SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO
Secretaría Académica, de Investigación e Innovación
Dirección de Posgrado, Investigación e Innovación

cenidet
Centro Nacional de Investigación y Desarrollo Tecnológico

Centro Nacional de Investigación y Desarrollo Tecnológico

Subdirección Académica

Departamento de Ciencias Computacionales

TESIS DE MAESTRÍA EN CIENCIAS

Transformación de Marcos de Aplicaciones Orientados a Objetos hacia Marcos con Arquitectura Orientada a Servicios

presentado por
Ing. Nandi Eliseo Legorreta Ruiz

como requisito para la obtención del grado de
Maestro en Ciencias de la Computación

Director de tesis
Dr. René Santaolaya Salgado

Cuernavaca, Morelos, México. Enero de 2017.

Dedicatoria

Con todo cariño y el amor del mundo a mi madre, quién sin ella no podría lograr una meta más en la vida.

Agradecimientos

A Dios por permitirme llegar hasta este día, porque a pesar de las dificultades que te pone la vida ha estado cada paso conmigo dándome la fortaleza para seguir adelante.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico que me permitió el desarrollo de esta tesis.

Al Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET) por la oportunidad de realizar una Maestría en Ciencias de la Computación. Al personal administrativo y académico, por su atención y amabilidad, quienes con su trabajo diario permiten el desarrollo de nuestras actividades.

A mi madre quien me estuvo brindando con su apoyo moral, su cariño y sus sabios consejos. Por hacer de mí una mejor persona.

A mi director de tesis, Dr. René Santaolaya Salgado por darme seguimiento y supervisión continúa en el desarrollo de esta investigación, por su paciencia y valiosa confianza depositada en mí, pero sobre todo por su motivación y el apoyo que me brindo día a día durante estos años.

A mis revisores, Dra. Olivia Graciela Fragoso Díaz, y al Dr. Moisés González García por el tiempo que le dedicaron a mi trabajo de investigación, por sus críticas constructivas y comentarios que hicieron madurar este trabajo.

A mis profesores en general, por los consejos que me llegaron a dar y brindaron su confianza y enseñanza profesional.

A mis amigos Candía Guadalupe que siempre estuvo conmigo desde un principio hasta el final apoyándome siempre en todo. Gracias por todo.

A mis compañeros y amigos: Guadalupe, Laura, Isaí, Roberto, Irving, Misael, Catalina por compartir conmigo momentos divertidos y por el apoyo que me brindo cada uno de ellos.

A toda mi familia en general. Les doy las gracias por su apoyo, cariño y comprensión.

Resumen

En la actualidad, el desarrollo de software se ha convertido en uno de los pilares fundamentales para las industrias a nivel mundial, debido a los rápidos avances en el área de las tecnologías de la información. A causa de ello, las industrias de desarrollo de software se preocupan por la adaptación a las tendencias tecnológicas donde cumplan con las especificaciones de los usuarios finales, buscando estrategias de desarrollo que permitan la adaptación con el menor costo y tiempo.

El reúso de software es una de las estrategias que se considera promisoria para que la industria de desarrollo de software pueda enfrentar el reto de desarrollar productos con niveles de calidad y productividad adecuados en un contexto de negocio altamente complejo, dinámico y con acelerados cambios tecnológicos. Una de estrategias son los servicios Web, donde se basa el trabajo de esta presente tesis.

En este trabajo se propone un procedimiento para transformar el código de un Marco Orientado a Objetos hacía Marcos de servicios Web, buscando mejorar la modularidad de los Servicios Web, mediante 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 fundamental es balancear los niveles de coherencia, cohesión, factor de acoplamiento y el tiempo de respuesta; de la arquitectura interna de Servicios Web, 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.

Además se presenta la herramienta SR2-Transforming (Sistema de reingeniería para transformación), que implementa mediante una interfaz de usuario el método de transformación presentado en este trabajo generando Marcos de Servicios Web a partir de código legado de un Marco Orientado a Objetos.

Por último, establecieron pruebas que comprueben estadísticamente que el procedimiento que se presenta, cumple con las mejoras perseguidas como objetivo.

Abstract

Today, software development has become one of the fundamental pillars for industries worldwide, due to rapid advances in the area of information technologies. Because of this, software development industries are concerned with adapting to technological trends where they meet the specifications of end users, looking for development strategies that allow adaptation with the least cost and time.

Software reuse is one of the strategies considered promising for the software development industry to face the challenge of developing products with adequate levels of quality and productivity in a highly complex and dynamic business context with accelerated technological changes. One of the strategies is the Web services, where the work of this present thesis is based.

In this paper we propose a procedure to transform the code of an Object Oriented Framework into Web Services Frameworks, seeking to improve the modularity of Web Services, through the rational grouping of functions that participate in interactive sequences of use cases, as well as The data that is manipulated by these functions. The fundamental purpose is to balance the levels of coherence, cohesion, coupling factor and response time; Of the internal architecture of Web Services, in such a way as to achieve a balance between these attributes of quality, to obtain a better performance and better conditions of reuse and maintenance.

In addition, the tool SR2-Transforming (Reengineering System for Transformation) is presented, which implements through a user interface the transformation method presented in this work generating Web Service Frames from legacy code of a Object Oriented Framework.

Finally, they established tests that prove statistically that the procedure that is presented, complies with the improvements pursued as objective.

Índice

Lista de figuras

Lista de tablas

Capítulo 1

Antecedentes

1.1 Introducción

Un alto nivel de abstracción en las técnicas de reutilización, reduce el esfuerzo requerido para ir del concepto inicial (requerimiento) a su representación en una especificación y reduce el esfuerzo para ir de la abstracción a una implementación ejecutable. (de Páez, 2012)

El principal desafío en el desarrollo de software es mejorar la calidad y reducir el costo de las soluciones basadas en computadoras. Una manera de ayudar a cumplir con este objetivo es maximizar el reuso.

El reuso de software es una de las estrategias que se considera promisoria para que la industria de software pueda enfrentar el reto de desarrollar productos con niveles de calidad y productividad adecuados en un contexto de negocio altamente complejo y dinámico y con acelerados cambios tecnológicos. Para este fin, el uso de plantillas genéricas, componentes de diferentes niveles de granulación, patrones de diseño, marcos orientados a objetos, entre otros, son mecanismos cada vez más utilizados por los desarrolladores de software. El objetivo de dichas prácticas es lograr que el reuso se integre de forma sistémica en las diferentes etapas del desarrollo, de tal manera que su impacto en los diferentes artefactos resultantes del proceso de desarrollo sea efectivo y, en lo posible, medible. (Jacobson, Griss, & Jonsson, 1997)

El concepto de Marcos de aplicaciones orientado a objetos (MOO's) es un enfoque para facilitar el reuso de arquitecturas de software. Los MOO's representan soluciones en un contexto general y proveen mecanismos eficientes para su adaptación a comportamientos particulares en nuevas aplicaciones. Algunas de sus principales desventajas, que presentan, es que su arquitectura está fuertemente integrada, dependen de la plataforma y del lenguaje nativo en que fue desarrollado el MOO.

Una de las principales ventajas que ofrecen los marcos de aplicaciones orientados a objetos están: la reducción de costos de los procesos de desarrollo de aplicaciones de software para dominios específicos y la mejora de la calidad del producto final.

Sin embargo, no es posible aprovechar este beneficio en ambientes distribuidos por la incompatibilidad de plataformas y lenguajes de desarrollo. Por ejemplo, aún no se cuenta en la red con Servicios Web organizados por dominios, que puedan ser reusados en nuevas aplicaciones de dominios en diferentes plataformas y lenguajes de desarrollo o combinarlos con servicios desarrollados en otra plataforma.

Para tratar esta carencia se tendría que utilizar alguna de las estrategias que aparecen en los trabajos relacionados que incluyen métodos manuales, automáticos, estáticos, dinámicos, desde el código fuente o desde el modelado de sistemas, con el objetivo de identificar y generar Servicios Web. A la fecha no se tiene conocimiento de que existan ambientes de reingeniería que den soporte a la transformación de Marcos de Aplicaciones Orientados a Objetos hacia Marcos con arquitectura orientada a Servicios, que propongan

diferentes estrategias de transformación para obtener servicios web auto-suficientes a diferentes niveles de granulación y calidad balanceada entre rendimiento, acoplamiento y cohesión.

Realizar una transformación manual, implica que el desarrollador deberá tener la habilidad y conocimiento para realizar la descomposición del Marco original, la selección de clases, funciones y atributos agrupándolos de alguna manera estratégica para brindar las mejores características de calidad, en este proceso existe el riesgo de introducir defectos debido al proceso de conversión manual.

Los servicios Web (SW) emergen como una nueva tecnología que permite que las aplicaciones se comuniquen, a partir de la descripción de un conjunto de operaciones a las cuales se puede acceder por la red a través de mensajería XML estandarizada, de forma que no dependan de la plataforma. La tecnología de SW utiliza protocolos basados en lenguaje XML con el objetivo de describir las operaciones del servicio, o los datos a intercambiar con otros Servicios Web. Estos protocolos habilitan la independencia de plataforma de desarrollo, así como del lenguaje de programación, además en un nivel correcto de granulación es posible manejar los factores de calidad en su arquitectura interna. De esta manera con los SW se obtiene más ventajas para el reuso.

En investigaciones de antecedente, se ha planteado como objetivo aprovechar las ventajas de ambas tecnologías, la de MOO´s y la de SW. Para este fin, se han realizado investigaciones de desarrollo de tecnología para convertir Marcos Orientado a Objetos hacia Marcos de Servicios Web con arquitectura SOA (MSOA). Una de tales tecnologías fue desarrollada en el trabajo de Francisco León Pérez, “Generación de servicios Web desde marcos orientados a objetos a partir de clases colaborativas en casos de uso” (León, 2009). Esta investigación es el principal antecedente de la propuesta de tesis que se plantea en este documento. Con la investigación de (León, 2009) los Servicios Web obtenidos conforman un nuevo marco, pero ahora de Servicios Web en lugar de clases orientadas a objetos. Este conjunto de Servicios Web son generados desde la identificación de Casos de Uso en el MOO original.

Debido al nivel de granulación de los Servicios Web, generados con el enfoque de selección e integración de clases del trabajo citado anteriormente, y que conforman su estructura interna, se produce una menor autonomía entre sus clases, lo que ocasiona problemas de dependencias entre éstas, así como una degradación del tiempo de respuesta de los Servicios Web participantes para atender casos de uso de aplicaciones de negocios.

El proyecto de tesis de desarrollo tecnológico que se describe, en este documento, consiste en mejorar la modularidad de los Servicios Web que conforman Marcos de Servicios Web con arquitectura SOA, mediante 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 balancear los niveles de coherencia, cohesión, factor de acoplamiento y el tiempo de respuesta; de la arquitectura interna de Servicios Web, 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.

1.2 Trabajos relacionados

Para la evaluación de trabajos relacionados se hizo un estudio de nueve trabajos recientes, principalmente de los últimos seis años, es decir, se trató de seleccionar una muestra heterogénea de publicaciones de trabajos que incluyen diferentes valores de las facetas o criterios de evaluación. La comparativa incluye trabajos que difieren un poco en las siguientes facetas o criterios de evaluación:

- **Objetivo.** Se considera este criterio de comparación para establecer una diferencia en cuanto al objetivo que se persigue con la aportación de cada uno de los trabajos estudiados, en contra con el objetivo perseguido en este trabajo de tesis. El objetivo de algunos de los trabajos estudiados consiste en:

  - CU: Recuperación de la documentación asociada a procesos de negocios o procesos de software.
  - CMP: Identificación de componentes de software.
  - BS: Obtención de la especificación de servicios de negocios sin llegar a su implementación.
  - SS: Obtención de la especificación de servicios de software sin llegar a su implementación.
  - SW: Obtención de servicios web implementados.
- **Enfoque metodológico**. Los trabajos analizados difieren en el enfoque de tratamiento para alcanzar su objetivo. El enfoque de algunos de los trabajos estudiados consiste en:

  - *Top-Down* (T): parte del análisis del negocio o el modelado del negocio o de software, para identificar a través de un proceso de ingeniería directa, casos de uso o servicios.
  - *Bottom-Up* (B): parte del análisis del código fuente de sistemas legados para derivar casos de uso o servicios por medio de un proceso de ingeniería inversa.
  - *Meet-in-the-Middle* (M): Realiza una combinación de estrategias *Top-Down* y *Bottom-Up*.

- **Proceso**. Este criterio se utiliza para distinguir el grado de automatización del método o enfoque propuesto. Los valores son:
  - AA: Si el enfoque propuesto es automático.
  - SA: Si el proceso es automático con un mínimo de intervención humana.
  - MAN: Si el enfoque del proceso es principalmente manual, quizás con algún grado mínimo de automatización.

- **Utilidad**. Este criterio de la comparativa indica si el producto/resultado es útil para:
  - Actores externos (AE): Que interactúan con el proceso a través de una interfaz al usuario (IU)
  - Actores lógicos (AL): Que interactúan e intercambian información en un proceso
  - (AE+AL) Para ambos tipos de actores

- **Entrada al proceso**. Los diferentes enfoques metodológicos estudiados, difieren en las entradas al proceso. Algunos trabajos empiezan el proceso con:
  - Modelos de Clases (AC).
  - Modelos de Casos de Uso (CU).
  - Modelos de Escenarios (ME).
  - Documentación del Negocio (DN).
  - Modelos de Proceso (MP).
  - Meta-Modelos (MM).
  - Modelos Arquitecturales (*Model Driven Architecture*) en diferentes niveles de abstracción (CIM, PIM, PSM, ISM).
  - Código Fuente (COD).

- **Aportación**. Este criterio se utiliza para establecer el grado de innovación u originalidad de los diferentes trabajos relacionados, se consideran trabajos que ofrecen:
  - Una nueva Técnica (T)
  - Una metodología (Me)
  - Un modelo (Mo)
  - Una herramienta de software (H)
  - Un marco de trabajo de soporte al desarrollo (FrmW)

  Se considera que la aportación tendrá directamente un impacto total o parcial en el alcance del objetivo de cada enfoque estudiado.

- **Producto/Resultado**. Este es el producto derivado de la aportación realizada. El cual sirve de instrumento para alcanzar, parcial o totalmente, el objetivo del trabajo. Dentro del rango de valores podemos citar:
  - Modelo de Casos de Uso (MCU).
  - Modelo de Secuencias de Interacción (MSEQ).
  - Servicios de Negocios (BS).
  - Servicios de Software (SS).
  - Servicios Web (SW).
  - Framework de Ingeniería (FrmW).
  - Paquetes de Software (PS) ligados a Servicios Web.
  - Especificación Funcional de Servicios (Fun).

- Marcos de Servicios Web con arquitectura SOA (MSOA).
- Modelo de Servicio (MS).
- Componentes de software (CMP).

Tabla 1. Tabla comparativa de trabajos relacionados.

Trabajo Relacionado	Objetivo	Enfoque	Proceso	Entrada	Aportación	Producto/ Resultado	Utilidad
"Service identification in interorganizational process design". (Bianchini, Cappiello, De Antonellis, & Pernici, 2014)	BS	T	SA	MP	Me	BS+PS	AE+AL
"Extracting SOA Candidate Software Services from an Organization's Object Oriented Models". (Yousef, Adwan, & Abushariah, 2014)	SS	T	MAN	AC+MCU	Me	SS	AE
"Dynamic Decision Tree for Legacy Use-case Recovery". (Dugerdil & Sennhauser, 2013)	CU	T	MAN	ME	T+H	MCU	AE
"Creating Web Services from Legacy Code". (Goyal & Jain, 2012)	SW	B	SA	COD	FrmW	SW	AE+AL
"Identification and analysis of business and software services—a consolidated approach".(Kohlborn, Korthaus, Chan, & Rosemann, 2009)	BS+SS	T+M	MAN	DN+MP	Me	SS	AE+AL
"Identification of the Web Services starting from the existing Web applications". (Nakkach, Kraiem, & Ben Ghezala, 2006)	SS	M	SA	MM	FrmW	SS	AE
"Recovering Use Case Diagrams from Object Oriented Code: An MDA-based Approach". (Claudia, Liliana, & Liliana, 2011)	CU	B	MAN	PIM	FrmW	MCU	AE
"Component identification method with coupling and cohesion". (Lee, Jung, Kim, Jang, & Ham, 2001)	CMP	T	MAN	CU	Me	CMP	AL
"Use case-based service-oriented analysis and modeling". (Liu, Fu, Luo, & Zou, 2013)	SW	T	MAN	MCU	Me	MS	AE+AL
En este trabajo	SW	B	SA	COD	Me+H	MSOA	AE+AL

El desarrollo tecnológico que se describe en esta tesis, al cuál de aquí en adelante denominaremos "*Inlineclass",* plantea obtener Marcos de servicios Web desde el código de Marcos Orientados a Objetos (MOO), al igual que el desarrollo tecnológico de antecedente en (León, 2009). La diferencia entre estos dos trabajos está en la arquitectura interna de cada servicio Web generado. En la tesis de (León, 2009), la arquitectura interna de cada servicio Web consiste en un conjunto de clases relacionadas, con los atributos y funciones necesarias para atender un requerimiento como meta de valor. En esta tesis, se ha planteado que la arquitectura interna de cada servicio Web consista en reunir en una única clase de objetos, los atributos y funciones necesarias para atender un requerimiento como meta de valor, esperando obtener mejor tiempo de respuesta de los servicios Web obtenidos a partir del MOO original.

El propósito que se persigue en el trabajo "*Dynamic Decision Tree for Legacy Use-case Recovery*" (Dugerdil & Sennhauser, 2013) es re-documentar los casos de uso con todos sus flujos relevantes, desde escenarios simples descritos por los usuarios. La técnica aclamada utiliza un método que se caracteriza en gran medida por la intervención humana y por la ejecución de trazas de eventos para determinar los escenarios alternos de los casos de uso.

En el trabajo "*Recovering Use Case Diagrams from Object Oriented Code: An MDA-based Approach*" (Claudia et al., 2011), los autores describen como recuperar diagramas de Casos de Uso, a nivel PIM's, desde código Java a través de un proceso de ingeniería inversa. Muestran como los Metamodelos MOF pueden ser usados para analizar la consistencia de procesos de recuperación de modelos. Para ello proponen un framework a partir de una Arquitectura dirigida por Modelos (Model Driven Architecture) basado en ingeniería inversa con intervención humana.

El trabajo "*Component identification method with coupling and cohesion*" (Lee et al., 2001), en comparativa con el trabajo de esta tesis, difiere en su objetivo, ya que éste tiene como objetivo identificar componentes reusables de software mediante un proceso manual de ingeniería directa teniendo como entrada Modelos de Caso de Uso, mientras que el de este trabajo de tesis tiene como objetivo identificar e implementar Marcos de Servicios Web a partir de los Casos de Uso encontrados en el código legado de Marcos Orientados a Objetos, mediante un proceso de reingeniería Bottom-Up.

De los trabajos estudiados cuyo objetivo coincide con el que se propone en este documento, en cuanto a la identificación y creación de servicios web, sólo "*Creating Web Services from Legacy Code*" (Goyal & Jain, 2012) y "*Use case-based service-oriented analysis and modeling*" (Liu et al., 2013) coinciden. La diferencia entre estos trabajos contra nuestra propuesta está en la metodología, la aportación y el producto/resultado final obtenido.

En el trabajo propuesto por Vinay Goyal (Goyal & Jain, 2012) la técnica aclamada utiliza un enfoque Bottom-Up que parte del análisis del código fuente, su método se caracteriza por la intervención de un experto en el negocio, quien realiza el análisis del flujo de datos para determinar los estatutos que los acceden y agruparlos en bloques de código, los cuales son empacados para proveerles de una interfaz WSDL y posteriormente

ensamblados en aplicaciones mediante un software intermediario que enlaza cada servicio web con los procedimientos y protocolos del negocio.

En cuanto al trabajo de Wenjun Liu (Liu et al., 2013), éste presenta un método Top-Down de análisis y diseño orientado a servicios (SOAD) basado en Casos de Uso. El método se enfoca en las fases de análisis de un proyecto SOA, para identificar y describir servicios basados en los requerimientos de aplicaciones.

Los trabajos relacionados (Bianchini et al., 2014), (Yousef et al., 2014), (Kohlborn et al., 2009) y (Nakkach et al., 2006), todos tienen por objetivo identificar servicios, su alcance llega a la especificación funcional tanto de servicios de negocios como de servicios de software, todos tienen un enfoque *Top-Down* o *Meet-in-the-Middle*, por lo que la entrada a sus procesos son los modelos del negocio en alguna de sus variantes.

La intención del trabajo descrito en (Bianchini et al., 2014), está dirigida hacia la obtención de candidatos a servicios bajo arquitectura SOA. Su enfoque es el *Top-Down*, su método parte del análisis manual de modelos de procesos de negocios, de acuerdo a dos perspectivas: análisis de valor y análisis de dependencia de tareas, desde el que se definen las dependencias de flujo y de datos entre tareas inter-organizacionales, agrupándose las secuencias de tareas interactivas en servicios, considerando las métricas de cohesión, acoplamiento para determinar su granulación.

La intención del trabajo descrito en (Yousef et al., 2014) también está orientada a la obtención de candidatos a servicios bajo arquitectura SOA, los cuales, posteriormente, pueden ser implementados como Componentes reusables de Software. Se utiliza la documentación técnica del modelado de clases y de casos de uso del software existente. La metodología parte del análisis manual de la documentación técnica asociada al sistema desde la que se definen pares entidad-tarea mediante una matriz que representa como filas a las tareas o funciones distinguidas desde los casos de uso y como columnas a las entidades de software distinguidas desde las clases de objetos, agrupando en servicios a los pares entidad-tarea por sus secuencias interactivas. Permite descubrir casos de uso que pueden ser utilizados por actores externos. Para agrupar las actividades en servicios se utilizan las sugerencias de actores humanos que operan los procesos del negocio.

En (Kohlborn et al., 2009) la intención está orientada a la obtención de candidatos a servicios bajo arquitectura SOA tanto de servicios de negocios como de servicios de software a nivel de servicios de procesos, compuestos por servicios de tarea, de entidad y utilitarios. El trabajo incorpora las mejores prácticas de enfoques de otras metodologías atendiendo subjetivamente los principios de diseño de servicios SOA y atributos de calidad como cohesión, acoplamiento, autonomía y reuso. El método es manual; empieza con un proceso de derivación de servicios de negocios que parte de un análisis manual del contexto y de procesos del negocio. La salida de este proceso es la definición de servicios de negocios que incorporan la estrategia SOA definida, así como un modelo de operación que orquesta las funcionalidades de los procesos identificados, que deben intervenir para alcanzar la meta de la operación y el modelo de interacción que representa la forma o coreografía en que un servicio de negocios define cómo será realizada la colaboración.

En (Nakkach et al., 2006) se propone un *framework* de ingeniería de web. El *framework* cuenta con un método de clasificación de aplicaciones Web; una etapa para la descomposición de aplicaciones web para identificar funcionalidades lógicas de alto nivel de granulación y reuso; y un software de transformación de modelos. A la salida, el framework proporciona las especificaciones funcionales de Servicios basados en Web, justificadas en términos de interoperabilidad, flexibilidad, acoplamiento débil y mayor nivel de reuso.

Las diferencias fundamentales, de estos trabajos, desde la perspectiva del enfoque metodológico de este trabajo de tesis, son:

1. El proceso propuesto en este trabajo de tesis es mayormente automático.
2. El enfoque es hacia la obtención de Marcos de Servicios Web de dominios de aplicaciones de negocios.
3. Implementados con un enfoque *Bottom-Up*, iniciando con un proceso de ingeniería inversa del código legado del dominio del negocio y termina con un proceso de ingeniería directa que produce el marco de servicios web del mismo dominio original.
4. Como punto de partida, utiliza el código fuente del software existente en MOO's del dominio del negocio.
5. Brinda soporte tanto a actores externos como a actores lógicos.
6. Cada Servicio Web es implementado en una única clase, con objeto de mejorar el tiempo de respuesta del servicio.
7. Se obtienen Marcos de Servicios Web en niveles de granulación de método o función, requerimiento o componente y de proceso del negocio.

1.3 Planteamiento del problema

La presente tesis busca resolver el problema que radica en el enfoque dado en el trabajo mencionado en el antecedente, "Generación de Servicios Web desde marcos orientados a objetos a partir de clases colaborativas en casos de uso" (León, 2009), para seleccionar las clases de objetos que integran la arquitectura interna de un componente o Servicio Web produce poca autonomía interna, lo que ocasiona dependencia de clases en la estructura interna de los Servicios Web. Además, según (Guadarrama, 2013) "Estudio Experimental para determinar la relación entre la estructura interna de Servicios Web y valores de QoS", con este nivel de granulación, no se tiene el mejor tiempo de respuesta entre los Servicios Web participantes para atender casos de uso de aplicaciones de negocios.

