XAVIER SALVADOR MEDIANERO PASCO

METODOLOGÍA PARA EL SOPORTE DEL DISEÑO DE FRAMEWORK BASADA EN PROGRAMACIÓN ORIENTADA A ASPECTOS

Universidad Tecnológica de Panamá
Facultad de Ingeniería en Sistemas Computacionales
Maestría en Ciencias de Tecnología de Información y Comunicación

Panamá
2012

XAVIER SALVADOR MEDIANERO PASCO

METODOLOGÍA PARA EL SOPORTE DEL DISEÑO DE FRAMEWORK BASADA EN PROGRAMACIÓN ORIENTADA A ASPECTOS

Tesis presentada a la Facultad de Ingeniería en Sistemas Computacionales de la Universidad Tecnológica de Panamá para la obtención del título de:

Magister en Ciencias de Tecnología de Información y Comunicación con Especialización en Ingeniería de Software

Asesor
PhD. Sérgio Crespo C.S. Pinto

Co-Asesor:
PhD. Clifton Eduardo Clunie Beaufond

Panamá
2012

Agradecimiento

Al finalizar este arduo y difícil trabajo es inevitable agradecerle primeramente a Dios por darme fuerza y sabiduría a lo largo de todo el proyecto, a mis asesores por sus guías y consejos, a la universidad de UNISINOS en Brasil por brindarme sus instalaciones y facilidades para llevar a cabo la parte práctica de mi proyecto, a SENACYT por financiar la investigación, a mi familia por su apoyo incondicional y a todas las personas que de una u otra manera me apoyaron en el desarrollo de este proyecto.

Resumen

La programación orientada a aspectos presenta valorables ventajas frente a otros paradigmas de programación, pero a su vez presenta dificultades a la hora de aplicar conceptos dentro de las etapas de análisis y desarrollo de la Ingeniería de Software. En este trabajo se propone una metodología para reducir los inconvenientes del paradigma, a la vez de proporcionar pasos que integran elementos de análisis comunes de la Ingeniería de Requerimientos con los aspectos (unidad básica del paradigma) con el propósito de generar framework para un dominio específico. La metodología propuesta reúne algunas bondades otras metodologías, pero está enfatizada al tratamiento de las mayores desventajas de la programación a aspectos, además, este artículo proporciona una herramienta que apoya algunos pasos de la metodología generando parte del código base del framework. Dentro del proceso de tratamiento de aspectos, el análisis se orienta a la especificación de aspectos según AspectJ con reglas para la ubicación y determinación dentro de sus cuatro etapas cíclicas. Finalmente, se incluye un caso de estudio donde se evalúan los pasos de esta metodología, la herramienta de apoyo y se analizan los resultados.

Palabras Clave: Ingeniería de Software, Metodología, Programación Orientada a Aspectos, Desarrollo de Software Orientado a Aspectos, AspectJ, Framework.

Abstract

The aspect-oriented programming has valuable advantages over other programming paradigms, but in turn presents difficulties applying the concepts within the stages of analysis and development to reduce the drawbacks of this paradigm. This paper proposes a methodology to reduce the drawbacks of the paradigm, at same time providing steps that involve elements of common analysis in the Requirements Engineering with Aspects (basic unit of paradigm) in order to create framework for a specific domain. The proposed methodology brings together some benefits methodologies, but it emphasized the treatment of older disadvantages of the programming aspects, in addition, this article provides a tool that supports some steps of the methodology generating part of the framework code base. In the process of treatment issues, the analysis is oriented to the specification of aspects using AspectJ, with rules to locate and determinate aspects within its four cyclical stages. Finally, it includes a case study which evaluates the steps in this methodology, support tool and results are analyzed.

Keywords: Software Engineering, Methodology, Aspect Oriented Programming, Aspect Oriented Software Development, AspectJ, Framework.

Contenido

Índices de Tablas XI

Índices de Imágenes XIII

Introducción 1

1.1. Motivación 4
1.2. Justificación 5
1.2.1. Definición del Problema 6
1.3. Alcances 7
1.3.1. Alcance de la Metodología 7
1.3.2. Alcance de la Herramienta de Apoyo 8
1.4. Objetivos 8
1.4.1. Objetivo General 8
1.4.2. Objetivos Específicos 8
1.5. Metodología 9

Marco Referencial 13

2.1. Programación Orientada a Aspectos 15
2.2. Desarrollo de Software Orientado a Aspectos 19
2.3. Inconvenientes de la Programación Orientada a Aspectos 23
2.3.1 Conflicto Estructural: 23
2.3.2. Conflicto de Comportamiento: 23
2.3.3. Conflicto por Multifuncionalidad: 24
2.3.4. Facilidades Dinámicas: 24
2.4. Metodologías Orientadas a Aspectos 25
2.4.1. Ingeniería AORE 25
2.4.2 Modelo por Puntos de Vistas: 29
2.4.3. Ingeniería AOCRE 32

2.4.4. Metodología SLAI 33
2.5. Modelos de Tratamiento de Inconvenientes 36
2.6. Frameworks Basados en Programación Orientada a Aspectos 37
2.6.1. Framework basado en Componentes y Aspectos 37
2.6.2. Framework basado en el Modelo Conceptual 38
2.6.3. Framework de Tejido Dinámico 39
2.6.4. Framework THAOP 39
2.6.5. Framework TITAN 40
2.7. ASPECTJ 41
2.8. Otros medios para determinar Asuntos Transversales 45

Metodología Propuesta: MEDFOAR 47
3.1. Identificación de Asuntos 50
3.1.1. Enfoque a Vistas 51
3.1.2. Elaboración de Casos de Uso 52
3.1.3. Contabilización de identificadores determinantes de asuntos 52
3.1.4. Identificación de Módulos Multifuncionales 53
3.1.5. Actualización de Casos de Uso 54
3.1.6. Contabilización de identificadores determinantes de asuntos 54
3.2. Identificación de Aspectos: 55
3.2.1. Identificación de Influencias y Dependencias 55
3.2.2. Aplicación de la Regla de Descomposición: 56
3.2.3. Aspectos a través de Requerimientos No Funcionales 58
3.2.4. Selección de Aspectos Candidatos 59
3.2.5. Selección de Aspectos Clásicos y Aspectos Dinámicos 59
3.2.6. Integración de Aspectos Candidatos 60
3.2.7. Aplicación de Metadatos 60
3.3. Especificación de Aspectos Candidatos. 61
3.3.1. Control de Rango de los Aspectos y Cortes Transversales 62
3.3.2. Determinación y Ubicación de los Elementos Propios de los Aspectos 63
3.3.3. Control de Redundancia 66
3.3.4. Compactación de Componentes 67
3.3.5. Especificación de Metadatos 67
3.4. Catalogación y Conflictos de Aspectos 68
3.4.1. Catalogación de Aspectos 69
3.5. MEDFOAR y los Inconvenientes de la POA 71

3.5.1. Conflicto Estructural: 71
3.5.2. Conflicto de Comportamiento: 72
3.5.3. Conflicto por Multifuncionalidad: 73
3.5.4. Facilidades Dinámicas: 74

Herramienta de Soporte: HSTAR 75
4.1. Vista de Formularios: 77
4.1.1. Pestaña: Principal <MAIN> 77
4.1.2. Pestaña: Acciones <ACTIONS> 81
4.1.3. PESTAÑA <XML> 86
4.1.4. PESTAÑA <GRAPHICS> 88
4.2. Uso de la Herramienta de Apoyo: 89
4.2.1. Fase 1 90
4.2.2. Fase 2 90
4.2.3. Fase 3 92
4.3. Acciones Opcionales: 93

Caso de Aplicación y Análisis de Resultados 97
5.1. Etapa1: Definición de Requerimientos: 99
5.1.1. Enfoque a Vistas e Identificación de elementos claves: 99
5.1.2. Elaboración de Casos de Uso. 101
5.1.3. Contabilización de identificadores determinante de asuntos. 104
5.1.4. Identificación de Módulos Multifuncionales 105
5.2. Etapa 2: Identificación de Aspectos: 105
5.2.1. Identificación de Influencias 105
5.2.2. Aplicación de la matriz de descomposición 109
5.2.3. Aspectos a través de Requerimientos No Funcionales 109
5.2.4. Aspectos Candidatos 110
5.2.5. Selección de Aspectos Clásicos y Aspectos Dinámicos 111
5.2.6. Integración de Aspectos Candidatos 111
5.2.7. Aplicación de Metadatos 112
5.3. Etapa 3: Especificación de Aspectos 113
5.3.1. Control de Rango de los Aspectos y Cortes Transversales 113
5.3.2. Determinación y Ubicación de los Elementos Propios de los Aspectos: 114
5.3.3. Control de Redundancia 116
5.3.4. Compactación de Componentes 118
5.3.5. Especificación de Metadatos 119

5.4. Etapa 4: Catalogación y Conflicto de Aspectos:.....120
5.4.1. Catalogación de Aspectos.....120
5.4.2. Ponderación de Aspectos en Conflictos.....121
5.5. Análisis de Resultados.....122

Conclusiones y Trabajos Futuros.....127

Referencias Bibliográficas.....131

Anexos.....133
A. MEDFOAR y Panamá.....133
A.1. Antecedentes.....133
A.2. Beneficios y principales beneficiarios.....135
A.3. Impacto esperado.....135
B. Trabajos científicos desarrollados.....136

Índices de Tablas

Tabla 1. Comparación entre los Enfoques que involucran POA en la toma de requerimientos. 22

Tabla 2. Representación del dimensionamiento de datos (Rashid, Sawyer et al. 2002) 29

Tabla 3. Elementos considerados en un Modelo por Puntos de Vista. 31

Tabla 4. Definición para los Puntos de Cortes 43

Tabla 5. Elementos de un diagrama de vistas 52

Tabla 6. Elementos abstraídos de los casos de uso 56

Tabla 7. Elementos abstraídos en los metadatos. 61

Tabla 8. Control de Rango y Corte Relacional 62

Tabla 9. Representación de los elementos de los aspectos 63

Tabla 10. Resolución de Conflictos 71

Tabla 11. Vistas por Usuario Interno. 99

Tabla 12. Vistas por Usuario Externo. 100

Tabla 13. Clasificación Por Vistas 100

Tabla 14. Repositorio de Identificadores Léxicos 104

Tabla 15. Matriz de Dependencias e Influencias 106

Tabla 16. Simbología para la Matriz de Dependencias e Influencias 106

Tabla 17. Matriz de Dependencias e Influencias Generalizada 107

Tabla 18. Datos correspondientes a los metadatos 112

Tabla 19. Control del Rango de los Aspectos 113

Tabla 20. Estructura de paquetes para los aspectos 119

Tabla 21. Ponderación de aspectos en conflictos 121

Índices de Imágenes

Imagen 1. Creación de ejecutables a través de la POA 17

Imagen 2. Representación de Multiplicidad de los asuntos. 28

Imagen 3. Modelo de Procesos de Visión. 30

Imagen 4. Principales etapas de la Metodología MEDFOAR. 48

Imagen 5. Etapas de la Metodología 50

Imagen 6. Ubicación de los aspectos y sus elementos. 65

Imagen 7. Procesos de la Metodología y la Herramienta frente a logros parciales 76

Imagen 8. Pantalla correspondiente a la pestaña <MAIN> 78

Imagen 9. Pantalla <MAIN> mostrando el proceso de inserción de aspectos. 79

Imagen 10. Elementos de la Clase Principal y Relaciones dados por el signo más (+)..... 79

Imagen 11. Pantalla de la pestaña <ACTIONS>. 82

Imagen 12. Estructura de relaciones permitidas para cada opción. 83

Imagen 13. Pantalla correspondiente a la pestaña XML. 87

Imagen 14. Estructura de Archivos generada por la herramienta. 88

Imagen 15. Gráfica formada con graphviz que representa el diseño lógico del sistema. 89

Imagen 16. Tipo de elemento seleccionado. 91

Imagen 17. Control de Aspectos dinámicos 92

Imagen 18. Formulario de búsqueda de carpetas 93

Imagen 19. Resolución de Conflictos 94

Imagen 20. Diagrama de Casos de Uso. 103

Imagen 21. Elementos y sub-elementos presentados en la primera pestaña. 108

Imagen 22. Ingreso de relaciones y dependencias en la pestaña de ACTIONS 108

Imagen 23. Aspectos candidatos 110

Imagen 24. Aspectos Dinámicos 112

Imagen 25. Relaciones de los aspectos con sus elementos y los métodos que interceptan. 117

Imagen 26. Casos de Uso con los aspectos incorporados en el mismo. 118

Imagen 27. Estructura XML para uno de los aspectos existentes en el caso de estudio. 120

Capítulo 1

Introducción

Una de las más grandes complicaciones del proceso ingenieril de desarrollo de software es cumplir con un nivel de calidad que no perjudique el costo o el tiempo que se necesita en este proceso. La ingeniería de software es una disciplina que nace como medida para tratar de garantizar un producto de calidad que cumpla con los estándares exigidos mediante un procedimiento estructurado y cíclico de continua mejora de software tanto en diseño, implementación y mantenimiento del mismo.

La ingeniería de Software consta de varias etapas, cada una de ellas presenta un grado significativo de importancia para el proceso; este trabajo se enfoca en la primera etapa: La Ingeniería de Requerimientos debido a su fuerte impacto en las etapas posteriores y a los resultados finales de esta etapa: El diseño lógico del sistema, la interpretaciones de las necesidades y requerimientos del cliente, los documentos con la información detallada del software, entre otros.

Uno de los paradigmas de programación que provee un alto grado de beneficios en procesos de desarrollo y mantenimiento de la Ingeniería de Software es el paradigma de programación orientada a aspectos (POA), el cual promueve e impulsa la separación de conceptos en asuntos con características transversales proporcionando ventajas sobre otros paradigmas modernos.

La especificación correcta de funcionalidades y la descripción de los componentes del software en la Ingeniería de Requerimientos permitirá crear una base robusta para las etapas posteriores reduciendo la complejidad de las mismas.

La incorporación de la Programación Orientada a Aspectos, cuya función es la encapsulación de funciones en elementos denominados aspectos en beneficio de procesos de mantenimiento, depuración y reducción del acoplamiento, en la ingeniería de requerimiento, permitirá incorporar desde etapas tempranas este paradigma y sus beneficios al diseño y productos de la fase.

La separación de funcionalidades consiste en realizar cortes transversales en el diseño del software en bloques denominados “Aspectos”. Estos aspectos son unidades de abstracción y composición que recopilan instrucciones que son difíciles de encapsular debido a su presencia en diferentes funcionalidades, los aspectos no son unidades identificables e independientes como las clases, sino que son elementos abstractos que, generalmente, brindan características y agrega funcionalidades a otros elementos debido a su diseminación lógica por los mismos.

Como los aspectos encapsulan características transversales, las clases y métodos sólo tendrán la información relevante a su funcionalidad, lo que hace estas tareas más eficientes y fáciles; también, agrupan las características extras de ciertos elementos, el código entrelazo y disperso será reducido pues no estará presente en todas las funcionalidades sino sólo en el aspecto, permitiendo que el código y el análisis sea más sencillo y completo.

Debido a estas ventajas, las principales investigaciones modernas apuntan al desarrollo y el estudio del paradigma de orientación a aspectos, se enfocan en tratar los problemas existentes en la naturaleza de los cortes transversales que provocan inconvenientes entres los aspectos y entre los aspectos y el lenguaje bases; además de tratar de minimizar las complicaciones y dificultades que involucra el análisis de los aspectos y de sus componentes en etapas tempranas del proceso de desarrollo del software.

A pesar de todos estos beneficios, la programación orientada a aspectos presenta desventajas e inconvenientes que se intentan resolver o minimizar, según su complejidad y naturaleza, con el fin de depurar el paradigma haciéndolo fácil, robusto y reduciendo su complejidad. En esta tesis, se tratan los problemas referentes a la complejidad de diseño que involucran la ubicación y especificación de aspectos dentro del proyecto.

El paradigma de POA, también presentan algunos inconvenientes que dificultan su uso, estas desventajas radican en pequeños problemas debido a su naturaleza de cortes transversales y de comportamiento frente al lenguaje base.

Entre las desventajas (G. Cugola, C. Ghezzi et al. 1999) tenemos:

- Conflicto entre aspectos.
- Conflicto entre el lenguaje base y los aspectos
- Complejidad en el análisis orientado a aspectos.

De este último punto, nacen los problemas de dificultad nata de los aspectos: conflictos estructurales (Nusayr 2008; Hu, He et al. 2009), de comportamiento (Cazzola, Jézéquel et al. 2006; Hu, He et al. 2009) y por características de dinamismo (Jun-Wei, Rong et al. 2008), estos puntos serán detallados en el capítulo II.

Dentro de la Ingeniería de Software, otro de los productos finales es el diseño de framework de dominios específicos. Los frameworks son generadores de aplicación hacia un dominio específico (Markiewicz and Lucena), contienen módulos, métodos y características de software cuya función es ser un molde para aplicaciones que se deriven del mismo, es decir, que utilicen el mismo núcleo y estén cubiertas por un área específica.

Los frameworks están compuestos por dos secciones: el núcleo conocido como “los puntos congelados”, que representan las clases, librerías, métodos y módulos que son iguales y que sirven como base para todas las aplicaciones, es decir la parte inmutable e invariable del framework, en otras palabras, todas las aplicaciones generadas por este framework tendrán las mismas características base guardadas en esta capa. La segunda sección son las ranuras conocidas como “los puntos calientes”, que representan los elementos que pueden adaptarse, agregarse o simplemente prescindir de ellos en la aplicación, puede imaginarse como un checklist donde se eligen los métodos extras que debe tener la aplicación, los elementos en las ranuras no deben ser obligatorios para el funcionamiento del framework (Markiewicz and Lucena).

Existen varios paradigmas utilizados en la elaboración de estos frameworks, dentro de los cuales se encuentra la orientación a objetos, a aspectos, a servicios, entre otros. El modelado de aspectos toma en consideración dos aspectos según (Zhang and Zhang 2005): modelos dinámicos que tratan los eventos y la traza de eventos y los modelos estáticos que ocupa los puertos de aspectos y el tipo de mensaje.

La utilización de frameworks basados en orientación a aspectos (Fb-AOP) presenta ventajas sobre las otras arquitecturas debido a que permite:

- Encapsular la abstracción de los aspectos
- Incrementa la reusabilidad logrando mayor separación de conceptos.
- Integra los elementos del núcleo con las propiedades y métodos de las ranuras que están en función de los aspectos.

Combinando estos dos elementos, esta tesis se propone normas que adaptan e incluyen la orientación a aspectos en esta etapa inicial de análisis combinando mecanismos y pasos provenientes de otras metodologías como la Ingeniería de Requerimiento Orientada a Aspectos, a Componentes y Aspectos, Modelos de Puntos de Vistas, Ingeniería de orientación a aspectos utilizando Casos de Uso, entre otras; para diseñar framework orientados a aspectos.

El propósito de la misma es el de establecer una serie de etapas con pasos definidos que refuerce la identificación y especificación de los elementos del paradigma de orientación a aspectos dentro del proceso de diseño del sistema de la Ingeniería de Requerimientos en la aplicación de la Ingeniería de Software, mediante la utilización de prácticas rutinarias como casos de Uso, enfoque a objetivos y puntos de vistas.

1.1. Motivación

La motivación que impulsó esta investigación radica en la necesidad de incorporar a etapas de análisis y diseño de sistemas de software (Ingeniería de

Requerimientos) atributos de la programación orientada a aspectos y de esta forma, abstraer todas las ventajas de la misma para consolidar una base sólida para etapas posteriores de la Ingeniería de Software.

Debido a la complejidad de este proceso de integración y de los diferentes enfoques de otras metodologías se ha diseñado un conjunto de normas que recopilan las ventajas de otras técnicas y que su vez, aportan ideas para la incorporación, identificación y especificación de aspectos dentro de los procedimientos rutinarios de tratamiento de clases, componentes y otros elementos de diseño de software.

Con esta investigación se pretende integrar las ventajas de metodologías de orientación a aspectos que utilizan como soporte diversos elementos como diagramas de caso de uso, enfoque a objetivos y por puntos de vistas, orientación a componentes que serán detallados durante el trabajo, de manera que se pueda contar con normas más acopladas y con un procedimiento cíclico para el tratamiento de aspectos.

1.2. Justificación

En el área de la ingeniería de Software, la ingeniería de requerimientos es una de las etapas más importantes ya que la identificación correcta y completa de los requisitos del sistema facilitará la tarea en todas las próximas etapas, si este procedimiento no se realiza adecuadamente las etapas posteriores de análisis, desarrollo e implementación serán mermadas por el arrastre de unos requerimientos no adecuados que no fueron proporcionado de esa manera por el cliente.

La mayoría de metodologías en la ingeniería de requerimientos no toman en consideración la inclusión de aspectos y sus elementos, desaprovechando las ventajas que presenta en enfoque a aspectos, las cuales incluyen:

- Abstraer los requerimientos para que las decisiones y las características generales se coloquen juntas en métodos y no se encuentren dispersas por toda la aplicación.

- Reduce las dependencias entre los requerimientos para así desacoplar los elementos que intervienen en el software.
- Permite diseñar un código más fácil de depurar.
- Hace más fácil el mantenimiento del software.
- Mediante la integración de aspectos en el diseño del sistema, su identificación y especificación mediante el cruce transversal de los requerimientos funcionales y no funcionales del software y su extracción a partir de otros esquemas permitirá contar con todas sus ventajas en la fase inicial del desarrollo del software.

Por lo tanto, la creación de métricas permitiría optimizar la captación de requerimientos y funcionalidades del sistema mediante la utilización de Programación Orientada a Aspectos, ya que esta permite la separación de los conceptos eliminando la cohesión y segmentando en pequeños grupos dependiendo de su funcionalidad cada uno de los requerimientos.

Estas normas, al combinar bondades de diferentes métodos, permitirán servir de guía para identificar los aspectos y especificar influencias, dependencias, nivel de impacto sobre otros aspectos; determinar si es posible la segmentación, ubicar y desglosar los aspectos en sus atributos: puntos de corte, puntos de unión, corte transv ersal, entre otros; lo que reforzará plenamente la etapa de desarrollo e implementación debido a que los aspectos formaran parte del mismo sistema en sí.

1.2.1. Definición del Problema

Acorde con la justificación del trabajo, el problema radica en la importancia del tratamiento de aspectos dentro de la Ingeniería del requerimiento y la deficiencia actual en este enfoque (Budwell and Mitropoulos 2008), definiéndose de la siguiente manera:

En el desarrollo de software basado en programación orientada a aspectos, para obtener un software que cumpla con estándares de calidad, además de los procedimientos estándares de la Ingeniería de Software, es necesario analizar y diseñar el software enfocada a la identificación temprana de aspectos y de esta

manera incluirlo en todos los componentes que sean necesarios. Junto a este inconveniente, se le añaden las debilidades en la ubicación y especificación de los aspectos en la ingeniería de requerimientos lo cual dificulta el aprovechamiento al máximo de este paradigma.

Las metodologías de AOSD (Desarrollo de Software Orientado a Aspectos) presentan diversas ventajas y procedimientos para el tratamiento de los aspectos, los cuales se basan en métricas diferentes para el propósito de abstracción.

El problema nace de la necesidad de una metodología con normas más acopladas que constituyan un procedimiento cíclico de mejoramiento del diseño y estructura del sistema integrando los aspectos en estas definiciones, de tal forma que sea posible obtener un diseño más robusto que sirva de base para las etapas futuros mejorando de esta forma la calidad del software final.

Sin un conjunto de normas y pasos, los beneficios de la orientación a aspectos no pueden obtenerse.

1.3. Alcances

Esta tesis extiende su alcance al tratamiento de Aspectos e identificación de sus elementos para la etapa de Ingeniería de Requerimientos de la Ingeniería de Software, con el diseño de una metodología, una herramienta de apoyo y su evaluación en un caso de estudio.

1.3.1. Alcance de la Metodología

Este tratamiento consiste en incluir los aspectos dentro del análisis de requerimientos ubicándolos correctamente entre procesos correspondientes al paradigma de orientación a objetos, de tal manera que no sean inconvenientes en los análisis realizados. Además, la metodología pretende minimizar los inconvenientes de la programación orientada a aspectos que están bajo la ramificación de dificultad

por naturaleza de la POA, a la vez de aprovechar las ventajas desde inicios del proceso de desarrollo de software.

La metodología finaliza con la obtención de los aspectos estáticos y dinámicos, en conjunto con la determinación y especificación de sus elementos. Además de proporcionar un sistema que permita solucionar conflictos entre aspectos.

1.3.2. Alcance de la Herramienta de Apoyo

La herramienta de apoyo a la metodología se limita a permitir la determinación de aspectos candidatos, la resolución de conflictos y el control de redundancia, la concepción gráfica de dependencias e influencias de funcionalidades y la resolución de conflictos.

También permite la generación del case del framework en una nueva aplicación orientada a aspectos de Eclipse en base a lo diseñado por el usuario en la herramienta.

1.4. Objetivos

Los objetivos a cumplir en esta investigación son los siguientes:

1.4.1. Objetivo General

- Crear una metodología para el diseño de framework basada en programación orientada a aspectos, que mantengan las ventajas y minimice los inconvenientes del paradigma.

1.4.2. Objetivos Específicos

- Maximizar el aprovechamiento del paradigma de orientación a aspectos y minimizar los inconvenientes en la etapa de análisis de la Ingeniería de Software.

- Diseñar un procedimiento estructurado con normas y mecanismos que permitan la identificación y el tratamiento de aspectos dentro del proceso de análisis y diseño del framework de la Ingeniería de Software.
- Diseñar una herramienta que soporte la metodología y permita aplicar de manera automática algunos procedimientos para el tratamiento de aspectos en la Ingeniería de Requerimientos.
- Evaluar las normas de la metodología y la herramienta en un caso de estudio.

1.5. Metodología

La metodología que se aplica en esta investigación consta de un período de cuatro fases para el cumplimiento de los objetivos propuestos. Estas fases están orientadas a investigación y conocimiento del tema, planteamiento de una solución (metodología), diseño y desarrollo de la herramienta de apoyo y la aplicación de todo lo desarrollado en un caso de estudio.

Fase 1: Investigación sobre la Programación Orientada a Aspectos.

Para la primera fase del proyecto, se analizaron diferentes trabajos sobre metodologías existentes de la POA, Frameworks orientados a aspectos de tejidos dinámicos y estáticos, además de modelos para el análisis y la incorporación de aspectos. Los mismos se presentan en el capítulo II de este documento.

Fase 2: Desarrollo de la Metodología

En esta fase, la misión es la de identificar puntos estratégicos en las actividades de análisis, diseño, implementación y depuración que permitan una fuerte integración entre la programación orientada a aspectos y las demás etapas de la Ingeniería de Software.

Los principios en que se basa la metodología son los siguientes:

- La metodología se basa y aplica, en algunos casos, ideas de otros estudios actuales sobre metodologías de orientación a aspectos, frameworks para el

control de los aspectos y modelos de enfoque de los mismos, de tal forma que se cuente con un trabajo más robusto y con base científica.

- El tratamiento de aspectos se implementará dentro del proceso establecido para el tratamiento y diseño de elementos aplicando el paradigma de programación a objetos.
- Los pasos de la metodología son estructurados linealmente de manera que sirven de soporte para los pasos posteriores hasta la identificación y especificación de aspectos.
- Los pasos de la metodología están orientados a minimizar los inconvenientes del paradigma de orientación a aspectos relacionados con la dificultad nata de los puntos de corte y puntos de unión.
- El producto final después de la aplicación de la metodología serán el grupo de aspectos estáticos y dinámicos, la ubicación e identificación de los aspectos y sus elementos, descripción en metadatos de los mismos.
- Valorizar los aspectos dependiendo de sus influencias y funcionalidad de tal forma que se pueden priorizar en importancia frente a un problema de conflictos.

Fase 3: Herramienta de Apoyo

Dentro de los objetivos del proyecto se encuentra el desarrollo de una herramienta de apoyo para la metodología. La existencia de una herramienta donde aplicar la metodología incrementaría aún más las ventajas del paradigma de POA.

El propósito de esta herramienta es la de agilizar algunos procesos de la metodología que se pueden automatizar, de tal forma que sea más rápida la obtención de los aspectos candidatos, además de proporcionar una vista lógica visual de las influencias y dependencias del framework.

Para el diseño de esta herramienta se tomó en cuenta lo siguiente:

- La herramienta será un pluggin de Eclipse Helios, de tal manera que estará hecha en lenguaje Java.
- Permitirá la introducción de clases, funcionalidades, aspectos y funciones macro, presentando una visión lógica de la relación de las mismas.
- Permitirá entrelazar aspectos, puntos de corte, puntos de unión con los demás elementos.

- Admitirá relacionar funcionalidades de tal manera que sea posible determinar el orden de ejecución según el flujo lógico del sistema.

Las funcionalidades de apoyo a la metodología incluyen:

- Representación de aspectos
- Migración de código.
- Integración de aspectos
- Uso de metadatos (elementos de nivel superior que le dan valor adicional de información a funciones y métodos)
- La pasantía garantizará el asesoramiento y la colaboración de expertos para el desarrollo de la misma.
- Los pasos que involucran este procedimiento son:
- Establecer interconexión con Eclipse.
- Creación de interfaces y elementos de la misma.
- Manipulación de elementos e interfaces de diagramado.
- Codificación de elementos.
- Control de Excepciones.
- Funciones de migración de código.

Fase 4: Evaluación y Caso de Estudio

Como última parte, se evaluará la metodología y la herramienta aplicando las mismas en un caso de estudio y analizando su comportamiento y comparando los resultados con los expuestos en la teoría.

En resumen, la metodología y acciones a tomar se basan en esas cuatro etapas, donde se establecerán actividades de análisis, identificación, especificación, pruebas, evaluación para incluir la POA de manera fácil a los procedimientos cotidianos del desarrollo de software impactando en todo proyecto que utilice el mismo.

Capítulo 2

Marco Referencial

En todo proceso de ingeniería de software, una de las etapas más importante es la toma de requerimiento y esto es debido a que en este período se determina los requerimientos funcionalidades y no funcionales del proyecto que se va a desarrollar. La identificación de la estructura, funciones y relaciones de estos elementos es indispensable ya que se utilizarán como base en la etapa de análisis y diseño del software.

Este proceso de ingeniería de toma de requerimientos, mediante la utilización de metodologías intenta representar “todas” las características y funcionalidades del proyecto en base a lo que el usuario solicita, este proceso generalmente no cubre totalmente la realidad debido a diversos factores latentes en esta etapa: ideas no claras en las solicitudes del usuario, pérdida de información al momento de transmitir los requerimientos, visiones diferentes de lo qué se desea realizar, entre otras.

La disciplina de Desarrollo de Software Orientado a Aspectos (AOSD) fue creada para hacer frente a la dificultad del código transversal debido a su complejidad y su dificultad para entenderse, en otras palabras, AOSD fue ideada para minimizar la complejidad de las tareas de la programación orientada a aspectos, en especial la identificación de las funciones utilizadas varias veces dentro de un software. La disciplina AOSD se enfoca en la identificación, separación, representación y la composición de los asuntos (objetivos del programa) de carácter transversal.

Estos asuntos son capturados en módulos independientes denominados aspectos. Los aspectos son estas funciones transversales (invocadas en diferentes partes del software) que interceden y pueden modificar el comportamiento de una aplicación. Los mismos están clasificados, de tal forma que cuando se requiera realizar alguna operación o modificar la funcionalidad de un software, se modifica el aspecto sin la necesidad de buscar a través del código las funciones en las cuales se necesitan los cambios.

El proceso de separación de asuntos es indispensable en el análisis realizado de la ingeniería de software (Dahiya and Sachdeva 2006), en la toma de requerimientos; la misma consiste en la separación de dominios del problema, con el fin reducir el grado de complejidad, de disminuir fallas y evitar conflictos que podrán presentarse en etapas posteriores causadas por el mal entendimiento de las necesidades y objetivos del proyecto (Tabares, Anaya et al. 2007).

La ingeniería de requerimientos consta de las siguientes etapas cíclicas:

- Estudio de la viabilidad:
- Informe de la viabilidad
- Obtención de los Análisis de Requerimientos
- Modelado del Sistema
- Especificación de los Requerimientos
- Requerimiento del usuario del sistema
- Validación de los Requerimientos
- Documento de Especificación de los Requerimientos.

El presente estudio se basa en el paso de Modelado del Sistema y Especificación de Requerimientos, cuales son más complejos en esta etapa y en donde es relevante la separación de conceptos y la incorporación de aspectos.

