases internas y el agrupamiento racional de funciones que participan en secuencias interactivas de casos de uso, así como los datos que son manipulados por estas funciones. El propósito que se persigue es obtener servicios web con niveles balanceados de coherencia, cohesión, factor de acoplamiento y el tiempo de respuesta en su arquitectura interna, de tal manera que se logre un equilibrio entre estos atributos de calidad, para obtener un mejor desempeño y mejores condiciones de reuso y mantenimiento.

La investigación "Estudio Experimental para determinar la influencia del nivel de granulación de Servicios Web en factores de calidad QoS"(Guadarrama Rogel, 2013). Consistió en un estudio experimental para medir el impacto que la estructura interna de Servicios Web tiene sobre un único atributo de calidad QoS 'el tiempo de respuesta'. En este estudio se

propusieron cuatro arquitecturas de Diseño Orientado a Objetos, que proporcionan una definición para crear un conjunto de Servicios Web de granulación gruesa, media, fina y muy fina. El Estudio consistió en estresar fuertemente cada uno de los casos de prueba bajo una gran cantidad de peticiones simultáneas (throughput) en un ambiente controlado de pruebas. Al final de la experimentación, se analizaron los resultados obtenidos y se realizaron conclusiones detalladas acerca del impacto que tiene cada arquitectura de diseño de servicios Web.

El producto resultado de estos tres trabajos de antecedente son implementados en una extensión a la herramienta SR2-Transforming para dar soporte a la Reingeniería de Marcos de Aplicaciones Orientados a Objetos, y punto de partida del método de transformación de esta investigación de tesis para lograr construir Marcos de Microservicios equivalentes.

2.8 Comparativa de los trabajos relacionados

La comparativa de estos trabajos relacionados contra el trabajo de esta investigación de tesis se basa principalmente en los siguientes criterios de evaluación:

Objetivo: Se realiza una comparativa entre los objetivos que se plantean en los trabajos estudiados, contra el objetivo que se persigue en esta investigación.

- Identificar componentes de software. (IDC)
- Obtener un servicio de negocio. (SN)
- Identificar servicios. (IS)
- Obtención de servicios web. (SW)
- Obtención de microservicios. (MS)

Enfoque: Se analizó si el enfoque que persiguen los trabajos estudiados para lograr sus objetivos difieren del enfoque que se plantea en este trabajo de tesis.

- Top-Down. (T-D): Este enfoque identifica a través de un proceso de ingeniería directa casos de uso o servicio basándose en el modelo de negocio o de software.

- Botton-Up. (B-U): Este enfoque deriva casos de uso o servicios a través de un proceso de ingeniería inversa partiendo del análisis del código fuente de sistemas legados.

- Meet-in-the-middle. (M-M): Este enfoque es una combinación de las estrategias de Top-Down y Bottom-Up.

Proceso: Este criterio determina el grado de automatización del método propuesto para cada uno de los trabajos analizados contra el de esta investigación.

- Automático.(A)
- Semi-Automático. (SA)
- Manual. (M)

Entrada: Este criterio determina cual es la entrada al método para iniciar su proceso.

- Modelo de clases. (MC)
- Modelo de casos de uso. (MCU)
- Modelo de escenarios. (ME)
- Documentación del negocio. (DN)
- Modelo de proceso de negocio. (MPN)
- Código fuente. (CF)

Aportación: Este criterio determina el grado de innovación y originalidad de los trabajos relacionados.

- Una nueva técnica. (T)
- Una metodología. (MT)
- Un modelo. (MD)
- Una herramienta de software. (HS)
- Un marco de trabajo de soporte al desarrollo. (MTD)

Producto/Resultado: Es el criterio que determina la salida obtenida.

- SW: Servicios web
- MS: Microservicios
- MSOA: Marcos de Servicios Web con arquitectura SOA.

A continuación en la Tabla 1, se describe la evaluación de cada uno de los documentos relacionados en cada uno de los criterios establecidos en los puntos anteriores.

Tabla 1 Tabla comparativa de trabajos relacionados y trabajos de antecedente vs esta investigación

Trabajos relacionados	**Objetivo**	**Enfoque**	**Proceso**	**Entrada**	**Aportación**	**Producto/ resultado**
"Service Identification in Interorganizational Process Design" (Bianchini et al., 2014)	SN	T-D	SA	MPN	MT	SW
"Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud" (Villamizar et al., 2015)	MS	T-D	SA	MPN	MTD	MS
"Extracting SOA Candidate Software Services from an Organization's Object Oriented Models" (Yousef et al., 2014)	SW	T-D	M	MC+MCU	MT	SW
"Use Case-Based Service-Oriented Analysis and Modeling" (Liu et al., 2011)	SW	T-D	SA	MPN	MT	MSOA
"The ENTICE Approach to Decompose Monolithic Services into Microservices" (Kecskemeti et al., 2016)	MS	T-D	A	MPN	MT	MSOA
"Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems" (Levcovitz et al., 2016)	MS	T-D	M	CF	T	MS
"Extraction of Microservices from Monolithic Software Architectures" (Mazlami et al., 2017)	MS	T-D	A	CF	T	MS
Trabajos antecedentes	**Objetivo**	**Enfoque**	**Proceso**	**Entrada**	**Aportación**	**Producto/ resultado**
"Transformación de Marcos de Aplicaciones Orientadas a Objetos hacia Marcos con Arquitectura Orientada a Servicios" (Legorreta, 2017)	IS	B-U	A	CF	MT+AS	SW
"Generación de Servicios Web desde Marcos Orientados a Objetos con el Enfoque de Clases Internas" (Gallardo, 2018) (Gallardo, 2018)	IS+SW	B-U	A	CF	MT + HS	SW
Esta Investigación de Tesis (MSWeb2MMServ)	IS+MS	B-U	A	CF	MT + HS	MS

El trabajo que se describe en esta tesis, al cuál de aquí en adelante llamaremos ***MSWeb2MMServ***, obtiene Marcos de Microservicios desde el código de Marcos Orientados a Objetos (MOO), a través de cualquiera de los enfoques desarrollados en los trabajos de antecedente en (León, 2009), (Legorreta, 2017) y (Gallardo, 2018). La diferencia entre estos tres trabajos consiste en la arquitectura interna de cada Microservicio generado.

Los microservicios se generan mediante un método de transformación de servicios web y un algoritmo que implementa el método, el cual es integrado a la herramienta "SR2-Transforming" para generar microservicios. Además, se mejora la modularidad de los microservicios obtenidos ya que cada método descrito anteriormente con su enfoque correspondiente mejora los atributos de calidad tales como coherencia, cohesión y acoplamiento y los microservicios generados cuentan con esas características según el método seleccionado.

Las capacidades encontradas en este trabajo de tesis y comparadas contra los trabajos relacionados analizados son:

1. El proceso de transformación es automático.

2. La obtención de los microservicios ya cuenta con una mejora en las características de calidad tales como coherencia, cohesión, acoplamiento e independencia, mismas que fueron mejoradas en los trabajos que anteceden a esta investigación.

3. Se obtiene un mejor rendimiento con base en el tiempo de respuesta al envolver los servicios en una capa REST bajo el protocolo HTTP.

4. La obtención de microservicios se realiza a partir del código fuente del MOO de entrada.

5. Cada Microservicio está preparado para ser desplegado de forma independiente y cuenta con las características del estilo de microservicios como son la resiliencia, tolerancia a fallas, principio de única responsabilidad etc.

Capítulo 3: Marco Conceptual

3.1 Aplicación Monolítica

Una aplicación de software monolítica es una aplicación de software compuesta de módulos que no son independientes de la aplicación a la que pertenecen (Dragoni et al., 2017).

Una aplicación monolítica es una aplicación donde toda la lógica se ejecuta en un único servidor de aplicaciones. Las aplicaciones monolíticas típicas son grandes y están construidas por equipos múltiples, lo que requiere una cuidadosa orquestación en la implementación para realizar cada cambio en la aplicación.

La figura 1 muestra el ejemplo de una arquitectura monolítica.

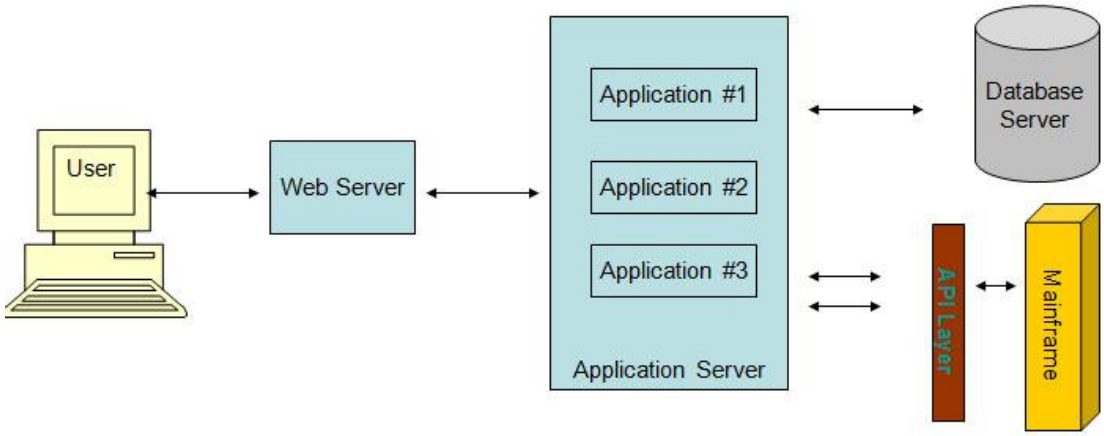

Figura 1 Ejemplo de la arquitectura monolítica (Shahir Daya , Valerie Lampkin, 2015)

3.2 Modularidad

La modularidad es una característica inherente de los sistemas orientados a objetos. El objetivo de la modularidad es descomponer la aplicación en objetos altamente modulares, cada uno de los cuales encapsula funcionalidades coherentes que permiten la máxima reutilización de los objetos. Una correcta modularidad y una encapsulación bien diseñada dan como resultado objetos altamente reutilizables (Shadija & Rezai, 2017).

3.3 Servicio

El Modelo de Referencia de OASIS para Arquitectura Orientada a Servicios define un servicio como "un mecanismo para habilitar el acceso a una o más capacidades, donde el acceso se proporciona usando una interfaz prescrita y se ejerce consistente con las restricciones y políticas especificadas por la descripción del servicio". En otras palabras, un servicio tiene algunas capacidades empresariales, tiene una interfaz bien definida y un contrato para acceder a ese servicio (Oasis, 2006).

3.4 Marco de Servicios Web

En el contexto de este trabajo de tesis, un Marco de Servicios Web se define como un conjunto de servicios web independientes, que contienen la experiencia y el conocimiento para satisfacer las necesidades de aplicaciones de dominios (Santaolaya, Fragoso, Rojas, Álvarez, & León, 2016).

3.5 Servicios Web

El consorcio W3C define los Servicios Web como sistemas software diseñados para soportar una interacción interoperable máquina a máquina sobre una red. Los Servicios Web suelen ser APIs Web que pueden ser accedidas dentro de una red (principalmente Internet) y son ejecutados en el sistema que los contiene.

El servicio web es un componente de software que toma datos de entrada y con base en ellos produce los datos de salida. Los servicios web se acoplan libremente y permiten a los desarrolladores crearlos, generarlos y componerlos en tiempo de ejecución, son interfaces que describen una colección de operaciones que son accesibles en red a través de la web estandarizada. Los protocolos del servicio web así como sus características se describen mediante el uso de un lenguaje basado en (XML). Sin embargo, los servicios web son sintácticamente usuales y se describen con estándares tales como el Protocolo simple de acceso a objetos (SOAP), el Lenguaje de descripción de servicios web (WSDL) y el Universal Description Discovery and Integration (UDDI) (Rostami & Kheirkhah, 2013).

La descripción del servicio web consiste en los parámetros técnicos, las restricciones y las políticas que definen los términos para invocar al servicio. Un servicio web se define como un WS de cuatro tuplas (*nombre, desc, in, out*), el *nombre* representa el nombre del servicio y se

utiliza como un identificador único; *desc* representa la descripción del servicio; *in* representa los parámetros de entrada de un conjunto de servicios; y *out* representa el conjunto de parámetros de salida de servicio.

SOAP es un protocolo para intercambiar información estructurada en un entorno descentralizado y distribuido. WSDL es un formato XML para describir los servicios web y sólo describe la interfaz sintáctica de los servicios web que, por sí sola, no puede utilizarse para la composición automática de servicios web. Así, los protocolos estándar semánticos como WSDL-S (Web Service Description Language-Semantic), WSMO (Web Service Modeling Ontology), OWL-S (Ontology Web Language-Service) y SAWSDL (Semantic Annotations para Web Service Description Language) han sido desarrollados para la composición automática de servicios web y la UDDI es un registro virtual que expone información sobre servicios web (Rostami & Kheirkhah, 2013).

3.6 SOA (Arquitectura Orientada a Servicios)

El enfoque y los productos de SOA surgieron de la necesidad de crear una arquitectura distribuida basada en componentes que promueva la agilidad empresarial, es decir, poder configurar rápidamente una solución para abordar una necesidad de negocio en particular y proporcionar cambios de solución a la medida ya que las necesidades del negocio evolucionan rápidamente (Xiao & Qiang, 2016).

Algunas de las primeras definiciones son:

- La arquitectura orientada a servicios (SOA) es un enfoque emergente que aborda los requisitos de la informática distribuida débilmente acoplada, basada en estándares y dependiente de protocolos. Normalmente, las operaciones comerciales que se ejecutan en una SOA comprenden una serie de invocaciones de estos diferentes componentes, a menudo de forma controlada por eventos o de forma asíncrona, que reflejan las necesidades subyacentes del proceso de negocio (Papazoglou, 2007)

- SOA: es un paradigma para organizar y utilizar capacidades distribuidas que tal vez estén bajo el control de diferentes dominios de propiedad (Oasis, 2006).

Los componentes principales en una arquitectura SOA según (Josuttis., 2007), son:

- Servicios SOA: un servicio es una pieza de funcionalidad empresarial autónoma: es decir, un servicio en SOA está diseñado para encapsular algunas funcionalidades que son significativas para la empresa. La funcionalidad puede ser simple (como la recuperación de una dirección de cliente) o compleja (por ejemplo, encapsulando el proceso comercial para cumplir con el pedido de un cliente).

- Enterprise Service Bus (ESB): Es la infraestructura que permite alta interoperabilidad entre proveedores de servicios y consumidores, proporciona servicios de valor agregado y ofrece una plataforma de alto rendimiento, confiabilidad y escalabilidad entre proveedores y consumidores .