La solución que se propone en esta tesis consiste en obtener Servicios Web en el nivel de granulación gruesa, basados en estudios experimentales del trabajo de antecedente "Estudio Experimental para determinar la relación entre la estructura interna de Servicios Web y valores de QoS" (Guadarrama, 2013), desde la perspectiva de ingeniería de software para lograr mayor auto-suficiencia para reuso, coherencia, cohesión y mejora en tiempo de respuesta.

1.4 Objetivo

1.4.1 Objetivo general

Reducir el acoplamiento que existe entre las clases colaborativas para satisfacer un requerimiento para un Caso de Uso, mediante un enfoque alternativo de atributos y funciones en una única clase para mejorar la auto-suficiencia y tiempo de respuesta de Servicios Web en aplicaciones web de negocios.

1.4.2 Objetivos específicos

1. Contar con un medio alternativo estratégico para obtener Servicios Web de diferente granulación, que permita tener balance estratégico entre las medidas de autosuficiencia, tiempo de respuesta, cohesión, acoplamiento y coherencia.
2. Extender la funcionalidad del "SR2 Refactoring", para dar soporte a la generación de servicios web desde marcos de aplicaciones orientados a objetos.

Capítulo 2

Procedimiento de generación de Servicios Web

2.1 Criterio de identificación de casos de uso

Para identificar el inicio de cada caso de uso, se emplea el siguiente razonamiento: Si una clase es pública significa que está disponible a clientes del Marco Orientado a Objetos (MOO), otorgando posibilidad de acceder a ésta y a sus miembros públicos. La invocación de un método público de una clase pública del MOO representa la solicitud de un requerimiento por cliente, es por esto se considera el inicio de un caso de uso.

Desde la perspectiva anterior, todos los métodos públicos de las clases públicas no abstractas del MOO son iniciadores de casos de uso. Se considera que el extremo final de un caso de uso sucede cuando un método iniciador termina satisfactoriamente.

Cuando un método iniciador invoca a otro método se inicia una secuencia de interacción, la cual puede continuar hasta que el último método invocado de la secuencia ya no invoca a ningún otro método. En la Figura 1 se representa un ejemplo de secuencias de interacción entre métodos mediante un grafo dirigido.

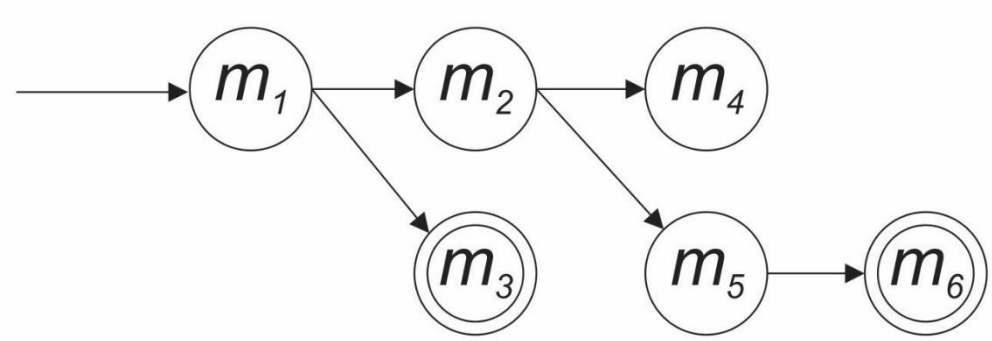

Figura 1. Ejemplo de un grafo de secuencia de interacción entre métodos para un caso de uso.

En el grafo de la Figura 1, se considera que el método m_1 es público y que pertenece al comportamiento de una clase pública no-abstracta, por lo tanto m_1 es una entrada para este Caso de Uso, en el cual existen dos secuencias de interacción. En este ejemplo, cada secuencia de interacción representa un escenario alternativo que lleva a la meta de valor del Caso de Uso, partiendo del método iniciador m_1. Para este ejemplo, cada una de las secuencias termina en m_3 y m_6. En la Figura 2, se describe el comportamiento del Caso de Uso del ejemplo en el grafo de la Figura 1 mediante un diagrama de secuencias.

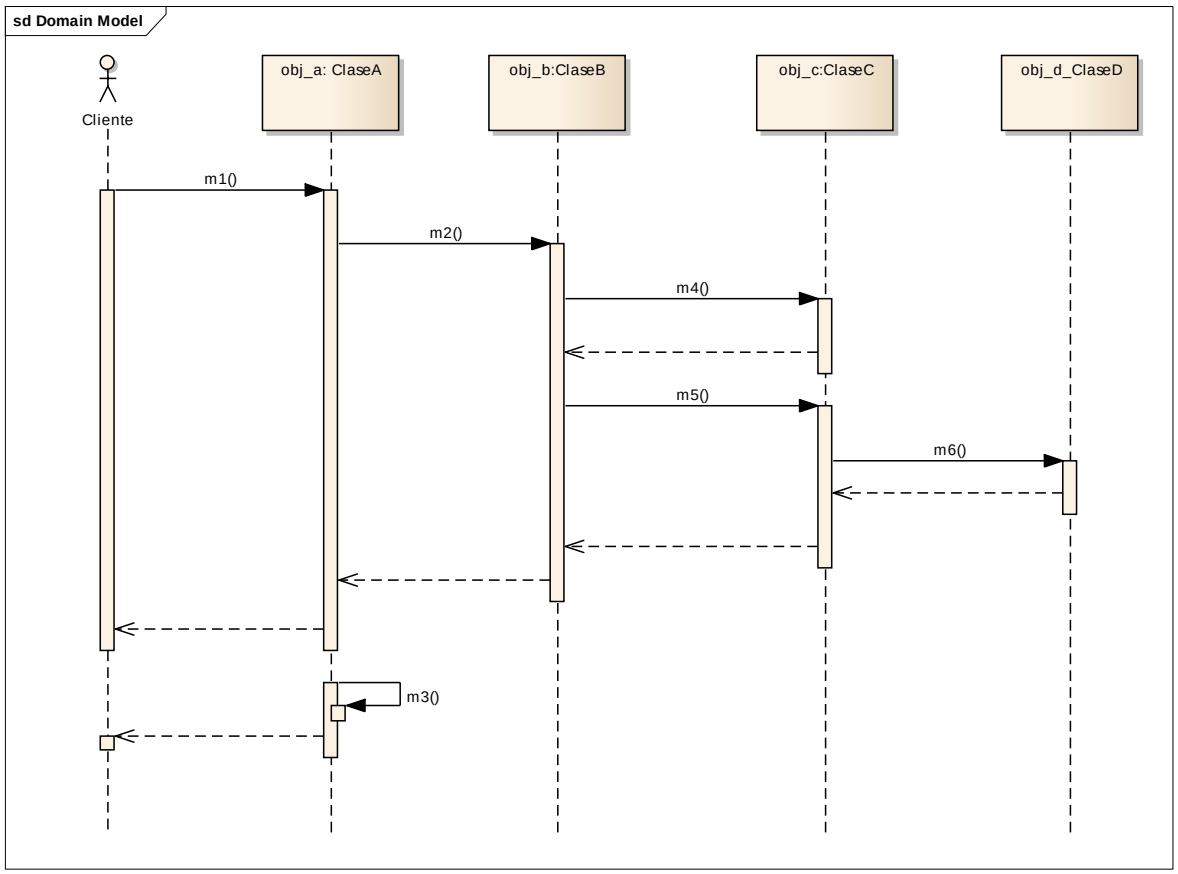

Figura 2. Diagrama de secuencias del ejemplo de Caso de Uso del grafo de la Figura 1.

Todos los métodos son necesarios para satisfacer cualquier variante de ejecución del caso de uso. A los métodos que integran las secuencias de interacción encontradas se les llamará *métodos del caso de uso*.

Los métodos del caso de uso utilizan, además de los métodos que invocan, a otros miembros de clase durante su ejecución. Estos miembros son: atributos, clases internas e interfaces internas, que también son necesarios para la satisfacción del caso de uso. Los constructores se consideran también como métodos del caso de uso y forman nuevas secuencias de interacción.

Con base en lo anterior, el código necesario para satisfacer el caso de uso iniciado por un método *m* perteneciente a una clase es el correspondiente a:

1. El método *m*.
2. Los métodos del caso de uso. Éstos son los métodos integrantes de todas las secuencias de interacción *S1*, *S2*,... ,*Sn* a partir de *m*.
3. Los miembros de clase *x1*, *x2*,... *xn* que los métodos del caso de uso utilicen.
4. Las declaraciones de las clases a las que pertenecen los métodos del caso de uso (punto 2) y los miembros de clase utilizados (punto 3).

5. La declaración *package* y *l*as declaraciones *import* utilizadas (en caso de existir).

Para determinar el nombre de los métodos de un caso de uso se considera utilizar el nombre del método invocado seguido del nombre de la clase. Este proceso se realiza de la misma forma que el proceso igual al anterior, es decir, considerando las secuencias. En conclusión para el procedimiento de identificación de CU observar los siguientes puntos:

- Elegir sólo el código necesario para la satisfacción del CU e ignorar el resto.
- Toda la funcionalidad del Caso de Uso agruparla en una única entidad de software (clase).

2.2 Descripción del procedimiento de transformación

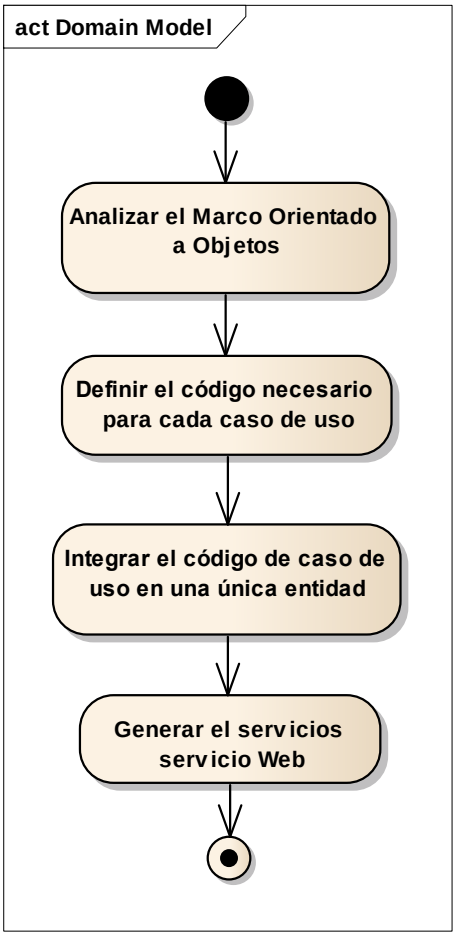

Figura 3. Diagrama de actividades para el procedimiento de transformación.

El procedimiento que se propone consta de los siguientes pasos:

1. Analizar el Marco Orientado a Objetos.
2. Identificar el código necesario para cada caso de uso.
3. Integrar el código de caso de uso en una única entidad.
4. Generar el servicio Web.

En las siguientes secciones se describen cada uno de estos pasos:

2.2.1 Analizar el Marco Orientado a Objetos

Este proceso consiste en ejecutar un analizador sintáctico (Gesser, 2008) sobre el código del marco orientado a objetos (MOO) y obtener un repositorio con la información de cada elemento de clases definidas en el marco. A este repositorio se le denomina *Diccionario de datos*, propuesto en (León, 2009).

El MOO deberá estar desarrollado en el lenguaje java y el código deberá estar libre de errores sintácticos y un funcionamiento correcto. Este proceso requiere de las siguientes acciones:

a) Selección del MOO que debe analizarse desde la ruta de directorio en el que se crearán los servicios Web. El usuario deberá proporcionar la estructura de directorios donde se encuentra el código del MOO.
b) Análisis sintáctico al código del MOO de entrada.
   Cada archivo *.java* del marco se procesará con un analizador sintáctico de lenguaje Java. La información que deberá almacenarse consistirá en:
   1) Información básica de las clases: nombre calificado de la clase, modificador de acceso, nombre del archivo en el que se encuentra definida, nombre de su clase base e interfaces cuando aplique e información de sus constructores, atributos, métodos y clases internas.
   2) Información básica para las interfaces: nombre calificado de la interfaz, nombre del archivo en el que se encuentra definida, nombre de las interfaces que implementa (cuando aplique) e información de sus atributos.
   3) Información de constructores de una clase: nombre del constructor, firma del constructor y clase origen.
   4) Información de atributos de clases e interfaces: modificador de acceso, nombre del atributo, clase origen y tipo de atributo.
   5) Información de métodos de una clase: modificador de acceso, nombre del método, firma del método, tipo de retorno y clase origen.
   6) Información de clases internas: será la misma que en el inciso 1) y además la clase base.
   7) Información de interfaces internas es la misma que el inciso 2) y además la clase origen.
   8) Información de inicializadores sintácticos: es sólo la línea y columna en donde se encuentra.
   9) Información de enumeraciones: son los números de línea y columna en donde se encuentra, y la cantidad de valores que posea.

2.2.2 Definir el código para cada caso de uso

Consiste en seleccionar las clases (y sus elementos) del marco orientado a objetos 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).

Se debe contar con el diccionario de datos resultante de analizar el marco orientado a objetos. Al finalizar este paso se tendrán dos listas para cada caso de uso:

1. Una lista que contiene las clases e interfaces que cada caso de uso requiere, llamada *lista de clases*.
2. Para cada clase de la lista anterior, una lista de 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, 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:

a) Identificar los métodos iniciadores de cada caso de uso.
b) Identificar precondición del caso de uso.
c) Identificar los métodos de caso de uso y miembros de clase utilizados.
d) Identificar cláusulas de manejo de excepciones utilizados.
e) Depurar el contenido de las listas.
f) Agrupar el código del caso de uso a una única entidad.

La realización de las primeras cuatro acciones dará como resultado clases y elementos de ellas que se agregarán a las correspondientes listas de cada caso de uso. El inciso *e* tiene el objetivo de garantizar que el código seleccionado para satisfacer cada caso de uso funcione correctamente en el servicio Web, al agregar las declaraciones que le sean indispensables. La última acción tiene como objetivo agrupar toda la funcionalidad del caso de uso en una única clase, partiendo de esta manera el método de transformación presentado en este trabajo con el nombre *Inlineclass*. Cada acción se describe en las secciones siguientes.

2.2.2.1 Identificar los métodos iniciadores de cada caso de uso

Los métodos que se consideran iniciadores de caso de uso son los métodos públicos de las clases públicas no abstractas del marco. Para identificarlos, se realiza la siguiente evaluación como en (León, 2009):

Considerar una clase del marco:

1. Si la clase no es pública, no existen iniciadores de caso de uso en ella.
2. Si la clases es pública, verificar si la clase es abstracta:
    a. Si la clase es abstracta, no existen iniciadores de caso de uso en ella.
    b. Si la clase no es abstracta (concreta), todos los métodos públicos son iniciadores de caso de uso.

El método iniciador es el primer *método del caso de uso*. La clase en la que está declarado se agregará a la lista de clases del correspondiente caso de uso como *Clase Iniciadora del caso de uso (CI)* donde representará la clase que contiene el método iniciador de caso de uso y se tomará como la entidad para agrupar el código total del caso de uso mediante el método de transformación. Para la *lista de elementos de clase* se

agregará el método iniciador y también se agregará a la *lista de clases de cada caso de uso* a la que defina al tipo de retorno del método iniciador.

2.2.2.2 Identificar precondición de cada caso de uso

Un constructor es una subrutina cuya misión es inicializar cada objeto de una clase. En el constructor se asignan los valores iniciales del nuevo objeto. Es por ello que se define como precondición de un caso de uso, si existe cohesión entre el método iniciador y los atributos de la clase cuando se asigna los valores iniciales del objeto que el método requiere para su adecuada funcionalidad. Se deben considerar los siguientes criterios para agregar a *la lista de elementos de clase*:

1. Si el constructor no contiene comportamiento dentro del bloque de código, no considerarlo como precondición del caso de uso.
2. Si el constructor tiene comportamiento dentro de bloque de código, verificar:
    a. Los atributos a inicializar son cohesivos con el método iniciador, considerar como precondición del caso de uso.
    b. Si los atributos a los que se le inicializa un valor no son cohesivos con el método iniciador, no considerar el constructor como precondición del caso de uso.
    c. Si se invoca a un método dentro del constructor y el método invocado está valorado como *método de caso de uso*, considerarlo como precondición del caso de uso.

En la siguiente figura se representa un ejemplo para una precondición de caso de uso, donde los atributos *VectorB* y *VectorC* de la clase *BrayCurtis* son cohesivos con el método *calcular()* que para la clase es el método iniciador de caso de uso (§2.2.2.1). A razón de ello, se considera como precondición de caso de uso:

```
public class BrayCurtis extends EspacioVectorial {

    ArrayList VectorB;
    ArrayList VectorC;

    public BrayCurtis(ArrayList VectorB, ArrayList VectorC) {
        this.VectorB = VectorB;
        this.VectorC = VectorC;
    }

    @Override
    public double calcular() {
        Estadistico oEs;
        double sum = 0, dif = 0;
        SumatoriaXY sumatoriaxy = new SumatoriaXY();
        sum = sumatoriaxy.obtener(VectorB, VectorC);
        oEs = new SumaDiferenciasAbsolutas();
        dif = oEs.obtener(VectorB, VectorC);
        return 1 - (dif / sum);
    }
```

Figura 4. Ejemplo de precondición de caso de uso.

Si el constructor contiene o no una firma (recibe parámetros), es posible que los valores con los que se inicialice los atributos sean mediante una literal o un objeto. Dado este hecho, es irrelevante la firma del constructor para considerarlo como precondición del caso de uso. Por ejemplo, se podría considerar el constructor de la clase *BrayCurtis* (del ejemplo anterior) donde se inicialicen los atributos sin ningún objeto de valor recibido como parámetro del constructor:

```
public BrayCurtis() {
    this.VectorB = new ArrayList<String>() {{
                    add("A");
                    add("B");
                    add("C");
                    }};
    this.VectorC = new ArrayList<String>() {{
                    add("d");
                    add("e");
                    add("f");
                    }};
}
```

Figura 5. Ejemplo de inicialización de atributos sin firma en el constructor.

2.2.2.3 Identificar los métodos de caso de uso y miembros de clase utilizados

Los métodos de cada caso de uso se determinan al identificar las secuencias de interacción entre métodos descritas en el criterio en el que se basa este procedimiento (§2.1). Para determinar estos métodos es necesario conocer cuáles son los métodos iniciadores de cada caso de uso (§2.2.2.1). Cada caso de uso tendrá sus propios métodos.

Se analiza el código de cada método iniciador de caso de uso y se identificarán los mensajes o invocaciones a otros métodos. Cada método invocado será un método del caso de uso. La búsqueda de invocaciones se realiza para cada método del caso de uso hasta que el código de todos haya sido analizado. Estos métodos del caso de uso tienen como origen una invocación en el código de un método, sin embargo, éste no es el único lugar donde se puede invocar un método: también pueden invocarse desde el código de un constructor.

Por lo anterior, también se deben identificar los constructores que se utilizan en el caso de uso con el fin de localizar otras invocaciones a métodos o constructores que formarán nuevas secuencias de interacción. Los métodos de estas secuencias y los constructores usados también serán métodos del caso de uso. Los constructores que participan en un caso de uso serán los que:

- Se utilicen en invocaciones explicitas para la creación de instancias de clase con la cláusula *new*. Esto se realiza en el código de un método del caso de uso o en la declaración de un atributo que usa el caso de uso.
- Se invoquen en un constructor del caso de uso mediante las palabras *this* o *super* (invocación implícita de constructor).
- Contengan comportamiento dentro del bloque código del constructor, ya que sin un comportamiento en el constructor no se define un valor inicial para los atributos dentro de la clase o invocaciones a métodos dentro del constructor.
- Exista cohesividad entre el constructor y los atributos del caso de uso.

2.2.2.4 Identificar las cláusulas de manejo de excepciones utilizados

Aunque en esta tesis no se están tratando las cláusulas de manejo de excepciones, es importante considerarlo como parte del proceso de definición de código para cada caso de uso.

En el código obtenido hasta el momento para cada caso de uso se debe identificar el uso de cláusulas de excepciones, para incluir a las clases que los definen en la lista de clases de caso de uso en donde se utilicen. Las clausulas se usan en operaciones como throws y try… catch, entre otras. (León, 2009).

2.2.2.5 Depurar listas

La depuración de las listas tiene el objetivo de garantizar que el código seleccionado funcione correctamente al utilizarse de forma independiente. Es por ello importante incluir aquellos elementos indispensables para la lista de clases y sus elementos que se necesitan para satisfacer cada caso de uso. Los aspectos que se consideran en esta depuración para cada caso de uso son:

- Herencia en las clases de la lista de clases.
- Profundidad de herencia.
- Declaraciones de métodos en interfaces del caso de uso y en las clases que implementen los métodos.
- Clases internas en la lista de clases.

A continuación se describe cada uno de los aspectos mencionados:

Herencia en las clases de la lista de clases.

Se debe verificar que en la *lista de clases* no existan clases "huérfanas" que se puedan reflejar en el código para el caso de uso. Esto es, si una clase es derivada, la clase que extienda otra clase o implemente una interfaz se deben encontrar también en la *lista de clases*; y si una interfaz es derivada, la interfaz o interfaces que extienda se deben encontrar también en la *lista de clases*. (León, 2009)

Profundidad de herencia.

Se debe verificar que todos los métodos sobre-escritos o atributos heredados existan de su clase origen. En otras palabras, si una clase extiende a otra clase o implementa a una interfaz deben de existir en la *lista de elementos de clase* los métodos que fueron sobre-escritos en la clase/interfaz base. Se deben encontrar los miembros utilizados sin importar en el nivel de la jerarquía de herencia que contenga la clase derivada.

El siguiente ejemplo (figura 6) ilustra un diagrama de clases un una profundidad de herencia.

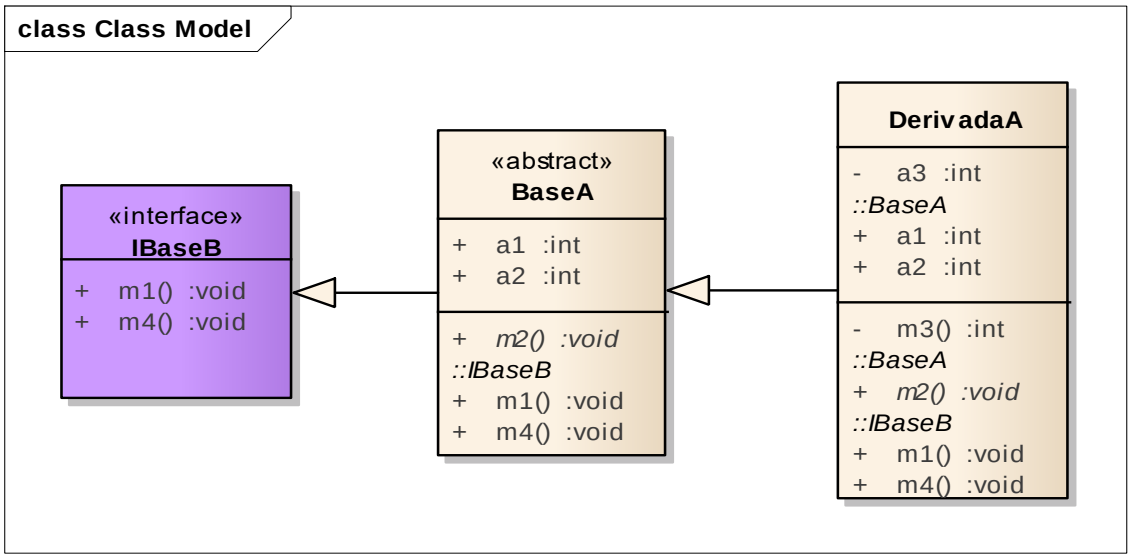

Figura 6. Diagrama de clases para profundidad de herencia.