Para maximizar la separación de asuntos se utilizan la segmentación por funcionalidades transversales, es decir funciones que se cruzan entre sí y que tienen características en común, las cuales se van a agrupar dentro de un módulo determinado, se incluye el tratamiento de aspectos en esta etapa, ya que los mismos permiten esta clasificación. La programación orientada a aspectos disminuye los problemas de código enmarañado y dispersos (Tabares, Anaya et al. 2007).

2.1. Programación Orientada a Aspectos

Reseña Histórica:

Para los años de 1991, la programación orientada a aspectos no existía. Dentro de los trabajos de Demeter y su grupo se encuentra la Programación Adaptativa (PA), que es considerada pionera de la POA, con algunas ideas similares: la descomposición de bloques constructores de cortes transversales.

En 1995, se separa la POA de la PA, surgiendo el concepto de aspecto como: "Un aspecto es una unidad que se define en términos de información parcial de otras unidades." (López, Asteusuain et al. 2005). Posteriormente, Lopes y Kickzales introdujeron el nombre de Programación Orientada a Aspectos.

La programación orientada a aspectos llega a ser un complemento a la programación orientada a objetos, debido a que puede encapsular conceptos y asuntos que no se podían anteriormente.

Complemento a la Programación Orientada a Objetos:

La programación orientada a objetos presenta las siguientes desventajas estructurales:

- Código disperso: el código no se encuentra encapsulado en un solo módulo sino que se presenta por todas partes.
- Código enredado: el código presente en un módulo tiene diversas funcionalidades, siendo requerido muchas veces por otras funciones.

A su vez, replican en las siguientes consecuencias:

- Dificultad en las tareas de mantenimiento, puesto que el código es difícil de comprender.
- Complejidad en las tareas de depuración y corrección de errores.

Frente a lo expresado de la POO, la incorporación de aspectos beneficia de la siguiente manera:

- Separación de asuntos en módulos denominados aspectos,

- Ahorra tiempo de diseño (donde se decide como será el software que se creará),
- Ahorra tiempo de depuración (el proceso de corrección de errores)
- Reduce tiempo de implementación (el proceso en sí de desarrollo y de programación),
- Mejora las tareas de mantenimiento (consiste en mantener el software actualizado) del software,
- Facilita las actividades de reutilización

Mediante estas cuatro ventajas, la implementación del proyecto orientado en POA reduce los costos y tiempo del proyecto de software y se incrementará la calidad, ya que se contará con un software más estructurado y de fácil mantenimiento.

Además de estas ventajas, al tratar los aspectos dentro de la etapa de requerimientos se obtienen otros beneficios y es por eso que las más recientes investigaciones se han enfocado en el estudio de la separación de asuntos en las primeras (Budwell and Mitropoulos 2008). Estos beneficios son:

- Gozar de los beneficios de la abstracción de asuntos desde la etapa de diseño del framework, lo que permite tener una mejor visión del proyecto en sí.
- Se anticipa la estructura de los aspectos y se analiza cómo será su comportamiento e integración con los demás elementos.
- Permite identificar con mayor facilidad los requerimientos y funcionalidades, determinando el comportamiento de cada uno de ellos a lo largo de su implementación.

Para aprovechar al máximo cada una de estas ventajas se deben identificar, clasificar y evaluar los aspectos desde las primeras etapas del proceso de análisis y diseño del software, es decir mediante un estudio a los requerimientos del sistema. Tanto los requerimientos funcionales como no funcionales pueden llegar a ser catalogados aspectos dependiendo de su grado de significancia.

El Paradigma:

Debido a que el paradigma de POA es reciente, el mismo está en continuo mejoramiento y presenta algunas desventajas entre las cuales están la complejidad del proceso de identificación de las funciones transversales, la ubicación de sus elementos puede tergiversar en algunas situaciones una funcionalidad provocando que el software desempeñe una acción incorrecta, algunas decisiones de diseño no se pueden tomar hasta la implementación, debido a la carencia de pasos estructurados que indiquen como hacerlo, entre otros inconvenientes.

En el tratamiento de la programación orientada a aspectos, el código donde se encuentran los aspectos se teje mediante un Weaver (Tejedor), es decir el Tejedor integra los aspectos en localizaciones específicas de software en el paso de pre-compilación para luego ser combinado con las clases obteniendo una clase tejida dando como resultado el ejecutable. En la Imagen 1 se presenta de manera gráfica la formación del ejecutable.

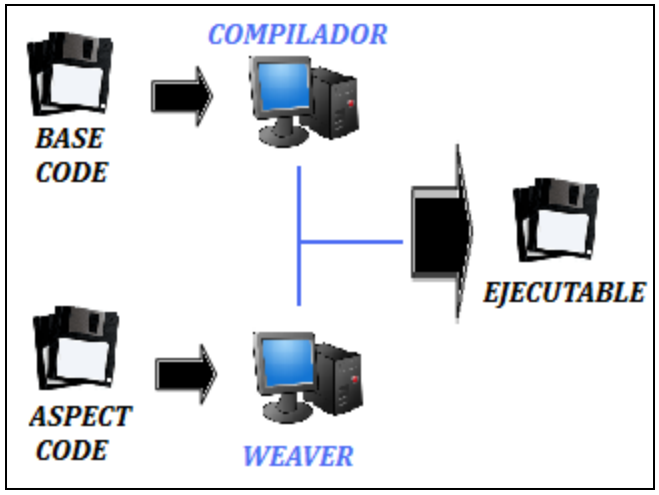


Imagen 1. Creación de ejecutables a través de la POA

Usualmente los aspectos son considerados como propiedades y requerimientos no funcionales de un sistema que no se alinean con los requerimientos funcionales, pero que tienden a cortar los componentes funcionales y aumentar sus interdependencias (Cai, Zhang et al. 2009), en conjunto con reducir la reutilización innecesaria de estas funciones (Debnath, Baigorria et al. 2008).

La separación de los asuntos y requerimientos del sistema en aspectos es una de los aportes más importantes del paradigma de POA (Hu, He et al. 2009), debido a esto se han creado disciplinas como lo son la AORE (Aoyama and Yoshino 2008), SLAI (Budwell and Mitropoulos 2008), AOCRE (Grundy 1999) y Modelos de Puntos de Vistas (Yu-Ning and Qiang 2009) que proponen mecanismos para el análisis de los aspectos en la etapa de Ingeniería de Requerimientos.

Los aspectos son funcionalidades transversales que se ejecutan de forma independiente del sistema. Presenta diferentes componentes:

- *JoinPoint (JP o Puntos de Unión): lugar* donde un aspecto puede ser colocado, el aspecto añadirá funcionalidad extra. Una correcta ubicación del JP permite que el aspecto desempeñe correctamente la función esperada. Son elementos dinámicos para el Tejedor.

- *CutPoint (CP o Puntos de Corte):* son los elementos donde se realizan las intervenciones a clases y métodos. Sirven para determinarles a los advice en que intervención deben ejecutarse. Son elementos dinámicos para el Tejedor.

- *Advice (AP o Consejeros):* define el comportamiento de un objeto frente a un evento determinado, permite manipular las invocaciones de métodos antes, durante o después de su ejecución. Son elementos dinámicos para el Tejedor.

- *Introduction (Instrucciones)*: permite la manipulación de clases, variables y datos existentes, durante la ejecución. Son elementos estáticos para el Tejedor. Estos componentes generalmente no se analizan en la etapa de requerimiento del software.

Las principales desventajas que presentan la POA (G. Cugola, C. Ghezzi et al. 1999) son las siguientes:

- Conflicto entre aspectos.
- Conflicto entre el lenguaje base y los aspectos
- Complejidad en el análisis orientado a aspectos.

De este último punto nacen las siguientes debilidades:

- Dificultad en la tarea de ubicar los JoinPoint dentro del código del framework que radica en la complejidad de la misma para lograr que el aspecto desempeñe la función esperada (Cazzola, Jézéquel et al. 2006; Hu, He et al. 2009).

- Problema estructural debido a frágil ubicación de los CutPoint (Nusayr 2008; Hu, He et al. 2009), que al presentarse este conflicto desencadena problemas para el mantenimiento y puede crear acoplamiento.

- Complejidad para tratar aspectos dinámicos (Jun-Wei, Rong et al. 2008), cuyos errores pueden provocar inestabilidad en el sistema. Dependiendo a la función de un framework en particular para evitar la interrupción durante su ejecución, los aspectos dinámicos pueden apagarse, modificarse o agregarse estando el mismo en funcionamiento.

- Conflictos por multifuncionalidad (Fernando Asteasuain and Contreras 2002; Jun-Wei, Rong et al. 2008) donde es posible encapsular una funcionalidad en diferentes aspectos, pero obteniendo diferentes diseños lógicos.

2.2. Desarrollo de Software Orientado a Aspectos

Debido a la complejidad en el tratamiento de código transversal nace la disciplina de AOSD (Desarrollo de Software Orientado a Aspectos) la cual fue creada para hacer frente a la dificultad del código transversal debido a su complejidad y su dificultad para entenderse.

La disciplina AOSD se enfoca en la identificación, separación, representación y la composición de los asuntos de carácter transversal (Budwell and Mitropoulos 2008). Estos asuntos son capturados en módulos independientes denominados aspectos.

De acuerdo con (Rashid, Sawyer et al. 2002) , algunas de las ventajas que proporciona la AOSD son:

- La reducción de desarrollo
- Permite un mantenimiento del software más eficiente.
- Reduce los costes de esfuerzo de desarrollo en todas las etapas del ciclo de vida.

Uno de los mayores problemas que existen en la AOSD se debe a las carencias de completas y sistemáticas metodologías, que a pesar que existen algunas de ellas no son aún completas (Budwell and Mitropoulos 2008), para el tratamiento de aspectos en las primeras etapas de ciclo de vida del software que se centren en la manipulación de requerimientos funcionales y no funcionales.

Enfoques de la Disciplina de Desarrollo de Software Orientado a Aspectos:

Dentro de la AOSD se ha diseñado importantes enfoques que se aplican en la Ingeniería de Requerimientos para el tratamiento de aspectos, siendo a su vez utilizados por algunas de las metodologías orientadas a aspectos. Entre estos enfoques cuya aplicabilidad es el tratamiento de los aspectos en la etapa de requerimientos y cuyo propósito es facilitar la localización transversal de las funcionalidades y la clasificación por intereses y composiciones (Londoño, Anaya et al. 2008; Guzmán 2009) , tenemos:

- Separación Multifuncional de Intereses.
- Aspectos en Modelos de Objetivos de Requerimientos.
- Identificación de Aspectos en Requerimientos
- Aspectos dentro del enfoque a Casos de Uso

Separación Multifuncional de Intereses (MDSOC)

Es uno de los primeros enfoques de la AOSD (Londoño, Anaya et al. 2008). MDSOC consiste en un proceso de segmentación por vistas. Tiene su fortaleza en la descomposición de reglas por intereses y en la forma de manejar los conflictos. Todas las funcionalidades se hacen de forma homogénea, agrupando los requerimientos funcionales y los no funcionales en las vistas.

Basados en (Dahiya and Sachdeva 2006), existen tres enfoques principales:

- *Puntos de vista de los interactuadores*: donde se considera a todos los actores o elementos que interactúan directamente con el sistema.
- *Puntos de vista indirectos*: considerando a los stakeholders que influyen de forma indirecta sobre los requerimientos del sistema.
- *Puntos de vista del dominio*: involucra las restricciones del dominio que repercuten sobre los requerimientos.

Aspectos en Modelos de Objetivos de Requerimientos (ARGM)

El propósito de ARGM es la segmentación recursiva de tareas, desde el grado más bajo hasta el nivel superior. En la descomposición se mide la calidad en base al cumplimiento de los objetivos, se detectan los conflictos y las funcionalidades; las cuales se vuelven candidatos para los aspectos. Su enfoque está dirigido a objetivos, identificación de aspectos en base a objetivos y la resolución de conflictos.

Identificación de Aspectos en los Requerimientos (Theme/Docs)

Theme / Docs se basa en la orientación a temas. Separa los requerimientos del sistema dependiendo a temas que representan asuntos de interés o funcionalidades. Inicia de una data de descripción textual para realizar un proceso de análisis gramatical entre los requerimientos y las acciones, lo que deriva los procedimientos con mayor granularidad.

Enfoque por los casos de uso (AOSD)

El objetivo de este enfoque es obtener los posibles aspectos a través del análisis de los casos de uso. Se representan los requerimientos no funcionales y funcionales en grupos diferentes y dependiendo al comportamiento se segmenta por cortes transversales y se determinan los aspectos por la funcionalidad base del caso de uso.

Cada uno de estos enfoques presenta ventajas en diferentes aspectos y debilidades en otros. La Tabla 1 presenta comparaciones entre los enfoques basados en (Londoño, Anaya et al. 2008):

Elemento	*Cualidades de cada enfoque*
Alcance	*ARGM*: Permite en todo momento mantener la relación entre los requerimientos y el estado del sistema. *MSDOC* y *Theme/Doc* crean espacios multidimensionales pero no toman elementos de calidad. Mientras que *AOSD/UC* toma atributos que calidad que no satisfacen completamente el enfoque deseado.
Trazabilidad	*MSDOC*: con una trazabilidad relacional entre intereses y requisitos y *ARGM*: con un sistema jerárquico entre objetos y tareas. *Theme/Doc* aplica trazabilidad directa por relación mientras que *AOSD/UC* utiliza un trazado UML.
Composición de Requerimientos	Todos ofrecen un enfoque sistemático de descomposición con diferencias de la forma como se realiza. *MSDOC* aplica reglas, *ARGM* utiliza una visión descendente desde las reglas de negocio hasta las metas más pequeñas. *Theme/Doc* detallas los requisitos para posteriormente formas los asuntos principales en un procedimiento ascendente. Mientras que AOSD/UC se basa en la representación de todos los elementos en los casos de uso.
Validación y Verificación	*MSDOC* y *Theme/Doc* utilizan validación a través de un proceso denominado multidimensonal para evaluar las matrices de asuntos. *ARGM* Aplica Heurística para la verificación de inconsistencias.
Mapeos	*MSDOC* propone una matriz de asuntos que descompone funciones, aspectos y decisiones. *ARGM* se basa en la descomposición de metas. Mientras que *Theme/Doc* utiliza reglas de mapeo que Theme/XML; y *AOSD/UC* hace un mapeo por UML basado en los casos de Uso.

Tabla 1. Comparación entre los Enfoques que involucran POA en la toma de requerimientos.

Estos enfoques están presentes en las metodologías para la AOSD (Punto 2.4. Metodologías Orientadas a Aspectos) de las cuales se basa la metodología propuesta y explicada en capítulos más adelantes.

2.3. Inconvenientes de la Programación Orientada a Aspectos

El paradigma de POA presenta algunas desventajas que radican en pequeños problemas debido a su naturaleza de cortes transversales, los cuales son los siguientes:

2.3.1 Conflicto Estructural:

Este inconveniente radica en las dificultades de conocer los objetivos, elementos y características de funciones y aspectos después de períodos prolongado de mantenimiento. Es producido debido a la difícil tarea de ubicar los puntos de corte en el framework, su presencia puede causar problemas de acoplamiento y dificultar las tareas de mantenimiento. Presentado en los trabajos de: Morocho (Morocho, Chuico et al.) , Hu (Hu, He et al. 2009), Cazzola (Cazzola, Jézéquel et al. 2006)

Este conflicto no debe confundirse con una de las ventajas de la POA, que es la facilidad de mantenimiento, evitando el código disperso. Estas dos características se diferencian debido a que el Conflicto Estructural radica en la pérdida semántica de la información de las funcionalidades aun manteniendo el código entendible y ordenado, que es la ventaja de la POA sobre la POO, ya que en el último, el código pierde ordenamiento y se dispersa o entreteje, haciendo difícil su comprensión.

2.3.2. Conflicto de Comportamiento:

Esta dificultad radica en la complejidad nata de los aspectos, en la ubicación de sus elementos dentro de la ubicación correcta del sistema, puesto que debido a su

característica de transversalidad, es más fácil caer en el error de ubicación, provocando ambigüedades a la hora de ejecución, provocando de esta manera que el aspecto realice tareas erradas y deficientes, causando conflictos dentro del framework. Presentado en los trabajos de: Hu (Hu, He et al. 2009), Nusayr (Nusayr 2008)

Se debe a la posibilidad de ubicar los puntos de unión en lugares no adecuados, causando ambigüedad en el desempeño del aspecto. La tarea de colocación es difícil de realizar.

2.3.3. Conflicto por Multifuncionalidad:

Este problema se origina también por la naturaleza de la POA. Como la POA pretende la encapsulación de asuntos (funcionalidades) en aspectos, esta tarea se empaña si este asunto es multifuncional, es decir si un asunto presenta varios objetivos y se utiliza en procesos totalmente diferentes, la tarea de encapsulación se dificulta causada por la carencia de reglas para decidir en qué aspecto se ubica la funcionalidad dada. Presentado en los trabajos de *Cugola (G. Cugola, C. Ghezzi et al. 1999), Asteausain (Fernando Asteasuain and Contreras 2002), Jun-Wei(Jun-Wei, Rong et al. 2008).*

2.3.4. Facilidades Dinámicas:

Una ventaja de la POA es que permite prescindir, activar y modificar aspectos en tiempo de ejecución, con el propósito de cambiar ciertas funcionalidades sin la necesidad de bajar el sistema lo cual es muy beneficioso, pero la tarea de decidir cuáles serán los aspectos considerados como dinámicos y la medición del impacto de los mismos es una tarea muy compleja. De allí, es que se prefiere omitir los beneficios de esta ventaja frente a los costes de desarrollo y análisis. Presentado en los trabajos de: Jun-Wei (Jun-Wei, Rong et al. 2008), Asteausain (Fernando Asteasuain and Contreras 2002)

Las metodologías, framework y modelos en los cuales se denotan las principales carencias tanto lógicas como estructurales se presentan a continuación:

- AORE (Tabares, Anaya et al. 2007; Aoyama and Yoshino 2008),
- AOCRE (Grundy 1999),
- SLAI (Budwell and Mitropoulos 2008),
- THAOP (Hwang and Choi 2007),
- Framework basado en Componentes y Aspectos (Zhang and Zhang 2005),
- Dynamic Weaver Framework (Fletcher, Akkawi et al. 2004),
- TITAN (Perez-Toledano and Canal 2007),
- Modelo Semántico de JoinPoint (Cazzola, Jézéquel et al. 2006),
- Modelo de Puntos de Vistas (Yu-Ning and Qiang 2009),
- Modelos Conceptuales (Hu, He et al. 2009).

2.4. Metodologías Orientadas a Aspectos

Las metodologías tratadas y analizadas para el desarrollo del proyecto fueron:

2.4.1. Ingeniería AORE

La Ingeniería AORE (Ingeniería de Requerimientos Orientada a Aspectos) permite el modelo del paradigma de orientación a aspectos identificando funciones transversales (Rashid 2008; Yu-Ning and Qiang 2009).

La metodología AORE utiliza formas de composición basados en el lenguaje de XML, la misma trata los asuntos como un conjunto coherente de requerimientos y donde los aspectos transversales son matrices conformadas por elementos que denotan funcionalidades (Tabares, Anaya et al. 2007), de esta forma es posible determinar cuándo las funcionalidades trabajan de manera negativa o positiva frente a otros asuntos; determinando de esta manera el grado de influencia que tienen unos asuntos sobre otros a través de la matriz de composición.

Según (Jingjun, Furong et al. 2007), la mayor cantidad de modelos desarrollados para AORE son basados en el enfoque Theme/Doc. La metodología

AORE, enfocada en el tratamiento y la abstracción de aspectos, cuenta con las siguientes etapas (Rashid, Sawyer et al. 2002; Tabares, Anaya et al. 2007):

Identificación de Asuntos:

La primera etapa consiste en la identificación de asuntos a partir de los requerimientos del sistema, se separa los requisitos funcionales y no funcionales del software para ser tratados de forma separa. El proceso de identificación en esta etapa se basa a través de la manipulación de diagramas de casos de uso, modelados por puntos de vistas, segmentos orientados a objetivos entre otros enfoques de AOSD.

Especificación de Asuntos:

Posteriormente, se evalúan la compatibilidad de cada uno de ellos como paso previo para su clasificación por funcionalidad. La especificación de aspectos está basada en la integración de los enfoques y los aspectos identificados según propiedades y cortes transversales.

Identificación y Especificación de Influencias:

Se utiliza la matriz de composición de elementos, que consiste en un alineamiento y clasificación de funciones y asuntos relacionados de forma 1:1 (uno a uno) o 1:m (uno a muchos) con otros asuntos, de tal forma que sea posible identificar aspectos candidatos por el número de dependencias de un asunto relacionado; este proceso es conocido como la regla de descomposición. La regla de descomposición de AORE utilice asuntos como elementos de aplicación, en donde se representan las dependencias e influencias de los mismos. El formato de apoyo para AORE es el lenguaje XML. En esta etapa se realizan las siguientes actividades:

- Descubriendo conceptos y su relación con los aspectos:

Se crea un enfoque de vistas en base a los requerimientos funcionales y no funcionales que un asunto determinado involucra. Dependiendo a la clasificación de funcionalidad a la que pertenece se determina el corte transversal (aspecto) de esa vista.

- Identificación de Aspectos candidatos:

Los aspectos son determinados por su presencia transversal en las diferentes vistas. Un aspecto llega a ser determinado candidato si el corte transversal asociado tiene un alto número de dependencias y de relaciones según la regla de descomposición.

- Especificación y priorización de aspectos:

Se redefine los aspectos determinando de una forma más sólida sus elementos y propiedades, este proceso es determinado como Especificación. Según el impacto que tienen los aspectos sobre la funcionalidad del software, dependiendo de la magnitud de la función, cantidad de métodos que intercepta, número de influencias, o si pertenece a un requerimiento funcional se prioriza, es decir, se determina como un aspecto verdadero (deja de ser candidato) y se clasifican dentro de su corte transversal.

Regla de Descomposición de AORE:

En esta fase se utiliza la regla de descomposición, conocida como reglas de tejido, son normas difíciles de aplicar cuyo propósito es la identificación y separación de aspectos; su impacto es enorme en la identificación de asuntos y funcionalidades en la etapa de diseño. El proceso de descomposición consta de los siguientes pasos:

- Identificación de la matriz de asuntos relacionada:

La matriz de asunto está compuesta por las funcionalidades identificadas en el proceso anterior, a partir de estos elementos se crear las relaciones por influencias y dependencias de los asuntos transversales del proyecto.

- Verificar la multiplicidad de los asuntos origen hacia los asuntos destino:

Se identifican las relaciones y se determina las influencias de un asunto y sobre un asunto determinado.

- Repetir para cada asunto origen y evaluar la cantidad de repeticiones de los asuntos destino:

Una vez determinadas todas las influencias, basados en la cantidad de dependencias se determina el grado de cardinalidad de los asuntos.

- Cuando el asunto aparece con un número de n veces, se identifica el asunto como un aspecto transversal:

Los asuntos con un número de 2 o más en su cardinalidad pasan a ser analizados culminando con la identificación de un aspecto candidato cuyo corte transversal envuelve el dominio de la relación.

En la Imagen 2 (Tabares, Anaya et al. 2007) es posible visualizar un ejemplo de multiplicidad de asuntos en donde el asunto RB4 es un requisito transversal puesto que dos asuntos RA1 y RA3 lo influyen directamente.

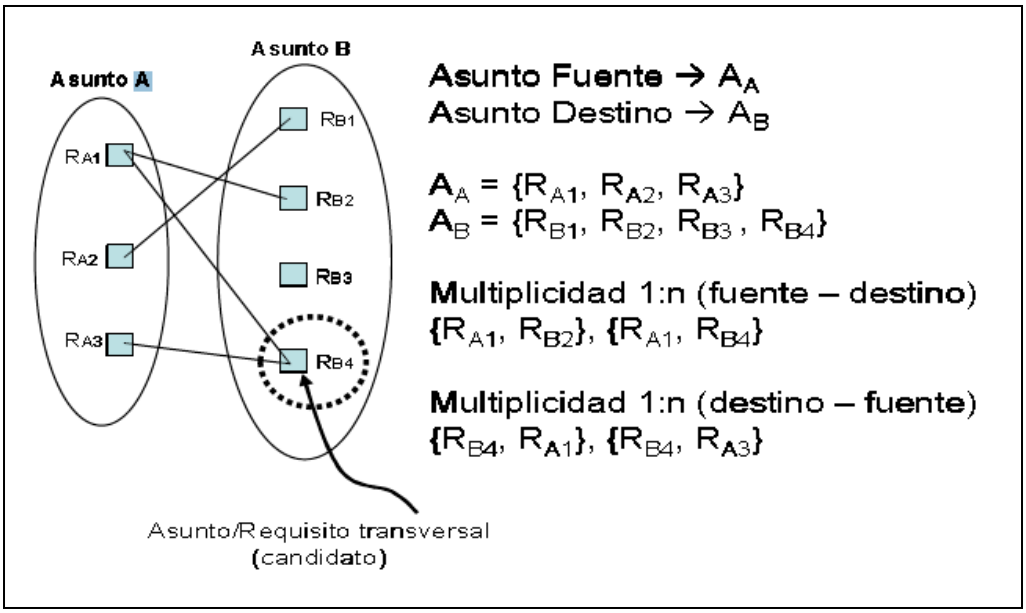


Imagen 2. Representación de Multiplicidad de los asuntos.

Manejo de Conflictos entre asuntos:

Conforman las actividades que se llevaran a cabo para la solución de conflictos que surgen por las relaciones entre los asuntos.

Especificación de las dimensiones de los asuntos:

Define como los aspectos serán soportados en la arquitectura según su influencia y mapeo. Se analiza como los aspectos se comportarán en las siguientes

etapas de la Ingeniería de Software, en especial frente al ciclo de cambio y redefinición de requerimientos; para este punto es necesario tomar en consideración los aspectos, las funciones y las decisiones para cada etapa del ciclo de vida. Además, los aspectos identificados pasan por una última etapa de revisión, lo cual permite diferenciarlos de funciones, decisiones o aspectos reales.

- Especificación de dimensiones de los aspectos:

Se consideran los aspectos identificados según el ciclo de vida del software. Esta clasificación se realiza por aspectos, Influencia y Mapeo. Ejemplo:

Aspecto Candidato	Influencia	Mapeo
Autenticación	Arquitectura Diseño	Aspecto
Sistema de Usuario Múltiple	Arquitectura Diseño	Aspecto
Aspectos Legales	Especificación de Requerimiento	Función

Tabla 2. Representación del dimensionamiento de datos (Rashid, Sawyer et al. 2002)

En este proceso evolutivo, AORE agrega elementos de toma de decisiones frente a grupo de participantes (stakeholders), debido a que es necesario identificar elementos como inseguridad, dudas sobre ciertos requerimientos para crear medidas preventivas y planes de acción para posibles cambios en ciertos requerimientos.

2.4.2 Modelo por Puntos de Vistas:

La Ingeniería de Requerimientos tratadas desde el enfoque de Orientación a vistas para la Integración a Aspectos introduce la conceptualización de aspectos dentro de los casos de uso (Rashid, Sawyer et al. 2002).

Basados en el AOSD se han aplicado casos de estudios basados en el enfoque por vistas, dentro de ellos se encuentra el proyecto Visión que integra nuevos elementos al AORE.

Según (Yu-Ning and Qiang 2009), uno de estos modelos enfocados a puntos de vistas es: Visión el cual también agrega componentes UML para la representación y la integración de aspectos. La unión entre estos dos enfoques crea una estructura más sólida para el tratamiento de los requerimientos.

Las principales etapas del Modelo de Visión, Se muestran en la Imagen 3 (Yu-Ning and Qiang 2009), son las siguientes:

- Solicitud de Requerimientos.
- Formato de Requerimientos.
- Identificación de puntos de vista.
- Definición de características y objetivos.
- Definición de casos de uso.
- Identificación de Requerimientos funcionales.
- Resolución de Conflictos.
- Identificación de Aspectos.
- Tratamiento de Conflictos.

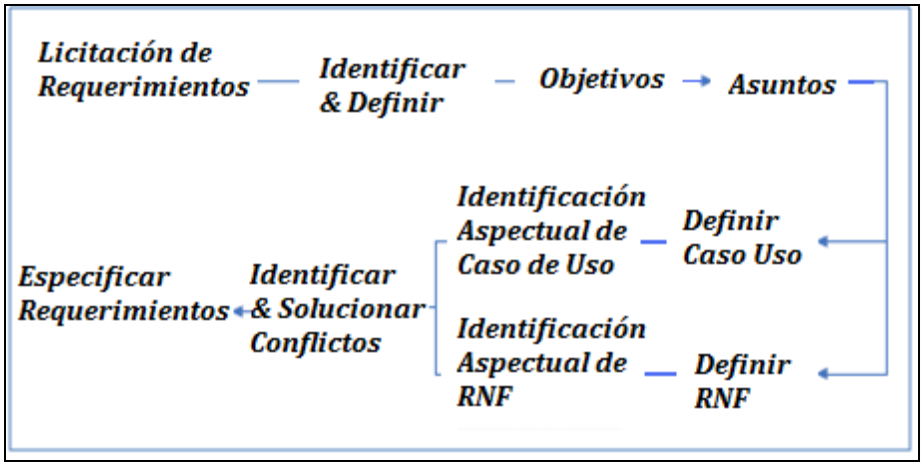


Imagen 3. Modelo de Procesos de Visión.

Solicitud de Requerimientos

La forma de obtención de Requerimientos se realiza mediante entrevistas con los usuarios, cuestionarios y reuniones (este procedimiento no es detallado debido a

que no impacta fuertemente con la metodología de identificación y especificación de aspectos).

Dar formato de los Requerimientos:

Este proceso consta de varias sub etapas:

- Identificación de Puntos de Vista:

El modelo por Puntos de Vistas (Viewpoint Model) detalla elementos de funcionalidad en forma de vistas, es decir información referente a un enfoque determinado dependiendo de usuarios, y objetivos.

Como se muestra en la Tabla 3, El modelo por Puntos de Vista (Yu-Ning and Qiang 2009) incluye elementos como nombre (identificador), objetivos (el propósito de la vista), stakeholders (usuario que proveyó la información), historias (seguimiento de los cambios), características (consistencia), modelo de casos de uso y requerimientos no funcionales.

ViewPoint	Objetivo	StakeHolder	Historia	Características	Caso de Uso

Tabla 3. Elementos considerados en un Modelo por Puntos de Vista[1].

- Definición de objetivos y definición de características:

Asocia las metas y los objetivos a cada una de las vistas. Los objetivos se basan en propiedades y en características. Mediantes la definiciones previas es posible especificar la funcionalidad asociada a cada vista.

- Definición de Casos de Uso:

Con las características identificadas se crean los casos de uso, los mismos pueden tener conflictos con otros debido a que uno de ellos puede estar asociado a varias vistas, varias características o puntos de vistas.

[1] La manera como se rellenan las tablas del Enfoque a Vistas puede observarse detalladamente en el Caso de Aplicación del Capítulo V

- Definición de Requerimientos no Funcionales.

Elementos que no son cubiertos por los pasos anteriores son los requerimientos no funcionales. La definición de los mismos se asocia a la definición de casos de uso, puntos de vistas y objetivos. Estos elementos también pueden presentar conflictos entre sí debido a enfoques y funcionalidades.

Resolución de Conflictos:

Se aplican varios procesos: El enfoque de Casos de Uso y por Requerimientos No funcionales, un corte transversal representa una necesidad funcional, como se encuentran definidas las características es fácil este proceso. En este punto se resolverán los conflictos que se presentan en las funcionalidades transversales.

2.4.3. Ingeniería AOCRE

La metodología AOCRE (Ingeniería de Requerimientos en Componentes Orientadas a Aspectos) se enfoca a la identificación y especificación de requerimientos funcionales y no funcionales relacionando como llave “aspectos” de cada uno de los componentes de un sistema (Grundy 1999).

Según (Grundy 1999), su procedimiento se basa en la descomposición de aspectos en “detalle de aspectos”, por ejemplo: Un Aspecto que controle el asunto de Interfaz de Usuario puede ser descompuesto en Vistas, Calidad frente a las acciones del usuario, mecanismo de retroalimentación, escalabilidad y extensibilidad.

El procedimiento de AOCRE consiste de los siguientes pasos:

- Análisis general de los requerimientos de la aplicación o de grupo de componentes.

