MODELADO Y DESARROLLO DE APLICACIONES ORIENTADAS A LA WEB DESDE UNA PERSPECTIVA DE INGENIERÍA DEL SOFTWARE ORIENTADA A ASPECTOS

LUISA FERNANDA DÍAZ ARENAS
NOHORA MARCELA PINEDA SUÁREZ

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍA DE SISTEMAS
SISTEMAS DE INFORMACIÓN E INGENIERÍA DE SOFTWARE
BUCARAMANGA
2008

MODELADO Y DESARROLLO DE APLICACIONES ORIENTADAS A LA WEB DESDE UNA PERSPECTIVA DE INGENIERÍA DEL SOFTWARE ORIENTADA A ASPECTOS

LUISA FERNANDA DÍAZ ARENAS
NOHORA MARCELA PINEDA SUÁREZ

Trabajo de grado para optar al título de:
Ingeniero de Sistemas

Director
MCC Daniel Arenas Seleey

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍA DE SISTEMAS
SISTEMAS DE INFORMACIÓN E INGENIERÍA DE SOFTWARE
BUCARAMANGA
2008

Nota de Aceptación

_____

_____

_____

_____

Firma del Jurado

_____

Firma del Jurado

_____

Firma del Director

Bucaramanga, 27 de Mayo de 2008

DEDICATORIA

A ti Dios que me diste la oportunidad de vivir y regalarme una familia maravillosa, con cariño a mis padres que me dieron la vida, a mis hermanos que han estado en todo momento conmigo, a mi novio por todo el apoyo y comprensión y a mis primas Gutiérrez Arenas por brindarme la oportunidad de seguir adelante, crecer como persona y prepararme profesionalmente.

A todos ellos doy infinitas gracias

LUISA FERNANDA

Principalmente a Dios, por sus enseñanzas a través de las personas que me ha puesto en mi camino durante mi vida y por regalarme una encantadora familia.

A mis padres, Jairo Pineda y Nohora Suárez; por su amor, comprensión y apoyo haciendo posible el logro de mis sueños.

A mi hermano Andrés por su apoyo y comprensión.

NOHORA MARCELA

AGRADECIMIENTOS

Deseamos expresar nuestros más sinceros agradecimientos a:

Nuestro apreciado director de proyecto, Daniel Arenas Seleey**,** por confiar en nosotras y darnos la oportunidad de realizar este proyecto, brindándonos y compartiéndonos sus conocimientos adquiridos en su carrera como profesional, docente e investigador.

Nuestros estimados evaluadores, Karol Reyes y Robby Vega, por cada uno de sus aportes y sugerencias durante el desarrollo del proyecto que permitieron llevar a cabo los objetivos planteados.

A nuestros amigos, en especial a Madeleyne Pérez y Francis Roa, por su colaboración, brindándonos sus conocimientos, su tiempo, su compañía, su paciencia y comprensión.

A nuestros padres y hermanos, por el apoyo, esfuerzo y tolerancia en los momentos más difíciles de nuestras vidas.

A Nicolás Kicillof de la Universidad de Buenos Aires (Argentina), por brindarnos sus conocimientos y compartirnos información sobre la Programación Orientada a Aspectos por medio mail.

A la Universidad Autónoma de Bucaramanga por las facilidades suministradas durante el desarrollo de la carrera y el material de apoyo para la realización de nuestro proyecto.

CONTENIDO

pág.

INTRODUCCIÓN 26

1. PLANTEAMIENTO DEL PROBLEMA Y JUSTIFICACIÓN 28

2. ESTADO DEL ARTE 31

3. MARCO TEÓRICO 37

3.1 ASPECTOS 37

3.2 CROSSCUTTING (INCUMBENCIAS) 39

3.3 PROGRAMACIÓN ORIENTADA A ASPECTOS 42

3.3.1 Punto de Unión (joinpoint) 43

3.3.2 Intersección o Puntos de Corte (pointcut) 44

3.3.3 Consejo o Avisos (advine) 44

3.3.4 Introducciones (introduction) 45

3.3.5 Tejedor (Weaving) 45

3.4 REQUERIMIENTOS PARA DESARROLLO DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS POA 46

3.5 OBJETIVOS DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS POA 48

3.6 VENTAJAS DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS 49

3.7 DESVENTAJAS DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS 50

3.8 INGENIERÍA DE SOFTWARE Y DESARROLLO ORIENTADO A ASPECTOS 51

3.8.1 Etapa 1: Identificar concerns 52

3.8.2 Etapa 2: Implementar competencias 53

3.8.3 Etapa 3: Componer el sistema final 53

3.9 MÉTRICAS PARA LA PROGRAMACIÓN ORIENTADA A ASPECTOS 56

3.10 TECNOLOGÍAS PARA LA PROGRAMACIÓN ORIENTADA A ASPECTOS POA 58

3.10.1 Avanti 58

3.10.2 Aopmetrics 58

3.10.2.1 Extensiones de métricas para POA 59

3.10.2.2 Implementación actual 59

3.10.3 ARJ 59

3.10.4 THEME/UML 60

3.11 ASPECTOS Y FRAMEWORK 60

3.12 CARACTERÍSTICAS DE APLICACIONES ESCRITORIO Y WEB 62

3.12.1 Web vs Escritorio 62

3.12.2 Pros y contras de aplicaciones de escritorio y Web 63

3.12.3 Bajo mantenimiento y mejoras forzadas 63

3.12.4 Riesgos crecientes de la seguridad 63

3.12.5 Costo 64

3.12.6 Conectividad 64

3.12.7 Mas lento 64

3.12.8 Reservas y propiedades 64

3.13 LENGUAJE UNIFICADO DE MODELADO UML 65

3.13.1 Diagrama de estructura 66

3.13.2 Diagrama de comportamiento 66

3.13.3 Diagrama de interacción 67

3.14 METODOLOGÍA DE DESARROLLO 68

3.14.1 Rup (Rational Unified Process) 68

3.14.2 Metodología Theme / Approach 71

3.14.2.1 Reglas para la separación de candidatos 74

3.14.3 Modelo arquitectónico para el desarrollo orientado a aspectos 74

3.14.4 Identificación temprana de aspectos 76

4. DISEÑO METODOLÓGICO 77

4.1 ETAPA 1: ESTUDIO 77

4.2 ETAPA 2. ELABORACIÓN DE LA METODOLOGÍA 78

4.2.1 Fase 1: Inicio 79

4.2.1.1 Análisis 79

4.2.2 Fase 2: Elaboración 79

4.2.2.1 Análisis 79

4.2.2.2 Diseño 79

4.2.3 Fase 3: Construcción 80

4.2.3.1 Análisis 80

4.2.3.2 Diseño 80

4.2.3.3 Implementación 80

4.3 ETAPA 3: APLICACIÓN 80

4.4 ETAPA 4: RESULTADOS 81

5. METODOLOGÍA PLANTEADA 83

5.1 RUP PARA METODOLOGÍA PLANTEADA 83

5.2 DIAGRAMAS DE LA METODOLOGÍA PLANTEADA 84

5.2.1 Etapa 1 84

5.2.2 Etapa 2 84

5.2.3 Etapa 3 86

5.2.4 Etapa 4 86

5.2.5 Etapa 5 86

5.2.6 Etapa 6 87

5.2.7 Etapa 7 87

5.3 DESARROLLO DE LA METODOLOGÍA PARA EL CASO DE ESTUDIO: SIMULADOR DE UN CAJERO AUTOMÁTICO ATM 87

5.3.1 Etapa 1. Especificar el sistema con diagramas de casos de uso 87

5.3.1.1 Caso de uso, sin aspectos 87

5.3.1.2 Requerimientos funcionales con temas 88

5.3.2 Etapa 2. Modelo de diseño 91

5.3.2.1 Diagrama de secuencia 91

5.3.2.2 Descomponer el sistema en componentes de diseño 91

5.3.2.3 Arquitectura Orientada a Objetos 91

5.3.3 Etapa 3. Identificar concerns 94

5.3.3.1 Analizar requerimientos 94

5.3.4 Etapa 4. Elección de aspectos candidatos 96

5.3.4.1 Identificar crosscutting concerns y concerns funcionales 97

5.3.4.2 Seleccionar aspectos candidatos 96

5.3.5 Etapa 5. Especificar aspectos candidatos 99

5.3.5.1 Describir responsabilidades 100

5.3.5.2 Identificar relaciones entre aspectos candidatos y elementos del modelo 102

5.3.5.3 Arquitectura POA 103

5.3.6 Etapa 6. Identificar conflictos 105

5.3.7 Etapa 7. Modelar UML 106

5.3.8 Etapa 8. Diagramas de secuencia extendido con aspectos 107

5.3.9 Etapa 9. Expresar el Sistema con Aspectos 107

5.3.10 Etapa 10. Generar Prototipo para el Sistema 108

5.3.11 Etapa 11. Ejecutar el Prototipo 108

6. CONCLUSIONES 110

BIBLIOGRAFÍA 114

LISTA DE TABLAS

pág.

Tabla 1. Actividades a realizar en las etapas de inicio, elaboración y construcción 69

Tabla 2. Operaciones con temas y requerimientos 73

Tabla 3. Metodología RUP en la metodología planteada 82

Tabla 3. Relación de los Aspectos y requerimientos 102

LISTA DE FIGURAS

pág.

Figura 1. Estructura de un programa con POA 38

Figura 2. Estructura de un aspecto 38

Figura 3. Ejemplo del comportamiento de los aspectos. 40

Figura 4. Crosscutting 41

Figura 5. Comparación LPG y POA 43

Figura 6. Punto de unión y punto de corte. 44

Figura 7: Lenguajes para la programación orientada a Aspectos 47

Figura 8. Generar ejecutable, enfoque tradicional y poa 52

Figura 9. Estructura de una implementación de lenguajes de aspectos 54

Figura 10. Jerarquía de diagramas UML 67

Figura 11. Diagrama RUP 69

Figura 12. Proceso del Theme/Doc, detallando la vista. 72

Figura 13. Modelo arquitectónico para el diseño orientado a aspectos. 75

Figura 14. Identificación temprana de aspectos 76

Figura 15. Metodología ajustada por las autoras para la programación orientada a aspectos 84

Figura 16. Caso de Uso de la simulación del cajero automático 87

Figura 17. Arquitectura del sistema Orientada a Objetos 91

Figura 18. Arquitectura para el simulador de un cajero automático ATM 92

Figura 19. Theme-relationship view inicial para el sistema ATM. 96

Figura 20. Theme-relationship view con los aspectos del sistema ATM. 100

Figura 21. Evolución del crosscutting-relationship view. 101

Figura 22. Arquitectura POA 103

Figura 23. Arquitectura tres capas para el Simulador del Cajero Automático ATMUNAB. 104

Figura 24. Funcionalidad del Sistema 108

LISTA DE ANEXOS

pág.

Anexo A. Diagrama de secuencias orientado a objetos 119

Anexo B. Diagrama de clases orientado a objetos 127

Anexo C. Diagrama de secuencias orientado a aspectos 142

Anexo D. Base de datos del sistema ATMUNAB 154

Anexo E. Navegación del cajero automático ATMUNAB 162

GLOSARIO

ADVICE: se tratan de pedazos de código asociados a pointcuts, que tratan un nuevo comportamiento en todos los joinpoints representados por el pointcut.

AOPMETRICS: es una herramienta métrica que ayuda a medir la reutilización de código y sirve tanto para la programación OO como programación OA, se desarrolla en apoyo para aspectos a través del ciclo de vida del software.

AOSD: desarrollo de software orientado a aspectos, es una nueva tecnología para la separación de incumbencias.

ARJ: refinamiento paso a paso y desarrollo de software en forma incremental, este método mejora las capacidades de aspectos en la POA, aplicando una tecnología que permite, componer y evolucionar los aspectos en una manera progresiva en las etapas del desarrollo.

ASPECTJ: es un lenguaje de programación orientado por aspectos construido como una extensión del lenguaje Java creado en Xerox PARC. Un compilador de AspectJ hace llegar la noción de aspecto hacia el código de máquina virtual implementando así una noción de relación. Los aspectos en si se escriben en Java extendido generándose un archivo java o compilado con código de máquina compatible con el generado por los compiladores de Java.

ASPECTO: es una unidad modular que se disemina por la estructura de otras unidades funcionales. Se parecen a las clases de Java.

AVANTI: es una herramienta para el análisis y pruebas de programas orientados a aspectos.

CICLO DE VIDA: un modelo de ciclo de vida define el estado de las fases a través de las cuales se mueve un proyecto de desarrollo de software.

CROSSCUTTING: en algunos casos donde dos aspectos del comportamiento de un sistema parecen relacionados entre sí en el código. En tales casos, se dice que, dos aspectos incurren en crosscutting entre ellos.

FRAMEWORK: es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un *framework* puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

JAVA: un lenguaje de programación de alto nivel, orientado a objetos.

JOINPOINT: puntos en el código Java dónde un aspecto puede interceptar a las clases.

LDA: Lenguaje De Descripción Arquitectónica (Metodología)

LOA: lenguajes Orientados A Aspectos

LPG (Lenguaje De Programación General): son lenguajes que pueden ser usados para varios propósitos, acceso a bases de datos, comunicación entre computadoras, comunicación entre dispositivos, captura de datos, cálculos matemáticos, diseño de imágenes o páginas, crear sistemas operativos, manejadores de bases de datos, compiladores, entre muchas otras cosas.

METODOLOGÍA: se encarga de elaborar estrategias de desarrollo de software que promuevan prácticas adoptativas en vez de predictivas; centradas en las personas o los equipos, orientadas hacia la funcionalidad y la entrega, de comunicación intensiva y que requieren implicación directa del cliente.

PARADIGMA DE PROGRAMACIÓN: representa un enfoque particular o filosofía para la construcción del software. No es mejor uno que otro sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro.

POA: Programación Orientada a Aspectos, es un paradigma de programación relativamente reciente cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos. Gracias a la POA se pueden encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos.

POINTCUT: grupos de joinpoints concatenados lógicamente.

POO: programación Orientado a Objetos, es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento.

RELATIONSHIP VIEW: conjunto de requerimientos y temas

REQUERIMIENTOS: características que se desea que posea un sistema o un software.

RUP: es un proceso para el desarrollo de un proyecto de un software que define claramente quien, cómo, cuándo y qué debe hacerse en el proyecto.

TEMAS: palabras que representan una acción, un verbo, sustantivo, las cuales se convertirán en aspectos.

THEME/UML: es una extensión orientada a aspectos para UML, permite encontrar los temas bases y temas de aspectos teniendo un proceso de refinamiento de temas.

UML: es un conjunto de herramientas, que permite modelar (analizar y diseñar) sistemas orientados a objetos.

WEAVING: recomposición de aspectos

RESUMEN

Hoy en día, es necesario conocer sobre la programación orientada a aspectos, porque es un avance en la ingeniería de software, se fundamenta en la correcta modularización y óptima separación de componentes, siendo un nuevo campo que ha abierto la programación orientada a objetos, se puede observar que los progresos más significativos se han obtenido gracias a la descomposición de un sistema complejo en partes que sean más fáciles de manejar. Partiendo de la programación orientada a objetos y basándose en el principio de la programación a aspectos se empieza aplicar la ingeniería de software para el modelado y el diseño de aplicaciones, se integra la metodología Theme Approach, partiendo de los requerimientos dados por el analista para el sistema.

Al entender el fundamento de la programación orientada a aspectos, se busca demostrar que este paradigma es más eficiente y eficaz que la programación orientada a objetos, el propósito es hacer una comparación entre estos dos paradigmas, para esto se tomo un programa de escritorio: "Simulador de un cajero automático ATM" orientado a objetos; basándonos en su metodología se aplicará la metodología planteada para desarrollar el cajero orientado a aspectos, de esta forma se evidencian las ventajas y desventajas de utilizar este paradigma. El desarrollo de este proyecto, busca estudiar este paradigma y seguir la metodología planteada pasó por paso, para llegar a la construcción del nuevo sistema con interfaz Web.

Palabras claves: Theme Approach, Theme/Doc, Aspects, Crosscutting, Tangling, Scattering.

INTRODUCCIÓN

El desarrollo de software no es una tarea fácil a la hora de realizar modelado de proyectos, de tal manera que puedan ser llevados por todo el proceso del ciclo de vida del software, ya que cada uno de ellos tiene características que los hacen diferentes. Prueba de ello es que existen numerosas metodologías que influyen de distintas maneras en el proceso de desarrollo.

La Programación Orientada a Objetos POO introdujo un avance importante forzando el encapsulamiento y la abstracción, por medio de una unidad que captura tanto funcionalidad como comportamiento y estructura interna. A esta entidad se la conoce como clase y su principal característica es que enfatiza datos y algoritmos. A través de la Programación Orientada a Objetos POO o de otras técnicas de abstracción de alto nivel, se logra un diseño y una implementación que satisface la funcionalidad básica y que presenta niveles de calidad aceptable. Sin embargo, existen conceptos entrecruzados que no pueden encapsularse dentro de una unidad funcional, debido a que atraviesan todo el sistema o varias partes de él.

Durante la evolución de la ingeniería de software se puede observar que los progresos más significativos se han obtenido gracias a la aplicación de uno de los principios fundamentales a la hora de resolver cualquier problema, incluso de la vida cotidiana, la descomposición de un sistema complejo en partes que sean más fáciles de manejar.

El proyecto se enfoca en estudiar el paradigma de Ingeniería del Software Orientado a Aspectos, encontrando herramientas y metodologías que permitan plantear una metodología para la aplicabilidad en el desarrollo de aplicaciones orientadas a la Web. Para realizar el comparativo de los paradigmas orientados a objetos y orientados a aspectos; se analiza un programa desarrollado bajo el paradigma orientado a objetos para evidenciar las ventajas y desventajas entre estos paradigmas.

1. PLANTEAMIENTO DEL PROBLEMA Y JUSTIFICACIÓN

La ingeniería de software tradicional carece actualmente de mecanismos adecuados para abstraer y encapsular conceptos que no forman parte de la funcionalidad básica de los sistemas, tales como la sincronización, distribución, seguridad, administración de memoria y otros[1]. El resultado de esta insuficiente abstracción es una notable disminución de la calidad del software final.

Durante la evolución de la ingeniería de software se puede observar que los progresos más significativos se han obtenido gracias a la aplicación de uno de los principios fundamentales a la hora de resolver cualquier problema, incluso de la vida cotidiana, la descomposición de un sistema complejo en partes que sean más fáciles de manejar[2].

En los primeros desarrollos de lenguajes de programación se tenía un código en el que no había separación de conceptos, datos y funcionalidad, a esta etapa se la conoce como la del código spaghetti, se tenía un enredo entre datos y funcionalidad. Luego, se da paso a la aplicación de la descomposición funcional, poniendo en práctica el principio de 'divide y vencerás'.

La principal ventaja que ofrece la descomposición funcional es la facilidad de integración de nuevas funciones. También ofrece inconvenientes frecuentes como

[1] ASTEASUAIN, Fernando; CONTRERAS, Bernando; ESTÉVEZ, Elsa y FILLOTTRANI, Pablo. Programación Orientada a Aspectos: Metodología y Evaluación. Universidad Nacional del Sur, Argentina, 2005. p. 1-12.

[2] REINA QUINTERO, Antonia Maria. Visión Genereal de la Programación Orientada a Aspectos. Universidad de Sevilla, 2000.

son, las funciones poco claras debido a la utilización de datos compartidos y los datos esparcidos por todo el código, al integrar un nuevo tipo de datos, este modifica varias funciones produciendo un enredo de los objetos en funciones de alto nivel que involucran a varias clases de la Programación Orientada a Objetos POO; lo que busca la programación orientada a aspectos es solucionar este problema.

Al trabajar y dar soluciones a las desventajas planteadas, se dio otro paso en el desarrollo de los sistemas de software. La Programación Orientada a Objetos POO, uno de los avances más importantes de los últimos años en la ingeniería del software para construir sistemas complejos utilizando el principio de descomposición, el modelo de objetos se ajusta mejor a los problemas del dominio real que al de la descomposición funcional.

Continuando con la Ingeniería de Software Orientada a Aspectos AOSD, por Aspect-Oriented Software Development es un paradigma para el desarrollo de software, resultante de la evolución de la Ingeniería de Software Orientada a Objetos y de otros modelos de desarrollo. Una parte importante de la Ingeniería de Software Orientada a Aspectos es el Modelado Orientado a Aspectos OA, que se centra en las técnicas para identificar, analizar, manejar y representar aspectos en el proceso de diseño del software; entendiendo como aspecto una unidad modular que se disemina por la estructura de otras unidades funcionales. El principal objetivo consiste en la definición de técnicas, métodos y herramientas basadas en UML[3].

La Programación Orientada a Aspectos POA y la Ingeniera de Software Orientada a Aspectos AOSD están en su primera etapa, constantemente surgen nuevos problemas, herramientas y contextos en los cuales es posible aplicar aspectos.

[3] RODRÍGUEZ ECHEVERRIA, Roberto. Modelando Procesos de Negocio Web desde una Perspectiva Orientada a Aspectos. Universidad de Extremadura.2007.

Este panorama hace pensar que la Programación Orientada a Aspectos POA se encuentra en el mismo lugar que se encontraba la Programación Orientada a Objetos POO hace veinte años.

Atendiendo los conceptos enunciados en este capítulo de planteamiento y justificación, a continuación, en el numeral dos del presente documento se presentan a través de los objetivos del proyecto, el alcance del mismo.

2. ESTADO DEL ARTE

La Programación Orientada a Aspectos POA, paradigma para el desarrollo de aplicaciones de software, ha sido una evolución de más alto nivel respecto a la Programación Orientada a Objetos POO, pero no se considera una extensión de ésta, porque puede utilizarse con los diferentes estilos de programación. Existen proyectos que están en marcha para determinados problemas basados en ideas orientadas a aspectos; un ejemplo es la implementación de los patrones de Gamma con AspectJ, realizada por Gregor Kiczales[4].

La Programación Orientada a Aspectos POA ha traído consigo tecnologías, lenguajes y métricas. En la actualidad existen diferentes tecnologías que permiten el buen desarrollo del sistema, algunas de estas son AOPMETRICS, ARJ, THEME/UML, AFANTI [5].

En los lenguajes aplicables para la Programación Orientada a Aspectos POA, el más conocido y utilizado es AspectJ, aunque existen otras plataformas más o menos maduras para iniciarse en el tema de aspectos. Composition Filters es un trabajo previo al de Kiczales, desarrollado por el grupo TRESE en el departamento de Ciencias de Computación de la Universidad de Twente, en Holanda. Éste ha sido adaptado a los conceptos de aspectos y es soportado sobre varios lenguajes tradicionales, como Java, C++, Smalltalk y .NET. HyperJ es un proyecto de IBM

[4] HANNEMANN, Jan y KICZALES, Gregor. "Design Pattern Implementation in Java and AspectJ". University of British Columbia, [En línea] 2003 [Citado marzo 2007]. Disponible en Internet: <http://www.cs.ubc.ca/labs/spl/papers/2002/oopsla02-patterns.pdf>

[5] AOSD STEERING COMMITTEE. AOSD [online], 2005. [Citado 10 febrero 2006]. Disponible en Internet: <http://www.aosd.net>.

que soporta un modelo por aspectos. Otras implementaciones son JBoss-AOP, Spring AOP, AspectC++, PHPAspect, AspectS y AspectXML1[6].

A fin de cuantificar los beneficios de la Programación Orientada a Aspectos POA y AspectJ se definen y se aplican métricas orientadas a aspectos. Las métricas tradicionales no son de significativa utilidad aplicadas sobre programas basados en este paradigma para el desarrollo de software, esto se debe a que no incorporan en sus cálculos el efecto de una Programación Orientada a Aspectos POA. Se definen dos métricas orientadas a aspectos Beneficio Programación Orientada a Aspectos y Limpieza, las cuales incluyen los conceptos relacionados al paradigma de aspectos, reflejando de una manera más fiel y precisa el desempeño de la Programación Orientada a Aspectos POA.

El padre de la Programación Orientada a Aspectos POA desde sus comienzos ha dado conferencias en distintos países, algunas de ellas han sido publicadas en la Web, en las cuales se destacan las siguientes: Aspect Oriented Programming: Radical Research in Modularity; Google TechTalks, May 16 de 2006 y Aspect Oriented Programming.

La Programación Orientada a Aspectos POA es usada para referirse a varias tecnologías relacionadas como los métodos adaptivos, los filtros de composición, la programación orientada a sujetos o la separación multidimensional de competencias[7].

---

[6] KICKZALES, Gregor; LAMPING, John; MENDHEKAR, Anurag; MAEDA, Chris; VIDEIRA LOPES, Cristina; LOINGTIER, Jean-Marc y IRWIN, John. “Aspect- Oriented Programming”, in Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland, 1997.

[7] WIKIPEDIA Foundation, Inc.. Programación Orientada a Aspectos (POA) [En línea], 2007. [Citado 10 febrero 2007]. Disponible en Internet: <http://es.wikipedia.org/wiki/Programaci%C3%B3n_Orientada_a_Aspectos>.

El uso de la Programación Orientada a Aspectos POA se extiende día a día, sin embargo a nivel empresarial todavía no se considera como una opción a tener en cuenta, quizás porque se desconocen las grandes posibilidades que ofrece, como la reducción de código, eficiencia y por su falta de estandarización, ahora empiezan a surgir movimientos de unificación como el promovido por los creadores de AspectJ y AspectWerkz (su competidor directo) que van a unirse para crear un único framework Java para la Programación Orientada a Aspectos POA[8].

Los aspectos han tenido un avance significativo en los sistemas para dar soluciones al problema de las incumbencias (crosscutting). Como respuesta a este problema de incumbencias nace la Programación Orientada a Aspectos POA. Permitiendo a los programadores escribir, ver y editar un aspecto diseminado por todo el sistema como una entidad por separado, de una manera inteligente, eficiente e intuitiva. La Programación Orientada a Aspectos POA es una metodología de programación que aspira a soportar la separación de las propiedades para los aspectos que afectan al análisis y al desarrollo del software como:

- La correspondencia: La implementación simultánea de varios conceptos oscurece la correspondencia entre un concepto y su implementación, resultando en un pobre mapeo.

- Menor productividad: La implementación simultánea de múltiples conceptos distrae al desarrollador del concepto principal, por concentrarse también en los conceptos periféricos, disminuyendo la productividad.

- Poco reusable: Al tener implementados en un mismo módulo varios conceptos, resulta en un código poco reusable.

[8] KICKZALES, Op. cit., referencia [6]

- Baja calidad de código: El Código Mezclado produce un código propenso a errores. Además, al tener como objetivo demasiados conceptos al mismo tiempo se corre el riesgo de que algunos de ellos sean subestimados.

- Evolución más dificultosa: Como la implementación no está completamente modularizada los futuros cambios en un requerimiento implican revisar y modificar cada uno de los módulos donde esté presente ese requerimiento. Esta tarea se vuelve compleja debido a la insuficiente modularización.