Suponer que el método *m1()* y *m2()* son sobre-escritos por la clase *DerivadaA* y el método *m4()* es sobre-escrito por la clase abstracta *BaseA*, entonces cada uno de los métodos sobre-escritos deben existir de su clase origen. En este caso *m1()* y *m4()* pertenecen a la interfaz *IBaseB* y *m2()* a la clase abstracta *BaseA*. En el caso de no existir los métodos desde su origen, el caso de uso no tendrá un adecuado funcionamiento si se llegará utilizar polimorfismo. En este ejemplo al crear un objeto del tipo *BaseA* y darle forma de la clase *DerivadaA* para utilizar el comportamiento implementado en *m2()* y de la misma manera para la interfaz *IBaseB* con los métodos *m1()* y *m4()*.

Además deben ser considerados los atributos de la clase base dentro de la *lista de elementos de clase* en la clase origen. El siguiente ejemplo muestra la utilización de los atributos *a1* y *a2* de la arquitectura de ejemplo.

```
private int m3(){
  int resultado = 0;
  a1*= 3;
   a2 += Math.abs(a1-10);
   resultado = 10 +a2;
}
```

Figura 7. Código de ejemplo de utilización de atributos heredados.

Declaraciones de métodos del caso de uso en interfaces y en las clases que las implementan.

Se debe verificar que todos los métodos que se encuentren en la *lista de elementos de clase* y de las interfaces que se encuentren en la *lista de clases* existan también en la *lista de elementos de clase* que implementan la interfaz.

Las declaraciones de los métodos de las interfaces en la lista de clases deben agregarse a la lista de elementos de clase de todas las clases que las implementan.

Clases internas en la lista de clases.

Se debe de verificar que en la *lista de clases* para cada clase interna exista una clase origen de la misma. Dichas clase origen debe de contener en su lista de elementos de clase a la clase interna. Esto se debe a que una clase interna se considera como miembro de una clase. (León, 2009).

2.2.3 Integrar el código del caso de uso en una única entidad

A partir de la lista de clases y la lista de elementos conformadas en los pasos anteriores, se integrará el código para cada caso de uso en una única entidad. Es decir, una sola clase que contenga todo la funcionalidad correspondiente para cada caso de uso. Por lo tanto, lo que se persigue es obtener servicios Web en el nivel de granulación que, desde la perspectiva de la ingeniería de software, se logré ventajas en auto-suficiencia y en tiempo de respuesta. Esto se basa en estudios experimentales de la investigación de Luis César Guadarrama (Guadarrama, 2013); donde se concluye que el mejor tiempo de respuesta lo dan aquellos servicios Web que contenga una arquitectura de grano grueso.

Este enfoque de agrupamiento considera construir servicios Web de granulación gruesa, es decir, agrupar todos los elementos y miembros que participan en el caso de uso en una única clase. Por lo tanto, cada servicio Web construido deberá tener un único punto de arranque (método iniciador de caso de uso) mientras que el resto de los métodos de caso de uso que participan en la secuencia interactiva, no deberán estar accesibles hacia el cliente lógico o externo. A diferencia con el enfoque propuesto por Francisco León Pérez (León, 2009), este enfoque no contempla respetar la arquitectura original del Marco Orientado a Objetos.

A continuación se el proceso a considerar para agrupar toda la funcionalidad del caso de uso en una única clase de objetos:

2.2.3.1 Remover herencia

Se deben remover abstracciones (clases abstractas e interfaces) dentro de la lista de clases e integrar sus elementos hacia la clase derivada que le sea de utilidad para su funcionamiento. Estos elementos pueden ser enumeradores, atributos y métodos que contengan un comportamiento. Considerar el siguiente diagrama de clases que se presenta en la figura 8:

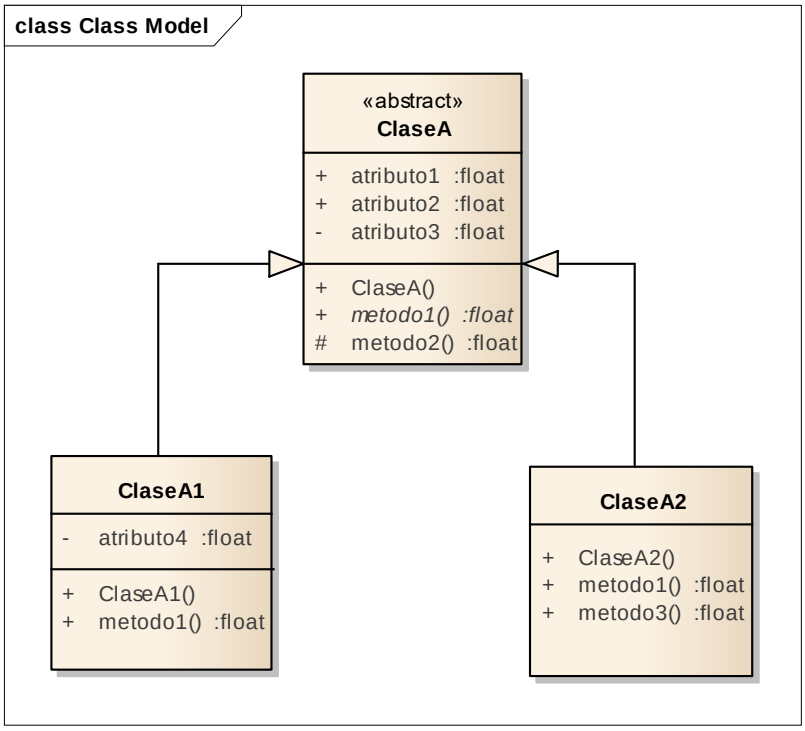

Figura 8. Diagrama de clases para ejemplo de agrupamiento de código.

La *ClaseA* es una clase abstracta que se reconoce como la abstracción para las clases *ClaseA1* y *ClaseA2* donde heredan sus atributos y métodos públicos y protegidos de *ClaseA*. Además, dichas clases implementan distinto comportamiento al método abstracto *metodo1()* el cual, para este ejemplo, se consideran como métodos iniciadores de caso de uso. En este ejemplo el *metodo3()* de la clase *ClaseA2* no se considera como método iniciador de caso de uso.

En el código del método siguiente: El método *ClaseA1.metodo1()* dentro de su bloque de código utiliza los atributos, *atributo1* y *atributo4* e invoca al método *metodo2()*.

```
public  float metodo1(){
  atributo1 = metodo2()*2;
  atributo4= atributo1 * Math.PI;
  return atributo4;
}
```

El método *ClaseA2.metodo1()* dentro de su bloque de código utiliza los atributos, *atributo1* y *atributo2* e invoca al método *metodo3()*. Por lo tanto, el código del método es el siguiente:

```
public  float metodo1(){
  atributo2 = metodo3()*2;
  if(atributo2!=0)
      atributo1 = atributo2 * Math.PI;
  return atributo1;
}
```

De acuerdo al criterio anterior, para remover la herencia en las clases concretas (*ClaseA1* y *ClaseA2*) se deben integrar los elementos de clase base (*ClaseA*) hacia sus clases derivadas donde sea de utilidad para su comportamiento, que para este ejemplo se analiza el *metodo1()*. Entonces, removiendo la herencia el diagrama de clases quedaría de la siguiente manera:

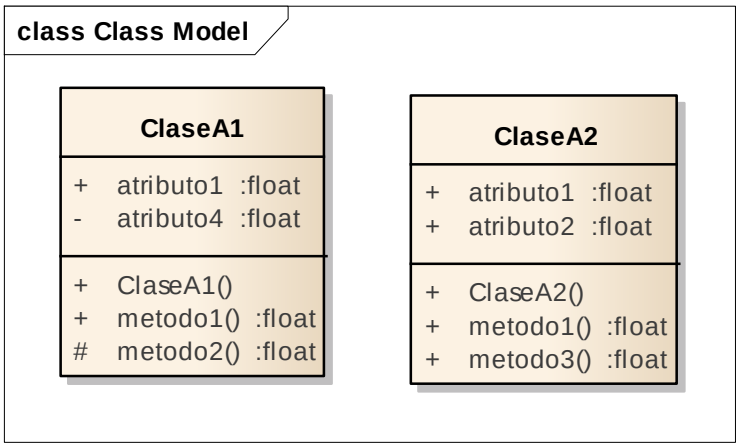

Figura 9. Diagrama de clases para ejemplificar el remover herencia.

Además de integrar todos los miembros utlizados de la clase base y remover la clase abstracta *ClaseA*, en cada una de las clases derivadas se debe remover las sentencia *extends* o en el caso de que la abstracción sea una interface se debe remover la sentencia *implements*, ya que no existe dentro de la lista de clases. Por otro lado, los modificadores de acceso de los elementos integrados hasta el momento se respetan.

2.2.3.2 Incorporar elementos y miembros de caso de uso hacia la clase iniciadora

Para incorporar los elementos y miembros de caso de uso hacia la clase iniciadora (CI) se deben de seguir los siguientes pasos:

2.2.3.2.1 Eliminar ambigüedad e incorporar elementos y miembros de casos de uso

Se debe eliminar cualquier ambigüedad que provoque error de compilación del código para el caso de uso, lo cual afectaría a la ejecución del servicio Web. Se debe analizar el código de las clases que se incorporarán a la CI, mediante un análisis de los nombres de enumeradores, atributos y métodos para que no exista ambigüedad entre nombres de los mismos elementos dentro de la CI. Considerar la figura 10 como el diagrama original de un Marco Orientado a Objetos:

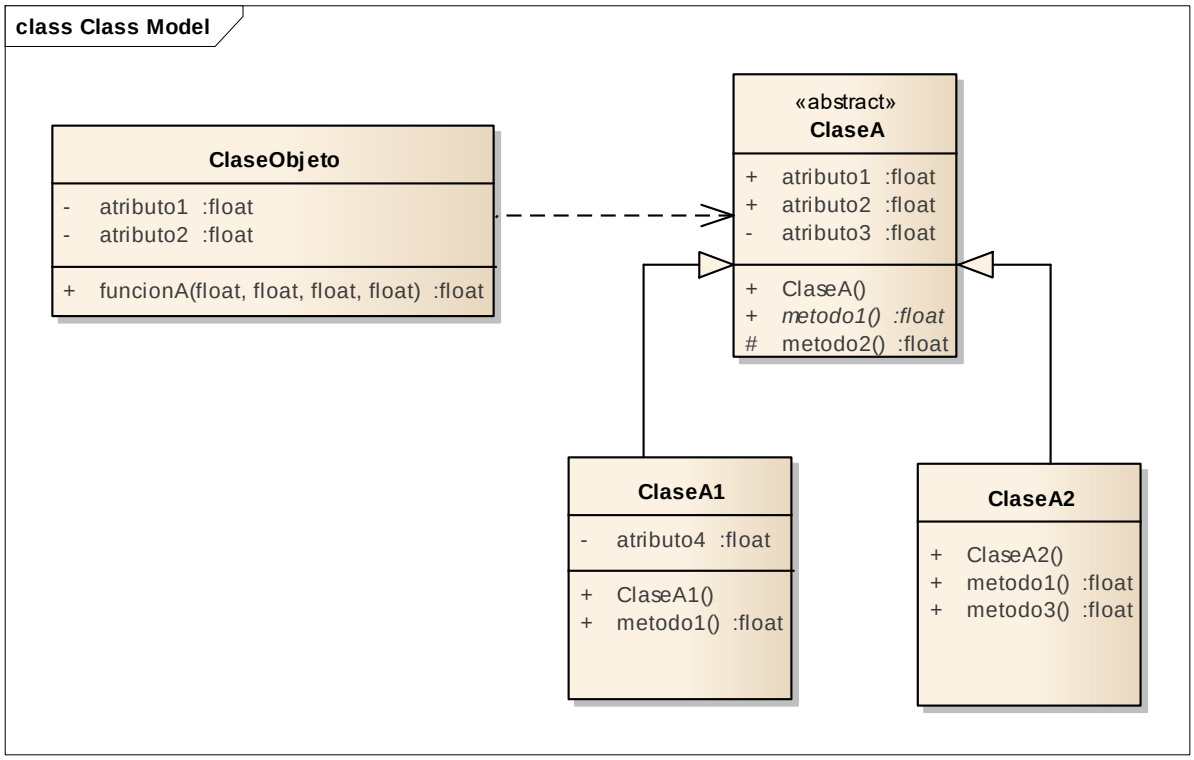

Figura 10. Arquitectura para ejemplificar el proceso de eliminar herencia.

Suponer que el diagrama presentado en la figura anterior, el método iniciador (*MI*) a considerar es *funcionA()* que se encuentra en la *ClaseObjeto*, el cual tiene una relación de dependencia con la *ClaseA*. La *funciónA()* solicita un servicio al *metodo1()* que toma la forma de *ClaseA1.metodo1()* y *ClaseA2.metodo1()*. El código para el método *funcionA()* es el siguiente:

```
public  float funcionA(){
  ClaseA obj;
  Obj = new ClaseA1;
  atributo1 = obj.metodo1();
  obj = new ClaseA2();
  atributo2 = obj.metodo1();
  int retorno = atributo1 > atributo2 ? Math.Pow(atributo1,2) : 0;
  return retorno;
}
```

Aplicando el punto 2.2.3.1 remover herencia, la clase base se eliminaría y la arquitectura quedaría de la siguiente forma:

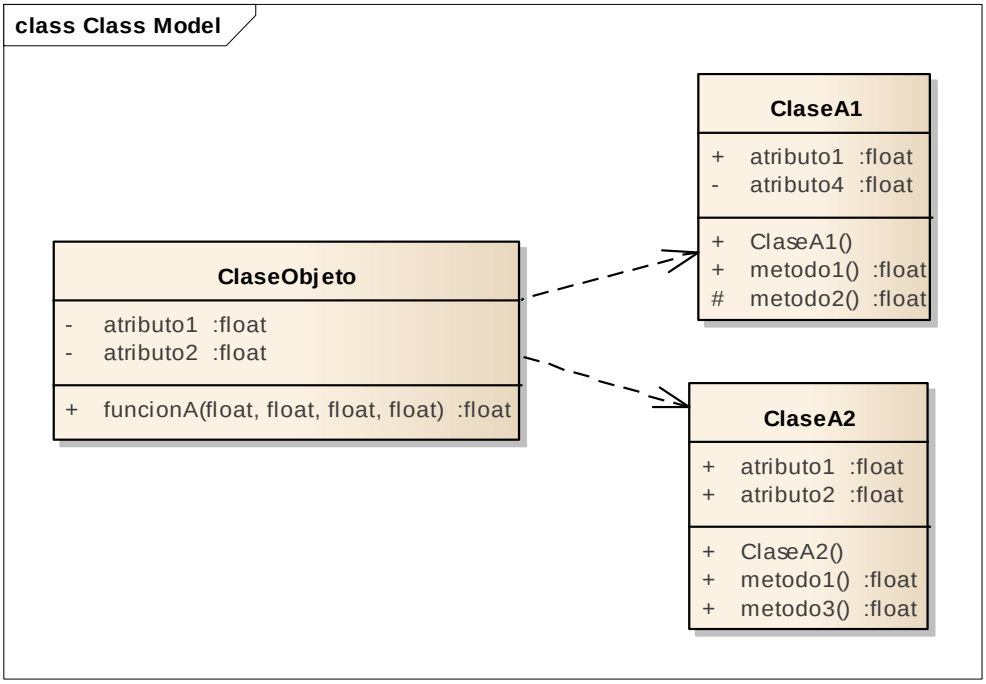

Figura 11. Arquitectura después de aplicar de remover herencia en el caso de uso.

El código correspondiente a este caso de uso no funcionaría de manera correcta, puesto que se ha eliminado la clase abstracta *ClaseA* por medio de la cual la *ClaseObjeto* tenía acceso al *metodo1()* de las clase *ClaseA1* y *ClaseA2* mediante la propiedad de polimorfismo. Por lo tanto, deben incorporarse los elementos y miembros de las clases hacia la *ClaseObjeto*, que para este ejemplo se considera como Clase Iniciador de caso de uso. Además, deben considerarse los siguientes puntos:

1. Evaluar que dentro del bloque de código de los constructores exista alta cohesión con los miembros de la clase.
    a. Si en el constructor existe alta cohesión con los miembros de la clase, entonces convertir el constructor a un método sin retorno (*void*) y con el nombre del tipo dinámico. Finalmente incorporar a la CI.
    b. Si en el constructor no existe alta cohesión con los miembros de la clase o no implementa comportamiento alguno, entonces no incorporar a la CI.
2. Evaluar el nombre de cada miembro a incorporar a la CI antes de integrarlo.
    a. Si el nombre coincide con algún miembro de la CI, entonces renombrar el miembro de la clase como: nombre de la clase + _+ nombre del miembro.
    b. Si el nombre no coincide con algún miembro de la CI, entonces no renombrar el miembro de la clase e incorporarlo a la CI.
3. Evaluar los modificadores de acceso de atributos y métodos de la clase a incorporar.
    a. Si no es privado, entonces cambiar el modificador de acceso a privado e incorporar a la CI.

b. Si es privado, entonces incorporar el miembro directamente a la CI.

Para este ejemplo los constructores: *ClaseA1()* y *ClaseA2()* no implementan ningún comportamiento, es por ello que no serán considerados para integrarlos a la Clase Iniciadora.

Existe una ambigüedad entre los atributos *atributo1* y *atributo2, por lo que* deben renombrarse los atributos de las clases *ClaseA1* y *ClaseA2* antes de incorporarlos a la CI.

Como el método *funcionA()* invoca los métodos *ClaseA1.metodo1()* y *ClaseA2.metodo1(),*debe renombrarse uno de los dos métodos, esto dependerá de cuál método se incorporé primero a partir del orden dentro de la *lista de clases y sus elementos*.

Aplicando el punto (3) se dejó únicamente disponible al Cliente acceso al método iniciador.

Al final la arquitectura después de eliminar ambigüedades es la siguiente:

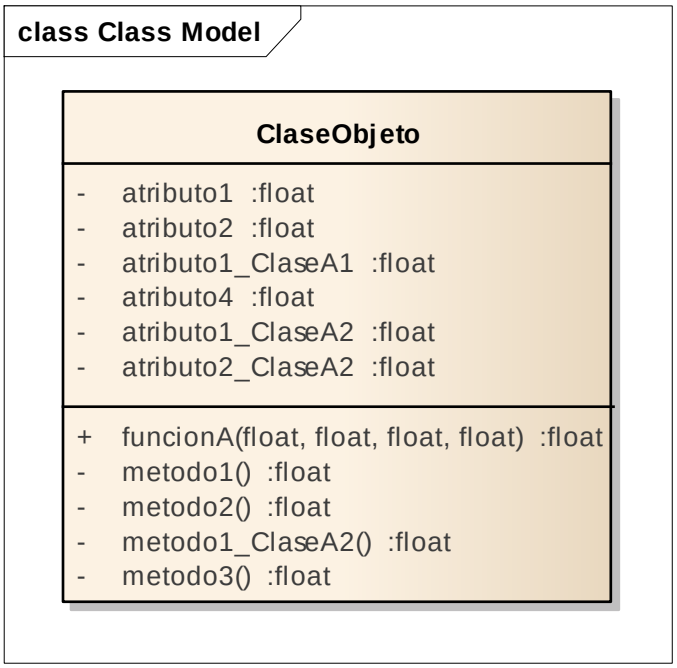

Figura 12. Arquitectura después de incorporar elementos y miembros del caso de uso.

Hasta el momento, todos los elementos y miembros del caso de uso han sido incorporados hacia la Clase Iniciadora, teniendo como resultado toda la funcionalidad del caso de uso en una única clase.

Remover instancias y el ámbito de la clase de los miembros de casos de uso

Un objeto es una variable de un nuevo tipo definido por el programador (no primitivos del lenguaje Java). Los nuevos tipos de objeto definidos por el usuario contienen características (atributos) y comportamientos (métodos). Una instancia es una variable del tipo objeto y un objeto es una instancia de la clase.

Con el propósito de que la incorporación de entidades de software en una única clase sea libre de errores de compilación, se debe analizar el código de cada miembro de caso de uso e identificar instancias de clase de la *lista de clases* del caso de uso, en el cual se hará una depuración del ámbito de instancias para cada miembro de caso de uso.

Para llevar a cabo este análisis de depuración se deberán efectuar los siguientes pasos:

a) Identificar el tipo de cada atributo de la Clase Iniciadora para cada caso de uso, si es del tipo no primitivo (Clase), almacenar el atributo en una *lista de atributos de referencia*.
b) Remover las invocaciones dentro del código de cada método.
c) Remover el ámbito de variables y métodos invocados.

a) Identificar el tipo de cada atributo de la Clase Iniciadora para cada caso de uso

Se identifica el tipo de cada atributo de la Clase Iniciadora para cada caso de uso, si es del tipo no primitivo (Clase), entonces se agregará a una lista denominada *lista de atributos de referencia*.

Dentro de la *lista de atributos de referencia*, buscar el tipo del atributo dentro de la *lista de clases* del caso de uso. Si existe dentro de dicha lista, entonces remover el atributo de la Clase Iniciadora.

b) Remover las invocaciones dentro del bloque de código de cada método.

Se identifican las invocaciones explícitas (creación de instancias de clase con la cláusula *new*) e implícitas que existan dentro del bloque de código de cada método de del caso de uso (incluyendo el método iniciador y constructor). En el caso de que se hagan invocaciones dentro del bloque, se debe de eliminar dicha invocación y reemplazarla por la llamada del método con el nombre equivalente de la invocación de la clase asignado como tipo dinámico aplicado en el punto 2.2.3.2.1. Ejemplo, considerar el siguiente diagrama de clases:

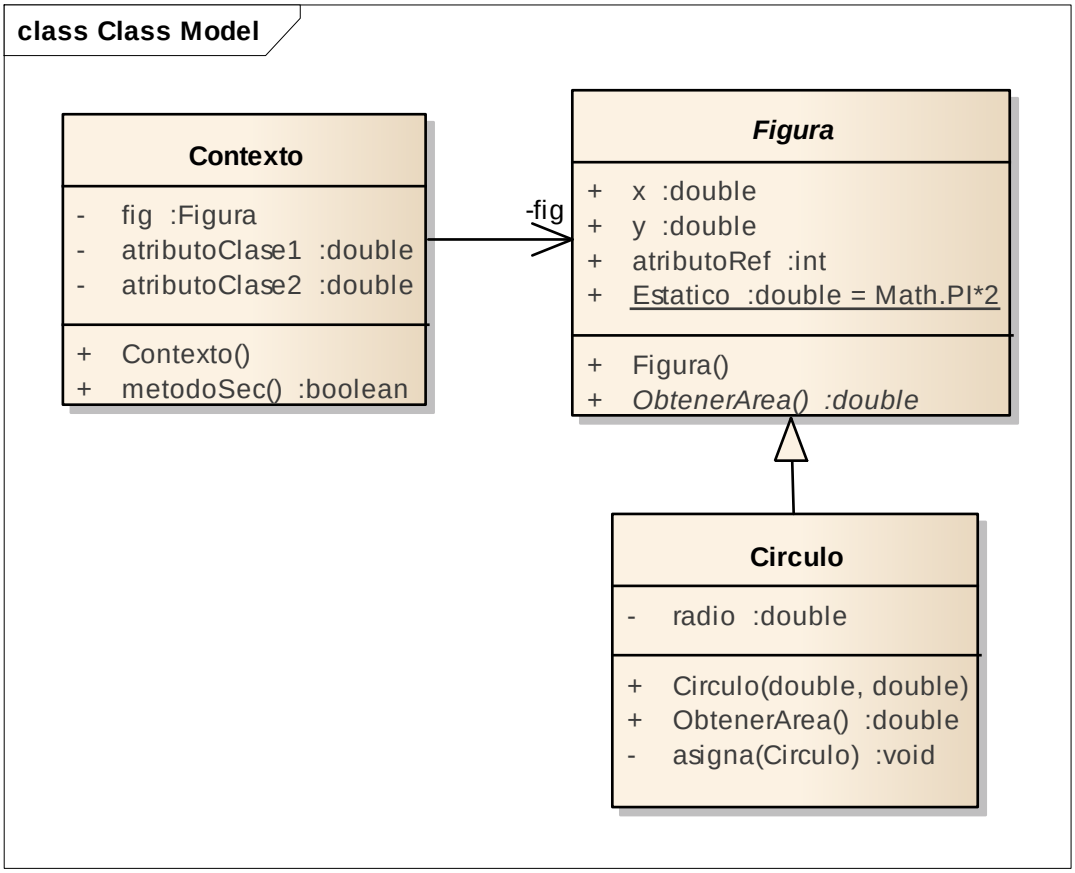

Figura 13. Diagrama de clases para ejemplificar remover invocaciones.