Los requerimientos son analizados bajo el modelo iterativo de especificación arriba – abajo (se ven las funcionalidades como un todo y se va segmentando hasta obtener requisitos jerarquizados y detallados), que cual permite la granularidad en la definición de los requisitos.

- Especificación de requerimientos:

Se determinan los componentes de los aspectos, funcionalidades requeridas y proporcionadas por un aspecto, propiedades funcionales y no funcionales. Los componentes son refinados en los detalles de los componentes.

- Evaluación de escenarios y reusabilidad

Mediante un procedimiento cíclico los aspectos llegaran a formar parte de los diferentes componentes del sistema.

Como se ha visto, AOCRE cuenta con etapas similares a AORE: Identificación y Descripción de Requerimientos, Razonamiento con Aspectos que consiste en el análisis de los requerimientos identificando aspectos y clasificándolos por componentes y luego el Diseño e implementación en tiempo de ejecución en donde se evaluaran los aspectos y su comportamiento según el razonamiento aplicado en las etapas posteriores. Siendo su diferencia el enfoque a detalles de aspectos en donde se buscar tratar de determinar los aspectos según los componentes asociados al mismo.

2.4.4. Metodología SLAI

La Metodología SLAI (Léxico Estructurado para la Identificación de Aspectos) consiste en la identificación de aspectos potenciales en la fase de diseño de proyectos, es decir, en SLAI los aspectos son identificados y especificados en la fase de requisitos donde se detallan los requisitos funcionales y no funcionales (Budwell and Mitropoulos 2008).

Este procedimiento de lleva a cabo a partir de la abstracción de los aspectos a partir de Casos de Uso y un procedimiento de conteo léxico de identificadores cuya frecuencia determinará los posibles aspectos candidatos.

El objetivo primordial de la metodología SLAI es el de servir de cómo herramienta sistemática para la captura las funciones transversales funcionales y no funcionales como aspectos (Budwell and Mitropoulos 2008).

SLAI trabaja con el enfoque a casos de uso, mediante un procedimiento de división según las influencias y dependencia de los casos de uso. La clasificación es la siguiente:

- *Casos de Uso de Inclusión,* que consiste en un caso de uso que en conjunto con los que incluye especifican una funcionalidad específica obligatoria de ejecución;
- *Caso de uso de Extensión,* consiste en un caso de uso que en conjunto con los que extiende le da alternativas al sistema en sus procesos de ejecución.
- *Caso de uso Transversal*: representa un caso de uso de corte transversal, es decir que soporta a muchos casos de uso, este es el más importante del enfoque debido a que cumplen con la función del aspecto y a partir de los mismos se determinan los aspectos candidatos.

La metodología SLAI trata a los requerimientos funciones y no funcionales de dos diferentes formas:

Requerimientos Funcionales:

Los requerimientos funcionales son tratados con la siguiente metodología de identificación de aspectos:

- Identificación de los actores.

Usuarios, sistemas o componentes que interactúan con el sistema.

- Se guarda los verbos utilizados en los casos de uso.
- Se crea un repositorio de verbos.

Se registran los verbos e identificadores claves en una base de información, de manera que sea posible contabilizar la frecuencia de cada uno de ellos.

- Contabilización de identificadores para cada caso de uso existente o nuevos productos del proceso cíclico.

Cuando se agregan nuevas funcionalidades mediante los casos de uso, se verifican si existen similitudes en el repositorio, de ser así, se van seleccionando estos como aspectos candidatos.

- Se contemplan los aspectos para nuevos casos de uso.
- Se repite el procedimiento para optimizar los casos de uso.

Se hace el procedimiento cíclico para cada optimizar el diseño del sistema en relación a la integración de aspectos. La reutilización evita que se utilicen diferentes términos para la misma determinar la misma acción.

Requerimientos No Funcionales:

Los requerimientos no funcionales son tratados con la siguiente metodología de identificación de aspectos:

- Se utilizan requerimientos no funcionales preestablecidos como acceso al sistema, acceso a la información, usabilidad.
- Se evalúan los requerimientos proporcionados por los usuarios.
- Se identifican elementos no funcionales del sistema mediante experiencia o términos proporcionados por los usuarios.
- Se convierten en casos de uso.

Luego se aplica un procedimiento similar al procedimiento de requerimientos funcionales.

El procedimiento para la identificación de Aspectos que utiliza la metodología SLAI se basa en el corte transversal de los casos de uso mediante el uso de una tabla de descomposición en la que se refleja el sesgo transversal de los elementos aspectuales sobre las interfaces y los casos de uso sobre los otros casos de uso. Los elementos de esta tabla definen los aspectos, los requerimientos que estos afectan, sus condiciones, reglas que actúan sobre el mismo y los pasos en los que influyen.

SLAI utiliza el siguiente procedimiento ontológico para diferenciar los casos de uso dependiendo a las funciones de <include>, <extend> y <cross-cutting> las cuales apoyan respectivamente, a otro caso de uso para especificar funciones, se apoya en otro caso de uso para proporcionar rutas alternativas al proceso y como asistente para la determinación de funciones transversales al apoyar más de un caso de uso.

2.5. Modelos de Tratamiento de Inconvenientes

A continuación se presentan un resumen de los modelos desarrollados para el tratamiento de aspectos.

Modelo Theme / Docs

Theme/ Docs utiliza la orientación a temas. Separa los requerimientos del sistema según los temas que representan asuntos de interés. Se basa en procedimientos de análisis léxico para la separación en asuntos bajo el concepto de la POA (Rashid 2008).

Modelo de Casos de Uso

El modelo de Caso de Uso trata los requerimientos no funcionales mediante los casos de uso que representan la unidad mínima del sistema, mientras que los requerimientos no funcionales se ven como casos de uso de infraestructura lo que analiza el comportamiento e identifica los puntos de cruce de los casos de uso base (Londoño, Anaya et al. 2008).

Modelo de Puntos de Vista

El modelo de Punto de Vista se basa en la separación multifuncional de intereses utilizando el enfoque de vistas mediante reglas de descomposición, de definición y manejo de conflictos (Londoño, Anaya et al. 2008).

Modelo de Tejidos

Model-Driven base Aspect-Oriented Model Weaving (MAMW) es un nuevo modelo de concepción que transforma el modelo de la Programación orientada a aspectos. El MAMW es utilizado para la transformación en meta-modelos que implementará el nivel de abstracción.

Utiliza un modelo de vínculos para contener los Join Points y elementos de información de la sintaxis. MAMW mejora el nivel de abstracción y unifica el modelo de la Programación POA.

El procedimiento de Tejido (Weaving Model) es un modelo que modifica el programa original con el fin de tejer los aspectos y asociarlos con los Join-point correspondientes (Majumdar and Bhattacharya 2009). Weaving no es más que el procedimiento de integración de advices a sus correspondientes Join-points del código principal. Los aspectos son conformados por los Join-points, Cut-points y los advices.

El proceso de Tejido teje varios segmentos de código antes de la compilación, no permite la modificación dinámica del núcleo, ni de los módulos si contiene aspectos. Entre sus desventajas está la reutilización de código para el conjunto de aspectos ya que están separados a los Join-point del programa principal (Majumdar and Bhattacharya 2009).

2.6. Frameworks Basados en Programación Orientada a Aspectos

A continuación se describen Framework orientados a programación orientada a aspectos.

2.6.1. Framework basado en Componentes y Aspectos

Es un framework cuyo propósito es el de combinar la estructura de los componentes y los aspectos dentro del mismo modelo (Zhang and Zhang 2005). El modelado de aspectos se hace de forma estática y dinámica.

Su arquitectura está dividida en tres capas:

- Componentes y Aspectos: Los componentes son relacionados con los requerimientos funcionales y los aspectos con los requerimientos no funcionales del sistema.

- Interfaz e información: Las interfaces son divididas en entrada y salida.
- Conexión: Es la capa que permite interconectar a los aspectos y a los componentes manteniendo la unidad estructural del sistema.

Los componentes se encuentran a través de la red, mientras que los aspectos están desarrollados en el servidor. Como todo framework el área de aplicabilidad está limitada a los ambientes que soporten su estructura, cuya implementación, según el autor, resulta práctica para su implementación.

La metodología a desarrollar incorpora mecanismos de construcción dinámica en tiempo de ejecución de los aspectos y métodos a invocar y ejecutar. También se aplica el modelo conceptual para evitar los errores comunes de los POA Frameworks al momento de la expansión del código con la implementación de una vista lógica y física de las relaciones.

2.6.2. Framework basado en el Modelo Conceptual

Este framework tiene como propósito formar paquetes y métodos de clases, metadatos para la expresión de conceptos, funciones lógicas del software en conjunto con información física.

Este framework basado en el modelo conceptual utiliza un modelo basado en sintaxis XML para conectar la información física y la información lógica; resultando fácil aplicar ontología para lograr encontrar mediante instrucciones (querys) la información física mediante la referencia a la información lógica y viceversa (Hu, He et al. 2009).

Tiene como aplicabilidad solucionar el problema de los débiles Puntos de Corte mediante la utilización de meta niveles, meta-clases que están vinculadas con las clases y que son generadas por la reflexión de la clase la cual es un aspecto.

2.6.3. Framework de Tejido Dinámico

The Dynamic Weaver Framework (DWF) es un framework orientado a aspectos independiente de lenguaje, separa propiedades del sistema como autenticación, seguridad, calendarización, entre otras funcionalidad (Fletcher, Akkawi et al. 2004).

Este framework incorpora el dinamismo al concepto de la POA, los aspectos pueden ser agregados, modificados y removidos en tiempo de ejecución sin la necesidad de reconfigurar, bajar el sistema o detener la actividad del mismo, brinda una solución al problema presentado anteriormente.

Los elementos característicos del mismo es el repositorio de aspectos donde se almacenan todos los aspectos candidatos para convertirse en aspectos dinámicos, su ubicación se determina mediante una llave hash; además de poseer un tejedor de aspectos que es quien intercepta las invocaciones en tiempo de ejecución.

Permite aplicar la reusabilidad, puesto que el acoplamiento entre los aspectos es mínimo; existe un alto nivel de abstracción de conceptos a aspectos y en la codificación no existen restricciones, su arquitectura es independiente del lenguaje. Permite separar el código completamente de los aspectos.

2.6.4. Framework THAOP

Es un framework liviano orientado a aspectos que utiliza la técnica de tejido de aspectos. THAOP consiste principalmente de Librerías POA, Especificación de Lenguaje de Definición de Aspectos ADLS (Hwang and Choi 2007). Es un framework liviano lo que lo hace mucho más flexible que Fb-AOP más complejos y facilita la forma de describir aspectos, está basado en XML.

EL framework THAOP utiliza una estructura similar a la de AspectJ basada en puntos de corte, puntos de unión, advice y otros elementos de los aspectos. La creación de un nuevo framework basado en aspectos utilizando un componente

TH_CORE, el cual se ejecuta en tiempo de ejecución como una serie de herramientas, incluyendo un compilador.

THAOP consiste principalmente de Librerías AOP, Especificación de Lenguaje de Definición de Aspectos ADLS. Está basado en Join-Point, Puntos de Corte, Advice, Interceptores (similares a los advice). Estos modelos tiene funcionalidad es similar a AspectJ.

2.6.5. Framework TITAN

TITAN es un framework que permite el modelado de sistemas y la integración de aspectos (Perez-Toledano and Canal 2007).

EL framework TITAN se enfoca a actividades de modelado UML utilizando IPS (Patrones Específicos de Interacción por sus siglas en inglés) y así describir el comportamiento de los aspectos.

TITAN utiliza un procedimiento de integración de aspectos, cuando un aspecto no se encuentra determinado en las primeras etapas de ingeniería y es necesaria la adición del mismo, este puede agregarse a partir de un punto de unión, de tal forma que sea posible controlar su ejecución mediante un advice interrumpiendo el proceso original (Perez-Toledano and Canal 2007).

La arquitectura de TITAN sigue un procedimiento basado en los siguientes elementos (Perez-Toledano and Canal 2007):
1) Descripción del sistema (IPS) y especificaciones UML.
2) Instancias de Patrones.
3) Validación de especificaciones UML
4) Revisión y validación del modelo.
5) Simulación, verificación y prueba.
6) Verificación del comportamiento del sistema frente a nuevos aspectos.

Propósito: permite modelar a través de aspectos procedimientos y diseños UML.

2.7. ASPECTJ

El framework de ASPECTJ es una extensión al lenguaje de programación Java. Son librerías y extensiones de código del lenguaje Java que están orientadas a aspectos (Eclipse 2012). Estos módulos captan la esencia de los Join-point, Cut-Point, código adicional (Advice), weaving process (Objetos creados en el point-cut al aplicar instanciación de clases).

AspectJ es uno de los modelos de POA más populares en la actualidad, permite la ejecución de programas bien definidos en concepto de organización de programación y ejecución de aspectos (López, Asteusuain et al. 2005; Kumar, Kumar et al. 2008).

AspectJ implementa una serie de complementos Java para trabajar los Join-Point, descripción de los Cut-Point, implementa el uso de advice, introduction y aspectos (Kumar, Kumar et al. 2008).

AspectJ trabaja de la siguiente forma:

- **Join-Point:** El mecanismo de invocación de métodos, ejecución, constructores y manejadores de excepción orientados a aspectos forman parte del Join Point.

- **Cut-Point:** Maneja operaciones booleanas para determinar y marcar los puntos para activar los Join Point, estas llaves se encuentran colocadas antes, durante y después de una llamada a un Join Point, de tal forma que son un punto importante en la flexibilidad y para controlar el flujo de un módulo o método.

- **Los Cross-Cutting:** permiten agregar conceptos y módulos de seguridad en los mismos, puesto que estos son los puntos de acceso y de vínculo para la ejecución de las demás funciones del framework (Shah and Hill 2003).

- **Advice:** Son las instrucciones de programación que se ejecuta antes, durante y después la intervención de un Join-point. Se ejecuta cuando un Join-point es alcanzado, de esta forma es capaz de agregar un comportamiento extra a la funcionalidad. Los advice se definen para cada Cut-Point.

- **Introduction:** permite a los aspectos modificar la estructura estática de un programa. La introduction permite la adición de variables, clases, métodos, declaraciones y el control de excepciones dentro mientras se ejecuta un advice.

El principal objetivo de la AspectJ es proporcionar que mediante las librerías, los métodos e instrucciones orientadas a objetos y a aspectos, tanto como los bloques secuenciales de código trabajen acopladamente a la par sin problemas al momento de la ejecución del software.

El uso de AspectJ puede crear sistemas imprevisibles (Perez-Toledano and Canal 2007), efectos secundarios no deseados provocados por la incorporación de aspectos, weaving parcial, alteración semántica de los advice, conflictos con los estados invariantes del sistema al adicionar un nuevo aspecto.

La aplicabilidad de AspectJ permitirá la incorporación de la POA dentro del Framework mediante la utilización de la herramienta que se pretende diseñar, permitiendo el uso de cada uno de los elementos de este paradigma de programación. Existen otros framework similares a ASPECTJ, pero este fue utilizado debido al contante desarrollo que ha existido en el diseño del mismo, siendo uno de los más robustos y completos (Vanhaute, Win et al. 2001).

Estructura de los elementos de los Aspectos

A continuación se expresan las definiciones de los elementos de los aspectos que se utilizó para la generación del framework mediante la herramienta de apoyo. AspectJ permite la creación de cortes primitivos a través de varias funciones las cuales se presentan en la Tabla 4.

Instrucción del Corte	Función
Call (patrón)	Captura todas las llamadas de los JP cuyo constructor encaje con el establecido.
Execution (patrón)	Captura todas las llamadas de los JP cuyo constructor encaje con el establecido
Get (patrón)	Captura todos los JP cuyo atributo coincida con el patrón.
Set (patrón)	Captura todos los JP cuya asignación a un atributo coincida con el patrón.
This (patrón)	Captura todos los JP cuyo objeto ligado al this() es una instancia de la clase del patrón.
Target (patrón)	Captura todos los JP cuyo objeto objetivo es una instancia de la clase del patrón.
Args (patrón)	Captura todos los JP donde los argumentos son instancias de una clase que coincida con el patrón.

Tabla 4. Definición para los Puntos de Cortes

A continuación se muestran las posibles definiciones para los aspectos utilizando las librerías de AspectJ según los trabajos de recopilación del paradigma de Asteasuain (Fernando Asteasuain and Contreras 2002).

```
<Aspecto>::=[privileged] aspect <Identificador>
      [extends <Nombre_Clase>]
      [implements <Lista_Clases>]
      [dominates <Lista_Clases>]
{
  {<Atributos>}
  {<Métodos>}
  {<Def_Cortes>}
  {<Introducciones>}
  {<Aviso>}
 }

<Def_Cortes>::= abstract [<Modificadores>]
          pointcut <Identificador>(<Parametros_formales>); |
```

```
    [<Modificadores>]
     pointcut  <Identificador>(<Parametros_formales>):<Corte>;

<Introducciones>::=<IntroduccionesdeAtributos>  |
        <IntroduccionesdeMetodos>  |
        <IntroduccionesdeMetodosAbstractos>|
        <IntroduccionesdeConstructores>

<IntroduccionesdeAtributos>::=
[<Modificadores>] <Nombre_Clase> <PatrondeClase>.<Identificador>
 [= <expresión>] ;

<IntroduccionesdeMetodos>::=
[<Modificadores>] <Nombre_Clase>
<PatrondeClase>.<Identificador>(<Parametros_formales>)
  [ throws <Lista_Clases>] {<Cuerpo>}

<IntroduccionesdeMetodosAbstractos>::=
abstract [<Modificadores>] <Nombre_Clase>
<PatrondeClase>.<Identificador>(<Parametros_formales>)
  [ throws <Lista_Clases>] ;

<IntroduccionesdeConstructores>::=
[<Modificadores>] <PatrondeClase>.new(<Parametros_formales>)
  [ throws <Lista_Clases>] {<Cuerpo>}
<Aviso>::=<Tipo_aviso>  : <Corte> {<Cuerpo>}
<Tipo_aviso>::= <Aviso_antes> | <Aviso_despues> | <Aviso_durante>

<Aviso_antes>::=  before (<Parametros_formales>)
<Aviso_despues> ::= after (<Parametros_formales>)  <Forma_terminacion>
<Forma_terminacion>::=  returning  [(<Parametro_formal>)  ] |
                              throwing   [(<Parametro_formal>)  ] |  λ

<Aviso_durante>::=  <Nombre_Clase> around (<Parametros_formales>)
                         [ throws <Lista_Clases>]
```

Y finalmente, las definiciones de los avisos:

```
<Aviso>::=<Tipo_aviso> : <Corte> {<Cuerpo>}
<Tipo_aviso>::= <Aviso_antes> | <Aviso_despues> | <Aviso_durante>
<Aviso_antes>::= before (<Parametros_formales>)

<Aviso_despues> ::= after (<Parametros_formales>) <Forma_terminacion>
<Forma_terminacion>::= returning [(<Parametro_formal>) ] |
                       throwing [(<Parametro_formal>) ] |
                       λ

<Aviso_durante>::= <Nombre_Clase> around (<Parametros_formales>)
                   [ throws <Lista_Clases>]
```

2.8. Otros medios para determinar Asuntos Transversales

En la toma de requerimientos es necesario tomar en cuenta la Seguridad del sistema como asunto transversal, en especial factores de confiabilidad, integridad y disponibilidad (Kim and Lee 2008), de esta manera

Para implementar la seguridad es necesaria considerar:
Una estructura Vistas-Aspectos-Componentes bien establecida identificando puntos de vulnerabilidad. Es necesario considerar el funcionamiento de los aspectos no sólo para el diseño sino también para las etapas implementación y puesta en marcha.

Los involucrados deben conocer los cambios que ocurren en el software: diseño y análisis al incorporar los aspectos dentro del software, sellando las posibles vulnerabilidades.

Estos métodos incluyen la utilización de diagrama de casos de uso, identificación y representación de aspectos mediante extensiones, inclusiones y

generalizaciones (Guo, Teng et al. 2007). Incluyen diagramas de actividades y de secuencia tomando en consideración los aspectos.

Capítulo 3

Metodología Propuesta: MEDFOAR

MEDFOAR (Metodología para Diseñar Framework Orientada a Aspectos en Requerimientos) es una propuesta sistemática para la identificación y especificación de aspectos en la Ingeniería de Requerimientos enfocada para el diseño de Framework genéricos.

MEDFOAR recopila algunos beneficios de otras metodologías de orientación a aspectos integrándolas, además de plantear un mecanismo para minimizar los inconvenientes y desventajas que presentan por naturaleza la misma. (Estas desventajas se presentan en el capítulo II). Estos conflictos radican en la naturaleza de la POA tanto a nivel estructural como en dificultad de aplicación. La metodología combate los inconvenientes referente a su dificultad lógica.

La metodología incorpora diferentes etapas para la determinación de aspectos y sus elementos, algunas de las fases se basa en metodologías como AORE, AOCRE, ViewPoint Model y la metodología SLAI. La principal diferencia entre **MEDFOAR** y las metodologías presentadas radica en que la misma combate todos los inconvenientes de la POA, incorpora el proceso dentro de las etapas de básicas de la Ingeniería de Requerimientos, además de brindar una metodología con estructura donde elementos de un aspecto quedan determinados.

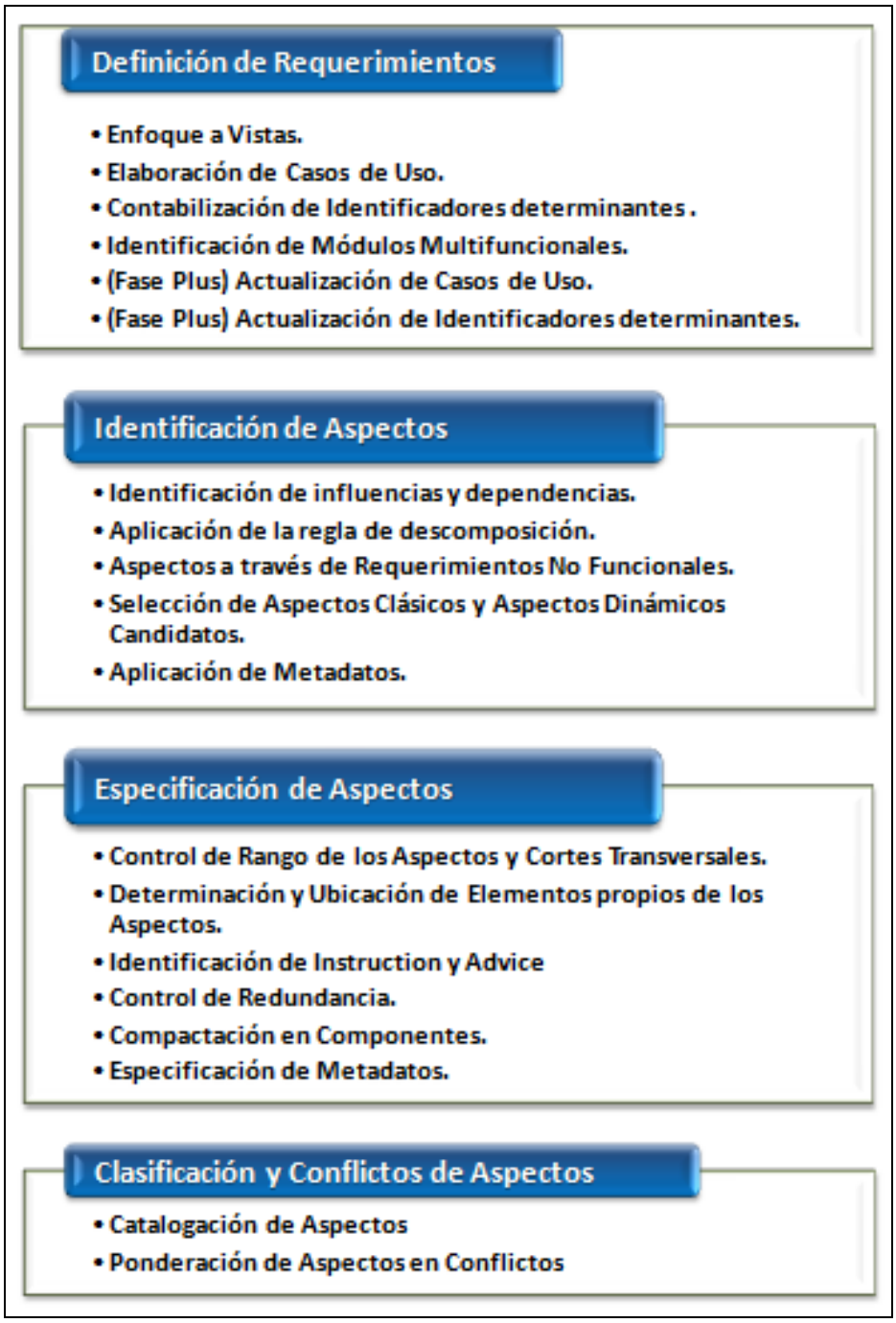


Imagen 4. Principales etapas de la Metodología MEDFOAR. [2]

[2] (Fase Plus) hace referencia a etapas que varían después de su primera ejecución cíclica.

La metodología se enfoca en los problemas lógicos en el análisis, por causa de esto, el proceso de identificación de aspectos está basado en esquemas y modelos de Casos de Uso, Puntos de Vistas, Enfoque a Objetivos, Identificación Léxica y Control de Influencias y Dependencias en el flujo del sistema. También, está enfocada a la identificación de aspectos y sus elementos de manera posicional y lógica, si detallar la funcionalidad en sí del aspecto, sino los cortes del aspecto, los cortes de sus puntos de cortes, la asociación de sus puntos de unión y el momento más indicado para la ejecución de un advice.

La metodología utiliza cuatro grandes elementos en su análisis[3]:

- CORTES: hace referencia a una función macro.

Para eliminar ambigüedad es necesario destacar que la metodología hace referencia a “cortes transversales” que representan los cortes a través de diferentes partes del programa (aspectos).

- ASPECTO: elemento base de la POA.

La metodología trabaja con los puntos de corte, puntos de unión, corte asociados, y advice según tiempo de ejecución. Debido a la gran variedad de definiciones para estos elementos, la herramienta utiliza execution() al generar el código del case para los puntos de corte.

- CLASES: elemento de la POO.

Dentro de la metodología las clases sólo son utilizadas para evaluar prioridades en los aspectos, y especificación de JoinPoint e Instructions.

- FUNCIONES: son las funcionalidades que en conjunto forman un objetivo cuantificable.

La Metodología propuesta MEDFOAR plantea cuatro etapas (Ver Imagen 5) para la identificación y el tratamiento de los aspectos, este proceso es cíclico para evitar posibles redundancias al momento de analizar los aspectos y como optimización al proceso de especialización.

[3] La relación de estos elementos pueden verse claramente dentro del capítulo correspondiente a la herramienta de apoyo.

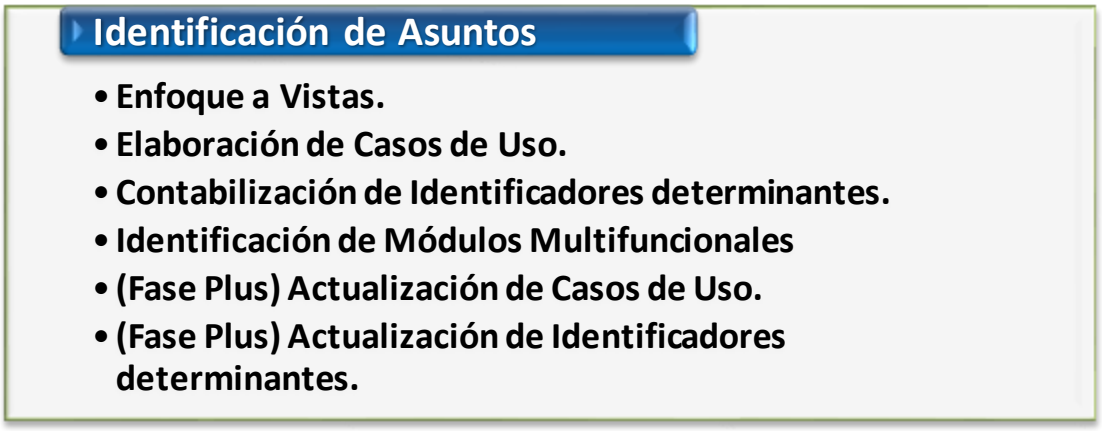
Imagen 5. Etapas de la Metodología

Las fases son las siguientes:

3.1. Identificación de Asuntos

La primera etapa del proceso consiste en la identificación de los elementos que tendrá el framework, estos elementos incluyen los actores o entes involucrados de alguna manera con el software, las funcionalidades que tendrá el sistema. La determinación de las clases, componentes puede realizarse, pero no es indispensable. También se denota como Definición de Requerimientos.

***Justificación**:* dentro del proceso rutinario en la Ingeniería de Requerimientos se realiza la identificación de elementos y funciones, para posteriormente analizar las rutinas, obtener las clases, y definir ciertos elementos con más prioridad que otros. A su vez se intenta incluir el estudio de los aspectos dentro del análisis estándar, lo cual minimiza el impacto en el tiempo para estas tareas.



3.1.1. Enfoque a Vistas

En esta etapa se utiliza el enfoque de orientación a vista por objetivos, dependiendo de los requerimientos funcionales del sistema y de los actores involucrados, se definen las vistas a partir de los escenarios posibles en el framework.

Los elementos de los esquemas orientados a vistas que se determinan en esta etapa son los siguientes:

***Nombre de la Vista**:* es un identificador que nombra el esquema.
Actores Involucrados: los entes que interactúan en él o con el sistema dentro de la vista seleccionada, los procesos semiautomáticos son tomados en cuenta en esta clasificación.
Funcionalidades Asociadas: incluye las funciones que aparecen en la vista.
Influencia sobre funcionalidad: el proceso que cubre la vista puede influir directa o indirectamente en otro proceso.

De los datos expresados anteriormente se obtiene la Tabla 5. En la misma se ve que tres últimos elementos forman parte de una vista determinada, ya que cada uno de ellos se especifica según una vista determinada.

El valor influencia puede ser cuantificado si se desea, suele aplicarse en caso de que el usuario exprese un especial interés sobre esta función en particular o si la misma es significamente más importante que las demás en el sistema.

Nombre de la Vista	Actores Involucrados	Funcionalidad Asociada	Influencias
Vista1	Actor1	Funcion1	Inicia
	Actor2	Funcion2	
		Funcion3	

Tabla 5. Elementos de un diagrama de vistas.

JUSTIFICACIÓN:
El enfoque por vistos facilita el entendimiento del sistema y permite abstraer los actores y funcionalidades presentan en ciertas partes del framework.

3.1.2. Elaboración de Casos de Uso

El diagrama de casos de uso involucra los actores del sistema en conjunto con un flujo de actividades que se realiza para lograr un proceso determinado.

JUSTIFICACIÓN:
Los diagramas de caso de uso son esquemas que abstraen el flujo, rutinas y actores que intervienen en el proceso, de tal forma que es más sencillo entender el funcionamiento del mismo.

3.1.3. Contabilización de identificadores determinantes de asuntos

Similar en la idea a la metodología SLAI (Budwell and Mitropoulos 2008), pero con aplicaciones diferentes. La metodología SLAI busca especificar casos de uso transversales relacionando cada uno de ellos con sus influencias, obteniendo identificadores de frases, verbos y segmentando frases de ser necesario, replicando los diferentes casos de uso a través de los diagramas e identificando de esta manera los aspectos; MEDFOAR utiliza los identificadores con el propósito de contabilizar la presencia de funcionalidades a lo largo de todo el sistema.

Los identificadores (verbos y sustantivos significativos en el nombre de los casos de uso) se almacenan en un repositorio, de tal forma que sea posible determinar la frecuencia de cada uno de ellos dentro del procedimiento.

Además, se incorpora una influencia de los identificadores sobre la cardinalidad de las funcionalidades y vistas, de tal forma que sea posible asociar también cuantos cortes transversales serán posibles realizar a un asunto determinado.

Este procedimiento se realiza complementando la elaboración de los casos de uso. Los ID (identificadores determinantes) serán obtenido de la frecuencia según su utilización en estos diagramas.

JUSTIFICACIÓN:

Debido a que los diagramas de casos de uso encapsulan el procedimiento con un nombre apropiado al mismo, y en caso de presentar en diferentes etapas del proceso, mantiene su correspondencia entre los esquemas, contabilizar la frecuencia de los ID permitirá conocer que tan frecuente la funcionalidad se ejecuta en el framework.

El proceso de Contabilización de los ID, es un prerrequisito en esta metodología para la obtención de los aspectos candidatos.