Esto implica separar la funcionalidad básica y los aspectos, y los aspectos entre sí, a través de mecanismos que permitan abstraerlos y componerlos para formar todo el sistema. La Programación Orientada a Aspectos POA es un desarrollo que sigue a la Programación Orientada a Objetos POO, y como tal, soporta la descomposición orientada a objetos, además de la procedimental y la funcional.

A medida que la ingeniería de software fue creciendo, se fueron introduciendo conceptos que la llevaron a una programación de alto nivel, la noción de tipos, bloques estructurados, agrupamientos de instrucciones a través de procedimientos y funciones como una forma primitiva de abstracción, unidades, módulos, tipos de datos abstractos, genericidad y herencia. Los progresos más importantes se han obtenido aplicando tres principios, los cuales están estrechamente relacionados entre sí: abstracción, encapsulamiento y modularidad. Esto último, consiste en la noción de descomponer un sistema complejo en subsistemas más fáciles de manejar, siguiendo la antigua técnica de “divide y vencerás”.

Un avance importante lo introdujo la Programación Orientada a Objetos POO, donde se fuerza el encapsulamiento y la abstracción, a través de una unidad que captura tanto funcionalidad como comportamiento y estructura interna. Esta entidad se conoce como clase. La clase hace énfasis tanto en los algoritmos como

en los datos. La Programación Orientada a Objetos POO está basada en cuatro conceptos[9]:

- Definición de tipos de datos abstractos.
- Herencia
- Encapsulamiento
- Polimorfismo

Ya sea a través de la Programación Orientada a Objetos POO o con otras técnicas de abstracción de alto nivel, se logra un diseño y una implementación que satisface la funcionalidad básica, y con una calidad aceptable. Sin embargo, existen conceptos que no pueden encapsularse dentro de una unidad funcional, debido a que atraviesan todo el sistema, o varias partes de él (crosscutting concerns). Algunos de estos conceptos son: sincronización, manejo de memoria, distribución, chequeo de errores, profiling, seguridad o redes, entre otros.

Las descomposiciones actuales no soportan una completa separación de conceptos, clave para manejar un software entendible y evolucionable. Se afirma, que las técnicas tradicionales no soportan de una manera adecuada la separación de las propiedades de aspectos distintos a la funcionalidad básica, esta situación tiene un impacto negativo en la calidad del software.

[9] PROGRAMEMOS.COM. AspectJ por Fernando Asteasuain [En línea], 2005. [Citado 12 febrero 2007]. Disponible en Internet: <http://www.programemos.com/index.php?option=com_content&task=view&id=%20158&Itemid=68>.

Teniendo en cuenta la forma en que la ingeniería de software ha crecido, siempre está de la mano de nuevas formas de descomposición que implicaron luego nuevas generaciones de sistemas, es válido preguntar si también junto con la Programación Orientada a Aspectos POA nacerá una nueva generación de sistemas de software[10].

[10] ASTEASUAIN, Fernando y CONTRERAS, Bernando. Programación Orientada a Aspectos, Análisis del paradigma. Universidad Nacional del Sur, Argentina, 2002. p. 1-130.

3. MARCO TEÓRICO

3.1 ASPECTOS

Según Gregor Kiczales, un aspecto es una unidad modular que se disemina por la estructura de otras unidades funcionales, entendiendo como unidad modular la separación en pequeños grupos, dependiendo su funcionamiento y su comportamiento. Los aspectos existen tanto en la etapa de diseño como en la de implementación. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. Un aspecto de programa de código es una unidad modular del programa que aparece en otras unidades modulares del programa, como se muestra en la figura, un programa es separado en pequeños grupos, dependiendo de su funcionamiento como lo es la sincronización, la distribución, gestión de memoria[11], quedando como programa la funcionalidad básica. Ver (figura 1).

Según Guadalupe Ibargüengoitia, un aspecto es una característica, comportamientos o incumbencia (concern), entre los comportamientos se encuentran la seguridad, persistencia, presentación, distribución y manejo de errores, que es importante para algún involucrado en el desarrollo de software y que debe poder rastrearse desde la captura de requerimientos hasta el código[12].

---

[11] DÍAZ, Elizabeth; ÁLVAREZ, Juan y CONGOTE, John. Aspect-oriented Programming AOP [En línea], 2006. [Citado 20 febrero 2006]. Disponible en Internet: <http://www.slideshare.net/jcongote/programacion-orientada-a-aspectos/>.

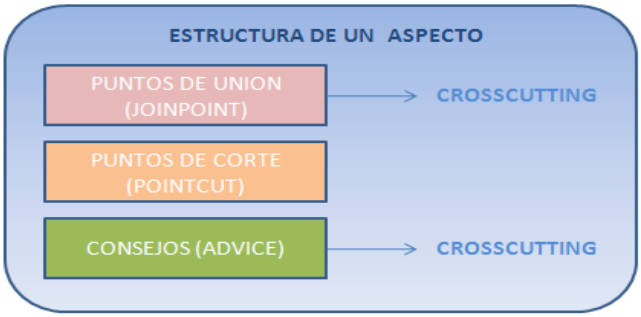
[12] IBARGÜENGOITIA, Guadalupe. Desarrollo de Software Orientado a Aspectos. UNAM, México, 2006. p. 1-34.

Figura 1. Estructura de un programa con POA.

Fuente. DÍAZ, Elizabeth; ÁLVAREZ, Juan y CONGOTE, John. Aspect-oriented Programming AOP [En línea], 2006. [Citado 20 febrero 2006]. Disponible en Internet: <http://www.slideshare.net/jcongote/programacion-orientada-a-aspectos/>.

De manera más informal, se dice que, los aspectos son la unidad básica de la Programación Orientada a Aspectos POA y pueden definirse como las partes de una aplicación que describen los asuntos claves relacionadas con el rendimiento y el funcionamiento. También pueden verse como los elementos que se diseminan por todo el código y que son difíciles de describir localmente con respecto a otros componentes, en la (figura 2) se muestra la estructura de un aspecto, mas adelante se explica cada una de las partes que componen los aspectos.

Figura 2. Estructura de un aspecto



Fuente., Ibíd.

Es necesario diferenciar un componente de un aspecto, debido a que el componente, son módulos que componen la aplicación, sin importar que sean independientes unos de otros y los aspectos son una unidad modular que se disemina por la estructura de otras unidades funcionales, a continuación mostramos las diferencias.

Podemos diferenciar un aspecto de un componente:

- Un componente: puede encapsularse claramente dentro de un procedimiento generalizado. Un elemento es claramente encapsulado si está bien localizado, es fácilmente accesible y resulta sencillo componerlo.

- Un aspecto: no puede encapsularse claramente en un procedimiento generalizado. Los aspectos tienden a ser propiedades que afectan la performance o la semántica de los componentes en forma sistemática.

Los aspectos no suelen ser unidades de descomposición funcional del sistema, son las propiedades que afectan al rendimiento o la semántica de los componentes. Algunos ejemplos de aspectos son, los patrones de acceso a memoria, la sincronización de procesos concurrentes, el manejo de errores y demás[13].

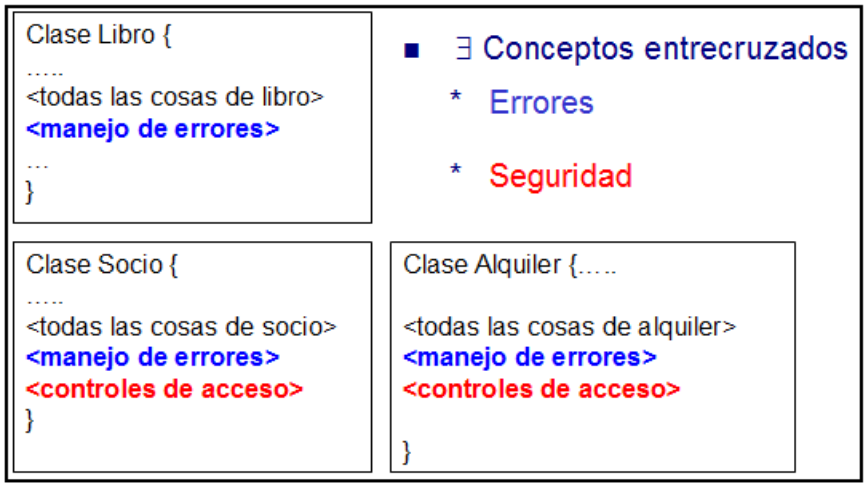
3.2 CROSSCUTTING (INCUMBENCIAS)

Hay algunos casos donde los aspectos del comportamiento de un sistema parecen relacionados entre sí en el código. Como se muestra en la figura, la funcionalidad

[13] REINA. Op. cit., referencia [2]

básica, la clase libro, socio y alquiler están bien, pero el manejo de errores y la seguridad se esparcen por todo el sistema creando crosscutting entre ellos. (Ver figura 3)

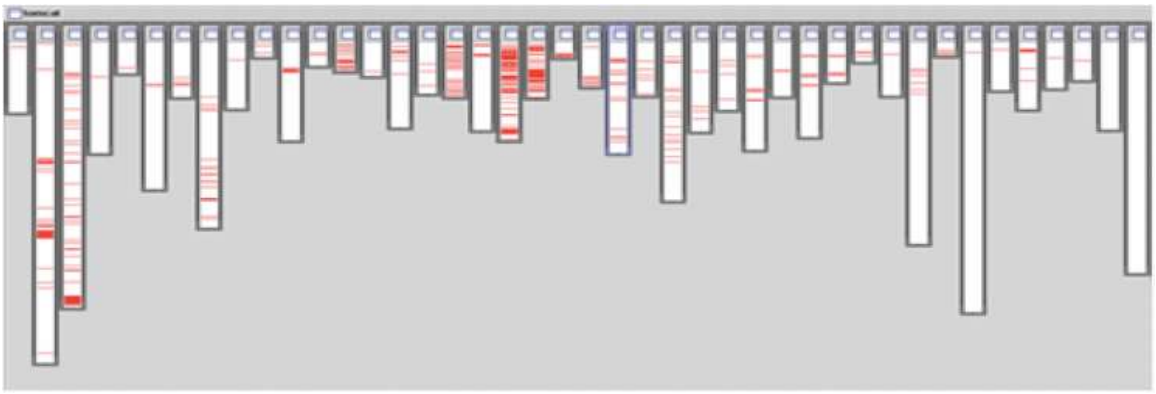
Figura 3. Ejemplo del comportamiento de los aspectos.



Fuente. ASTEASUAIN, Fernando y CONTRERAS, Bernando. Programación Orientada a Aspectos, Análisis del paradigma. Universidad Nacional del Sur, Argentina, 2002. p. 1-130.

Las consecuencias directas de estas incumbencias transversales son el código disperso y enredado (*scattering and tangling*). Se habla de código disperso cuando un mismo servicio es invocado de manera similar desde muchas partes del programa. Ver (figura 4), cada barra representa un módulo del sistema, las líneas de código que invocan el servicio de registro de la actividad de la aplicación están en rojo.

Figura 4. Crosscutting



Fuente. CÁCERES, Abdiel, Programación Orientada a aspectos. Centro de Investigación y de Estudios Avanzados - IPN, México 2004.

Uno de los objetivos de la Programación Orientada a Aspectos es hacer posible el trato con aspectos del comportamiento de un sistema que incurran en crosscutting, de forma, lo más separadamente posible. Los programadores deberán primero expresar cada uno de los aspectos de un sistema en forma separada y natural, automáticamente esas descripciones serán combinadas en un ejecutable final mediante el tejedor de aspectos. El tejedor se encargará de combinar los lenguajes, de la funcionalidad básica y de los aspectos, que mas adelante se nombrarán.

Los sistemas reales suelen tener cierto “crosscutting inherente”, donde en lugar de ignorar este problema se debe buscar la tecnología adecuada para tratar con el. Este problema suele aparecer en las fases avanzadas del proceso de desarrollo, normalmente se comienza con un claro y jerárquico diseño funcional y después se añadirán manualmente varios aspectos como sincronización, gestión de memoria, entre otros, quedando un código más o menos enmarañado. En cambio, si se separan los aspectos del código funcional se tendrá un código más simple. Una vez identificado los aspectos relevantes del sistema es necesario encontrar:

- Mecanismos eficientes para la composición de los aspectos

- Medios lingüísticos apropiados para expresar los propios aspectos[14].

3.3 PROGRAMACIÓN ORIENTADA A ASPECTOS POA

Otro concepto que debe definirse es el de programación orientada a aspectos, la programación orientada a aspectos es un paradigma para el desarrollo de software cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos. Gracias a la Programación Orientada a Aspectos POA se pueden capturar los diferentes conceptos que componen una aplicación en entidades bien definidas, de manera apropiada en cada uno de los casos y eliminando las dependencias inherentes entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables. Varias tecnologías con nombres diferentes se encaminan a la consecución de los mismos objetivos y así, el término Programación Orientada a Aspectos POA es usado para referirse a varias tecnologías relacionadas como los métodos adaptivos, los filtros de composición, la programación orientada a sujetos o la separación multidimensional de competencias[15].

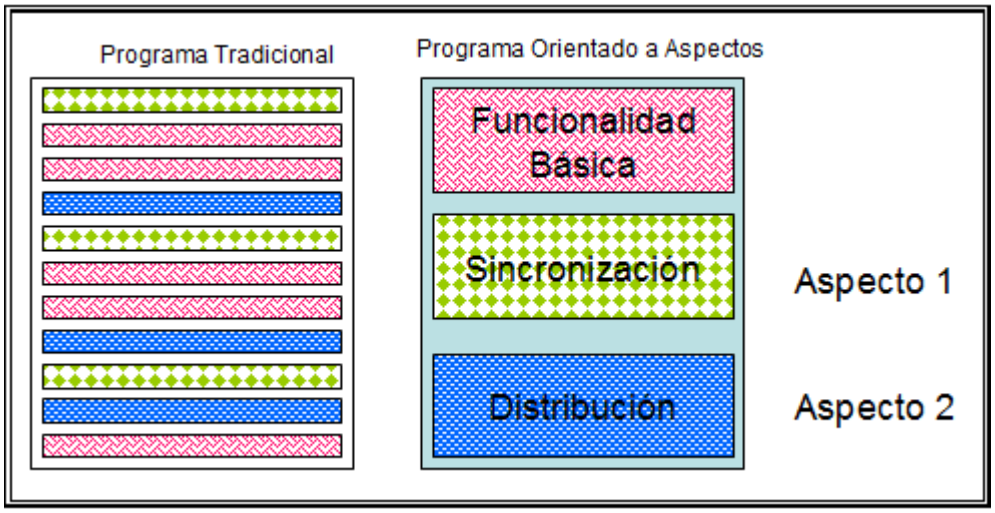
En la (figura 5), se puede observar la comparación de un Lenguaje de Programación General LPG con la Programación Orientada a Aspectos POA. En la parte izquierda identificada como estructura del programa tradicional, se puede

[14] GÓMEZ, Pedro y VILLALOBOS, Jesus. Introducción a la Programación Orientada a Aspectos (POA). Universidad de Castilla la Mancha, 2005.

[15] WIKIPEDIA. Op. cit., referencia [7]

ver que está formada por una serie de bandas horizontales. Cada banda está rellena utilizando una trama distinta y cada trama representa una funcionalidad. En la parte derecha identificada como la Programación Orientada a Aspectos POA, está formado por tres bloques compactos, cada uno de los cuales representa un aspecto o competencia dentro del código.

Figura 5. Comparación LPG y POA



Fuente. DÍAZ. Op. Cit., P. 18

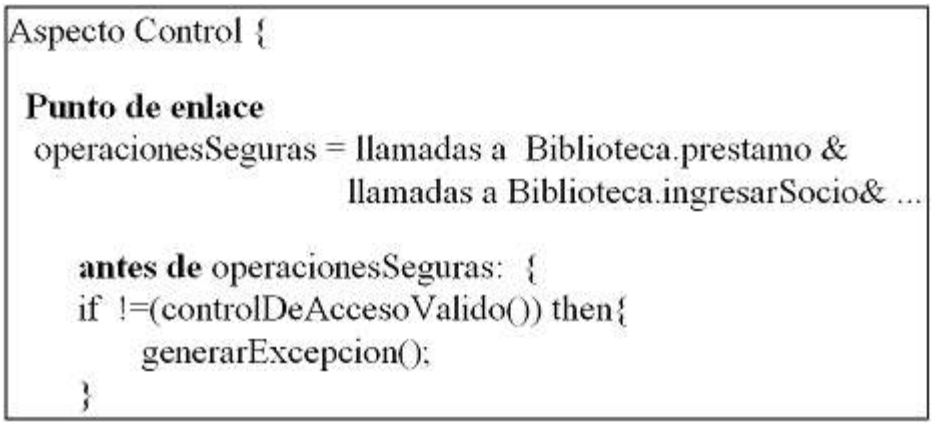
Como se puede observar, en el Lenguaje de Programación General LPG estos mismos bloques de funcionalidad quedan repartidos por todo el código, mientras en que la versión orientada a aspectos tenemos un programa más compacto y modularizado[16].

3.3.1 Punto de Unión ó puntos de enlace (joinpoint). Los puntos de unión dentro de la Programación Orientada a Aspectos POA están definidos como una clase

[16] REINA. Op. cit., referencia [2]

especial de interfaz entre los aspectos y el código base, explicando esta definición más a fondo se podría decir que son puntos que representa un "momento" en la ejecución de un programa, por ejemplo, una llamada a un método, o un constructor, o el acceso a un método de una clase en particular. (Ver Figura 6)

Figura 6. Punto de unión y punto de corte.



Fuente. ASTEASUAIN. Op. Cit., P. 18

3.3.2 Intersección o Puntos de Corte (pointcut). Los puntos de corte en sí son una declaración de un conjunto de puntos de unión, los cuales son parte de los aspectos e indican en donde o en que puntos de unión se aplicará el aspecto.

Es importante tener en cuenta lo siguiente: permite capturar o identificar puntos de unión en el flujo de un programa”

3.3.3 Consejo o Avisos (advice). Un aviso define el comportamiento que se quiere invocar cuando se alcance un determinado punto de intersección o de corte, dicho en otras palabras, son las acciones a tomar, es decir indica que hay que hacer en

esos puntos de unión. Se debe tener en cuenta que: permite definir las acciones a tomar en el flujo del programa.

Existen varios tipos de advice, por ejemplo el before, que se ejecuta antes de un jointpoint o el after que se ejecuta después.

3.3.4 Introducciones (introduction). La introducción permite definir un comportamiento que se añade a un objeto en tiempo de ejecución para que implemente una interfaz adicional aparte de la suya original.

3.3.5 Tejedor (Weaving). Es el proceso de insertar aspectos en el código de la aplicación en el punto apropiado, el sistema que se quiere obtener es uno solo. Pues ahora se tendrá que definir una determinada relación entre los aspectos y el código base. Cuando se tienen definidos y separados los código base y los aspectos, será necesario establecer un tipo de enlace entre estos, de tal forma que puedan interactuar y así poder lograr lo que se planteó.

En la Programación Orientada a Aspectos POA se trata de separar al máximo el código base de los aspectos, pero de alguna forma en algún momento se tienen que unir, y es aquí donde se hace el uso del tejedor (weaver), que es el que permite tejer los aspectos en el código base.

Se clasifica a los Tejedores en dos tipos, definidos según el momento en el que introducen los mecanismos para decidir sobre la aplicación de dichos aspectos:

- Estáticos: en este caso son los compiladores y preprocesadores que generan un único ejecutable con el código de los aspectos y el código base.

- Dinámicos: su característica se controla a través de distintos eventos que suceden durante la ejecución de un programa para incorporar los mecanismos que decidan la aplicación de aspectos [17].

3.4 REQUERIMIENTOS PARA DESARROLLO DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS POA

- Un lenguaje para definir la funcionalidad básica, conocido como lenguaje base o componente. Podría ser un lenguaje imperativo, o un lenguaje no imperativo (C++, Java, Lisp, ML, C.Net).

- Uno o varios lenguajes de aspectos, para especificar el comportamiento de los aspectos. (COOL-sincronización, RIDL-distribución, AspectJ-propósito general.)

- Un tejedor de aspectos (Weaver), que se encargará de combinar los lenguajes. Tal proceso se puede retrasar para hacerse en tiempo de ejecución o en tiempo de compilación.

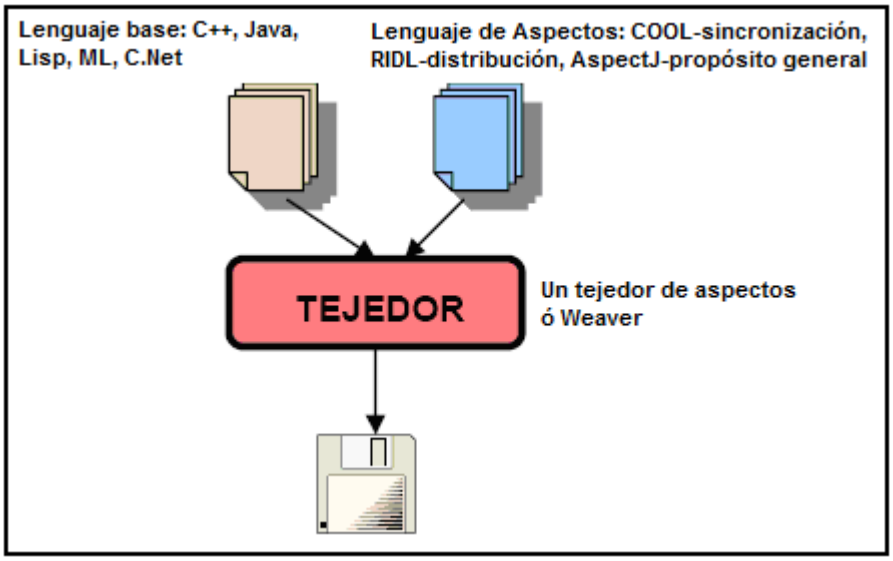
En la figura 7, podemos encontrar como están distribuidos los lenguajes, ya sea para el código fuente, para los aspectos y el tejedor.

Los lenguajes orientados a aspectos definen una nueva unidad de programación de software para encapsular aquellos conceptos que cruzan todo el código. A la hora de "tejer" los componentes y los aspectos para formar el sistema final, es claro que se necesita una interacción entre el código de los componentes y el

[17] RASHID, Awais; MOREIRA, Ana y ARAÚJO, João. Modularisation and Composition of Aspectual. Lancaster University y FCT, Universidade Nova de Lisboa, 2002.

código de los aspectos. También es claro que esta interacción no es la misma interacción que ocurre entre los módulos del lenguaje base, donde la comunicación está basada en declaraciones de tipo, llamadas a procedimientos y funciones. La Programación Orientada a Aspectos POA define entonces una nueva forma de interacción, provista a través de los puntos de enlace (join points).

Figura 7. Lenguajes para la programación orientada a Aspectos



Fuente. MANZANARES, Guillén. Programación Orientada a Aspectos. Una experiencia práctica con AspectJ, Universidad de Murcia, 2005.

Los puntos de enlace brindan la interfaz entre aspectos y componentes; son lugares dentro del código donde es posible agregar el comportamiento adicional que destaca a la Programación Orientada a Aspectos POA. Dicho comportamiento adicional es especificado en los aspectos.

En la Programación Orientada a Aspectos POA el principal encargado es el tejedor quien debe realizar la parte final y más importante: “tejer” los diferentes

mecanismos de abstracción y composición que aparecen tanto en los lenguajes de aspectos como en los lenguajes de componentes, guiado por los puntos de enlace.

3.5 OBJETIVOS DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS POA

Entre los objetivos que se ha propuesto la Programación Orientada a Aspectos POA están, principalmente, separar conceptos y minimizar las dependencias entre ellos. Con el primer objetivo se persigue que cada decisión se tome en un lugar concreto y no diseminado por la aplicación. Con el segundo, se pretende desacoplar los distintos elementos que intervienen en un programa.

La idea central que persigue la Programación Orientada a Aspectos POA es permitir que un programa sea construido describiendo cada concepto separadamente.

El soporte del paradigma para el desarrollo de software se logra a través de una clase especial de lenguajes, llamados Lenguajes Orientados a Aspectos LOA, ellos brindan mecanismos y constructores para capturar aquellos elementos que se diseminan por todo el sistema. A estos elementos se les da el nombre de aspectos. Una definición para tales lenguajes sería: los Lenguajes Orientados a Aspectos LOA son aquellos lenguajes que permiten separar la definición de la funcionalidad pura de la definición de los diferentes aspectos [18].

[18] WIKIPEDIA. Op. cit., referencia [7]

Los Lenguajes Orientados a Aspectos LOA deben satisfacer varias propiedades deseables [19], entre ellas:

- Cada aspecto debe ser claramente identificable
- Cada aspecto debe auto-contenerse
- Los aspectos deben ser fácilmente intercambiables
- Los aspectos no deben interferir entre ellos
- Los aspectos no deben interferir con los mecanismos usados para definir y evolucionar la funcionalidad, como la herencia

3.6 VENTAJAS DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS POA

La Programación Orientada a Aspectos POA permite, de una manera comprensible y clara, definir aplicaciones considerando los problemas como crosscutting. Por aspectos, se entiende, problemas que afectan a la aplicación de manera horizontal y que este paradigma para el desarrollo de software persigue el poder tenerlos de manera aislada de forma adecuada y comprensible, dando la posibilidad de construir el sistema componiéndolos junto con el resto de los componentes.

---

[19] DEPARTAMENTO DE CS. E ING. DE LA COMPUTACIÓN. Tesis de Licenciatura "Programación Orientada a Aspectos: Análisis del Paradigma" Fernando Asteasuain - Bernardo Ezequiel Contreras [En línea], 2007. [Citado 10 febrero 2007]. Disponible en Internet: <http://www.angelfire.com/ri2/aspectos/TesisLic.htm>.

Su consecución implicaría las siguientes ventajas:

- Un código menos enmarañado, más natural y más reducido.

- Mayor facilidad para razonar sobre los conceptos, pues, están separados y las dependencias entre ellos son mínimas.

- Un código más fácil de depurar y más fácil de mantener.

- Se consigue que un conjunto grande de modificaciones en la definición de una materia tenga un impacto mínimo en las otras.

- Se tiene un código más reusable y que se puede acoplar y desacoplar cuando sea necesario.

3.7 DESVENTAJAS DE LA PROGRAMACIÓN ORIENTADA A ASPECTOS POA

En el análisis de los Lenguajes Orientados a Aspectos LOA se han hallado tres falencias[20], las cuáles surgen del hecho que la Programación Orientada a Aspectos POA está en su infancia:

- Posibles choques entre el código funcional (expresado en el lenguaje base) y el código de aspectos (expresados en los lenguajes de aspectos). Usualmente estos choques nacen de la necesidad de violar el encapsulamiento para implementar los diferentes aspectos, sabiendo de antemano el riesgo potencial que se corre al utilizar estas prácticas.