1. Se hace una invocación explícita de la clase figura dentro del bloque de código del constructor perteneciente a la clase Contexto:

```
public Contexto(){
	fig = new Cuadrado(20.5,10.5);
}
```

2. Se aplica el paso b) Identificar las invocaciones dentro del bloque de código del constructor de la CI:

```
public Contexto(){
	Circulo(20.5,10.5);
}
```

c) Remover el ámbito de variables y métodos invocados

Remover todos aquellos calificadores de ámbito en la invocación implícita o explícita de clase que pertenezcan a la *lista de clases* del caso de uso, ignorando aquellos ámbitos que sean invocados mediante la palabra reservada *this* (invocación explícita del constructor) de los miembros de la Clase Iniciadora, que existan dentro del bloque de

código de cada método de del caso de uso (incluyendo el método iniciador y constructor).

Ejemplo, considerar el siguiente bloque de código del método metodoSec de la clase Contexto representada en la figura 13:

```
public boolean metodoSec(){
     //…
 fig = new Circulo(10.5,10.5);
 this.atributoClase1 =  fig.ObtenerArea();
 atributoClase2 = Math.Sqrt(atributoClase1);
 Double resultado = atributoClase1 *atributoClase2 + Figura.Estatico;
 fig.atributoRef = resultado.intValue();
     //…
 }
```

1. Se aplica el paso b) Identificar las invocaciones dentro del bloque de código del constructor del CI:

```
public boolean metodoSec(){
     //…
 Circulo(10.5,10.5);
 this.atributoClase1 =  fig.ObtenerArea();
 atributoClase2 = Math.Sqrt(atributoClase1);
 Double resultado = atributoClase1 *atributoClase2 + Figura.Estatico;
 fig.atributoRef = resultado.intValue();
     //…
  }
```

2. Se aplica el paso c) Remover el ámbito de variables y métodos invocados:

```
public boolean metodoSec(){
     //…
 Circulo(10.5,10.5);
 this.atributoClase1 = ObtenerArea();
 atributoClase2 = Math.Sqrt(atributoClase1);
 Double resultado = atributoClase1 *atributoClase2 + Estatico;
 atributoRef = resultado.intValue();
     //…
 }
```

En el ejemplo anterior se puede apreciar que se remueven todos aquellos ámbitos de invocación clase ignorando aquellos que no pertenecen a la lista de clases del caso de uso. Esto se debe a que ya no existe un acoplamiento entre las clases que participan para satisfacer el caso de uso, mediante el paso 2.2.3.2.1 ahora toda la funcionalidad del caso de uso se encuentra en una única clase; por lo tanto es necesario remover instancias y ámbitos derivados de clase**.**

2.3 Generar los servicios Web

A partir de la incorporación a una única clase de objetos (Clase Iniciadora de caso de uso) de todos los elementos y miembros que lo conforman, del paso anterior, para cada caso de uso se generará un servicio Web cuyo nombre se obtendrá con base en el nombre del método iniciador y el nombre calificado de su clase. Para cada caso de uso se creará un directorio y se colocará en él a los archivos que contengan el código seleccionado para satisfacer el caso de uso. Según los criterios *Generar los servicios Web* en (León, 2009)

Capítulo 3

Funcionamiento de la herramienta

3.1 Especificación del Sistema

El sistema "SR2 Refactoring" (Sistema de Reingeniería para Reuso) da soporte a métodos de refactorización. Actualmente, este sistema implementa tres métodos de refactorización que permiten mejorar la arquitectura de marcos de aplicaciones orientados a objetos, escritos en lenguaje C++. El sistema implementa dos métricas para detectar problemas específicos de diseño en los marcos de aplicaciones orientados a objetos.

El sistema SR2-Refactoring fue implementado en lenguaje Java, utilizando el ambiente Eclipse, y como soporte utiliza el manejador de Base de Datos MySQL.
La información técnica sobre los métodos de refactorización y las métricas se encuentra englobada en tres tesis de maestría del CENIDET, las cuales son:

- Método de Refactorización de Marcos de Aplicaciones Orientados a Objetos por la Separación de Interfaces. (Valdes, 2004)
- Adaptación de Interfaces de Marcos de Aplicaciones Orientados a Objetos por Medio del Patrón de Diseño Adapter. (Santos, 2004)
- Refactorización de Marcos Orientados a Objetos para Reducir el Acoplamiento Aplicando el Patrón de Diseño Mediator. (Cárdenas, 2004)

Sin embargo, este sistema no cubre la transformación de MOO's de su implementación adecuada hacia otros modelos de programa, o cambios en la plataforma de desarrollo como lo es la transformación de un lenguaje a otro. El sistema desarrollado en esta tesis, titulado "SR2-Transforming" (Sistema de Reingeniería para Transformación) aporta en este sentido, organizar el código de MOO's en MSOA, con algoritmos estratégicos para agrupar el código en diferentes niveles de granulación, las cuales son: MOOaSWCU (León, 2009) e Inlineclass (método desarrollado en esta tesis), que es el método de reingeniería desarrollado en esta tesis.

El objetivo del sistema es alcanzar un balance entre las métricas de rendimiento, calidad para reuso y mantenimiento y control de la evolución de los servicios web.

3.2 Análisis del Sistema

Para automatizar la transformación de Marcos de Aplicaciones Orientados a Objetos (MOO) hacia Marcos de Arquitectura Orientada a Servicios, mediante la detección de Casos de uso dentro del código del MOO, se debe de contar con una herramienta que facilite al cliente incorporar el código del MOO que se desea transformar, de tal manera se realice la transformación mediante el enfoque de agrupamiento de casos de uso que el cliente consideré mejor para su objetivo.

El desarrollo de una aplicación Web que comunica el servicio de reingeniería para la transformación de MOO's a MSOA se realizó en los siguientes pasos:

1. Crear una interfaz de usuario para la herramienta que permita elegir el enfoque de agrupamiento y la ruta del código de entrada del Marco. De esta manera, el consumo del servicio web que implementa el sistema de reingeniería será independiente de plataforma y lenguaje de desarrollo.
2. Incorporar la implementación de los dos métodos de transformación (MOOaSWCU e Inlineclass).

A continuación se describe el desarrollo del sistema de reingeniería para la transformación de los métodos antes mencionados:

3.2.1 Diagrama de casos de uso

Para modelar las funcionalidades principales del sistema de reingeniería de esta tesis con la interacción con el usuario, se desarrolló el diagrama de casos de uso que se muestra en la Figura 14, describiendo cada uno de los casos como se muestra a continuación:

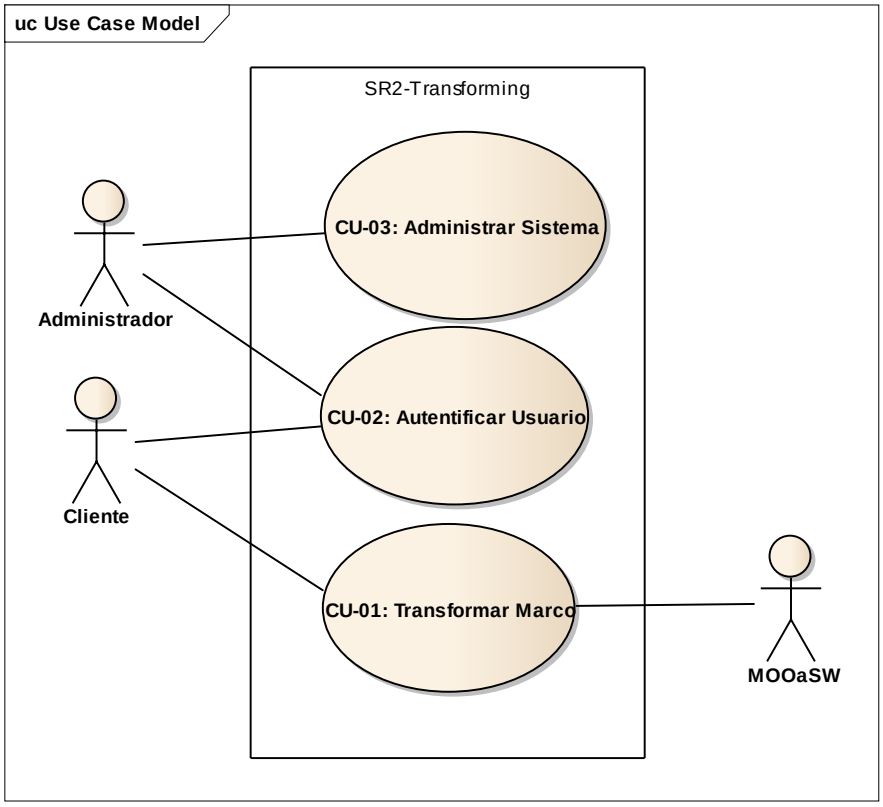

Figura 14. Diagrama de casos de uso del sistema SR2-Transforming

3.2.1.1 Descripción de actores

A continuación se describe cada uno de los actores que participan en el caso de uso del sistema de reingeniería SR2-Transforming, representado en la figura 14:

Tabla 2. Descripción del actor Cliente.

Actor:	**Cliente**
Caso de uso:	Autentificar Usuario, Transformar Marco
Descripción	Representa un cliente externo que utiliza el sistema vía internet

Tabla 3. Descripción del actor Administrador

Actor:	**Administrador**
Caso de uso:	Autentificar Usuario, Administrar Sistema
Descripción	Representa al usuario con privilegios de administrar el sistema para dar de alta, baja o modificación para clientes registrados y catálogos.

Tabla 4. Descripción del actor MOOaSW

Actor:	**MOOaSW**
Caso de uso:	Transformar Marco
Descripción	Es el servicio Web que proporciona la transformación de Marcos Orientados a Objetos hacia Marcos de servicios Web mediante algunos de los enfoques de agrupamiento.

3.2.2 Descripción de casos de uso

CU-01	**Transformar Marco**
Descripción:	El sistema deberá permitir al Cliente subir el código fuente de un Marco Orientado a Objetos para su transformación a MSOA mediante la detección de casos de uso desde el código fuente del MOO, con un enfoque de agrupamiento y componente de reuso que el usuario elija.
Requisitos especiales:	• Contar con servicio de internet • Disponibilidad del servicio Web MOOaSW
Prioridad:	Alta
Actor primario:	Cliente
Actor Secundario:	MOOaSW
Casos de uso relacionados:	CU-02 Autentificar Usuario
Precondición:	1. El cliente debe de estar autentificado en el Sistema (CU-02) 2. El cliente debe de contar con el código fuente del Marco Orientado a Objetos, en lenguaje Java, dentro de un archivo comprimido .zip.

Secuencia normal:	1. El Cliente proporciona el archivo .zip con el código fuente del Marco Orientado a Objetos que desea transformar. 2. El Sistema descomprime el archivo y almacena el código fuente en el repositorio del servidor. 3. El Cliente selecciona el enfoque de agrupamiento y componente de reuso que desea para la transformación. 4. El Sistema envía la petición del servicio al MOOaSW con los parámetros establecidos por el Cliente (enfoque de agrupamiento y componente de reuso). 5. El Sistema visualiza el número de casos de usos detectados y el enlace de descarga del MSOA equivalente al código de entrada al Cliente.
Alternativas:	2a. Si el código de entrada no está en el lenguaje java • El sistema advierte al Cliente que el código de entrada debe estar en lenguaje Java.
Fracaso:	1. El servicio Web MOOaSW no está disponible. Se envía un mensaje al Cliente indicando que la transformación no puede realizarse. 2. El código de entrada del Marco Orientado a Objetos contiene errores sintácticos. Se envía un mensaje al Cliente indicando que la transformación no puede realizarse debido a errores de sintáxis en el código de entrada.
Postcondición:	Como resultado del proceso de salido es el código de MSOA del código de entrada de un Marco Orientado a Objetos a partir del enfoque de agrupamiento y componente de reuso indicado.
Observaciones:	Ninguno

Autor:	Nandi E. Legorreta Ruiz	**Fecha:**	04/12/16
Reviso:	René Santaolaya Salgado		

CU-02	**Autentificar Usuario**
Descripción:	El sistema deberá permitir al Cliente autentificarse para acceder a la interfaz de usuario con las opciones establecidas para usuarios registrados según su rol.
Requisitos especiales:	• Contar con servicio de internet
Prioridad:	Alta
Actor primario:	Cliente, Administrador

Actor Secundario:	Ninguno
Casos de uso relacionados:	CU-01 Transformar Marco
Precondición:	1. El Cliente debe de estar registrado en el Sistema.
Secuencia normal:	1. El Cliente proporciona el nombre de usuario y una contraseña para acceder al Sistema. 2. El sistema valida los datos proporcionados y obtiene el rol asignado para el Cliente. 3. El sistema muestra la interfaz de usuario con las opciones correspondiente según el rol del tipo de Cliente.
Alternativas:	1a. Si el Cliente no se encuentra registrado en el Sistema. • El sistema muestra en formulario de registro. 2a. Si el rol es del tipo Administrador • El sistema muestra la interfaz de usuario con las opciones correspondiente según el rol del tipo Administrador.
Fracaso:	1. Los datos proporcionados no son válidos.
Postcondición:	El Cliente y/o Administrador accede al Sistema.
Observaciones:	Sólo existe un usuario con el rol de administrador, el cual es dado de alta directamente en la Base de datos Local. Por lo cual, el formulario de registro es para usuario con rol tipo Cliente.

Autor:	Nandi E. Legorreta Ruiz	**Fecha:**	04/12/16
Reviso:	René Santaolaya Salgado		

CU-03	**Administrar Sistema**
Descripción:	El sistema deberá permitir al Administrador dar de alta, modificar o eliminar Clientes registrados y catálogos de configuración para el Sistema.
Requisitos especiales:	• Contar con servicio de internet. • Disponibilidad del servicio Web MOOaSW
Prioridad:	Alta
Actor primario:	Administrador
Actor Secundario:	Ninguna
Casos de uso relacionados:	CU-02 Autentificar Usuario
Precondición:	1. El Administrador debe de estar autentificado (CU-02)

Secuencia normal:	1. El Administrador modifica, elimina o agrega un elemento del catálogo de enfoques de agrupamiento para la transformación. 2. El Sistema guarda la configuración establecida.		
Alternativas:	1a. El Administrador modifica, elimina o agrega un elemento del catálogo de componentes de reuso para la transformación. 1b. El Administrador modifica, elimina o agrega parámetros de entrada para el servicio Web MOOaSW.		
Fracaso:	1. El servicio Web MOOaSW no está disponible. El sistema notifica al Administrador que el servicio Web no se encuentra disponible.		
Postcondición:	El Sistema guarda la configuración establecida por el Administrador.		
Observaciones:	Ninguno		
Autor:	Nandi E. Legorreta Ruiz	**Fecha:**	04/12/16
Reviso:	René Santaolaya Salgado		

3.2.3 Diagrama de secuencia

Con base a la especificación de los casos de uso descritas en la sección anterior, se crearon los diagramas de secuencia para cada caso de uso del sistema de reingeniería de esta tesis. De esta manera se identifican los objetos y clases que conforman la arquitectura de clases.

3.2.3.1 Transformar Marco

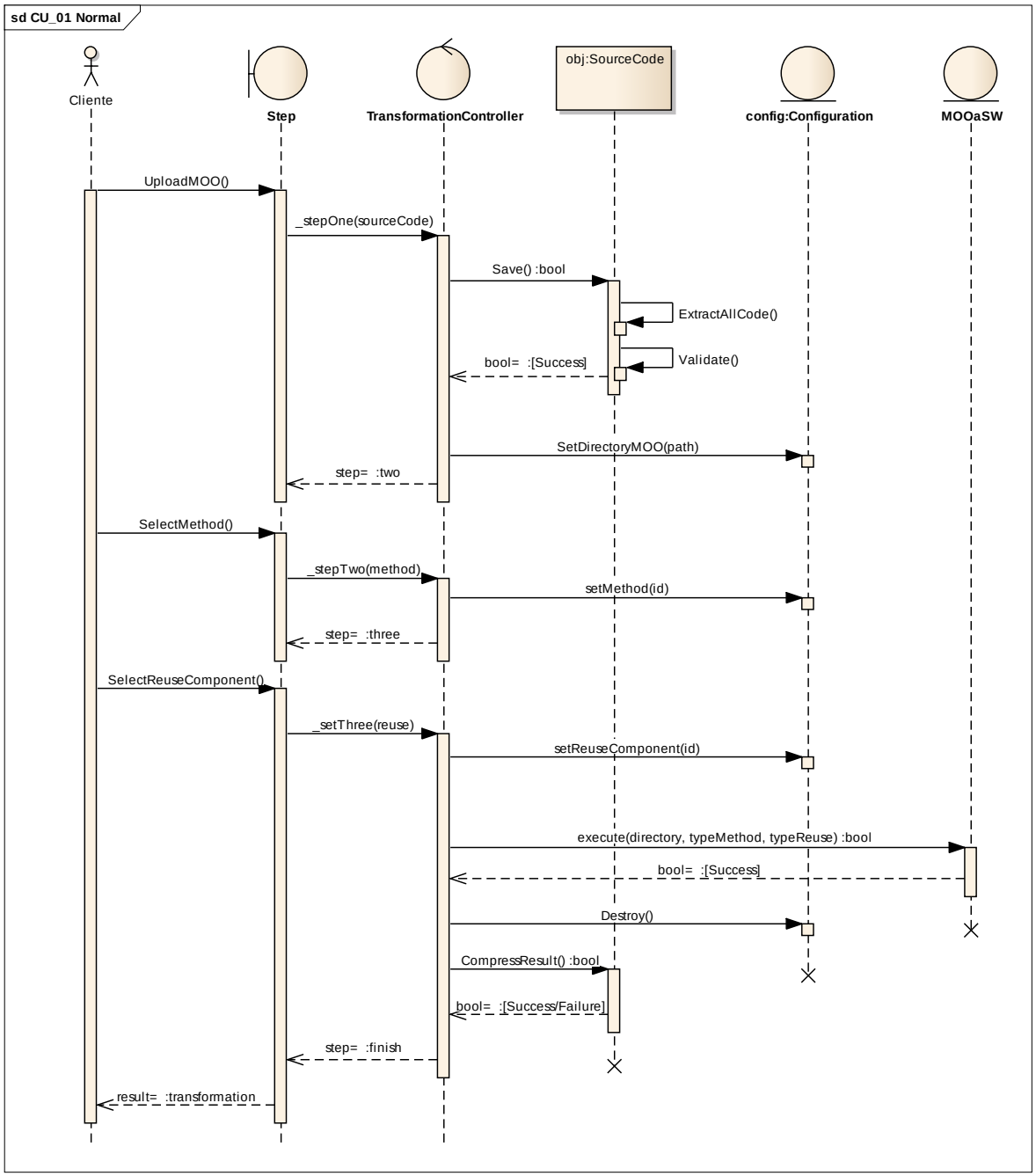

Figura 15. Diagrama de secuencia de la secuencia normal para caso de uso Transformar Marco.

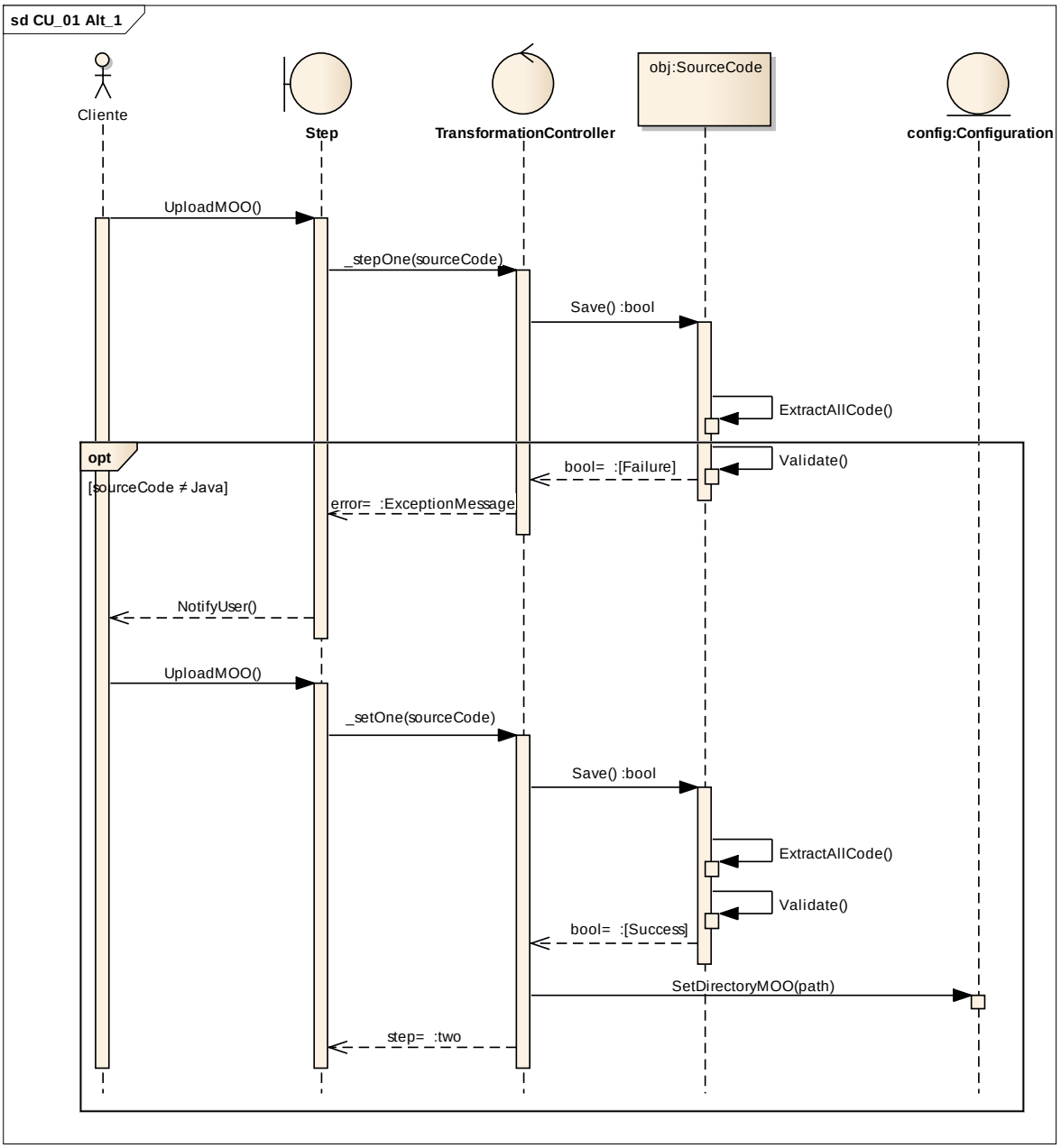

Figura 16. Diagrama de secuencia de la secuencia alternativa para caso de uso Transformar Marco.