- Gestión y supervisión de servicios: la gestión de servicios a través de un registro o repositorio de servicios proporciona descubrimiento de servicios y la gestión del ciclo de vida del mismo.

3.7 Microservicios

El término "Microservicios" describe un nuevo estilo de desarrollo de software que ha crecido a partir de las tendencias recientes en las prácticas de desarrollo y gestión de software, destinadas a aumentar la velocidad y la eficiencia del desarrollo de soluciones de aplicaciones. (Dragoni et al., 2017). Micro es un término relativamente ambiguo, que no siempre describe el tamaño de un servicio, ya que puede variar, por lo que en este trabajo de investigación la granularidad del servicio se define con base la capacidad del servicio para atender una responsabilidad o meta de valor y puede ser desde una función, o un requerimiento hasta un proceso de negocio.

Microservicios es un estilo arquitectónico en el que las aplicaciones de software grandes y complejas se componen de uno o más servicios más pequeños. Cada uno de estos microservicios se enfoca en completar una tarea que representa una capacidad de negocios pequeños. Los microservicios se pueden desarrollar en cualquier lenguaje de programación, se comunican entre sí utilizando protocolos de lenguaje neutral, como HTTP basado en (REST), o aplicaciones de mensajería (Shahir Daya, Van Duy, 2015).

3.7.1 Características de los microservicios

Los autores de (Harrison, 2011), mencionan como características clave de los microservicios las siguientes:

1. Diseño impulsado por el dominio. La descomposición funcional se puede lograr fácilmente utilizando el enfoque DDD (Diseño Dirigido por Dominios).

2. Principio de única responsabilidad. Cada servicio es responsable de una parte única de la funcionalidad y lo debe hacer de forma correcta.

3. Interfaz explícitamente publicada. Un servicio producido publica una interfaz que es utilizada por el consumidor del servicio.

4. DURS independientes (*Deploy*, *Update*, *Replace*, *Scale*). Cada servicio se puede implementar, actualizar, reemplazar y escalar de forma independiente.

5. Aspectos inteligentes y tubería de comunicación ligera. Cada microservicio posee su lógica de dominio y se comunica con otros a través de protocolos simples como HTTP sobre REST.

Los autores de (Lewis & Fowler, 2014), mencionan que las características generales de los microservicios son:

- Se componen a través de servicios
- Están organizados en torno a las capacidades empresariales
- Son productos no proyectos
- Son puntos finales inteligentes y tuberías ligeras de comunicación
- Tienen gobernanza descentralizada
- Usan una gestión de datos descentralizada
- Automatizan la infraestructura
- Diseñados para tolerar fallas
- Usan diseño evolutivo
- Son resilientes
- Son autónomos

Un ejemplo de la arquitectura de monolítica vs microservicios se muestra en la figura 2

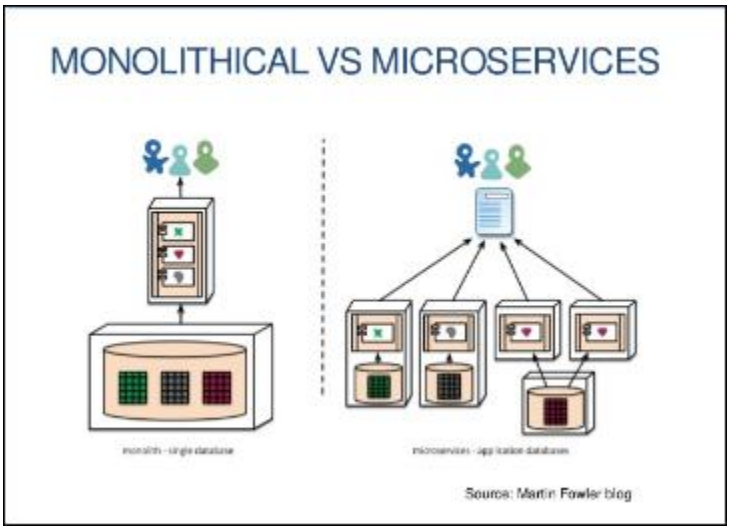

Figura 2 Ejemplo de la arquitectura monolítica vs microservicios (Lewis & Fowler, 2014)

3.7.2 Contextos delimitados

En la literatura de microservicios, *"la noción de contexto delimitado se origina en el diseño impulsado por el dominio y se presenta como un fundamento de diseño prometedor para los microservicios y sus límites"* (Newman, 2015) y (Lewis & Fowler, 2014). De acuerdo con este razonamiento, un microservicio debe corresponder a una funcionalidad única, identificada dentro del alcance de cada uno de ellos. Esto da como resultado microservicios escalables y mantenibles que se centran en una sola responsabilidad.

3.8 REST

REST (Representational State Transfer) es un estilo arquitectural de software para sistemas hipermedias distribuidos tales como la Web. El término fue introducido en la tesis doctoral de Roy Fielding en 2000, quien es uno de los principales autores de la especificación de HTTP.

En realidad, REST se refiere estrictamente a una colección de principios para el diseño de arquitecturas en red. Estos principios resumen cómo los recursos son definidos y diseccionados. El término frecuentemente es utilizado en el sentido de describir a cualquier interfaz que transmite datos específicos de un domino sobre el protocolo HTTP sin una capa adicional. Es posible diseñar un sistema de software de gran tamaño de acuerdo con la arquitectura propuesta por Fielding sin utilizar HTTP o sin interactuar con la Web. Así como

también es posible diseñar una simple interfaz XML+HTTP que no sigue los principios REST, y diseñar un modelo RPC (Leader, 2010).

Cabe destacar que REST no es un estándar, ya que es tan solo un estilo de arquitectura, pero está basado en estándares por ejemplo:

- HTTP
- URL
- Representación de los recursos: XML/HTML/GIF/JPEG etc.
- Tipos MIME: text/xml, text/html,

3.8.1 Principios de REST

REST se basa en los siguientes principios:

- Escalabilidad de la interacción con los componentes. La Web ha crecido exponencialmente sin degradar su rendimiento. Una prueba de ellos es la variedad de clientes que pueden acceder a través de la Web: estaciones de trabajo, sistemas industriales, dispositivos móviles.

- Generalidad de interfaces. Gracias al protocolo HTTP, cualquier cliente puede interactuar con cualquier servidor HTTP sin ninguna configuración especial. Esto no es del todo cierto para otras alternativas, como SOAP para los Servicios Web.

- Puesta en funcionamiento independiente. Los clientes y servidores pueden ser puestas en funcionamiento durante años. Por tanto, los servidores antiguos deben ser capaces de entenderse con clientes actuales y viceversa. Diseñar un protocolo que permita este tipo de características resulta muy complicado. HTTP permite la extensibilidad mediante el uso de las cabeceras, a través de las URIs, con la habilidad para crear nuevos métodos y tipos de contenido.

- Compatibilidad con componentes intermedios. Los más populares intermediaros son varios tipos de proxys para Web. Algunos de ellos, son las caches, que se utilizan para mejorar el rendimiento. Otros permiten reforzar las políticas de seguridad: firewalls. Y por último, otro tipo importante de intermediario es el gateway, que permite encapsular sistemas no propiamente Web.

3.9 SOAP

SOAP (Service Oriented Access Protocol) es un protocolo, que se desarrolló como una alternativa al estándar CORBA (Common Object Request Broker Architecture). Para garantizar el transporte de datos en SOAP, se utilizan protocolos como HTTP, SMTP, etc., y los datos son enviados en formato XML(Serrano, Hernantes, & Gallardo, 2014).

El principio de este enfoque es el siguiente: un proveedor de servicios publica una descripción o interfaz de servicio para el registro de servicio, de modo que el solicitante del servicio puede encontrar una instancia de servicio correcta y usarla (Castillo, Bernier, Arenas, Merelo, & Garcia-Sanchez, 2011).

Los servicios web basados en SOAP incluyen una variedad de estándares, tales como WSDL, WSBPEL, WS-Security, WS-Addressing (responsable del servicio web y direccionamiento de mensajes). Estos estándares fueron desarrollados por organizaciones de estandarización, como W3C y OASIS (Potti, 2012).

3.10 Ventajas y desventajas de REST vs SOAP

La tabla 2 muestra algunas de las ventajas y desventajas del estilo arquitectónico REST vs el protocolo SOAP según los autores (Richards, 2015).

Tabla 2 Ventajas y desventajas de REST vs SOAP

Protocolo	Ventajas	Desventajas
REST	• Los servicios Web RESTful son completamente sin estado, ello puede ser comprobado mediante el reinicio el servidor y comprobando si las interacciones son capaces de mantenerse. • Rest es muy ligero, sus respuestas contienen exactamente la información que se necesita. • Servicios RESTful proporcionan una buena infraestructura de almacenamiento en caché a través de HTTP método GET, esto mejorará el rendimiento, si los datos que devuelve el servicio Web no se altera con	• La seguridad es una deficiencia y puede llegar a ser una tarea muy difícil de implementar correctamente. • No hay un estándar en sus respuestas por lo que no se definen tipos de datos.

	frecuencia y no son de naturaleza dinámica. • Servicios REST son fáciles de integrar con los sitios web existentes y están expuestos a XML para que las páginas HTML pueden consumir la misma con facilidad. Casi no hay necesidad de refactorizar la arquitectura de sitio web existente.	
SOAP	• El Web Services Description Language (WSDL) contiene y describe el conjunto de normas comunes para definir los mensajes, los enlaces, las operaciones y la ubicación del servicio Web. WSDL es un tipo de contrato formal para definir la interfaz que ofrece el servicio Web. • SOAP requiere menos código de plomería o regulado de servicios REST, (es decir, las transacciones, la seguridad, la coordinación, direccionamiento, la confianza, etc.) • Es más seguro debido a que su implementación siempre o la mayoría de las veces se hace del lado del servidor. • Soporta varios protocolos y tecnologías, incluyendo WSDL, XSD, SOAP etc.	• Si se desea modificar la información en el servidor esto impacta de una forma negativa en los clientes. • Si no se cuenta con las herramientas correctas, la interpretación de la información puede tornarse demasiado compleja y difícil.

3.11 Orquestación y Coreografía

Los términos orquestación y coreografía describen dos aspectos de la creación de procesos de negocio a partir de la composición de servicios web (Peltz, 2003).

3.11.1 Orquestación

La orquestación se refiere a un proceso de negocio que puede interactuar con servicios web internos y externos. Las interacciones ocurren a nivel de mensaje, incluyen lógica de negocio y orden de ejecución de tareas, pueden abarcar aplicaciones y organizaciones para definir un modelo de proceso de múltiples pasos, transaccionales y de larga duración (Peltz, 2003).

La orquestación siempre representa control desde la perspectiva de una parte. Esto difiere de la coreografía, que es más colaborativa y permite a cada parte involucrada describir su parte en la interacción.

3.11.2 Coreografía

La coreografía rastrea las secuencias de mensajes entre varias partes y fuentes, y por lo general, son el intercambio de mensajes públicos que se producen entre los servicios web, en lugar de un proceso de negocio específico que ejecuta una sola parte (Peltz, 2003) .

Las coreografías son descripciones de alto nivel entre comunicaciones que suceden en un sistema, en contraste con la metodología típica de definir el comportamiento de cada servicio por separado. Las coreografías se utilizan en algunos modelos para interfaces de comportamiento, pero en realidad se originan en los esfuerzos del W3C de definir un lenguaje que describa el comportamiento global de los sistemas de servicio. Durante la última década, las coreografías se han investigado para apoyar un nuevo paradigma de programación llamado Programación coreográfica. En Programación Coreográfica, el programador usa coreografías para programar sistemas de servicio y luego se usa un compilador para generar automáticamente implementaciones compatibles. Esto arroja una metodología de corrección por construcción, que garantiza propiedades importantes como la libertad de interbloqueo y la falta de errores de comunicación (Dragoni et al., 2017).

3.12 Throughput

Throughput "*es la cantidad de solicitudes de servicio web atendidas en un período de tiempo determinado. El tiempo de respuesta de un sistema aumenta a medida que aumenta el rendimiento"*. Una decisión política importante es hacer un compromiso entre maximizar el rendimiento y minimizar el tiempo de respuesta (Kalepu et al., 2004).

De acuerdo con (Kalepu et al., 2004), en el contexto de esta investigación *Throughput* se refiere a la realización de tareas por un servicio informático o dispositivo durante un período específico de tiempo. Mide la cantidad de trabajo completado contra el tiempo consumido.

Capítulo 4: Descripción y Análisis del Método MSWeb2MMServ

4.1 Descripción del Método MSWeb2MMServ

Las comunidades de programación definen refactorizar como "introducir una transformación del código preservando su comportamiento." Esto se resume a mantener iguales sus APIs externas mientras cambia la manera en la que opera o se empaqueta su código. Por lo tanto, re-factorizar a microservicios significa integrar microservicios a una aplicación sin necesidad de cambiar lo que hace.

Actualmente en el laboratorio de Ingeniería de Software del CENIDET se han desarrollado tres métodos de transformación que permiten organizar el código de MOO´s, generando como resultado marcos de servicios web equivalentes. Estos tres métodos denominados: MOOaSWCU (León, 2009), Inlineclass (Legorreta, 2017) y MOOinMS (Gallardo, 2018) fueron implementados como algoritmos estratégicos para agrupar el código de entrada en diferentes niveles de granulación en el sistema SR2-Transforming. De esta manera se aprovechan las ventajas obtenidas en los trabajos de antecedentes que fueron implementados en la herramienta antes mencionada.