[20] KICKZALES. Op. cit., referencia [6]

- Posibles choques entre los aspectos. El ejemplo clásico es tener dos aspectos que trabajan perfectamente por separado pero al aplicarlos conjuntamente resultan en un comportamiento anormal.

- Posibles choques entre el código de aspectos y los mecanismos del lenguaje. Uno de los ejemplos más conocidos de este problema es la anomalía de herencia[21]. Dentro del contexto de la Programación Orientada a Aspectos POA, el término puede ser usado para indicar la dificultad de heredar el código de un aspecto en la presencia de herencia.

3.8 INGENIERÍA DE SOFTWARE Y DESARROLLO ORIENTADO A ASPECTOS

Diseñar un sistema basado en aspectos requiere entender qué se debe incluir en el lenguaje base, qué se debe incluir dentro de los lenguajes de aspectos y qué debe compartirse entre ambos lenguajes. El lenguaje componente ó lenguaje base debe proveer la forma de implementar la funcionalidad básica y asegurar que los programas escritos en ese lenguaje componente no interfieran con los aspectos. Los lenguajes de aspectos tienen que proveer los medios para implementar los aspectos deseados de una manera intuitiva, natural y concisa[22].

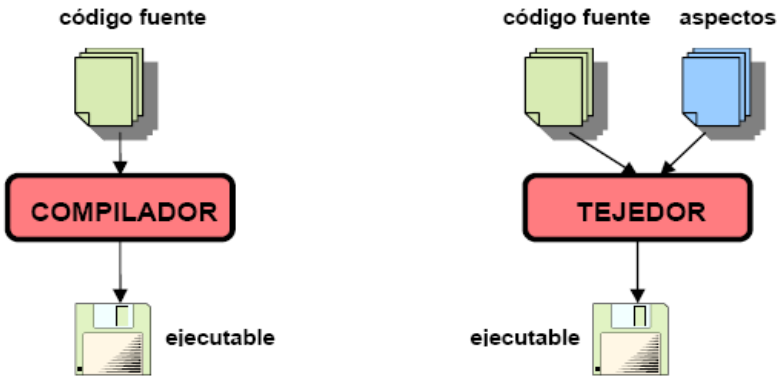
En la figura 8, se puede ver la estructura de una implementación de lenguajes tradicionales, donde un programa es desarrollado y compilado en un lenguaje especifico para generar el resultado final que es el ejecutable del software, y podemos ver la implementación con los lenguajes anteriormente nombrados en la para la programación orientada a aspectos.

---

[21] MONTES, Pablo. programación Orientada a Aspectos. Politécnico Grancolombiano, Bogotá, 2006. p. 1-101.

[22] ASTEASUAIN. Op. cit., referencia [10]

Figura 8. Generar ejecutable, enfoque tradicional y poa



Fuente. MANZANARES, Guillén. Programación Orientada a Aspectos. Una experiencia práctica con AspectJ, Universidad de Murcia, 2005.

El desarrollo de un sistema usando orientación a aspectos suele dividir en tres etapas:

3.8.1 Etapa 1. identificar competencias/intereses (concern). Consiste en descomponer los requisitos del sistema en competencias y clasificarlas en:

- Competencias básicas (core-concern): las que están relacionadas con la funcionalidad básica del sistema, de carácter funcional. Son las que Gregor Kiczales denomina componentes[23].

- Competencias transversales (crosscutting-concern): las que afectan a varias partes del sistema, relacionadas con requerimientos no funcionales del sistema, normalmente de carácter no funcional. Son las que Gregor Kiczales denomina aspectos[24].

[23] KICKZALES. Op. cit., referencia [6]
[24] Ibíd., referencia [6].

3.8.2 Etapa 2. implementar competencias/intereses. Consiste en implementar cada interés independientemente: para implementar las competencias básicas se debe utilizar el paradigma para el desarrollo de software que mejor se ajuste a ellas y el lenguaje que mejor satisfaga las necesidades del sistema y de los desarrolladores a este lenguaje se denomina lenguaje base.

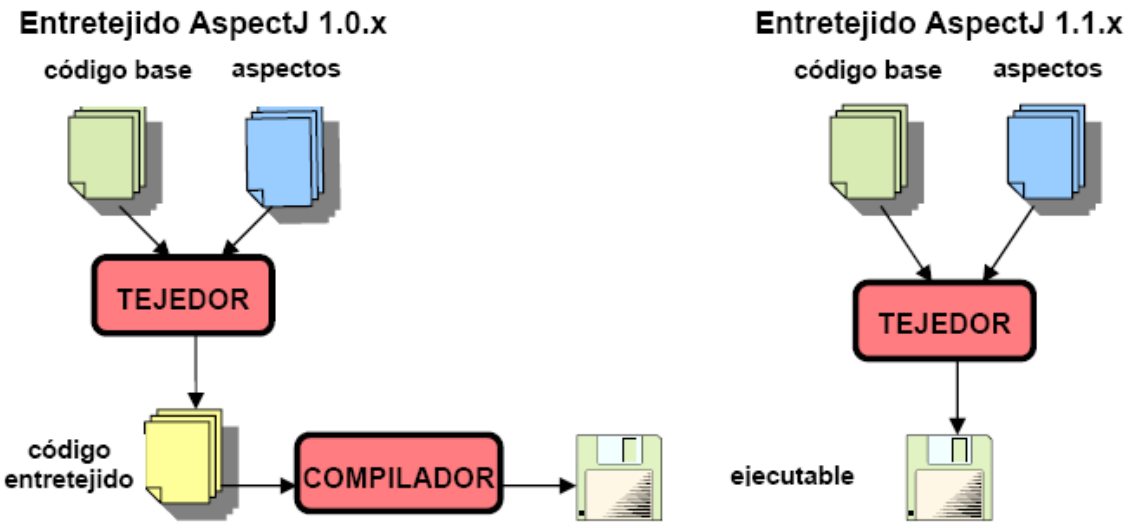
Para implementar las competencias transversales se debe utilizar uno o varios Lenguajes Orientados a Aspectos, de propósito específico o general, encapsulando cada competencia en unidades llamadas aspectos. Los Lenguajes Orientados a Aspectos deben ser compatibles con el lenguaje base para que los aspectos puedan ser combinados con el código que implementa la funcionalidad básica y así obtener el sistema final. Normalmente, los Lenguajes Orientados a Aspectos suelen ser extensiones del lenguaje base, como es el caso de AspectJ y AspectC, pero también puede ser lenguajes totalmente independientes.

3.8.3 Etapa 3. componer el sistema final. Proceso que se conoce como entretejido (weaving) o integración**.** Consiste en combinar los aspectos con los módulos que implementan la funcionalidad básica del sistema dando lugar al sistema final. El módulo encargado de realizar este proceso recibe el nombre de tejedor de aspectos (weaver) y hace uso de unas reglas de entretejido para llevar a cabo el proceso.

En la Figura 9, se recogen los conceptos basados en las etapas mencionadas anteriormente y se observa la integración de los lenguajes escogidos que comprenden los requisitos de la Programación Orientada a Aspectos POA, un lenguaje base y un lenguaje de aspectos, son compilados o entretejidos para poner a ejecutar el sistema. A diferencia que en AspectJ 1.0, se debía programar código para la compilación, del programa, pero AspectJ 1.1 ya existe un tejedor

que entreteje los dos lenguajes y compila, para obtener el ejecutable del programa.

Figura 9. Estructura de una implementación de lenguajes de aspectos



Fuente., Ibid.

Ahora se habla sobre como definir los lenguajes, identificar los aspectos y la funcionalidad básica en la etapa de diseño, y como son propuestos por diferentes autores.

John Lamping propone una visión diferente del diseño[25]. Asegura que la decisión sobre qué conceptos son base y cuáles deben ser manejados por aspectos es irrelevante, ya que no afectará demasiado a la estructura del programa, sino que a clave está en definir cuáles serán los ítems de la funcionalidad básica y cómo obtener una clara separación de responsabilidades entre los conceptos. La primera parte se hereda de la programación no orientada a aspectos y tiene la

[25] KICKZALES. Op. cit., referencia [6]

misma importancia dentro de la POA ya que la misma mantiene la funcionalidad básica. La segunda parte es inherente a la programación orientada a aspectos.

Esta parte es fundamental para una descomposición de aspectos exitosa. Diferentes aspectos pueden contribuir a la implementación de un mismo ítem de la funcionalidad básica y un sólo aspecto puede contribuir a la implementación de varios ítems. Una buena separación de responsabilidades entre los conceptos es lo que hace esto posible, porque el comportamiento introducido por los diferentes aspectos se enfoca en diferentes temas en cada caso, evitando gran parte de los conflictos. Lamping concluye que el trabajo del programador que utiliza la Programación Orientada a Aspectos POA es definir precisamente los ítems de la funcionalidad básica y obtener una buena separación de responsabilidades entre los conceptos. Luego los aspectos le permitirán al programador separar los conceptos en el código.

La filosofía de diseño orientada a aspectos consiste en cuatro pasos:

- Un objeto es algo: un objeto existe por sí mismo, es una entidad.

- Un aspecto no es algo. Es algo sobre algo: un aspecto se escribe para encapsular un concepto entrecruzado. Por definición un aspecto entrecruza diferentes componentes, los cuales en la POO son llamados objetos. Si un aspecto no está asociado con ninguna clase, entonces entrecruza cero clases, y por lo tanto no tiene sentido en este contexto. Luego, para que un aspecto tenga sentido debe estar asociado a una o más clases; no es una unidad funcional por sí mismo.

- Los objetos no dependen de los aspectos: un aspecto no debe cambiar las interfaces de las clases asociadas a él. Sólo debe aumentar la implementación de dichas interfaces. Al afectar solamente la implementación de las clases y no

sus interfaces, la encapsulación no se rompe. Las clases mantienen su condición original de cajas negras, aún cuando puede modificarse el interior de las cajas.

- Los aspectos no tienen control sobre los objetos: esto significa que el ocultamiento de información puede ser violado en cierta forma por los aspectos porque éstos conocen detalles de un objeto que están ocultos al resto de los objetos.

Esta filosofía de diseño permite a los aspectos hacer su trabajo de automatización y abstracción, respetando los principios de ocultamiento de información e interfaz manifiesta.

3.9 MÉTRICAS PARA LA PROGRAMACIÓN ORIENTADA A ASPECTOS POA

Para cuantificar los beneficios de la Programación Orientada a Aspectos POA, se definen y aplican métricas orientas a aspectos. Las métricas tradicionales no son de significativa utilidad aplicada sobre programas basados en este paradigma para el desarrollo de software, pues no incorporan en sus cálculos el efecto de una programación orientada a aspectos. Por esta razón, se definen dos métricas orientadas a aspectos, las cuales incluyen los conceptos relacionados al paradigma de aspectos, reflejando de una manera más fiel y precisa el desempeño de la POA.

En primer lugar se define la métrica Beneficio *POA*, que mide cuál es el beneficio de introducir aspectos. Para ello, define una relación de esfuerzo de desarrollo entre una implementación sin considerar aspectos, y otra donde se aplica el paradigma. Está definida en función de dos factores claves de un proyecto: el

tamaño del producto, y el esfuerzo de desarrollo. Considera el tamaño de una implementación sin y con aspectos, y pondera estos valores con un factor de desarrollo. Para esto último, se tomó en cuenta el factor de desarrollo propuesto por Rich Rice[26]. Este factor establece que para un sistema pequeño el tiempo de desarrollo promedio en Java es de 10 horas/programador, mientras que en AspectJ es de 2 horas/programador. A .n de considerar el tamaño de ambas implementaciones, se evalúa el tamaño físico de los archivos de las clases y del aspecto, medidos en Kb. La métrica se define como un cociente entre los valores aplicados a ambas implementaciones. Un valor de resultado mayor que uno significa un impacto positivo de la utilización de aspectos. La definición de la métrica es como se muestra en la fórmula 1 y fórmula 2, beneficio POA y tamaño con aspectos respectivamente.

$$Beneficio\_POA = \frac{Tamaños\_\text{sin}\_aspectos * Factor\_de\_desarrollo\_java}{Tamaño\_con\_aspectos * Factor\_de\_desarrollo\_AspectJ} \quad (1)$$

Donde,

$$Tamaño\_con\_aspectos = Tamaño\_Funcionalidad\_basica + tamaño\_Aspectos \quad (2)$$

Como ya se ha mencionado, la utilización de aspectos produce un código de funcionalidad básica más “puro”, debido a que el comportamiento de los conceptos entrecruzados queda encapsulado en los aspectos. Para definir el porcentaje de código que se reduce al introducir aspectos, se define la segunda métrica, llamada *Limpieza*. La misma, mide el porcentaje de código que se depura de la funcionalidad básica al introducir aspectos. Utiliza el tamaño de las clases, medido en Kb. La definición es como se detalla en la fórmula 3.

[26] KICKZALES. Op. cit., referencia [6]

$$Liempieza = \frac{(tamaño\_\sin\_aspectos - tamaño\_funcionalidad\_básica)*100}{Tamaño\_\sin\_aspectos} \quad (3)$$

3.10 TECNOLOGÍAS PARA LA PROGRAMACIÓN ORIENTADA A ASPECTOS POA

Otro concepto importante, para la Ingeniería de Software Orientado a Aspectos, son las tecnologías, que han evolucionado con este paradigma. Ahora se definen cada una de ellas.

3.10.1 Avanti. es un proyecto que apunta a técnicas e instrumentos característicos para sostener el análisis y pruebas de programas orientados a aspectos.

Por ejemplo, para analizar y probar los programas orientados a aspectos, las características específicas que se tienen en cuenta son los puntos, puntos de corte y los aspectos, además, aunque estas características específicas proporcionan gran fuerza para modelar los crosscutting en los sistemas de software, ellos también introducen dificultades en el análisis del programa y pruebas.

3.10.2 Aopmetrics. Su funcionalidad ayuda al usuario con el proyecto de medición de métricas, dependencias entre paquetes y el número de líneas de código. Utiliza el concepto de cortes horizontales por componentes verticales. La Ingeniería del Software orientada a aspectos se desarrolla en apoyo para aspectos

a través del ciclo de vida del software. Proporciona una extensión orientada a aspectos con unas métricas clásicas propuestas por Chidamber y Kemerer que pueden ser utilizadas para medir la reutilización, Robert Martín, Henry y Long Island. El proyecto es realizado por miembros de SPICE y e-Informatyka.pl. Aopmetrics está bajo la Licencia Pública Común, el proyecto tiene las siguientes características:

3.10.2.1 Extensiones de métricas para la POA.

- Métricas clásicas (CK metrics) de Chidamber y Kemerer
- Métricas clásicas (package dependencies metrics) de Martín de Robert
- Soporta Java (desde 1,5) y AspectJ (desde 1,5).
- Exporta medidas a XML y XLS.

3.10.2.2 La implementación actual.

- Basado en el compilador de AspectJ (ver. 1,5)
- Escrito en el lenguaje Java5 y puede ser compilado sólo en VM Java5.

3.10.3 ARJ. Es una extensión del lenguaje AspectJ. Está basado en el compilador de AspectBench (abc). Se le incluyen extensiones al lenguaje de AspectJ utilizando constructores para refinar los aspectos. El refinamiento de aspectos es una noción novedosa que proclama que los aspectos deben ser

susceptibles al refinamiento una y otra vez para volver a emplearlos y para especializar la funcionalidad. Con esto, para el refinamiento de aspectos se permite el uso de las sobresalientes metodologías de desarrollo de software: refinamiento pasó a paso y desarrollo de software en forma incremental. ARJ introduce y aplica para herencia de aspectos el método de herencia de la POA, denominado mixin-based aspect inheritance, este método mejora las capacidades de aspectos en la POA para lograr un diseño incremental, aplicando una tecnología que permite, componer y evolucionar los aspectos en una manera progresiva en las etapas del desarrollo.

3.10.4 THEME/UML. Es una extensión orientada a aspectos para UML. Presenta un acercamiento a los sistemas que se diseñan basados en el modelo orientado al objeto, pero se han agregando nuevas capacidades de descomposición, estas capacidades apoyan de una forma directa para alinear modelos del diseño con requisitos individuales. Cada modelo contiene su propio tema o diseño de un requisito individual, con conceptos del dominio (que pueden aparecer en múltiples requisitos) desde la perspectiva de ese requisito. El estándar UML se utiliza para diseñar modelos descompuestos de esta manera. Las extensiones al UML requieren composición de los modelos temáticos del diseño, esto se alcanza con una relación de composición, la cual especifica, cómo los modelos deben ser compuestos identificando conceptos solapados en diversos modelos y especificando cómo los mismos deben ser integrados.

3.11 ASPECTOS Y FRAMEWORKS

En la actualidad los Framework han tenido un gran auge, porque son diseñados para facilitar el desarrollo de software, permitiendo a los diseñadores y

programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional. Ahora vamos a entender un poco más lo que es un Framework.

En el desarrollo de software, un framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje de scripting entre otros softwares para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Un framework representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

Los Frameworks son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional. Por ejemplo, un equipo que usa Apache Struts para desarrollar un sitio web de un banco puede enfocarse en cómo los retiros de ahorros van a funcionar en lugar de preocuparse de cómo se controla la navegación entre las páginas en una forma libre de errores. Sin embargo, hay quejas comunes acerca de que el uso de frameworks añade código innecesario y que la preponderancia de frameworks competitivos y complementarios significa que el tiempo que se pasaba programando y diseñando ahora se gasta en aprender a usar frameworks.

Fuera de las aplicaciones en la informática, un framework puede ser considerado como el conjunto de procesos y tecnologías usados para resolver un problema

complejo. Es el esqueleto sobre el cual varios objetos son integrados para una solución dada[27].

En principio los framework no se pueden trabajar con aspectos, debido a que el AspectJ no soporta interfaz, pues solo trabaja modo dos. Mas adelante en las conclusiones tratamos más sobre el tema.

3.12 CARACTERÍSTICAS DE APLICACIONES ESCRITORIO Y WEB

Debido a que la aplicación tomada para analizar los aspectos es escritorio y el proyecto esta orientado a la Web, buscamos las características para estas aplicaciones, ya que se requieran mas herramientas de programación; para luego llevar a cabo una migración de Programación Orientada a Objetos a la Programación Orientada a Aspectos.

3.12.1 Web vs Escritorio. El uso de la red es un uso entregado a los usuarios de un Web Server como el Internet. Algunos negocios funcionan usando la red en una Intranet, hoy en día el uso de Web se ha convertido en el más popular.

Las aplicaciones de la Web se diseñaron específicamente para la optimización del Search Engine, y han llegado a ser cada vez más populares. Es fácil entenderlos porqué adelantan los usos de la red que se relacionan con el Internet, mientras que los usos de negocio pueden tener menos súplica en un ambiente de red.

---

[27] WIKIPEDIA Foundation, Inc. Proceso Unificado de Rational [En línea], 2007. [Citado marzo 2007]. Disponible en Internet:
<http://es.wikipedia.org/wiki/Proceso_Unificado_de_Rational>.

Una aplicación de escritorio es un programa autónomo que realiza un sistema definido de tareas bajo control del usuario. El funcionamiento de escritorio da una impulsión local y no requiere una red o una conectividad para que funcione correctamente, aunque si estuvo unido a los usos de escritorio de una red pudo utilizar los recursos de la red.

3.12.2 Pros y contras de aplicaciones de escritorio y Web. los recursos accesibles de la Web se pueden alcanzar fácilmente de cualquier computadora o localización que tenga acceso a Internet. Los usuarios se benefician en la accesibilidad. Esto significa que si un usuario tiene acceso a una computadora, teléfono o handheld con conectividad a Internet pueden utilizar los recursos de la red.

3.12.3 Bajo mantenimiento y mejoras forzadas. los recursos de escritorio necesitan ser instalados individualmente en cada computadora, mientras que los usos de red requieren una sola instalación. Muchos recursos de la red se reciben por una tercera persona y el mantenimiento bajo queda a responsabilidad de los propietarios del negocio. La capacidad de poner al día y de mantener recursos de la red sin distribuir y la instalación de software en millares de computadoras del cliente es una razón dominante del renombre de usos basados en red.

3.12.4 Riesgos crecientes de la seguridad. hay siempre un riesgo implicado cuando se trabaja en línea, la seguridad en el uso de Internet es más significativo que el uso de escritorio independiente. Algunos usos requieren más seguridad que otras, jugar Sudoku en un uso de la red que causaría poca preocupación, pero el ocuparse de fórmulas o de detalles corporativos sensibles de la contabilidad en un ambiente de red que puede ser expuesto.

3.12.5 Costo. sobre el ciclo de vida del software, los usos de la red son típicamente y considerablemente más costoso. Las aplicaciones de escritorio se compran y son raramente actualizadas, aunque algunas aplicaciones de escritorio tienen costos de mantenimiento.

Muchas aplicaciones corporativas de la red utilizan un modelo diverso, los usuarios pagan típicamente el servicio mensual para el funcionamiento del software. Los pagos se consideran los “pagos de la suscripción”. Si no pueden renovar la suscripción pueden no tener acceso a los datos almacenados en el uso de la red.

3.12.6 Conectividad. los recursos de la Web confían en la conectividad. Si no se tiene una conexión de Internet no se puede tener acceso a la información. Los negocios críticos que son tiempo sensible no pueden arriesgar la negación de los ataques del servicio o de las interrupciones de la energía para interrumpir sus operaciones y datos de acceso que sean sensibles.

3.12.7 Más lento. las compañías que tiene aplicaciones en Web y confían en el Internet para transferir datos, pueden funcionar más lento, ya que la velocidad puede variar dependiendo del número de usuarios que tienen acceso a la red.

3.12.8 Reservas y propiedad. sin importar la plataforma, las compañías necesitan tener seguros sus datos y tenerlos actualizados apropiadamente. Al usar una aplicación en la red, las compañías deben determinar claramente quién posee los datos en la compañía, y que sean seguras las políticas de aislamiento para prevenir que los datos puedan ser utilizados de una forma inadecuada por la red.

En última instancia la accesibilidad de aplicaciones basadas en red los hace muy deseables. Las aplicaciones de Web tienen algunas limitaciones fundamentales en su funcionalidad, y se satisfacen mejor para tareas específicas.

3.13 LENGUAJE UNIFICADO DE MODELADO UML

Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está apoyado en gran manera por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables[28].

Es importante decir que UML es un "lenguaje" para especificar y no un método o un proceso, se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir; es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para soportar una metodología de desarrollo de software (tal como el Proceso Unificado de Rational) pero no especifica en sí mismo qué metodología o proceso usar.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas. En UML 2.0 existen 13 tipos diferentes

[28] PRESSMAN, Roger S. Ingenieria de Software: un enfoque pracitico. Quinta edición, 2002. Ed. McGraw-Hill.

de diagramas. Para comprenderlos de manera concreta, a veces es útil categorizarlos jerárquicamente.

3.13.1 Diagramas de estructura. enfatizan en los elementos que deben existir en el sistema modelado, entre ellos se encuentran:

- Diagrama de clases
- Diagrama de componentes
- Diagrama de objetos
- Diagrama de estructura compuesta
- Diagrama de despliegue
- Diagrama de paquetes

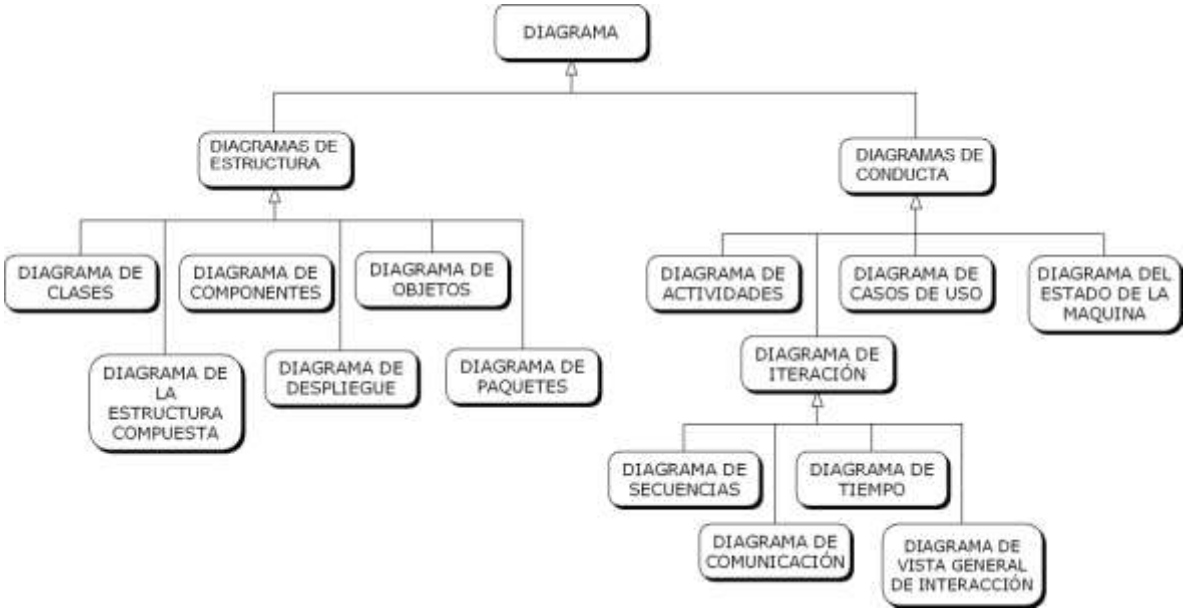
Como se muestra en la figura 10, se representa la jerarquía de los diagramas de UML antes mencionados.

3.13.2 Diagramas de comportamiento. enfatizan en lo que debe suceder en el sistema modelado:

- Diagrama de actividades
- Diagrama de casos de uso

- Diagrama de estados

Figura 10. Jerarquía de diagramas UML



Fuente. WIKIPEDIA Foundation, Inc. Proceso Unificado de Rational [En línea], 2007. [Citado marzo 2007]. Disponible en Internet: <http://es.wikipedia.org/wiki/Proceso_Unificado_de_Rational>.