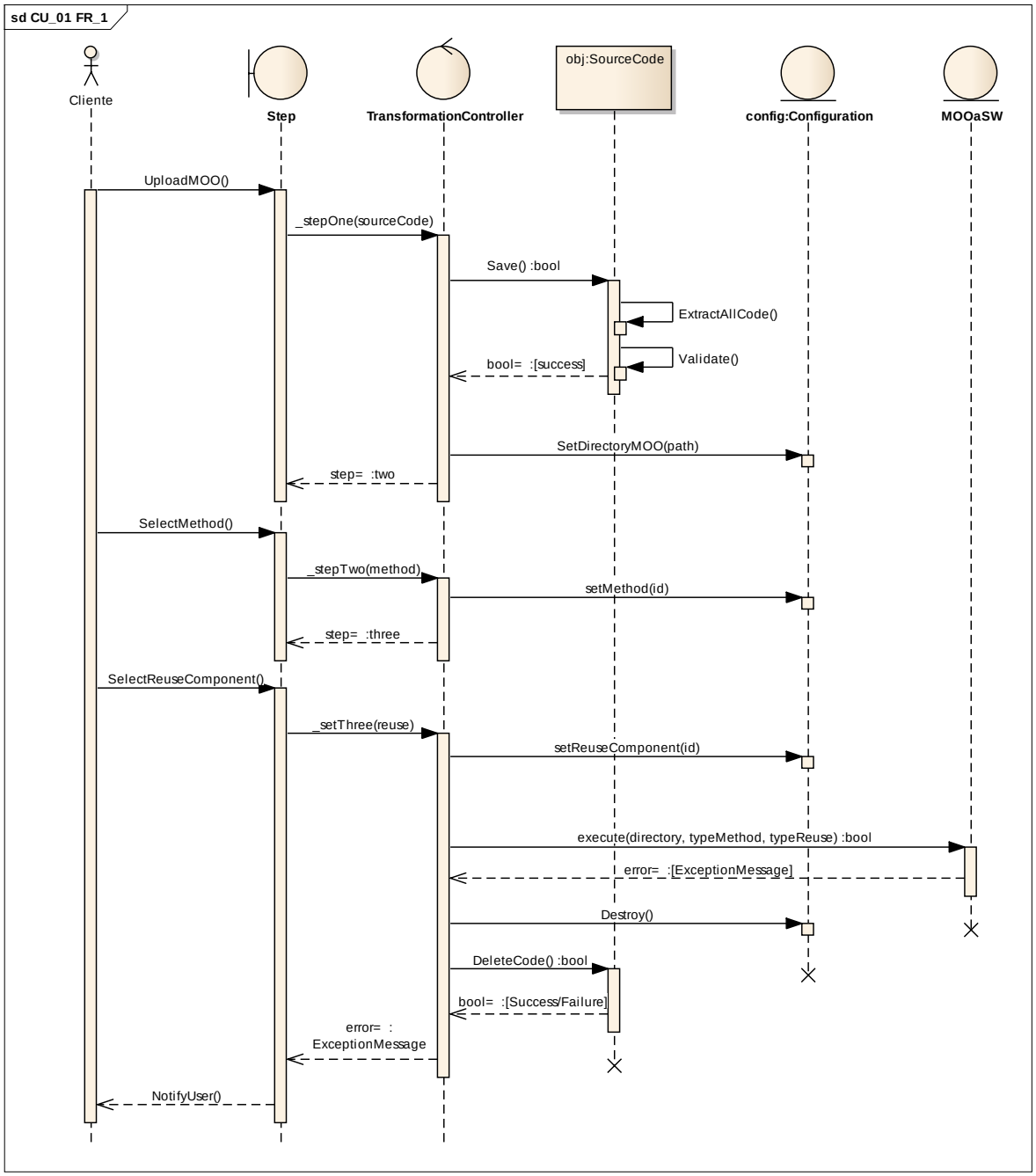

Figura 17. Diagrama de secuencia de la primera secuencia de fracaso para caso de uso Transformar Marco.

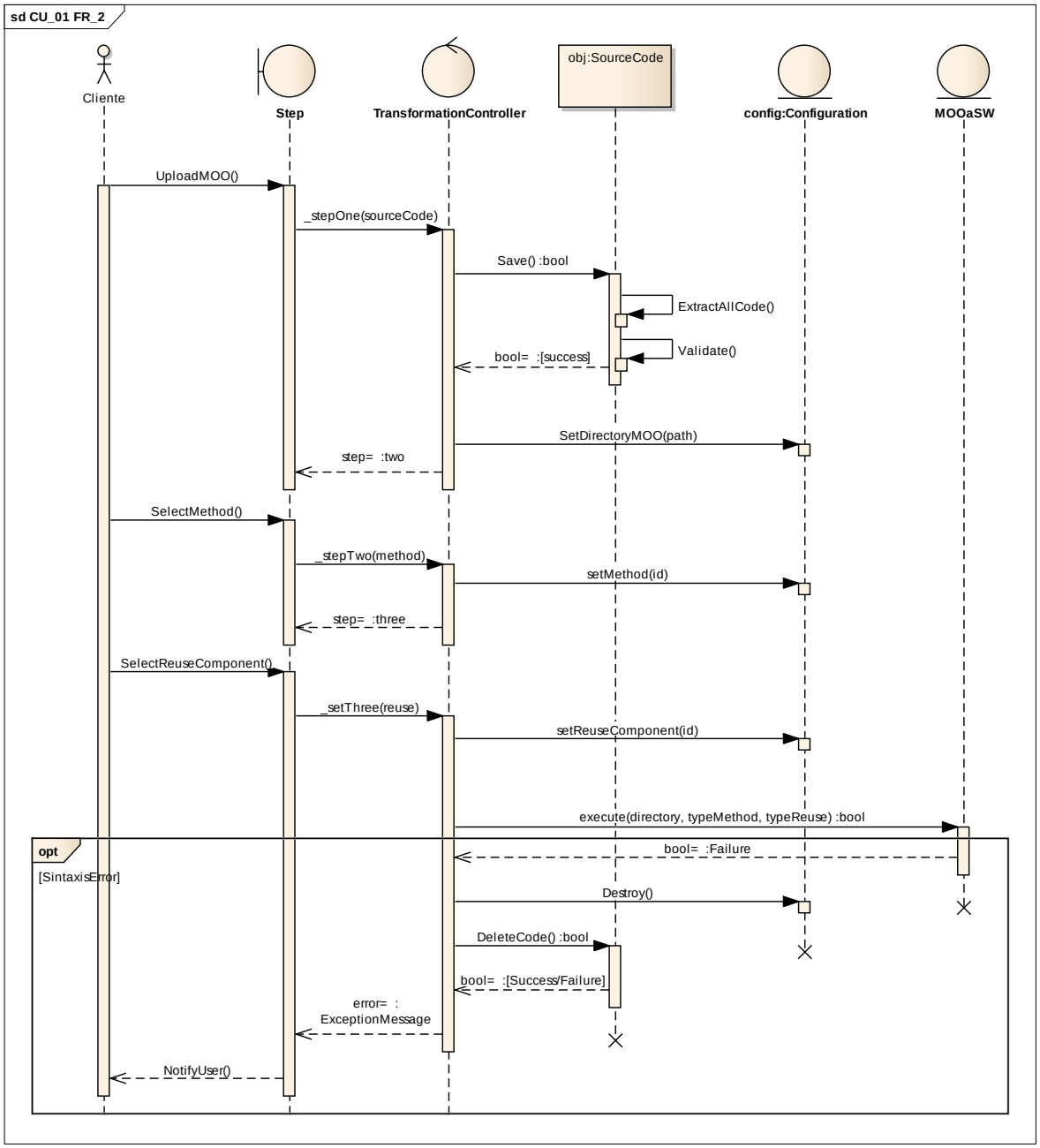

Figura 18. Diagrama de secuencia de la segunda secuencia de fracaso para caso de uso Transformar Marco.

3.2.3.2 Autentificar Usuario

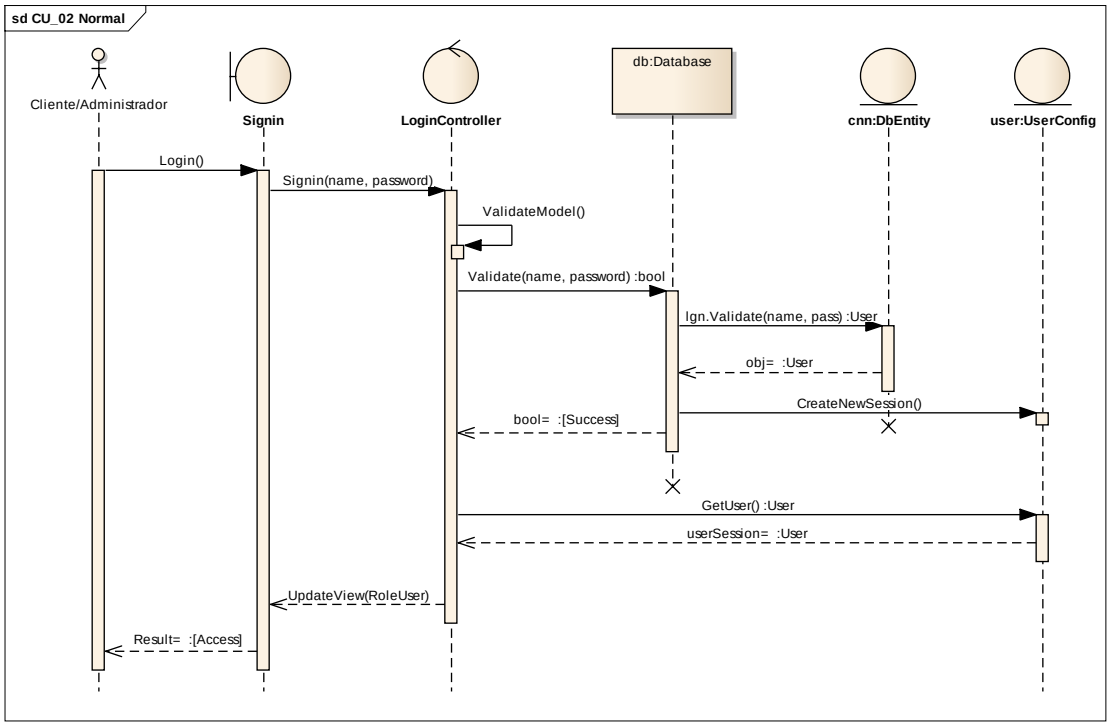

Figura 19. Diagrama de secuencia de la secuencia normal para caso de uso Autentificar Usuario.

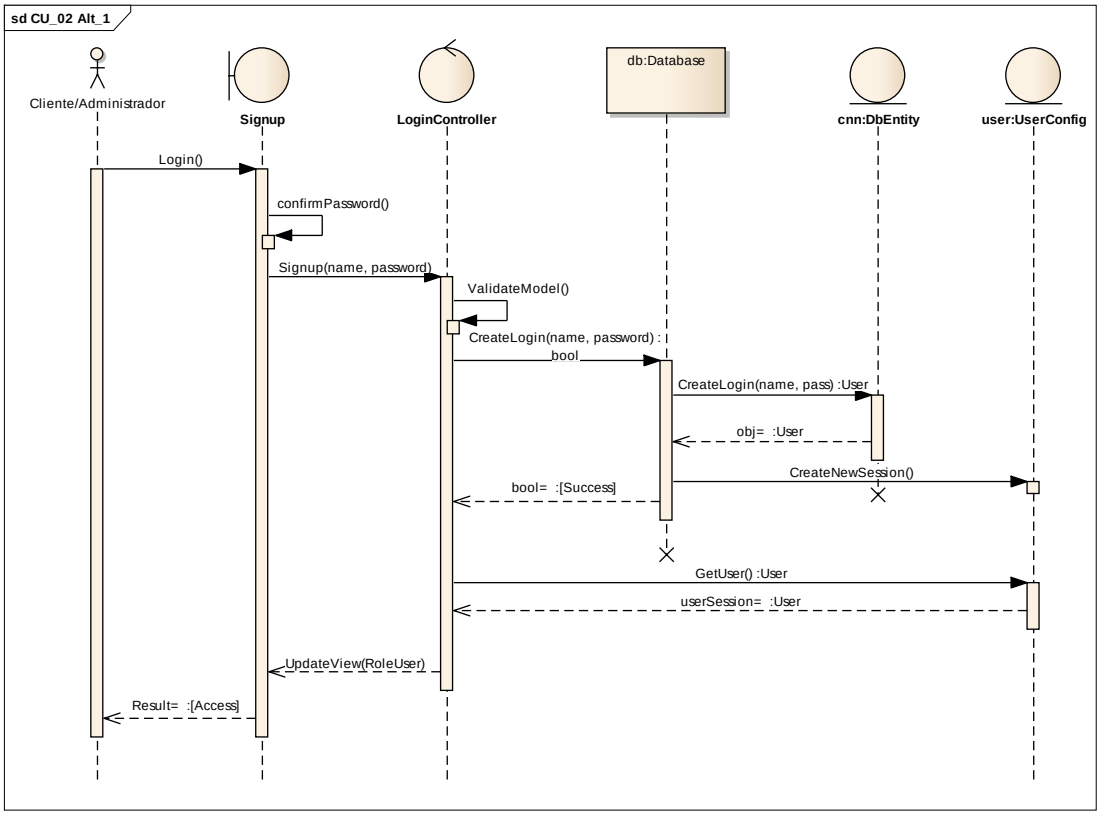

Figura 20. Diagrama de secuencia de la secuencia alternativa para caso de uso Autentificar Usuario.

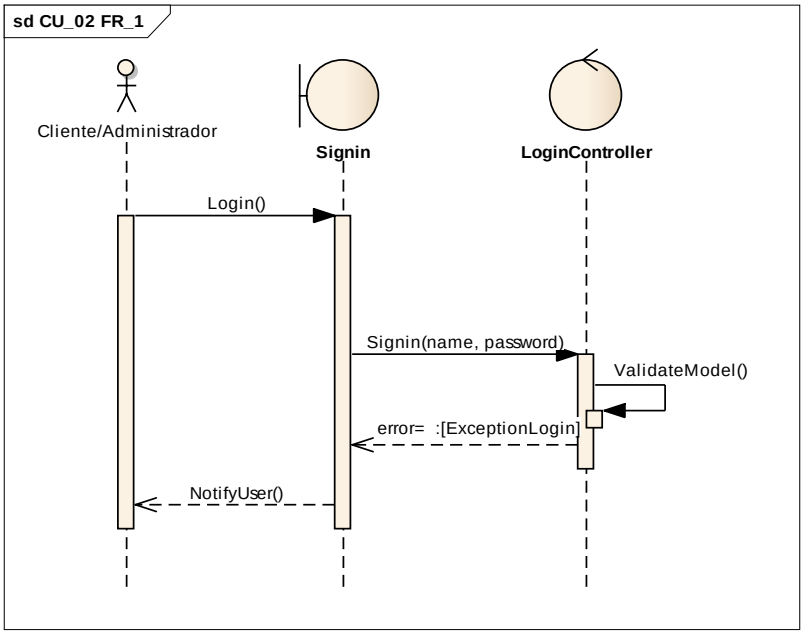

Figura 21. Diagrama de secuencia de la secuencia de fracaso para caso de uso Autentificar Usuario.

3.2.3.3 Administrar Sistema

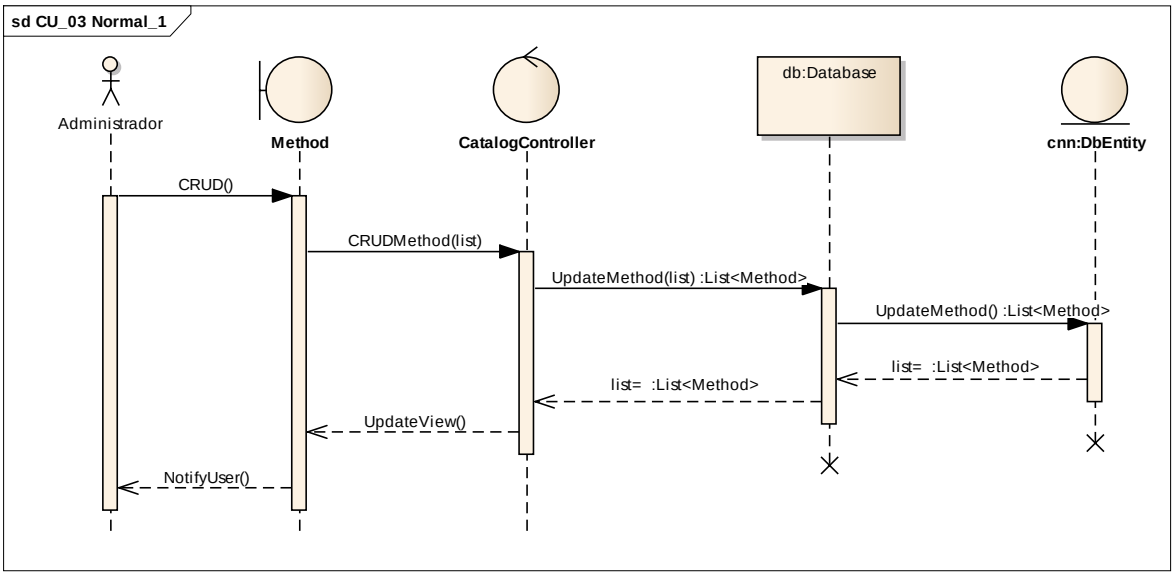

Figura 22. Diagrama de secuencia de la secuencia de normal para caso de uso Administrar Sistema.

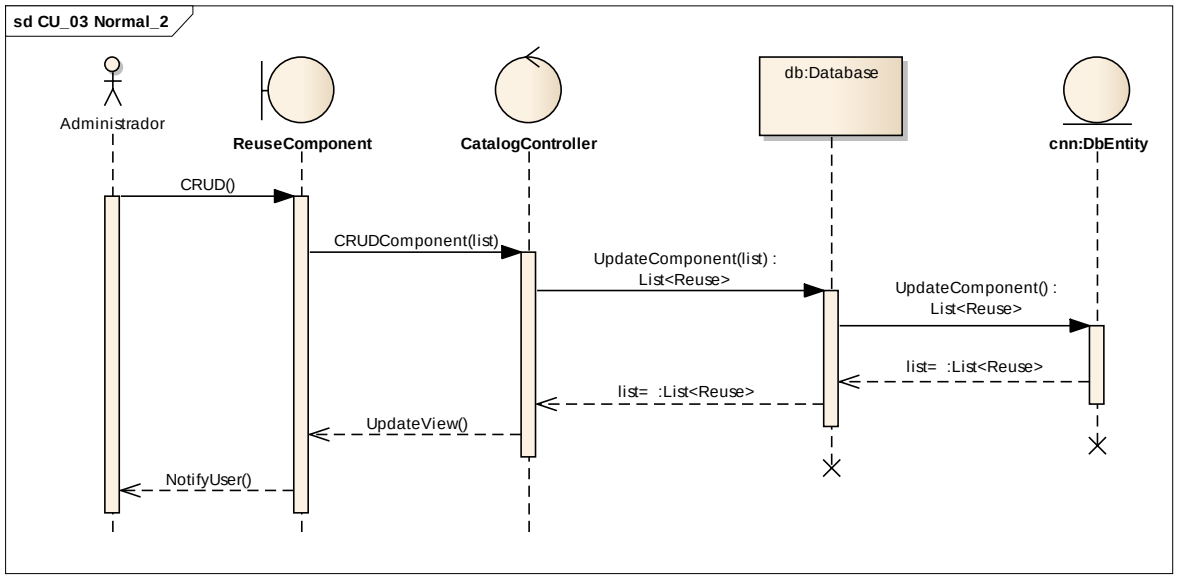

Figura 23. Diagrama de secuencia de la primera secuencia alternativa para caso de uso Administrar Sistema.

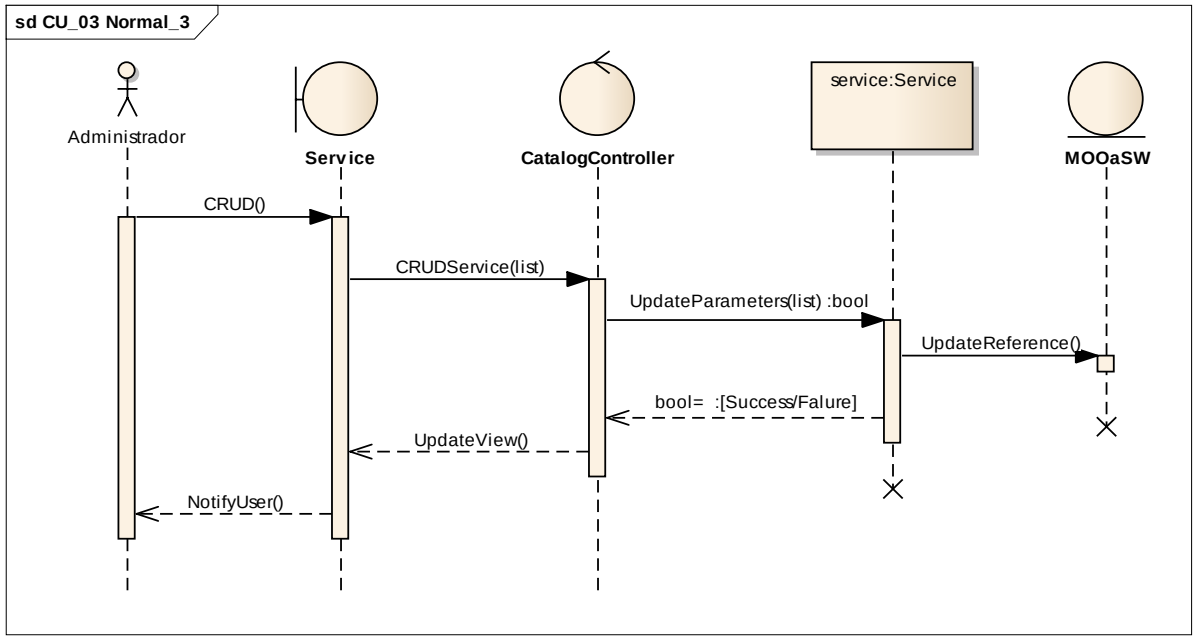

Figura 24. Diagrama de secuencia de la segunda secuencia alternativa para caso de uso Administrar Sistema.

3.3 Diseño del Sistema

3.3.1 Diagrama de clases

Con base a los objetos y clases identificadas en los diagramas de secuencias se diseño el diagrama de clases utilizando el patron de diseño arquitectural Modelo-Vista-Controlador (MVC). La siguiente figura muestra el diagrama de clases para la transformación de Marcos Orientados a Objetos y así satisfaciendo el caso de uso CU_01 Transformar Marco:

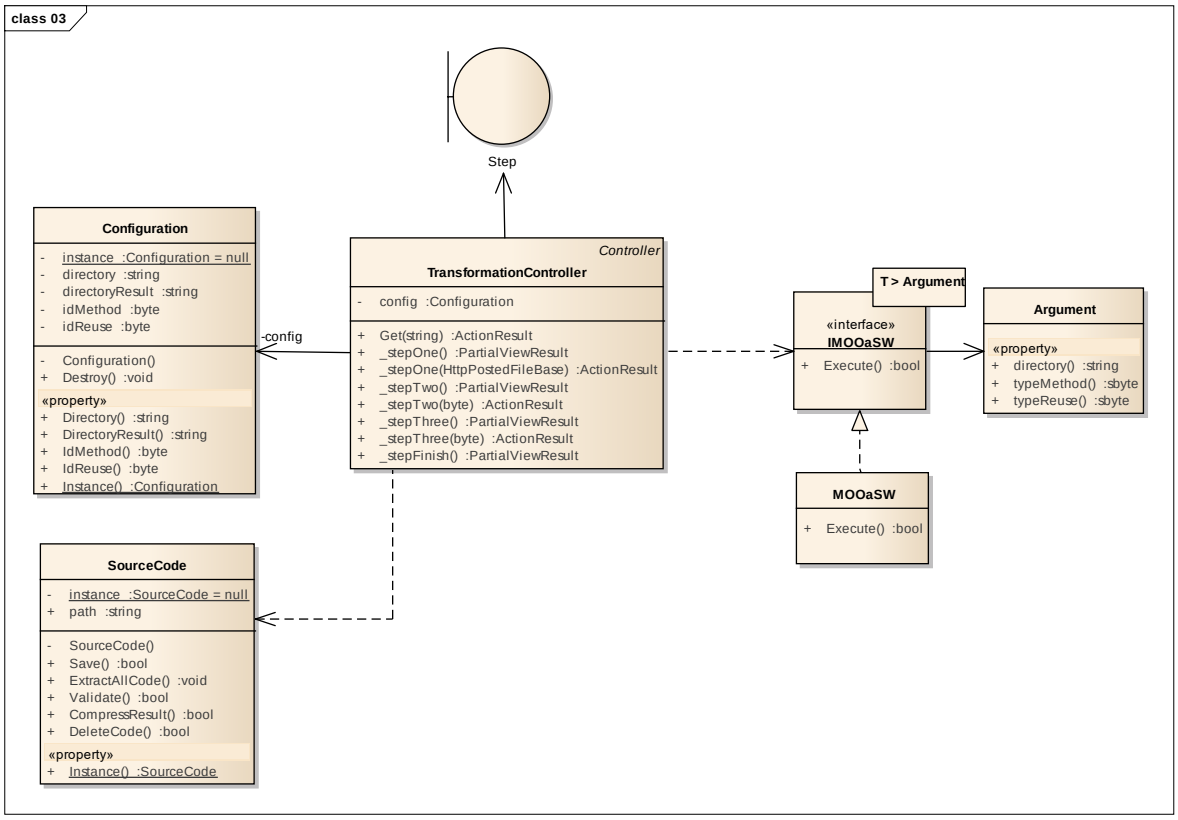

Figura 25. Diagrama de clases para transformar marco.

Se tiene la clase TransformationController que sirve como el controlador que responde a eventos del usuario e invoca peticiones al Modelo. Contiene una asociación directa con la clase Configuration, está clase almacena en memoria toda la configuración establecida por el Cliente mediante variables de Sesión. Además, el controlador tiene una dependencia con la clase SourceCode, donde se tiene el manejo del código de entrada. Tanto para esta clase y la clase Configuration utilizan el patrón de diseño *Singleton*. La clase MOOaSW hereda de la interfaz IMOOaSW que depende de la clase Argument como tipo decorador. Estas clases son las instanciadas para el consumo del servicio Web MOOaSW donde se obtiene los diferentes enfoques de agrupamiento para la transformación del Marco Orientado a Objetos hacia Marco de Servicios Web. Cada uno de los tributos de la clase Argument es utilizado como parámetros de entrada para servicio Web.