El método desarrollado en esta tesis fue implementado también en el SR2-Transforming, extendiendo sus capacidades y se desarrolló sobre el lenguaje java por lo que para re-factorizar a microservicios en este estudio, se utilizarán los principios del mismo lenguaje. Los pasos del método propuesto son los siguientes:

1. Definir los límites del servicio (se revisa cómo está empaquetada y creada la aplicación).

2. Re-factorizar el código de la aplicación.

3. Re-factorizar los datos.

El diagrama de procesos del método MSWeb2MMServ se muestra en la Figura 3:

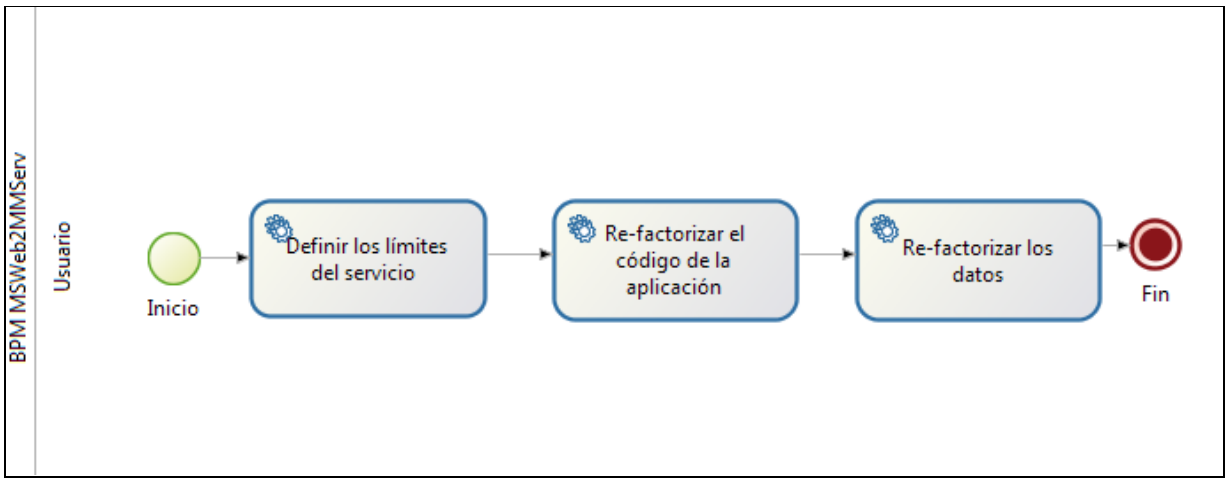

Figura 3 Diagrama de procesos del Método MSWeb2MMServ para refactorizar a microservicios.

4.1.2 Definir los límites del servicio

Definir los límites del servicio o contextos delimitados, es hacer que una parte del código se pueda tratar de forma aislada y trabajar sin afectar el resto del código base. Los contextos delimitados son excelentes, ya que por definición representan fronteras cohesivas y poco acopladas en una organización.

La idea de los contextos acotados es que cada servicio tenga una responsabilidad del mundo real y tenga una interfaz explícita.

En el lenguaje de programación java el concepto de paquete nos permite identificar y organizar los conceptos de espacios de nombres y relacionarlos con un proceso en particular, de esta manera en el paquete se puede agrupar un código similar que realiza una meta especifica.

Para organizar los paquetes que identifican la meta de valor u objetivo de un microservicio, el método propuesto realiza los pasos del proceso para definir los límites del servicio como se muestra en la figura 4:

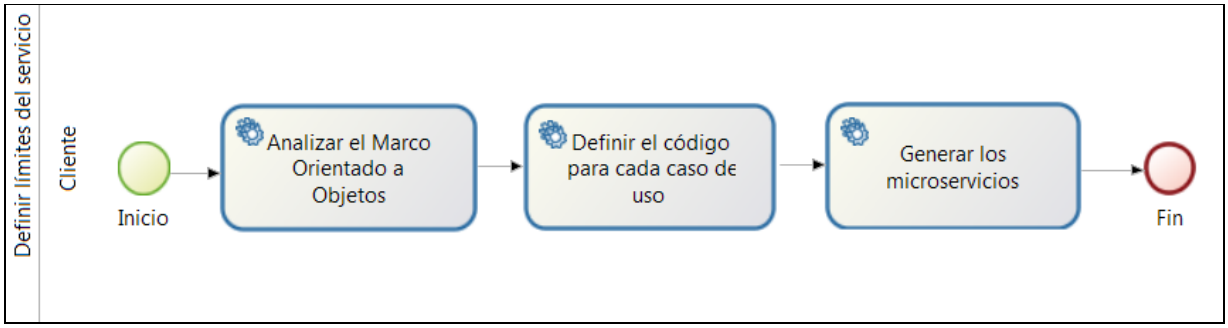

Figura 4 Diagrama de procesos para definir los límites del servicio

1. Analizar el Marco Orientado a Objetos: Este proceso consiste en ejecutar un analizador sintáctico (Gesser 2018) sobre el MOO de entrada y obtener un almacén de datos con la información de cada elemento de las clases definidas en el marco. A este repositorio se le denomina Diccionario de datos, y se siguen los mismos criterios que el procedimiento propuesto en (León, 2009).

   Se requiere que el código del marco se encuentre en lenguaje Java y esté libre de errores sintácticos. Esto implica que el código debe compilar sin arrojar errores antes de efectuar este procedimiento.

2. Definir el código para cada caso de uso: Consiste en seleccionar las clases (y sus elementos) del marco orientado a objetos original, que son necesarias para satisfacer cada caso de uso encontrado en el mismo. Los elementos de las clases son: constructores, métodos inicializadores, enumeraciones y miembros de clase (atributos, métodos, clases internas e interfaces internas) (Legorreta, 2017). Un caso de uso representa la meta de una interacción entre un actor y el sistema y contienen los requerimientos funcionales de los sistemas, en un formato fácil de seguir y fácil de leer (Lewis & Fowler, 2014). En este paso del proceso, cada caso de uso se identifica a través de encontrar las secuencias interactivas de los métodos que interactúan para alcanzar una meta de valor o capacidad del dominio.

   Al finalizar este paso se tendrán dos listas para cada caso de uso:

   Una lista que contiene las clases e interfaces que cada caso de uso requiere, llamada lista de clases.

Para cada nodo, representando a una clase de la lista anterior, se forma otra lista que incluye los elementos utilizados en el caso de uso. Se llamarán *listas de elementos de clase*. No incluirán los elementos que, aunque pertenezcan a la clase original, no sean útiles al caso de uso.

Para definir el código para cada caso de uso se deberán integrar correctamente las listas mencionadas con la información de las clases y de sus elementos. Para ello, se efectuarán las siguientes acciones propuestas en (Legorreta, 2017):

- Identificar los métodos iniciadores de cada caso de uso.
- Identificar precondición del caso de uso.
- Identificar los métodos de caso de uso y miembros de clase utilizados.
- Identificar las relaciones de agregación, composición y dependencia.
- Identificar cláusulas de manejo de excepciones utilizados.
- Depurar el contenido de las listas.
- Agrupar el código del caso de uso a una única entidad.

3. Generar los microservicios: A partir de haber identificado el código para cada caso de uso y al cual se le ha integrado el código correspondiente, según el método de refactorización que se haya seleccionado, se procede a generar un microservicio por cada caso de uso cuyo nombre será asignado con base en el id del método iniciador y el nombre calificado de su clase. Posteriormente, se procede a envolver el código de cada microservicio en su capa REST, integrar el código de las listas generadas y al final se generan los microservicios.

   El procedimiento inicia con la generación del microservicio por cada caso de uso identificado en los pasos anteriores, posteriormente el método crea tres archivos: El primero contiene la clase del controlador REST. Este controlador maneja las solicitudes HTTP en la implementación del servicio web basado en REST de Spring. El segundo archivo generado, se encarga de hacer ejecutable el microservicio y en él está integrado el código con la anotación SpringBootApplication. El tercer archivo generado integra el código nuevo con base a las listas de las clases seleccionadas y los asigna en nuevos archivos de código Java quedando almacenados en directorios específicos para cada caso de uso.

Los archivos quedan empaquetados dentro de una carpeta que representan el contexto limitado de cada microservicio generado y se genera un *war* por cada microservicio.

4.1.3 Re-factorizar el código de la aplicación

Una vez que se tenga cada servicio en empaquetado en un archivo WAR independiente, se deben encontrar las oportunidades para refactorizar y en un dado caso mejorar el código utilizado en cada servicio. Se pueden encontrar los siguientes casos:

- REST existentes: Es posible que se tenga servicios existentes que ya son compatibles con la arquitectura de microservicios, o que podrían hacerse compatibles. Una alternativa es separar cada servicio REST o JMS simple del resto de los WAR y luego desplegar cada servicio con su propio WAR.

- Servicios SOAP o EJB existentes: si ya se cuenta con servicios, probablemente se hayan creado con una estrategia funcional (como el patrón Facade). De ser así, se debe re-implementar la interfaz de la sesión del bean EJB o la interfaz JAX-WS como una interfaz JAX-WS. Para hacerlo, es necesario convertir representaciones de objetos a JSON.

  En los casos en los que no es un simple conjunto de operaciones CRUD, entonces se aplicarán estrategias diferentes para construir los servicios RESTful (tales como crear servicios funcionales simples) que implementen variantes del patrón de Comandos.

- Interfaces de Servlet/JSP simples existentes: muchos programas de Java en realidad son simples interfaces Servlet/JSP hacia tablas de bases de datos. Es posible que no tengan lo que se conoce como una Capa de "Objeto de Dominio", especialmente si siguen patrones de diseño como el patrón del Registro Activo.

  El método para este caso propone realizar lo siguiente:

  1.- Crear una capa o modelo de dominio que después se pueda representar como servicio RESTful. El objetivo es crear un modelo de dominio coherente y único para cada microservicio de negocio de contexto limitado. Se debe tener en cuenta que en ocasiones un microservicio de negocio puede estar físicamente compuesto por varios servicios que comparten un modelo del dominio. El modelo del dominio debe capturar

las reglas, el comportamiento, el lenguaje de negocios y las restricciones del contexto limitado o microservicio de negocio que representa.

2.- Identificar los objetos de dominio mediante la aplicación del Diseño Dirigido por el Dominio (DDD). Una entidad de dominio en DDD debe implementar la lógica del dominio o el comportamiento relacionado con los datos de entidad para una tarea específica. Por ejemplo, como parte de una clase de entidad de "pedido" debería implementar la lógica de negocios y las operaciones como métodos para tareas como agregar un elemento de pedido, la validación de datos y el cálculo total. Los métodos de la entidad se encargan de las invariables y las reglas de la entidad en lugar de tener esas reglas distribuidas por el nivel de aplicación.

3.- Una vez que se haya creado y empaquetado cada servicio nuevo en su propio WAR, se debe re-factorizar la aplicación cliente para usarla en un nuevo servicio o crear una interfaz totalmente nueva con JavaScript, HTML5 y CSS, o quizás como una aplicación móvil nativa.

4.1.4 Re-factorizar los datos

Refactorizar los datos de una aplicación monolítica hacia microservicios es considerada la parte más difíciles al momento de decidir migrar a una arquitectura de microservicios. Para esta investigación se tomaron en cuenta las mejores prácticas tomadas de varios artículos de investigación, así como de diferentes libros considerados en el estado del arte sobre temas de microservicios y que proponen varias estratégicas para re-factorizar los datos.

El método propuesto en esta tesis para el caso de la migración de los datos, se realizó de forma manual tomando en cuenta las consideraciones que se definen a continuación y tomadas de las fuentes de información mencionadas anteriormente:

1. Revisar el código de entrada ver qué partes se leen y escriben desde la base de datos. Una práctica común es tener una capa de repositorio, respaldada por algún tipo de marco como Hibérnate.

2. Una vez identificadas las partes de código que se leen y escriben desde la base de datos se debe agrupar el código en paquetes que representen los contextos acotados, se debe tener una capa de repositorio donde queden almacenados los paquetes de contextos

delimitados para cada microservicio y que representan las relaciones hacia las bases de datos.

Tener el código de asignación de la base de datos colocado dentro del código para un contexto dado, ayuda a comprender qué partes de la base de datos se usan para cada proceso del microservicio.

3. Una vez que se tienen los códigos de base de datos en contextos delimitados se deben realizar las siguientes técnicas para factorizar la base de datos esto implica realizar lo siguiente:

   - Compartir las tablas que intervienen entre uno o más microservicios, significa que dos o más microservicios de un sistema se comunican a través de la lectura y escritura en el mismo conjunto de tablas de una base de datos. Para realizar esto, se deben replicar copias de las tablas compartidas dentro de cada esquema de base de datos de cada microservicio. La figura 5 y 6 muestran un ejemplo de cómo compartir una tabla entre dos servicios que acceden a ella.

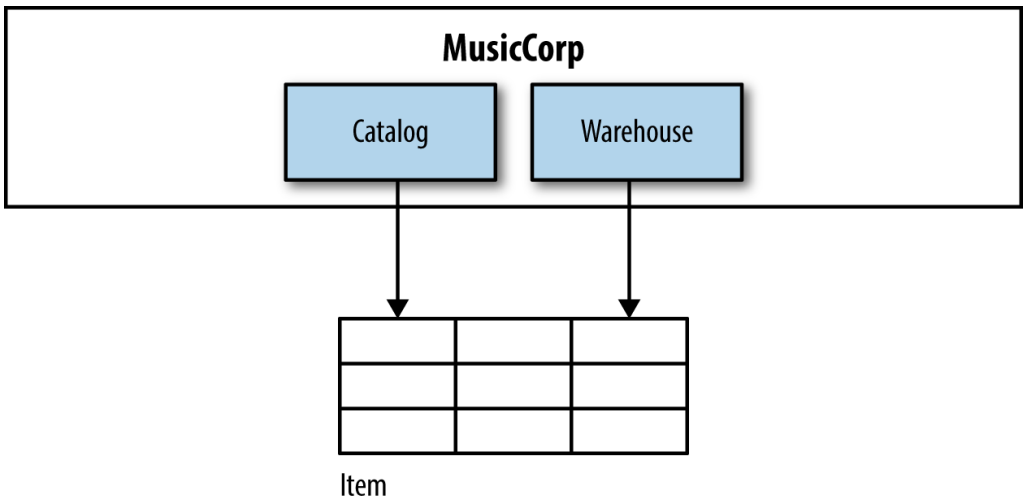

Figura 5 Ejemplo de una tabla compartida por dos servicios diferentes (Newman, 2015)

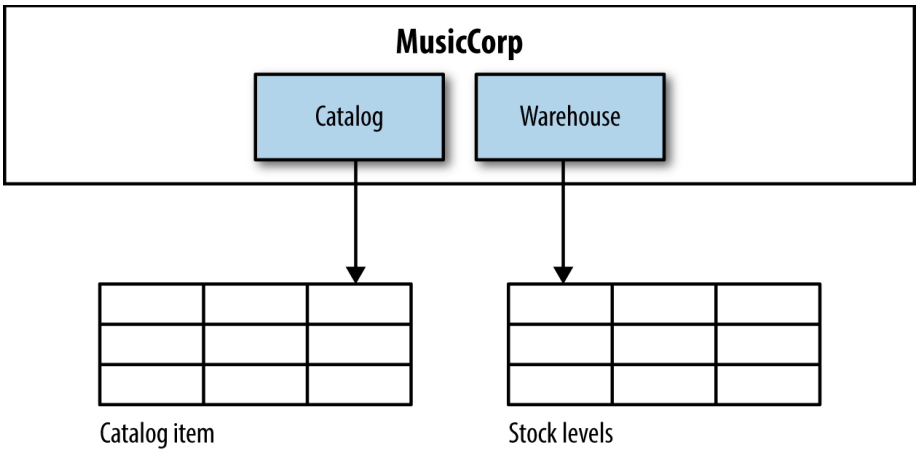

Figura 6 Ejemplo de copias de tablas replicadas en cada servicio (Newman, 2015)

- Implementar un patrón observador, es patrón debe verificar si hay un cambio en una tabla que modifique a una o más tablas de otros microservicios, el observador debe avisar al otro microservicio que hubo un cambio y actualizar las tablas correspondientes.