3.13.3 Diagramas de Interacción. Un subtipo de diagramas de comportamiento, que enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado:

- Diagrama de secuencia

- Diagrama de comunicación

- Diagrama de tiempos (UML 2.0)

- Diagrama de vista de interacción (UML 2.0)

3.14 METODOLOGÍA DE DESARROLLO

3.14.1 Rup (Rational Unified Process). es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización[29].

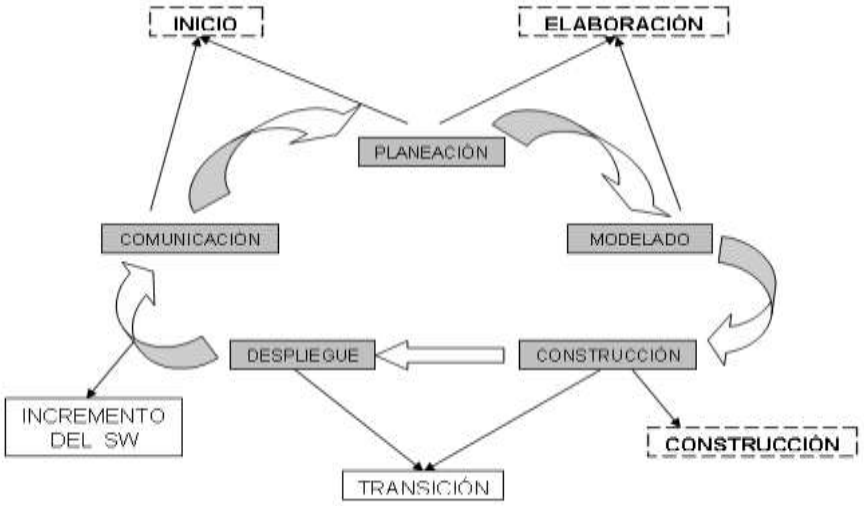
En el RUP, el esfuerzo del análisis y diseño se encuentra localizado en tres fases: inicio, elaboración y construcción. Ver (figura 11)[30].

Dentro de cada fase se desarrollan las iteraciones (ciclos completos de desarrollo incluyendo análisis, diseño, implementación y pruebas) que construyen gradualmente el sistema. La Tabla 1 muestra un conjunto de actividades que según Jacobson, Booch y Rumbaugh se realizan en las diferentes etapas. Como se puede ver, la principal diferencia en cada etapa consiste en el nivel de detalle alcanzado al final de las iteraciones que se realizan en cada una de ellas[31].

---

[29] KICKZALES. Op. cit., referencia [6]
[30] PRESSMAN. Op. cit., referencia [28]
[31] Grupo de Agentes de Software: Ingeniería y aplicaciones, Ciclo de Vida [En línea], 2006. [Citado marzo 2007]. Disponible en Internet: <http://grasia.fdi.ucm.es/ingenias/Spain/integracion/index.php>

Figura 11. Diagrama RUP



Fuente. PRESSMAN, Roger S. Ingeniería de Software: un enfoque practico. Quinta edición, 2002. Ed. McGraw-Hill.

Tabla 1. Actividades a realizar en las etapas de inicio, elaboración y construcción.

	FASES		
	INICIO	**ELABORACIÓN**	**CONSTRUCCIÓN**
OBJETIVO	Hacer creíble, que el sistema se pueda construir. Flujos de trabajo fundamentales	Hacer realidad la arquitectura	Hacer crecer el sistema

	FASES		
	INICIO	**ELABORACIÓN**	**CONSTRUCCIÓN**
ANÁLISIS	Generar casos de uso y escenarios de casos de uso Expresar requisitos como casos de uso Elaborar un boceto de arquitectura	Refinar casos de uso Perfilar la resolución de casos de uso relevantes para la arquitectura en función de paquetes y clases	Estudiar el resto de casos de uso
IMPLEMENTACION		A partir de un conjunto reducido de casos de uso se plantea: Implementación de la arquitectura Implementación de clases y de subsistemas relevantes para la arquitectura Componer los elementos identificados en el diseño para el sistema	Lograr una arquitectura finalmente asentada, con todas sus clases y subclases implementadas Realizar pruebas de que involucren varios módulos integrados para comprobar que la integración es completa y satisfactoria Realizar planes de integración de los componentes en cada iteración

	FASES		
	INICIO	**ELABORACIÓN**	**CONSTRUCCIÓN**
PRUEBAS		Seleccionar los objetivos que evaluaran la arquitectura Diseñar procedimientos de prueba en base a esos objetivos Realizar pruebas de integración entre componentes Una vez integrado, plantear pruebas del sistema	Continuar con las pruebas de la iteración anterior Añadir nuevas pruebas que tengan en cuenta el nuevo software integrado o nueva funcionalidad del software ya existente Evaluar las pruebas para verificar que se consiguen los objetivos planteados

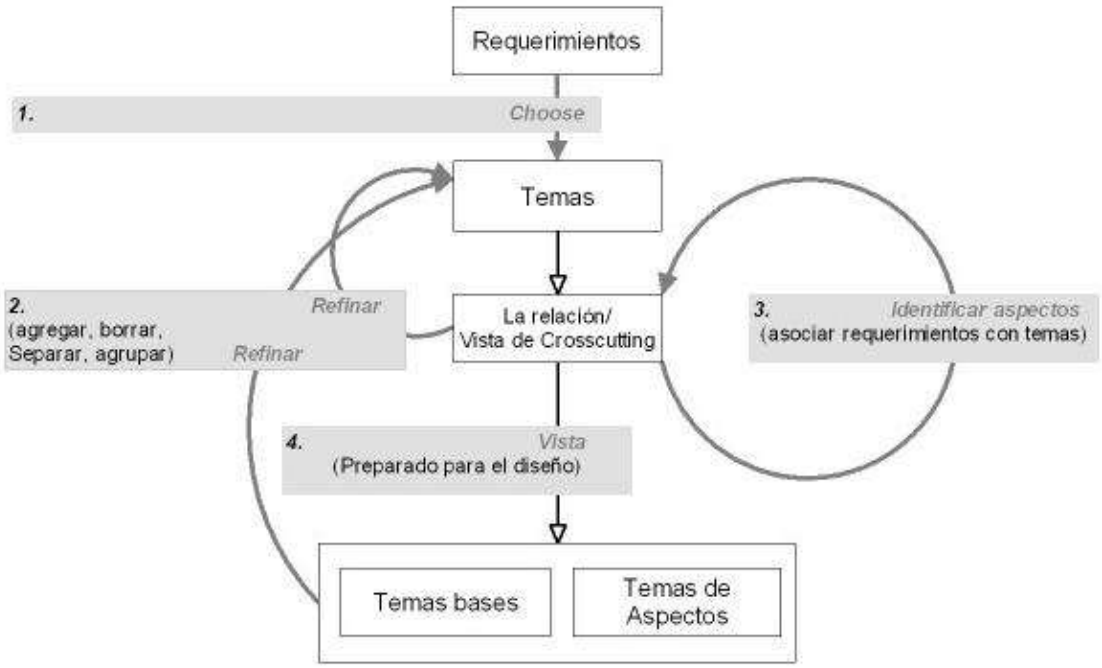
Fuente. Autoras del proyecto.

3.14.2 Metodología Theme/Approach

La metodología, básicamente identifica los concerns, los Temas (Theme), representación gráfica y esta basado en la documentación de requisitos.

Esta metodología es útil en etapas tardías de la ingeniería de requisitos, permiten detectar características no deseables del sistema, como lo son las Características ocultas, características esparcidas, características sutiles y facilita el refinamiento de requisitos.

Figura 12. Proceso del Theme/Doc, detallando la vista.



Fuente. CLARKE, Siobhàn y BANIASSAD, Elisa. Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley.

El enfoque del tema permite el diseño individual de características diferentes del sistema. En la orientada a objetos, no todos los nombres en un documento de requisitos son diseñados como objetos ni clases. Similarmente, en el enfoque del tema, no toda funcionalidad es capturada separadamente en su propio tema.

Alguna funcionalidad es demasiado pequeña para justificar la separación. En este paso, se identifican los temas del sistema identificando cuál de los temas potenciales es suficientemente mayor para ser modelado eso puede ser utilizado para refinar los temas que son mostrados en la tabla 2.

Tabla 2. Operaciones con temas y requerimientos.

Operaciones en temas	**Operaciones con requerimientos**
Agregar	Agregar/Separar
Borrar	Conectar (a otros temas)
Separación	Asociar (con un tema); Seleccionar los temas como un aspecto.
Agrupar	Aplazar (Decisión sobre los requerimientos en la etapa de diseño)

Fuente., Ibít

Para el refinamiento de los temas, se utiliza una vista de Theme/Doc llamado *theme-relationship view*, ó *relationship view*. Las vistas de la relación son creadas automáticamente dado un conjunto de requerimientos y un conjunto de los temas como diamantes. Si el nombre de un tema es mencionado en un requerimiento, hay una conexión del requisito a ese tema en la vista de la relación.

Como se pueden imaginar, las vistas de las relaciones pueden llegar a ser bastante grandes, si se tiene muchos requerimientos y muchos temas potenciales, para esto, se puede ver los requerimientos como etiquetas (o sea su número de requerimiento), o se puede ampliar para ver su contenido (Escribiendo dentro del rectángulo el requerimiento). The relationship view es *nonhierarchical*, así parezcan unos temas más grandes que otros, es sólo una coincidencia de disposición[32].

[32] CLARKE, Siobhàn y BANIASSAD, Elisa. Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley.

3.14.2.1 Reglas para la separación de los aspectos candidatos. Existen tres reglas para identificar un aspecto:

- No hay que separar el trabajo (no hay que reordenar, se pueden aislar los temas y remover los que están compartidos)

- La dominación significa la asociación (asociar un requerimiento con un tema escoger ese tema como un aspecto)

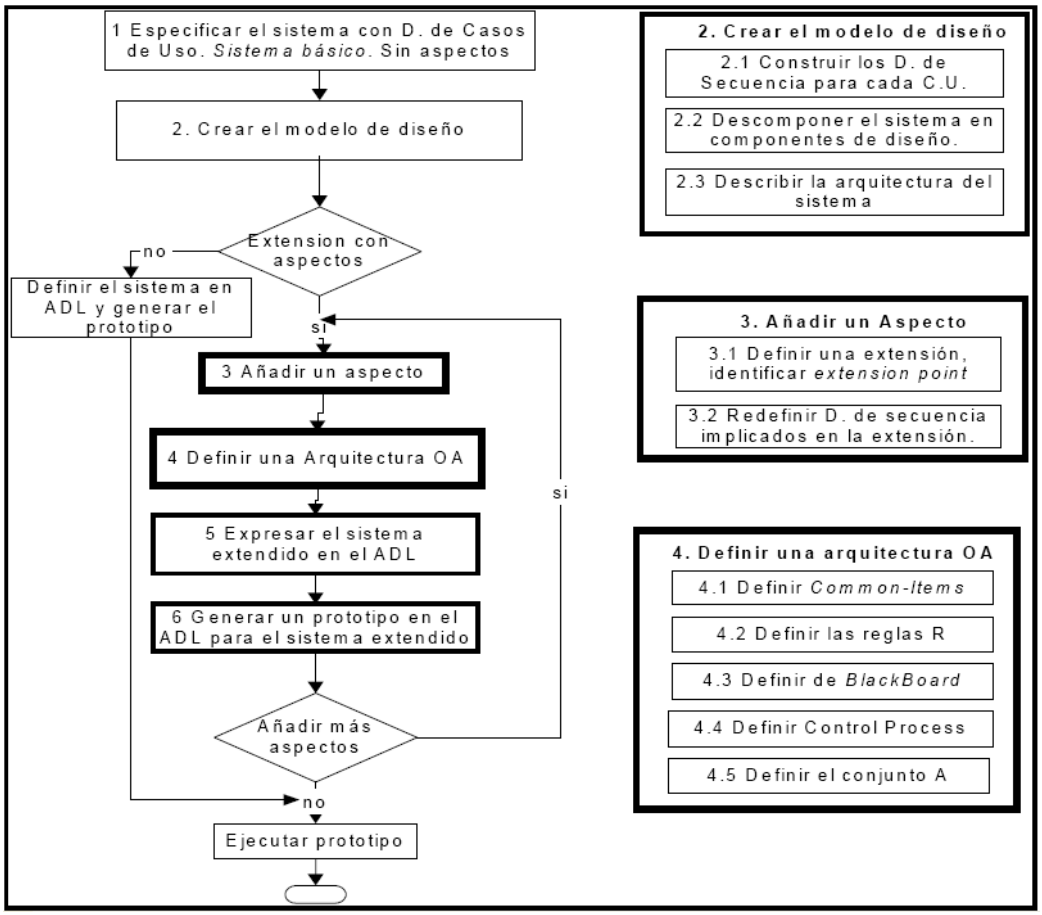
- los aspectos son apuntadores dentro del comportamiento base.

3.14.3 Modelo arquitectónico para el diseño orientado a aspectos. En el primer numeral mencionado, se propone una aproximación metodológica en la que nuevos aspectos se integran en un sistema de un modo coordinado, se definen 2 niveles arquitectónicos, se utiliza un LDA[33] (lenguajes de descripción arquitectónica) genérico (LEDA), y un modelo de coordinación (*Coordinated Roles* [Mu+99]) para gestionar la interacción de componentes funcionales y de aspecto.

La propuesta se desglosa en una serie de pasos (ver figura 9) en los que se destaca la necesidad de hacer una identificación temprana de los aspectos. Se propone hacer una especificación inicial detallada del sistema sin considerar aspectos (paso 1), creando el modelo de diseño correspondiente (paso 2), y describiendo el modelo en el LDA[34].

[33] LDA: Definen de un modo formal los componentes de diseño y su interacción.
[34] NAVASA, A; PALMA, K; MURILLO, J.M y ETEROVIC. Dos modelos arquitectónicos para el DSOA, Universidad de Extremadura (España) y pontificia Universidad católica (Chile). 2005.

Figura 13. Modelo arquitectónico para el diseño orientado a aspectos.



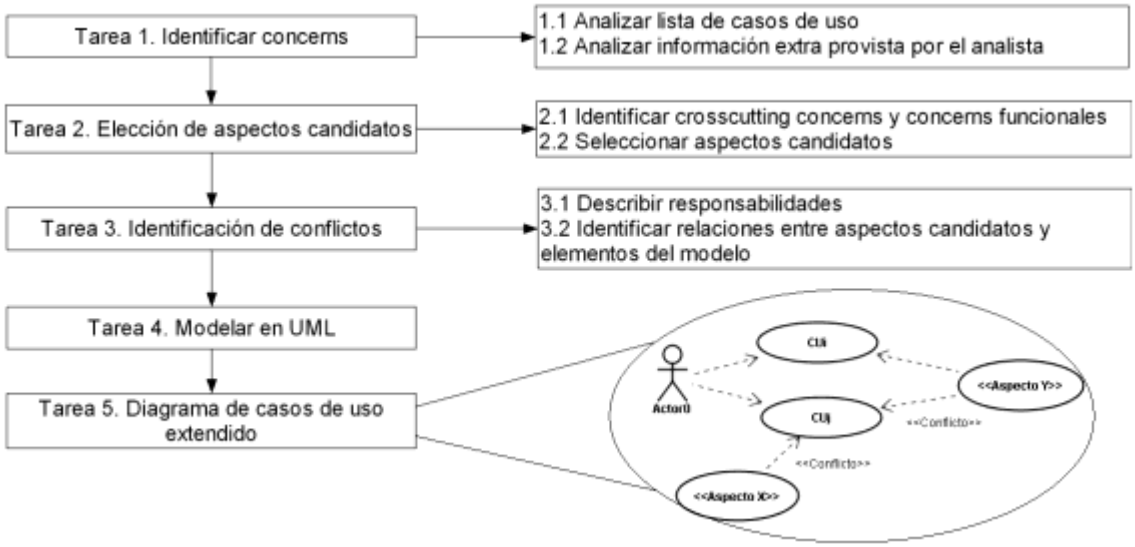
Fuente. NAVASA, A; PALMA, K; MURILLO, J.M y ETEROVIC. Dos modelos arquitectónicos para el DSOA, Universidad de Extremadura (España) y pontificia Universidad católica (Chile). 2005.

3.14.4 IDENTIFICACIÓN TEMPRANA DE ASPECTOS

Se presenta un enfoque automatizado para identificar aspectos en etapas tempranas del ciclo de desarrollo de software. Para ello se desarrolla una herramienta en el contexto de ingeniería de software que sirva de soporte para identificar los diferentes concerns del sistema. En etapas posteriores estos

concerns se convertirán en los elementos básicos y en los aspectos del sistema. De esta manera, se identifican los aspectos en etapas tempranas logrando la separación de concerns y reduciendo los costos de desarrollo, mantenimiento y evolución. Ver (figura 14)[35].

Figura 14. Identificación temprana de aspectos



Fuente. HAAK, Betina; DÍAZ, Miguel; MARCOS, Claudia y PRYOR, Jane. Identificación temprana de aspectos, ISISTAN Instituto de sistemas Tandil (Argentina). 2004.

[35] HAAK, Betina; DÍAZ, Miguel; MARCOS, Claudia y PRYOR, Jane. Identificación temprana de aspectos, ISISTAN Instituto de sistemas Tandil (Argentina). 2004.

4. DISEÑO METODOLÓGICO

El diseño metodológico que se pretende plantear, permite cubrir el ciclo de vida de la metodología RUP en el desarrollo del software. Cada fase tiene definidas un conjunto de objetivos y un punto de control específico, proceso que decide claramente cómo y qué debe hacerse en la aplicación a desarrollar, utilizando los lenguajes apropiados para tal fin. El modelado y desarrollo de aplicaciones orientadas a la Web desde una perspectiva de Ingeniería del Software Orientada a Aspectos permite que una aplicación sea construida describiendo cada dato separadamente con el fin de tener un código menos esparcido y enredado, mejorar el funcionamiento, rendimiento, facilidad, eficiencia, reusabilidad y tiempo de la aplicación. La metodología será explica con más detalle en la etapa 3: elaboración de la metodología, en donde se centra el ciclo de vida.

4.1 ETAPA 1. ESTUDIO

En esta etapa se hace la recopilación de información, fundamentos teóricos que permitan entender el concepto de Programación Orientada a Aspectos POA, Ingeniería de Software Orientada a Aspectos AOSD, aplicaciones, avances, características, herramientas y metodologías aplicadas a la Programación Orientada a Aspectos POA.

Actividades a desarrollar:

- Recopilación de información acerca de la Programación Orientada a Aspectos

- POA; definiciones de Aspectos por diferentes autores, características generales y sus aplicaciones.

- Búsqueda de herramientas, métodos, técnicas y lenguajes disponibles para el desarrollo de la Programación Orientada a Aspectos POA, que permitan usarse para desarrollar la aplicación orientadas a la Web y que sirvan como base para el desarrollo del proyecto.

- Estudio de las diferentes herramientas, métodos y técnicas en los que se apoya la Ingeniería de Software Orientada a Aspectos AOSD.

- Búsqueda de las principales características y objetivos, así como ventajas y desventajas que ofrece la Programación Orientada a Aspectos POA.

- Revisión de la documentación existente, con el fin de tener clara su funcionalidad e identificar el problema que se pretende solucionar y demás especificaciones que se han tenido en cuenta para la realización del proyecto.

4.2 ETAPA 2. ELABORACIÓN DE LA METODOLOGÍA

En esta etapa se plantea la metodología que se llevará a cabo en el ciclo de vida RUP en sus fases de inicio, elaboración, construcción y transición, teniendo en cuenta dentro de ellas sus iteraciones como son, análisis, diseño, implementación y pruebas, posteriormente llevar a cabo su aplicabilidad en el desarrollo de la aplicación orientada a la Web con aspectos, tomando como base una aplicación orientada a objetos existente. Básicamente la metodología a plantear se enfoca en el análisis y diseño puesto que la programación orientada a Aspectos POA se centra en el desarrollo. Diseñar un sistema basado en aspectos requiere entender

qué se debe incluir en el lenguaje base, qué se debe incluir dentro de los lenguajes de aspectos y qué debe compartirse entre ambos lenguajes.

Actividades a desarrollar:

4.2.1 Fase 1. inicio. Para esta fase se recopilará los requerimientos necesarios para el desarrollo de la metodología Orientada a Aspectos. Se utiliza el ciclo de vida RUP.

4.2.1.1 Análisis. en esta etapa se analizará metodologías ya planteadas para la Programación Orientada a Aspectos POA, tomándolas como referencias para el diseño de la metodología que se llevará a cabo en este proyecto.

4.2.2 Fase 2. elaboración. En esta fase, se aplicará la información recopilada y analizada en la fase anterior, tomando la información relevante para la elaboración de la metodología.

4.2.2.1 Análisis. en esta fase se revisará la información recopilada y estudiada en la fase anterior para la metodología previamente analizada.

4.2.2.2 Diseño. en esta fase se diseñará las capas del diseño para la metodología y se define claramente el contenido de cada una de ellas.

4.2.3 Fase 3. construcción.

4.2.3.1 Análisis. mejorar el desarrollo y contenido de cada capa del diseño para la metodología.

4.2.3.2 Diseño. se diseñará la metodología de acuerdo al análisis establecido en las fases anteriores.

4.2.3.3 Implementación. se desea que el diseño metodológico, sean de gran ayuda para la Programación Orientada a Aspectos.

4.3 ETAPA 3. APLICACIÓN.

En esta etapa se aplicará la metodología planteada en la aplicación a desarrollar (Simulador de Cajero Automático), descrita anteriormente en la fase elaboración de la metodología, se hará un análisis de los resultados obtenidos, comprobando que se lleve acabo el buen funcionamiento de la aplicación.

Actividades a desarrollar:

- Aplicar la metodología planteada en la aplicación a desarrollar y analizar los resultados.

- Determinar y estudiar uno o varios lenguajes de aspectos, con el fin de especificar su comportamiento.

- Determinar y estudiar un lenguaje para identificar la funcionalidad básica de los componentes de la aplicación.

- Determinar y estudiar el tejedor de aspectos, encargado de combinar los lenguajes.

- Implementación del diseño básico

4.4 ETAPA 4. RESULTADOS.

En esta fase se pretende mostrar las diferencias existentes en un modelo desarrollado en Programación Orientada a Aspectos POA y Programación Orientada a Objetos POO, así como las ventajas que tiene el aplicar la Programación Orientada a Aspectos POA en cualquier aplicación Web teniendo el modelo construido en su versión final y evaluar los resultados finales.

Actividades a desarrollar:

- Integración de los aspectos con la funcionalidad básica del sistema.

- Realizar la documentación de cada uno de los resultados ejecutados en la aplicación.

- Análisis de las diferencias existentes en la Programación Orientada a Objetos POO y la Programación Orientada a Aspectos POA, de acuerdo a la aplicación desarrollada.

- Análisis de las ventajas y las desventajas de la Programación Orientada a Aspectos POA, de acuerdo a la aplicación desarrollada.

- Manejo de los Cambios y verificación de resultados

5. METODOLOGÍA PLANTEADA

Se ejecutará, la etapa 2 del diseño metodológico, aplicando la metodología Rup para esta etapa, después de haber pasado por la investigación de las metodologías que han nacido con este paradigma.

5.1 RUP PARA METODOLOGÍA PLANTEADA

Tabla 3. Metodología RUP en la metodología planteada

	Inicio	**Elaboración**	**Construcción**	**Transición**
Etapa 1				
Etapa 2				
Etapa 3				
Etapa 4				
Etapa 5				
Etapa 6				
Etapa 7				
Etapa 8				
Etapa 9				
Etapa 10				
Etapa 11				

Fuente. Autoras del proyecto

Al igual que en todo el proyecto se aplica la metodología RUP, puesto que en la fase de elaboración se crea la metodología a utilizar en el sistema, en la fase de inicio se lleva a cabo la etapa 1 (especificar el diagrama de casos de uso sin aspectos) y la etapa 2 (crear modelo de diseño). En la fase de elaboración la etapa 3 (identificar concerns), etapa 4 (elección de aspectos candidatos), etapa 5 (especificar aspectos), etapa 6 (identificar conflictos), etapa 7 (modelar en UML), etapa 8 (Diagramas de secuencia con aspectos). En la fase de construcción la etapa 9 (expresar el sistema con AspectJ) y la etapa 10 (Generar prototipo para el sistema). Y en la fase de transición la etapa 11 (Ejecutar el prototipo), de este modo se aplica la metodología RUP. Ver (tabla 3).

5.2 DIAGRAMA DE LA METODOLOGÍA PLANTEADA

Con el análisis y estudio de las metodologías anteriormente mencionadas y los conceptos adquiridos de la programación orientada a aspectos se plantea para este proyecto una metodología que consta de 11 etapas que muestran de una forma clara el proceso de llevar a cabo una aplicación orientada a objetos a una aplicación orientada a aspectos. Ver (figura 15).

5.2.1 Etapa 1. especificar el sistema con diagramas de caso de uso.

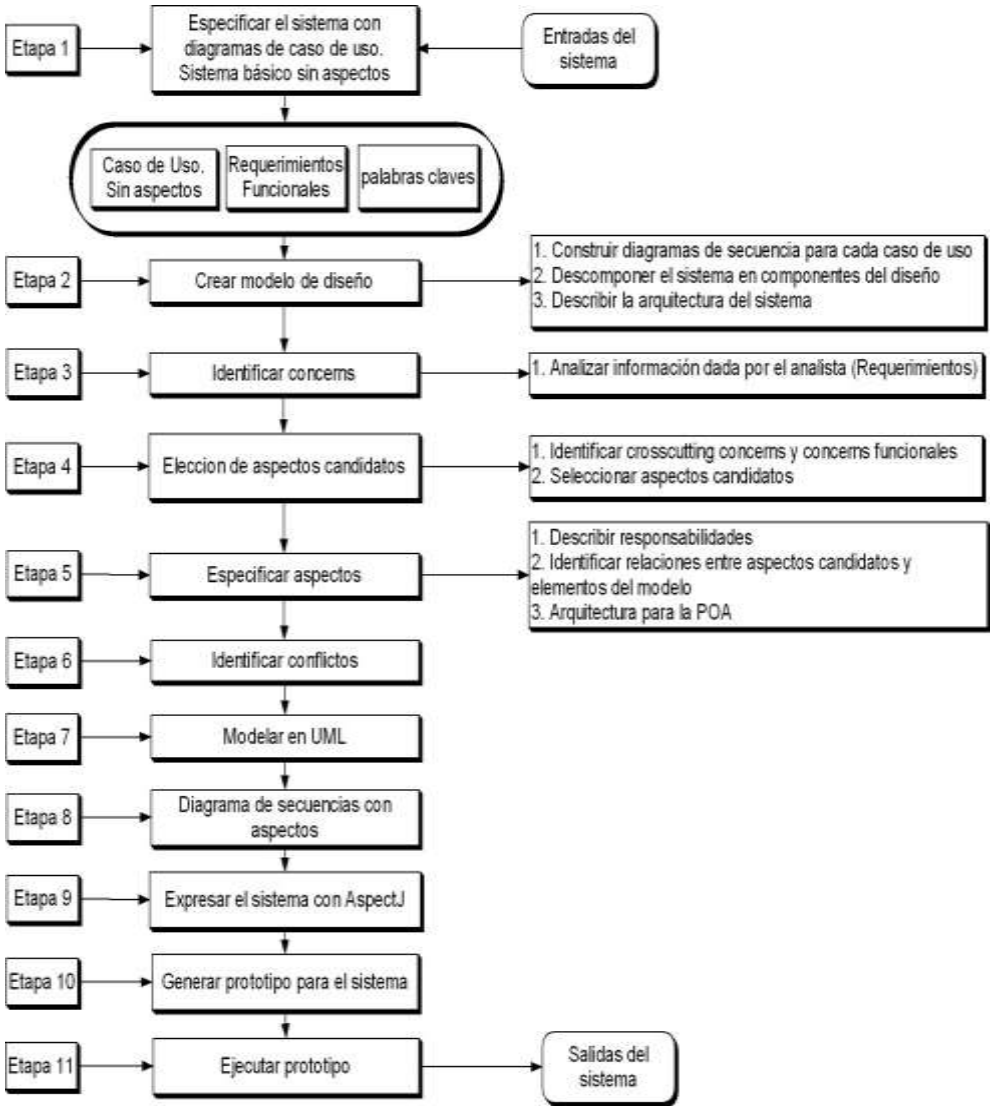
En esta etapa se modela orientado a objetos y se toman los requerimientos dados por analista del sistema.