3.1.4. Identificación de Módulos Multifuncionales

Mediante las necesidades y los requerimientos del software, se determinan los módulos que serán multifuncionales, es decir, que se utilizan en varios procedimientos.

Para esta labor se utilizan los diagramas de casos de uso, los escenarios y vistas. Cuando un modulo presente en un caso de uso utilice una funcionalidad que ya se encuentre abstraída en otro y las mismas sean diferentes, se establece esta función como multifuncional. Además, se aprecia de la misma manera en los diagramas de vistas.

RESPUESTA A:

Este proceso de la metodología, es una respuesta a la dificultad que existe en el tratamiento de módulos multifuncionales al momento de abstraer a un aspecto en particular. En esta etapa se identifican los módulos multifuncionales y bajo que segmento de abstracción de encuentran. (Un paso que atenta resolver el conflicto multifuncional)

A partir de la segunda iteración del ciclo, se realizan las siguientes fases:

3.1.5. Actualización de Casos de Uso

Consiste en adaptar los casos de uso de la primera etapa incorporando el resultado final de las otras etapas del proceso en su primera fase. Este procedimiento es necesario debido a la segmentación de aspectos en sub-aspectos mediante el enfoque de especialización.

3.1.6. Contabilización de identificadores determinantes de asuntos

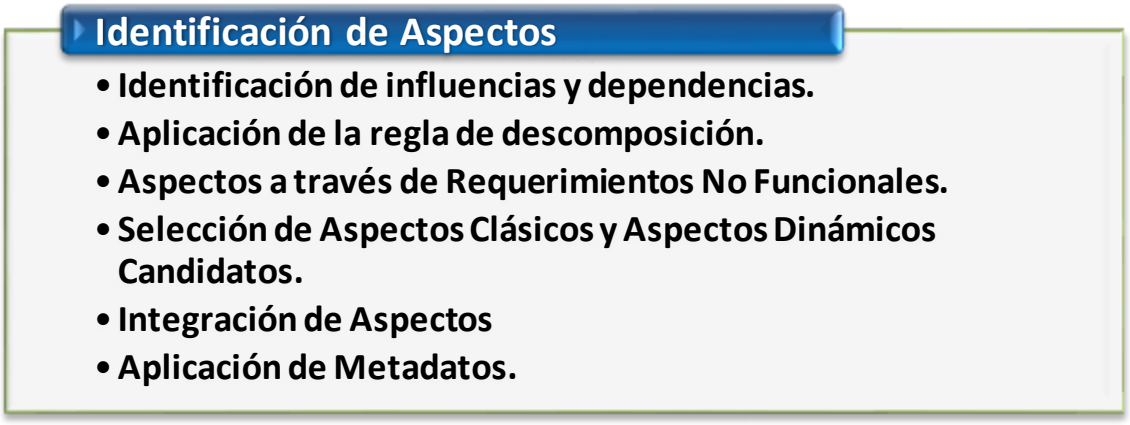
Como fase de repetición de ciclo, en esta fase de planea contabilizar los nuevos Identificadores léxicos obtenidos de los nuevos casos de uso y eliminar los que fueron borrados por redundancia, este procedimiento es más corto que el anterior debido a que los posibles cambios que se pueden realizar en las etapas del ciclo anterior son mínimos.

JUSTIFICACIÓN:

La repetición de estas dos etapas se justifica en la segmentación, especificación y el control de redundancia de los aspectos que se presenta en la etapa 3 de la metodología. Debido a esta división los aspectos candidatos pueden llegar a ser redundante con otros, por lo que es necesario validar los nuevos aspectos en estas fases.

RESULTADOS DE ESTA FASE:

- Diagramas de Casos de Uso.
- Diagramas de Vistas.
- Escenarios y Actores (opcional).
- Repositorio de Identificadores Léxicos.
- Módulos con Tendencia Multifuncional.



3.2. Identificación de Aspectos:

En esta etapa se utilizan los elementos obtenidos en la etapa anterior aplicándoles algunas reglas para obtener los aspectos candidatos.

Como prerrequisito a esta fase está una tarea aislada de la metodología, pero que se debe realizar en todo procedimiento de análisis, que es la identificación de métodos (procesos), clases, objetos, componentes y otros elementos de la POO.

3.2.1. Identificación de Influencias y Dependencias

Para el proceso de identificación de influencias y dependencias de funcionalidades se utiliza el modelado de casos de uso debido a que en estos

diagramas se expresan las funciones que deben ejecutarse antes y después, además de las exigencias y reglas de ciertos procesos.

En los diagramas de casos de uso se pueden observar relaciones como la Tabla 6 y la manera en que esta metodología las interpreta.

Enlace	Induce
A → B	Influencia básica de A sobre B
A extend B	Probable Influencia de A sobre B
A include B	Forzada Influencia de B sobre A
A,B specialice C	A y B se le aplica lo mismo que a C
Features	Secuencia de actividades
Objectives	Funcionalidades

Tabla 6. Elementos abstraídos de los casos de uso.

JUSTIFICACIÓN:

Obteniendo ventajas de los diagramas de casos de uso, se induce las influencias, dependencias y cortes transversales de algunas funcionalidades, de tal forma que se tiene una vista analítica de las interrelaciones de los diferentes módulos del software

.

3.2.2. Aplicación de la Regla de Descomposición:

Esta fase se basa en AORE, diferenciándose en que AORE trata los elementos de las relaciones como asuntos transversales y mira la influencia de los mismos, identificando el tipo de aspecto, MEDFOAR aplica la regla para influencias y dependencias, con la finalidad de identificar la presencia de un aspecto y la importancia de su existencia en esa parte del proyecto, sin importarle el tipo o la funcionalidad del mismo

La Regla de descomposición es un procedimiento que identifica posibles aspectos candidatos mediante un sistema de influencias y dependencias de funcionalidades.

Mediante los pasos anteriores, según las dependencias e influencias se aplica la regla de descomposición cuyo objetivo es determinar cuáles son los aspectos candidatos.

El proceso de descomposición cuenta con las siguientes fases:

Identificación de la Matriz de Asuntos:

Se coloca tanto como dominio x y codominio f(x) las posibles funcionalidades o asuntos que pueden representar un aspecto candidato.

Verificar la multiplicidad.

Se verifica f(x) para el cada elemento x colocándole la cardinalidad por cada influencia sobre f(x), en otras palabras, la cardinalidad de una funcionalidad f(x) está dada por cuantas funciones x influyen sobre este.

Determinación de Aspecto Candidato

Cuando el asunto aparece de un número de n veces (cardinalidad n), donde n sea un valor significativo frente a la cardinalidad de otros f(x), se establece este elemento como aspecto candidato. Esto aplica tanto a influencias como dependencias.

Además, de la combinación entre los aspectos candidatos obtenidos a través de contabilización de los asuntos determinantes y de la matriz de influencias y dependencias. La metodología propone una clausula:

Clausula Incluye: las relaciones por <include> en los diagramas de casos de uso, que no tienen dependencias y cuya influencia es sólo una funcionalidad, también serán interceptados por un aspecto candidato.

Nota: Es importante destacar el concepto de aspectos en esta etapa. Recordemos que los aspectos no representan entidades como las clases, sino que hacen referencia a “algo” sobre las clases y objetos, por ende, los aspectos saldrán de los

cortes transversales aplicados sobre estos. En otras palabras, el aspecto es una característica de la funcionalidad transversal que corta a clases y objetos, no siendo así, la misma funcionalidad, este concepto será aplicado posteriormente. Las funcionalidades que componen las relaciones son acciones y objetivos como tal, y no representan asuntos como en AORE.

JUSTIFICACIÓN:

Como una de las técnicas que utiliza la metodología para la determinación de los aspectos candidatos que tendrá el framework, la misma es auto sostenible.

3.2.3. Aspectos a través de Requerimientos No Funcionales

Mediante un listado de los más comunes requerimientos no funcionales, se determina los más aplicables al sistema dependiendo a su adaptabilidad con el mismo.

Este procedimiento se complementa con la identificación de RNF (Requerimientos no funcionales) obtenidos a partir de licitaciones con los usuarios del sistema.

La metodología ofrece el siguiente listado de posibles funcionalidades que representan, generalmente, un RNF:

- Disponibilidad de Servicios.
- Nivel de Seguridad.
- Rendimiento del Sistema.
- Tiempo de Servicio.
- Confiabilidad de Procesos.
- Confiabilidad.
- Capacidad de Usuarios Múltiples
- Casos Legales
- Adaptabilidad a la Red.

Es importante resaltar que las reglas de negocio deben formar parte de los requerimientos a tomar en consideración en la verificación si merecen considerarse aspectos candidatos.

Los aspectos candidatos propuestos en esta etapa, representan cortes transversales y no son obtenidos de funcionalidades macro, es decir, no serán analizados como los otros ya que su presencia dentro del sistema es global y afectan a ciertas clases en particular.

JUSTIFICACIÓN: los requerimientos no funcionales deben ser tomados como elementos de análisis dentro del proyecto, debido a que su presencia cambia el enfoque y el direccionamiento del sistema.

3.2.4. Selección de Aspectos Candidatos

Los aspectos candidatos serán obtenidos por la unión de los siguientes conjuntos. Un conjunto conformado por los elementos con mayor frecuencia obtenidos en el proceso de selección de Identificadores Determinantes. El otro conjunto está formado por los obtenidos mediante la regla de descomposición.

Los elementos obtenidos de la regla de descomposición tendrán sus elementos ubicados dentro del sistema, para los aspectos identificados, sólo a través de identificadores léxicos puede que sus elementos no sean definidos completamente.

JUSTIFICACIÓN:

Mediante la unión de las dos técnicas utilizadas para la extracción de aspectos candidatos se obtienen los aspectos candidatos.

3.2.5. Selección de Aspectos Clásicos y Aspectos Dinámicos

Los aspectos candidatos están seleccionados por la Unión de las funciones transversales establecidas.

Estos dos aspectos tienen diferencias en sus características, aunque mantienen su núcleo, los aspectos dinámicos definidos como “dynamic aspect” y los

“static aspect” brindan al framework características de modificación en tiempo de ejecución, como solidez en su estructura, respectivamente.

Posteriormente, se presentan unas medidas para determinar cuando un aspecto puede considerarse dinámico y cuando no, de tal forma que no involucre la estabilidad del sistema o el performance del mismo:

- Si el aspecto candidato tiene influencias de pequeña ramificación.

Al determinar las funcionalidades dentro de los diagramas de caso de uso es posible obtener aspectos que tengan varias influencias, si se desea que una de estas actividades pueda manipularse como dinámica entonces se considera el aspecto como candidato a ser dinámico.

- Si el aspecto candidato tiene alguna influencia de larga ramificación

Si las ramificaciones son largas, es necesario evaluar el hecho de que al cortar alguna funcionalidad en tiempo de ejecución se suprima a la vez alguna funcionalidad importante o si se complica el diseño debido a la magnitud de las ramificaciones, de ser este el caso no debería considerarse el dinamismo. Aún así, esta decisión queda bajo análisis de impacto y según el diseño.

3.2.6. Integración de Aspectos Candidatos

En esta sección, se unifican los aspectos candidatos dependiendo de su agrupación dentro del grupo de determinadores léxicos. Aspectos identificados que se encuentran bajo el mismo grupo de identificadores se unen, formando un nuevo aspecto que intercepta la unión de todas las clases y funciones que intercepta los anteriores.

3.2.7. Aplicación de Metadatos

Se establecen identificadores semánticos a los aspectos, clases y componentes del Framework a través de una capa de metadatos.

La Tabla 7 muestra la información que almacenará un metadato, la sección de funciones hace referencia a las identificadas en procedimientos anteriores.

ID	Tipo	Funciones Relacionadas	Relevancia
Nombre	Aspectos Componentes Clases	Función A Función B	1 2… 7

Tabla 7. Elementos abstraídos en los metadatos.

El valor de Relevancia es obtenido según las funcionalidades asociadas.

$$\text{Reelevancia} = \frac{\text{Parcial Dependencias} + \text{Parcial Influencias}}{\text{Total Dependencias} + \text{Total Influencias}} \text{ x } 7$$

EN RESPUESTA A: La deficiencia de conocer el objetivo de cada elemento del framework, después de varias tareas de mantenimiento y modificación constante.

Como resultado de esta fase se obtiene:

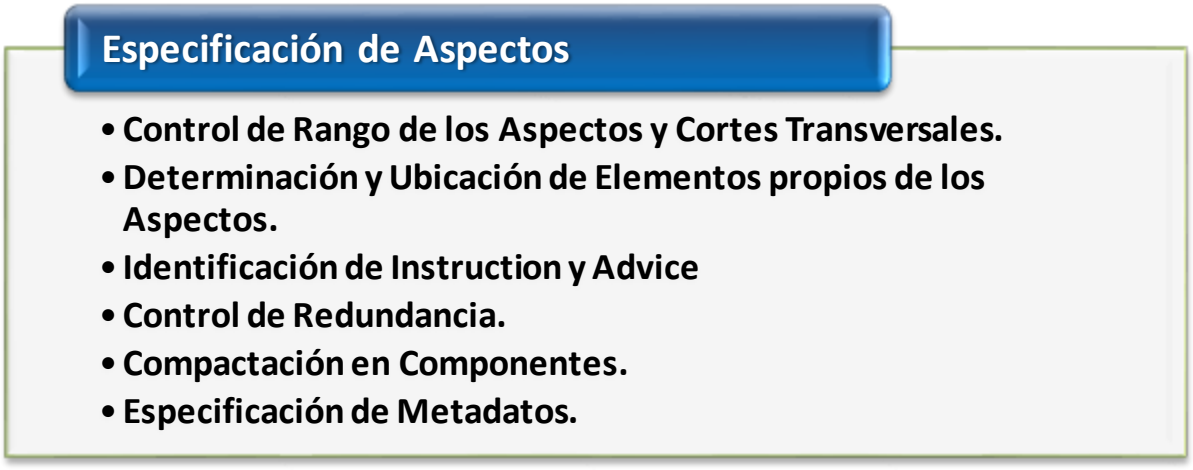
- Matriz de Descomposición
- Aspectos Candidatos
- Aspectos Dinámicos Candidatos
- Metadatos

3.3. Especificación de Aspectos Candidatos.

En esta etapa se pretende analizar los componentes de los aspectos, especificándolos y ubicándolos dentro del esquema de análisis, además de controlar la redundancia.

JUSTIFICACIÓN:

Debido a que los aspectos contienen sub elementos, cada uno de ellos debe ser especificado para la mejor comprensión del framework y de los aspectos que contienen.



3.3.1. Control de Rango de los Aspectos y Cortes Transversales

A cada aspecto candidato se le determina, su rango. Se especifica que aspectos dependen de este, y cuales influyen sobre él.

A diferencia del procedimiento de la regla de descomposición, se especifica cuando se ejecutan y si presentan condiciones para ejecución. Se determina los puntos de acceso según las funciones transversales y se determina las acciones que se ejecutan antes y después. Este punto hace referencia a los puntos de corte y los advice.

La Tabla 8 presenta los elementos que se especifican en esta etapa.

ID	Corta a	Condiciones	Depende de	Corte
..	..	..	..	..

Tabla 8. Control de Rango y Corte Relacional

La sección de corte hace referencia al asunto transversal de la funcionalidad. Por ejemplo: sistema de autenticación, mensaje a clientes, entre otros.

JUSTIFICACIÓN:

Dentro del proceso de análisis es importante determinar que características y funciones del framework dependen de otras, en mayor proporción si se trata de los aspectos dinámicos, debido a que la sostenibilidad del sistema está sujeta a ellos.

EN RESPUESTA A:

Combatiendo la dificultad que existen al momento de establecer asuntos dentro de un corte transversal determinando, después de todo el proceso de análisis relativo al aspecto de manera global cada uno de ellos se asocia a una funcionalidad macro.

3.3.2. Determinación y Ubicación de los Elementos Propios de los Aspectos

En esta etapa se identifican y analizan los puntos de corte (CP), puntos de enlace (JP), advices (AP). Los instructions (IS) no serán evaluados con reglas en ninguna etapa, debido a las operaciones son no predictivas dentro de las demás funciones, pero se puede hacer referencia a clases u objetos.

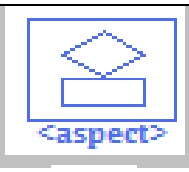
Autor	Elemento	Representación
Zakaria, et al 2003	Aspecto	
A. Sanchez 2003	JoinPoint	
N/A	CutPoint	
Basado en Steinmacher 2003	Advice	
Aldawud, et al 2003	Cross-Cutting	

Tabla 9. Representación de los elementos de los aspectos

El análisis de cada elemento se realiza a través de las siguientes reglas:

- Crosscutting (Cortes Transversales):

Se visualizan en la relación directa entre los aspectos y los casos de uso que interceptan, es decir, dependiendo de la funcionalidad del caso de uso se conceptualizan dentro de una función transversal, en otras palabras, representa la

funcionalidad transversal en cada uno de los casos de uso al elemento que la genera.

- Puntos de Corte (CP):

Los puntos de corte son elementos que permite el acceso del aspecto en una etapa determinada de código. Como afectar directamente las clases, objetos y métodos, su ubicación puede presentarse dentro de las llamadas o ejecución de métodos, a constructores, al inicializar objetos, al asignar un atributo y en muchas otras circunstancias.

La metodología establece como puntos de ubicación los casos de uso que encapsulan cada uno de estos elementos, haciendo referencia al nombre de los objetos involucrados. Su representación en el diagrama se coloca con la simbología correspondiente (Ver Tabla 9), su presencia es asumida con la presencia del aspecto.

- Puntos de Enlace (JP):

Se asocian con un aspecto determinado, contiene los métodos que forman parte de la funcionalidad transversal. Los puntos de enlace son el tejido de los CP que hace referencia a todos los CP bajo el mismo corte transversal, por ende su ubicación se asocia de manera relativa al aspecto en cualquier parte del mismo.

En esta metodología, los JP serán ubicados dentro del diagrama de Casos de Uso, solamente si, se requería un acceso global a todos los cortes transversales.

- Advice (AP):

Deben ubicarse en el caso de uso al que el aspecto interviene, los advice serán colocados de forma intermediaria entre los predecesores del caso de uso y el mismo para los eventos before(), posterior al caso de uso para los eventos after() en caso de tener y around().

Para su colocación dentro del diagrama, se expresaría con la simbología de la Tabla 9 para los casos de antes, durante o después.

La Tabla 9, muestra la representación que proporciona la metodología (tomada de trabajos anteriores) para los elementos de los aspectos, no obstante, la herramienta que se presenta en este trabajo, utiliza un sistema de diagramado sencillo, debido a que se enfoca a operaciones que realiza la metodología.

La Imagen 6, muestra la ubicación que llevan los elementos dentro del diagrama de casos de uso. En el caso de estudio de la metodología se presenta un diagrama de casos de uso completo donde se encuentran los elementos especificados para cada aspecto candidato (Ver Imagen 26 en la página 118).

Con la metodología se reúnen todos los Advice que se ejecuten antes en el mismo bloque, al igual que los que se ejecuten en medio de la intervención y al final. El proceso de detallado de los AP será a partir de las dependencias e influencias, pero de los IS será determinar elementos que puedan ser modificados en tiempo de ejecución.

JUSTIFICACIÓN:

En este proceso solo se determina y se expresa la presencia de esos elementos y no así su contenido debido al carácter de los mismos.

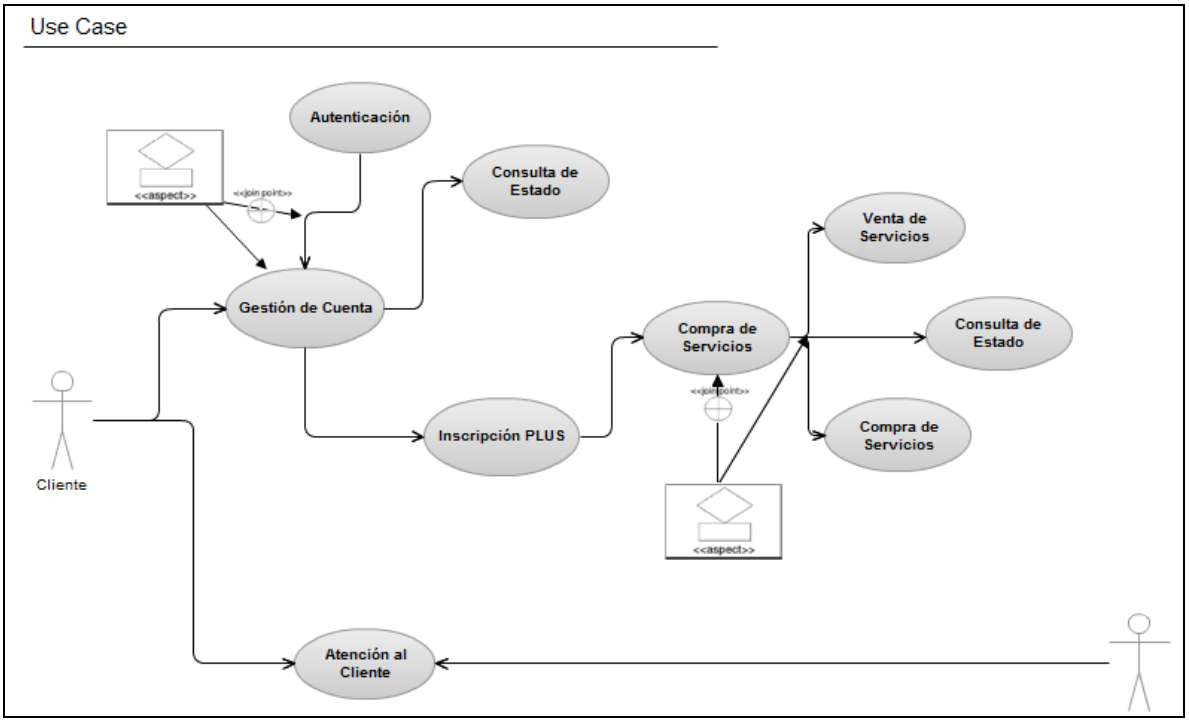


Imagen 6. Ubicación de los aspectos y sus elementos.

3.3.3. Control de Redundancia

Esta sección estaba basada en AOCRE, las metodología comparten el principio de especialización (segmentar un elemento en elementos más pequeños) de aspectos; se diferencian en que AOCRE realiza el proceso enfocado al refinado de requerimientos, mientras que esta metodología aplica el proceso para la segmentación de aspectos, en análisis de estos sub elementos y la reducción de duplicidad. Se asemejan en que los aspectos y clases asociadas se integran en componentes. Para esta tarea se debe definir si se encuentran asociados bajo el mismo corte transversal y si los elementos presentan una fuerte relación al momento de ejecutarse las rutinas del framework.

Debido al proceso de especificación de aspectos, se pueden encontrar aspectos con funciones y elementos muy similares, lo que ocasiona la redundancia de las definiciones, lo cual debe ser analizado. Después, de este proceso se puede decidir su unificar los aspectos candidatos como uno sólo o simplemente mantenerlos de forma separada. Para esto se puede obtener ayuda de los análisis de los puntos 2.3 y 2.2 y 3.1.

El proceso de Verificación de redundancia es el siguiente:

Un aspecto se le considera redundante si presenta un alto grado de similitud con otro aspecto (en la herramienta se designó un 85%, estando sujeto a consideraciones del analista), estas similitudes están asociadas tanto a las características del aspecto, de sus elementos y los cortes de los mismos.

Las características que se evalúan son:

- Propiedades del aspecto,
- Cortes a las clases
- Puntos de corte asociado a métodos
- Advice que se ejecutan en cada punto de corte
- Instruction existentes
- Puntos de Unión del Aspecto.

Nótese, los aspectos con la misma funcionalidad se agruparon en etapas anteriores. Esta etapa busca reducir redundancia es sistemas muy complejos donde generalmente se presentan.

JUSTIFICACIÓN:

Dentro del análisis del framework no se deben obtener declaraciones dobles, ni se funcionalidades, clases, rutinas, al igual que de aspectos, pues puede causar difusión en el entendimiento del software.

3.3.4. Compactación de Componentes

Como se aplica en AOCRE y en procesos de análisis de Ingeniería de Software. Se compactan los elementos en componentes, a diferencia que se toman los aspectos en consideración.

Los aspectos y clases asociadas se integran en componentes. Para esta tarea se debe definir si se encuentran asociados bajo el mismo corte transversal y si los elementos presentan una fuerte relación al momento de ejecutarse las rutinas del framework.

JUSTIFICACIÓN:

Debido a que los aspectos no forman parte de un bloque aislado del sistema, su compactación junto con las clases y rutinas mejoraría la comprensión global del framework.

3.3.5. Especificación de Metadatos

Se aplica el proceso de metadatos para integrar los elementos de los aspectos: puntos de corte, puntos de unión, advice, instruction y cross-cutting.

JUSTIFICACIÓN:

Los elementos de los aspectos también deben ser expresados como metadatos para salvar el valor semántico de los mismos.

EN RESPUESTA A: La carencia de conocimiento semántico de cada uno de los elementos de los aspectos cuando el sistema tiende a ser muy grande o después de muchos procesos de modificación.

Como resultado de esta etapa se tiene:

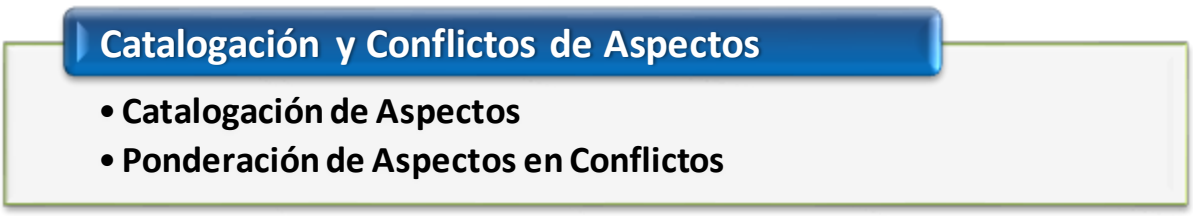
- Especificación de los Aspectos
- Metadatos de los elementos de los aspectos
- Ubicación de los elementos de los aspectos
- Especialización de aspectos (reducción de duplicidad).
- Detallado de aspectos

3.4. Catalogación y Conflictos de Aspectos

El propósito de esta etapa consiste en valorizar y cuantificar el impacto de los aspectos y demás elementos frente a un conflicto de importancia.

En el diseño del framework, ciertos módulos llegaran a tener conflictos referentes a la prioridad en tiempo de ejecución, uso de recursos, entre otros. Es importante cuantificar la relevancia de cada uno de ellos frente al sistema y darle privilegios a los más imprescindibles.

Justificación: los conflictos entre los elementos del sistema están frecuentes inclusive sin la incorporación de los aspectos, ahora en esta metodología, se enfatiza en el tratamiento de los aspectos, por ende, en la resolución de conflictos de las funcionalidades transversales.



3.4.1. Catalogación de Aspectos

Esta etapa es similar a AORE en ejecución, pero difieren en el objetivo; AORE clasifica los asuntos en aspectos, decisiones y funciones, pues su elemento de análisis son los cortes transversales; esta metodología trata de catalogar los aspectos candidatos separándolos de las funciones o decisiones, este proceso sólo aplica a cierto grupo de aspectos.

La catalogación de aspectos es un proceso que clasifica los aspectos candidatos en aspectos reales, funciones o decisiones. Los aspectos son funciones transversales que no se pueden encapsular en funciones o clases normales, los catalogados como funciones son aquellos que en el análisis son aplicados a elementos simples como clases, métodos y de fácil encapsulación, los determinados decisiones son aquellos que en ejecución solo desempeñan el papel de tomar decisiones en etapas determinadas.

Esta etapa es sólo aplicada cuando los aspectos identificados tienen débil persistencia, es decir, puede que no sean realmente aspectos; se aplica para el siguiente caso:

Los aspectos identificados carecen de puntos de cortes, puntos de unión o / y advice. Aplicables para los reconocidos solamente por identificadores léxicos, debido a que los mismos pueden no tener elementos reconocibles.

JUSTIFICACIÓN:

Se puede decir que los aspectos son elegidos a través de los cortes transversales debido a su característica multiposicional en el framework, pero a pesar de ellos, pueden desempeñar funciones diferentes a mandatos, rutinas e instrucciones como el de tomar decisiones o actuar simplemente como clases o rutinas normales. Por esta causa la metodología utiliza tanto Identificadores Léxicos como Diagrama de casos de uso para determinar los aspectos.

Ponderación de Aspectos en Conflictos

En esta etapa se ponderan los aspectos dependiendo de varios factores. Este proceso se realiza 1 a 1 entre los cortes con conflictos. La selección de los actores

de los conflictos se visualiza según el esquema del sistema y el punto de vista de los stakeholders.

La clasificación es la siguiente:

- Basados en la cantidad de influencias y dependencias obtenidas en etapas anteriores.

La priorización se basa en la cantidad de funciones dependientes. Un porcentaje del máximo de influencias actuales se aplicará a cada funcionalidad en conflicto de tal forma que quien tenga más funciones dependientes merecerá más prioridad; en este punto es importante resaltar que la magnitud de significancia es restada si una función en conflicto es predecesora de la misma, debido a que la otra función se ejecutará primero.

Para este proceso se utiliza la Tabla 10, que a su vez integra el otro método, pero hace referencia a la cantidad de dependencias e influencias, posibilidad de ejecución (decisiones) y anteposición.

- Basado en el criterio de los stakehoulders y usuarios del framework.

La valoración proporcionada por estos entes es importante tomarla en consideración, y tiene un impacto sobre la resolución anterior, pero no decide estrictamente en cual aspecto es necesario priorizar.

$$\text{ValorAspecto} = \sum_{\text{Función Asociada}}^{\text{Todas}} (\text{Valor} + \text{n\# Dependencias} + \text{ClaseAsociada})$$

La valoración en la Tabla 10, otorga el rango de 0 a 7 para los Stakeholders y Usuarios (de ser aplicable) donde 7 es el más alto posible para cada elemento interceptado que está valorado. Por cada influencia que tiene la función interceptada obtiene 1 de importancia y 3 por cada clase interceptada. Esta valoración está dada por la relevancia de cada elemento dentro del diseño.

El aspecto con más valor, será denominado el de más impacto entre los dos seleccionados.

Elemento a Valorar	Valor	Justificación
Usuario / Stakeholders	0 – 7	Influencia del usuario y de los stakeholders sobre el aspecto. Este valor aplica para cada elemento al que intercepta. Aspecto en sí, Clases y Funciones
Funcionalidad Asociada a las que intercepta	1x	Funciones sobre las que influye las funciones que corta el aspecto
Clase Asociada a las que intercepta	3x 1x	Clase a las que intercepta Elemento del aspecto que intercepta la clase

Tabla 10. Resolución de Conflictos.

Finalmente, la metodología sugiere un mecanismo para resolver la resolución de conflictos, siendo la solución final en manos de los analistas del sistema.

Debido al grado de redundancia de aspectos se puede elegir repetir el procedimiento desde la etapa 1 y aplicar los cambios sólo donde sea necesario. En esta etapa se obtienen:

- Aspectos Cuantificados por importancia.

3.5. MEDFOAR y los Inconvenientes de la POA

La metodología propuesta reacciona frente a los inconvenientes de la POA mediante las reglas en las diferentes etapas, minimizando el impacto de los mismos dentro del Framework. Estas reglas atacan de forma personalizada cada uno de ellos, de la siguiente manera:

3.5.1. Conflicto Estructural:

Como solución propuesta para este conflicto, se tienen los pasos de

(Etapa: 0) y Especificación de Metadatos (Etapa: 3.3.5).

La herramienta proporciona un sistema de almacenaje de la información y características de cada elemento en estructura XML.

Estos pasos de la metodología proveen un mecanismo de almacenamiento semántico de los aspectos, funciones, clases, influencias, dependencias y otros elementos de tal forma que el framework genere información sobre este, reconociendo de manera rápida los objetivos de cada elemento.

Por ende, cada elemento tiene una descripción de su funcionalidad en conjunto con todos los elementos pertenecientes y relacionados al mismo, de su influencia en las funcionalidades del sistema y las clases asociadas al mismo. Además, contiene la relevancia según el usuario de las funcionalidades, clases y aspectos.

El tratamiento semántico de los datos guardados en operaciones de búsqueda y filtrado en la herramienta forma parte de los trabajos futuros.