- Implementar desencadenadores de base de datos, esto es ejecutar código que automáticamente actualice las bases de datos en respuesta a eventos. Algunos eventos comunes de la base de datos que se deben tomar en cuenta son AFTER INSERT, AFTER UPDATE y AFTER DELETE. Se debe utilizar el código dentro de un disparador o iniciador de ejecución para actualizar las tablas integradas en diferentes microservicios.

4.2 Análisis de casos de uso del método MSWeb2MMServ

Para el diseño del método para migrar Servicios Web con protocolo SOAP obtenidos de un Marco Orientado a Objetos MOO hacia Microservicios basados en REST, se identificaron los casos de uso principales, a continuación, se describe cada uno de ellos.

1) Analizar el marco orientado a objetos.
2) Identificar el código para cada caso de uso.
3) Generar microservicios.

La figura 7 muestra la descripción del caso de uso principal para migrar a microservicios.

4.2.1 Diagrama principal de casos de uso del Método MSWEB2MMServ.

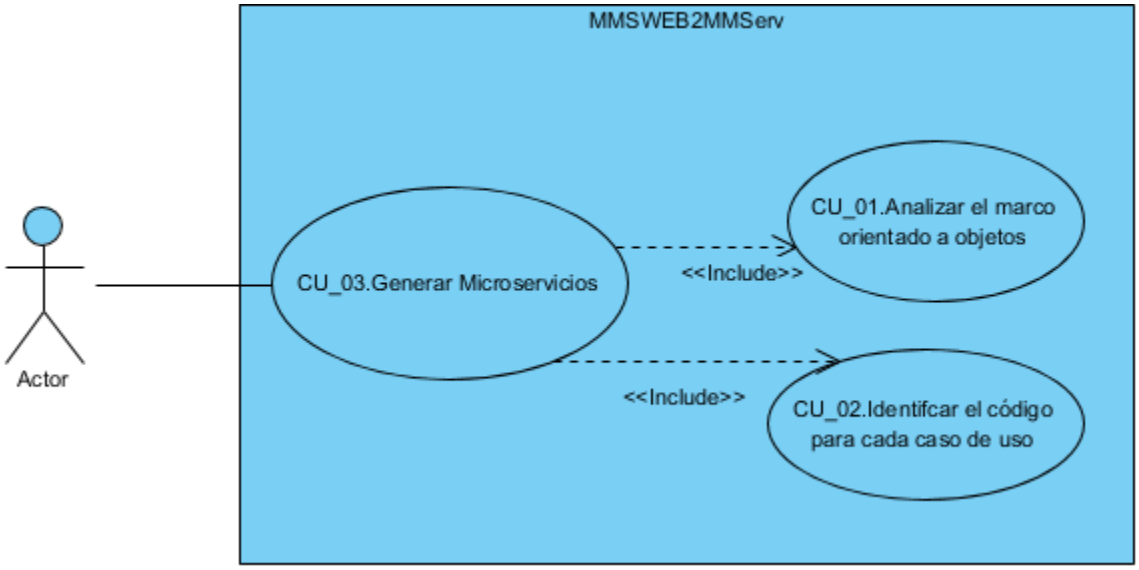

Figura 7 Diagrama principal de casos de uso del método MSWEB2MMServ .

4.2.2 CU_01 Analizar el marco orientado a objetos

Consiste en analizar el código del MOO de entrada a la herramienta SR2-Transforming a través de un analizador sintáctico, para obtener información de cada una de las clases contenidas en él. Esta información es almacenada en un diccionario de datos. El MOO debe estar en lenguaje java y no debe tener errores por lo que es importante que antes de realizar el análisis, éste debe ser compilado y verificar que funcione correctamente.

Los archivos *.java* del MOO deben estar contenidos dentro de un directorio, o en subdirectorios que dependan del nodo principal. Cada directorio representa un paquete y cada paquete contiene los archivos que incluyen las declaraciones de las clases del MOO.

Se debe acceder a la ubicación del directorio principal (raíz) y se inicia el procedimiento para leer cada uno de los archivos que estén contenidos en él para obtener las relaciones de cada una de las clases. La información obtenida se registra en un diccionario de datos para después ser usada en la generación de los casos de uso que atienden a una responsabilidad o una meta de valor. La tabla 3 muestra la descripción del caso de uso.

Tabla 3 Descripción del caso de uso Analizar el marco orientado a objetos

CU_01	Analizar el marco orientado a objetos
Descripción:	Ejecutar un analizador sintáctico sobre el código del marco orientado a objetos y obtener la información que identifique a cada elemento de los tipos definidos en el marco.
Prioridad:	Alta
Actores:	Cliente
Casos de uso relacionados:	CU_02, CU_03
Precondición:	El código en lenguaje Java de un marco orientado a objetos libre de errores y sin advertencias de compilación.
Secuencia normal:	1. Selección del marco que deberá analizarse a partir del cual se crearán los microservicios. 2. Ejecución del analizador sintáctico sobre el código del marco seleccionado.
Escenario de fracaso:	1. Selección de marco que debe analizarse a partir del cual se crearán los Microservicios. 2. No se puede realizar el análisis sintáctico: informar al cliente y detener el proceso.
Post-condición:	Se contará con la información de las clases de objetos contenidos en el marco y de los elementos que incluyen éstas.
Observaciones:	Para proceder este caso de uso, el Marco Orientado a Objetos debe estar escrito en lenguaje Java. La herramienta no corrige errores de diseño dentro del Marco.

4.2.3 CU_02 Identificar el código para cada caso de uso

Esta paso consiste en descubrir cada una de las capacidades del código del MOO original e identificar los elementos que integrarán los casos de uso correspondientes. Este caso de uso depende del servicio solicitado. Cada caso de uso identificado atiende a una sola responsabilidad o meta de valor. Los elementos identificados para cada caso de uso y que corresponde a una capacidad son: definiciones de clase, constructores, inicializadores y miembros de cada clase (métodos, atributos y constantes).

Al final se generan las listas de clases con sus elementos relacionados para cada caso de uso identificado. La tabla 4 muestra la descripción del caso de uso.

Tabla 4 Caso de uso Identificar el código para cada caso de uso

CU_02	Identificar el código para cada caso de uso
Descripción:	Obtener cada uno de los elementos del código del MOO relacionados, que se usan para satisfacer cada caso de uso encontrado en el marco. Estos elementos son: definiciones de clase, constructores, inicializadores y miembros de cada clase (métodos, atributos y constantes).
Prioridad:	Alta
Actores:	Cliente
Casos de uso relacionados:	CU_01, CU_03
Precondición:	Contar con la información de cada clase resultante del análisis aplicado al MOO.
Secuencia normal:	1. Determinar los métodos iniciadores de cada caso de uso. 2. Identificar la secuencia interactiva de cada caso de uso y miembros de clase utilizados. 3. Identificar las relaciones de dependencia, agregación y composición. 4. Integrar el código complementario. 5. Identificar los tipos utilizados. 6. Depurar las listas de los casos de uso. 7. Integrar el código que satisface a un caso de uso en una única clase de objetos.
Alternativas:	No existen métodos iniciadores de casos de uso: se informa al usuario y no se generará ningún Microservicio.
Post-condición:	Se tendrá identificada la información de los elementos de código necesarios para satisfacer cada caso de uso encontrado en el marco.
Observaciones:	Para proceder este caso de uso, el Marco Orientado a Objetos debe estar escrito en lenguaje Java. La herramienta no corrige errores de diseño dentro del Marco.

4.2.4 CU_03 Generar Microservicios

Consiste en generar un microservicio por cada caso de uso identificado en el MOO a partir de las listas generadas en el caso de uso CU_02.

La herramienta SR2-Transforming (Sistema de Transformación para Reuso), implementa los métodos realizados en trabajos anteriores en el CENIDET: MOOaSWCU (León, 2009), Inlineclass (Legorreta, 2017) y MOOinMS (Gallardo, 2018). La herramienta de trasformación

recibe como entrada un MOO y obtiene como salida su correspondiente marco de Servicios Web o Microservicios. Hacerlo así mejora el reuso de componentes de software puesto que los enfoques de los métodos mencionados anteriormente dan como resultado ventajas en coherencia, cohesión, independencia, autosuficiencia. La diferencia entre los tres enfoques de los métodos de transformación mencionados radica en que los tres logran en mayor o menor medida un mejor balance entre las métricas mencionadas anteriormente y mejoran el rendimiento en cuanto al tiempo de respuesta de los servicios.

En este trabajo se realizó una extensión a la herramienta SR2-Trasnforming con el objetivo de obtener microservicios, ahora la herramienta tiene la flexibilidad de obtener microservicios desde cada uno de los enfoques del SR2-Transforming que cada método utiliza. Desde el inicio el cliente decide qué tipo de servicio y qué método desea invocar haciendo dinámica la generación de los servicios. En una versión anterior del SR2-Transforming, quien decidía que servicio se iba a obtener era el propio método y eso provocaba la repetición de código en cada uno de ellos.

Es importante mencionar que los microservicios generados son envueltos en una capa REST y adaptados para trabajar con el framework SpringBoot, esto es debido a que la herramienta SR2-Transforming y los servicios generados están escritos en lenguaje java y el framework antes mencionado tienes muchas ventajas para generar microservicios. La tabla 5 muestra el caso de uso para generar Microservicios.

Tabla 5 Caso de uso Generar Microservicios

CU_03	Generar Microservicios
Descripción:	Genera los microservicios que satisfacen cada caso de uso encontrado en el marco orientado a objetos de entrada.
Prioridad:	Alta
Actores:	Cliente
Casos de uso relacionados:	CU_01, CU_02
Precondición:	Contar con una estructura de datos con información de los elementos del código del marco que integrará cada microservicio.
Secuencia normal:	1.- Generar la clase que envuelve al microservicio (capa REST) con el que se puedan consumir los microservicios para cada caso de uso. 2.- Generar el archivo con la clase que hace ejecutable el microservicio. 3.- Generar el archivo *.java* que realiza la funcionalidad del MS.
Escenario de fracaso:	1. No es posible envolver el código de cada caso de uso como Microservicio: se informa al cliente y se detiene el proceso.

Capítulo 5: Diseño e Implementación del Método en la Herramienta SR2-Transforming

La implementación del método se realizó como una extensión a la herramienta de transformación *SR2-Trasnforming* de la cual ya se han mencionado sus capacidades en los capítulos anteriores. Los diagramas de secuencias que se realizaron con base en los casos de uso se describen a continuación.

5.1 Diagrama de secuencias

A continuación, se describen los principales diagramas de secuencia que se generaron para obtener los microservicios, en la figura 8 se muestra el diagrama con el flujo principal para obtener microservicios. Posteriormente, se explica cada uno de los diagramas generados a partir de este flujo.

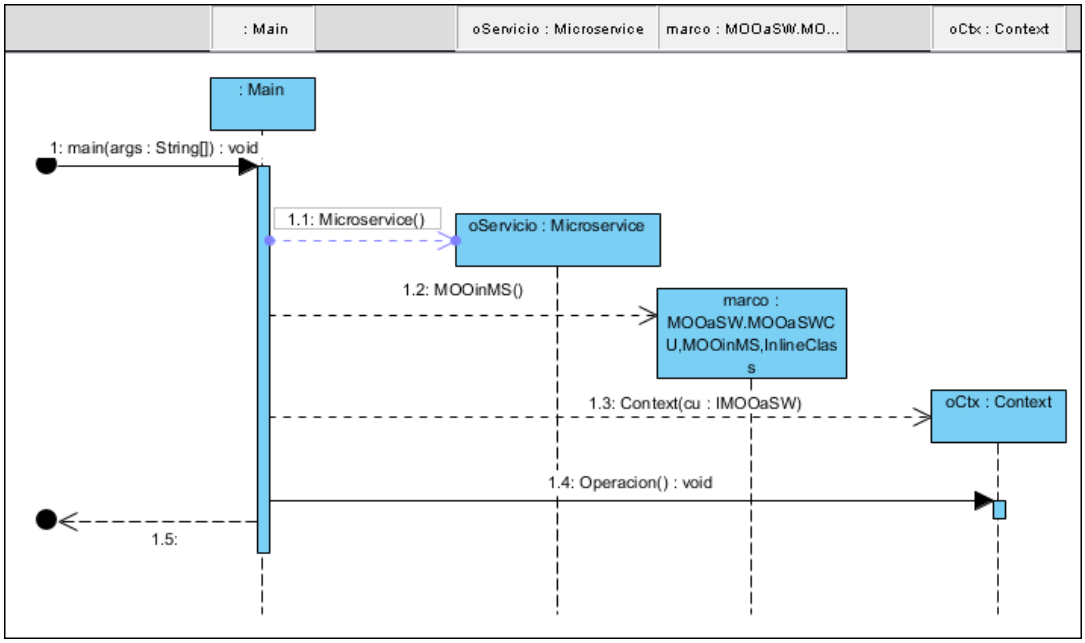

Figura 8 Diagrama de secuencias del flujo principal del método MSWeb2MMServ

En la función *main()* de la figura 8 se visualiza como el cliente lee el directorio donde se encuentra el MOO, y elije que tipo de servicio estratégico quiere utilizar para la generación de los servicios; esto es, un servicio web o un microservicio. También elije qué método estratégico quiere utilizar para la generación de las arquitecturas de los casos de uso; esto es, MOOaSW, MOOinMS o InlineClass. En seguida de esta elección, se analiza el marco, para identificar el código para cada caso de uso. Después de ésto se genera el microservicio correspondiente a cada Caso de Uso generado en el paso anterior.