5.2.2 Etapa 2: crear un modelo de diseño.

En esta etapa se propone hacer una descripción inicial detallada de acuerdo con la especificación del sistema sin considerar aspectos, creando el modelo de diseño correspondiente, construir los

diagramas de secuencia para cada caso de uso, descomponer el sistema en componentes de diseño y describir una arquitectura del sistema orientado a objetos.

Figura 15. Metodología ajustada por las autoras para la programación orientada a aspectos.



Fuente. Tomado de [36] y [37], y Modificado por Autoras del proyecto.

[36] NAVASA. Op. cit., referencia [34]

5.2.3 Etapa 3. identificar concerns. Esta etapa consiste en identificar a partir de los requerimientos dados por el analista, los posibles concerns de un sistema. Para especificar estos requerimientos existen diferentes tipos de técnicas; en este caso se ha elegido para este enfoque la utilización de casos de uso y requerimientos dados por el analista se genera automáticamente información de interés que se tomarán como posibles candidatos. Esto se hace a partir de la extracción de verbos, excluyendo por ejemplo aquellos que no representen una acción relevante tales como *es*, *debe*, *estar*; solo se deben seleccionar aquellos verbos que aparezcan en más de un requerimiento.

5.2.4 Etapa 4. elección de aspectos candidatos. Esta etapa se subdivide en dos subpasos: identificar crosscutting concerns y concerns funcionales, y elección de aspectos candidatos por parte del analista.

El primero se identifica cuando un concern corta transversalmente más de un elemento del modelo (casos de uso o requerimientos), se deberán eliminar aquellos que se consideren irrelevantes y en el segundo subpaso el analista debe seleccionar una lista de concerns utilizando su experiencia e intuición. Teniendo la lista de temas potenciales (acciones, entidades, característica y verbos), se definen los temas bases y aspectos.

5.2.5 Etapa 5. especificar aspectos candidatos. Esta etapa esta dividida en 3 subpasos: identificar responsabilidades, identificar las relaciones entre aspectos para que los conflictos puedan ser detectados y definir la arquitectura POA.
En el primero se debe describir el objetivo y la funcionalidad del aspecto candidato. Se realiza simplemente adjuntando una breve descripción a cada

[37] HAAK, Betina; DÍAZ, Miguel; MARCOS, Claudia y PRYOR, Jane. Identificación temprana de aspectos, ISISTAN Instituto de sistemas Tandil (Argentina). 2004.

aspecto candidato. En el segundo una relación entre aspectos y elementos del modelo define la forma en que un aspecto afecta a un elemento, para la definición de la arquitectura orientada a aspectos, se propone definir una arquitectura con 3 niveles, descrita en la sección 7.3.5.3 en la definición de la arquitectura orientada a aspectos.

5.2.6 Etapa 6. identificar conflictos. En este paso se identifican todas las posibles situaciones conflictivas entre concerns. Si más de un aspecto es aplicado sobre un mismo requerimiento, surgirá una situación conflictiva.

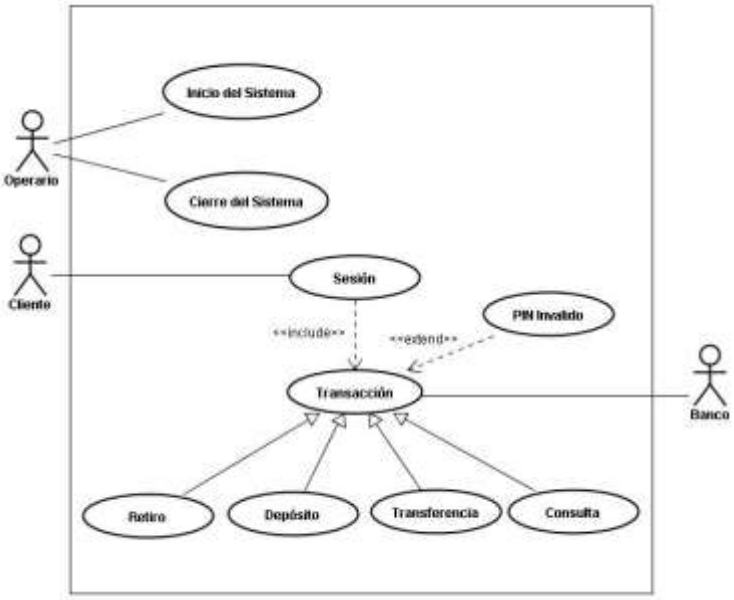
5.2.7 Etapa 7. modelar en UML. Basados en la información obtenida en la realización de los pasos anteriores, se construirá un modelo visual, el cual será una extensión de un diagrama de casos de uso con de los aspectos finalmente seleccionados.

5.3 DESARROLLO DE LA METODOLOGÍA PARA EL CASO DE ESTUDIO: SIMULADOR DE UN CAJERO AUTOMÁTICO ATM

5.3.1 Etapa 1. Especificar el sistema con diagrama de casos de uso

5.3.1.1 Caso de Uso. Sin aspectos. Se ha tomado un programa ya realizado orientado a objetos, de este se tomó los requerimientos, casos de uso y diagramas de secuencia para empezar con la metodología planteada. En los anexos se encuentran todo el diseño de estre programa base.

Figura 16. Caso de Uso de la simulación del cajero automático.



Fuente. Mathematics and Computer Science: Object-Oriented Software Development, [online], 2000. Disponible en Internet:
< http://www.math-cs.gordon.edu/courses/cs211/index.html>

5.3.1.2 Requerimientos funcionales con temas. Se identificaron las características potenciales del sistema ATM. Las características o temas potenciales, son las que se encuentran en color rojo.

R1: El cajero ATM *ATENDERÁ* a un cliente a la vez. Un cliente deberá *INGRESAR* la *TARJETA* de ATM e *INGRESAR* el número de identificación personal *(PIN),* no se deben observar los dígitos que ingresa, ambos serán enviados al banco para la validación como parte de cada *TRANSACCIÓN*. Entonces el cliente podrá *REALIZAR* una o más *TRANSACCIONES.*

TEMAS: *ATENDERÁ, INGRESAR TARJETA, PIN, TRANSACCIÓN, REALIZAR.*

R2: Un cliente debe poder hacer un *RETIRO* de *DINERO* en efectivo de cualquier *CUENTA* a la que pertenece la *TARJETA* en múltiplos de 20.000. La *TRANSACCIÓN* debe ser *APROBADA* por el banco antes de que el RETIRO se haga efectivo.

TEMAS: *RETIRO, DINERO, CUENTA, TARJETA, TRANSACCIÓN, APROBADA*.

R3: Un cliente debe poder hacer un *DEPÓSITO* a cualquier *CUENTA* a la que pertenece la *TARJETA*, en *DINERO* efectivo. El cliente *INGRESARÁ* la cantidad del depósito en el ATM, La *TRANSACCIÓN* debe ser *APROBADA* por el banco después de que el depósito haya sido *VERIFICADO* a través de un segundo mensaje. Si el cliente falla al *REALIZAR* el depósito dentro de un límite de tiempo, o en vez de eso presiona *CANCELAR*, el segundo mensaje no será enviado al banco y el depósito no será acreditado al cliente.
TEMAS: *DEPÓSITO, CUENTA, TARJETA, DINERO, INGRESARÁ, TRANSACCIÓN, APROBADA, VERIFICADO, REALIZAR, CANCELAR*

R4: Un cliente debe poder hacer una *TRANSFERENCIA* de dinero entre dos *CUENTAS* pertenecientes a la *TARJETA*.

TEMAS: *TRANSFERENCIA, CUENTAS, TARJETA.*

R5: Un cliente debe poder hacer una *CONSULTA* del *SALDO* de cualquier *CUENTA* perteneciente a la *TARJETA*.

TEMAS: *CONSULTA, SALDO, CUENTA, TARJETA*.

R6: Un cliente debe poder *CANCELAR* una *TRANSACCIÓN* en progreso oprimiendo la tecla cancelar en lugar de responder a un pedido de la máquina, se *RETORNA* la *TARJETA* al usuario y la máquina queda en estado disponible.

TEMAS: *CANCELAR, TRANSACCIÓN, RETORNA, TARJETA*

R7: La *TARJETA* será permanentemente *RETENIDA* por la máquina, si el cliente *INGRESA* el *PIN* incorrectamente cuatro veces.

TEMAS: *TARJETA, RETENIDA, INGRESA, PIN*

R8: El ATM proporcionará al cliente un recibo impreso para cada *TRANSACCIÓN* exitosa, mostrando la fecha, la hora, la ubicación de la máquina, el tipo de la transacción, la cuenta (cuentas), la cantidad, y el saldo disponible de la *CUENTA* afectada (justificar las transferencias)

TEMAS: *TRANSACCIÓN, CUENTA*

R9: El ATM mantendrá también un *REGISTRO* interno de las *TRANSACCIONES* para facilitar la solución de problemas que se puedan presentar en medio de una transacción. Las entradas serán hechas en el registro desde que el ATM es prendido y hasta que se apaga, para cada mensaje enviado al Banco, para *ENTREGAR* el *DINERO* en efectivo, y para recibir los *DEPÓSITOS*. Las entradas del registro pueden contener los números de tarjeta y cantidades de dinero, pero para la seguridad nunca contendrá el PIN.

TEMAS: *REGISTRO, TRANSACCIONES, ENTREGAR, DINERO, DEPÓSITOS.*

5.3.2 Etapa 2. Modelo de diseño

5.3.2.1 Diagramas de secuencia. Los diagramas para el Sistema ATM. Se describe como diagramas de secuencia (Inicio del sistema, cierre del sistema,

sesión y transacción), debido a que transacción es una generalización abstracta, los casos de uso (Retiro, depósito, transferencia, consulta y PIN inválido) son descritos en diagramas de colaboración. (Ver Anexo A)

5.3.2.2 Descomponer el sistema en componentes de diseño. Para la descomposición del sistema, lo que se hace es estudiar el diagrama de clases, y mirar cada clase con sus atributos y métodos. (Ver Anexo B)

5.3.2.3 Arquitectura del sistema Orientada a Objetos. La arquitectura del sistema del cajero automático ATMUNAB, consta de 3 niveles, siendo una de las más comunes en los sistemas de información, y además de tener una interfaz de usuario contempla la persistencia de los datos. Ver (figura 17).

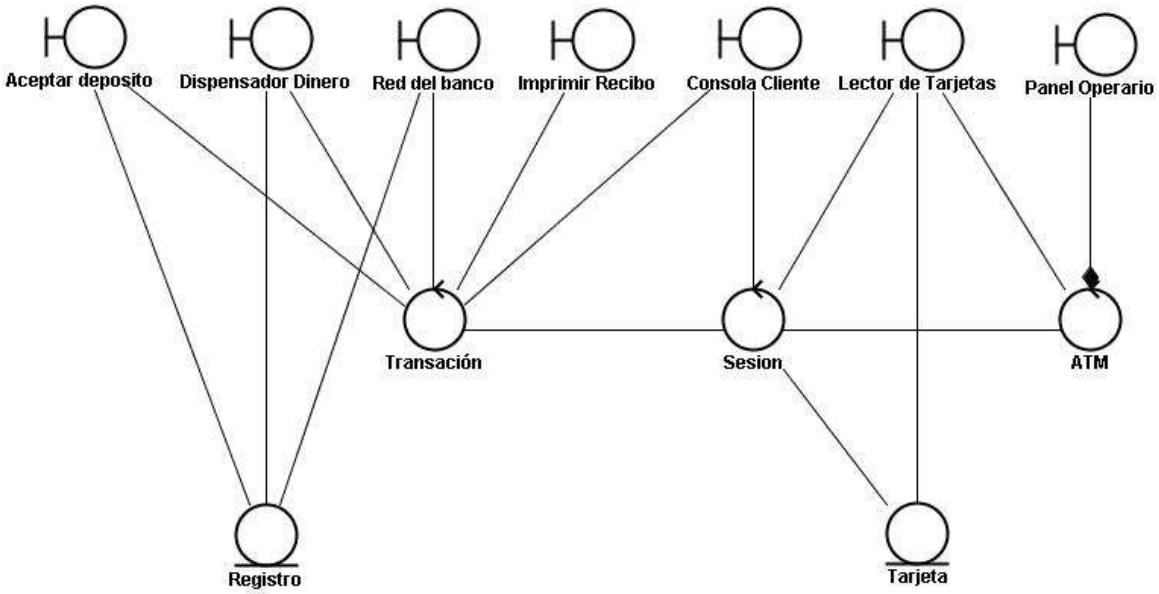
- Nivel 1: presentación – interfaces, informes, entre otros.

- Nivel 2: lógica de la Aplicación – controles, tareas y reglas que gobiernan el proceso.

- Nivel 3: almacenamiento – mecanismo de almacenamiento o entidades.

Vista de una forma metodológica, la arquitectura consta de cuatro vistas grupadas en dos partes: estructura lógica (vista de diseño, vista de procesos) y estructura física (Vista de implementación, Vista de despliegue) y la vista de casos de uso que son el comportamiento del sistema como se muestra en la figura 17.

Figura 17. Arquitectura del sistema Orientada a Objetos

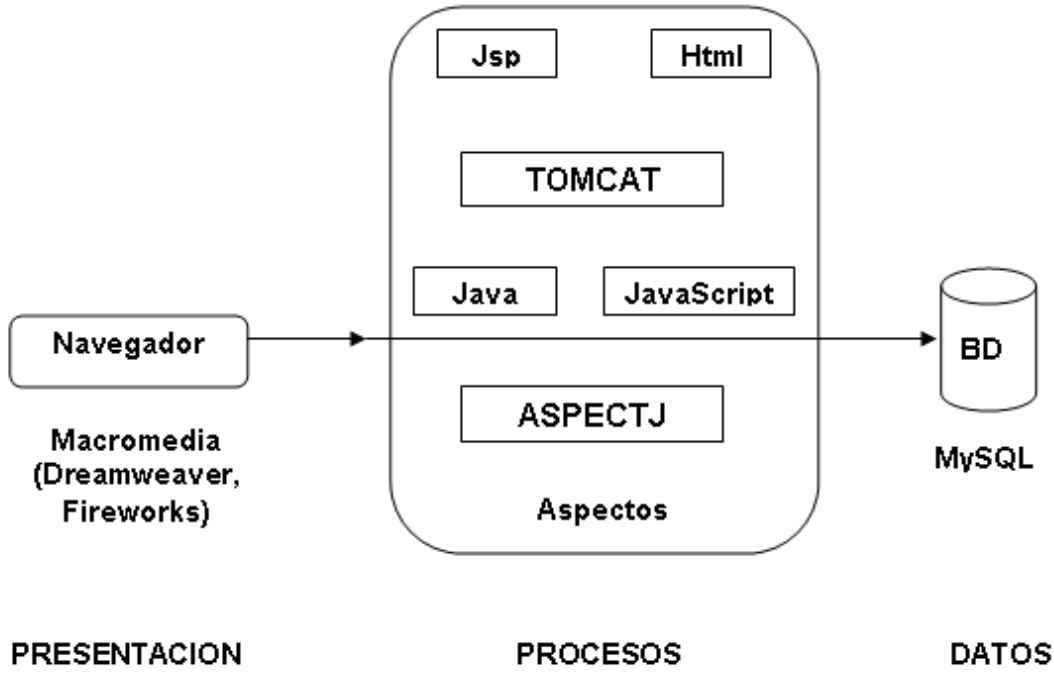


Fuente., Ibid.

La estructura lógica traduce los escenarios de uso creados en el diseño conceptual en un conjunto de objetos de negocio y sus servicios. El diseño lógico se convierte en parte en la especificación funcional que se usa en el diseño físico. El diseño lógico es independiente de la tecnología. El diseño lógico refina, organiza y detalla la solución de negocios y define formalmente las reglas y políticas específicas de negocios.

En el diseño físico se debe cuidar el nivel de granularidad (un componente puede ser tan grande o tan pequeño según su funcionalidad, es decir, del tamaño tal que pueda proveer de una funcionalidad compleja pero de control genérico) y la agregación y contención (un componente puede rehusar utilizando técnicas de agregación y contención, sin duplicar código).

Figura 18. Arquitectura para el simulador de un cajero automático ATM



Fuente. Autoras del proyecto

- Vista de casos de uso: muestra los requisitos del sistema tal como es percibido por los usuarios finales, analistas, programadores y encargados de pruebas. Se utiliza el diagrama de casos de uso, de estados y actividades.

- Vista de diseño: captura los atributos y métodos del problema para llevarlos a la solución. Se utilizan los diagramas de clases, estados y actividades.

- Vista de procesos: modela la distribución de los procesos. Se utiliza el diagrama de colaboración. Emplea el mismo diagrama de la vista de diseño, poniendo especial atención en las clases de procesos. Se utiliza el diagrama general de clases.

- Vista de implementación: modela los componentes y archivos que se utilizan para ensamblar y hacer disponible el sistema físico. Se utilizan los diagramas de Componentes, Estados y Actividades.

- Vista de despliegue: modela la parte física sobre el cual se ejecuta el sistema. Se utilizan los diagramas de estados y actividades.

5.3.3 Etapa 3. Identificar concerns. Se analizan los requerimientos dados por el analista.

5.3.3.1 Analizar requerimientos. Información extra prevista por el analista. Luego de estudiar los requerimientos del sistema de ATM, se identificaron los siguientes temas y entidades potenciales, después de buscar las características, acciones y verbos que son de suma importancia en el sistema.

Los temas potenciales son aquellos que representan acciones (algunos verbos), se identificaron los siguientes 9 temas potenciales:

- Transacción

- Ingresar (cargar cajero)

- Depositar

- Retirar

- Transferencia

- Consultar
- Tarjeta
- PIN
- Registro (log)
- Cancelar
- Verificar
- Dinero

También se identificaron 11 entidades, las entidades son aquellas que identifican el sistema, las identidades son las siguientes:

- Entregar
- Retornar
- Retener
- Aprobar
- Atender
- Realizar

- Cuenta

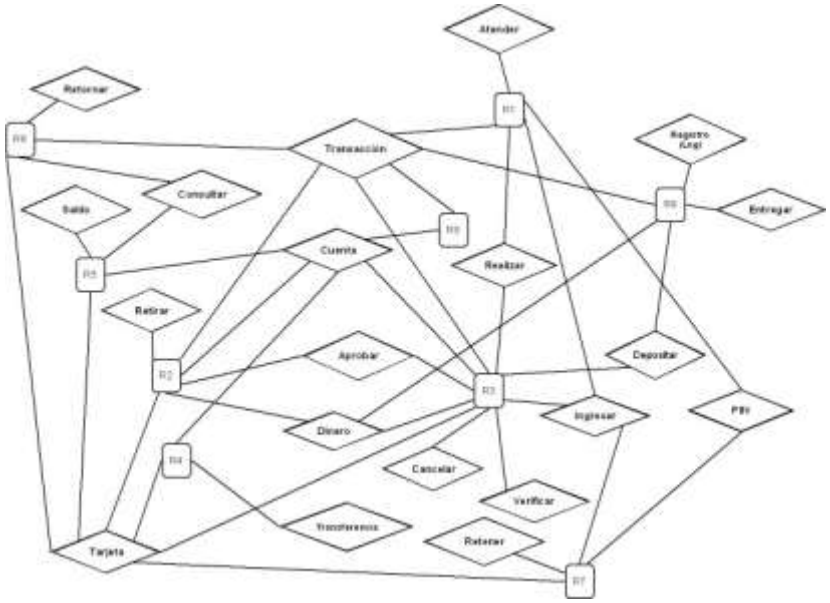
- Saldo

El propósito de escoger los temas y las entidades potenciales son de restringir una lista eventual de características y objetos para diseñar. Por supuesto, esto es una descripción algo simplista de cómo escoger esos elementos.

5.3.4 Etapa 4. Elección de aspectos candidatos. El siguiente paso a seguir después de identificar el conjunto de características, se itera decidiendo si se agrega, borra, separa o agrupa los temas anteriormente mencionados. Como en Orientado a Objetos, donde se utiliza algunas de las entidades para motivar las clases, en orientado a aspectos se utilizan algunas de las características para motivar los temas. Existen varias maneras de llegar al punto de partida de los temas. Se puede escoger los nombres de características, de los servicios, o de los casos del uso de su sistema si se sabe cuales podrían ser. Los investigadores de EES, no utilizan los casos de uso y no analizan los requerimientos en términos de características ni servicios. En lugar de eso, miran los requerimientos para identificar la funcionalidad del sistema.

5.3.4.1 Identificar crosscutting concerns y concerns funcionales. En este paso se refinan los aspectos, como se muestra en la figura 19.

Figura 19. Theme-relationship view inicial para el sistema ATM.



Fuente. Autoras del proyecto

5.3.4.2 Seleccionar aspectos candidatos. En esta etapa, lo que se quiere tomar en la vista anterior (Figura 15) es qué rol toma el comportamiento en los requerimientos, para determinar si ellos deben ser los temas o apenas una conducta (quizás métodos) dentro de temas. Escudriñamos los temas potenciales y preguntamos si tiene sentido para tener cada uno de ellos como una característica en nuestro sistema. Si ellos no son significativos, estos se quitarán de la lista.

Luego de analizar, y siguiendo con los pasos de la Metodología Theme Approach y teniendo en cuenta las reglas para la separación de aspectos candidatos, se ha determinado quitar algunos elementos de la lista de temas potenciales, estos temas quedan separados por temas bases y aspectos.

Los siguientes son los temas bases:

- Atender
- Realizar
- Cancelar
- Verificar
- Retornar
- Retener.
- Entregar.
- Dinero
- Saldo.
- Aprobar
- Cuenta

5.3.5 Etapa 5. Especificar aspectos candidatos

- Atender, está relacionado con un sólo requerimiento, se piensa que es algo secundario en el sistema.

- Realizar, está relacionado a dos requerimientos, no son primordiales para el sistema.

- Cancelar, sólo está relacionado a un requerimiento, esta acción no es tan relevante en el sistema, por lo tanto no es un aspecto.

- Verificar, esta característica la encontramos en un sólo requerimiento, lo que hace es verificar si el depósito ha sido realizado, esta característica puede ser agrupada con el de aprobar.

- Retornar, sólo está relacionado con un requerimiento y no es de suma importancia en el sistema.

- Retener, esta característica es relevante al sistema, por lo tanto no lo dejamos como aspectos.

- Entregar, es una acción, que está agrupada con algunas acciones anteriormente nombradas.

- Dinero, el dinero no es una característica que esté ligada a la funcionalidad del sistema, por lo tanto, no sería un aspecto.

- Saldo, hace parte más de la gramática de los requerimientos y no se hace tan necesario tenerlo como un aspecto.

- Cuenta, la cuenta se valida en otras clases por eso no se toma como un aspecto.

5.3.5.1 Describir responsabilidades. Los siguientes elementos, son los aspectos del sistema, representados en una maraña menos compleja (ver figura 20) que como se mostraba en la figura 15.

- Transacción, esta característica es la base del sistema, proporciona la funcionalidad del sistema ATMUNAB, esta clase agrupa las clases de depositar, retirar, consultar y transferir.

- Ingresar, esta característica es importante para el sistema, pues es la forma más cercana con la cual el cliente interactúa con el ATMUNAB.

- Depositar, sólo está relacionado con dos requerimientos, pero el depósito es algo primordial para el sistema ATMUNAB.

- Retirar, está ligado a un solo requerimiento, pero es ficha clave para el sistema ATMUNAB.

- Transferencia, sólo está relacionado a un requerimiento, esta característica representa una acción del sistema ATMUNAB, por lo tanto, es un tema.

- Consultar, está ligada a dos requerimientos y hace parte de la funcionalidad del sistema, por lo tanto, es un tema.

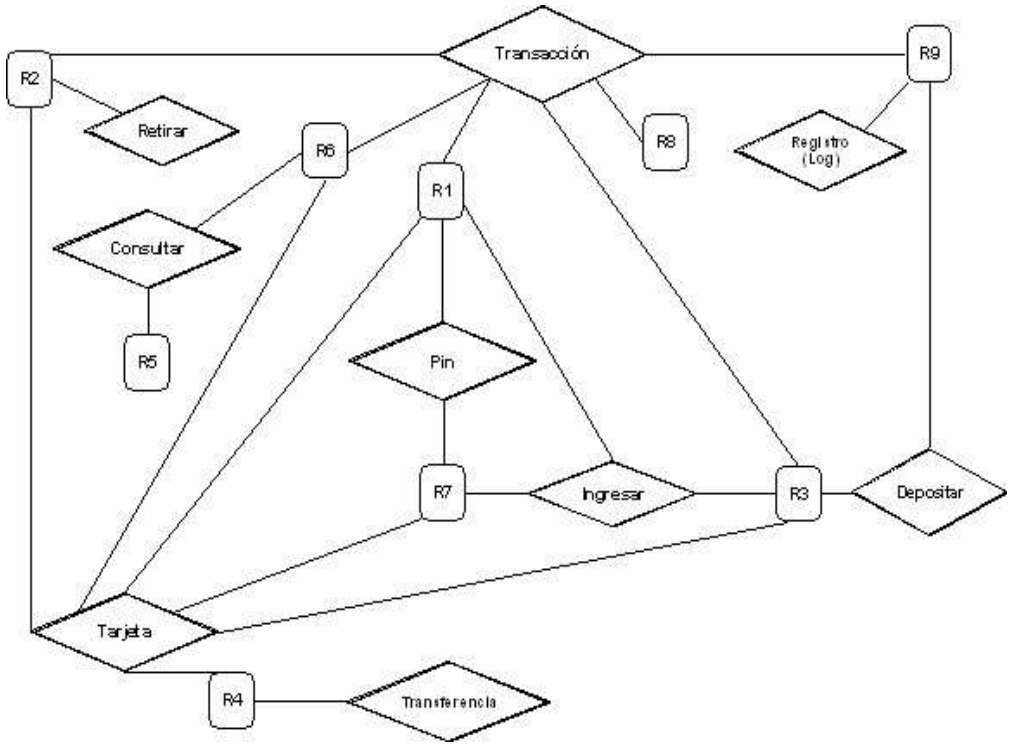
- Registro (log), es un módulo donde guarda la información de todas las transacciones, por lo tanto, se toma como un tema, por ser tan importante en la funcionalidad del sistema.