3.5.2. Conflicto de Comportamiento:

Para el tratamiento del conflicto de comportamiento se tienen reglas de ubicación tanto para los aspectos, como para sus elementos. Los pasos de la metodología que tratan este punto son: Identificación de Influencias y Dependencias (Etapa: 3.2.1), Determinación y Ubicación de los Elementos Propios de los Aspectos (Etapa: 3.3.2),

Compactación de Componentes (Etapa: 0)

Para la herramienta de apoyo, los elementos de los aspectos son automáticamente vinculados a sus padres y a los elementos (clases o funciones) a los que intercepta o se relaciona.

Estos pasos realizan las siguientes operaciones para minimizar el conflicto:

Mantienen la correlación de influencias y dependencias de las funciones a las que un aspecto intercepta, permitiendo asociar el rango en donde debe presentarse para que el flujo del sistema presente en el esquema sea el deseado. Esto es complementado con las demás etapas.

Ubican dentro de los diagramas de casos de uso un aspecto candidato y sus componentes, de tal forma que sea posible su fácil visualización dentro del análisis del sistema. Además, permite a su vez determinar el tiempo de ejecución y los cortes que realiza el aspecto determinado.

La herramienta de apoyo incluye una sección de diagramado que muestra las relaciones existentes entre las funcionalidades del sistema, permitiendo la visualización del diseño lógico del mismo, los cortes que realiza un aspecto determinado.

Permite indicar la existencia de advice de manera temprana.
Mediante el proceso de compactación de clases, aspectos y métodos dentro de un paquete de componentes, se encapsula la funcionalidad como se desee ejecutar, evitando la ambigüedad.

Como conclusión, este conflicto es tratado mediante la determinación, ubicación, compactación de los aspectos y sus componentes, en conjunto con la correlación de las funcionalidades y clases que el aspecto intercepta.

3.5.3. Conflicto por Multifuncionalidad:

Para determinar bajo que corte transversal un módulo multifuncional debe asociarse, se aplican los siguientes pasos de la metodología: Identificación de Módulos Multifuncionales (Etapa: 3.1.4), Control de Rango de los Aspectos y Cortes Transversales (Etapa: 3.3.1).

Estas medidas minimizan el inconveniente del conflicto multifuncional debido a que primero identifica los aspectos que cumplen con el requisito de poder pertenecer a diferentes cortes transversales.

Posteriormente, mediante el control de rango de los aspectos, según las influencias y dependencias asociadas a los asuntos, por magnitud, es posible asociar esta funcionalidad a este asunto transversal catalogándolo dentro de un corte transversal. La metodología determina los aspectos para módulos multifuncionales asociando puntos de unión en cada funcionalidad macro del sistema que acceda a estos módulos, de tal manera que los aspectos encapsulan estas multifuncionalidades interceptándolas a través de los puntos de corte y gestionando el enlace a través de los puntos de unión. Estos aspectos serán determinados bajo el corte transversal global, de tal manera que se distingan entre los demás, por la propiedad multifuncional de sus elementos.

3.5.4. Facilidades Dinámicas:

Debido a que la metodología está orientada a la utilización de AspectJ, las características estructurales de un aspecto dinámico y estático las proporcionarán este framework, la metodología determina cuando un aspecto puede ser considerado como dinámico y cuando no, dependiendo a su presencia en el programa.

La etapa de la metodología que trata este conflicto es la Selección de Aspectos Clásicos y Aspectos Dinámicos (Etapa: 3.2.5). Como se propone en este paso, dependiendo del nivel de influencia y compenetración de las funcionalidades a las que intercepta el aspecto dentro del sistema, la metodología determina si un aspecto puede ser denominado dinámico sin comprometer la estabilidad.

La herramienta de apoyo, realiza el procedimiento de esta etapa y determina los aspectos dinámicos candidatos, sin embargo no aplica ningún cambio, pues le deja la decisión final al analista.

Capítulo 4

Herramienta de Soporte: HSTAR

La Herramienta que se presenta, es un plug-in para Eclipse Helios que utiliza librerías de lenguaje java para el tratamiento de aspectos conocidas como AspectJ. Esta herramienta fue nombrada como HSTAR (Herramienta de Soporte para el Tratamiento de Aspectos en Requerimientos). La Imagen 7 muestra los resultados de cada etapa frente a las aplicaciones de la herramienta de apoyo y la metodología.

El objetivo de esta herramienta es la de soportar la metodología en las etapas más significativas de la misma: permitiendo la visualización lógica del flujo de las funcionalidades, facilitando las relaciones, influencias y dependencias de los requerimientos, identificando automáticamente los aspectos candidatos, determinando los aspectos que pueden considerarse dinámicos, eliminando redundancias y solucionando conflictos entre aspectos; para finalmente generar el case del framework.

La herramienta trabaja con cuatro elementos principales (llamados posteriormente: elementos básicos y secundarios según jerarquías):

- **CORTE:** hace referencia a las macro funcionalidades, es decir a un grupo de actividades y procesos que permiten logran un resultado final cuantificable.
- **ASPECTO:** son los elementos que utiliza la Programación Orientada a Aspectos.
    - **CUT POINT:** puntos que interceptan rutinas de clases.

- **JOIN POINT:** puntos de unión de los puntos de corte. Su funcionalidad es lógica.
- **ADVICE:** son las operaciones que se ejecutan en los puntos de corte. Se especifican si su intervención es antes, durante o después de un punto de intervención.
- **INSTRUCTION:** son intercepciones a las clases y sus elementos.

- **CLASE:** son los elementos class utilizado en la Programación Orientada a Objetos.
- **FUNCION:** son las rutinas, procesos o funcionalidades. Estos métodos pueden ser libres o estar asociados a un aspecto o a una clase.

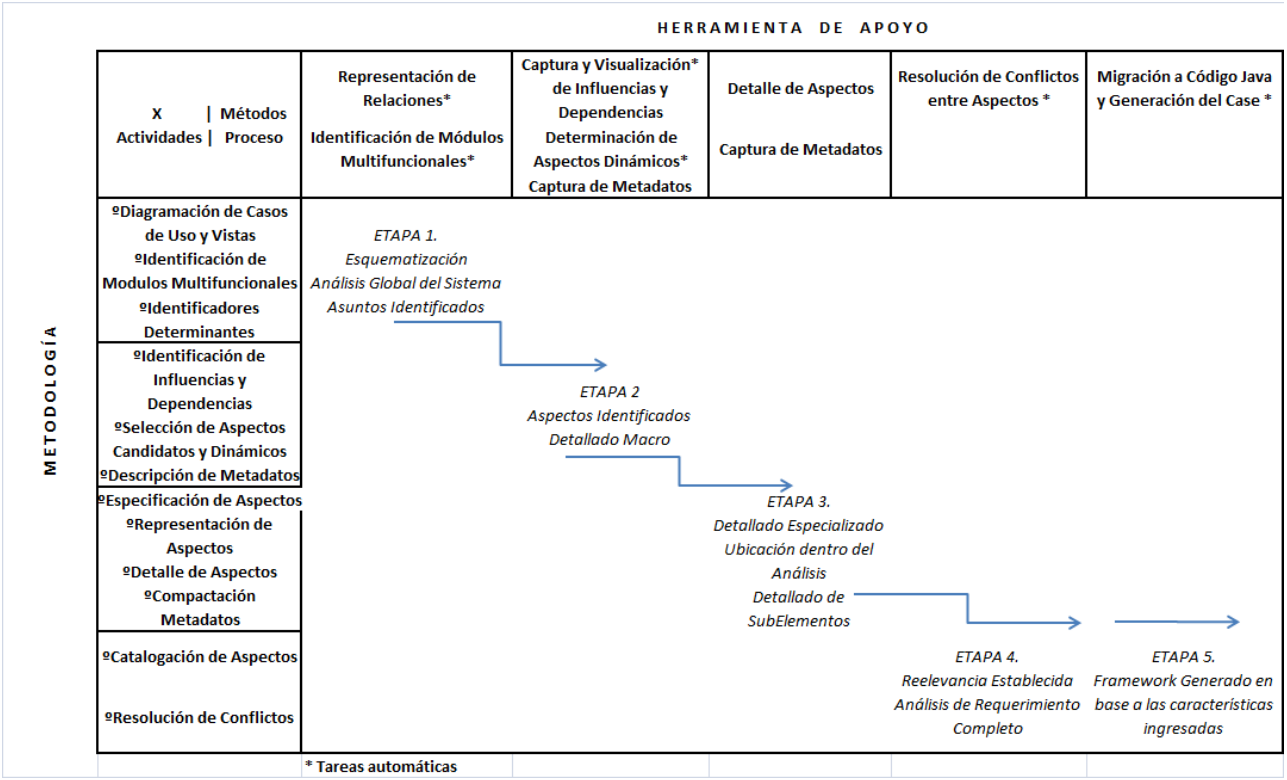


Imagen 7. Procesos de la Metodología y la Herramienta frente a logros parciales

Para la herramienta, los objetos FUNCION hacen referencia a la funcionalidad como requerimiento funcional de manera lógica, no identificando todos los pasos que se requieren para el logro de la misma. Por ejemplo: FUNCION RetirarDinero específica el procedimiento general de retiro de dinero durante la Ingeniería de Requerimientos, en etapas posteriores esta función tendrá sub procedimientos que

permitirán el control del hardware y las validaciones necesarias para el logro del objetivo. También, la herramienta trata los objetos CORTE como funcionalidades macro de importancia relevante para el proyecto, no debe confundirse con los cortes transversales en el código que son los aspectos.

El plug-in cuenta con una perspectiva y una vista: aquí se presentan todos los formularios separados en tabulaciones estilo carpetas que permiten realizar los objetivos e expuestos anteriormente.

A continuación se detallará la vista que contiene los formularios, debido a que en esta se realizan todos los procesos de soporte a la metodología; dentro de este análisis se presentan las porciones de código más importante en donde se aplica la metodología. Posteriormente, se explicará cómo se debe utilizar la herramienta de apoyo para realizar el análisis del sistema.

4.1. Vista de Formularios:

La vista de formularios consiste de cuatro pestañas que brindan todas las funcionalidades. Estas pestañas son las siguientes:

4.1.1. Pestaña: Principal <MAIN>

Este formulario tiene los objetivos de: permitir la inserción, modificación o eliminación de los elementos básicos, mostrar los hijos de estos elementos, cuantificar la importancia de los elementos, según el criterio de los usuarios y stakeholders y describir cada uno de los elementos.

La vista gráfica del formulario se tiene en la Imagen 8, se puede observar cuatro columnas o divisiones:

- **Componentes:** Permite la selección del elemento básico, colocarle un nombre único para cada uno de ellos, una descripción y el impacto otorgado por el stakeholders. Dependiendo del elemento básico, permite

rellenar otros elementos como se muestra en la Imagen 9, componentes, métodos, puntos de corte, puntos de unión, advice e instruction, cada uno de ellos único para el nombre elemento padre seleccionado.

- **Acciones:** Permite la búsqueda e inserción de los datos, cada botón agrega el elemento colocado a su izquierda, mientras que búsqueda permite conocer si el elemento ya existe o no.
- **Selección:** muestra todos los elementos básicos existentes del tipo seleccionado. Proporciona la facilidad de eliminar uno de ellos mediante una operación en el menú contextual. Esta operación, eliminará todas las relaciones de este elemento con los que se encuentre relacionados.
- **Instancias:** muestra todos los componentes del elemento seleccionado y permite eliminarlos mediante un menú contextual.

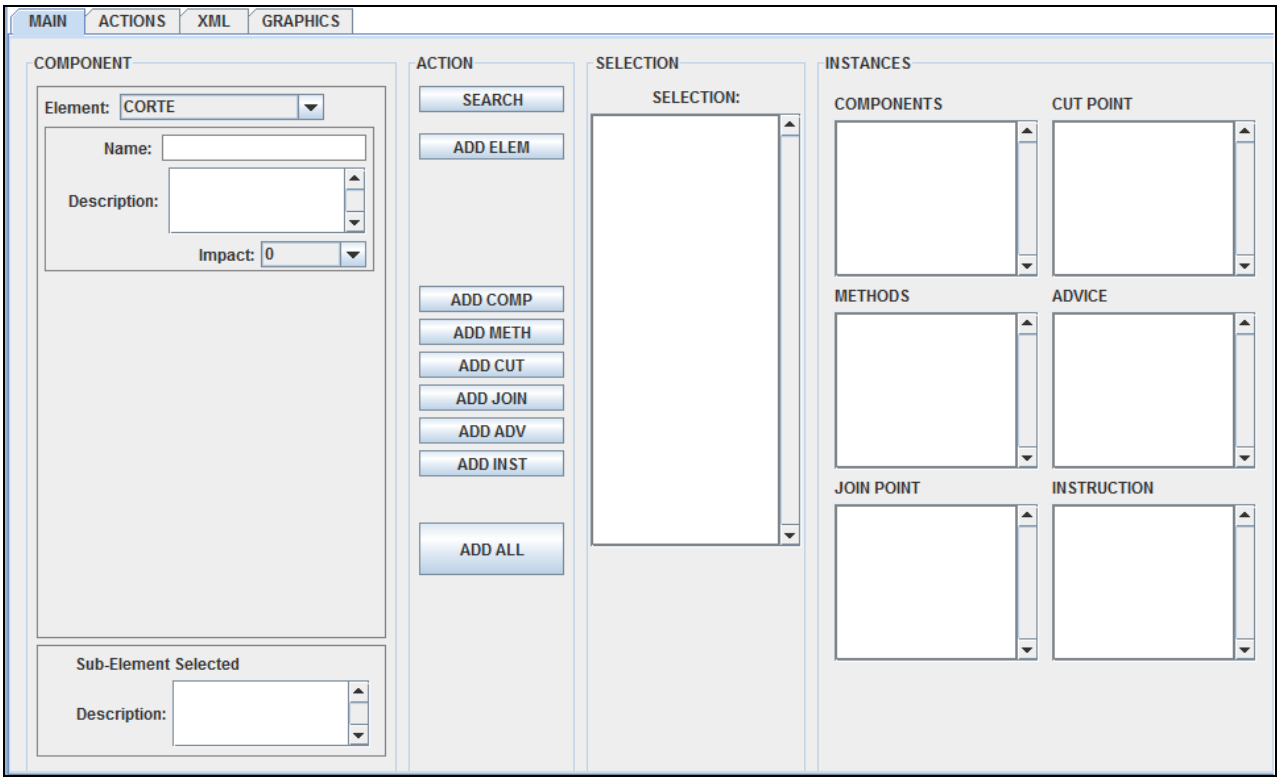


Imagen 8. Pantalla correspondiente a la pestaña <MAIN>

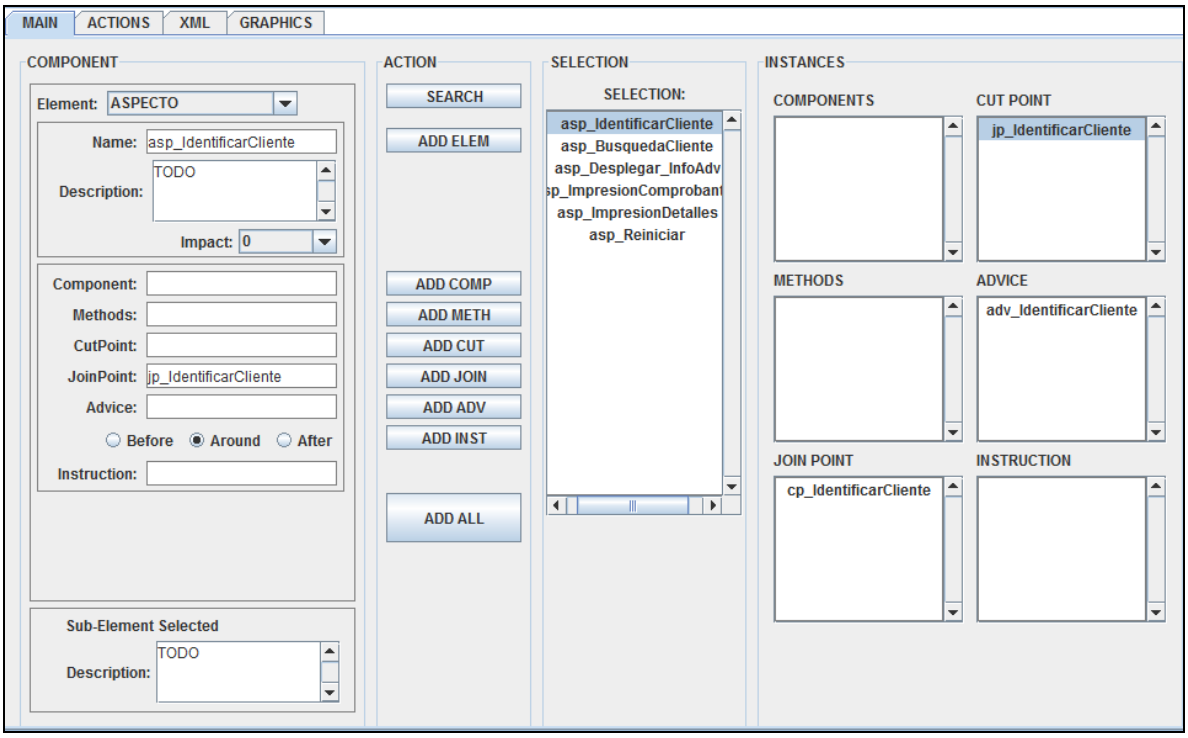


Imagen 9. Pantalla <MAIN> mostrando el proceso de inserción de aspectos.

La Herramienta maneja todos sus elementos básicos mediante una estructura de clases compleja. Esta relación se visualiza en la Imagen 10, donde cada bloque representa una clase, y en donde el signo de más (+) indica relaciones con otro elemento. Todos estos elementos pertenecen a una clase denominada principal.

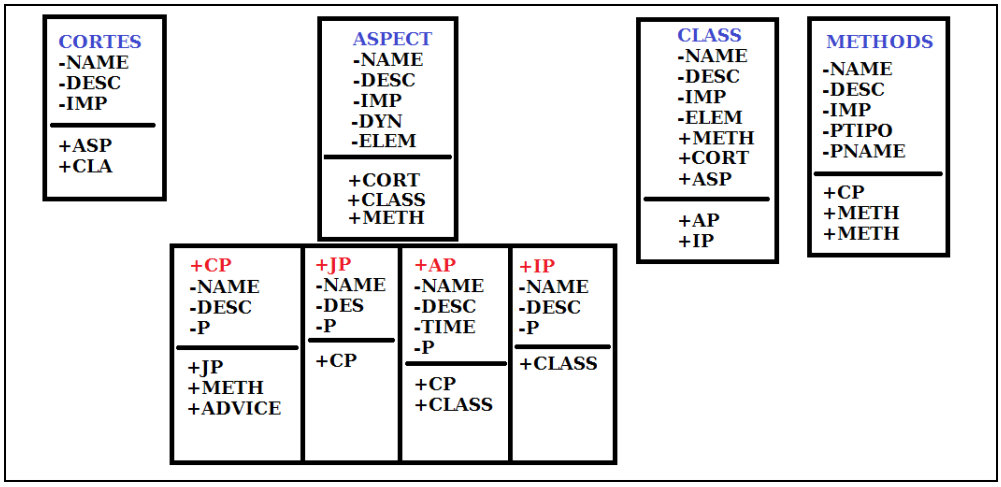


Imagen 10. Elementos de la Clase Principal y Relaciones dados por el signo más (+).

La clase Principal instancia las cuatro clases de elementos básicos:

```
public class Principal {
      private ArrayList<Cortes> Cor = new ArrayList<Cortes>();
      private ArrayList<Aspects> Asp = new ArrayList<Aspects>();
      private ArrayList<Clases> Cla = new ArrayList<Clases>();
      private ArrayList<Methods> Met = new ArrayList<Methods>(); }
```

La clase Cortes presenta:

```
public class Cortes {
      private String Nombre;
      private String Descripcion;
      private int Impacto;
      //relaciones
      private ArrayList<Aspects> Rel_Aspectos = new ArrayList<Aspects>();
      private ArrayList<Clases> Rel_Clases = new ArrayList<Clases>();
```

La clase Aspects que controla los aspectos se define a continuación:

```
public class Aspects {
      private String Nombre;
      private String Descripcion;
      private boolean dynamic;
      private int Impacto;
      private ArrayList<String> Componentes = new ArrayList<String>();
      private ArrayList<CutPoint> CP = new ArrayList<CutPoint>();
      private ArrayList<JoinPoint> JP = new ArrayList<JoinPoint>();
      private ArrayList<Advice> AP = new ArrayList<Advice>();
      private ArrayList<Instruction> IP = new ArrayList<Instruction>();
      //relaciones
      private ArrayList<Methods> Metodos = new ArrayList<Methods>();
      private ArrayList<Clases> Asoc_clase = new ArrayList<Clases>();
      private Cortes Asoc_corte; }
```

Para la clase Clases tenemos:

```
public class Clases {
      private String Nombre;
      private String Descripcion;
      private int Impacto;
      private ArrayList<Methods> Metodos = new ArrayList<Methods>();
      private ArrayList<String> Componentes = new ArrayList<String>();
      //relaciones
      private ArrayList<Cortes> Cortes = new ArrayList<Cortes>();
      private ArrayList<Aspects> Asp_Relation = new ArrayList<Aspects>();
      private ArrayList<Aspects.Advice> Advice_Relation;
      private ArrayList<Aspects.Instruction> Instruc_Relation; }
```

Finalmente, la clase Métodos:

```
public class Methods {
	private String Nombre;
	private String Descripcion;
	private String PadreTipo;
	private String PadreNombre;
	private int Impacto;
	//relaciones
	private ArrayList<Methods> Influye_Metodo;
	private ArrayList<Methods> Depende_Metodo;
	private ArrayList<Aspects.CutPoint> Relation_CutPoint; }
```

Todas las relaciones expresadas en la Imagen 10 se visualizan en cada componente bajo la marcación del comentario relaciones.

Esta es la primera parte de la herramienta a utilizar en la aplicación de la metodología, siendo los elementos más importantes para adicionar los de tipo FUNCION.

4.1.2. Pestaña: Acciones <ACTIONS>

Este formulario tiene los siguientes objetivos:

- Permitir establecer las relaciones entre los elementos básicos y secundarios.
- Visualizar todas las relaciones, influencias y dependencias.
- Permitir la eliminación de relaciones, influencias y dependencias.
- Determinar los aspectos candidatos de la estructura existente.
- Controlar y Eliminar aspectos redundantes. Esta tarea sólo aplica para Aspectos.
- Establecer y Determinar cuáles son los aspectos dinámicos.
- Priorizar un Aspecto frente a un conflicto.

La vista gráfica de esta sección se presenta en la Imagen 11, la cual está dividida en tres capas:

Capa1: Despliega los elementos, sus componentes, relaciones, influencias y dependencias.

Las influencias y dependencias sólo existen entre los métodos libres o métodos asociados a una clase, y representan el flujo del sistema.

Se le denomina relaciones a cualquier vínculo entre los diferentes elementos. La herramienta sólo permitirá la adición de relaciones posibles y reales, además de mostrar la descripción de cada elemento. Las relaciones posibles se ven en la Imagen 12.

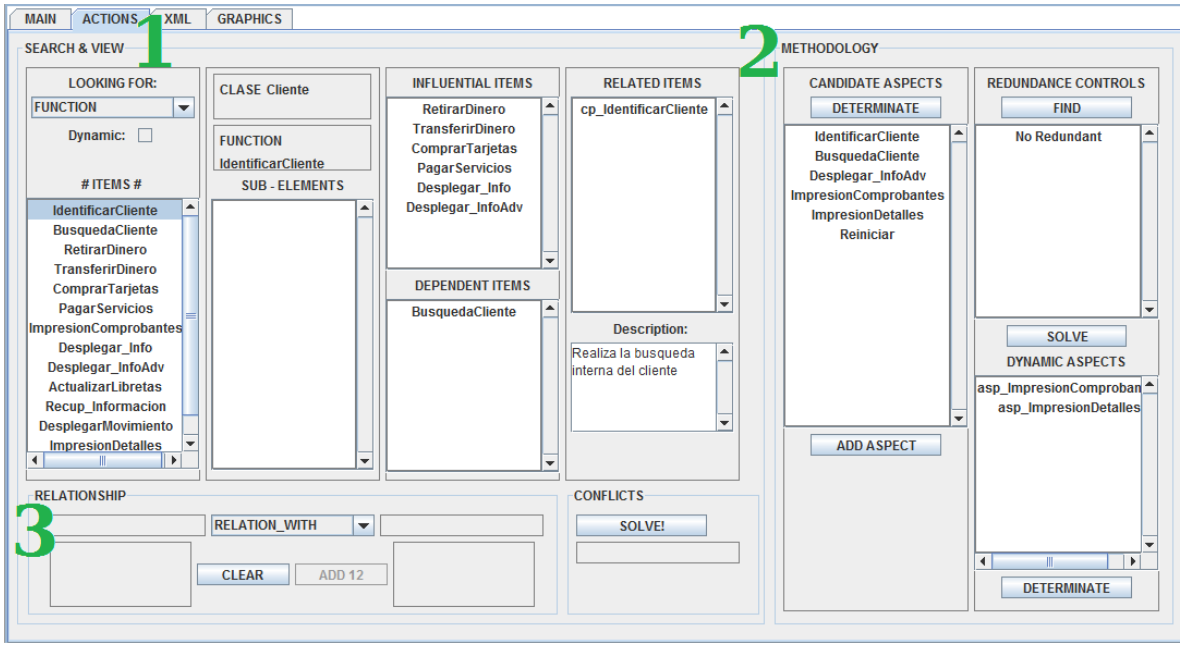


Imagen 11. Pantalla de la pestaña <ACTIONS>.

Capa2: La sección señalada como 2, contiene las reglas de la metodología aplicada, permite la identificación de aspectos candidatos, controlar la posible redundancia entre aspectos y determinar según el flujo cuales de los aspectos puede llegar a ser dinámico.

Proceso de Identificación de Aspectos Candidatos:

Aplicando lo designado en la metodología, para la identificación de aspectos candidatos se diseña lógicamente una matriz de descomposición, está matriz es obtenida por las influencias y dependencias para cada una de las funcionalidades. Al final de la contabilización, si la cantidad de influencias para cada función es mayor que la media, ya sea en el número de dependencias o el número de influencias este aspecto se determinará como aspecto candidato que corta la clase donde se encuentra dicha función.

A su vez, a cada aspecto candidato se le asocian sus elementos, basados de igual forma en la metodología:

- Puntos de Corte del aspecto hacia la función que intercepta.
- Puntos de Unión que vinculan el objeto CORTE bajo la clase determinada.
- Puntos de Advice antes de la ejecución (before) si el aspecto candidato es obtenido de una relación con muchas dependencias, (after) si el aspecto candidato es obtenido por una relación de muchas influencias y (around) si esta viene precisamente de un <include> (una relación include a la funcionalidad que no presenta dependencias)
- Debido a lo explicado en la metodología, le herramienta no aplica autogeneración para elementos Instructions.

```
RELACIONES:
case CORTE/*: + case ASPECTO/*

case CORTE/*: + case CLASE/*
case ASPECTO/CUTPOINT + case ASPECTO/JOINPOINT
case ASPECTO/CUTPOINT + case ASPECTO/ADVICE
case ASPECTO/CUTPOINT + case CLASE/FUNCTION
case ASPECTO/CUTPOINT + case */FUNCTION
case ASPECTO/JOINPOINT + case ASPECTO/CUTPOINT
case ASPECTO/ADVICE + case ASPECTO/CUTPOINT
case ASPECTO/ADVICE + case CLASE/*
case ASPECTO/INSTRUCT + case CLASE/*
case ASPECTO/* + case CORTE/*
case ASPECTO/* + case CLASE/*
case ASPECTO/* + case ASPECTO/*
case CLASE/FUNCTION + case ASPECTO/CUTPOINT
case CLASE/* + case CORTE/*
case CLASE/* + case ASPECTO/ADVICE
case CLASE/* + case ASPECTO/INSTRUCTION
case CLASE/* + case ASPECTO/*
case */FUNCTION + case ASPECT/CUTPOINT


INFLUENCIAS / DEPENDENCIAS:
case CLASE/FUNCTION + case CLASE/FUNCTION
case CLASE/FUNCTION + case */FUNCTION
case */FUNCTION + case CLASE/FUNCTION
case */FUNCTION + case */FUNCTION

PERTENENCIAS:
case ASPECTO/* + case */FUNCTION
case CLASE/* + case */FUNCTION
case */FUNCTION + case CLASE/*
case */FUNCTION + case ASPECT/*
```

Imagen 12. Estructura de relaciones permitidas para cada opción.

El código representativo de este proceso se encuentra a continuación:

```
media = media / inf.length ;
for (a = 0; a < inf.length; a++ ){
if (inf[a] / media > 1){
xmethods.add(xp.GetMethods().get(a));
xreturn.add(enum_elementos.AFTER.getvalor());
if (xp.GetMethods().get(a).GetDependencias().size() == 1){
Methods posible = xp.GetMethods().get(a).GetDependencias().get(0);
if (posible.GetDependencias().size() == 0){
xmethods.add(posible);
xreturn.add(enum_elementos.AROUND.getvalor());
}
}
}
}
media = media / inf.length ;
for (a = 0; a < inf.length; a++ ){
if (inf[a] / media > 1){
xmethods.add(xp.GetMethods().get(a));
xreturn.add(enum_elementos.BEFORE.getvalor());
}
}
```

En donde, el primer “ciclo for” representa la contabilización de las influencias, haciendo caso especial que dependa de una función sin dependencias (para el caso del include), el segundo ciclo aplica lo mismo para el caso de las dependencias del elemento evaluado.

Proceso de Control de Redundancia

El proceso de control de redundancia se encarga de comparar todos los aspectos existentes tanto en componentes como en funciones y clases asociadas, dando una valoración mayor a las funcionalidades vinculadas al mismo.

La valoración establecida es la siguiente: 2puntos de equivalencia para relaciones hacia una función, 2puntos por advice relacionados a clases, 1punto por instruction asociado y 1punto por método existente. Estos valores se justifican debido a la etapa de Ingeniería de Requerimientos se plantean los requerimientos y las funcionalidades, y las evaluaciones se realizan en base al flujo de estas funcionalidades y objetivos.

Para determinar la existencia de redundancia se verifica si el 85% de similitud en los aspectos, dado por la cantidad de puntos de equivalencias. El aspecto con menor cantidad de elementos será eliminado si se aplica el método del botón "SOLVE".

Proceso de Determinación de Aspectos Dinámicos:

Para determinar los aspectos que pueden ser considerados dinámicos (la herramienta no los modifica automáticamente) se aplica, nuevamente, lo establecido en la metodología: Para las clases y funciones a las que intercepta un aspecto candidato, si estas presentan gran cantidad de dependencias de otras funciones y clases e influyen de manera muy escasa sobre otros elementos, entonces se le considera al aspecto candidato.

- **Capa3:** Esta sección permite agregar las relaciones mediante las validaciones explicadas anteriormente, y para las combinaciones entre los aspectos a evaluar (***case ASPECTO/* + case ASPECTO/*).*** Se aplica el método de resolución de conflicto como se presenta en la metodología.

Resolución de Conflictos entre Aspectos:

La herramienta permite valorizar y priorizar dos aspectos, determinando cual de los dos tiene más valoración dentro del flujo.

El proceso de determinación se enfoca en la valoración dado en la pestaña <MAIN> para cada función y clase relacionada con los elementos que se evalúan, al igual que las relaciones presentes en cada uno de ellos; la valoración utilizada por la herramienta es la siguiente:

- Por cada función relacionada a sus elementos: 1 punto por influencia, valoración de la función.
- Por cada clase asociada a sus elementos o asociada al aspecto en sí: 3 puntos por cada clase intervenida por un advice y 1 punto por un instruction, más valoración base de la clase (otorgada por el usuario).

Finalmente, el aspecto con mayor cantidad de puntos asociados será determinado con mayor prioridad. Este proceso no influye internamente en ningún aspecto, solamente, permite conocer cual tiene mayor importancia según ramificaciones lógicas y punto de vista del usuario.

4.1.3. PESTAÑA <XML>

Esta pestaña se encarga de permitir las acciones de guardar y abrir proyecto, además de ver la representación en XML de la estructura de los elementos básicos y secundarios que utiliza la herramienta para trabajar. La Imagen 13, muestra la pantalla referente a esta pestaña. Además, esta sección permite la generación del case del framework a través de un proyecto con los elementos y la estructura diseñada.