Es importante mencionar que, antes, se realizó una refactorización a la herramienta SR2-Transforming en la cual se agregó una nueva clase Contexto que es la que permite decidir el tipo de servicio a generar y el método que se va a utilizar. Para esto, en la arquitectura SR2-Transforming se utilizó el patrón de diseño *Strategy* y se cambió la forma de generación de los microservicios. Antes de este cambio el proceso se realizaba por *coreografía* en donde la clase *MOOInMS* correspondiente al método jugaba el papel de coreógrafo y cada método decidía que servicio (servicio web o microservicio) se tenía que generar. Con la nueva arquitectura, el proceso se conduce combinando coreografía y *orquestación*, donde el servicio decide qué tipo de método estratégico desea generar y el llamado al servicio se realiza por coreografía.

5.1.1 Diagrama de secuencias del proceso principal

Siguiendo el flujo del diagrama anterior, el cliente invoca el método Operación() para generar el servicio cuyo tipo fue previamente seleccionado. Este método lee el directorio en el que radica el código del marco orientado a objetos que se transformará en un marco de microservicios, y realiza el análisis del marco para identificar el código de cada caso de uso. Después de esto se generan los microservicios. En la figura 9 se observa el diagrama de secuencias del método EjecutaProceso() para obtener los microservicios.

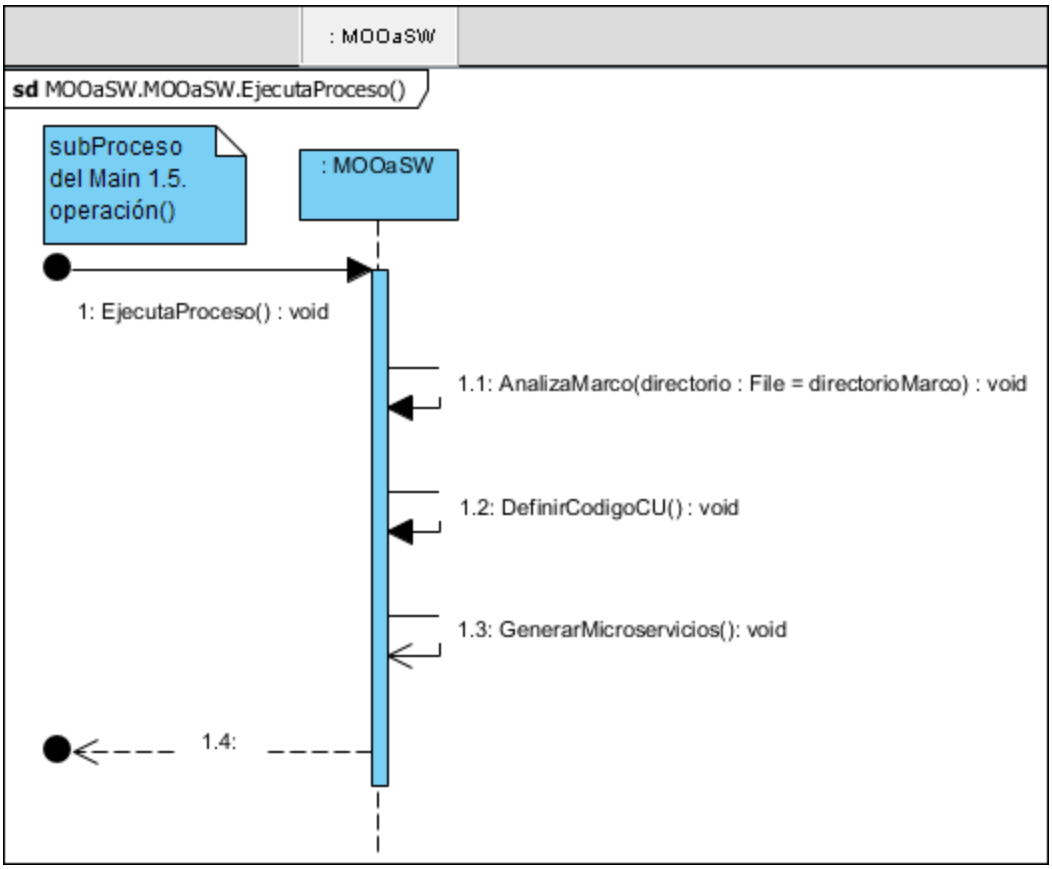

Figura 9 Diagrama de secuencias del flujo EjecutaProceso () para obtener microservicios

5.1.2 Diagrama de secuencias para analizar el MOO

Para el análisis del MOO se deben realizar dos pasos:

1.- La selección del MOO original, a partir del cual se generarán los microservicios
2.- El análisis sintáctico de este código de entrada.

Para el primero se debe revisar que el MOO esté escrito en lenguaje *java* y que todas las clases *.java* se encuentren contenidas dentro de un mismo directorio o en subdirectorios contenidos dentro del directorio principal. En el análisis cada archivo *.java* se utiliza un analizador sintáctico para identificar los Casos de Uso implícitos en el código original y se extrae la información necesaria para generación de código de cada uno de ellos. La información extraída se almacena en el diccionario de datos. En la figura 10 se observa el diagrama de secuencias para analizar el MOO, esta secuencia consta de un ciclo de dos mensajes: uno al *parser* con el fin de obtener una referencia a la raíz (objeto de tipo Compilation Unit) del AST que se forme de una unidad de compilación en particular (archivo.java); y un mensaje a la misma clase MOOaSWCU, cuyo fin sea obtener información de los miembros heredados en cada unidad de compilación. (León, 2009) (§ 4.4.1 Analizar el MOO). Es importante mencionar que en este caso se seleccionó el marco MOOInMS pero la herramienta tiene la implementación para poder elegir cualquiera de los tres métodos utilizados en los trabajos que anteceden a esta investigación.

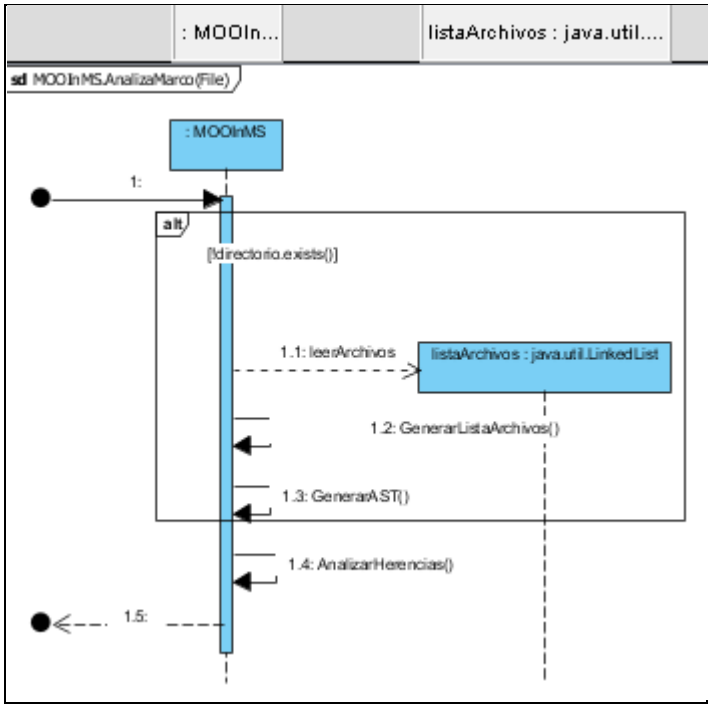

Figura 10 Diagrama de secuencias del flujo para leer el marco orientado a objetos

5.1.3 Diagrama de secuencia para definir el código para cada caso de uso

La identificación de Casos de Uso desde el código del MOO original se realiza mediante la identificación de las secuencias interactivas de métodos que satisfacen una capacidad o requerimiento del dominio de aplicaciones. Se parte desde los métodos, de clase, que son iniciadores de casos de uso. Estos son aquellos métodos públicos de clases públicas no abstractas del marco. Si la clase es abstracta no existen iniciadores de casos de uso, de lo contrario todos sus métodos públicos son iniciadores de casos de uso. Un método iniciador de caso de uso es considerado como el punto de entrada al servicio para satisfacer una meta de valor requerida por algún usuario. El detalle del diagrama de flujo se encuentra en el trabajo de tesis de (Gallardo 2018) en la sección (§ 5.2 Definir código para cada caso de uso) se explica el funcionamiento del diagrama de secuencia que se muestra en la figura 11.

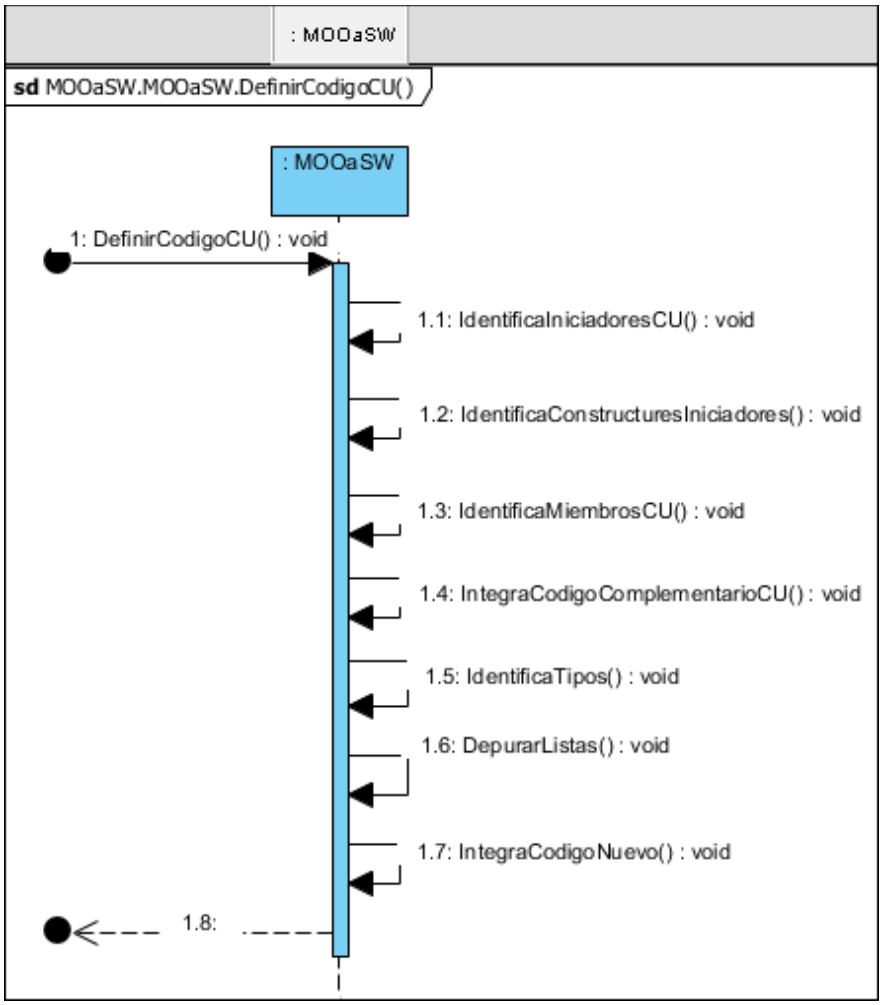

Figura 11 Diagrama de secuencias para definir el código para cada caso de uso

5.1.4 Diagrama de secuencias para generar los microservicios

A partir de haber identificado el código para cada caso de uso se integra el código correspondiente a cada uno de ellos, según el método de refactorización que se haya seleccionado. Se procede a generar un microservicio por cada caso de uso cuyo nombre será asignado con base en el identificador del método iniciador y el nombre calificado de su clase. Posteriormente, para generar los microservicios, se procede a envolver el código de cada Caso de Uso integrado en una capa REST.

La figura 12 muestra el diagrama de secuencias correspondiente a la generación de microservicios.

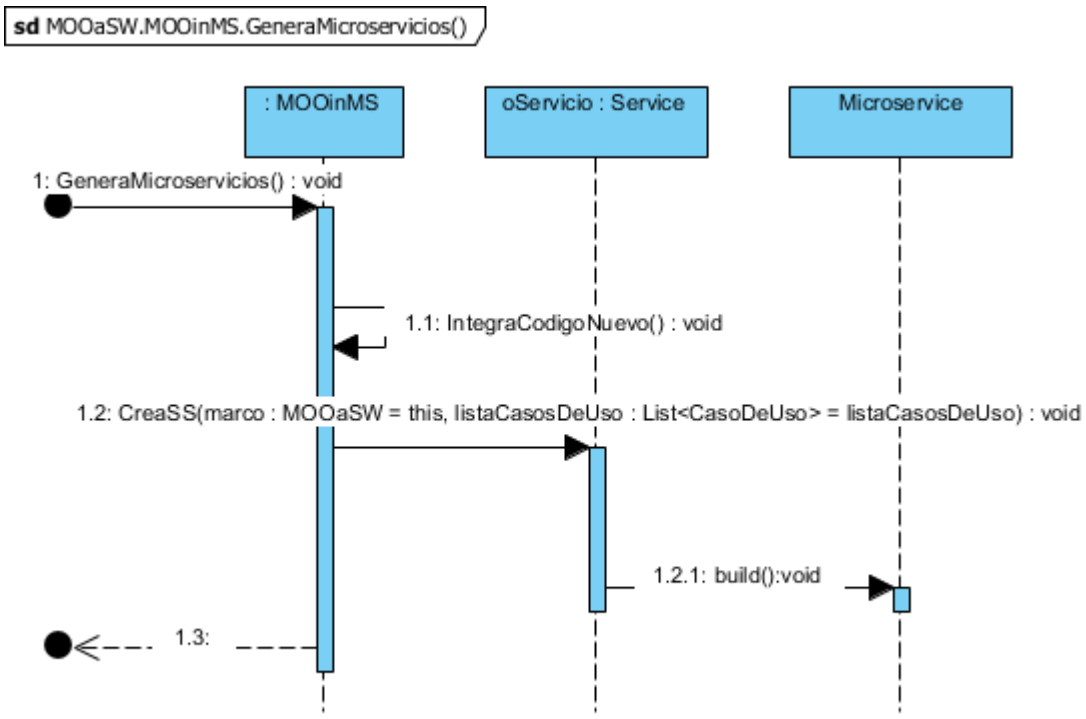

Figura 12 Diagrama de secuencias para generar microservicios

Finalmente, los microservicios generados están preparados para ejecutarse con la herramienta SpringBoot para el desarrollo de servicios REST. SpringBoot es una herramienta de Spring Framework creada por Pivotal y lanzada en abril de 2014 (GA), se encuentra dentro del estado del arte en el desarrollo de servicios. Fue desarrollada a pedido de SPR-9888 (https://jira.spring.io/ browse / SPR-9888) con el título *soporte mejorado para arquitecturas de aplicaciones web "sin contenedor"*, porque el entorno de nube actual o PaaS proporciona la mayoría de las características que ofrecen las arquitecturas web basadas en contenedores, como la confiabilidad, la administración o la escalabilidad. Por lo tanto, Spring Boot se centra

en convertirse en un contenedor ultraligero que funciona perfectamente con la arquitectura de microservicios.