- Aprobar, es importante para el sistema, pues el banco debe aprobar cada una de las transacciones que se realizan.

- Tarjeta, esta ligada a 7 requisitos la tarjeta es relevante para el sistema, pues esta ligada a muchos de los requerimientos importantes del sistema.

- PIN, es algo importante en el sistema, pues el sistema debe validar el PIN para realizar las demás funcionalidades del sistema.

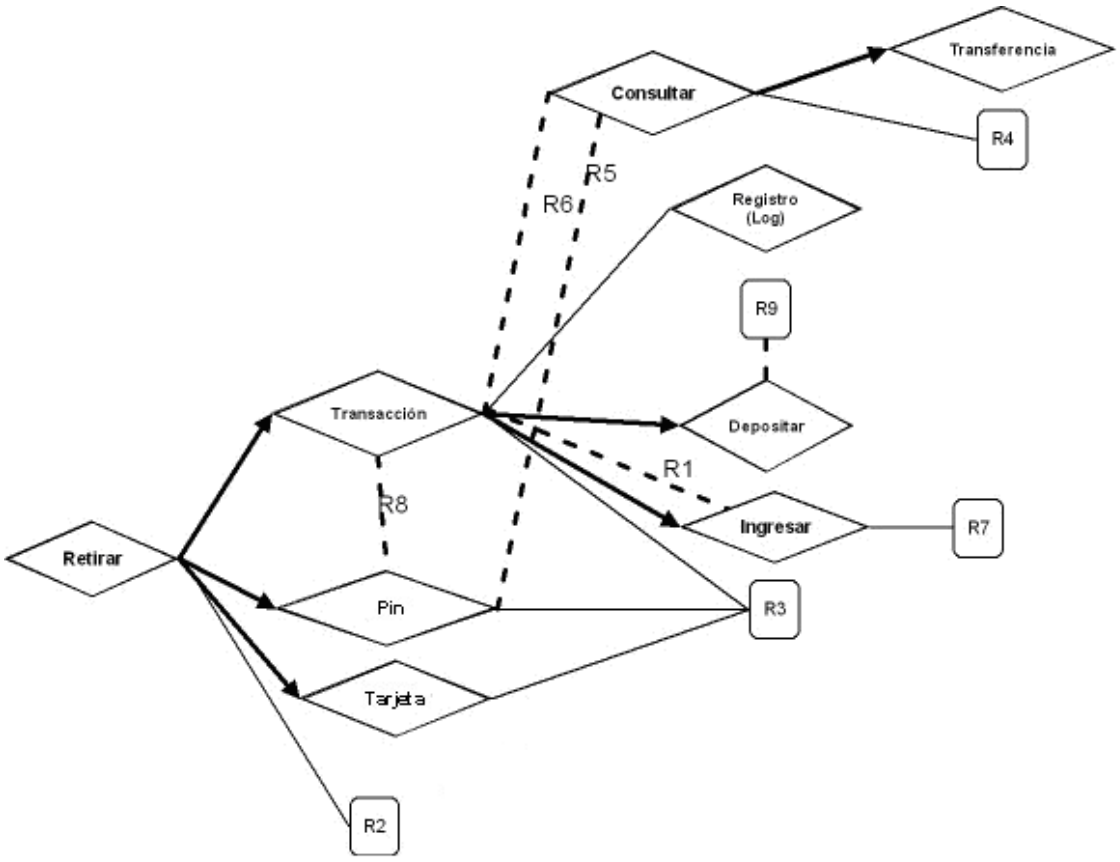
Figura 20. Theme-relationship view con los aspectos del sistema ATM.



Fuente. Autoras del proyecto

Luego de tener la vista de la relación de los temas de todos los aspectos para el sistema, el siguiente paso es hacer el diagrama de vistas donde se vea más claro el crosscutting, en el sistema ATM. Ver (figura 21).

Figura 21. Evolución del crosscutting-relationship view.



Fuente. Autoras del proyecto

5.3.5.2 Identificar relaciones entre aspectos candidatos y elementos del modelo. El sistema observará la columna *Requerimientos* de la Tabla 4 para determinar con cuáles requerimientos está relacionado un determinado aspecto.

Tabla 4: Relación de los Aspectos y requerimientos

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**
Retirar		X							
Tarjeta	X	X	X	X	X	X	X		

Pin	X						X		
Consultar					X				
Transferir				X					
Depositar			X						**X**
Registro									**X**
Ingresar	X		X				X		
Transacción	**X**	**X**	**X**			**X**		**X**	**X**

Fuente. Autoras del proyecto

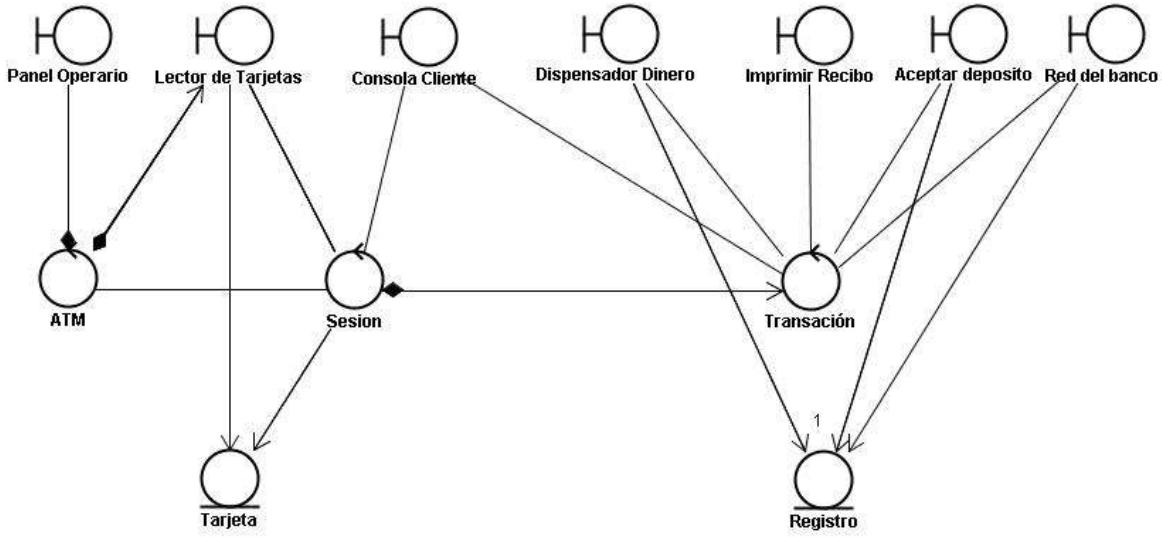
5.3.5.3 Arquitectura POA. La arquitectura del sistema del cajero automático ATMUNAB con aspectos, consta al igual que la Orientada a Objetos de 3 niveles, siendo una de las más comunes en los sistemas de información, y además de tener una interfaz de usuario contempla la persistencia de los datos. Ver (figura 22).

- Nivel 1: presentación – interfaces, informes, entre otros.

- Nivel 2: lógica de la Aplicación – controles, tareas y reglas que gobiernan el proceso.

- Nivel 3: almacenamiento – mecanismo de almacenamiento o entidades.

Figura 22. Arquitectura POA



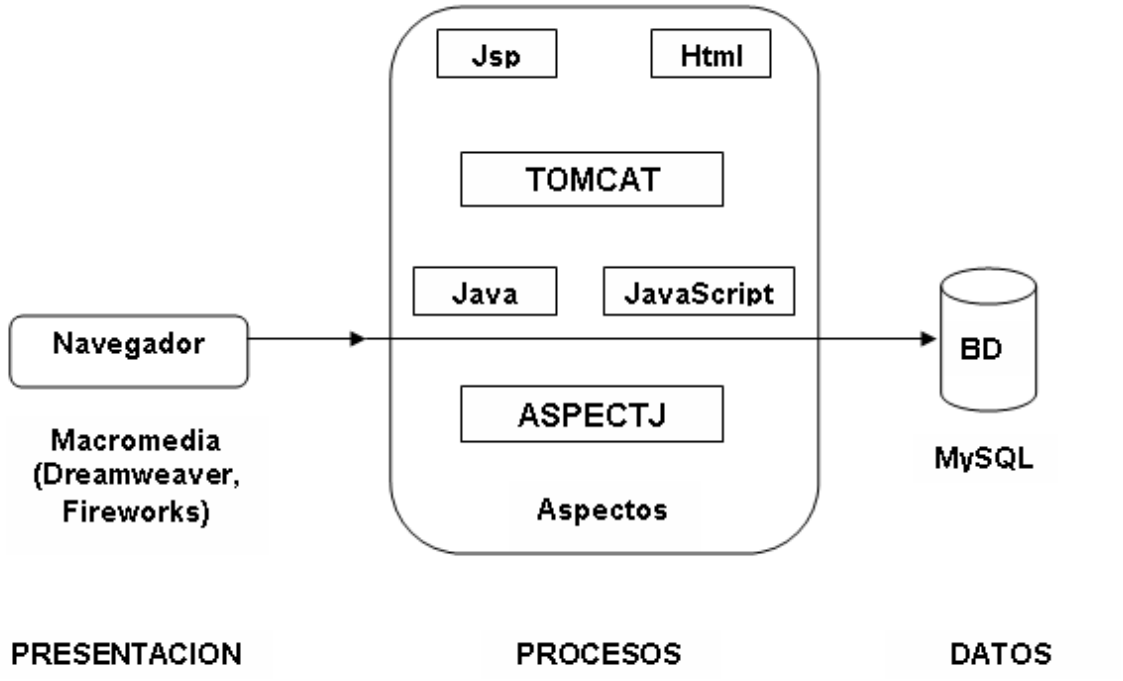
Fuente. Autoras del proyecto

El Simulador del Cajero Automático ATMUNAB está estructurado en base a la arquitectura tres capas. La (figura 19) muestra el modelo tres capas para el Simulador del Cajero Automático ATMUNAB. En la primera capa se presenta la visualización del sistema, esto se hace a través de un navegador. En la segunda capa se encuentra el servidor del sistema denominado tomcat en cual se puede hacer uso de tecnologías tales como: jsp y html; y los aspectos de software programados a través de Aspectj, que es una herramienta de JAVA especial para el manejo de aspectos. En la tercera y última capa se encuentra la base de datos del sistema gestionada a través de MySQL.

El primer nivel sobre el cual se realizará el proceso de inspección y pruebas es la de datos en donde se encuentra inmersa la base de datos del sistema, posteriormente se dará paso al nivel de proceso o del negocio donde se evaluarán los aspectos y por último se pasará al nivel de presentación donde se valorará el

sistema de acuerdo a los requerimientos del sistema. Todo este proceso se lleva a cabo teniendo en cuenta el ciclo de vida RUP.

Figura 23. Arquitectura tres capas para el Simulador del Cajero Automático ATMUNAB.



Fuente. Autoras del proyecto

5.3.6 Etapa 6. Identificar conflictos. Como se muestra en la tabla 4 se detectan todas las situaciones conflictivas existentes.

Se detecta una posible situación conflictiva entre los aspectos:

- Tarjeta, pin, ingresar, transacción con el R1

- Tarjeta, retirar, transacción con el R2
- Tarjeta, depositar, ingresar, transacción con el R3
- Tarjeta y Transferir con el R4
- Tarjeta y Consultar con el R5
- Tarjeta y Transacción con el R6
- Tarjeta, pin, ingresar con el R7
- Transacción con el R8
- Depositar, Registro y Transacción con el R9

Estos conflictos se presentan ya que la suma de cruces de la columna R1, R2, R3, R4, R5, R6, R7, R8 Y R9 es mayor que 1, sin embargo se puede ver que el aspecto transacción no presenta conflictos con ningún otro aspecto, ya que este pertenece solamente al R7.

5.3.7 Etapa 7. Modelar UML. Para modelar los requerimientos y los aspectos identificados, utilizamos diagramas de secuencia de UML al cual se le agregan nuevos elementos mediante la construcción de un profile para aspectos. En este enfoque solamente se marca la existencia de conflictos (<<Conflicto>>) y luego serán tratarlos de acuerdo a la conveniencia en cada uno de los casos. Según la Tabla 4. se pueden observar ocho posibles conflictos. Por ejemplo, el R1 *"El cajero ATMUNAB atenderá a un cliente a la vez. Un cliente deberá ingresar la tarjeta de ATMUNAB e ingresar el número de identificación personal (PIN), no se*

deben observar los dígitos que ingresa, ambos serán enviados al banco para la validación como parte de cada transacción. Entonces el cliente podrá realizar una o más transacciones", se ve afectado por los aspectos funcionales Retirar, Ingresar y Transacción. (Ver Anexo C)

5.3.8 Etapa 8. Diagramas de Secuencia extendido con aspectos. Luego de analizar los requerimientos, se llega a los diagramas de secuencia y el diagrama de clases que contiene tanto las clases sin aspectos como las clases con aspectos (Ver Anexo C)

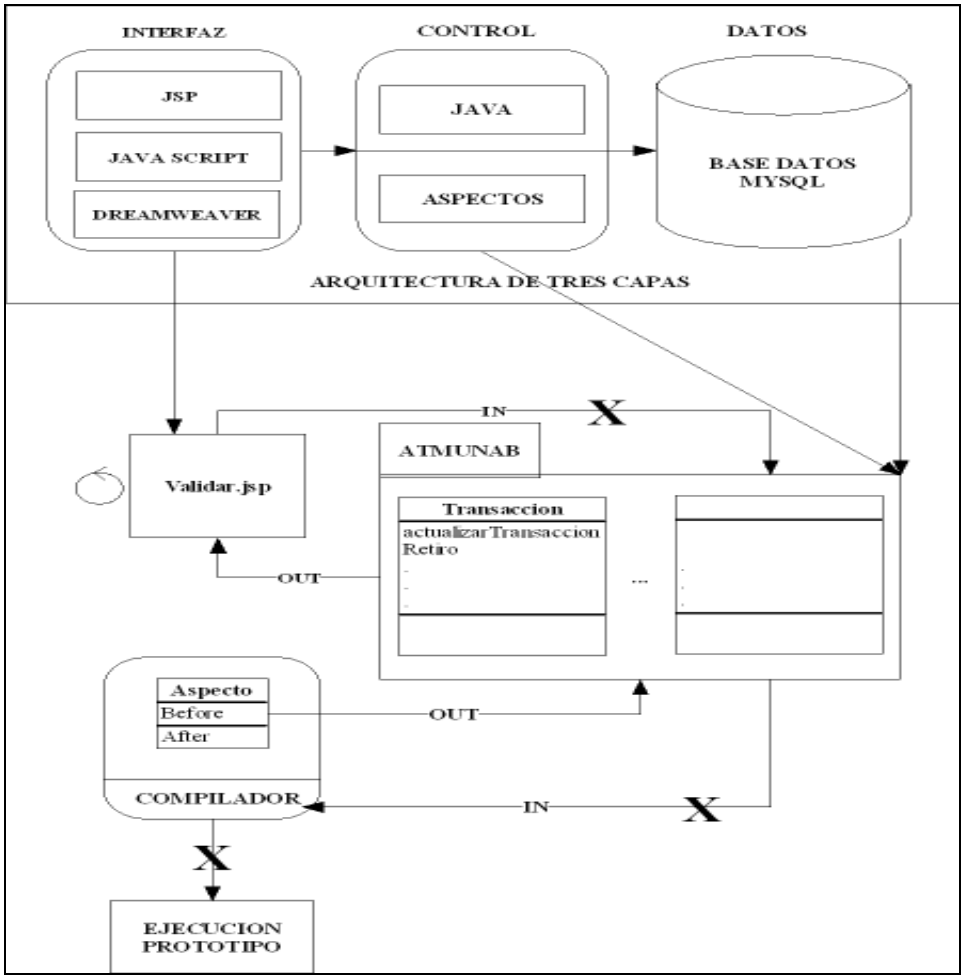
5.3.9 Etapa 9. Expresar el sistema con Aspectos. Se tiene instalado el AspectJ, se crean los aspectos necesarios para cada una de las clases, teniendo en cuenta los procesos de selección de temas refinados; mencionados anteriormente, se tiene un prototipo con AspectJ.

5.3.10 Etapa 10. Generar prototipo para el sistema. Para esta etapa se tiene la interfaz del sistema orientado a Web con sus respectivas clases en java y la base de datos (Ver Anexo D), se implementa la etapa 9, se crea el archivo .lst, con las clases y archivos de AspectJ, esto es el tejido de clases en Java y Aspectos en AspectJ

5.3.11 Etapa 11. Ejecutar el prototipo. Ejecución del prototipo final. Básicamente la aplicación se realizó seguida de una arquitectura de tres capas, en la primera capa se tiene la Interfaz, segunda capa el control del sistema, modelando las clases en Java con los aspectos programados en aspectJ y finalmente una capa de Datos, donde se almacena la información del Sistema. (Ver Anexo E)

En la ejecución del prototipo final (ver figura 20), se muestra el flujo del sistema, donde la capa Interfaz apunta a los .Jsp programados, a su vez estos apuntando junto con las capas de Control y Datos a las clases en programadas en Java, permitiendo pasar los datos y funciones; como vemos al empear a realizarse el ciclo de la validación de los .Jsp, estos no llegan a introducirse en las clases, dado que las clases que interfieren con los aspectos no permiten asociarse, produciendo error a la hora de entretejer las clases en Java, los aspectos y los .Jsp, el compilador de Aspectos no reconoce la capa de Interfaz ya que solamente trabaja con comandos, llegando a la no ejecución del prototipo.

Figura 24. Funcionalidad del Sistema



Fuente. Autoras del proyecto

6. CONCLUSIONES

La investigación permitió evidenciar que los aspectos con los frameworks no se llevan bien debido a que los Framework son diseñados para ser modificados a gusto del programador, cuando este Framework se modifique de nuevo habrá código enredado.

La separación de los aspectos en todos los niveles: análisis, diseño e implementación es un paso importante para llevar a cabo la programación orientada a aspectos para obtener aspectos candidatos, en cada etapa es necesario llevar a cabo el refinamiento de estos para tener al final los aspectos del sistema con el cual se van a trabajar, esto es sobre todo para cuestiones de eficiencia.

Al programar nuevamente el Sistema Orientado a Aspectos después de haber aplicado la Ingeniería de Software Orientado a Aspectos con la metodología ajustada por las autoras, se evidenció la principal ventaja en el código, es mucho más limpio y entendible, esto se debe a la abstracción del código mezclado.

El basarnos en un sistema orientado a objetos, permitió llevar a cabo en forma eficiente la integración y reconocimiento de los aspectos, pues se partió de los requerimientos, caso de uso y diagramas de secuencia, para encontrar los aspectos que se entremezclaban en el código.

Al realizar la integración de los diferentes conceptos utilizados en este proyecto, se pudo comprobar que la creación de una metodología ajustada de ejecución para aplicaciones orientadas a aspectos, puede ser utilizada para futuros trabajos relacionados.

Los requisitos funcionales del sistema permiten el hallazgo de aspectos, el resultado esperado de la aplicación de la metodología está en "limpiar" el código original OO. La separación modulariza los intereses del sistema OO. Esa modularización facilita el trabajo para el programador ya que es un sistema considerablemente más legible.

Con el proyecto sobre Modelado y Desarrollo de Aplicaciones Orientadas a la Web desde una perspectiva de Ingeniería del Software Orientada a Aspectos se buscó estudiar este paradigma para llegar a la construcción de un nuevo sistema con interfaz Web. Al comparar la POO con POA a través un programa de escritorio "Simulador de un Cajero Automático ATMUNAB se evidencian ventajas de la POA como la simplicidad, la facilidad de integración de nuevas funciones a través de la descomposición de un sistema complejo permitiendo su rendimiento, su eficiencia y su seguridad. Se observan, aún, desventajas como los posibles choques entre el código de aspectos y los mecanismos del lenguaje.

Con el diseño e implementación de un sistema basado en aspectos se requiere entender qué se debe incluir en el lenguaje base, qué se debe incluir dentro de los lenguajes de aspectos y qué debe compartirse entre ambos lenguajes**.** Así se pretende aportar una aplicación a los analistas permitiéndoles facilitar el modelado y desarrollo de aplicaciones orientadas a la Web desde una perspectiva de Ingeniería del Software Orientada a Aspectos describiendo cada dato

separadamente con el fin de tener un código menos esparcido y enredado, mejorar el funcionamiento, rendimiento, facilidad, eficiencia, reusabilidad y tiempo de la aplicación.

La investigación entrega una estrategia metodológica que ofrece la descomposición funcional, mecanismos eficientes para la composición de los aspectos, técnicas, métodos y herramientas basadas en UML. Una parte importante de la Ingeniería de Software Orientada a Aspectos es el Modelado Orientado a Aspectos OA, que se centra en las técnicas para identificar, analizar, manejar y representar aspectos en el proceso de diseño del software se fundamenta en la correcta modularización y óptima separación de componentes, siendo un nuevo campo que ha abierto la programación orientada a objetos, se puede observar que los progresos mas significativos se han obtenido gracias a la descomposición de un sistema complejo en partes que sean más fáciles de manejar.

La Programación Orientada a Aspectos es un paradigma de programación relativamente nuevo que ha logrado un nivel de desarrollo interesante y está generando grandes expectativas a nivel mundial con respecto al futuro del desarrollo de software. Este paradigma presenta una nueva forma de modularizar las aplicaciones junto con nuevas formas de relacionar dichos módulos buscando, en últimas, construir aplicaciones más fáciles de construir, mantener y extender.

En las pruebas realizadas con las clases de la aplicación Web ATMUNAB con aspectos, se encontró un problema en la compilación de los aspectos, ya que la arquitectura de la aplicación está realizada en tres capas (Interfaz, procesos y datos).

El problema radica en que el java llama a los aspectos con el compilador de AspectJ, el cual requiere de una clase principal main que no permite compilado adecuadamente por que las aplicaciones Web no requieren de esta clase, además que los datos de las clases se obtienen del Jsp que se encuentra en la primera capa.

Consideramos que el compilador no se encuentra adecuado para aplicaciones Web, debido a que trabaja con líneas de comando (DOS).

BIBLIOGRAFÍA

ASTEASUAIN, Fernando; CONTRERAS, Bernando; ESTÉVEZ, Elsa y FILLOTTRANI, Pablo. Programación Orientada a Aspectos: Metodología y Evaluación. Universidad Nacional del Sur, Argentina, 2005. p. 1-12.

ASTEASUAIN, Fernando y CONTRERAS, Bernando. Programación Orientada a Aspectos, Análisis del paradigma. Universidad Nacional del Sur, Argentina, 2002. p. 1-130.

AOSD STEERING COMMITTEE. AOSD [online], 2005. [Citado 10 febrero 2006]. Disponible en Internet: <http://www.aosd.net>.

CÁCERES, Abdiel, Programación Orientada a aspectos. Centro de Investigación y de Estudios Avanzados - IPN, México 2004.

CLARKE, Siobhàn y BANIASSAD, Elisa. Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley.

[8] AOSD STEERING COMMITTEE. AOSD [En línea], 2005. [Citado 10 febrero 2006]. Disponible en Internet: <http://www.aosd.net>.

DEPARTAMENTO DE CS. E ING. DE LA COMPUTACIÓN. Tesis de Licenciatura "Programación Orientada a Aspectos: Análisis del Paradigma" Fernando

Asteasuain - Bernardo Ezequiel Contreras [En línea], 2007. [Citado 10 febrero 2007]. Disponible en Internet: <http://www.angelfire.com/ri2/aspectos/TesisLic.htm>.

DÍAZ, Elizabeth; ÁLVAREZ, Juan y CONGOTE, John. Aspect-oriented Programming AOP [En línea], 2006. [Citado 20 febrero 2006]. Disponible en Internet:<http://www.slideshare.net/jcongote/programacion-orientada-a-aspectos/>.

GÓMEZ, Pedro y VILLALOBOS, Jesús. Introducción a la Programación Orientada a Aspectos (POA). Universidad de Castilla la Mancha, 2005.

Grupo de Agentes de Software: Ingeniería y aplicaciones, Ciclo de Vida [En línea], 2006. [Citado marzo 2007]. Disponible en Internet: <http://grasia.fdi.ucm.es/ingenias/Spain/integracion/index.php>

HAAK, Betina; DÍAZ, Miguel; MARCOS, Claudia y PRYOR, Jane. Identificación temprana de aspectos, ISISTAN Instituto de sistemas Tandil (Argentina). 2004.

HANNEMANN, Jan y KICZALES, Gregor. “Design Pattern Implementation in Java and AspectJ”. University of British Columbia, [En línea] 2003 [Citado marzo 2007]. Disponible en Internet: <http://www.cs.ubc.ca/labs/spl/papers/2002/oopsla02-patterns.pdf>

IBARGÜENGOITIA, Guadalupe. Desarrollo de Software Orientado a Aspectos. UNAM, México, 2006. p. 1-34.

KICKZALES, Gregor; LAMPING, John; MENDHEKAR, Anurag; MAEDA, Chris; VIDEIRA LOPES, Cristina; LOINGTIER, Jean-Marc y IRWIN, John. “Aspect-Oriented Programming”, in Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland, 1997.

MANZANARES, Guillén. Programación Orientada a Aspectos. Una experiencia práctica con AspectJ, Universidad de Murcia, 2005.

MARTÍNEZ, Luis Vinuesa. Separación Dinámica de Aspectos independiente del Lenguaje y Plataforma mediante el uso de Reflexión Computacional. Universidad de Oviedo. 2007.

MONTES, Pablo. programación Orientada a Aspectos. Politécnico Grancolombiano, Bogotá, 2006. p. 1-101.

NAVASA, A; PALMA, K; MURILLO, J.M y ETEROVIC. Dos modelos arquitectónicos para el DSOA, Universidad de Extremadura (España) y pontificia Universidad católica (Chile). 2005.

PRESSMAN, Roger S. Ingeniería de Software: un enfoque practico. Quinta edición, 2002. Ed. McGraw-Hill.

PROGRAMEMOS.COM. AspectJ por Fernando Asteasuain [En línea], 2005. [Citado 12 febrero 2007]. Disponible en Internet: <http://www.programemos.com/index.php?option=com_content&task=view&id=%20158&Itemid=68>.

RASHID, Awais; MOREIRA, Ana y ARAÚJO, João. Modularisation and Composition of Aspectual. Lancaster University y FCT, Universidade Nova de Lisboa, 2002.

REINA QUINTERO, Antonia Maria. Visión Genereal de la Programación Orientada a Aspectos. Universidad de Sevilla, 2000.

RODRÍGUEZ ECHEVERRIA, Roberto. Modelando Procesos de Negocio Web desde una Perspectiva Orientada a Aspectos. Universidad de Extremadura.2007.