En la Figura 26 se representa el diagrama de clases para Autentificar Usuario donde la clase LoginController es utilizada como la capa de controlador, teniendo dependencia con la clase UserConfig utilizada para almacenar en memoria información necesaria del Cliente mediante una variable de Sesión. Además existe una asociación directa con la clase DataBase que a través de la interfaz IRepository se estable comunicación con la base de datos. Esta interfaz utiliza el patrón de diseño *Repository*.

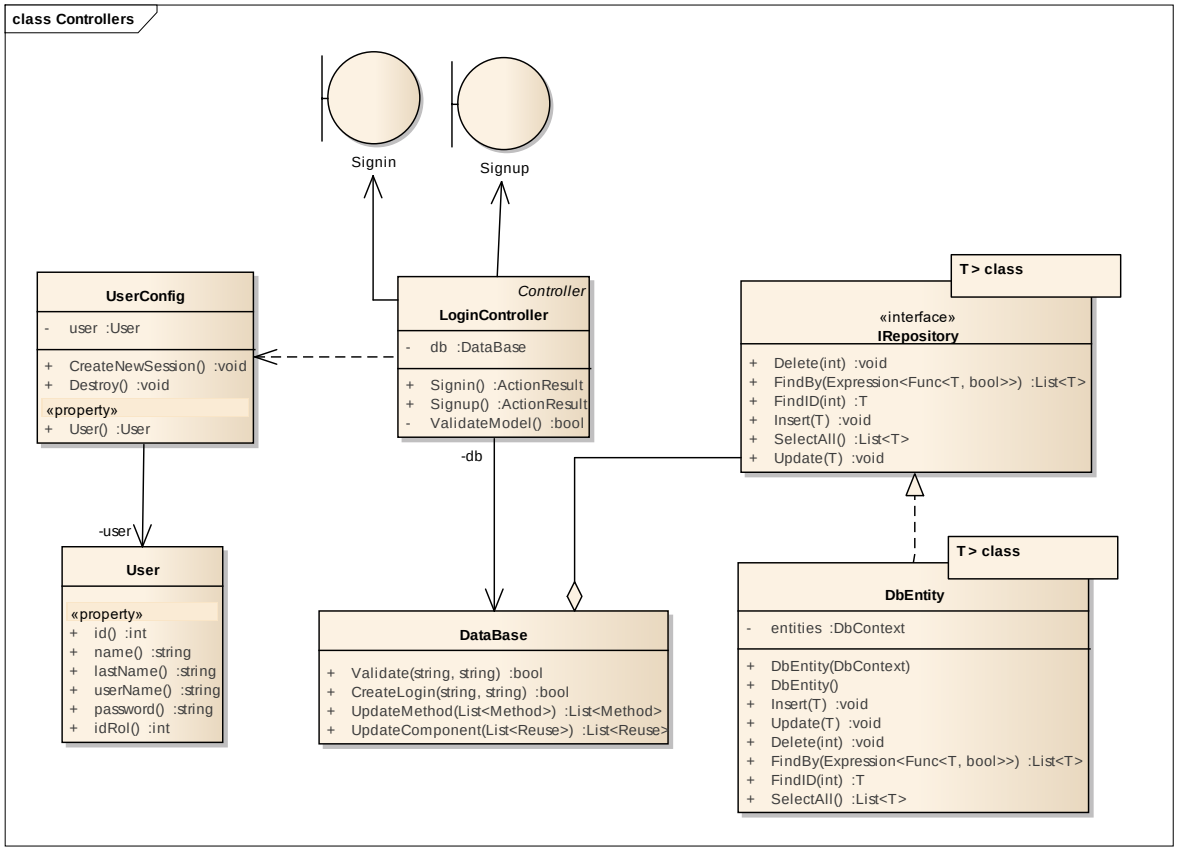

Figura 26. Diagrama de clases para Autentificar Usuario.

La figura 27 muestra el diagrama de clases para Administrar Sistema, la clase controlador CatalogController tiene una asociación con las clases Reuse y Method. A través de esta asociación se estable modificaciones del tipo crear, leer, actualizar y eliminar (*CRUD*) para cada uno de los elementos.

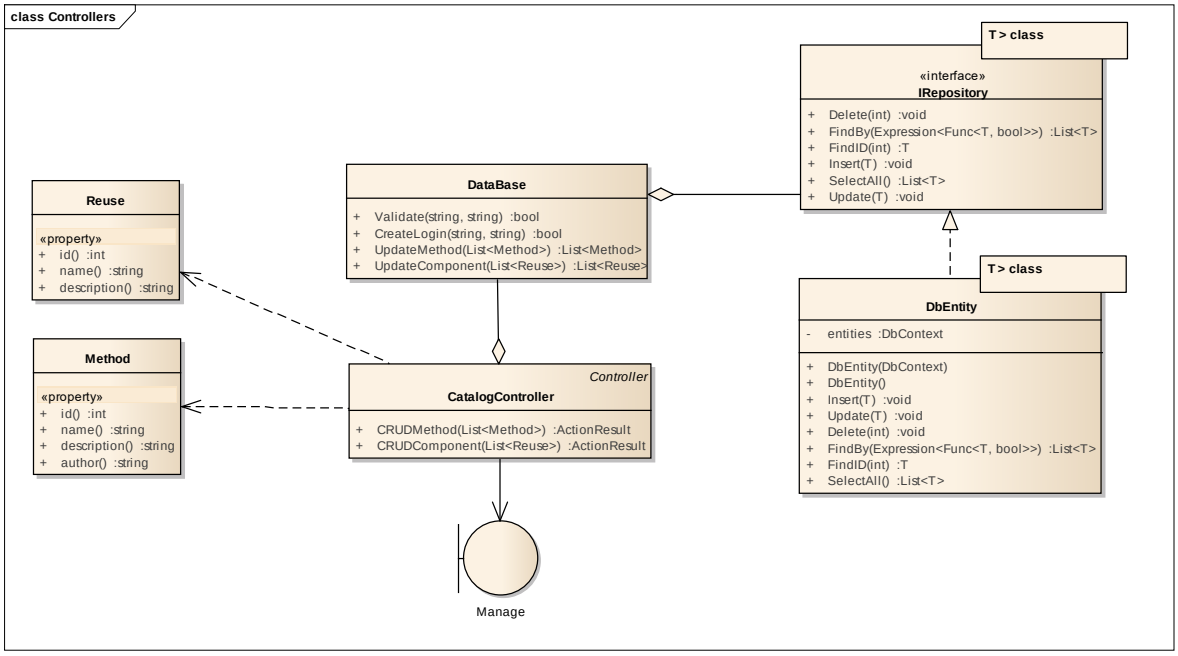

Figura 27. Diagrama de clases para Administrar Sistema.

Todas las clases representadas en las figuras 25,26 y 27 son parte del sistema de reingeniería para esta tesis. A pesar que se utiliza el patrón MVC, no se representa la interfaz de usuario (vista) en los diagramas ya que no existe una regla de negocio para la capa.

3.3.2 Diagrama de actividad

En la figura 28 se representa un diagrama de actividad para el proceso más importante que realiza el sistema de reingeniería.

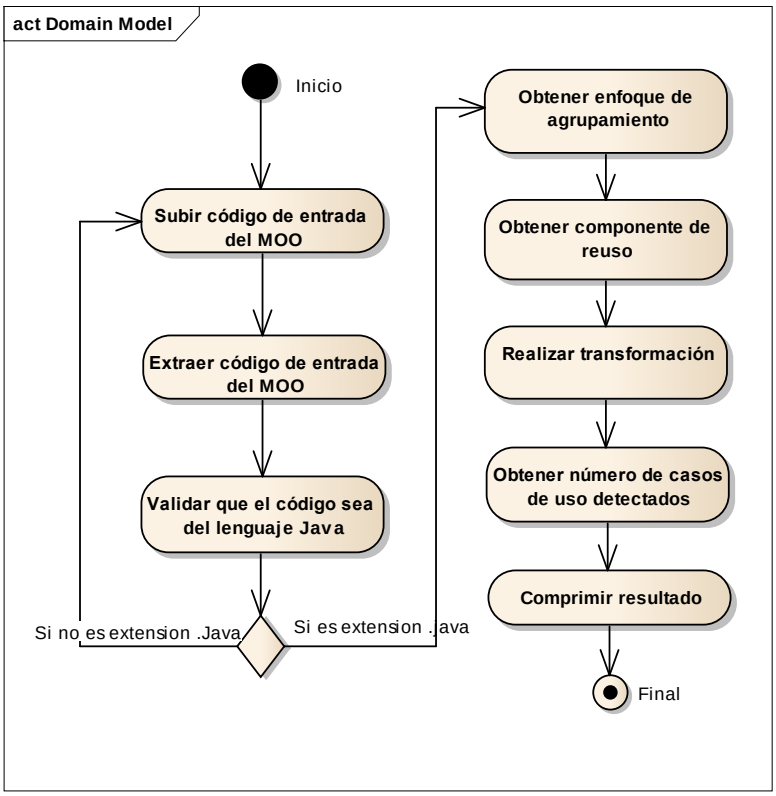

Figura 28. Diagrama de actividad para el proceso de transformar Marco.

Cada una de las actividades se describen a continuación:

- **Subir código de entrada del MOO**: El usuario del sistema mediante la interfaz de usuario (IU) sube el código del Marco Orientado a Objetos que desea transformar. El código del MOO deberá estar comprimido mediante la extensión .zip.

- **Extraer código de entrada del MOO**: Se obtiene el código del MOO mediante la extracción del archivo y se elimina el archivo .zip.

- **Validar que el código sea del lenguaje Java**: Se valida que cada uno de los archivos que el usuario del sistema subió mediante la IU sea del lenguaje Java. Esto se valida mediante la extensión .java, en caso contrario se le hace una petición al usuario del sistema que vuelva a subir el código de entrada el MOO.

- **Obtener enfoque de agrupamiento**: Se obtiene el identificador del enfoque de agrupamiento (método de transformación) que el usuario del sistema seleccionó y se almacena el identificador en memoria para su posterior uso como parámetro de entrada del servicio Web *MOOaSW*.
- **Obtener componente de reuso**: Se obtiene el identificador del componente de reuso (En el caso particular de esta tesis el componente de reuso es servicio Web pero pensando en un trabajo futuro, esto puede cambiar hacia otra tendencia como lo es Microservicio) que el usuario del sistema seleccionó y se almacena el

identificador en memoria para su posterior uso como parámetro de entrada del servicio Web *MOOaSW*.

- **Realizar transformación**: Se invoca el servicio Web MOOaSW mediante los parámetros almacenados en memoria, se espera respuesta de disponibilidad del servicio Web y posteriormente respuesta del servicio Web donde se realizó la transformación de forma satisfactoria.

- **Obtener número de casos de uso detectados**: Se hace una invocación al servicio Web para obtener el número de casos de uso detectados en el código de MOO.

- **Comprimir resultado**: Se comprime el código resultante de la transformación, mediante la extensión .zip para mostrarlo en la IU y el usuario del sistema pueda descargar el código resultante.

Capítulo 4

Evaluación experimental

4.1 Diseño experimental

4.1.1 Generación de hipótesis

Para efectos del presente diseño experimental, fueron establecidas las siguientes hipótesis:

Hipótesis general: El nivel de granulación, en función de la estructura interna de los Servicios Web afecta al tiempo de respuesta.

Hipótesis específica: Las pruebas realizadas bajo diversos niveles de granulación de Servicios Web requieren recursos de memoria, lo que ocasiona sobrecarga al Sistema Operativo e influye de forma directa sobre el tiempo de respuesta.(Guadarrama, 2013)

Hipótesis estadísticas:

- *Hipótesis nula* (H_0): Los tiempos de respuesta entre la arquitectura del Servicio Web generada por el método *Inlineclass* (M_1) y el método *MOOaSWCU* (M_2) son estadísticamente iguales.

- *Hipótesis alternativa* (H_1): El tiempo de respuesta del método *Inlineclass* tiene, estadísticamente, un mejor tiempo de respuesta sobre el método *MOOaSWCU*. $M_1<M_2$

4.1.2 Formulación de un modelo general en términos de importancia

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 las variables intervinientes o factores de ruido. (Guadarrama, 2013)

Variables dependientes

Son las variables de respuesta que se observan en el estudio experimental y que podrían estar influenciadas por los valores de las variables de entrada. Para el presente estudio experimental se definió la siguiente variable dependiente:

- *Tiempo de respuesta*: el objetivo principal a alcanzar es determinar si el tiempo de respuesta de los Servicios Web generados, mejora considerablemente con respecto al método de transformación.

Variables independientes.

Son las variables que pueden cambiar libremente su valor y que no dependen de otra variable. Para el presente estudio experimental se definieron las siguientes variables independientes:

- *Método de transformación*: Los casos de prueba fueron diseñados en función a los Servicios Web generados por los métodos Inlineclass y MOOaSWCU, cuyas estructuras internas, basadas en un diseño orientado a objetos, tienen un nivel de granulación distinta.

- *throughput*: Es una variable de entrada utilizada para estresar cada una de las pruebas experimentales y no depende ni directa ni indirectamente de otra variable externa.

Variables intervinientes o factores de ruido.

- *Factores controlables.* Entre los factores controlables se encuentran las especificaciones técnicas de los equipos de cómputo utilizados, 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, entre otros.

4.1.3 Diseño del Plan de Pruebas

El Plan de Pruebas 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 (Guerrero, 2015).
2. Se establecen las entradas y el valor del *throughput* que recibirán cada uno 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 inferenciales para aceptar o rechazar las hipótesis experimentales y realizar conclusiones sobre el fenómeno de estudio.

4.1.4 Diseño de pruebas

Refinamiento del enfoque: Para realizar las pruebas, se deben crear los servicios Web, aplicando los métodos *Inlineclass* desarrollado e implementado en esta tesis y el método *MOOaSWCU* propuesto por (León, 2009), a cada uno de los casos de uso identificados en los marcos orientados a objetos del Plan de Pruebas de (Guerrero, 2015). Por ejemplo, la figura 14 muestra la arquitectura de clases del Caso de Uso obtenido de la prueba CP45

mediante el método *MOOaSWCU*. Esta arquitectura consiste de un conjunto de clases en colaboración las cuales únicamente contienen los atributos y los métodos que participan en la secuencia interactiva para satisfacer el caso de uso; la figura 15 muestra la arquitectura de clases de la misma prueba CP45 obtenida por el método *Inlineclass* de esta tesis. Esta arquitectura consiste de una única clase, la cual contiene únicamente los atributos y los métodos de la secuencia interactiva que satisface el mismo caso de uso. El diseño de ambas arquitecturas implementa el mismo caso de uso para satisfacer al mismo requerimiento, sólo difieren en el nivel de granulación del servicio Web obtenido apropiadamente a partir de su propia arquitectura de clases.

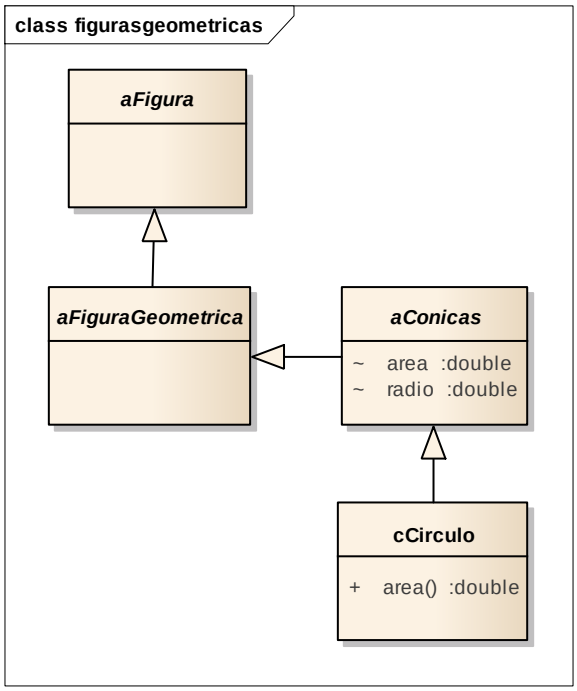

Figura 29. Diagrama de clases del Caso de Prueba CP45 mediante el método MOOaSWCU.

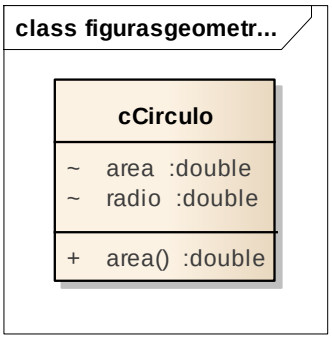

Figura 30. Diagrama de clases del Caso de prueba CP45 mediante el método InlineClass.

Características que se probarán: En primer lugar se probará que la funcionalidad de los casos de uso del marco orientado a objetos de entrada es equivalente al de los servicios Web obtenidos correspondientemente por ambos métodos (*Inlineclass y* MOOaSWCU).

Los Servicios Web creados con el código que genera el método de esta tesis y el sistema que lo implementa deben proporcionar los mismos resultados funcionales que el marco orientado a objetos original y por los servicios web obtenidos por el método MOOaSWCU de (León, 2009).

Así mismo, las pruebas consisten en obtener el tiempo de respuesta de un conjunto de Servicios Web seleccionados, a partir de una muestra aleatoria simple para una población finita, con un nivel de confianza del 90% y un error permitido del 10%; donde la población considerada son 312 Servicios Web, omitiendo aquellos Servicios Web que se implementaron con defectos en el diseño del Marco Orientado a Objetos original o el Servicio no arroja resultados de valor para su evaluación. La muestra obtenida para los casos de prueba fue de 56 Servicios Web. La comparativa de ambos resultados para determinar el mejor tiempo de respuesta, partiendo de las conclusiones de (Guadarrama, 2013) cuyo estudio arrojó que el mejor tiempo de respuesta lo dan aquellas arquitecturas de grano grueso, es decir, toda la funcionalidad requerida para un Caso de Uso, integrada en una única clase. Además, se establece un *throughput* calculado a partir de una muestra aleatoria simple para poblaciones infinitas con un nivel de confianza del 95% y un error permitido del 5%; Dando como resultado 1067 iteraciones por ejecución.

Criterio pasa/falla: Desde la ejecución de cada servicio Web con ambos métodos de transformación se obtendrán como resultado los tiempos de respuesta de cada servicio Web. Si los resultados son mayores a cero, la prueba pasa, de lo contrario, al tener un resultado igual o menos a cero, la funcionalidad del servidor no es la adecuada, en este caso se considera que la prueba fallo.

4.1.5 Especificación de los casos de prueba

Elemento de prueba: Tiempo de respuesta entre los métodos *MOOaSWCU* e *Inlineclass*.

Especificaciones de entrada: Las entradas para las pruebas serán las siguientes:

- 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 (Guerrero, 2015).
- Los archivos *.java* que contienen el código de cada muestra seleccionada. Estos archivos deberán estar en una carpeta con el nombre del Marco original.

La muestra obtenida de 56 Servicios Web considerada como se establece en el diseño de los casos de prueba, se muestra en la siguiente tabla:

Tabla 5. Tabla con los Servicios Web obtenidos de la muestra aleatoria simple del Plan de Pruebas de (Guerrero, 2015)**.**

Caso de prueba	Nombre del Caso de Uso
CP1	calcular_BrayCurtis_espaciovectorial
CP2	calcular_Canberra_espaciovectorial

CP3	calcular_Coseno_espaciovectorial
CP4	calcular_Chebishev_espaciovectorial
CP5	calcular_Dice_coeficientes
CP6	calcular_jaccardCom_coeficientes
CP7	calcular_Tversky_coeficientes
CP8	calcular_cHamming_coeficientes
CP9	obtener_NormaL_estadisticos
CP10	calcular_SokalSneath_coeficientes
CP11	calcular_Hamming_espaciovectorial
CP12	calcular_jaccardDistance_coeficientes
CP13	calcular_Manhattan_espaciovectorial
CP14	calcular_NavarroLee_espaciovectorial
CP15	obtener_Varianza_estadisticos
CP16	generar_EliminarDuplicados_FactorComun
CP17	calcular_SimpleMatchingDistance_coeficientes
CP18	obtener_ProductoPunto_estadisticos
CP19	calcular_JaccardString_coeficientes
CP20	EjecucionExcepcion_EjecucionExcepciones_FactorComun
CP21	obtener_DesviacionEstandar_estadisticos
CP22	calcular_RogerTanimoto_coeficientes
CP23	getSumaxy_SumatoriaXY_estadisticos
CP24	obtener_MediaAritmetica_estadisticos
CP25	calcular_Tanimoto_espaciovectorial
CP26	m1_F21_dummy2
CP27	m3_E211_dummy2
CP28	m3_F2_dummy2
CP29	Calcular_Multiplicacion_basicas
CP30	addToList_context_ctx_statistic
CP31	calcula_cDevY_statistic
CP32	calcula_cGaussJordan_statistic
CP33	calcula_cMultiRegress_statistic
CP34	calcula_cRange_statistic
CP35	ordena_cBuble_statistic
CP36	ordena_cBubleXY_statistic
CP37	ordenacion_List_statistic
CP38	proximo_List_statistic
CP39	setSumax_context_ctx_statistic
CP40	sumCalc_context_ctx_statistic
CP41	tao_cDistributionX2_statistic
CP42	altura_cDecagono_figurasgeometricas
CP43	altura_cTrapecioRectangulo_figurasgeometricas
CP44	apotema_cHexagono_figurasgeometricas
CP45	area_cCirculo_figurasgeometricas
CP46	area_cTrapecioRectangulo_figurasgeometricas
CP47	area_cTrapezoide_figurasgeometricas
CP48	arealat_cDodecaedro_figurasgeometricas
CP49	circircuns_cCirculo_figurasgeometricas
CP50	circircuns_cHexagono_figurasgeometricas
CP51	cirinsc_cHeptagono_figurasgeometricas
CP52	diagonal_cDecagono_figurasgeometricas
CP53	diagonal_cHexagono_figurasgeometricas
CP54	diagonales_cTrapecioRectangulo_figurasgeometricas
CP55	perimetro_cIcosaedro_figurasgeometricas
CP56	volumen_cPrismas_figurasgeometricas

Especificación de salida: La salida correspondiente para cada caso de prueba, será un archivo con extensión *.txt* con los tiempos arrojados por cada método (*MOOaSWCU* e *Inlineclass*) obtenidos desde la herramienta "JMeter" la cuál fue utilizada como

instrumento de medición para el tiempo de respuesta. De tal manera que posteriormente se puedan interpretar estadísticamente para hacer una inferencia hacia todos los Casos de Uso en el Plan de Pruebas de (Guerrero, 2015). Los tiempos obtenidos deberán ser en milisegundos (*ms*).

4.2 Instrumentos de medición aplicados

Instrumento de medición para el tiempo de respuesta.

Para obtener resultados significativos acerca del fenómeno de estudio es necesario usar una herramienta computacional que sirva como instrumento de pruebas para recopilar los resultados de las pruebas experimentales con respecto al tiempo de respuesta. Como idea inicial se planteó utilizar la herramienta desarrollada en (Guadarrama, 2013) para evaluar los tiempos de respuesta de los casos de prueba por cada método de transformación. Sin embargo, la herramienta está fuertemente integrada a los casos de pruebas utilizadas en las pruebas experimentales del estudio experimental del trabajo antes mencionado. Por lo que se decidió utilizar una herramienta computacional de código abierto con el nombre *JMeter*.

JMeter es un proyecto de Apache que puede ser utilizado como una herramienta de pruebas para analizar y medir el desempeño de una variedad de servicios, con énfasis en aplicaciones Web. Donde se pueden establecer pruebas de estrés para Servicios Web con capa SOAP y medir el tiempo de respuesta según el número de *throughput* establecido. La herramienta fue obtenida del sitio de Internet http://jmeter.apache.org/ en la versión 2.6.

La herramienta computacional requiere de los siguientes elementos descritos en la tabla 3:

Tabla 6. Especificación de la infraestructura tecnológica para la ejecución de la herramienta de pruebas.