La estructura básica de un XML generado a través de la herramienta esta dado por:

```
<principal>
  <ALL_CORTES />
  <ALL_ASPECTS />
  <ALL_CLASSES />
  <ALL_FUNCTIONS  />
</principal>
```

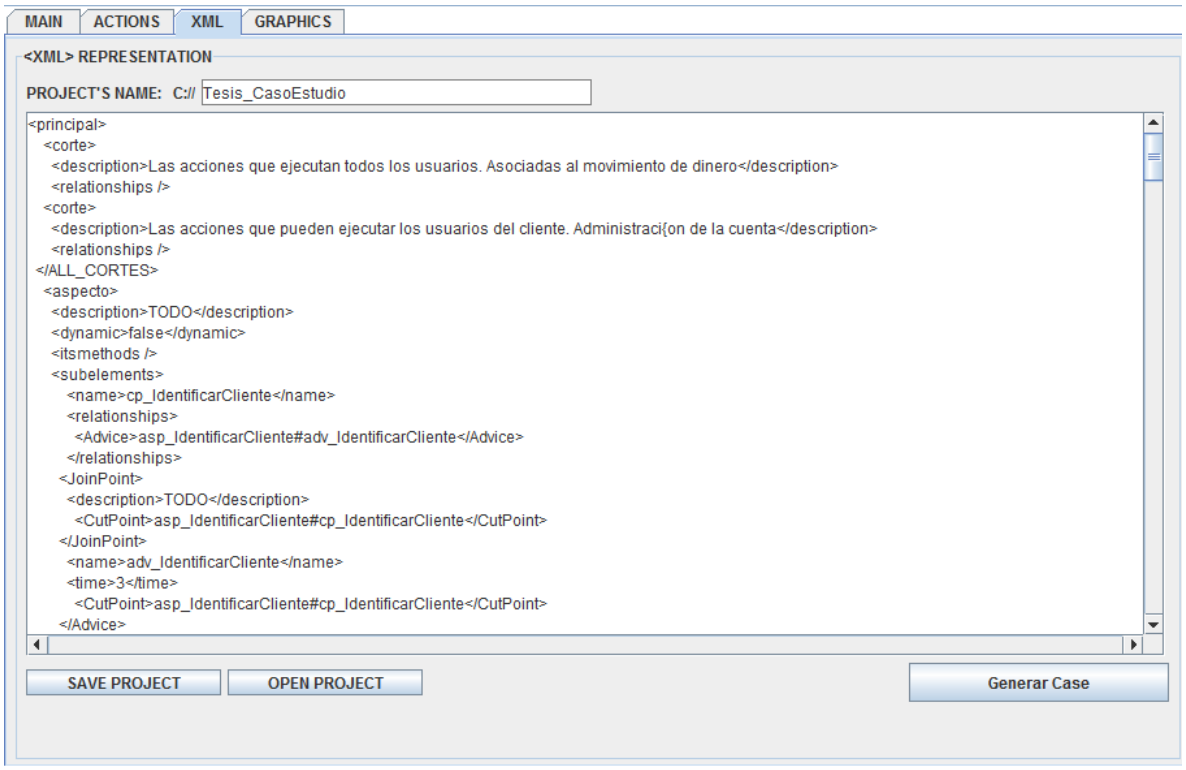


Imagen 13. Pantalla correspondiente a la pestaña XML.

El proyecto que se genera es un plug-in RCP en Eclipse, similar a los autogenerados al abrir un nuevo proyecto, este tendrá como nombre principal denominada "test" en las carpetas designadas por el usuario. Cada elementos estará dentro del paquete con el nombre del CORTE asociado, en caso de no tener alguno, se usara un paquete con nombre por defecto para guardarlos; similar sucede con las funciones que no están asociadas a las clases, las cuales se crearán asociada a un clase con un nombre por defecto. En la Imagen 14 se puede ver la estructura de un proyecto generado a partir de la metodología y la herramienta de apoyo; nótese que una vez estando dentro del proyecto, el nombre de los elementos puede reestructurarse a través de eclipse.

El proceso que se realiza consiste en escribir los aspectos, clases y funciones de acuerdo a las relaciones establecidas dentro de la herramienta, siguiendo los patrones determinados dentro del marco de referencia que describe a AspectJ, que es el framework utilizado en este desarrollo.

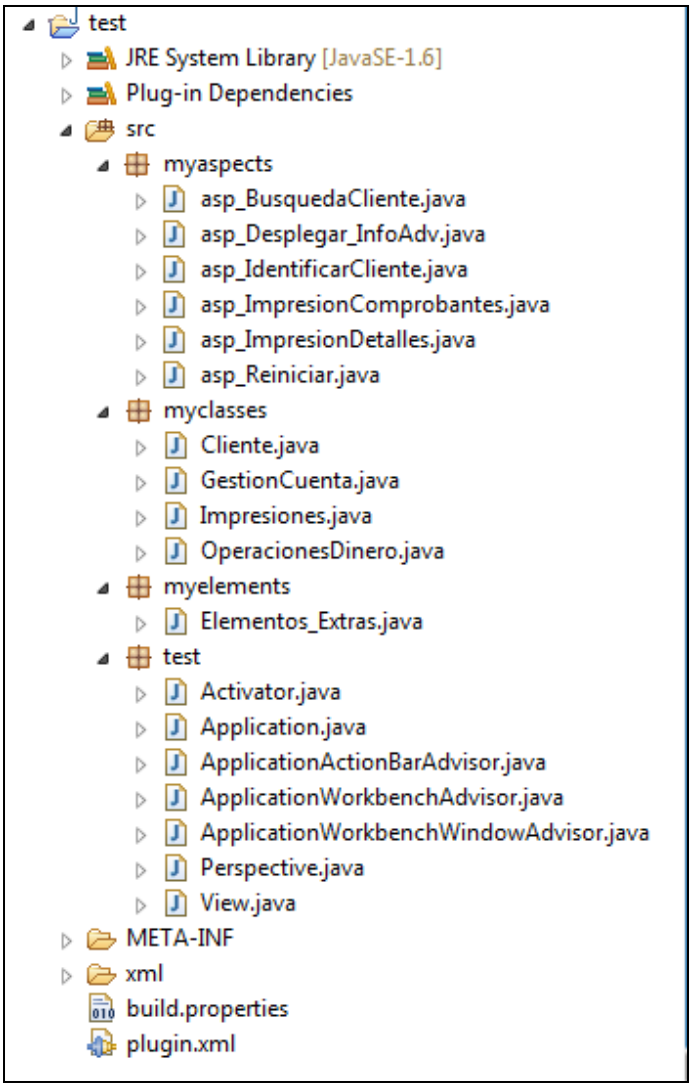


Imagen 14. Estructura de Archivos generada por la herramienta.

4.1.4. PESTAÑA <GRAPHICS>

Mediante la utilización de graphviz se generan las gráficas representativas del diseño lógico de los elementos existentes. La Imagen 15 muestra una representación gráfica de las relaciones entre las funcionalidades, clases y aspectos asociados a las mismas; los elementos dentro de diagrama de color azul hacen referencia a los

aspectos candidatos, los de color negro a las funcionalidades y los de color verde a las clases identificadas.

El propósito de esta sección es el de permitir la visualización gráfica del diseño lógico actual en la clase principal que contiene los elementos de la POA.

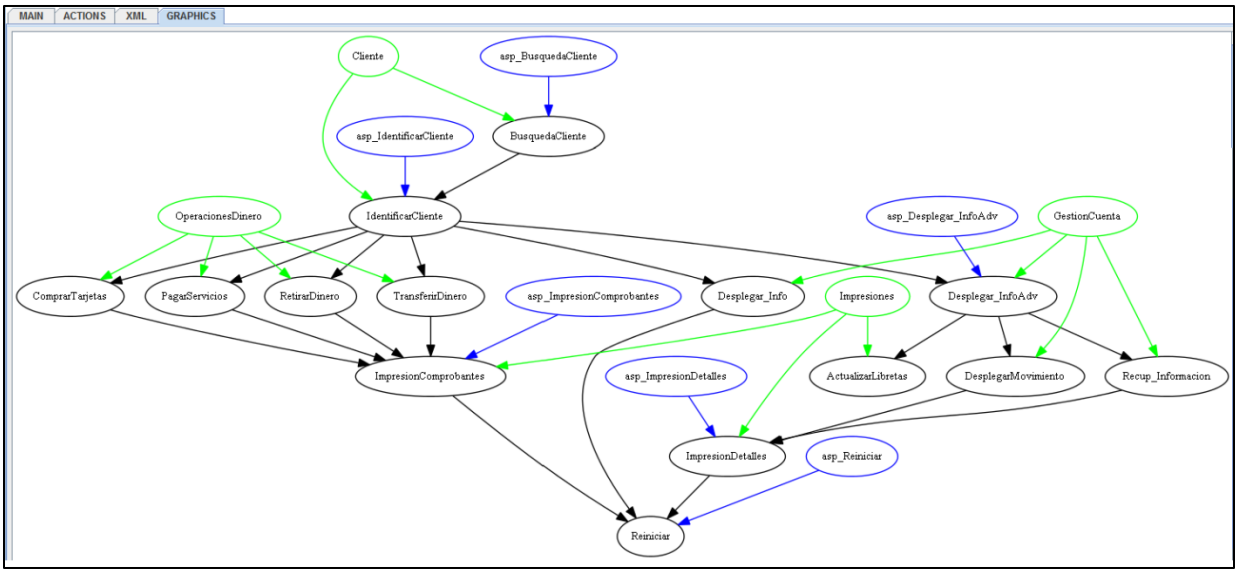


Imagen 15. Gráfica formada con graphviz que representa el diseño lógico del sistema.

4.2. Uso de la Herramienta de Apoyo:

Dentro de los requerimientos del sistema está el de instalar AspectJ para Eclipse y el de utilizar Eclipse 3.6 o posterior.

La manera de utilizar la herramienta de apoyo en base a la metodología se divide en las siguientes fases:

4.2.1. Fase 1

En la etapa de requerimientos de la Ingeniería de Software, posterior a la toma de requerimiento y empezando el análisis del sistema. Se elaboran los casos de uso asociados a las funcionalidades como determina la metodología, una vez se culmine el paso 3.2.1 Identificación de Influencias y Dependencias se plasma la información (elementos identificados) en la pestaña <MAIN>, no es importante determinar las clases o cortes asociados; con la matriz de influencias y dependencias es posible avanzar a la fase 2.

La adición de cada elemento se realiza con el botón situado en la parte derecha del nombre del elemento que se ingresa. La forma de borrar un elemento se realiza a través del listado de los mismos, mediante un menú contextual. Esta eliminación borrará al elemento y todos sus vínculos.

4.2.2. Fase 2

En la pestaña referente a las acciones <ACTIONS>, se crean todas las relaciones de los elementos identificados de la siguiente manera:

Se seleccionan los elementos en las listas y se agregan a la parte inferior a través de un menú contextual. Automáticamente, se seleccionará una relación posible para ellos (contiene, dependencia / influencia o está relacionado) permitiendo agregar la misma. Esta relación se verá reflejada en las listas localizadas en la parte superior izquierda, estos vínculos pueden eliminarse mediante un menú contextual.

El tipo de elemento que se selecciona se puede visualizar en dos partes: la selección básica del elemento en la parte superior y una vez adicionado para ser modificado en la parte inferior (Ver Imagen 16).

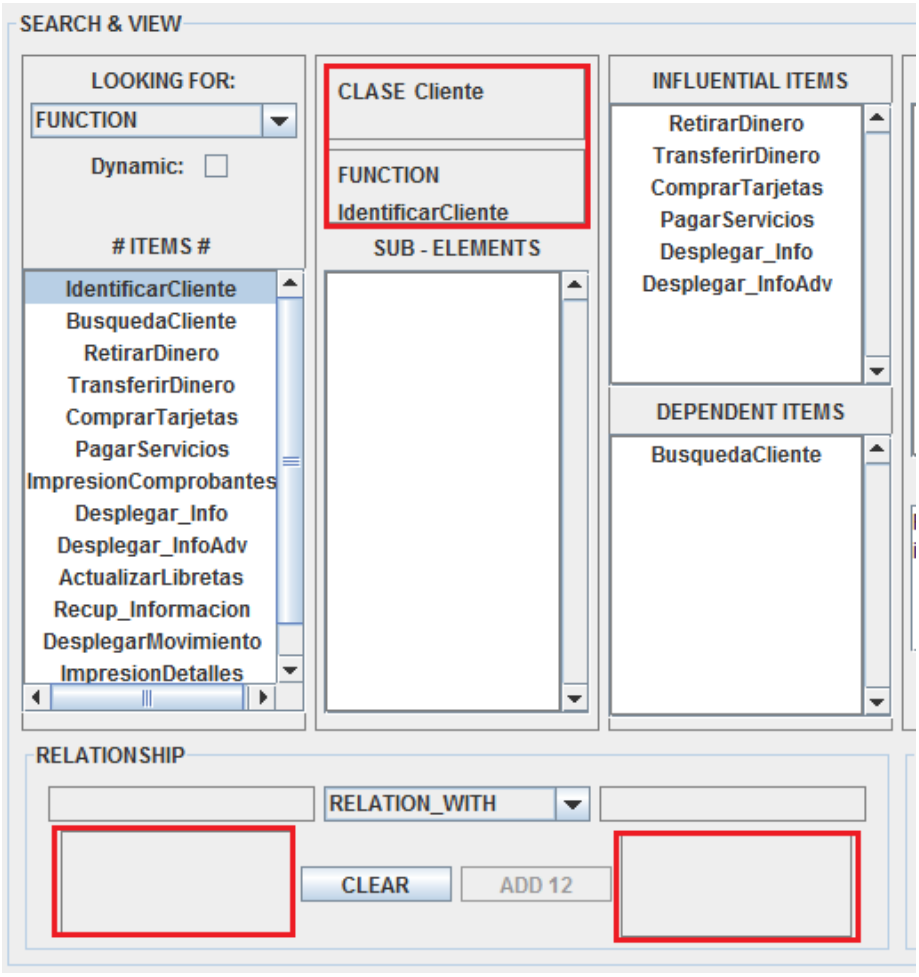


Imagen 16. Tipo de elemento seleccionado.

Una vez, todas las influencias y dependencias están colocadas, se pueden visualizar las relaciones en cualquier momento, en la pestaña de <GRAPHICS>; además de permitirle a la herramienta proponer los aspectos candidatos que el sistema tendrá y agregarlos al sistema con las relaciones entre sus elementos y a las funciones que interceptan. La herramienta permite agregar nuevos aspectos, funcionalidades y clases; y realizar cualquier vínculo lógicamente aceptable (relaciones válidas entre los elementos y subelementos, no aplica según la lógica de las funcionalidades.) entre sus elementos, permitiendo al aspecto intervenir en muchas clases y en muchas funcionalidades a través de uno o varios de sus elementos.

Para sistemas muy complejos, es necesario corroborar, tanto en el análisis como con la herramienta, la presencia de posibles redundancias. La herramienta

verificará los aspectos existentes y comparando sus intervenciones a clases y métodos, estimando como redundante cualquier aspecto con más del 85% de similitud con otro. La herramienta muestra ambos aspectos redundantes y el porcentaje de similitud en la lista.

Posteriormente, una vez identificados y relacionados los aspectos, se utiliza la opción de verificar cuales son los aspectos cuya presencia dinámica no afectará el equilibrio del sistema. La herramienta determinará estos aspectos basados en la metodología, pero no los actualiza instantáneamente, sino que permite al usuario su cambio manual mediante el control de chequeo dinámico (Ver Imagen 17).

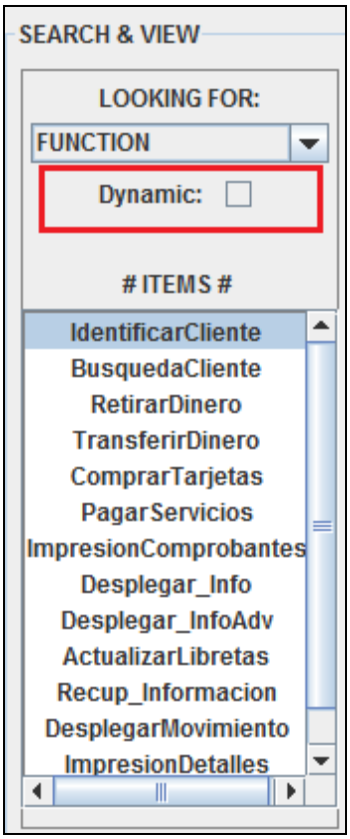


Imagen 17. Control de Aspectos dinámicos

4.2.3. Fase 3

Una vez toda la estructura lógica es añadida, se pasa a la pestaña <XML>, en esta sección se genera el case (componentes .java y archivos de eclipse) del

proyecto lógico diseñado. Los mismos se generarán en un paquete con el corte funcional señalado, en caso de no estar asignado, se generará en uno con nombre por defecto.

Como se expresó en el marco referencial**,** existe una gran cantidad de maneras para ejecutar un punto de corte y un punto de unión en AspectJ, por eso, la herramienta de apoyo genera los mismos a través de execution() como manera predeterminada. Todos los comentarios y enlaces serán visibles en el código generado con el objetivo de mantener la idea de los mismos.
El proyecto está listo para ser continuado, una vez se llegue a la etapa de desarrollo.

4.3. Acciones Opcionales:

Para guardar y abrir un proyecto se utiliza las opciones de la pestaña <XML> la cual abre una ventana (Ver Imagen 18) que permite seleccionar un archivo, de esta manera se cargan todos los valores existentes en la estructura del XML.

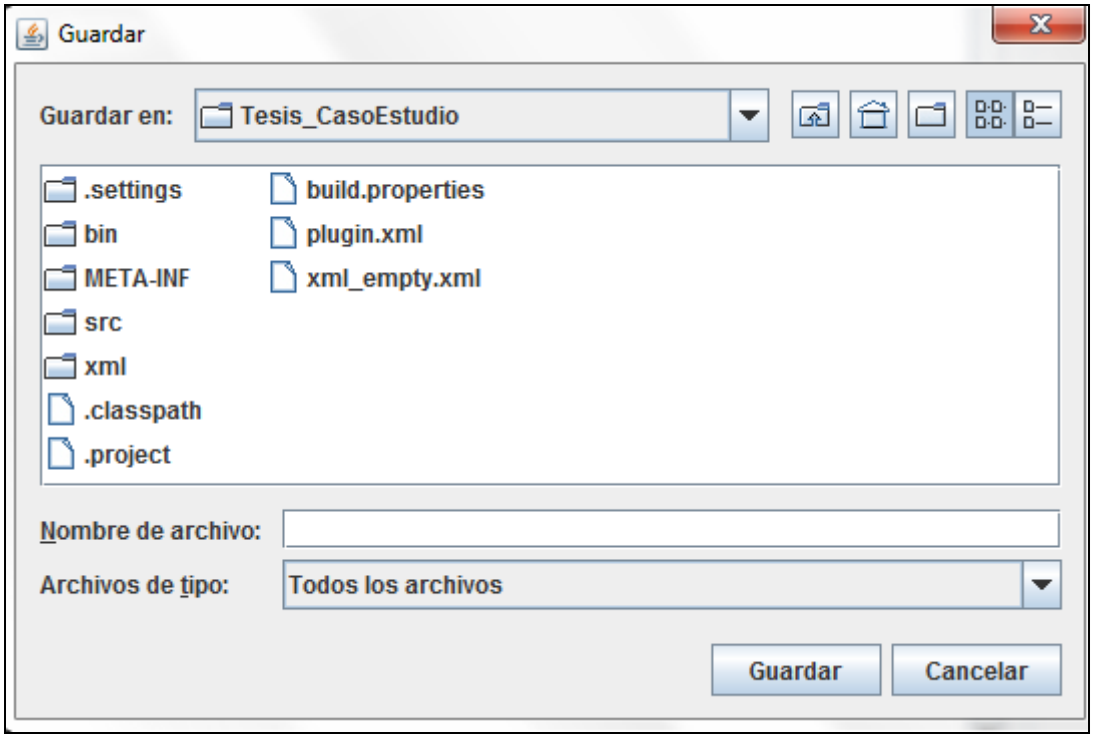


Imagen 18. Formulario de búsqueda de carpetas

En caso de presenciar un conflicto entre los aspectos, se tiene la posibilidad de valorar cuál de ellos tiene más prioridad, está sección se encuentra en la parte inferior (Ver Imagen 19) y estará válida para cuando sean agregados dos aspectos.

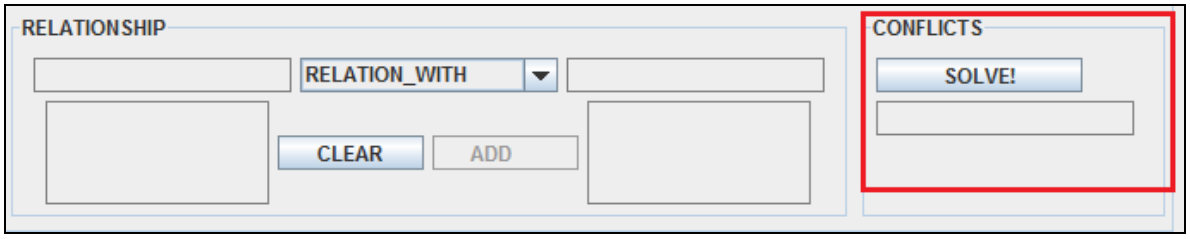


Imagen 19. Resolución de Conflictos

A continuación, las principales características de los principales elementos expuestos en la herramienta:

Los Cortes

- Son las funcionalidades del sistema y presentan un corte transversal en particular
- **Un corte transversal puede ocupar muchos aspectos**

Los Aspectos

- El aspecto es único para todas las estancias de una clase que utiliza el mismo, por lo que los valores se mantienen.
- Los aspectos pueden extender las clases, pero no se pueden instanciar. Los aspectos pueden extender otros aspectos que sean abstractos.
- Los aspectos interceptan varias clases a través de las funciones,
- Pueden tener métodos y atributos.
- No representan ninguna entidad en particular y no pueden encapsularse con facilidad.
- **Cada aspecto debe pertenecer a un corte**

Puntos de Corte

- Permite la selección y ubicación de los puntos de unión (JP) dentro del aspecto.
- Son utilizados por los avisos.

- Son utilizados para crear otros puntos de corte a través de decisiones booleanas.
- Pointcut x(): call(función(x,y));
- Se asocian a los puntos de unión.
- Interceptan los métodos que se ejecutaran.

Puntos de Unión

- Hacen referencia a las llamadas.
- No se manifiestan explícitamente en el código.
- Hacen referencia a métodos, llamadas, definiciones, etc
- Los puntos de corte están asociados a ellos

Advice

- Interceptan Métodos en particular
- La clausula around puede necesitar el nombre de la clase.
- Puede intervenir: before(), around(), after().
- Utilizan los Puntos de Corte para su ejecución.

Instruction

- Permite la introducción de variables, métodos.
- **Interceptan las clases.**

Funciones

- Los métodos pertenecen a clases o aspectos; en caso de no ser identificado la herramienta permite no asociarlo a ninguna clase.
- Las funciones influyen en otras funciones.

Capítulo 5

Caso de Aplicación y Análisis de Resultados

Se ha diseñado el siguiente caso de estudio en donde se aplicará la Metodología propuesta en conjunto con la Herramienta de Apoyo con la finalidad de representar la visión lógica del proyecto en la primera etapa de la ingeniería de software, la ingeniería de requerimientos. Los resultados esperados radican en la identificación de los aspectos y sus sub elementos, al igual que definir de forma lógica las intercepciones a los métodos de las clases correspondientes, finalizando con la generación del case de framework como aplicación de Eclipse.

El caso de estudio que se presenta se basa en el presentado por Yu-Ning y colaboradores para su metodología de Modelos por Puntos de Vistas (Yu-Ning and Qiang 2009), el cual trata sobre una maquina de ATM (Cajero Automático como terminal bancaria) que contiene un software para manejar el hardware, dar soporte al usuario y comunicarse con la base de datos del banco. Al mismo tiempo, se le ha agregado funcionalidades extras del sistema, de tal forma que sea posible aplicar todos los pasos de la metodología, ya que algunos pasos no son imprescindibles en proyectos pequeños.

El sistema que se necesita se describe a continuación:
Se requiere un software para una maquina ATM que le permita realizar las siguientes operaciones:

- Aceptar las solicitudes de clientes.
- Permitir retirar dinero en efectivo.
- Proporcionar información de cuenta.
- Permitir transferencia de saldo.

- Proporcionar reconocimiento de usuarios del banco y usuarios foráneos al mismo.
- El sistema debe estar disponible 24 horas al día.

Operaciones adicionales para la aplicación de todos los pasos de la metodología:

- Admitir la compra de Tarjetas Telefónicas y Boletos de Transporte.
- Permitir el pago de servicios públicos, por ejemplo, cuentas de agua, teléfono.
- Permitirle al usuario la impresión comprobantes de transacciones, estados de cuenta y demás funciones.
- * Admitir la visualización de los movimientos bancarios de la cuenta.
- * Permitir recuperar información de cuentas y contraseñas del usuario.
- * Actualización de libretas de ahorro.
- Las funcionalidades marcadas con asterisco (*) sólo deben ser válidas para usuarios del banco.
- El módulo de presentación de la información de la cuenta, debe presentar datos especiales para usuarios internos.

Los requerimientos presentados son funcionalidades (RF), a excepción de la disponibilidad de 24 horas, que no representa una función propiamente dicha (RNF). Planteados los requerimientos del sistema, se procede a aplicar la metodología propuesta:

Por conveniencia, para trabajar con dos módulos, este caso de estudio se ha divido la conceptualización lógica en dos secciones: Acciones básicas del ATM que son las funcionalidades para todo usuario y hacen referencia a los movimientos de la cuenta y Acciones completas del ATM que son las funciones de administración; también es posible trabajar todo el proyecto como un solo módulo.

5.1. Etapa1: Identificación de Asuntos

5.1.1. Enfoque a Vistas e Identificación de elementos claves

Se tienen dos puntos de vistas: el punto de vista por clientes foráneos y por clientes propios. Debido a que la metodología no altera procedimientos bases de la Ingeniería de Software, este procedimiento se realiza de manera cotidiana.

En la Tabla 11, se presenta el diagrama de vistas para el usuario interno, en la Tabla 12 para el usuario externo y en la Tabla 13 una compactación de las vistas.

Usuario Interno. Vista para el Módulo 1	
Actores	Usuario Interno
Objetivo	Movimiento de la Cuenta bancaria
Características	Permitir Movimiento y Manipulación del dinero
Casos de Uso	Retiro de dinero Transferencia de saldo Pago de Servicios Compra de Tarjetas y Boletos Impresión de Comprobantes Reconocimiento del Usuario
Usuario Interno. Vista para el Módulo 2	
Actores	Usuario Interno
Objetivo	Administración de la Cuenta bancaria
Características	Controlar, Manejar y Recuperar Información de la Cuenta
Casos de Uso	Visualización de Movimientos Actualización de Libretas Visualización Avanzada de las cuentas Recuperación de Información de la Cuenta Impresión de Detalles de la cuenta Reconocimiento del Usuario*

Tabla 11. Vistas por Usuario Interno.

Usuario Externo. Vista para el Módulo 1	
Actores	Usuario Interno
Objetivo	Movimiento de la Cuenta bancaria
Características	Permitir Movimiento y Manipulación del dinero
Casos de Uso	Retiro de dinero Transferencia de saldo Pago de Servicios Compra de Tarjetas y Boletos Impresión de Comprobantes Brindar Información Básica de la cuenta Reconocimiento del Usuario*
Usuario Externo. Vista para el Módulo 2	
Actores	Usuario Interno
Objetivo	Administración de la Cuenta bancaria
Características	Visualizar información básica de cuenta
Casos de Uso	Mostrar información básica de cuenta

Tabla 12. Vistas por Usuario Externo.

El caso de uso de Reconocimiento del Usuario, se dividirá posteriormente en dos sub casos: Identificación del Cliente y Búsqueda del Cliente.

Agregamos la clasificación por vistas.

ID Vista	Actores Involucrados	Funcionalidades	Dependencias
Usuario Interno			
- **Vista 1**	Usuario Interno	6 funciones	No aplica
- **Vista 2**	Usuario Interno	5 funciones	No aplica
Usuario Externo			
- **Vista 1**	Usuario Externo	7 funciones	No aplica

Tabla 13. Clasificación Por Vistas

5.1.2. Elaboración de Casos de Uso.

Mediante los requisitos se elaboran los diagramas de casos de uso, para el caso de aplicación, se tiene dos diagramas de casos de uso. La

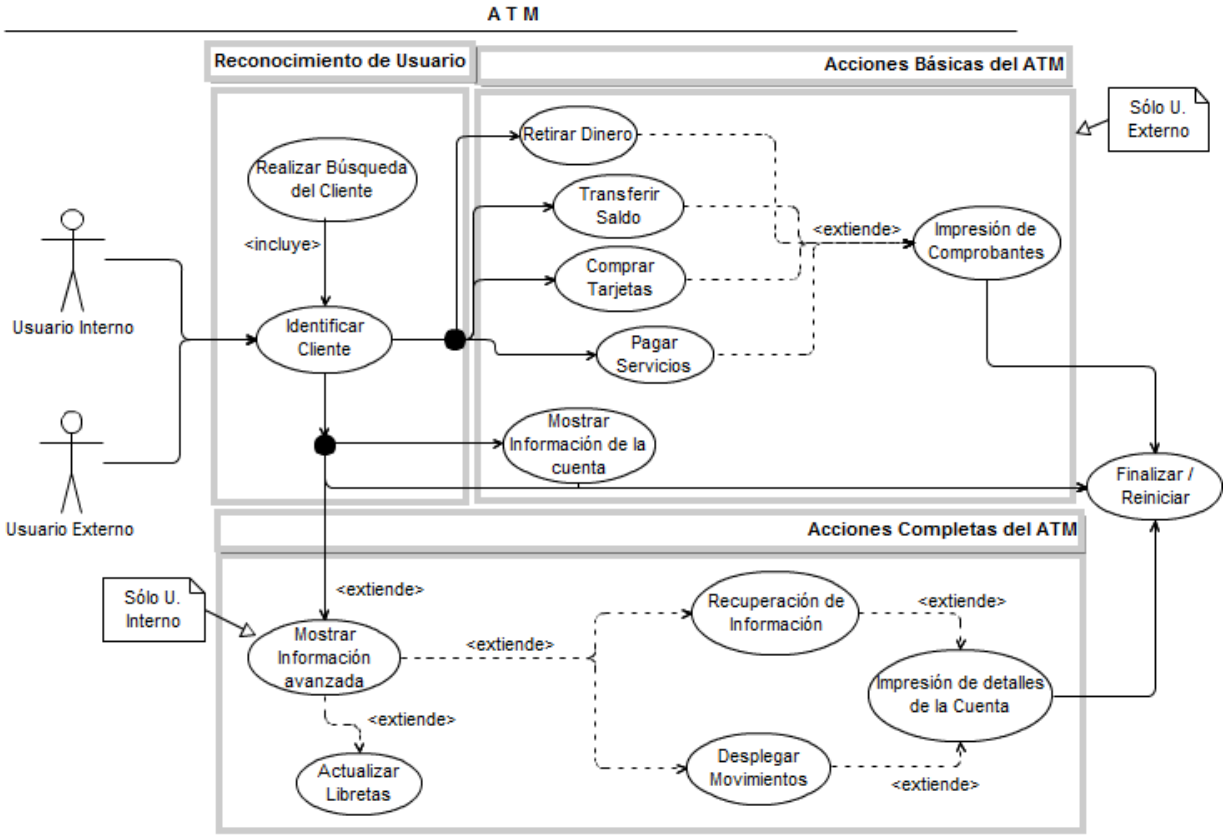
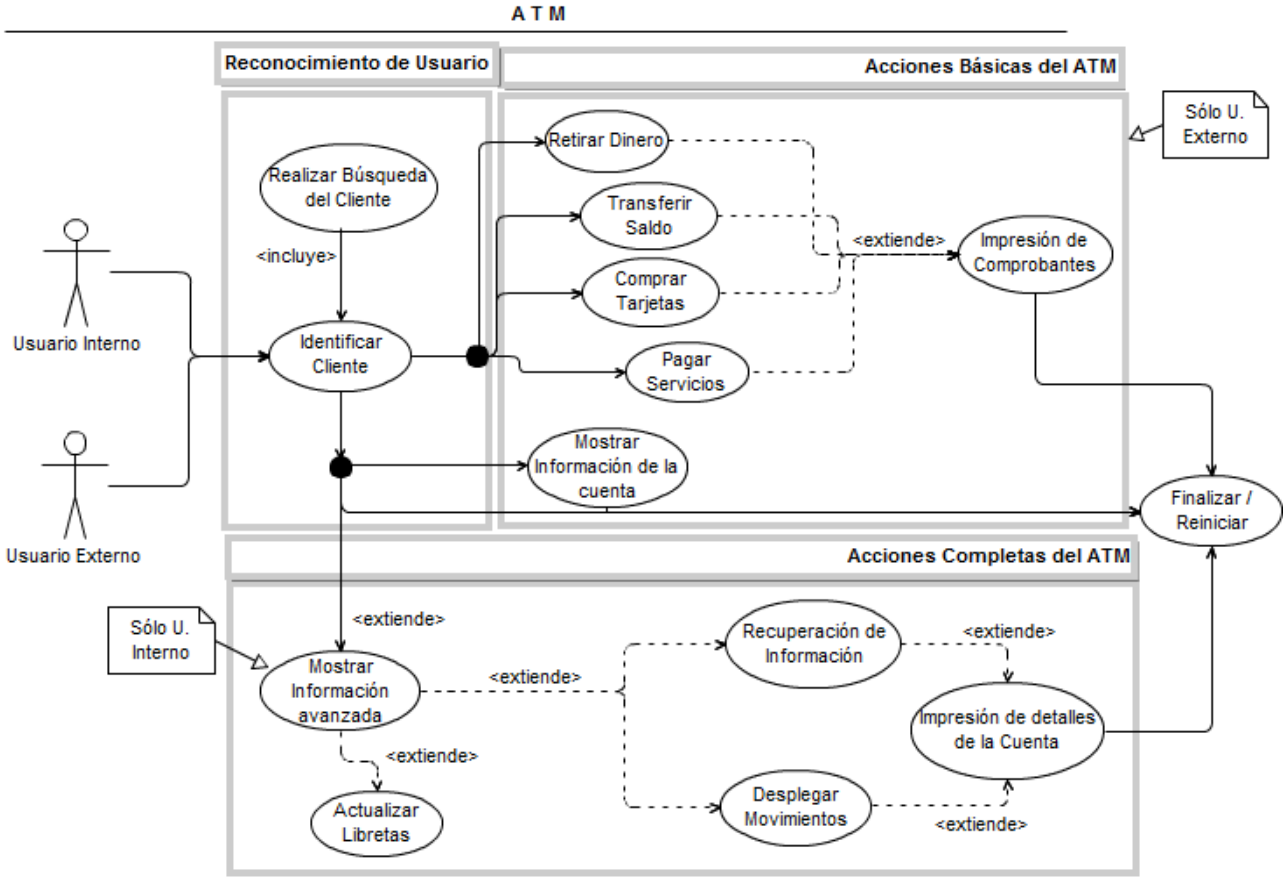


Imagen 20 muestra el diagrama de casos de uso unificado, representa las acciones que realiza el sistema ATM, a este diagrama se le ha agregado de fondo contenedores con el fin de ver la relación entre el diagrama y las demás etapas de la metodología, la aplicación de la misma no es necesaria en análisis reales.