WIKIPEDIA Foundation, Inc. Proceso Unificado de Rational [En línea], 2007. [Citado marzo 2007]. Disponible en Internet: <http://es.wikipedia.org/wiki/Proceso_Unificado_de_Rational>.

WIKIPEDIA Foundation, Inc. Proceso Framework [En línea], 2007. [Citado septiembre 2007]. Disponible en Internet: <http://es.wikipedia.org/wiki/Framework>

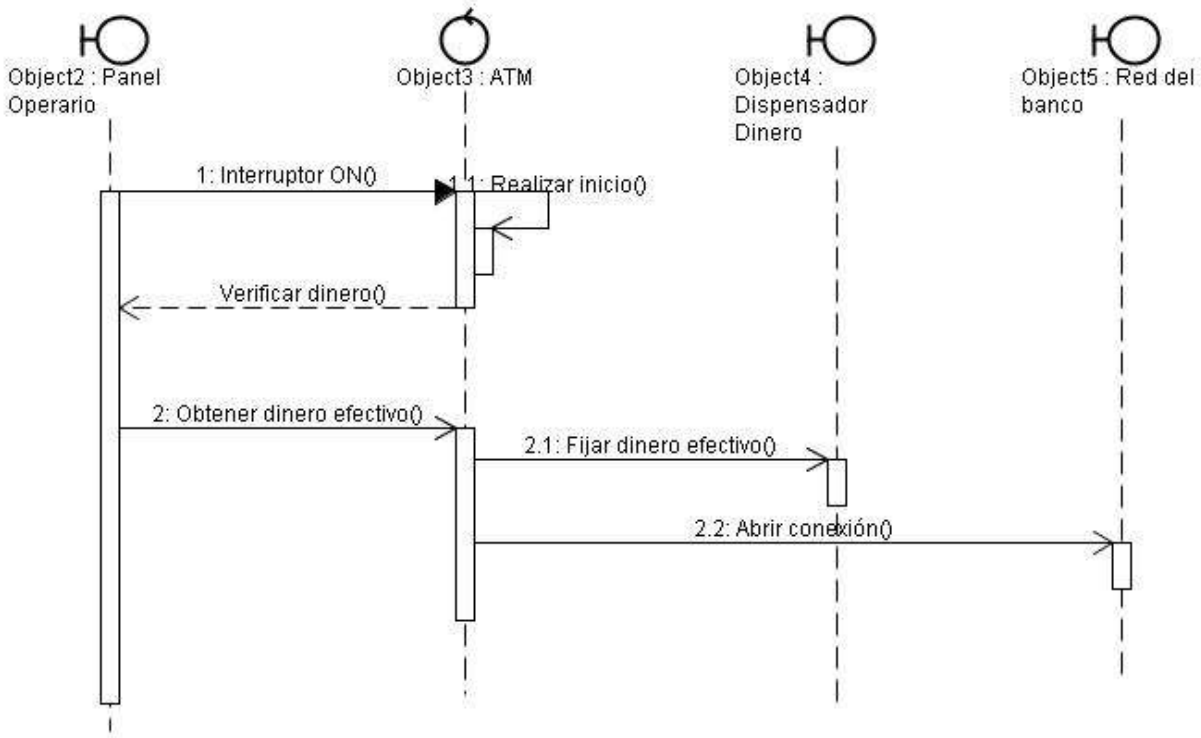
WIKIPEDIA Foundation, Inc.. Programación Orientada a Aspectos (POA) [En línea], 2007. [Citado 10 febrero 2007]. Disponible en Internet: <http://es.wikipedia.org/wiki/Programaci%C3%B3n_Orientada_a_Aspectos>.

Anexo A

1. DIAGRAMA DE SECUENCIAS ORIENTADO A OBJETOS

1.1 INICIAR SISTEMA

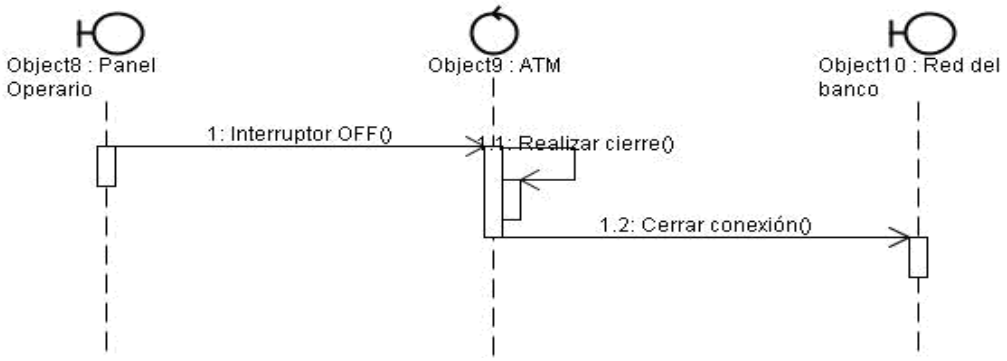
Figura 1. Diagrama de secuencia (Iniciar sistema)



Fuente. Mathematics and Computer Science: Object-Oriented Software Development, [online], 2000. Disponible en Internet:
< http://www.math-cs.gordon.edu/courses/cs211/index.html>

1.2 CIERRE DEL SISTEMA

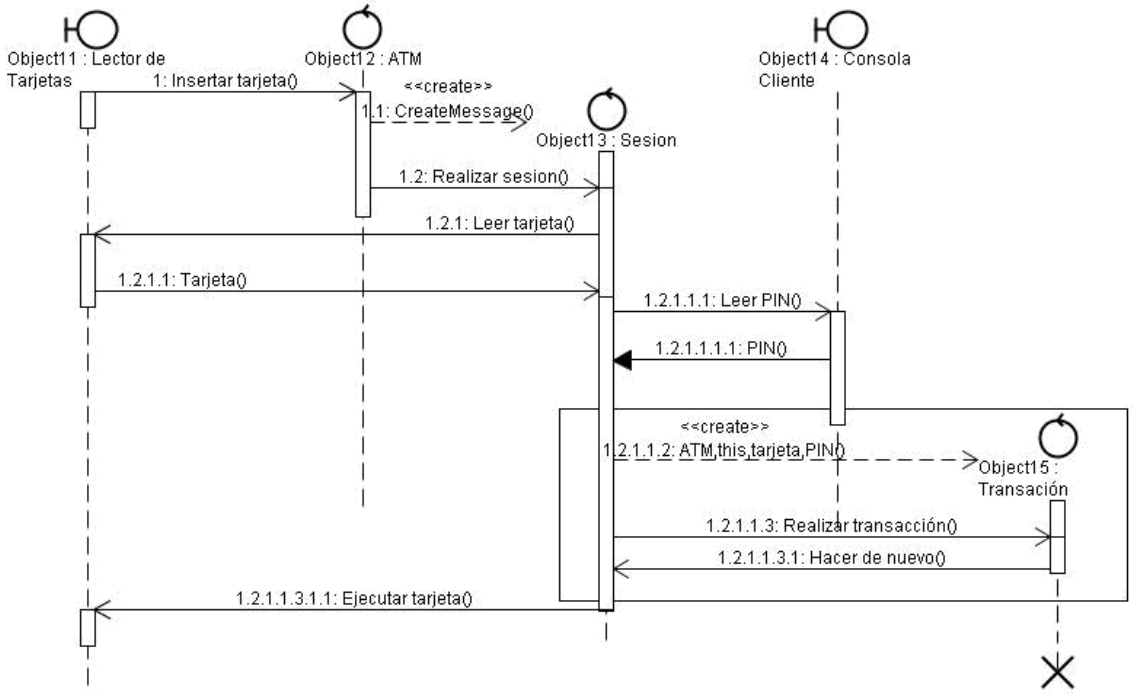
Figura 2. Diagrama de secuencia (Cierre del sistema)



Fuente., BIT.

1.3 SESIÓN

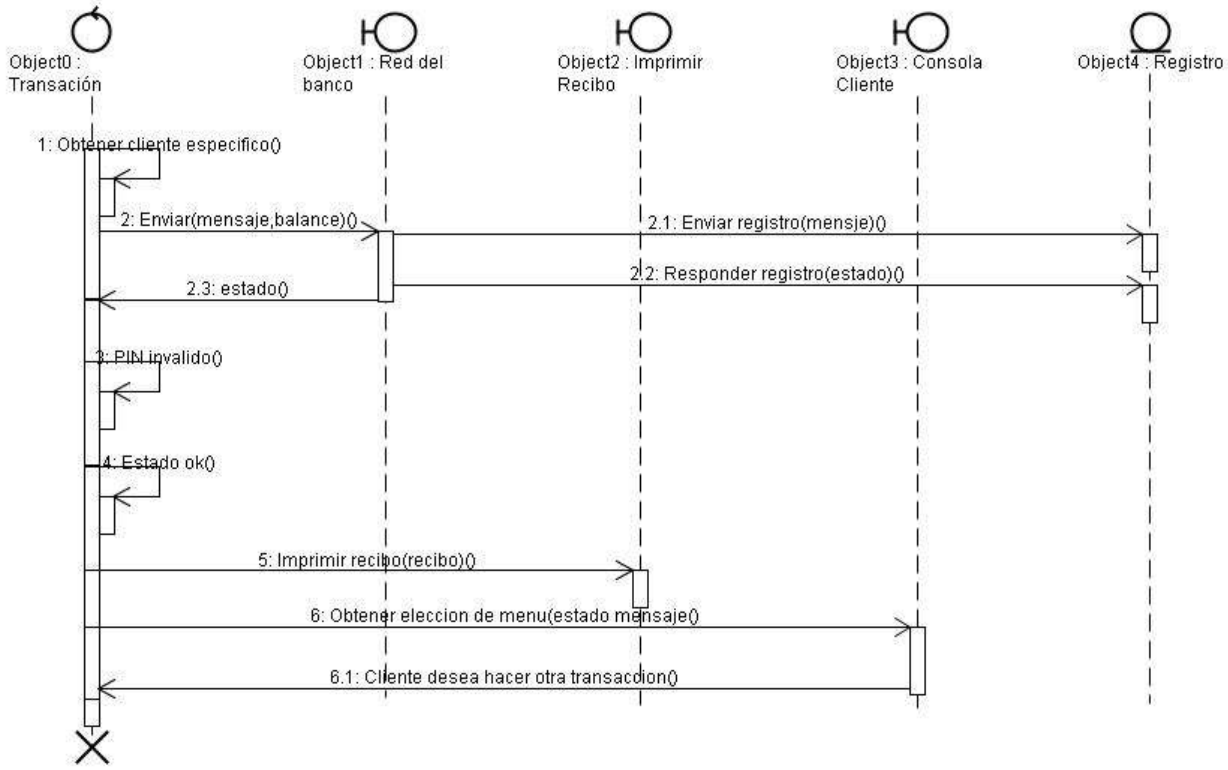
Figura 3. Diagrama de secuencia (Sesión)



Fuente., BIT.

1.4 Transacción

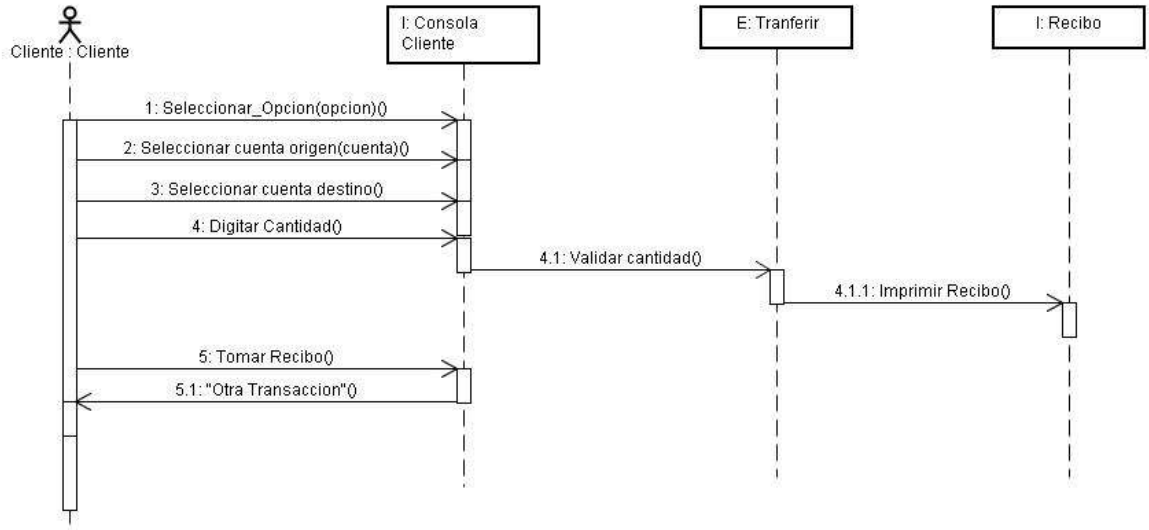
Figura 4. Diagrama de secuencia (transacción)



Fuente. , BIT.

1.5 Transferir

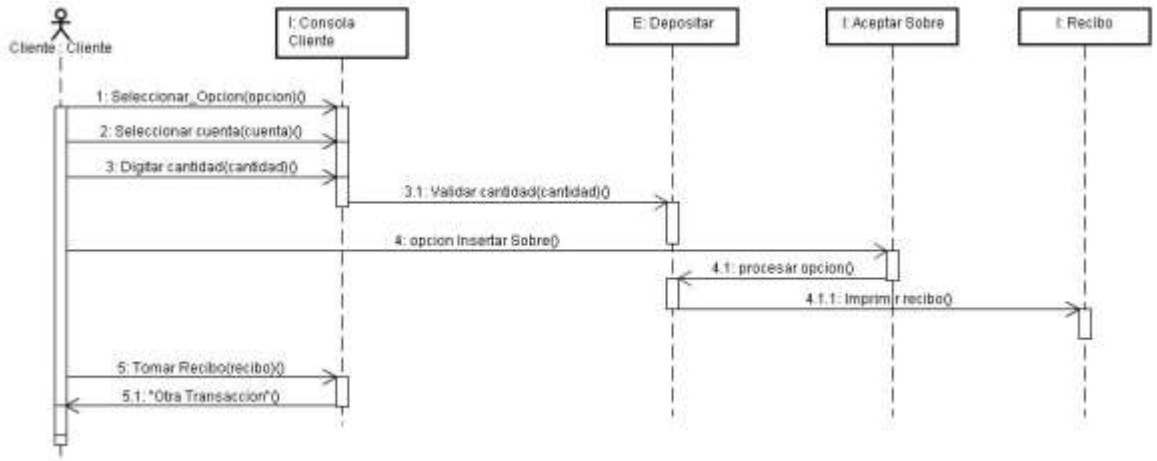
Figura 5. Diagrama de secuencia (transferir)



Fuente. , BIT.

1.6 Depositar

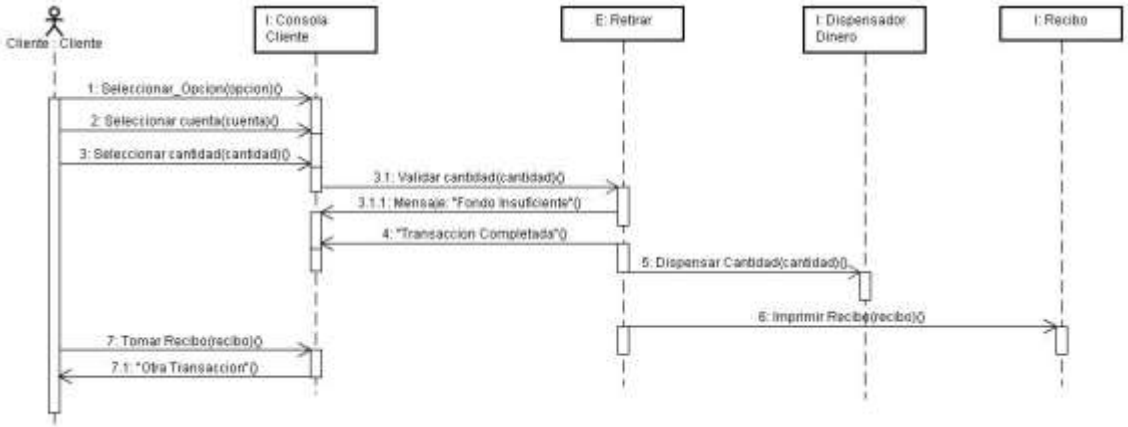
Figura 6. Diagrama de secuencia (Depositar)



Fuente., BIT.

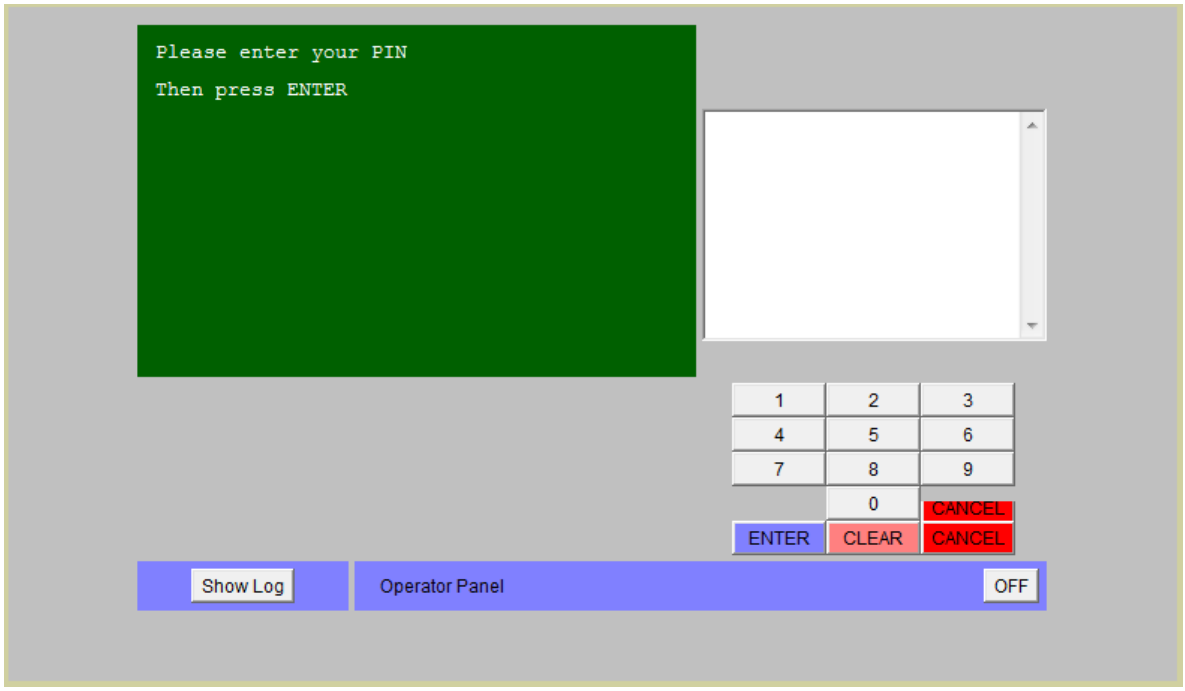
1.7 Retiro

Figura 7. Diagrama de secuencia (Depositar)



Fuente. , BIT.

Figura 8. Interfaz Cajero Automático ATM sin aspectos

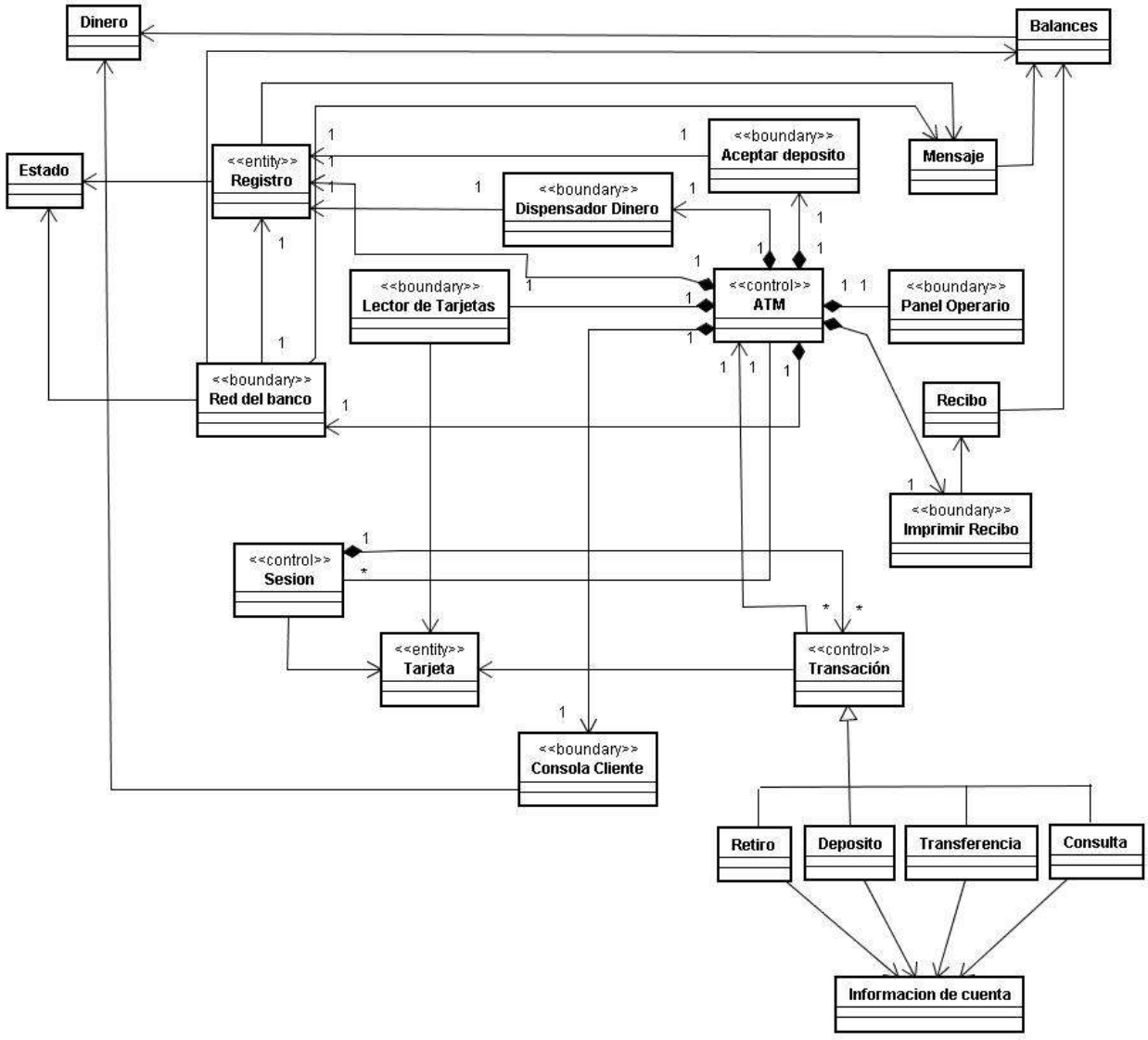


Fuente. , BIT.

Anexo B

1. DIAGRAMA DE CLASES ORIENTADO A OBJETOS

Figura 1. Diagrama de clases del sistema ATM

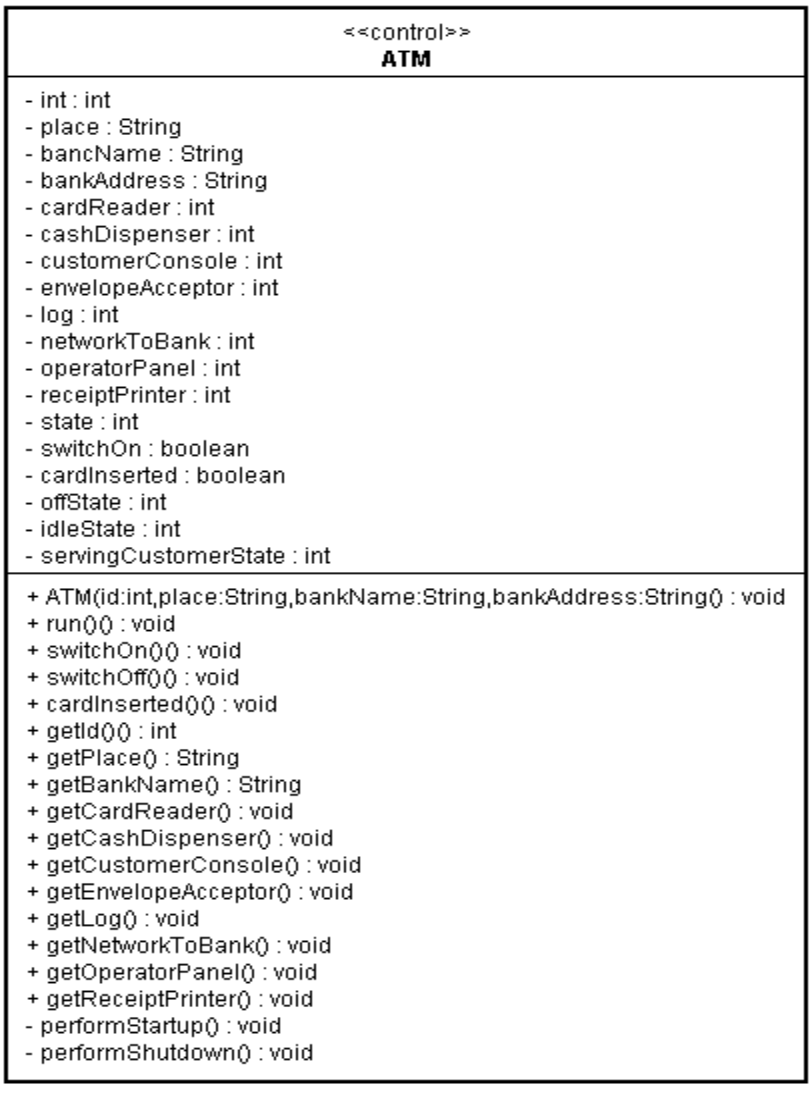


Fuente. , BIT.

1.1 MÉTODOS Y ATRIBUTOS DE CADA CLASE

1.1.1 ATM

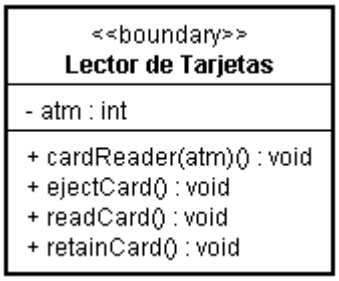
Figura 2. Métodos y atributos de la clase ATM



Fuente. , BIT.

1.1.2 Lector Tarjetas

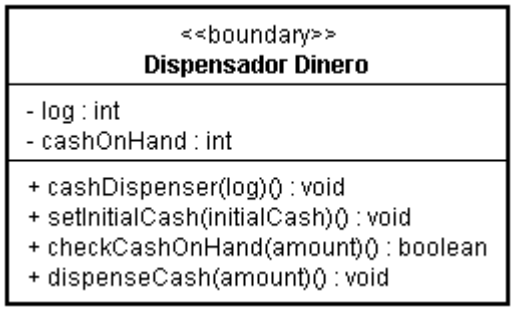
Figura 3. Métodos y atributos de la clase Lector de tarjetas



Fuente. , BIT.