El procedimiento inicia con la generación de un microservicio por cada caso de uso identificados en los pasos anteriores. Posteriormente, la herramienta crea tres archivos, el primero contiene la clase del controlador REST. El controlador maneja las solicitudes HTTP en la implementación del servicio web RESTful de Spring. El archivo generado está formado principalmente por las siguientes anotaciones:

@RestController es una anotación a nivel de clase utilizada para la clase de recursos presentada en Spring 4. Es una combinación de @Controller y @ResponseBody, y debido a ello, la clase devuelve un objeto de dominio en lugar de una vista.

La anotación **@RequestMapping** se usa a nivel de clase para mapear / calcular la URI a la clase controlador. Es decir, asegura que la solicitud HTTP sea asignada a la clase controladora. En función de la ruta definida y utilizando la anotación @RequestMapping del URI (postfijo de / cálculo. Por ejemplo, / nombre de la clase / función / parámetro), se hace relación con los métodos respectivos que se implementen.

Es importante mencionar que en esta primera aproximación para obtener microservicios sólo se usan las dos anotaciones explicadas anteriormente y que son necesarias para poder ejecutarlos. Hay otras anotaciones que pueden implementarse, pero podrían considerarse como trabajos futuros.

Además de las anotaciones, el archivo generado integra todo lo necesario para que la clase controlador funcione correctamente, como son los paquetes e importaciones a librerías propias de SpringBoot, así como la secuencia de invocaciones de clases y métodos del microservicio para atender una meta de valor.

Una parte interesante en el archivo generado es que debido al soporte del convertidor de mensajes HTTP de Spring, el objeto de cálculo se convierte, automáticamente, a JSON. No se necesita hacer esta conversión manualmente al momento de mandar llamar al microservicio.

La figura 13 muestra un ejemplo de generación del primer archivo por medio de la herramienta de transformación.

```
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/calculation")
public class CbRegistraUsuarioController {
    @RequestMapping("/registra")
    //public List<Calculation>  (@RequestParam(value = "bd") String b, @Req
    public String calcula(@RequestParam(value = "bd") String b,
            @RequestParam(value = "query")String q)throws SQLException {
            String bd=b;
            String query=q;

            ExecuteQueryCommand obj = new ExecuteQueryCommand(bd,query);
        return obj.execute(bd,query);
    }
}
```

Figura 13 Código de la clase controlador para el llamado al microservicio

El segundo archivo generado, se encarga de hacer el microservicio ejecutable y en él está integrado el código con la anotación SpringBootApplication. El método main () utiliza el método SpringApplication.run () de SpringBoot para iniciar la aplicación e implícitamente agrega las siguientes etiquetas:

- La anotación @Configuration etiqueta la clase como fuente de definiciones Bean para el contexto de la aplicación.

- La anotación @EnableAutoConfiguration indica que SpringBoot debe comenzar a agregar beans en función de la configuración de la ruta de clases y varias configuraciones de la propiedad.

- La anotación @EnableWebMvc se agrega si Spring Boot encuentra springwebmvc en classpath. Trata la aplicación como una aplicación web y activa los comportamientos clave, como la configuración de un DispatcherServlet.

- La anotación @ComponentScan le dice a Spring que busque otros componentes, configuraciones y servicios en el paquete dado.

La figura 14 muestra el ejemplo del código generado.

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
/**
 * @Omar H.L
 */
@SpringBootApplication
public class DistribucionX2App {
    public static void main(String[] args) {
        SpringApplication.run(DistribucionX2App.class, args);
    }
}
```

Figura 14 Código de la clase que hace al microservicio ejecutable.

El tercer archivo generado integra el código nuevo con base en las listas de las clases seleccionadas y los asigna en nuevos archivos de código Java y quedan almacenados en directorios específicos de cada caso de uso, la figura 15 muestra un ejemplo del código generado.

```java
public class ExecuteQueryCommand {
    private String query;
    BaseDatos receptor = this.new BaseDatos();
    Conexion con = this.new Conexion();

    public ExecuteQueryCommand(String bd,String query)
    {
      this.receptor = receptor;
        this.con = con;
        this.query = query;
    }