A T M
Reconocimiento de Usuario
Acciones Básicas del ATM
Sólo U. Externo
Realizar Búsqueda del Cliente
Retirar Dinero
Transferir Saldo
Comprar Tarjetas
Pagar Servicios
<extiende>
Impresión de Comprobantes
<incluye>
Usuario Interno
Identificar Cliente
Mostrar Información de la cuenta
Finalizar / Reiniciar
Usuario Externo
Acciones Completas del ATM
Sólo U. Interno
<extiende>
Mostrar Información avanzada
<extiende>
Recuperación de Información
<extiende>
Impresión de detalles de la Cuenta
<extiende>
Actualizar Libretas
Desplegar Movimientos
<extiende>

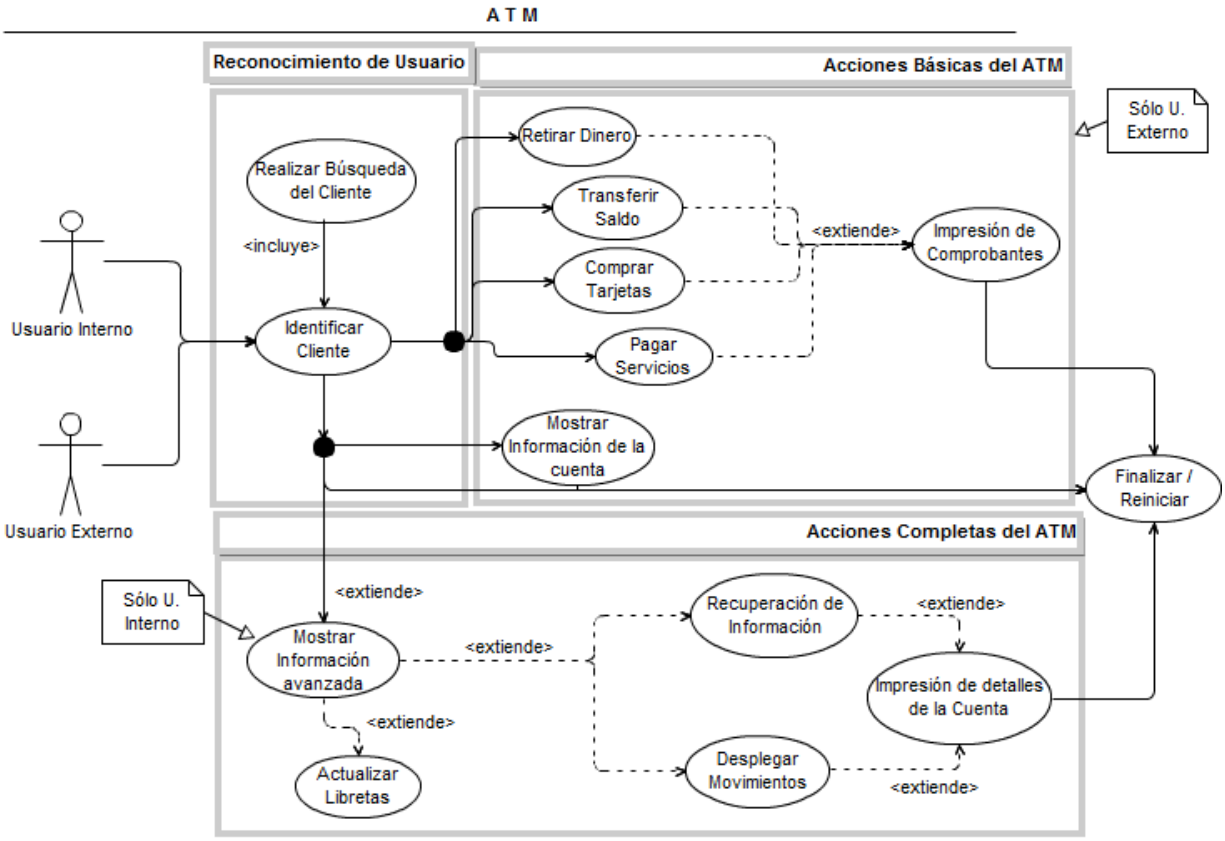


Imagen 20. Diagrama de Casos de Uso.

Los casos de uso obtenidos son un total de 14 y son los siguientes:

Proveer Servicio (no mostrado, requerimiento del usuario): Validaciones para comprobar la capacidad del ATM de procesar información.

#1 Identificar de Cliente.

#2 Realizar Búsqueda de Cliente: Búsqueda del Banco afiliado a la tarjeta.

#3 Retirar Dinero.

#4 Transferir Dinero.

#5 Comprar de Tarjetas.

#6 Pagar Servicios.

#7 Impresión de Comprobantes: Permite la generación de tiquetes que sirven como comprobante del trámite realizado.

#8 Mostrar Información de Cuenta: Verificar saldo e información de la misma, contenido y acciones limitadas para usuarios externos del banco.

#9 Mostrar Información Avanzada de la Cuenta: permite verificar saldo para usuarios internos del banco, mostrará información similar al caso de uso de Mostrar Información de Cuenta, pero con variaciones y extendiendo a

#10 Actualizar Libretas. Imprime los saldos de los últimos movimientos.

#11 Recuperación de Información: Permite que el usuario recupere datos privados de banca en línea o de su cuenta.

#12 Desplegar Movimientos: Refleja en pantalla las transacciones, retiros, ingresos en la cuenta.

#13 Impresión de detalles de la cuenta: Imprime la información de los trámites anteriores.

#14 Finalizar / Reiniciar: Elimina toda la información y reinicia el sistema.

5.1.3. Contabilización de identificadores determinante de asuntos.

Aplicando la regla de descomposición de asuntos determinantes tenemos la Tabla 14, en donde se evalúa la frecuencia de los identificadores léxicos de los casos de uso del paso anterior.

Identificadores Léxicos	Obtenido de	Frecuencia
Buscar	Búsqueda de Cliente	01
Identificar	Identificar Cliente	01
Retirar	Retirar Dinero	01
Transferir	Transferir Saldo	01
Comprar	Comprar Tarjeta	01
Pagar	Pagar Servicios	01
Impresión	Impresión de Comprobantes Impresión de Detalles Actualizar Libretas	03
Recuperación	Recuperación de Información	01
Presentar	Información de Cuenta Información Avanzada	02

Tabla 14. Repositorio de Identificadores Léxicos

5.1.4. Identificación de Módulos Multifuncionales

Para el sistema propuesto, dentro del concepto de módulos multifuncionales, tenemos los casos de uso de Mostrar Información Avanzada y Mostrar Información de la Cuenta (básica), debido a como se encuentra especificado en los casos de uso de comportamiento similar, pero dependiendo del usuario otorga un comportamiento diferente, este es un caso de uso multifuncional. Otra relación de módulo multifuncional sería la posibilidad de expandir el sistema y que en un futuro permitiese el pago de servicio a través de banca en línea, debido a la previa existencia (en este ejemplo) de un módulo para este proceso, este llegaría a ser multifuncional.

El módulo multifuncional existente es: Mostrar Información de la cuenta. (Presente en ambos módulos: cortes)

5.2. Etapa 2: Identificación de Aspectos:

5.2.1. Identificación de Influencias

Tomando como referencia el diagrama de caso de uso, tenemos la Tabla 15 que representa la matriz de influencia y dependencias de las funcionalidades, en donde la fila está relacionada con las columnas por uno de los vínculos <extendido>, <incluido> u otras especializadas en la metodología.

FUNCIONALIDADES	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1 Identificar Cliente	*		→	→	→	→		E	E					
2 Realizar Búsqueda de Cliente	I	*												
3 Retirar Dinero			*				E							X
4 Transferir Saldo				*			E							X
5 Comprar Tarjetas					*		E							X

6 Pagar Servicios		*	E										X
7 Impresión de Comprobantes			*										→
8 Mostrar Información de la cuenta				*									→
9 Mostrar Información avanzada de la cuenta					*	E	E	E					
10 Actualizar Libretas						*							X
11 Recuperación de la Información							*			E			X
12 Desplegar Movimientos								*		E			X
13 Impresión de detalles de la cuenta										*			→
14 Finalizar / Reiniciar													*

Tabla 15. Matriz de Dependencias e Influencias

En la Tabla 15, se relacionan los elementos utilizando abreviaturas debido al tamaño de las relaciones. El significado de la simbología se ve en la Tabla 16.

Relación	Simbología	Significancia
Flujo	→	Funcionalidad posterior en el flujo
<Incluye>	I	Influencia fuerte sobre f()
<Extiende>	E	Puede extender f()
Finaliza	X	Puede finalizar directamente.

Tabla 16. Simbología para la Matriz de Dependencias e Influencias

Finalmente, en este paso se tiene las dependencias e influencias totales dadas por la sumatoria en la Tabla 15, la misma se detalla en Tabla 17. La sumatoria vertical para cada funcionalidad representa el total de dependencias de la misma, la sumatoria horizontal representa la cantidad de influencias que ejerce esta función.

FUNCIONALIDADES	Dep.	Inf.
1 Identificar Cliente	**1** (1I)	**6** (2E)
2 Realizar Búsqueda de Cliente	**0**	**1** (1I)
3 Retirar Dinero	**1**	**1** (1E)
4 Transferir Saldo	**1**	**1** (1E)
5 Comprar Tarjetas	**1**	**1** (1E)
6 Pagar Servicios	**1**	**1** (1E)
7 Impresión de Comprobantes	**4** (3E)	**1**
8 Mostrar Información de la cuenta	**1** (1E)	**1**
9 Mostrar Información avanzada de la cuenta	**1** (1E)	**3**
10 Actualizar Libretas	**1** (1E)	**0**
11 Recuperación de la Información	**1** (1E)	**1**
12 Desplegar Movimientos	**1** (1E)	**1**
13 Impresión de detalles de la cuenta	**2** (2E)	**1**
14 Finalizar / Reiniciar	**3**	**0**

Tabla 17. Matriz de Dependencias e Influencias Generalizada

En esta sección se aplica la herramienta de apoyo. En la pestaña MAIN se ingresan todas las funcionalidades expresadas y en la sección ACTIONS se establecen las relaciones, como se muestra en la Imagen 21 y la Imagen 22 correspondiente.

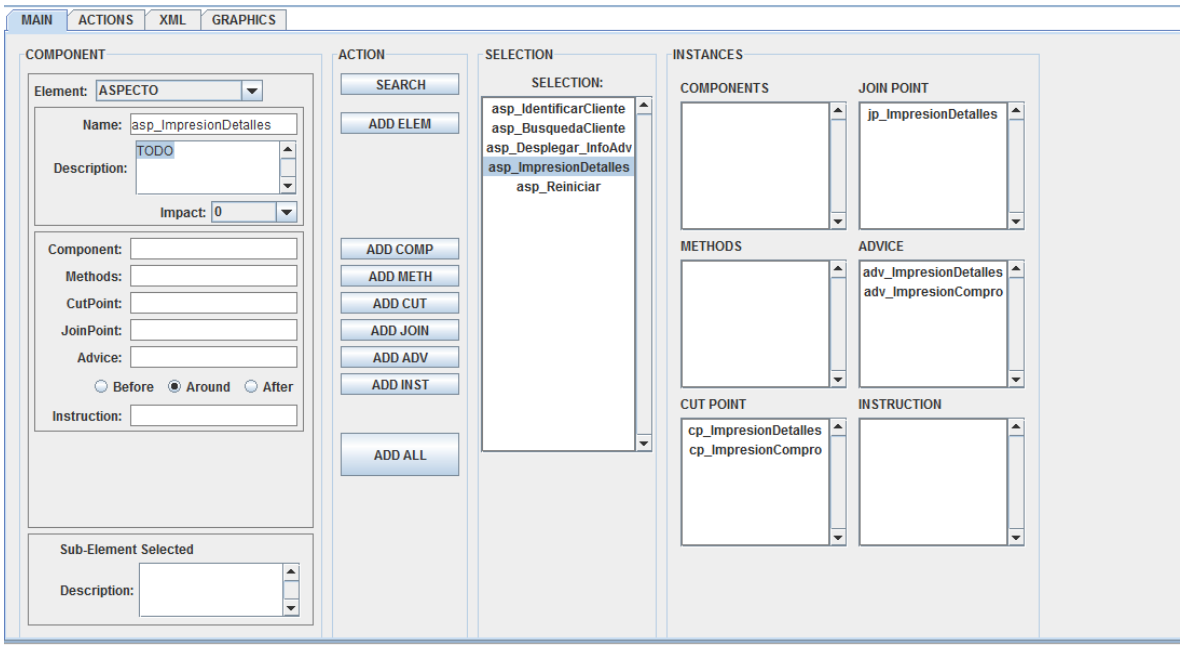


Imagen 21. Elementos y sub-elementos presentados en la primera pestaña.

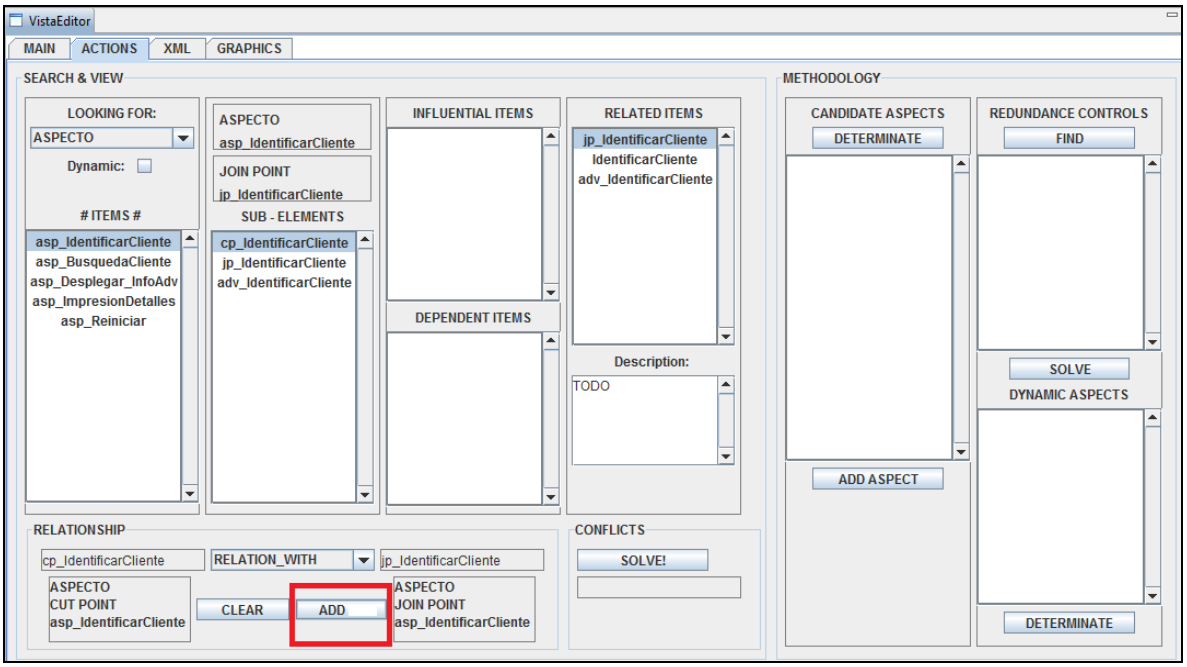


Imagen 22. Ingreso de relaciones y dependencias en la pestaña de ACTIONS

5.2.2. Aplicación de la matriz de descomposición

Como se presenta en la matriz de descomposición para diagramas (Ver Tabla 17), se tiene que los casos de uso con mayor número de influencias y dependencias son:

- Identificar Cliente, 6 influencias.
- Mostrar Información avanzada de la cuenta, 3 influencias.
- Impresión de Comprobantes, 4 dependencias.
- Impresión de Detalles, 2 dependencias (Pocas dependencias).
- Finalizar / Reiniciar, 3 dependencias.

Aplicando la clausula incluye se tiene:

- Realizar Búsqueda de Cliente.

Se tienen 5 posibles aspectos y un aspecto dudoso (que corta a impresión de Detalles) a través de este método.

5.2.3. Aspectos a través de Requerimientos No Funcionales

El usuario ha proporcionado como requerimiento no funcional la disponibilidad (Al querer el servicio disponible las 24 horas del día). Pero a su vez, podemos escoger aspectos que se apliquen al sistema.

Los aspectos candidatos a través de requerimientos no funcionales son los siguientes:

- Disponibilidad
- Confiabilidad de Procesos
- Nivel de Seguridad
- Rendimiento del Sistema
- Tiempo de Servicio

Debido a que los aspectos obtenidos de requerimientos no funcionales son abstractos, y generalmente tienen una presencia global en todo el sistema, en la Ingeniería de Requerimientos, no es necesario el detallado de los mismos.

5.2.4. Aspectos Candidatos

Combinando ambos procesos de identificación, tenemos:

El aspecto que cruza “Impresión de detalles” con un bajo nivel de dependencias entra a formar parte de los mismos por su presencia en la Tabla 14. Repositorio de Identificadores Léxicos, teniendo Impresión como identificador, lo cual ratifica su presencia.

Por ende, se tiene que los aspectos candidatos en este sistema son los siguientes:

- Aspecto a Identificar Cliente.
- Aspecto a Mostrar Información avanzada de la cuenta.
- Aspecto a Impresión de Comprobantes.
- Aspecto a Impresión de Detalles.
- Aspecto a Finalizar / Reiniciar.
- Aspecto a Realizar Búsqueda de Cliente.

El resultado proporcionado por la herramienta se presenta en la Imagen 23, la cual determina estos 6 aspectos como candidatos.

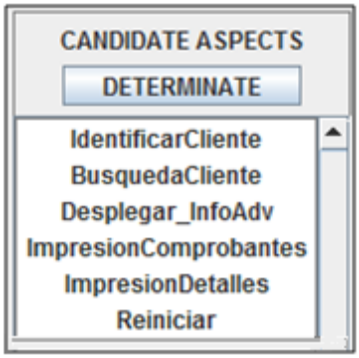


Imagen 23. Aspectos candidatos

Como interpretación lógica del sistema, los seis aspectos candidatos tienen razón lógica de existir, debido a las siguientes razones:

- Los aspectos que interceptan a las clases y funciones de Identificar Cliente y Mostrar Información avanzada de la cuenta, presentan seis

funcionalidades, si modificación directa podrá afectar cualquiera de ellas, un aspecto para el control de elementos generales llegará a ser necesario.

- Los aspectos que interceptan a las clases y funciones de Impresión de Comprobantes, Impresión de Detalles, Finalizar / Reiniciar, presentan la misma cualidad de las anteriores, con la diferencia que las modificaciones en las funciones de las que dependen pueden afectar esta y un aspecto que controle cambios indiferentes de la clase llegará a ser necesario.
- La justificación del aspecto que intercepta la clase de la función Búsqueda de Cliente, se da debido a que la única función sobre la cual influye tiene un aspecto asociado y se ejecutará siempre que se ejecute la misma.

5.2.5. Selección de Aspectos Clásicos y Aspectos Dinámicos

Aplicando la metodología, según las ramificaciones asociadas a cada una de las funciones principales que interceptan un aspecto, haciendo referencia a la Tabla 17. Matriz de Dependencias e Influencias Generalizada y observando la Tabla 17. Matriz de Dependencias e Influencias Generalizada, se tiene que muchas funciones presentan ramificaciones considerables para el tamaño del proyecto, siendo los únicos dos aspectos con funciones relacionadas con un pequeño número de ramificaciones y algunas dependencias: Impresión de Detalles e Impresión de Comprobantes.

La herramienta de apoyo facilita el proceso y determina los dos aspectos que pueden llegar a ser dinámicos (Ver Imagen 24).

5.2.6. Integración de Aspectos Candidatos

De los aspectos candidatos, dos de ellos están asociados bajo el nombre de “Impresión” en la Tabla 14. Repositorio de Identificadores Léxicos. Por ende, aplicando la metodología los unimos en un solo aspecto interceptando todas las funciones que individualmente cruzaban. A esta unión se le llamará Asp_ImpresionDetalles.

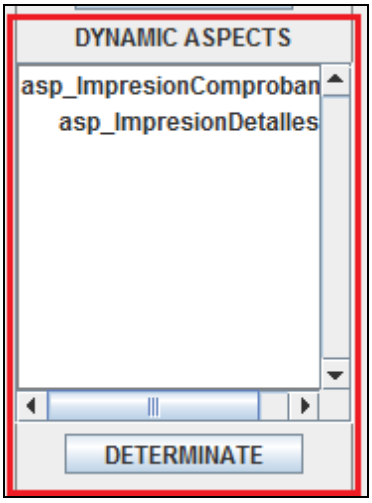


Imagen 24. Aspectos Dinámicos

5.2.7. Aplicación de Metadatos

Para guardar los objetivos de cada uno de los aspectos, se tiene la tabla de metadatos como se muestra en la Tabla 18. La relevancia está dada por la relación entre la influencia mayor y cada influencia.

ID	Tipo	Funciones Relacionadas	Relevancia
Asp_IdentificarCliente	Aspecto	#1. Identificar Cliente	7
Asp_BúsquedaCliente	Aspecto	#2. Realizar Búsqueda de Cliente	0
Asp_DesplegarInfo_Adv	Aspecto	#9 Mostrar información avanzada de la cuenta	4
Asp_ImpresionDetalles	Aspecto	#7 Impresión de Comprobantes #13 Impresión de Detalles de la cuenta	4
Asp_Reiniciar	Aspecto	Reiniciar / Finalizar	3

Tabla 18. Datos correspondientes a los metadatos

5.3. Etapa 3: Especificación de Aspectos

5.3.1. Control de Rango de los Aspectos y Cortes Transversales

Dentro de esta sección se tiene la Tabla que indica la encapsulación de los aspectos.

ID	Corta a	Condición	Depende de	Corte
Aspecto Identificar Cliente	Identificar Cliente	N/A	Aspecto Búsqueda Cliente	Reconocimiento de Usuario
Aspecto Búsqueda Cliente	Realizar Búsqueda de Cliente	Se ejecuta siempre antes del Aspecto Identificar Cliente	N/A	Reconocimiento de Usuario
Aspecto Desplegar Información Avanzada	Mostrar Información Avanzada de la Cuenta	N/A	N/A	Acciones Completas del ATM
Aspecto Impresión de Detalles	Impresión de Comprobantes Impresión de Detalles de la cuenta	N/A	N/A	Global
Aspecto Reiniciar	Reiniciar / Finalizar	Finaliza el proceso	N/A	Global

Tabla 19. Control del Rango de los Aspectos

Los cortes asignados como “Global” son siempre ejecutados al finalizar cualquier transacción, debido a esta particularidad, se creó un Corte denominado sistema para él.

5.3.2. Determinación y Ubicación de los Elementos Propios de los Aspectos:

Los aspectos se especifican de la siguiente manera:

- **Aspecto a Identificar Cliente:**

Cross-Cutting: Sistema ATM – Reconocimiento de Usuario.
Cut-Point: Se ubica interceptando a la funcionalidad "Identificar Cliente".
***Join-Point*:** es el punto de unión de todos los elementos, hace referencia a la clase que se intercepta. Se vincula con el punto de corte del aspecto.
***Advice*:** Se vincula con el punto de corte del aspecto, debido a la presencia del gran número de influencias, se crea un punto de corte con tiempo after ().
Justificación: Debido a su influencia en otras funciones, cambios en los mismos pueden ser controlados después de la ejecución de los mismos, sin modificar de esta manera la clase o el método relacionado.

- **Aspecto a Búsqueda de Cliente:**

Cross-Cutting: Sistema ATM – Reconocimiento de Usuario.
Cut-Point: Se ubica interceptando a la funcionalidad "Realizar Búsqueda de Cliente".
***Join-Point*:** es el punto de unión de todos los elementos, hace referencia a la clase que se intercepta. Se vincula con el punto de corte del aspecto.
***Advice*:** Se vincula con el punto de corte del aspecto, debido a su relación con la función "Identificar Cliente" por un <incluye> se tiene que su tiempo es around ().
Justificación: Debido a la ejecución obligatoria de la función asociada, se puede aplicar cambios necesarios al momento que se ejecuta. Queda a consideración del analista y programador, agregar advice antes y después si ameritan.

- **Aspecto a Desplegar Información Avanzada:**

Cross-Cutting: Sistema ATM – Acciones Completas.
Cut-Point: Se ubica interceptando a la funcionalidad "Mostrar información avanzada de la Cuenta".
***Join-Point*:** es el punto de unión de todos los elementos, hace referencia a la clase que se intercepta. Se vincula con el punto de corte del aspecto.
***Advice*:** Se vincula con el punto de corte del aspecto, debido a la gran cantidad de influencias se tiene que su tiempo es after ().

Justificación: Debido a su influencia en otras funciones, cambios en los mismos pueden ser controlados después de la ejecución de los mismos, sin modificar de esta manera la clase o el método relacionado. Tomando en consideración la anteposición de un aspecto, queda a consideración el agregar un advice before () relacionado con el punto de corte del aspecto.

- **Aspecto a Impresión de Detalles**

Cross-Cutting: Sistema ATM – Global

Cut-Point: Existen dos puntos de corte, uno de ellos se ubica interceptando a la funcionalidad "Impresión de Comprobantes" y el otro de ellos a "Impresión de Detalles de la Cuenta".

***Join-Point*:** es el punto de unión de todos los elementos, hace referencia a las clases que se intercepta, en caso de ser dos clases diferentes, existirán dos joinpoint, de lo contrario sólo habrá uno de ellos.

***Advice*:** Existen definidos dos advice, el primero está relacionado al primer punto de corte y su ejecución es before () debido a la cantidad de dependencias. El segundo advice se relaciona al según punto de corte y su ejecución es before ().

Justificación: Este aspecto intercepta a más de una funcionalidad. Por ende presenta un punto de corte a cada funcionalidad, cada uno de ellos tiene un advice en el cual realizará la operación determinada. El punto de unión juega un papel que une puntos de corte, para este caso de los elementos del mismo aspecto, pero puede hacerlo para diferentes aspectos.

- **Aspecto a Reiniciar:**

Cross-Cutting: Sistema ATM – Global.

Cut-Point: Se ubica interceptando a la funcionalidad "Reiniciar / Finalizar".

***Join-Point*:** es el punto de unión de todos los elementos, hace referencia a la clase que se intercepta. Se vincula con el punto de corte del aspecto.

***Advice*:** Se vincula con el punto de corte del aspecto, debido a la gran cantidad de influencias se tiene que su tiempo es after ().

Justificación: Debido a que este método finaliza y le permite al sistema empezar nuevamente el servicio. El aspecto interviene antes de su ejecución con un advice y el punto de corte relacionado al mismo.

Si posteriormente, el sistema se amplía el diseño, la aparición de los nuevos aspectos pueden utilizar los puntos de unión que cualquiera de los aspectos presentes.

Dado que los mayores beneficios de los aspectos se encuentran en la etapa de mantenimiento, depuración y encapsulación de funcionalidades, algunos de estos elementos pueden no tener impacto en el sistema, pero serán muy útiles cuando se realice algún cambio futuro y se necesite adaptar el sistema.

En la Imagen 25 se aprecia una relación creada por la herramienta de apoyo, la cual adiciona los aspectos y las clases al sistema con todos los elementos expuestos anteriormente. Adicionalmente, dentro del análisis del sistema se crea la Imagen 26 que muestra el diagrama de casos de uso con los aspectos identificados, mediante la notación propuesta.

5.3.3. Control de Redundancia

Verificando el alcance y la similitud de los aspectos identificados en la Tabla 19. Control del Rango de los Aspectos, y según las influencias y dependencias de las funciones interceptada. Se observa que no existen aspectos que intercepten las mismas funciones, de tal manera que en este problema no existe redundancia.
La redundancia se presenta generalmente en proyectos macro, donde existen un gran número de funcionalidades y el análisis en sí se hace complejo.

La herramienta, verificando el alcance de los aspectos, de las dependencias y clases que interceptan determina la existencia de redundancia, para este proceso no se determinó alguna, debido a la carencia de la misma.