Nombre	Descripción
Cliente	Es el equipo desde el cual se realizan las peticiones.
Servidor Web	Es el servidor Web el cual se encarga de procesar todas las peticiones enviadas por el cliente
Conexión a internet	Es el enlace de comunicación entre el cliente y el servidor.

Las especificaciones técnicas de los equipos utilizados así como los datos de la red local se describen en las tablas 4 y 5:

Tabla 7. Especificaciones técnicas del cliente y el servidor

	Nombre	Procesador	Memoria RAM
Equipo Cliente	Dell Inspiron 3646	Intel Pentium a 2.67 GHz	2 GB DDR3 a 1600 MHz
Servidor Web	Gateway NV57H	Intel Core i5-2410M a 2.30GHz	6 Gb DDR3 a 1333 MHz

Tabla 8. Datos de red

Datos de red

Nombre	Router TP-LINK
Ancho de banda	Descarga: 9 Mbps, Subida: 1.00 Mbps
Medio de comunicación	Conexión Ethernet mediante cables UTP cat 6
Topología de conexión	Punto a Punto (Cliente-Servidor)

Instrumento para el análisis estadístico.

Los resultados de las pruebas por sí solos no serían capaces de proporcionar información suficiente para identificar patrones de comportamiento o realizar conclusiones mediante una inferencia del fenómeno de estudio. Es por ello que se optó por utilizar una herramienta estadística para analizar los resultados y aceptar o rechazar las hipótesis establecidas en el presente trabajo.

La herramienta computacional utilizada, de la compañía IBM, con el nombre *SPSS Estadistic*, es un programa estadístico que servirá para realizar pruebas de normalidad y, en su caso, análisis estadístico paramétrico o no parámetro de los resultados obtenidos. La herramienta fue obtenida del sitio de Internet:

http://www.ibm.com/analytics/us/en/technology/spss/

Requisitos de software: En la tabla 6, se especifican las herramientas de software utilizadas para el entorno de las pruebas:

Tabla 9. Requisitos de software para la herramienta de pruebas

	Cliente	**Servidor Web**
Sistema Operativo	Windows 7 Home Edition de 32 bits.	Windows 7 Professional Edition de 64 bits.
Entorno Java	JDK EE 6.0 o superior	JDK EE 6.0 o superior con Servidor Apache Tomcat 6.0

Eliminación del ruido experimental

El ruido experimental son factores externos que pueden afectar los resultados de las pruebas experimentales. Antes de iniciar las pruebas experimentales se redujo el ruido experimental mediante el proceso propuesto en (Guadarrama, 2013).

4.3 Análisis estadístico e interpretación de resultados

El análisis de datos es el proceso de inspeccionar, limpiar y transformar datos con el objetivo de resaltar información útil y realizar conclusiones. Una vez finalizadas las pruebas, el análisis de los resultados es de suma importancia para aceptar o rechazar las hipótesis establecidas. (Guadarrama, 2013)

4.3.4 Pruebas de normalidad

Los datos obtenidos por sí solos no proporcionan la suficiente información para realizar conclusiones inferenciales, por ello es necesario realizar el análisis estadístico.

Conocer la distribución de densidad de los resultados experimentales es el primer paso para cumplir con el propósito anterior. Se dice que los resultados experimentales *son normales* cuando su función de densidad tiene una forma acampanada y es simétrica respecto de un determinado parámetro estadístico. (Guadarrama, 2013)

Para verificar la normalidad de cada una de las poblaciones en cada ciclo de pruebas acorde a los resultados obtenidos, se utilizó la prueba de normalidad Kolmogorov-Smirnov (para muestras grandes) con un intervalo de confianza del 95% mediante la herramienta *SPSS Estadistic*. Como resultado de la prueba de normalidad se permite concluir que ninguno de los resultados obtenidos en los casos de prueba proviene de una distribución normal, es decir, son datos *no paramétricos*.

4.3.5 Análisis estadístico

Una vez que se determinó que los resultados de las poblaciones son datos *no paramétricos*, las pruebas no paramétricas a realizar y que resultan adecuadas para un experimento con $k = 2$ poblaciones independientes son la prueba U de Mann Whitney, como alternativa no paramétrica de la prueba t Student para muestras independientes.

Prueba no paramétrica U de Mann Whitney

La prueba U de Mann Whitney es una prueba no paramétrica aplicada a dos muestras independientes. Es, de hecho, la versión no paramétrica de la habitual prueba t de Student. Esta prueba compara las medianas para dos poblaciones independientes.

Los resultados se muestran a continuación, indicando con una "x" los casos que difieren estadísticamente y donde $\mathcal{R}$ es igual al Rango promedio de los resultados obtenidos de cada método de trasformación correspondiente a los caso de prueba, el cual servirá para evaluar el mejor tiempo de respuesta, en el caso de ser aceptada la hipótesis alternativa:

Tabla 10. Resultados de la prueba no paramétrica: U de Mann Whitney

Prueba no paramétrica U de Mann Whitney				
Caso de prueba	**P-Valor**	**Difieren estadísticamente**	**$\mathcal{R}$ Inlineclass**	**$\mathcal{R}$ MOOaSWCU**
CP1	.000	x	998.51	1136.49
CP2	.000	x	1001.76	1133.24
CP3	.000	x	928.50	1206.50
CP4	.193	-	1050.26	1084.74
CP5	.001	x	1022.96	1112.04
CP6	.000	x	1014.03	1120.97
CP7	. 022	x	1036.99	1098.01
CP8	.000	x	954.93	1180.07
CP9	.296	-	1081.36	1053.64
CP10	.000	x	1217.13	917.87
CP11	.130	-	1047.33	1087.67
CP12	.000	x	890.18	1244.82
CP13	. 000	x	997.28	1137.72
CP14	.000	x	770.86	1364.14
CP15	.000	x	915.64	1219.36

CP16	.000	x	796.72	1338.28
CP17	.000	x	935.96	1199.04
CP18	.000	x	1019.61	1115.39
CP19	.000	x	846.81	1288.19
CP20	.000	x	978.21	1156.79
CP21	.000	x	935.49	1199.51
CP22	.000	x	1179.04	955.96
CP23	.000	x	994.56	1140.44
CP24	.000	x	994.00	1141.00
CP25	.000	x	1009.70	1125.30
CP26	.000	x	1570.03	564.97
CP27	.034	x	1095.75	1039.25
CP28	.001	x	1110.10	1024.90
CP29	.000	x	899.13	1235.87
CP30	.028	x	1038.21	1096.79
CP31	.000	x	809.25	1325.75
CP32	.000	x	1329.05	805.95
CP33	.000	x	753.57	1381.43
CP34	.000	x	982.27	1152.73
CP35	.000	x	1146.72	988.28
CP36	.005	x	1102.84	1032.16
CP37	.000	x	1303.86	831.14
CP38	.000	x	1365.39	769.61
CP39	.001	x	1024.60	1110.40
CP40	.000	x	1132.32	1002.68
CP41	.229	-	1083.47	1051.53
CP42	.946	-	1066.60	1068.40
CP43	.000	x	887.19	1247.81
CP44	.323	-	1080.67	1054.33
CP45	.000	x	982.88	1152.12
CP46	.000	x	1176.51	958.49
CP47	.000	x	674.38	1460.62
CP48	.000	x	1387.33	747.67
CP49	.000	x	961.51	1173.49
CP50	.000	x	971.35	1163.65
CP51	.000	x	1216.61	918.39
CP52	.000	x	1138.01	996.99
CP53	.000	x	645.35	1489.65
CP54	.000	x	905.04	1229.96
CP55	.001	x	1024.31	1110.69
CP56	.000	x	808.01	1326.99

Los resultados de la prueba U de Mann Whitney, presentados en la tabla 7, muestran que en el 89.29% de los casos difieren estadísticamente. De esta forma, se rechaza la hipótesis general y se acepta la hipótesis alternativa H_1.

Por último, se realizó un análisis comparativo entre tiempos de respuesta para cada caso de prueba y los métodos de transformación, como se describe en la siguiente sección:

4.3.6 Comparación entre las arquitecturas

La tabla 8, muestra la comparativa entre resultados para cada caso de prueba, a partir de los tiempos obtenidos durante las pruebas experimentales, considerando la arquitectura más eficiente como la arquitectura con menor tiempo de respuesta y la arquitectura menos eficiente como la arquitectura con mayor tiempo de respuesta, donde:

M_1 = Método *Inlineclass*
M_2 = Método *MOOaSWCU*

Tabla 11. Comparativa entre los tiempo de respuesta

Caso de prueba	Arquitectura más eficiente
CP1	M_1
CP2	M_1
CP3	M_1
CP4	-
CP5	M_1
CP6	M_1
CP7	M_1
CP8	M_1
CP9	-
CP10	M_2
CP11	-
CP12	M_1
CP13	M_1
CP14	M_1
CP15	M_1
CP16	M_1
CP17	M_1
CP18	M_1
CP19	M_1
CP20	M_1
CP21	M_1
CP22	M_2
CP23	M_1
CP24	M_1
CP25	M_1
CP26	M_2
CP27	M_2
CP28	M_2
CP29	M_1
CP30	M_1
CP31	M_1
CP32	M_2
CP33	M_1
CP34	M_1
CP35	M_2
CP36	M_2
CP37	M_2
CP38	M_2
CP39	M_1
CP40	M_2
CP41	-
CP42	-
CP43	M_1
CP44	-

CP45	M_1
CP46	M_2
CP47	M_1
CP48	M_2
CP49	M_1
CP50	M_1
CP51	M_2
CP52	M_2
CP53	M_1
CP54	M_1
CP55	M_1
CP56	M_1

En 35 de 56 casos resultó que las arquitecturas internas de los Servicios Web probados cuentan con el menor tiempo de respuesta. En 15 de 56 casos el método M2 obtiene el mejor tiempo de respuesta en función de su arquitectura. En 6 de 56 no se sabe qué sucede.

De estos resultados se deduce que el 82.99% de los Servicios Web obtenidos con el método M1 obtiene el mejor Tiempo de respuesta.

Capítulo 5

Conclusiones y Trabajos a Futuro

5.1 Conclusiones

En este trabajo se propuso hacer la identificación de casos de uso en el código de un marco orientado a objetos mediante un enfoque alternativo de agrupamiento, mediante la agrupación de toda la funcionalidad necesaria para satisfacer cada caso de uso, dentro de una sola clase de objetos. Esto con base en el razonamiento de que un caso de uso se satisface al obtener un resultado de valor del marco a una solicitud funcional del cliente. Por lo tanto, un caso de uso se inicia cuando el cliente invoca un método disponible del marco y se satisface cuando el cliente obtiene el resultado de valor de la ejecución del método.

La finalidad de la realización de este trabajo es reusar la funcionalidad del marco orientado a objetos en la satisfacción de los casos de uso en forma de servicios Web. Además, permitir que el enfoque propuesto tenga un mejor tiempo de respuesta con respecto al enfoque propuesto en (León, 2009) . Esto sustentado en los resultados del estudio de experimental (Guadarrama, 2013), donde concluye que las arquitecturas de grano grueso obtienen un mejor tiempo de respuesta.

Es por ello que se creó un conjunto de pruebas que evalúa el tiempo de respuesta del enfoque propuesto en este trabajo contra el enfoque propuesto en (León, 2009). El resultado fue que el enfoque propuesto en este trabajo obtiene un mejor tiempo de respuesta en un 62.5% de los casos, el enfoque de (León, 2009) obtiene el mejor tiempo de respuesta en un 26.78% de los casos y solo el 10.71% ambos métodos tienen un mismo tiempo de respuesta. Esto puede deberse a la complejidad de los casos de uso, cuando el caso de uso tiende a tener una mayor complejidad, el método presentado en esta tesis (InlineClass) obtiene un mejor tiempo de respuesta.

5.2 Problemas presentados

Las diferentes posibilidades de abstracción y programación de los marcos orientados a objetos, hacen necesario un análisis detallado de todos los escenarios posibles que puedan presentarse. Además, es posible que el marco orientado a objetos no tenga una arquitectura ideal, esto se puede deber a lo siguiente:

- El MOO original no tiene implementada de manera adecuada patrones de diseño y corrompe a los principios de diseño orientado a objetos.
- Se rompen principios orientados a objetos dentro del código del marco orientado a objetos.

Esto conlleva a síntomas en el código del marco orientado a objetos y es por ello que en ocasiones, a pesar que no exista un error sintáctico, algunos casos de uso no tienen implementación alguna.

Otro problema presentado es que algunos casos de uso inician mediante un método iniciador que no devuelve nada (*void*), esto provoca que no se obtenga un resultado para el cliente y para su evaluación.

Por último, durante la detección de la secuencia interactiva, hay métodos, que a pesar que no son invocados, participan en el caso de uso. Estos métodos son aquellos que encapsulan atributos de la clase, métodos conocidos como get y set. Se detectó que, en ocasiones, se inicializan atributos participantes del caso de uso y la falta de estos métodos conlleva a un resultado alternativo o de fracaso del caso de uso.

5.3 Trabajos Futuros

De estos problemas presentados y las áreas de oportunidad descubiertas durante el desarrollo de la tesis surgen nuevas alternativas de trabajo futuras para dar continuidad a la investigación de este trabajo. A continuación, se describen.

5.3.1 Nuevo enfoque de agrupamiento

Un trabajo posible es un nuevo enfoque de agrupamiento que considere las ventajas de agrupar en una única clase, mediante la composición de clases internas donde se consideré medidas de calidad como: bajo acoplamiento, alta cohesión, rendimiento y autosuficiencia.

5.3.2 Establecer un componente de reuso alternativo

Actualmente la transformación para detectar casos de uso dentro del código de un marco orientado a objetos, es mediante servicios Web. Es posible adaptar un nuevo componente de reuso alternativo como micro-servicios que permita al cliente elegir el componente de reuso que le parezca conveniente hacia su objetivo.

5.3.3 Pruebas a los diferentes enfoques de agrupamiento

Un complemento a esta tesis es diseñar un plan de pruebas que además de evaluar el tiempo de respuesta, también evalúe medidas de calidad como: carencia de acoplamiento, cohesión, rendimiento y autosuficiencia. Adicionalmente de permitir medir la complejidad de cada caso de uso detectado. De esta manera, las pruebas serán más completas para la evaluación.

5.3.4 Mejorar el procedimiento

Es importante actualizar el parser de la herramienta para permitir la detección de nuevas características del lenguaje java. Actualmente la herramienta se encuentra limitado a la versión 5 del lenguaje java, por tal motivo algunas expresiones no son reconocidas. Al actualizar el parser de la herramienta permitirá que la detección de casos de uso sea más completa y suficiente para cualquier código de marcos orientados a objetos de entrada.

Referencias

Araiza, E. (2007). *Procedimiento para la Obtención de Marcos de Servicios Web a Partir de Marcos Orientados a Objetos*. Centro Nacional de Investigación y Desarrollo Tecnológico.

Bäumer, D., Gryczan, G., Knoll, R., Lilienthal, C., Riehle, D., & Züllighoven, H. (1997). Framework development for large systems. *Communications of the ACM*, *40*(10), 52–59.

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

Cárdenas, L. A. (2004). *Refactorización de Marcos Orientados a Objetos para Reducir el Acoplamiento Aplicando el Patrón de Diseño Mediator*. Centro Nacional de Investigación y Desarrollo Tecnológico.

Claudia, P., Liliana, M., & Liliana, F. (2011). Recovering Use Case Diagrams from Object Oriented Code: An MDA-based Approach. In *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on* (pp. 737–742).

de Páez, R. A. (2012). Un acercamiento a la reutilización en ingeniería de software. *Revista Universidad EAFIT*, *35*(114), 51–63.

Dugerdil, P., & Sennhauser, D. (2013). Dynamic Decision Tree for Legacy Use-case Recovery. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing* (pp. 1284–1291). New York, NY, USA: ACM. http://doi.org/10.1145/2480362.2480602

Freeman, C., & Soete, L. (1997). *The economics of industrial innovation*. Psychology Press.

Froehlich, G., Hoover, J., Liu, L., & Sorenson, P. (1998). Designing object-oriented frameworks. *University of Alberta, Canada*.

García, F., Barras, J., Laguna, M., & Marqués, J. (2002). Líneas de productos, componentes, frameworks y mecanos. *Informe Técnico, Departamento de Informática. Universidad de Salamanca*.

Gesser, J. V. (2008). javaparser. Retrieved November 20, 2015, from https://code.google.com/p/javaparser/

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

Guadarrama, L. C. (2013). *Estudio Experimental para determinar la relación entre la estructura interna de Servicios Web y valores de QoS*. Centro Nacional de Investigación y Desarrollo Tecnológico.

Guerrero, M. (2015). *Plan de Pruebas para la Evaluación Experimental del Sistema MOOaSW*. Centro Nacional de Investigación y Desarrollo Tecnológico.

Jacobson, I., Griss, M., & Jonsson, P. (1997). *Software Reuse: Architecture, Process and Organization for Business Success*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Joyanes Aguilar, L. (1998). *Programación orientada a objetos*. *2da. Edición, Madrid-España, Editorial McGraw-HILL*.

Kohlborn, T., Korthaus, A., Chan, T., & Rosemann, M. (2009). Identification and analysis of business and software services—a consolidated approach. *Services Computing, IEEE Transactions on*, *2*(1), 50–64.

Krueger, C. W. (1992). Software reuse. *ACM Computing Surveys (CSUR)*, *24*(2), 131–183.

Lee, J. K., Jung, S. J., Kim, S. D., Jang, W. H., & Ham, D. H. (2001). Component identification method with coupling and cohesion. In *Software Engineering Conference, 2001. APSEC 2001. Eighth Asia-Pacific* (pp. 79–86).

León, F. (2009). *Generación de servicios Web desde marcos orientados a objetos a partir de clases colaborativas en casos de uso*. Centro Nacional de Investigación y Desarrollo Tecnológico.

Liu, W., Fu, M., Luo, S., & Zou, D. (2013). Use case-based service-oriented analysis and modeling. In *Advanced Technology in Teaching* (pp. 801–806). Springer.

Martín, A. R., & Martín, M. J. R. (2014). Aplicaciones Web. Ediciones Paraninfo, S.A.

McIlroy, M. (1969). Mass produced software components: Software engineering concepts and techniques. In *Proceedings of NATO Conference on Software Engineering* (pp. 88–98).

Misic, V. B. (2001). Cohesion is structural, coherence is functional: different views,different measures. *Proceedings Seventh International Software Metrics Symposium*, 135–144. http://doi.org/10.1109/METRIC.2001.915522

Mohamemed, F. Douglas, S. (2006). Object-Oriented Application Frameworks. Retrieved April 19, 2015, from http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html

Moreno, J. A., Terrén, D. R., & Jiménez, J. B. (2007). *Programación orientada a objetos*. UOC.

Nakkach, H., Kraiem, N., & Ben Ghezala, H. (2006). Identification of the Web Services starting from the existing Web applications. In *Innovations in Information Technology, 2006* (pp. 1–7).

Object Management Group, Version, M. D. a G., Kennedy, A., Carter, K., & Technologies, W. F. X. (2003). MDA Guide Version 1.0.1. *Object Management Group*, *234*(June), 51. http://doi.org/10.1074/jbc.M312687200

Opdyke, W. F. (1992). *Refactoring Object-Oriented Frameworks*. thesis, University of Illinois at Urbana-Champaign.

Santaolaya, R. (2003). *Modelo de representación de patrones de código para la construcción de componentes reusables*. IPN.

Santos, L. E. (2004). *Adaptación de Interfaces de Marcos de Aplicaciones Orientados a Objetos Utilizando el Patrón de Diseño Adapter*. Centro Nacional de Investigación y Desarrollo Tecnológico.

Valdes, M. A. (2004). *Método de Refactorización de Marcos de Aplicaciones Orientados a Objetos por la Separación de Interfaces*. Centro Nacional de Investigación y Desarrollo Tecnológico.

W3C. (2011). Web Services Activity. Retrieved May 31, 2015, from http://www.w3.org/2002/ws/

Yousef, R., Adwan, O., & Abushariah, M. A. M. (2014). Extracting SOA Candidate Software Services from an Organization's Object Oriented Models. *Journal of Software Engineering and Applications*, *7*(9), 770.

Zamudio, S. A. (2013). *Migración de Marcos Orientados a Objetos hacia Marcos de Servicios Web*. Centro Nacional de Investigación y Desarrollo Tecnológico.

Anexos

Anexo 1. Marco teórico

Clase: Una clase es una colección de objetos con una estructura común, comportamiento común, relaciones comunes, y semántica común. Los objetos son entidades que participan en el dominio de un problema de software.

Modularidad: Es la especificación de las fronteras de alcance de una entidad de software (función, clase, paquete, subsistema, sistema), protegiendo el acceso a los detalles internos de representación, mediante el encapsulado y las reglas de visibilidad. Como lo indica Liskov, modularizar consiste en dividir un programa en módulos que se puedan compilar por separado, pero que tienen conexiones con otros módulos. (Joyanes Aguilar, 1998)

En el contexto de este trabajo se definieron cinco niveles de granulación modular: Los primeros candidatos a formar módulos son las funciones y los procedimientos que forman parte de sistemas de software, su meta de valor para el usuario es la realización de una operación única por cada función. En el siguiente nivel de granulación, están las clases de objetos que son agrupaciones de funciones y tienen por meta de valor para el usuario modelar entidades del mundo real en contextos particulares. En el tercer nivel de granulación existen los paquetes o componentes, dependiendo del modelo de programación o infraestructura tecnológica utilizada. En este nivel los módulos están compuestos por comunidades o arquitecturas de clases de objetos y su meta de valor para el usuario es la satisfacción de requerimientos únicos por cada módulo de programa. En el cuarto y quinto nivel de granulación están los subsistemas y sistemas completos, vistos como servicios ofertados en la plataforma de la nube computacional. Los subsistemas agrupan paquetes o componentes, componentes para satisfacer procesos de negocios, a su vez los sistemas agrupan servicios que satisfacen los subsistemas para la administración de un negocio como meta de valor para el usuario.

En el trabajo de Luis César Guadarrama (Guadarrama, 2013) se demuestra que módulos a nivel de paquete logran un balance entre estos atributos de calidad. Por esta razón en el trabajo de Francisco León Pérez (León, 2009) se recomienda que los servicios web tengan el nivel de granulación de paquete o componente de programa porque éstos son suficientes, completos, autónomos para satisfacer requerimientos a la vez que se gana en coherencia, cohesión y acoplamiento interno y externo. Dependiendo del nivel de granulación modular, se pueden tener algunas ventajas y desventajas, por ejemplo, a nivel de funciones se tiene ventaja en rendimiento, pero se pierde en cohesión y coherencia, mientras que a nivel de subsistema o sistema se gana en coherencia, cohesión y acoplamiento, pero se pierde en rendimiento.

Los servicios web son entidades de software que envuelven a módulos de programa en cualquiera de sus niveles de granulación.

Las unidades de encapsulamiento con instancias virtualizadas de Servicios Web y la protección a los detalles internos de representación por medio de una capa XML que envuelve a los Servicios Web. Esto se conoce como formato binario.

Interfaces de Comunicación: Se refiere al número de canales de comunicación entre los módulos del sistema, en la arquitectura de un sistema de software. Cada módulo deberá comunicarse con el menor número posible de módulos. Si un sistema está compuesto de "n" módulos, entonces el número de conexiones inter-módulos debe permanecer lo más cercano al mínimo "n-1", que al máximo "n(n-1)/2".

Siempre que dos módulos "A" y "B" estén relacionados, la comunicación debe ser establecida explícitamente en el texto (acoplamiento directo), es decir, evitar el acoplamiento a través de datos globalmente compartidos (acoplamiento indirecto).

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.

Ocultamiento de Datos: Se refiere al control de acceso a los detalles internos de representación de las entidades de software. Si un módulo cambia y sólo afecta a sus elementos privados y no a la interfaz, entonces los otros módulos que lo usan, llamados módulos cliente, no deben de ser afectados, por lo tanto, toda la información acerca de un módulo se prefiere que sea privada a ese módulo, y controlar el acceso a detalles internos de los datos que sean públicos.

Encapsulamiento: Un objeto encapsulado es un componente de software único con fronteras propias y bien definidas que protegen los detalles de su representación interna y con una interfaz de comunicación explícita.

El encapsulado es usado para esconder detalles de implementación que no son importantes para otros objetos. También se utiliza para prevenir una alteración a los datos de un objeto sin autorización. Los cambios a los datos de un objeto sólo se pueden hacer a través de los métodos del objeto. Con el encapsulado, los detalles de implantación pueden cambiar en cualquier tiempo sin afectar otras partes del programa. En la solución de problemas utilizando la tecnología orientada a objetos, el componente de software encapsulado o la instancia de la clase es el objeto.