    public String execute(String bd,String query)
    {
        con.abrirConexion();
        receptor.executeQuery(con, query);
        con.cerrarConexion();
        return "operacion realizada con exito";}
```

Figura 15 Código de ejecución de tareas del microservicio

Capítulo 6: Diseño Experimental y Plan de Pruebas

6.1 Diseño experimental

Para efectos del presente experimento, fueron establecidas las siguientes hipótesis:

6.1.1 Hipótesis general

La capa con la cual están envueltos los Servicios Web o Microservicios, así como el protocolo de comunicación afectan su tiempo de respuesta.

6.1.2 Hipótesis específica

La capa en la cual están envueltos los servicios web y los microservicios, el protocolo de comunicación SOAP y HTTP bajo REST respectivamente, así como las variaciones en el throughput, influyen sobre su rendimiento afectando el tiempo de respuesta de cada uno de ellos.

6.1.3 Hipótesis estadísticas

- *Hipótesis nula (**H0**)*: Los tiempos de respuesta entre la arquitectura de Servicios Web con el protocolo SOAP generada por el método MOOInMS y la arquitectura de Microservicios con el protocolo HTTP basado en REST generada con el método MSWeb2MMServ son estadísticamente iguales.

- *Hipótesis alternativa (**H1**)*: El tiempo de respuesta de los Microservicios generados con el método MSWeb2MMServ, tienen estadísticamente, un mejor tiempo de respuesta que los servicios web obtenidos con el método MOOInMS.

6.1.4 Formulación del modelo experimental

Una variable experimental es un atributo que al ser medido en diferentes situaciones es susceptible a adoptar distintos valores. Para el presente estudio experimental, se han identificado 3 tipos de variables: las variables dependientes, las variables independientes y factores de ruido (Guadarrama Rogel, 2013).

6.1.5 Variables dependientes

Un variable dependiente es una variable que representa una cantidad cuyo valor depende de cómo se modifica en un experimento la variable independiente, es decir que pueden estar influenciados por las variaciones de entrada. En el presente estudio experimental se definió la siguiente variable dependiente:

- Tiempo de respuesta: el objetivo principal es concluir si el rendimiento de los servicios web y microservicios se ve afectado por su arquitectura interna y el protocolo de comunicación SOAP o HTTP basado en REST respectivamente.

6.1.6 Variables independientes

La variable independiente es el factor cambiable dentro del estudio y puede ser manejada o manipulada sistemáticamente, los cambios controlados de esta variable tienen un efecto directo en la variable dependiente. Para el presente estudio experimental se definieron las siguientes variables independientes:

- El protocolo de comunicación: las pruebas realizadas fueron realizadas con base en el método de transformación de Servicios Web (MOOInMS) y Microservicios (MSWeb2MMServ) cuyas arquitecturas son distintas en la capa que las envuelve y el protocolo de comunicación que los soportan SOAP y HTTP basado en REST respectivamente.

- Throughput: Es la carga de trabajo que asume un servicio para atender a una cantidad de peticiones del servicio simultaneas en una unidad de tiempo. Esta es la variable utilizada para estresar cada una de las pruebas experimentales. Es decir, es el número de veces que se solicita un servicio en un periodo corto de tiempo. Esta variable no depende directa ni indirectamente de otra variable externa.

6.1.7 Factores de ruido

- *Factores controlables*. Entre los factores controlables se encuentran la memoria RAM del equipo, el sistema operativo, el procesador, la administración de procesos en segundo plano, la configuración del servicio de internet, el acceso entre los equipos que acceden a la red local y la configuración de la misma.

- *Factores no controlables*. Algunos de los factores no controlables son la administración de memoria, el funcionamiento interno de la máquina virtual de Java, la forma en que el servidor internamente administra las peticiones, la energía eléctrica disponible entre otros.

6.2 Diseño del plan de pruebas

6.2.1 Convención de nombres

La convención de nombres de los identificadores de pruebas, será la misma que fue utilizada en el trabajo de tesis de (Saldívar, 2015) (§ 5, convención de nombres) como se muestra a continuación:

Tipo de Documento-Nombre del MOO-Identificador del documento, ejemplo: **PP-MSWeb2MMServ-01.**

Tipos de identificadores:

- PP- Plan de Pruebas
- DP- Especificación de diseño de pruebas
- CP- Especificación de casos de pruebas

El Plan de Pruebas (PP) para el estudio experimental se describe a continuación:

1. Se diseñaron los casos de prueba a partir de una muestra aleatoria simple de los Marcos Orientados a Objetos utilizados en el Plan de Pruebas en (Saldívar, 2015).

2. Se establecen las entradas y el valor del *throughput* que recibirán cada uno de los casos de prueba durante las pruebas experimentales.

3. Se ejecutan las pruebas experimentales mediante una herramienta computacional y se recolectan los resultados obtenidos.

4. Se realizan los análisis estadísticos para aceptar o rechazar las hipótesis experimentales y realizar conclusiones sobre el fenómeno de estudio.

6.3 Especificación DP-MSWeb2MMServ-01

6.3.1 Características que se probarán

Se prueba que el código de microservicios generado a través de la herramienta SR2-Transforming se genere de forma correcta. Esto incluye el archivo que envuelve al microservicio en una capa REST, el archivo que hace al microservicio ejecutable en la herramienta SpringBoot y el código generado a través del enfoque de clases internas del trabajo de tesis antecedente realizado por (Gallardo, 2018).

6.3.2 Refinamiento del enfoque

Los microservicios generados, estarán diseñados para funcionar con la herramienta SpringBoot del framework Spring. Esta herramienta tiene varias ventajas al implementar microservicios como son: soporta el patrón de diseño MVC, proporciona funciones para implementar la arquitectura REST, soporta la inyección de dependencia y es una herramienta que permite el desarrollo de microservicios de manera más rápida y fácil.

Las características básicas que deben tener los microservicios generados para funcionar con SpringBoot son las siguientes:

- Se debe generar de forma correcta un archivo que envuelva al microservicio con su correspondiente capa REST.
- Se debe generar correctamente un archivo de texto que contenga la clase *java* que hace ejecutable al microservicio.

Se debe generar de forma correcta el archivo mediante el enfoque de clases internas. Este enfoque obtiene la secuencia interactiva que atiende una meta de valor o única responsabilidad. El archivo contiene la funcionalidad del microservicio.

6.3.3 Criterio Pasa/No Pasa

Se compara y revisa el código generado con el sistema contra el código de entrada del marco siguiendo su secuencia interactiva. Si son iguales el sistema pasa la prueba de lo contrario no pasa la prueba. La comparación de los resultados se hará de forma manual.

6.4 Especificación DP-MSWeb2MMServ-02

6.4.1 Características que se probarán

Se prueba que los Microservicios generados realicen la misma funcionalidad que los Servicios Web correspondientes obtenidos en las pruebas de la tesis de (Gallardo, 2018) sobre el MOO *statistic*.

6.4.2 Refinamiento del enfoque

La herramienta SR2-Transforming soporta la generación de servicios web y microservicios mediante la selección específica de uno de tres enfoques del SR2-Transforming. Para esta investigación el método utilizado es el MOOInMS. En esta prueba se debe verificar que la funcionalidad de los servicios web contra los microservicios, ambos obtenidos del marco "statistic" sea la misma.

Para el caso de los servicios web se va a utilizar el IDE de desarrollo NetBeans en donde se harán pruebas unitarias para probar que los servicios funcionen de manera correcta. En el caso de los microservicios también se utilizará el IDE de desarrollo NetBeans, pero, para este caso, el proyecto en este IDE debe ser generado desde cero como proyecto SpringBoot. Posteriormente, se deben colocar los archivos con los microservicios generados de forma automática por la herramienta SR2Transforming en el lugar correspondiente. Por último, se deben ejecutar los microservicios para verificar que funcionen correctamente.

6.4.3. Criterio Pasa/No Pasa

Se compara el funcionamiento de los servicios web contra los microservicios correspondientes. Si el funcionamiento y resultados esperados por ambos servicios son los mismos se considera que la prueba pasa, de lo contrario la prueba no pasa.

6.5 Especificación DP-MSWeb2MMServ-03

6.5.1 Características que se probarán

Se prueba que los Microservicios generados realicen la misma funcionalidad que el Marco Orientado a Objetos "*PSPCenidet"*.

6.5.2 Refinamiento del enfoque

El MOO *PSPCenidet* es un sistema que mide los tiempos que una persona utiliza en realizar tareas específicas. A diferencia del marco estadístico "esta", este MOO utiliza una base de datos relacional para almacenar información.

En esta prueba se verificará que los microservicios generados con la herramienta SR2-Transforming utilizando como entrada el MOO "*PSPCenidet*", seleccionando el enfoque de

clases internas funcionen de forma correcta. Para probar la funcionalidad, se debe ejecutar el sistema original que implementa el MOO de entrada, probando al menos una tarea que atienda una responsabilidad contra su microservicio correspondiente.

Para probar el microservicio, éste debe ser desplegado bajo la herramienta SpringBoot y configurar lo necesario para verificar que el funcionamiento sea el mismo que el obtenido en el sistema "*PSPCenidet*".

6.5.3 Criterio Pasa/No Pasa

Se compara el funcionamiento de los servicios web contra los microservicios correspondientes obtenidos por el sistema. Si el funcionamiento y resultados esperados por ambos servicios son los mismos se considera que la prueba pasa, de lo contrario la prueba no pasa.

6.6 Especificación DP-MSWeb2MMServ-04

6.6.1 Características que se probarán

Específicamente se prueba el rendimiento en términos del tiempo de respuesta tanto de los microservicios como de los servicios web generados con la herramienta SR2-Trasnforming aplicando el enfoque de clases internas y utilizando el MOO "*Statistic"* utilizado en las pruebas de la tesis de (Gallardo, 2018).

6.6.2 Refinamiento del enfoque

El tiempo de respuesta se mide por el tiempo que tarda un servicio en responder a partir del momento en que se le envía una petición. En esta prueba se espera que el tiempo de respuesta al llamar un Microservicio sea menor que el tiempo que se tarda en responder un servicio web.

6.6.3 Criterio Pasa/No Pasa

Se compara el tiempo de respuesta de los Microservicios contra el tiempo de respuesta de los Servicios Web correspondientes. Si el tiempo de respuesta de los Microservicios es menor, el sistema pasa la prueba y confirma lo esperado, de lo contrario el sistema no pasa la prueba.

Capítulo 7: Ejecución de las Pruebas y del Experimento

7.1 Especificación de entrada

Se realizaron las pruebas de funcionalidad sobre el marco estadístico "*statistic*" utilizado en la tesis de (Gallardo, 2018) que cuenta con un total de 46 clases de objetos para realizar de forma correcta su funcionamiento.

Se realizaron pruebas de funcionalidad teniendo como entrada el MOO "*PSPCenidet"* sobre al menos una funcionalidad que atienda una meta de valor.

Se realizaron pruebas de rendimiento sobre el marco estadístico "*statistic*" utilizado en la tesis de (Gallardo, 2018) que cuenta con un total de 46 clases, y se compararon los Servicios Web generados contra los Microservicios ambos con el mismo MOO en igualdad de circunstancias.

La entrada es la ruta donde se encuentran ubicados los archivos "*.java"* correspondientes al Marco Orientado a Objetos de entrada. Todos éstos deben estar contenidos en una sola carpeta.

7.2 Especificación de los casos de prueba

7.2.1 CP- MSWeb2MMServ-01 : generación correcta de código de los microservicios, así como los archivos correspondientes para su despliegue.

Especificaciones de entrada:

Las entradas para las pruebas serán las siguientes:

1. La ruta del directorio en donde se encuentren los archivos *.java* que contienen el código de las muestras tomadas del Plan de Pruebas propuesto en (Saldívar, 2015) para el MOO *statistics*.

Especificaciones de salida:

Para todos los casos se generaron los 3 archivos con su código correspondiente a cada microservicios de forma correcta.

7.2.2 CP- MSWeb2MMServ-02: funcionalidad de los microservicios a través del método MOOInMS "statistic" contra el MOO estadístico original.

Para medir la funcionalidad de los microservicios se ejecutó cada uno de ellos por medio de la herramienta SpringBoot. Para esto es necesario configurar la herramienta en el IDE de desarrollo NetBeans, así como levantar un servidor (contenedor de servlets) y probar su funcionamiento usando la extensión de Chrome "Postman".

La extensión de "Chrome Postman", permite realizar pruebas de servicios usando la API de REST, con todos sus métodos y muestra un reporte con los resultados obtenidos al llamar a cada servicio.

Para el caso de los Servicios Web se utilizó la herramienta de NetBeans que ya viene configurada para poder probar los servicios generados y se generó el archivo "*war*" correspondiente para poder probarlo desde el servidor de aplicaciones, la tabla 6 muestra el resultado de las pruebas.

Tabla 6 Resultados de funcionalidad de los microservicios

Caso de prueba	Nombre del servicio Web	Resultado de la ejecución con el MOO-Original	Resultado de la ejecución con el MSWeb2MMServ
P01	cBubleXY	[1,2,3,4,5] [1,2,3,4,5]	[1,2,3,4,5] [1,2,3,4,5]
P02	cBuble	[1,2,3,4,5]	[1,2,3,4,5]
P03	cDistributionX2	6.011205773063438E-4	6.011205773063438E-4
P04	cDistributionT	0.0	0.0
P05	cDistribution	0	0
P06	cDistItems	0.25335693359375	0.25335693359375
P07	cCorrelation	0	0
P08	cCalt	0	0
P09	List1	[1,2,3,4,5]	[1,2,3,4,5]
P10	List	[1,2,3,4,5] [1,2,3,4,5]	[1,2,3,4,5] [1,2,3,4,5]
P11	CalculaCorrelacion_context_ctx	-2.8867513459481287	-2.8867513459481287
P12	cSumXY	0.117652	0. 117652
P13	cRange	No arrojo ningún resultado	No arrojó ningún resultado

7.2.3 CP- MSWeb2MMServ-03: funcionalidad de los Microservicios generados con la herramienta SR2-Transforming contra la funcionalidad del MOO de entrada "*PSPCenidet*".

Para medir la funcionalidad de los microservicios generados en esta investigación, contra la funcionalidad del "*MOO PSPCenidet"*, se probó el sistema *"PSPCenidet"* realizando una tarea específica en al menos un caso de uso que atienda una meta de valor, el resultado se comparó contra el microservicio generado por la herramienta MSWeb2MMServ para verificar que se realice la misma acción y que el resultado sea el mismo. La tabla 7 muestra el resultado de las pruebas.

Tabla 7 Funcionalidad de los Microservicios contra el Marco de entrada original *PSPCenidet*.

Caso de prueba	Nombre del servicio Web	MOO-*PSPCenidet*	MSWeb2MMServ
P01	Obtener perfil - CbPerfilCommandController	Prueba exitosa	Prueba exitosa
P02	Registrar Usuario – CbRegistraUsuarioController	Prueba exitosa	Prueba exitosa

7.2.4 CP- MSWeb2MMServ-04: Comparativa entre el rendimiento de los Microservicios obtenidos en esta investigación contra el rendimiento obtenido con los servicios web obtenidos en las pruebas de la tesis de (Gallardo, 2018) sobre el MOO statistic.

Se obtuvo el rendimiento de los microservicios en base al tiempo de respuesta de un conjunto de Servicios Web y Microservicios, a través de una muestra aleatoria simple para una población finita con un nivel de confianza del 90% y un error permitido del 10%. La muestra obtenida para los casos de prueba es de 13 servicios (Myers, 2012) descartándose aquellos cuya funcionalidad era muy atómica o no arrojaba un resultado de valor.

Se estableció el número de veces que se debe ejecutar la prueba, a partir de la muestra aleatoria simple para una población de infinita con un nivel de confianza del 95 % y un error permitido del 5%; dando como resultado una muestra de 56 ejecuciones con un throughput variable.

7.3 Instrumentos de medición aplicados

7.3.1 Instrumento de medición para el rendimiento con base en el tiempo de respuesta

Se obtuvieron los resultados del tiempo de respuesta de los servicios web y microservicios utilizando la herramienta computacional de código abierto JMeter, la cual es un proyecto de Apache que se utiliza como una herramienta para probar, analizar y medir el desempeño de servicios con un enfoque de sistemas web. JMeter soporta tanto los protocolos SOAP como HTTP-REST por lo que es una herramienta que permite realizar las pruebas de ambas arquitecturas. Para utilizar la herramienta es necesario contar con los elementos descritos en la tabla 8.

Tabla 8 Elementos necesarios para poder realizar las pruebas con la herramienta JMeter

Nombre	Descripción
Cliente	Equipo y aplicación de software a partir del cual se realiza el llamado a los servicios.
Servidor Web	Se encarga de procesar las peticiones enviadas por el cliente.
Conexión a internet	Enlace de la comunicación entre el cliente y el servidor.

La especificación de capacidades de los equipos utilizados es la misma para el cliente y servidor, la tabla 9 muestra la especificación del equipo:

Tabla 9 Especificaciones técnicas del equipo donde está instalado el cliente y servidor

Cliente y Servidor Web	• PC Portátil Dell Inspiron • Procesador Intel® Core™ I5 CPU M430 2.27GHz • Memoria Instalada RAM de 8 GB • Tipo de sistema de 64 bits

7.3.2 Instrumento para el análisis estadístico.

Al realizar el análisis estadístico es necesario interpretar los resultados obtenidos. Una herramienta práctica y reconocida por su precisión al comparar datos es la herramienta computacional de *IBM **SPSS Statistics*** (IBM-Analitycs, 2017)**.**

7.3.3 Eliminación del ruido experimental.

El ruido experimental son variables que no se pueden controlar durante la ejecución de un proceso y puede afectar el resultado de la prueba, Para realizar las pruebas se redujo el ruido utilizando el proceso propuesto por (Guadarrama Rogel, 2013).

Capítulo 8: Análisis Estadístico e Interpretación de Resultados

8.1 Pruebas de normalidad

Los métodos estadísticos se diseñan para contribuir al proceso de realizar juicios científicos frente a la incertidumbre y a la variación. Se han originado un gran número de métodos analíticos que permiten efectuar análisis de datos obtenidos de sistemas, lo cual refleja la verdadera naturaleza de la estadística inferencial. El uso de técnicas que permiten obtener conclusiones o inferencias sobre el sistema científico, permiten ir más allá de sólo reportar datos (Myers, 2012).

Es necesario realizar un análisis estadístico para poder interpretar los datos y así realizar conclusiones inferenciales. Para verificar la normalidad de los datos obtenidos de las poblaciones asociados al caso de prueba CP- MSWeb2MMServ-04, se utilizó la prueba Kolmogorov-Smirnof con un intervalo de confianza del 95%, la prueba fue realizada con la herramienta *SPSS statistics*. La prueba de normalidad permitió concluir que los valores dentro de la muestra **no** siguen una distribución normal, es decir, son datos **no** paramétricos.