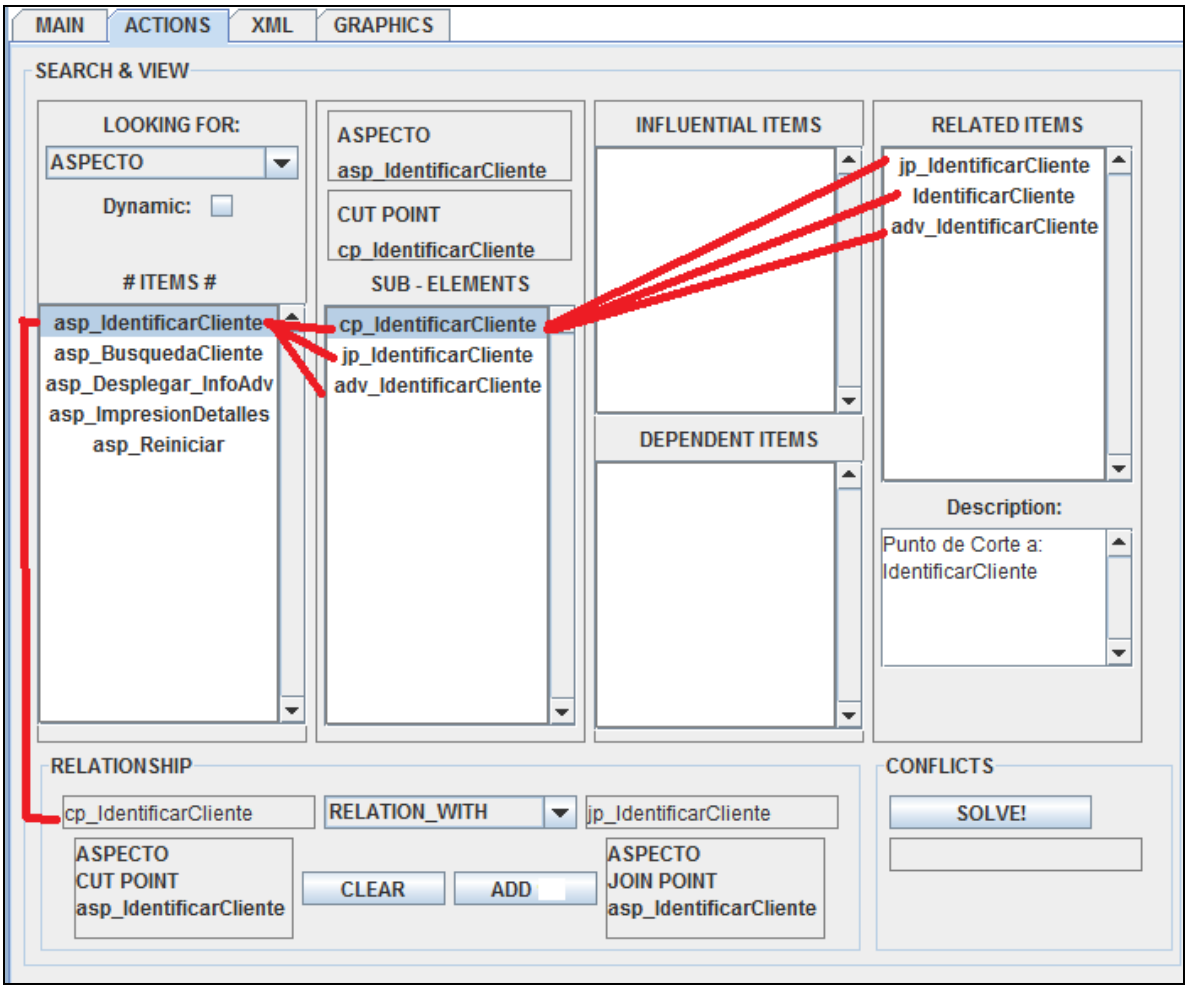


Imagen 25. Relaciones de los aspectos con sus elementos y los métodos que interceptan.

Basado en AOCRE, las metodología comparten el principio de especialización (segmentar un elemento en elementos más pequeños) de aspectos; se diferencian en que AOCRE realiza el proceso enfocado al refinado de requerimientos, mientras que esta metodología aplica el proceso para la segmentación de aspectos, en análisis de estos sub elementos y la reducción de duplicidad. Se asemejan en que los aspectos y clases asociadas se integran en componentes. Para esta tarea se debe definir si se encuentran asociados bajo el mismo corte transversal y si los elementos presentan una fuerte relación al momento de ejecutarse las rutinas del framework.

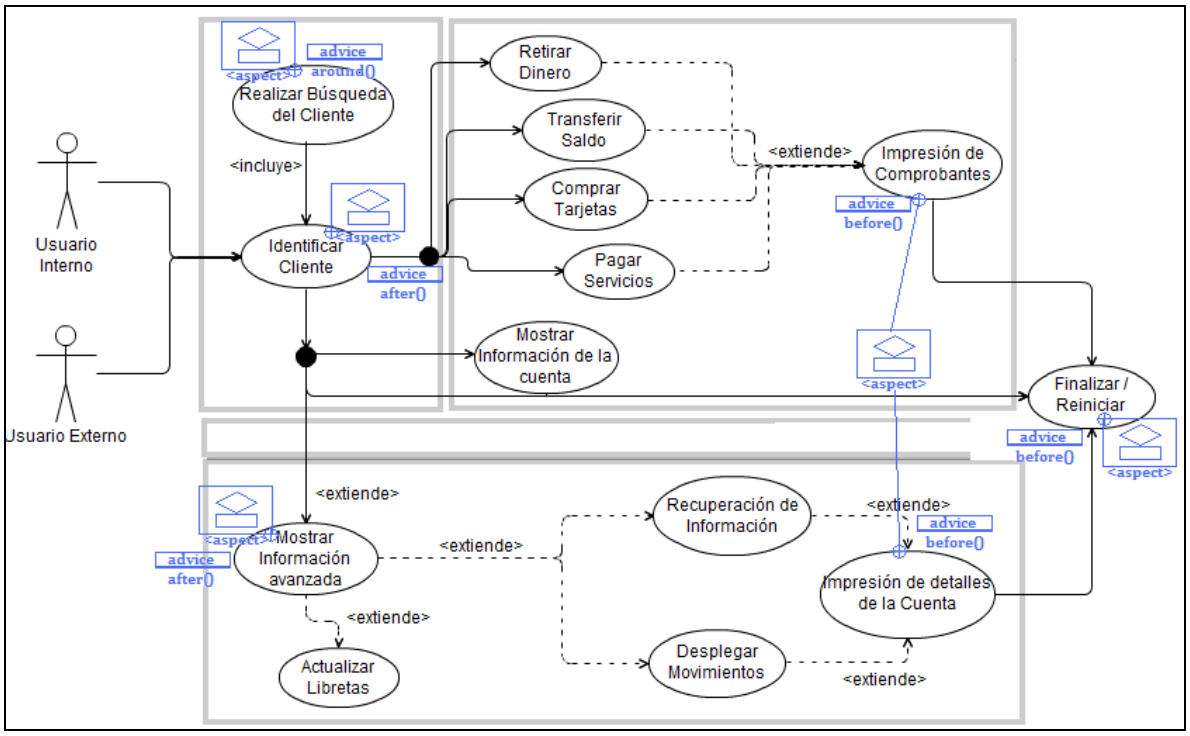


Imagen 26. Casos de Uso con los aspectos incorporados en el mismo.

5.3.4. Compactación de Componentes

Para el proceso de compactación, se agrupan todos los elementos encontrados dentro de sus respectivos paquetes.
El resultado es el siguiente:

	Metodología	***Herramienta***
	Componente1	***Paquete1***
Corte	*Reconocimiento de Usuario*	*RUsuario*
Aspectos	>Aspecto a Identificar Cliente >Aspecto a Realizar Búsqueda de Cliente	>asp_IdentificarCliente.java >asp_BusquedaCliente.java
Clases		>Cliente
	Componente2	***Paquete2***
Corte	*Movimientos Básicos*	*MovBas*

Aspectos	N/A	N/A
Clases		>OperacionesDinero
	Componente3	***Paquete3***
Corte	*Movimientos Completos*	*MovAdv*
Aspectos	>Aspecto a Desplegar Inf. Avanzada	>asp_DesplegarAvanzada
Clases		>GestionCuenta
	Componente4	***Paquete4***
Corte	*Global*	*myaspects / myclasses*
Aspectos	>Aspecto a Reiniciar >Aspecto a Impresión de Detalles	>asp_reiniciar >asp_ImpresionDetalles
Clases		>Impresiones

Tabla 20. Estructura de paquetes para los aspectos

En la generación de la herramienta de apoyo, se incorporó las posibles clases, de tal manera de obtener un framework más completo, donde existen en los paquetes clases, aspectos y las respectivas funciones. Todos los procesos realizados en esta metodología se pueden aplicar con las clases identificadas; este proceso se obvio debido a que no es significativamente importante.

5.3.5. Especificación de Metadatos

Se rellena con toda la información existente en una estructura XML, dando como resultado el almacenamiento descriptivo de los elementos.

La herramienta de apoyo permite añadirle descripciones a cada elemento, de tal forma de guarecer la información referente al elemento, ya sea descripción, importancia o condiciones especiales. Estos elementos son guardados en la estructura XML del proyecto.

En la Imagen 27 se tiene la estructura XML de uno de los aspectos generados.

```
<aspecto>
  <name>asp_IdentificarCliente</name>
  <description>TODO</description>
  <impact>0</impact>
  <dynamic>false</dynamic>
  <corte />
  <itsmethods />
  <relationships />
  <subelements>
    <CutPoint>
      <name>cp_IdentificarCliente</name>
      <description>Punto de Corte a: IdentificarCliente</description>
      <relationships>
        <JoinPoint>asp_IdentificarCliente#jp_IdentificarCliente</JoinPoint>
        <Advice>asp_IdentificarCliente#adv_IdentificarCliente</Advice>
        <function>Cliente#IdentificarCliente</function>
      </relationships>
    </CutPoint>
    <JoinPoint>
      <name>jp_IdentificarCliente</name>
      <description>TODO</description>
      <relationships>
        <CutPoint>asp_IdentificarCliente#cp_IdentificarCliente</CutPoint>
      </relationships>
    </JoinPoint>
    <Advice>
      <name>adv_IdentificarCliente</name>
      <description>TODO</description>
      <time>3</time>
      <relationships>
        <CutPoint>asp_IdentificarCliente#cp_IdentificarCliente</CutPoint>
      </relationships>
    </Advice>
  </subelements>
</aspecto>
```

Imagen 27. Estructura XML para uno de los aspectos existentes en el caso de estudio.

5.4. Etapa 4: Catalogación y Conflicto de Aspectos:

5.4.1. Catalogación de Aspectos

Los aspectos fueron identificados a partir de ambos mecanismos para la selección: Identificadores Léxicos y Diagrama de casos de uso. Por ende esta etapa no se aplica.

5.4.2. Ponderación de Aspectos en Conflictos

El proceso de análisis ha terminado. La Ponderación de Aspectos en Conflictos es un procedimiento que se aplica dentro del flujo del proceso para determinar relevancia de un aspecto sobre otro.
Supongamos que se desea conocer si “Aspecto a Identificar Cliente” o “Imprimir de Detalles” (para demostración de la metodología, debido a que los mismos no presentan conflictos).

Suponiendo que no tenemos influencias de usuarios o stakeholders (0 para cada valoración) y evitando colocar elementos en común para un cálculo más rápido, tenemos:

Elemento a Valorar	**Asp_IdentificarCliente**	**Asp_ImprimirDetalles**
Usuario / Stakeholders	*F: IdentificarCliente: 0* *F: RetirarDinero: 0* *F: TransferirSaldo: 0* *F: ComprarTarjetas: 0* *F: PagarServicios: 0* *F: MostrarInformación: 0* *F: Desplegar_InfoAdv: 0* *C: Cliente : 0* *C: OperacionesDinero : 0*	*F: ImpresionDetalles : 0* *F: ImpresionComprobantes:* *C: Impresión : 0*
Funcionalidad Asociada a las que intercepta	*7x = 7*	*2x = 2*
Clase Asociada a las que intercepta	*2x = 6* *1x 1x = 2*	*1x = 3* *2x = 2*
TOTAL	*15*	*7*

Tabla 21. Ponderación de aspectos en conflictos.

En la Tabla 21 se priorizan los aspectos. Las abreviaturas F, C e I significan función, clase e influencias asociadas, respectivamente, cada 0 representa el valor dado por el usuario.

Finalmente, entre los dos aspectos seleccionados, el aspecto que interviene en Identificar Cliente es el que posee más prioridad en este esquema.

5.5. Análisis de Resultados

En el objetivo de incorporar el tratamiento de aspectos en la etapa de diseño en la ingeniería de requerimientos, las metodologías AOCRE, AORE, SLAI, ViewPoint y MEDFOAR utilizan mecanismos diferentes, pero sustancialmente se basan en los mismos fundamentos para lograr este propósito.

Todas las metodologías empiezan traduciendo los requerimientos funcionales y no funcionales a diagramas, modelos, tablas de especificación de elementos que permiten separar las funcionalidades del sistema.

MEDFOAR combina los siguientes elementos: diagrama de casos de uso y escenarios funcionales (SLAI, AORE) con la clasificación por Vistas según usuarios y funciones (ViewPoint); este procedimiento puede considerarse esencial como primera etapa además de aplicarse en cualquier otro paradigma como el de orientación a objeto. La combinación de enfoques: vistas, caso de uso y objetivos plantean las bases para la metodología propuesta diferenciándose de las demás cuya base es uno o dos enfoques y la manera como trata los aspectos transversales.

Comparando los resultados con el caso de estudio de SLAI, MEDFOAR presenta una especificación detallada a nivel en el diagrama de casos de uso y la especificación de aspectos por influencias y dependencias; además de contener procedimientos de enfoques de puntos de vistas y contabilización de identificadores.

Además, MEDFOAR permite la agrupación de funciones y clases que se interceptan bajo un aspecto como se ve en este caso de estudio para los módulos de

impresiones, en donde existen puntos de corte que atrapan estas funciones. Esta característica también se encuentra en la herramienta de apoyo, y es una de las fundamentales en el tratamiento de aspectos debido a que está es una de las principales características de los aspectos.

La metodología propuesta permite el tratamiento de aspectos desde el punto de vista de identificación, ubicación y especificación de aspectos, determinando: el nombre, los puntos de cortes, los puntos de unión, los advices y las clases que intervienen, sin profundizar en el contenido y métodos detallados del aspecto.

Durante los procedimientos de abstracción de funcionalidades y de separación de asuntos, MEDFOAR se diferencia debido a que una vez más integra las demás metodologías y extrae los aspectos candidatos de los diagramas de casos de uso por la regla de descomposición (AORE) y son expandidos por la regla de identificadores determinantes (SLAI); además no utiliza el procedimiento de extracción de aspectos por las vistas (ViewPoint) sino que las mismas son utilizadas para especificar las funcionalidades.

Una vez validados los aspectos, MEDFOAR los representa dentro de los diagramas de casos de uso mediante su propia normativa, identificando los puntos de corte, putos de unión y advice, para su facilitar el proceso en la etapa de implementación. Al igual que las demás metodologías, se especifica los aspectos de una forma similar a SLAI, identificando las condiciones e influencias.

Es importante destacar que la particularidad más notable de MEDFOAR es la utilización de funcionalidades y métodos que representan el flujo de funcionalidades, en lugar de las funcionalidades macro o asuntos macros, en la determinación de aspectos candidatos y del tratamiento de los aspectos.

Además de enfocarse en determinar las intervenciones realizadas de los puntos de cortes, puntos de unión de los aspectos y en la ubicación del diseño lógico en lugar de profundizar mucho en el contenido de los mismos, debido a que esta metodología está diseñada para la etapa de requerimientos.

Además, adopta la idea de AOCRE de fragmentar los aspectos en otros más pequeños, bajo la idea de especialización, junto con la compactación de todos los elementos en componentes. Para el tratamiento de los aspectos no funcionales se adoptan los registrados en la metodología.

En la última etapa, está la resolución de conflictos, en donde MEDFOAR, AORE y ViewPoint utilizan el mismo enfoque de valoración por usuarios, pero con la diferencia de que la metodología propuesta incluye otros aspectos de ponderación que es la cantidad de dependencias y si una funcionalidad antecede la otra con la que presenta conflicto, es decir, toma en cuenta la cantidad de funcionalidades, clases y las intervenciones que tiene un aspecto.

Como principal ventaja de MEDFOAR tenemos la extracción de las principales ventajas y beneficios de las otras metodologías; los diversos mecanismos para la identificación de aspectos y la ubicación para el fácil reconocimiento en los casos de uso. Entre sus desventajas tenemos que durante el procedimiento de análisis es necesario diseñar los diversos enfoques: casos de uso, vistas, especificación de funciones para realizar el proceso completo.

En esta fase de aplicación de la metodología MEDFOAR y de su herramienta HSTAR, se ha podido obtener los siguientes resultados:

- Se han determinado los aspectos estáticos y dinámicos del sistema.
- El procedimiento realizado se basa en actividades necesarias en el análisis de la POO.
- Se conocen los elementos de los aspectos identificados.
- Se conoce la funcionalidad de cada uno de los elementos identificados.
- Se conoce la ubicación dentro de los diagramas de casos de uso de los elementos de los aspectos y de los propios aspectos.
- Se tiene una vista lógica del sistema.
- Se presenta a través de la herramienta el flujo de actividades de las funcionalidades involucradas y las intercepciones de los aspectos.
- Se presenta de manera estructurada
- Se ha generado una estructura de componentes, donde se incluyen los aspectos.

- Se controla la redundancia de los aspectos a través de las influencias y dependencias de las funcionalidades interceptadas por los aspectos.
- Es posible priorizar aspectos sobre otros en caso de conflictos.
- Se ha generado un proyecto orientado a aspectos con el código base diseñado a través de la herramienta.

Como se muestra al final del capítulo V, la aplicación de la metodología minimiza los inconvenientes expuestos en el marco referencial, por ende es una solución sistemática para el tratamiento de aspectos en la Etapa de Análisis Requerimientos de la Ingeniería de Software. Además, la herramienta de apoyo proporciona mecanismos automatizados de algunas etapas de la metodología, lo cual agiliza los procedimientos de identificación y tratamiento de aspectos.

Conclusiones y Trabajos Futuros

La incorporación del paradigma de programación orientada a aspectos dentro de las etapas de diseño de la Ingeniería de Requerimientos permite agregar al modelo una estructura más sólida debido a que se podrá contar con todos los beneficios de la misma desde fases tempranas.

Las normas establecidas y la metodología conformada por la reunión de características de otros modelos ha resultado un método alternativo para el tratamiento de aspectos, consta de procesos como Identificación de aspectos en casos de uso, segmentación de vistas, resolución de conflictos, determinación de puntos de corte, puntos de unión, advice, entre otros elementos.

Los aspectos en MEDFOAR son extraídos a partir de casos de uso (que a su vez son elaborados a partir de requerimientos funcionales del sistema), de requerimientos no funcionales y de los objetivos del sistema. En esta normativa es de suma importancia las influencias y dependencias de cada funcionalidad en particular y cuenta con técnicas alternativas según la complejidad del problema, como lo son la base Léxica de identificadores y la regla de descomposición.

La metodología en conjunto con la herramienta de apoyo, proporcionan las siguientes ventajas:

- El tratamiento de aspectos se realiza dentro de los procedimientos habituales de la POO, como lo es la identificación de casos de uso, diagramas de vistas, establecimiento de influencias y dependencias, encapsulación en componentes.
- Permite la identificación de los aspectos y sus elementos que impactan el sistema, además de brindar características de los mismos (ubicación, tiempo de ejecución, importancia, metadatos y otros) y de mantener los mismos documentados.

- Algunos de los procedimientos pueden ser automatizado lo cual facilita ciertas tareas en el diseño del framework.
- Mediante el procedimiento estructurado, que ataca los inconvenientes de la POA, se aprovechan al máximo las ventajas del paradigma desde etapas tempranas del software.
- Permite determinar cuando un aspecto intercepta varias funcionalidades y cruza varias clases, tratando conflictos de multifuncionalidades.
- Se pueden generar case del framework de proyectos diseñados mediante la herramienta, lo cual agiliza procedimientos en las etapas posteriores del Desarrollo de Software.
- La herramienta permite la búsqueda y comparación de elementos dentro de los aspectos de tal manera que es posible minimizar el impacto de posibles redundancias cuando se ingresan los aspectos de manera manual.

Finalmente, mediante la demostración de la metodología y la herramienta en el caso de aplicación, se puede determinar MEDFOAR como una solución a la carencia de metodologías sistemáticas y estructuradas que minimicen las desventajas de la POA en base a los inconvenientes de dificultad nata de los aspectos y que a su vez, permita el aprovechamiento de las ventajas de la POA desde etapas tempranas del proceso de desarrollo de software.

Como el paradigma de orientación a aspectos crece y se robustece cada día, de la misma manera se presentan trabajos futuros que le dan continuidad a los trabajos propuestos.

Para la Metodología:

- Extender las fases de la metodología para que involucren el tratamiento de aspectos en etapas posteriores a la Ingeniería de Requerimientos.
- Darle mantenimiento a la metodología, debido a que AspectJ se actualiza frecuentemente y la misma está basada según la arquitectura de la misma.
- Poner a prueba la metodología con muchos más casos de aplicación, de tal manera que se pueda complementar las reglas frente a casos y requerimientos especiales.
- Estudiar las posibilidades de incluir la determinación de cortes primitivos en etapas previas a la implementación del framework

Herramienta de Soporte:

- Aplicar mecanismos de lectura de clases en código, de tal forma que se pueda usar el código de la clase para identificar sus métodos, propiedades relevantes, revisión de redundancia, en la generación del case, entre otras funcionalidades.
- Buscar nuevos mecanismos para clasificar cuando usar las diferentes llamadas que tienen los elementos de los aspectos, en especial los JoinPoint y los CutPoint.
- Mantener la herramienta actualizada debido a que AspectJ se optimiza frecuentemente y adquiere propiedades y métodos más eficientes.
- Incorporar en la autogeneración diferentes llamadas a los cortes primitivos dependiendo de la más indicada frente a un escenario determinado.

Caso de Estudio:

- Aplicar la metodología en casos de uso con enfoques variados, de diferentes complejidades que presenten escenarios y ambientes especiales.
- Después de la extensión de la metodología a otras etapas de desarrollo de software, se pretenden realizar casos de uso completos desde la etapa de análisis de requerimientos hasta mantenimiento y puesta en marcha.

Referencias Bibliográficas

Aoyama, M. and A. Yoshino (2008). AORE (Aspect-Oriented Requirements Engineering) Methodology for Automotive Software Product Lines. Software Engineering Conference, 2008. APSEC '08. 15th Asia-Pacific. Beijing: 203 - 210.

Budwell, C. C. and F. J. Mitropoulos (2008). The SLAI Methodology: An Aspect-Oriented Requirement Identification Process. 2008 International Conference on Computer Science and Software Engineering, Wuhan, Hubei.

Cai, H., Y. Zhang, et al. (2009). Aspect-oriented Requirement and Reuse Aspect. Computational Intelligence and Natural Computing, 2009. CINC '09. Wuhan. 2: 475 - 477

Cazzola, W., J. Jézéquel, et al. (2006). Semantic Join Point models: Motivations, Notions and Requirements. In SPLAT 2006 (Software Engineering Properties of Languages and Aspect Technologies).

Dahiya, D. and R. J. Sachdeva (2006). Understanding Requirements: Aspect Oriented Software Development. Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06), Chicago, IL.

Debnath, N., L. Baigorria, et al. (2008). Metrics Applied to Aspect Oriented Design Using UML Profiles. IEEE Symposium on Computers and Communications, ISCC 2008., Marrakech

Eclipse. (2012). "AspectJ crosscutting objects for better modularity." 2012, from http://www.eclipse.org/aspectj/.

Fernando Asteasuain and B. Contreras (2002). Programación Orientada a Aspectos. Análisis del Paradigma. Ciencias y Computación. Argentina, Universidad Nacional del Sur. Licenciatura.

Fletcher, D. P., F. Akkawi, et al. (2004). "From Research to Operations: Integrating Components with an Aspect-Oriented Framework and Ontology." IEEE Aerospace Conference Proceedings: 15.

G. Cugola, C. Ghezzi, et al. (1999). Language Support for Evolvable Software: An Initial Assessment of Aspect-Oriented Programming. International Workshop on the Principles of Software Evolution, IWPSE99.

Grundy, J. (1999). Aspect-oriented requirements engineering for component-based software systems. IEEE International Symposium on Requirements Engineering, 1999., Limerick.

Guo, Y., G. Teng, et al. (2007). Improvement of Object-Oriented System Analysis and Design with Aspects. 31st Annual International Computer Software and Applications Conference, 2007. (COMPSAC 2007), Beijing

Guzmán, G. (2009). "La Orientación a Aspectos en la Ingeniería de Requisitos." Retrieved August 15th, 2010, from http://www.usmp.edu.pe/publicaciones/boletin/fia/info68/orientacion.pdf.

Hu, H., C. He, et al. (2009). "An AOP Framework and Its Implementation Based on Conceptual Model." ISECS International Colloquium on Computing, Communication, Control and Management: 4.

Hwang, H. J. and J. T. Choi (2007). "Design of an Aspect-Based Framework to Improve the Dynamic Management of MFID middleware." 6.

Jingjun, Z., L. Furong, et al. (2007). Aspect-Oriented Requirements Modeling. 31st IEEE Software Engineering Workshop, 2007. SEW 2007. Columbia, MD 35 - 40.

Jun-Wei, G., T. Rong, et al. (2008). A MDA based Aspect-Oriented Model Dynamic Weaving Framework. International Conference on Computer Science and Software Engineering. Wuhan, Hubei 2: 90 - 93.

Kim, T. and H. Lee (2008). Establishment of a security system using Aspect Oriented Programming. International Conference onControl, Automation and Systems 2008 (ICCAS 2008), Seoul

Kumar, A., R. Kumar, et al. (2008). Toward a Unified Framework for Cohesion Measurement in Aspect-Oriented Systems. 19th Australian Conference on Software Enginnering.

Londoño, L., R. Anaya, et al. (2008). Análisis de la Ingeniería de Requisitos Orientada por Aspectos según la Industria del Software. Revista EIA. Medellín, Colombia: 43-52.

López, M., F. Asteusuain, et al. (2005) "Programemos en AspectJ."

Majumdar, D. and S. Bhattacharya (2009). A Design Model Based Exceution Framework for Aspect Oriented Systems. International Conference on Methods and Models in Computer Science.

Markiewicz, M. E. and C. J. P. d. Lucena "El Desarrollo del Framework Orientado al Objeto."

Morocho, J., J. Chuico, et al. Paradigma de Programación Orientada a Aspectos. Argentina, Universidad Técnica Particular de Loja.

Nusayr, A. (2008). AOP as Formal Framewok for Runtime Monitoring. FSE-16 Doctoral Symposium, Atlanta, Georigia, USA.

Perez-Toledano, M. and C. Canal (2007). TITAN: a Framework for Aspect Oriented System Evolution. International Conference on Software Engineering Advances, Cap Esterel.

Rashid, A. (2008). Aspect-Oriented Requirements Engineering: An Introduction. 16th IEEE International Requirements Engineering, 2008. RE '08. Catalunya 306 - 309.

Rashid, A., P. Sawyer, et al. (2002). Early aspects: a model for aspect-oriented requirements engineering. Proceedings. IEEE Joint International Conference on Requirements Engineering, 2002: 199 - 202.

Shah, V. and F. Hill (2003). An Aspect-Oriented Security Framework. DARPA Information Survivability Conference and Exposition (DISCEX'03).

Tabares, M., R. Anaya, et al. (2007). "Aspect Oriented Software Engineering: An Experience of Application in Help Desk Systems." Dyna rev.fac.nac.minas 147(153).

Vanhaute, B., B. D. Win, et al. (2001). "Building Frameworks in AspectJ." 6.

Yu-Ning, P. and L. Qiang (2009). A Viewpoint-Oriented Requirements Elicitation Integrated with Aspects. World Congress on Computer Science and Information Engineering, 2009 WRI. Los Angeles, CA 7: 706 - 711.

Zhang, Y. and J. Zhang (2005). "Interactive Framework and its Supported Environment between Component and Aspect." 12.

Anexos

A. MEDFOAR y Panamá

A.1. Antecedentes

La Programación orientada a Aspectos (POA) es un paradigma moderno de programación, consiste en una técnica avanzada de programación cuyo propósito es la segmentación y agrupamiento de funciones que debe desempeñar un software en bloques transversales denominados aspectos. Esta técnica proporciona ventajas sobre otros paradigmas, debido a la estructura de la misma, entre ellas tenemos las siguientes: (1) La reducción de desarrollo, (2) Permite un mantenimiento del software más eficiente, (3) Reduce los costes de esfuerzo de desarrollo en todas las etapas del ciclo de vida.

Las ventajas de la POA influyen directamente en el proceso de desarrollo de software que tiene como prioridad cumplir con un nivel de calidad sin perjudicar el costo o el tiempo del mismo, incluyendo tareas de análisis, diseño e implementación del mismo. Como la ingeniería de software es una disciplina cuya función es tratar de garantizar un producto de calidad que cumpla con los estándares exigidos mediante un procedimiento estructurado y cíclico de continua mejora de software tanto en diseño, implementación y mantenimiento del mismo, esta resulta también beneficiada al orientado el desarrollo de un proyecto a la POA.

La especificación correcta de funcionalidades y la descripción de los componentes del software en la Ingeniería de Requerimientos permitirá crear una base robusta para las etapas posteriores reduciendo la complejidad de las mismas.

La incorporación de la Programación Orientada a Aspectos, uno de los actuales paradigmas de programación cuya función es la encapsulación de funciones es elementos denominados aspectos en beneficio de procesos de mantenimiento, depuración y reducción del acoplamiento, en la ingeniería de requerimiento, permitirá incorporar desde etapas tempranas este paradigma y sus beneficios al diseño y productos de la fase, impactando positivamente el desarrollo de proyectos informáticos en Panamá y en el mundo.

La disciplina de Desarrollo de Software Orientado a Aspectos (AOSD) fue creada para hacer frente a la dificultad del código transversal debido a su complejidad y su dificultad para entenderse, en otras palabras, AOSD fue ideada para minimizar la complejidad de las tareas de la programación orientada a aspectos, en especial la identificación de las funciones utilizadas varias veces dentro de un software. La disciplina AOSD se enfoca en la identificación, separación, representación y la composición de los asuntos (objetivos del programa) de carácter transversal (Budwell and Mitropoulos 2008). Estos asuntos son capturados en módulos independientes denominados aspectos.

Los aspectos son estas funciones transversales (invocadas en diferentes partes del software) que interceden y pueden modificar el comportamiento de una aplicación. Los mismos están clasificados, de tal forma que cuando se requiera realizar alguna operación o modificar la funcionalidad de un software, se modifica el aspecto sin la necesidad de buscar a través del código las funciones en las cuales se necesitan los cambios.

La incorporación de aspectos ahorra tiempo de diseño (donde se idea como será el software que se creará), depuración (el proceso de corrección de errores), implementación (el proceso en sí de desarrollo y de programación) y mantenimiento (consiste en mantener el software actualizado) del software, lo que significa que mediante la implementación de la POA los costos y tiempo del proyecto de software se reducirán y se incrementará la calidad, ya que se contará con un software más estructurado y de fácil mantenimiento.

Debido a que el paradigma de POA es reciente, el mismo está en continuo mejoramiento y presenta algunas desventajas entre las cuales están la complejidad

del proceso de identificación de las funciones transversales, la ubicación de sus elementos puede tergiversar en algunas situaciones una funcionalidad provocando que el software desempeñe una acción incorrecta, algunas decisiones de diseño no se pueden tomar hasta la implementación debido a la carencia de pasos estructurados que indiquen como hacerlo, entre otros inconvenientes.

A.2. Beneficios y principales beneficiarios

La aplicación de la metodología y la herramienta ayudará en:
(1) Reducción de Tiempo de implementación
(2) Minimización de conflictos con funciones y aspectos en implementación.
(3) Reducción al tiempo de depuración y corrección de errores.
(4) Facilidad en las tareas de mantenimiento de programas.
(5) Software más estructurados y confiables.
(6) Aumento de la calidad del software.

Los beneficiarios del proyecto son directamente las empresas desarrolladoras de software que utilicen la metodología e indirectamente las que utilicen software soportados por la metodología.

Las reglas de la metodología están estructuradas de manera que minimicen inconvenientes y faciliten el análisis del software, debido a que compacta las tareas cotidianas de la programación orientada a objetos con actividades específicas de la programación orientada a aspectos, integrando las actividades y proveyendo una metodología sólida y de fácil aplicación.

A.3. Impacto esperado

El impacto que tendría el proyecto sería dentro del sector productivo, en la disciplina de Desarrollo de Software por cualquier institución que aplique la metodología. A continuación se detallarán las principales ventajas:

- Reducción de Tiempo de desarrollo de software. Cuando se utilice la metodología se reducirá el tiempo promedio que llevarán todas las etapas posteriores al diseño del proyecto informático.
- Reducción de Costo de proyecto de desarrollo de software. Como lógica respuesta a la reducción en el tiempo sin modificar complejidad o uso de herramientas extras, el costo del proyecto se reducirá en base al menor esfuerzo de los programadores y analistas de proyectos.
- Mejora la calidad del software que se desarrolle bajo el paradigma. Debido a que si se utiliza la metodología durante el desarrollo del proyecto, el mismo tendrá las características de la metodología lo cual lo hará más estructurado y de fácil mantenimiento, lo cual incrementa la calidad del software.
- Reduce el esfuerzo que requerirán los diseñadores, analistas y desarrolladores para la creación de proyectos informáticos.
- Las empresas que utilicen la metodología contarán con una metodología que les permitirá diseñar software con un paradigma actual de programación y gozando de todas las ventajas de la misma.

B. Trabajos científicos desarrollados

Se presentan en las próximas páginas