Coherencia: Es el grado de relación entre las responsabilidades de una unidad de programa. La coherencia se basa en el principio de una única responsabilidad. En el contexto de investigación de esta tesis, una responsabilidad se refiere al número de secuencias interactivas necesarias para alcanzar un objetivo o meta de valor para un usuario.
Una secuencia interactiva se define como el número de métodos implicados para cumplir con un objetivo (Misic, 2001).

Cohesión: En el diseño orientado a objetos la cohesión es el grado en el cual todos los elementos (de un módulo) contribuyen a la realización de una única función bien definida. Se logra alta cohesión cuando se disminuyen al mínimo las responsabilidades de una clase (una clase-una responsabilidad). A nivel de paquetes, la cohesión tiene que ver con que una comunidad de clases colaboran para realizar una única responsabilidad entre todas, es decir,

que un paquete sea autosuficiente o auto-contenido para realizar una responsabilidad (varias clases-una responsabilidad). (Misic, 2001)

Cohesión- Interfaces de Comunicación-Autosuficiencia: La autosuficiencia es directamente proporcional al grado de coherencia y al grado de cohesividad e inversamente proporcional al No. de canales de comunicación.

Un módulo de programa con baja cohesión es un módulo incoherente, es decir que sus funcionalidades no persiguen un mismo fin o meta de valor para el cliente, sino que realizan varias cosas no relacionadas.

En un sistema de software, es deseable contar con pocos canales de comunicación, pues estos pueden ocasionar alto grado de dependencias. Para obtener pocos canales de comunicación, se necesitaría de pocos módulos interviniendo en el sistema. Para tal efecto, es necesario que los módulos sean de alta cohesión y coherencia. Módulos con baja cohesión presentan el problema de inmovilidad, es decir, no se puede fragmentar para reusar las funcionalidades por separado, lo cual impide su reuso, por lo tanto, es deseable mantener alta cohesión y coherencia.

Un buen diseño, es aquel que mantiene un equilibrio entre pocas interfaces de comunicación y alto grado de cohesión y coherencia. Esto se logra cuando en un paquete o unidad de programa se combinan conjuntos de clases o componentes altamente cohesivos en comunidades organizadas coherentemente y que colaboran (con pocas interfaces de comunicación) para alcanzar un mismo fin o meta. Es decir que alcancen un buen nivel de auto-suficiencia, tanto a nivel funcional, como de unidades de programa y paquetes.

Auto-suficiencia: Los Módulos, Componentes, Clases, o Servicios, que son autosuficientes, se definen como aquellas unidades de software completas y suficientes, es decir, que contienen solo la información y la funcionalidad completa y suficiente para realizar un objetivo o único o para alcanzar una única meta de valor para el cliente.

Auto-suficiencia es un término relativo que está en función del resultado de valor esperado por un actor. Por ejemplo, a nivel funcional un método puede ser auto-suficiente para producir una operación atómica o unidad lógica de trabajo, pero por sí solo no es auto-suficiente para satisfacer un requerimiento. Un componente puede ser auto-suficiente para satisfacer un requerimiento, pero puede estar sobrado para producir una operación funcional atómica e insuficiente para producir el resultado de un proceso de negocios o de software. A nivel de proceso éste es auto-suficiente para producir un objetivo del negocio o un resultado de un proceso de software. Puede estar sobrado para satisfacer un requerimiento y demasiado sobrado para satisfacer una operación funcional y puede ser insuficiente para producir todas las capacidades de un sistema de software o un negocio.

Bajo este razonamiento, la auto-suficiencia es relativa al nivel de granulación y al resultado de valor esperado por el actor. Así, auto-suficiencia debería de estar definida en términos de completes y suficiencia, es decir, que una entidad de software no tenga más ni menos funcionalidad sino la necesaria para producir el resultado de valor para un usuario.

La auto-suficiencia podría determinarse a partir de las relaciones o dependencias que tengan las entidades de software colaborativas. Es decir si alguno de los procesos que describen una unidad de programa requiere la ayuda de alguna entidad de software externa esta deja de ser auto-suficiente. Se puede medir verificando que las Unidades de Programa no contengan dependencias hacia otras unidades de programa y que las tareas a realizar se ejecuten o se realicen, coherentemente, dentro de sí misma.

Objetivo o Meta de software: A nivel de función como unidad de programa, una meta u objetivo es realizar una única operación funcional. A nivel de Módulo o subsistema, una meta u objetivo es el de realizar una capacidad alterna, un caso de uso o requerimiento específico. A nivel de subsistema una meta u objetivo de valor es realizar en proceso del negocio. A nivel de sistema una meta u objetivo es resolver una aplicación completa o procesos del negocio planteados por un cliente o usuario.

Caso de Uso: Representa la meta de una interacción entre un actor y el sistema. La meta representa un objetivo medible y significativo para el actor. Capturan y por lo tanto contienen los requerimientos funcionales de los sistemas, en un formato fácil de seguir y fácil de leer.

Marco de aplicaciones orientada a objetos (MOO): La orientación a objetos surge como una evolución de la programación estructurada, la cual permite el planteamiento de un programa como una modelación de los elementos del mundo real. Esta aproximación permite desarrollar aplicaciones de gran complejidad, facilitando su reutilización y modificación. Al mismo tiempo, pueden ser gestionadas por un equipo de trabajo heterogéneo de manera flexible y eficiente. (Moreno, Terrén, & Jiménez, 2007)

Un MOO es un conjunto semi-completo de clases, en colaboración, que incorpora un diseño genérico el cual puede ser adaptado a una variedad de problemas específicos para producir nuevas aplicaciones hechas a la medida.(Santaolaya, 2003)

Características de los marcos orientados a objetos (Froehlich, Hoover, Liu, & Sorenson, 1998) y (García, Barras, Laguna, & Marqués, 2002):

- Se emplean para ofrecer soluciones para un conjunto de problemas similares y relacionados dentro de un dominio.
- Son de naturaleza incompleta. El diseño de un marco incluye puntos que requieren de su extensión e instanciación añadiendo soluciones concretas para problemas específicos.
- No cubren toda la funcionalidad requerida por el dominio, sino que abstraen la funcionalidad común requerida por muchas aplicaciones, incorporada por el usuario del marco.
- Inversión del control. El código del marco tiene el hilo de control y llama al código de la aplicación cuando lo considera apropiado.
- Extensibilidad. Un marco ofrece una infraestructura bien diseñada, de forma que cuando se crean nuevos elementos de software, éstos pueden añadirse o sustituir a otros elementos con un mínimo impacto en el resto del marco.

Los beneficios principales que proporcionan los MOO's son (Mohamemed, F. Douglas, 2006):

- *Modularidad*: Mejora la modularidad encapsulando los detalles volátiles de implementación separados de las interfaces estables. La modularidad de un marco ayuda en la mejora de la calidad del software mediante la localización del impacto de diseño y cambios de implementación, reduciendo el esfuerzo requerido para entender y mantener el software existente.
- *Reusabilidad*: Las interfaces estables proporcionadas por los marcos de aplicaciones mejoran el reuso mediante la definición de componentes genéricos que pueden ser utilizados para el desarrollo de nuevas aplicaciones.
- *Extensibilidad*: Aumenta la extensibilidad, proporcionando métodos gancho explícitos (*hooks*), que atrapan nuevas funciones de aplicaciones específicas con las interfaces estables a nuevas funcionalidades. Los métodos gancho sistemáticamente desacoplan las interfaces estables del dominio de aplicaciones y las variaciones requeridas por la instanciación.
- *Inversión de control*. La arquitectura de ejecución de un marco de aplicaciones, es caracterizada por una "inversión de control". Esta arquitectura indica los pasos del proceso de aplicaciones específicas, necesarios para el manejo de objetos que son invocados por el propio marco de aplicaciones.

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 ésto no ocurra, una refactorización siempre cuenta con precondiciones y poscondiciones 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. (Opdyke, 1992)

Las refactorizaciones soportan el diseño y la evolución del software, reestructurando un programa de una manera que permite que otros cambios sean realizados de una manera más fácil. Los cambios más complicados pueden hacerse usando varias refactorizaciones y extensiones. (Opdyke, 1992)

Una refactorización mejora el diseño, si las unidades de código resultante corresponden a abstracciones significativas que hacen más fácil refinar o extender un MOO. La aplicación y el diseñador son los que deciden qué abstracciones son significativas. Esto implica que la tarea de refactorización, especialmente las más complejas, requieren algo de interacción con el diseñador. (Opdyke, 1992)

Servicio Web: Un servicio web presenta un recurso de información o de proceso de negocio, al que puede acceder otra aplicación a través de la web y con el cual se puede comunicar a través de protocolos estándares de Internet.

La particularidad que tienen los servicios web es que están diseñados para permitir la comunicación de una aplicación a otra, sin intervención humana. (Martín & Martín, 2014)

Los servicios web se definen a partir de las siguientes especificaciones o protocolos:

- XML (extensible Markup Language): es el lenguaje de marcas que se utiliza para describir la información; puede describir datos y documentos.
- SOAP (Simple Object Access Protocol - Protocolo simple de acceso a objetos): es un protocolo de mensajería (basado en XML), que indica cómo se deben codificar los mensajes que circularán entre las dos aplicaciones, cliente y proveedor del servicio. Este protocolo permite que se comuniquen programas que corren en diferentes sistemas operativos.
- WSDL (Web Services Description Language - Lenguaje de descripción de servicios web): lenguaje que define un mecanismo estándar para describir un servicio web. Los documentos WSDL deben estar disponibles en el servidor web que ofrece los servicios. En realidad, WSDL es un vocabulario XML para describir un servicio web.
- UDDI (Universal Description, Discovery and Integration): Este protocolo proporciona un mecanismo estándar para registrar y localizar los servicios web que se pueden ofrecer a los clientes. Los directorios UDDI actúan como una guía telefónica de servicios web. (Martín & Martín, 2014)

Marco de Servicios Web (MSOA): En el contexto de esta investigación de tesis, un marco de servicio web se puede definir como un conjunto de servicios, cada uno envuelto en una capa en formato XML, que si bien se encuentran desacoplados entre sí, conjuntamente guardan la experiencia y conocimientos de dominios de aplicaciones; y pueden ser integrados o compuestos por mecanismos de orquestación y coreografía para construir nuevas aplicaciones de alto nivel. (Zamudio, 2013)

La estructura de la arquitectura interna de los servicios Web ofrecen los servicios web del marco cuentan con una arquitectura interna que ofrece flexibilidad a cambios y/o extensiones para ofrecer mayor cobertura operacional del dominio, son independientes de la plataforma de desarrollo y son difundidos en la web y accedidos vía Internet a través de protocolos simples estandarizados.
Los marcos de servicios web proporcionan algunos de los beneficios de los marcos de aplicaciones orientada a objetos, pero ahora se comportan de manera parecida a componentes de reuso, en el sentido de que pueden verse como unidades de composición desarrolladas y utilizadas independientemente, ya que los servicios pueden ser ensamblados o compuestos para integrarlos en aplicaciones completas. Es deseable que sean autónomos y sin acceso a los detalles de implementación, con una clara especificación de lo que requieren y lo que entregan. (Zamudio, 2013)

Modelo Independiente de la Plataforma (PIM): Un modelo independiente de la plataforma es una vista de un sistema de la plataforma de punto de vista independiente. El punto de vista independiente de la plataforma se centra en el funcionamiento de un sistema al tiempo que oculta los detalles necesarios para una plataforma en particular.

Un PIM exhibe un grado específico de independencia de la plataforma de modo que sea adecuado para su uso con un número de diferentes plataformas de tipo similar. (Object Management Group, Version, Kennedy, Carter, & Technologies, 2003)

Anexo 2. Estado del Arte

La reutilización aparece a finales de la década de los 60's como una alternativa para superar la crisis del software. En el artículo de la conferencia "*Mass produced software components*" realizada en 1968 (McIlroy, 1969), McIlroy introduce el concepto de reusabilidad y propone una librería de componentes de código fuente y técnicas automatizadas para su adaptación en diferentes grados de precisión y robustez. A partir de esta propuesta surgen soluciones de reutilización para tareas específicas en diversas áreas de la computación como conversión entrada/salida, computación numérica y procesamiento y almacenamiento de texto en línea. La idea fue fundamental para el concepto de *factoría de software*; Este término que fue acuñado por SDC (*System Devepment Corp*) en 1974. El SDC, Estableció un conjunto de procedimientos bien definidos para un proceso de software consistente y repetible, donde los programadores reutilizaban segmentos de código de desarrollos anteriores. Aunque el concepto de reutilización no se encontraba definido dentro del proceso, la experiencia de la SDC fue el punto de partida para las futuras factorías de software, especialmente en Japón, donde el proceso de reutilización formal fue definido e integrado más tarde.

En el primer *Workshop on Reusability in Programming* (1983), en el cual hubo un consenso de investigadores donde se trataba la necesidad de extender la reutilización a diferentes niveles de abstracción. Además, se consiguieron importantes progresos al emerger el paradigma de programación orientada a objetos (POO). En este modelo de programación existe un plan, denominado clase o plantilla, para la creación de objetos. En este plan se define el protocolo de la clase en el que se establecen los niveles de visibilidad de los objetos encapsulados para interactuar con otros objetos desde el exterior, con lo cual se ocultan los detalles de implantación. Esta tecnología tuvo el potencial de incrementar el reuso de componentes de software, a nivel de objetos.

La POO permite factorizar las funcionalidades comunes, extender y especializar la funcionalidad por medio de los mecanismos de herencia y composición de objetos, con los que se habilita al desarrollador para adaptar un componente de código a nuevos usos o aplicaciones sin tener que modificar su definición actual.

A finales de la década de los 80's se intensificaron los trabajos de reutilización y se hicieron importantes avances en técnicas de clasificación, sistemas de librerías, creación y distribución de componentes reusables y ambientes de soporte al reuso. En 1987 Freeman y

Soete (Freeman & Soete, 1997) identificaron el reuso del conocimiento del contexto como uno de los principales objetivos de investigación en la reutilización de software, en esta misma línea, se extendió la definición de reuso para incluir el uso de cualquier cosa asociada con el proyecto de software, incluyendo conocimiento. Esta nueva visión abrió tendencias de investigación en otras áreas y ha contribuido a reconocer que el problema del reuso debe abordarse desde una confluencia de disciplinas.

A principios de la década de los 90's nació el concepto de Marcos de Aplicaciones Orientados a Objetos (MOO´s) o *Frameworks*. Este tiene como objetivo una solución integrada ejecutable de comunicación entre un conjunto de clases abstractas que representan un comportamiento útil en el espacio del problema. Los framework representan soluciones en un contexto particular y proveen mecanismos eficientes para la adecuación del comportamiento a una nueva necesidad. En (Bäumer et al., 1997) presenta un *framework* para desarrollo de sistemas a grande escala en proyectos financieros. Otros trabajos proponen el desarrollo de framework a partir de patrones de diseño. Los patrones de diseño ofrecen una terminología y comportamiento básico que permite capturar las especificaciones de un diseño reusable orientado a objetos.

Los elementos de reuso de este concepto consistieron en el Diseño o Arquitecturas de Clases de Dominios de Aplicaciones de Negocios, así como el código de sus operaciones. Los principales beneficios para reuso que ofrecen los marcos de aplicaciones orientados a objetos fueron (Moreno et al., 2007):

1) La modularidad es mejorada por medio del encapsulado de los detalles de implantación atrás de interfaces estables.
2) La reusabilidad es fomentada por medio de la estabilidad de las interfaces provistas por el Marco OO al definir componentes genéricos que pueden ser re-aplicados para crear nuevas aplicaciones.
3) Mejora la extensibilidad, proporcionando métodos gancho explícito que permiten a las aplicaciones extender sus interfaces estables.
4) La inversión del control permite al Marco OO, más que a cada aplicación, decidir cuál conjunto de métodos específicos de la aplicación invocar en respuesta a eventos externos.

Sin embargo por problemas que presentan los MOO´s, entre otros:

1) Los objetos sólo pueden ser utilizados en aplicaciones escritas en el mismo lenguaje y la misma plataforma.
2) Su utilización es centralizada y propietaria de los desarrolladores.
3) El alto grado de Cohesividad que presentan los MOO´s, impide el reuso de micro-arquitecturas o partes del MOO y su integración con otros componentes legados.
4) La dificultad para la integración de MOO, debido a que estos se diseñan para su extensión y no para su integración y sin pensar en la necesidad de incorporar componentes legados.
5) El choque en los ciclos de eventos de cada MOO que se produce cuando dos o más MOO llaman al código de la aplicación simultáneamente.

Debido a estos problemas, se produjo una brecha en la tecnología para reuso y se dio lugar a nuevos enfoques tecnológicos que solventaran algunos de estos problemas. Así se dio origen al lenguaje de descripción de interfaces (IDL), soportados por el modelo CORBA, para especificar las interfaces en ambientes de computación distribuida; así como al lenguaje de marcas extensible (XML), soportado por la Arquitectura Orientada a Servicios (SOA), que se propuso como estándar para el intercambio de información de manera estructurada entre plataformas diferentes.

Así nació el concepto de Servicios Web. Un Servicio Web es una aplicación lógica programable, accesible utilizando protocolos de comunicación estándar y formatos de datos tales como los protocolos de Internet, HTTP (Protocolo de Transferencia de Hipertexto) y XML (Lenguaje de Marcación Extensible). Son envueltos en una capa XML, lo que oculta los detalles de implementación.(W3C, 2011)

La tecnología de Servicios Web combina los mejores aspectos del desarrollo basado en componentes y la Web. Son envueltos en una capa XML, lo que oculta los detalles de implementación. Así como los componentes, los Servicios Web son independientes, auto-contenidos y sin estado. Ofrecen un conjunto de funcionalidades, encerradas en cajas negras, que pueden ser reusados sin preocuparse de cómo están internamente implementados.

En esta tecnología, los elementos de reuso son los Servicios Web como tales. Los clientes de éstos pueden ser implementados en cualquier plataforma y en cualquier lenguaje de programación, en tanto ellos puedan crear y consumir el mensaje definido para la interfaz del Servicio Web.

Entre los beneficios del concepto de Servicios Web están: la mejora en su reuso por integración o composición con otros componentes aún de diferente lenguaje; los Servicios Web pueden estar distribuidos en la red por lo tanto ser usados independientemente de los desarrolladores.

A pesar de los beneficios, y de la evolución de la tecnología para reuso, los Servicios Web tienen problemas que lo frenan o impiden. Las UDDI´s no están organizadas por Dominios; los desarrolladores con frecuencia tienen problemas para encontrar los componentes que satisfacen sus necesidades debido a que los mecanismos de Búsqueda y Selección no es precisa; los servicios web son compuestos o integrados con toda su funcionalidad aún cuando se requiera sólo una parte de ésta; no se pueden adaptar a necesidades específicas o extender a nuevas funcionalidades, más que por la combinación con otros componentes. Sin embargo, aquí se ubica el Estado del Arte en cuanto a componentes de reuso se refiere.

Un aspecto importante que frena el reuso de servicios web, es que no existen catálogos de Servicios Web, disponibles para todos los dominios de aplicaciones.

Una buena fuente para disminuir esta carencia es crear servicios web desde el código legado de las empresas de negocios o bien de dominios de aplicaciones. Una alternativa para hacerlo así es a través de procesos manuales, pero esto implíca que el desarrollador deberá tener la habilidad y conocimiento para realizar el análisis y la

descomposición del código legado, la selección de clases para cada servicio web y la generación de un servicio Web por de cada parte obtenida, en este proceso existe el riesgo de introducir defectos debido al proceso de conversión manual.

De manera alternativa, en esta tesis se propone la construcción de repositorios de servicios web especializados por dominios de aplicaciones. La idea es aprovechar los beneficios de los marcos de aplicaciones orientados a objetos y aprovechar los beneficios de los servicios web, mediante procesos automatizados de reingeniería hagan la transformación de MOO´s hacia lo que nosotros hemos llamado Marcos de Servicios Web (MSW), por la analogía de la descripción de ambos enfoques.

Desde esta perspectiva, se describe un marco de servicios web como: un conjunto de servicios envueltos en una capa en formato XML, que guardan la experiencia y conocimientos para satisfacer los requerimientos de dominios de aplicaciones, cuya arquitectura interna ofrece flexibilidad a cambios y/o extensiones para ofrecer mayor cobertura operacional del dominio. Estos son independientes de la plataforma de desarrollo y son difundidos en la Web y accedidos vía Internet a través de protocolos simples estandarizados. Son unidades independientes, lo cual mejora su reuso por integración o composición con otros Servicios Web disponibles. El problema de huecos que presentan los MOO´s es mejorado conforme se construyen más y nuevos Servicios Web del dominio de aplicaciones. No existe el problema de inter-operación debido a que son elaborados utilizando estándares de comunicación. El problema de empalmes que presentan los MOO's, se resuelve con la decisión de los desarrolladores de utilizar uno u otro servicio Web que ofrezca la misma funcionalidad con ventajas de calidad externa. Los elementos de reuso de los MSW son los propios servicios que lo conforman, usados como unidades de composición de los diferentes dominios de aplicaciones. El control de flujo de las aplicaciones puede ser orquestada por mecanismos de Composición de Servicios.

El reuso se realiza a nivel de Servicios Web, difundidos en los depósitos de proveedores y accedidos vía Internet. Las aplicaciones pueden ser desarrolladas por integración y/o composición de Servicios Web de diferentes lenguajes y plataformas. El ocultamiento de los detalles internos de representación se da por la capa XML en la que los Servicios Web son empaquetados. Su utilización es distribuida y dispuesta a los desarrolladores aun cuando no sean estos los proveedores.

Entre los beneficios de reuso con este enfoque están: Se conserva la modularidad al no tener acceso a los detalles de implementación. La reusabilidad es mejorada porque su comportamiento es más parecido a Componentes de Reuso. La reusabilidad es mejorada porque son unidades de composición de parte de terceros, desarrollados y utilizados independientemente. La reusabilidad es mejorada porque son auto-contenidos. La reusabilidad es mejorada porque cuentan con una clara especificación de lo que requieren y lo que entregan. Se fomenta el control de su evolución y mantenimiento puesto que su arquitectura interna es flexible para permitir la extensión a nuevas funcionalidades por herencia.

Los trabajos relacionados con el proyecto propuesto tienen que ver con el reuso y la generación de servicios Web, algunos a partir de código legado (Araiza, 2007) y (León,

2009) . Otros trabajos generan servicios Web a partir del código de MOO´s con un enfoque de construcción de servicios que presenta la siguiente problemática:

El trabajo que se describe en este documento genera servicios web a partir del código de MOO´s, reconociendo las diferentes secuencias interactivas del mismo. Se considera que cada método público, del marco original, es una interfaz de entrada para la petición de un servicio web, por lo tanto a partir de este punto de entrada se reconoce toda su secuencia interactiva. Posteriormente, se agrupan todas estas funciones en una única clase, es decir, toda la funcionalidad requerida para secuencia interactiva, pero se les eliminan los datos y las funciones que no están participando en la secuencia, sólo se deja aquellos atributos y funciones que están participando en la interacción. Con esta arquitectura se construye el servicio Web. El efecto que se logra radica en un mejor grado de tiempo de respuesta y mejor autonomía externa de los servicios web, para satisfacer requerimientos de programa, ya que cada uno de éstos atiende a un caso de uso, resultado de valor (a nivel de requerimientos), sin la necesidad de la participación de otros servicios web para el Caso de Uso. Así la orquestación de servicios será con un número menor de éstos y sólo será para satisfacer una aplicación completa o proceso de negocio, en lugar de la satisfacción del requerimiento o caso de uso, que ya de por sí está satisfecho en cada servicio web obtenido. Sin embargo, no se consiguen los mejores resultados en mantenimiento ya que no se respeta la arquitectura del MOO original, lo que ocasiona un bajo nivel de cohesión lo que impacta al mantenimiento y al control de la evolución del servicio Web.

La intención del proyecto en este documento consiste en agrupar de mejor manera las clases de objetos cuya funcionalidad deberá estar integrada en los diferentes servicios Web para satisfacer diferentes requerimientos del dominio de aplicaciones, logrando servicios con mejor independencia, autonomía y que proporcionen un mejor rendimiento y tiempo de respuesta.

La selección de clases que participarán en un servicio Web se realizará mediante un análisis léxico y sintáctico, automatizado, de los marcos orientados a objetos en estudio y un análisis semántico para la generación de código del Marco con Arquitectura Orientada a Servicios equivalente.