8.1.1 Análisis estadístico

Una vez realizada la prueba de normalidad, por la cual se determinó que los datos son *no paramétricos*, se utilizó la prueba U de Mann-Whitney(Wallace, 2004), la cual es una técnica estadística no paramétrica. Se usa para analizar las diferencias entre las medianas de dos conjuntos de datos. Se puede utilizar en lugar de una prueba *t* para muestras independientes en los casos en que los valores dentro de la muestra no siguen la distribución normal.

La prueba de U de Mann-Whitney también se basa en calcular los rangos de la muestra de cada población y comparar los promedios de los rangos.

La tabla 10 muestra los resultados obtenidos donde *R* significa el rango promedio obtenido de cada método aplicado a la muestra de cada servicio.

Tabla 10 Resultados de la prueba paramétrica U de Mann Whitney

Método		Protocolo	Ejecuciones variando el trouhgput	R-Rango prome-dio	Suma de rangos
1_cBubleXY_statistic	MOOInMS	SOAP	56	69.25	3878.00
	MSWeb2MMServ	HTTP	56	43.75	2450.00
	Total		112		
2_cBuble_statistic	MOOInMS	SOAP	56	63.85	3575.50
	MSWeb2MMServ	HTTP	56	49.15	2752.50
	Total		112		
3_cDistributionX2_statistic	MOOInMS	SOAP	56	57.99	3247.50
	MSWeb2MMServ	HTTP	56	55.01	3080.50
	Total		112		
4_cDistributionT_statistic	MOOInMS	SOAP	56	61.81	3461.50
	MSWeb2MMServ	HTTP	56	51.19	2866.50
	Total		112		
5_cDistribution_statistic	MOOInMS	SOAP	56	63.89	3578.00
	MSWeb2MMServ	HTTP	56	49.11	2750.00
	Total		112		
6_cDistItems_statistic	MOOInMS	SOAP	56	62.66	3509.00
	MSWeb2MMServ	HTTP	56	50.34	2819.00
	Total		112		
7_cCorrelation_statistic	MOOInMS	SOAP	56	58.74	3289.50
	MSWeb2MMServ	HTTP	56	54.26	3038.50
	Total		112		
8_cCalt_statistic	MOOInMS	SOAP	56	59.68	3342.00
	MSWeb2MMServ	HTTP	56	53.32	2986.00
	Total		112		
9_ordenacion1_List_statistic	MOOInMS	SOAP	56	58.45	3273.00
	MSWeb2MMServ	HTTP	56	54.55	3055.00
	Total		112		
10ordenacion_List_statistic	MOOInMS	SOAP	56	57.86	3240.00
	MSWeb2MMServ	HTTP	30	55.14	3088.00
	Total		112		
11CalculaCorrelacion_context_ctx _statistic	MOOInMS	SOAP	56	60.04	3362.00
	MSWeb2MMServ	HTTP	56	52.96	2966.00
	Total		112		

12_cSumXY_statistic	MOOInMS	SOAP	56	61.79	3460.50
	MSWeb2MMServ	HTTP	56	51.21	2867.50
	Total		112		
13_cRange_statistic	MOOInMS	SOAP	56	56.89	3186.00
	MSWeb2MMServ	HTTP	56	56.11	3142.00
	Total		112		

El rango está medido en milisegundos

Los resultados de la prueba U de Mann Whitney, presentados en la tabla 10, muestran que el 100% de los casos difieren estadísticamente. Por esto se rechaza la hipótesis general y se acepta la hipótesis alternativa H1 que enuncia "*El tiempo de respuesta de los servicios obtenidos con el método MSWeb2MMServ bajo el protocolo HTTP, estadísticamente tienen un mejor tiempo de respuesta sobre los servicios obtenidos con el método MOOInMS a través del protocolo SOAP*".

8.2 Comparación de tiempos de respuesta de los casos de prueba

La tabla 11, muestra los resultados de comparar los dos métodos, MOOInMS que genera Servicios Web y MSWeb2MMServ que genera microservicios, en ella se puede observar la comparativa a partir del promedio de los tiempos de respuesta, obtenidos de las pruebas experimentales. Se considera que la arquitectura más eficiente es aquella que obtuvo un menor tiempo promedio de respuesta.

Tabla 11 Comparativa entre los tiempos de respuesta

Caso de prueba	MOOInMS	MSWeb2MMServ	Mejor tiempo de respuesta
P01	69.25	43.75	MSWeb2MMServ
P02	63.85	49.15	MSWeb2MMServ
P03	57.99	55.01	MSWeb2MMServ
P04	61.81	51.19	MSWeb2MMServ
P05	63.89	49.11	MSWeb2MMServ
P06	62.66	50.34	MSWeb2MMServ
P07	58.74	54.26	MSWeb2MMServ
P08	59.68	53.32	MSWeb2MMServ
P09	58.45	54.55	MSWeb2MMServ
P10	57.86	55.14	MSWeb2MMServ
P11	60.04	52.96	MSWeb2MMServ
P12	61.79	51.21	MSWeb2MMServ
P13	56.89	56.11	MSWeb2MMServ

**los valores de cada prueba están dados en milisegundos.*

Capítulo 9: Conclusiones

9.1 Conclusiones generales

Mediante el estudio experimental y al comparar los resultados de las pruebas, se concluye que, en todos los casos, aunque es mínima la diferencia en escala de milisegundos, el menor tiempo de respuesta fue el obtenido con el método MSWeb2MMServ, es de tomar en cuenta que el tiempo del procesador en escala de milisegundos es de magnitud considerable. Por lo tanto, se deriva que el 100% de los microservicios obtenidos con base en su arquitectura y capa que lo envuelve, obtienen el mejor rendimiento.

Hipotéticamente, la ventaja que se originó en los resultados puede ser debida por un lado a que el protocolo de comunicación HTTP bajo la arquitectura REST es más ligero que el protocolo SOAP de la arquitectura SOA. Por otro lado, esta ventaja también puede ser debida a que los procesos integrados con microservicios obedecen a una coordinación coreografiada con respecto a los mecanismos orquestados.

Los resultados experimentales muestran que el tiempo de respuesta fue muy semejante al probar ambos tipos de servicios. Sin embargo, en todos los casos la arquitectura de microservicios dio los mejores resultados de tiempos de respuesta. Adicionalmente, se concluye que mediante un correcto análisis previo de los sistemas legados y MOO así como la identificación correcta de servicios pueden ser usados de manera independiente, por lo tanto, la migración hacia microservicios es la mejor opción si lo que se busca es mejorar las características de: independencia, aislamiento, autonomía, resiliencia, tiempo de entrega y rendimiento para favorecer el reuso de los componentes.

En cuanto al funcionamiento de los servicios web contra los microservicios, en el 100% de los casos se obtuvo el mismo resultado por lo que se concluye que el método funciona de forma correcta.

Para comprobar el funcionamiento del MOO *PSPCenidet* en al menos una capacidad de ese sistema, siendo éste un caso especial ya que es la primera prueba que se realizó con acceso a una base de datos, las pruebas realizadas fueron satisfactorias en un 100%. Para realizar las pruebas de este caso es importante mencionar que se realizó la construcción de la interfaz de usuario de una aplicación cliente, ya que los marcos originales no cuentan con estas interfaces.

También se llegó a la conclusión de que es necesario un pre-procesamiento en los MOO de entrada con el objetivo de obtener microservicios formados correctamente y que den un resultado de valor.

9.2 Problemas presentados

Al ejecutar las pruebas dando como entrada el sistema *PSPCenidet*, se encontró que algunas secuencias interactivas se cortaban inadecuadamente, entre otras cosas el problema radica en que cuando se invocan métodos abstractos, al no contar éstos con código que los implementen suspenden su secuencia interactiva. Esto es un defecto que el método propuesto tiene y que se sugiere corregir en el trabajo futuro (§9.3.3). Este problema se presentó en los tres enfoques de antecedentes y del cual se parte para el desarrollo de esta investigación.

Para las pruebas realizadas en el MOO *PSPCenidet*, fue necesario realizar una revisión del marco de entrada y, en algunos casos, corregir los errores en su arquitectura que se presentaron de inicio, con el objetivo de generar de forma correcta los servicios.

En el caso de las pruebas del *PSPCenidet* el cual accede a una base de datos, los métodos *get* y *set* no están contemplados al generar los servicios web y microservicios, por lo que fue necesario corregir el problema agregando los métodos antes mencionados en cada microservicio generado.

Debido a que cada servicio se origina desde los métodos calificados con el modificador de alcance *"public"*. Al realizar las pruebas en los MOO's de entrada, como resultado se obtuvieron varios servicios cuyas metas de valor era realizar una operación funcional muy atómica, tales como el cálculo de una raíz cuadrada, la sumatoria de una serie de números, etc., lo cual contradice el enfoque del método de transformación desarrollado, cuyo objetivo es obtener servicios en el nivel que atiendan a requerimientos o capacidades del negocio o dominio de aplicaciones. Por ejemplo, en el código del MOO *PSPCenidet*, todas las funciones fueron calificadas con el modificador de alcance *"public"*, obteniendo como resultado varios servicios cuya meta de valor era una única operación funcional atómica. De aquí se plantea que es necesario un pre-procesamiento en los MOO´s de entrada para asegurar que solamente los métodos iniciadores de secuencias interactivas de metodos que correspondan a requerimientos o capacidades del MOO del dominio, deban ser públicos. El resto de métodos que participan en las secuencias interactivas de cada caso de uso deben ser calificadas como

private o *friendly* y ubicarse en el mismo paquete. Este defecto se pretende corregir en el trabajo futuro (§ 9.3.2).

En algunos de los casos de prueba su arquitectura obedece al patrón de diseño *strategy*. La estructura genérica de este patrón incluye una clase que contiene el *contexto* de la aplicación en particular. El problema que se presentó en las implementaciones de los tres trabajos de antecedente, radica en que la clase que enmarca el *contexto* fue considerada como parte de los microservicios. Esta consideración es incorrecta puesto que la clase contexto debe ubicarse del lado del cliente ya que es inherente a cada aplicación que utilice los microservicios.

9.3 Trabajos futuros

9.3.1. Debido a que este trabajo es exploratorio e inicio para futuras investigaciones y tomando en cuenta que sólo se llegó hasta la descomposición y despliegue de microservicios en servidores locales, se propone realizar trabajos enfocados a la gestión de microservicios. Por ejemplo, sistema automatizados que incluyan: el registro y descubrimiento de servicios, balanceo de carga, configuración del servidor perimetral y de seguridad. Además del despliegue de microservicios en contenedores o máquinas virtuales.

9.3.2. Se propone realizar un pre-procesamiento en los MOO de entrada para asegurar que solamente los métodos iniciadores de secuencias interactivas que correspondan a metas de valor o capacidades del sistema, deben de ser calificadas con el modificador de alcance *"public"*. El resto de métodos que participan en las secuencias interactivas de cada caso de uso deben ser calificadas como *private* o *friendly* y ubicarse en el mismo paquete.

9.3.3 Queda pendiente la depuración del analizador sintáctico para localizar las llamadas a funciones virtuales o abstractas en interfaces y/o clases abstractas, con el objetivo de corregir el defecto que se tiene en la versión actual de romper una secuencia interactiva.

9.3.4 Se propone realizar pruebas y experimentos sobre Marcos Orientados a Objetos de dominios de aplicaciones reales.

Referencias Bibliográficas

Bianchini, D., Cappiello, C., De Antonellis, V., & Pernici, B. (2014). Service identification in interorganizational process design. *IEEE Transactions on Services Computing*, *7*(2), 265–278. https://doi.org/10.1109/TSC.2013.26

Bonér, J. (2017). *Reactive Microsystems*. (O'REILLY, Ed.) (2017th-8th–7th ed.). United States of America: Lightbend.

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004). Web Services Architecture. *W3C Note*. https://doi.org/10.1023/B:BTTJ.0000015492.03732.a6

Brito, F., Iseg, I., Goulão, M., Esteves, R., & Ist, I. (1995). Toward the Design Quality Evaluation of Object-Oriented Software Systems. *5th International Conference on Software Quality*, (October).

Brooks, F. (1975). *The mythical man-month*. *Proceedings of the international conference on Reliable software*. https://doi.org/http://doi.acm.org/10.1145/800027.808439

Castillo, P. A., Bernier, J. L., Arenas, M. G., Merelo, J. J., & Garcia-Sanchez, P. (2011). SOAP vs REST: Comparing a master-slave GA implementation. Retrieved from http://arxiv.org/abs/1105.4978

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow. In *Present and Ulterior Software Engineering*. https://doi.org/10.1007/978-3-319-67425-4_12

Eisele, M. (2016). *Developing Reactive Microservices*. https://doi.org/978-1-491-96236-7

Erdemir, U., & Buzluca, F. (2014). A learning-based module extraction method for object-oriented systems. *Journal of Systems and Software*, *97*, 156–177. https://doi.org/10.1016/j.jss.2014.07.038

Fowler, M. (2002). Refactoring. *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, *IEEE Confe*, 13472. https://doi.org/10.1145/581441.581453

Fowler, M. (2006). Continuous Integration. *ThoughtWorks*. https://doi.org/10.1007/978-1-4302-0142-7_9

Gallardo, S. (2018). *Generación de Servicios Web desde Marcos Orientados a Objetos con el Enfoque de Clases Internas*. Tesis de Maestría - Centro Nacional de Investigación y Desarrollo Tecnológico.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2002). *Design Patterns – Elements of Reusable Object-Oriented Software*. *A New Perspective on Object-Oriented Design*. https://doi.org/10.1093/carcin/bgs084

Guadarrama Rogel, L. C. (2013). *Estudio Experimental para determinar la relación entre la estructura interna de Servicios Web y valores de QoS*. CENIDET.

Harrison, F. (2011). Getting started with Microservices. *Methods in Ecology and Evolution*, *2*(1), 1–10. https://doi.org/10.1111/j.2041-210X.2010.00056.x

IBM-Analitycs. (2017). IBM SPSS Software.

Josuttis., N. M. (2007). *SOA in Practice, The Art of Distributed System Design*. O'Reilly Media.

Kalepu, S., Krishnaswamy, S., & Loke, S. W. (2004). Verity: A QoS metric for selecting Web

services and providers. In *Proceedings - 4th International Conference on Web Information Systems Engineering Workshops, WISEW 2003: 3rd International Workshop on Web and Wireless Geographical Information Systems, W2GIS 2003, 1st Web Services Quality Workshop, WQW 2003 and 1st Works*. https://doi.org/10.1109/WISEW.2003.1286795

Kecskemeti, G., Marosi, A. C., & Kertesz, A. (2016). The ENTICE approach to decompose monolithic services into microservices. In *2016 International Conference on High Performance Computing and Simulation, HPCS 2016* (pp. 591–596). https://doi.org/10.1109/HPCSim.2016.7568389

Leader, R. P. (2010). *Rest in practice*.

Legorreta, N. (2017). *Transformación de Marcos de Aplicaciones Orientados a Objetos hacia Marcos con Arquitectura Orientada a Servicios*. Tesis de maestría, Centro Nacional de Investigación y Desarrollo Tecnológico.

León, P. F. (2009). *Generación de Servicios Web desde Marcos Orientados a Objetos a partir de Clases Colaborativas en Casos de Uso (MOOaSW)*. Tesis de maestría, Centro Nacional de Investigación y Desarrollo Tecnológico.

Levcovitz, A., Terra, R., & Tulio Valente, M. (2016). Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems. *3rd Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM)*, (October), 97–104.

Lewis, J., & Fowler, M. (2014). Microservices. Retrieved from http://martinfowler.com/articles/microservices.html

Liu, W., Fu, M., & Luo, S. (2011). Use Case-Based Service-Oriented Analysis and Modeling. *2011 International Conference on Internet Computing and Information Services*, 94–96. https://doi.org/10.1109/ICICIS.2011.30

MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., & Metz, R. (2006). *Reference Model for Service Oriented Architecture 1.0*. *OASIS Standards Committee Specification 1*.

Martin, R. C. (2002). *AGILE SOFTWARE DEVELOPMENT: PRINCIPLES, PATTERNS, AND PRACTICES*. *Nursing management*. https://doi.org/10.1002/pfi

Mattsson, M., Bosch, J., & Ronneby, S.-. (1997). Framework Composition : Problems , Causes and Solutions Department of Computer Science and Business Administration. *Technology of Object-Oriented Languages and Systems, 1997. TOOLS 23. Proceedings*. https://doi.org/10.1109/TOOLS.1997.654724

Mazlami, G., Cito, J., & Leitner, P. (2017). Extraction of Microservices from Monolithic Software Architectures. In *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*. https://doi.org/10.1109/ICWS.2017.61

Myers, R. (2012). *Probabilidad y estadística para ingeniería y ciencias*. *Journal of Chemical Information and Modeling* (Vol. 53). https://doi.org/10.1017/CBO9781107415324.004

Newman, S. (2015). *Building Microservices*. *O'Reilly*. O'Reilly Media. https://doi.org/10.1109/MS.2016.64

Oasis. (2006). Reference Model for Service Oriented Architecture. *Public Review Draft 2*.

Papazoglou. (2007). Service oriented architectures: Approaches, technologies and research issues. *VLDB Journal*. https://doi.org/10.1007/s00778-007-0044-3

Peltz, C. (2003). Web Services Orchestration and Choreography.

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM*

SIGSOFT Software Engineering Notes. https://doi.org/10.1145/141874.141884

Potti, P. K. (2012). Comparing Performance of Web Service Interaction Styles : SOAP vs . REST, 1–24.

Richards, M. (2015). *Microservices vs. Service-Oriented Architecture*. https://doi.org/10.1017/CBO9781107415324.004

Rostami, N. H., & Kheirkhah. (2013). W EB S ERVICES C OMPOSITION M ETHODS AND T ECHNIQUES : A R EVIEW, *3*(6).

Safina, L. (2016). Data-driven workflows for microservices: Genericity in jolie. In *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*. https://doi.org/10.1109/AINA.2016.95

Saldívar, M. (2015). *Plan de Pruebas para la Evalucación Experimental del Sistema*. Tesis de maestría, Centro Nacional de Investigación y Desarrollo Tecnológico.

Santaolaya, R., Fragoso, O. G., Rojas, J. C., Álvarez, F. J., & León, F. (2016). *Method for Generating Web Services Frameworks from Object Oriented Frameworks Source Code*. *hoja de trabajo*.

Serrano, N., Hernantes, J., & Gallardo, G. (2014). Service-Oriented Architecture and Legacy Systems, (October), 15–19.

Shadija, D., & Rezai. (2017). Towards an Understanding of Microservices, (September), 7–8.

Shahir Daya, Van Duy, K. (2015). Microservices from Theory to Practice Creating Applications in IBM Bluemix Using the Microservices Approach. *IBM*, 170.

Shahir Daya , Valerie Lampkin, S. N. (2015). Microservices from Theory to Practice Creating Applications in IBM Bluemix Using the Microservices Approach. *Ibm*.

Szyperski, C., Bosch, J., & Weck, W. (1999). Component-oriented programming. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/3-540-46589-8_10

Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., & Gil, S. (2015). Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud Evaluando el Patrón de Arquitectura Monolítica y de Micro Servicios Para Desplegar Aplicaciones en la Nube. *10th Computing Colombian Conference*, 583–590. https://doi.org/10.1109/ColumbianCC.2015.7333476

Vinay, G., & Jain, A. (2012). Creating Web Services from Legacy Code. *International Journal of Computer Applications*, *1*(1), 21–23.

Wallace, D. (2004). The Mann-Whitney Test. *Journal of the American Society for Information …*, 1–5. Retrieved from http://onlinelibrary.wiley.com/doi/10.1002/asi.10347/abstract

Xiao, Z., & Qiang. (2016). Reflections on SOA and Microservices. https://doi.org/10.1109/ES.2016.14

Yousef, R., Adwan, O., & Abushariah, M. A. M. (2014). Extracting SOA Candidate Software Services from an Organization's Object Oriented Models. *JSEA - Journal of Software Engineering and Applications*, *7*(August), 770–778. https://doi.org/10.4236/jsea.2014.79071