1.1.3 Dispensador dinero

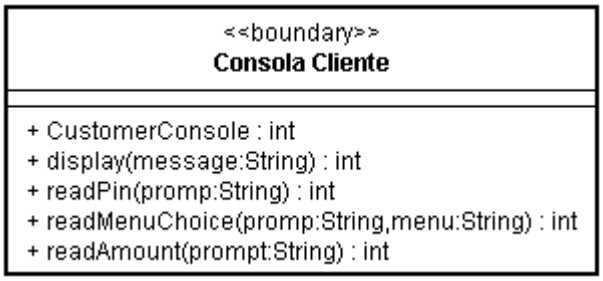
Figura 4. Métodos y atributos de la clase Dispensador de dinero



Fuente. , BIT.

1.1.4 Consola cliente

Figura 5. Métodos y atributos de la clase Consola del cliente



Fuente. , BIT.

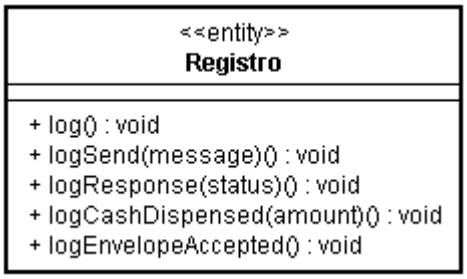
1.1.5 Aceptar depósito

Figura 6. Métodos y atributos de la clase aceptar depósito

Fuente., BIT.

1.1.6 Log

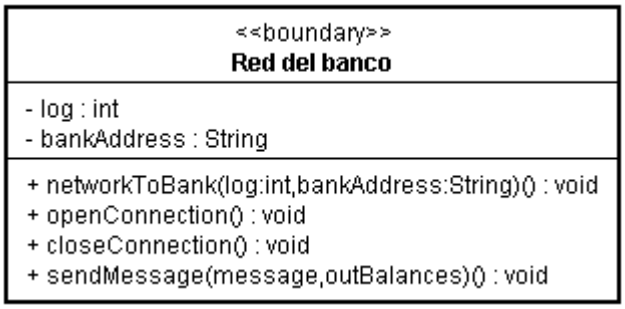
Figura 7. Métodos y atributos de la clase Log



Fuente., BIT.

1.1.7 Red del Banco

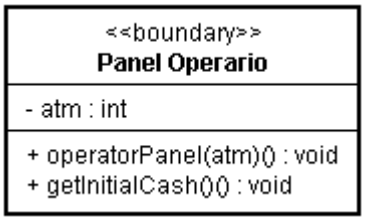
Figura 8. Métodos y atributos de la clase Red del banco



Fuente., BIT.

1.1.8 Panel operario

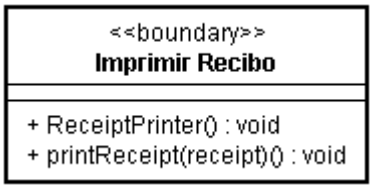
Figura 9. Métodos y atributos de la clase panel operario



Fuente., BIT.

1.1.9 Imprimir recibo

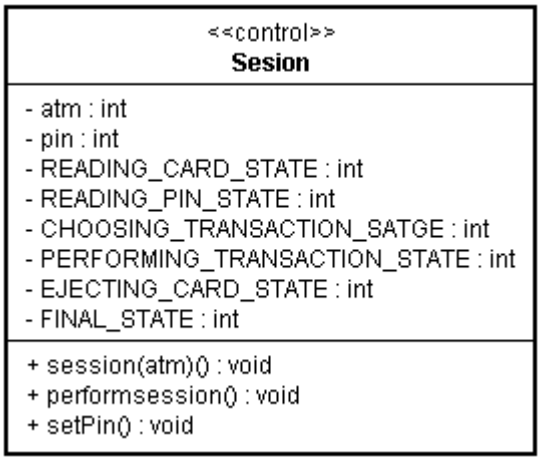
Figura 10. Métodos y atributos de la clase Imprimir recibo



Fuente., BIT.

1.1.10 Sesión

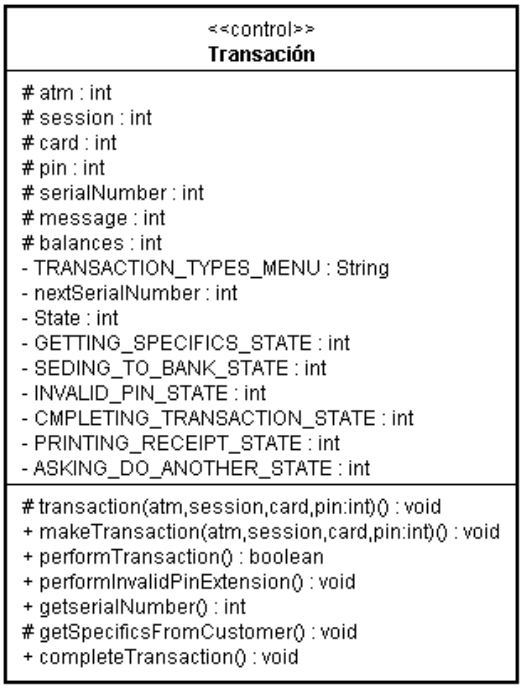
Figura 11. Métodos y atributos de la clase sesión



Fuente., BIT.

1.1.11 Transacción

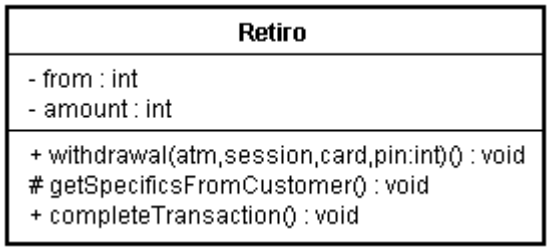
Figura 12. Métodos y atributos de la clase transacción



Fuente., BIT.

1.1.12 Retiro

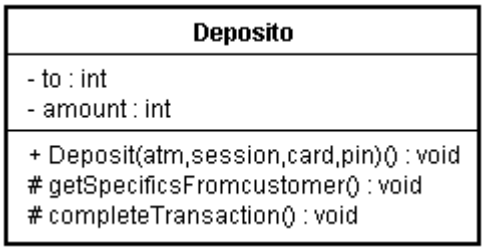
Figura 13. Métodos y atributos de la clase retiro



Fuente.,BIT.

1.1.13 Depósito

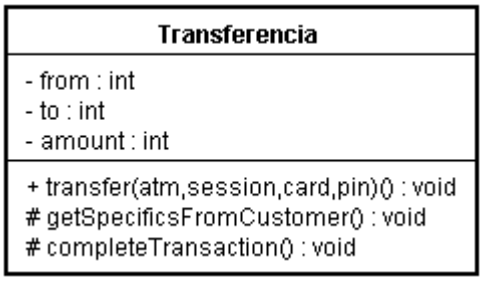
Figura 14. Métodos y atributos de la clase depósito



Fuente.,BIT.

1.1.14 Transferencia

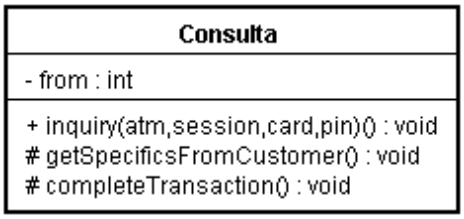
Figura 15. Métodos y atributos de la clase transferencia



Fuente.,BIT.

1.1.15 Consulta

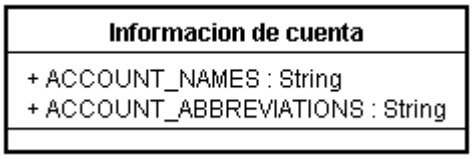
Figura 16. Métodos y atributos de la clase consulta



Fuente.,BIT.

1.1.16 Información de cuenta

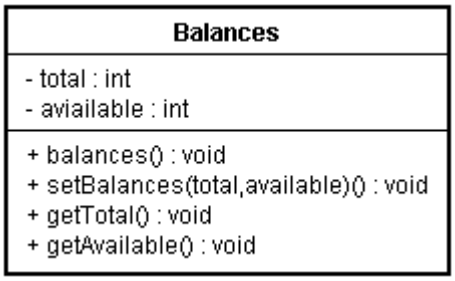
Figura 17. Métodos y atributos de la clase información de cuenta



Fuente.,BIT.

1.1.17 Balances

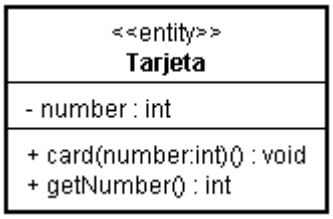
Figura 18. Métodos y atributos de la clase balances



Fuente.,BIT.

1.1.18 Tarjeta

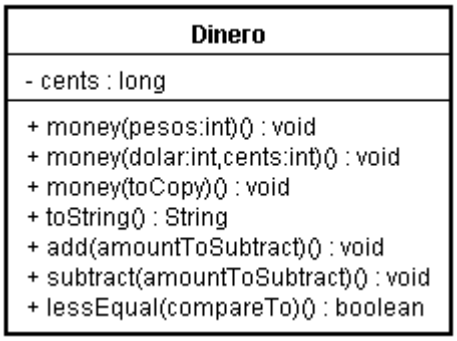
Figura 19. Métodos y atributos de la clase tarjeta



Fuente.,BIT.

1.1.19 Dinero

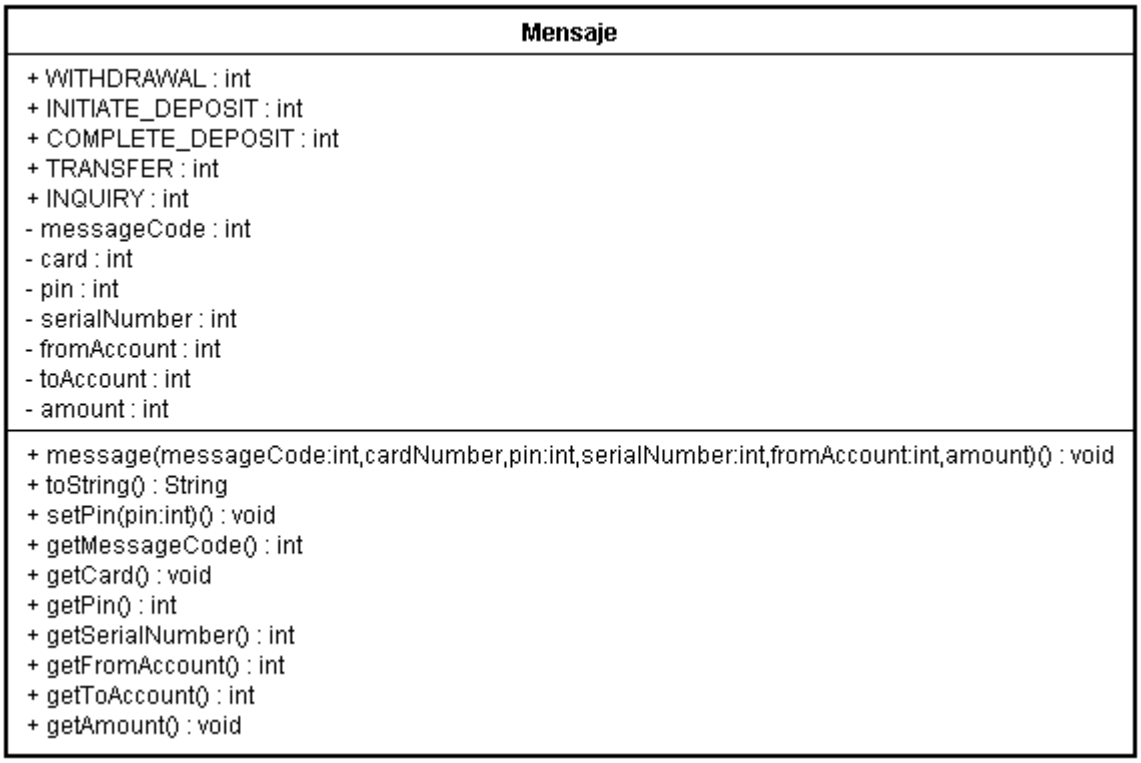
Figura 20. Métodos y atributos de la clase dinero



Fuente.,BIT.

1.1.20 Mensaje

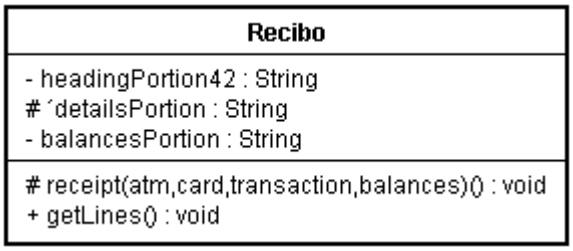
Figura 21. Métodos y atributos de la clase mensaje



Fuente.,BIT.

1.1.21 Recibo

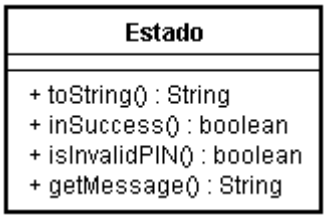
Figura 22. Métodos y atributos de la clase recibo



Fuente.,BIT.

1.1.22 Estado

Figura 23. Métodos y atributos de la clase estado



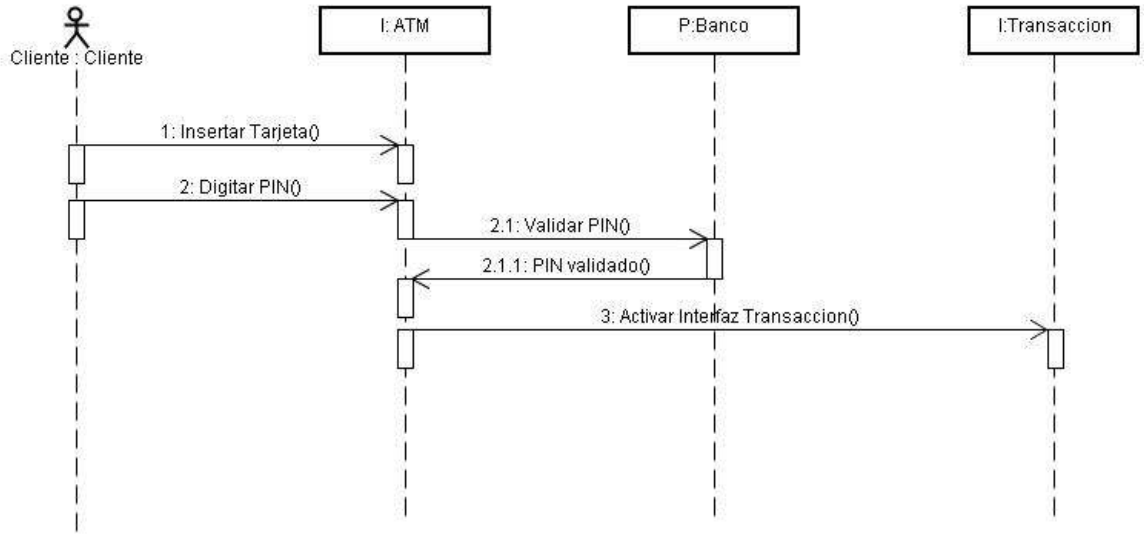
Fuente.,BIT.

Anexo C

1. DIAGRAMA DE SECUENCIAS ORIENTADO A ASPECTOS

1.1 INGRESAR

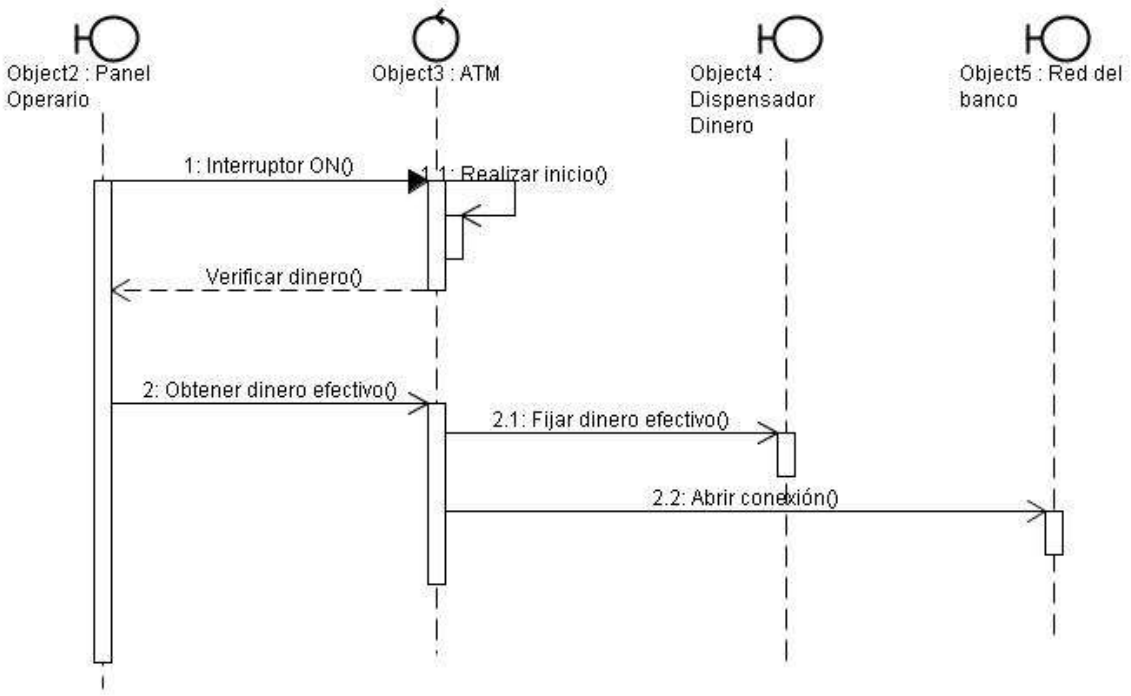
Figura 1. Ingresar



Fuente: Autoras del proyecto

1.2 INICIAR SISTEMA

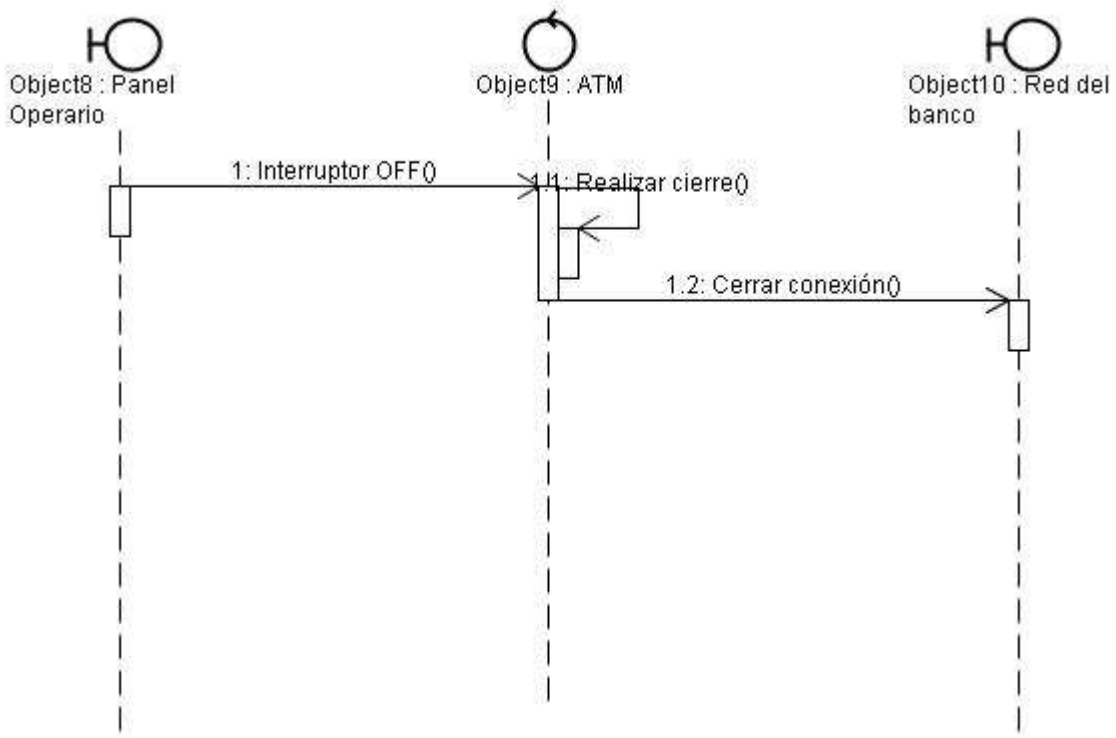
Figura 2. Iniciar sistema



Fuente: Autoras del proyecto

1.3 CIERRE DE SESIÓN

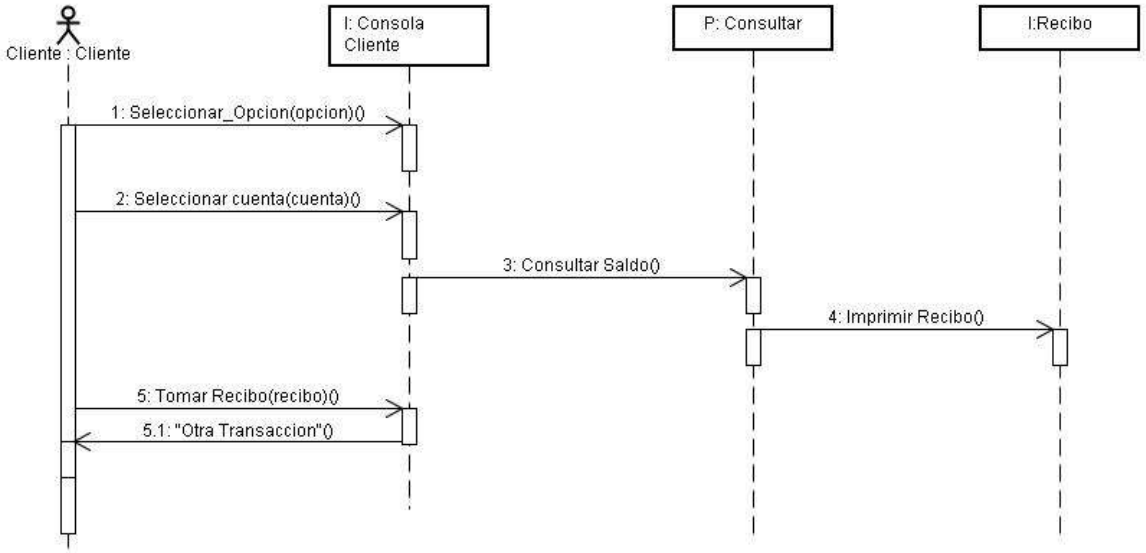
Figura 3. Cierre de sesión



Fuente: Autoras del proyecto

1.4 CONSULTAR

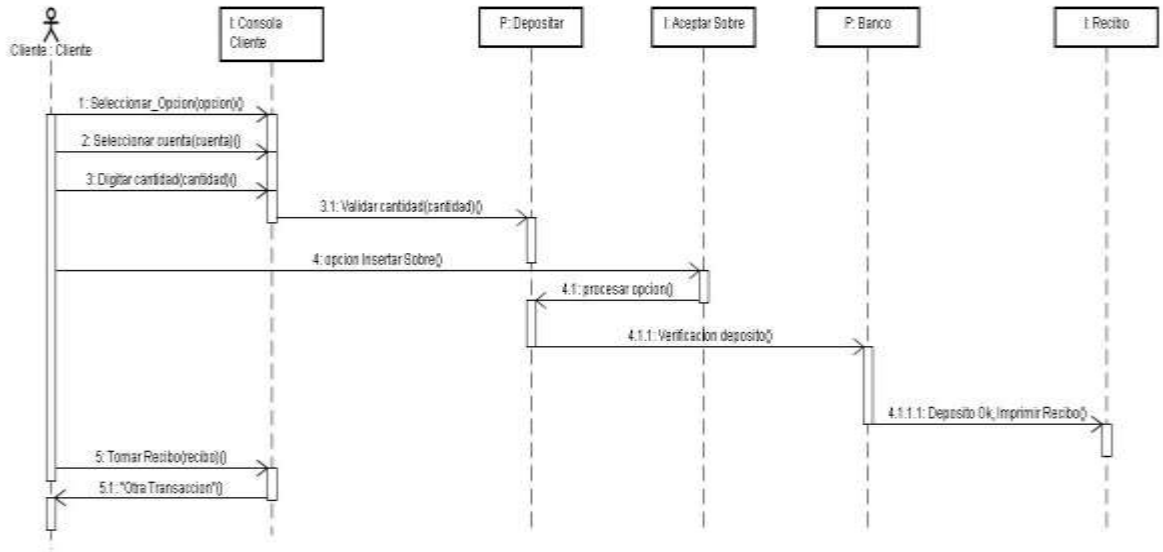
Figura 4. Consultar



Fuente: Autoras del proyecto

1.5 DEPOSITAR

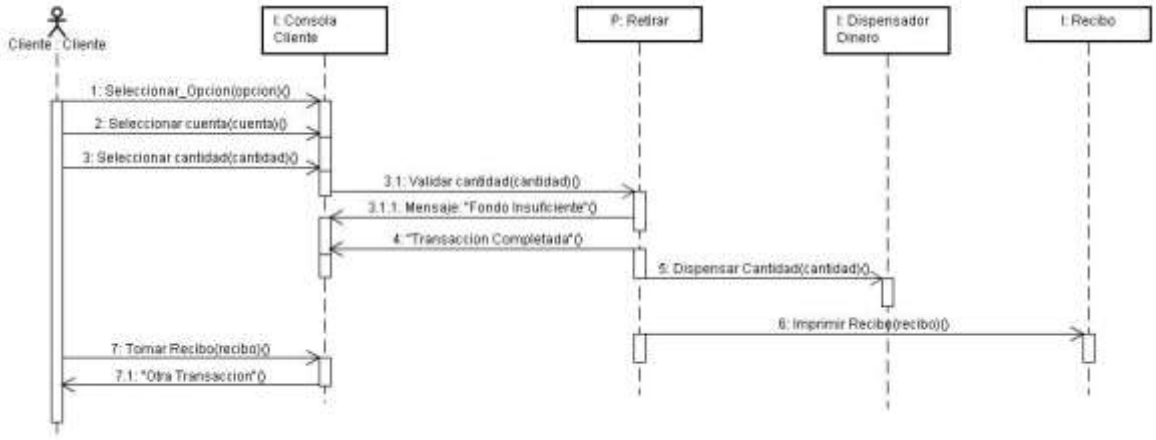
Figura 5. Depositar



Fuente: Autoras del proyecto

1.6 RETIRO

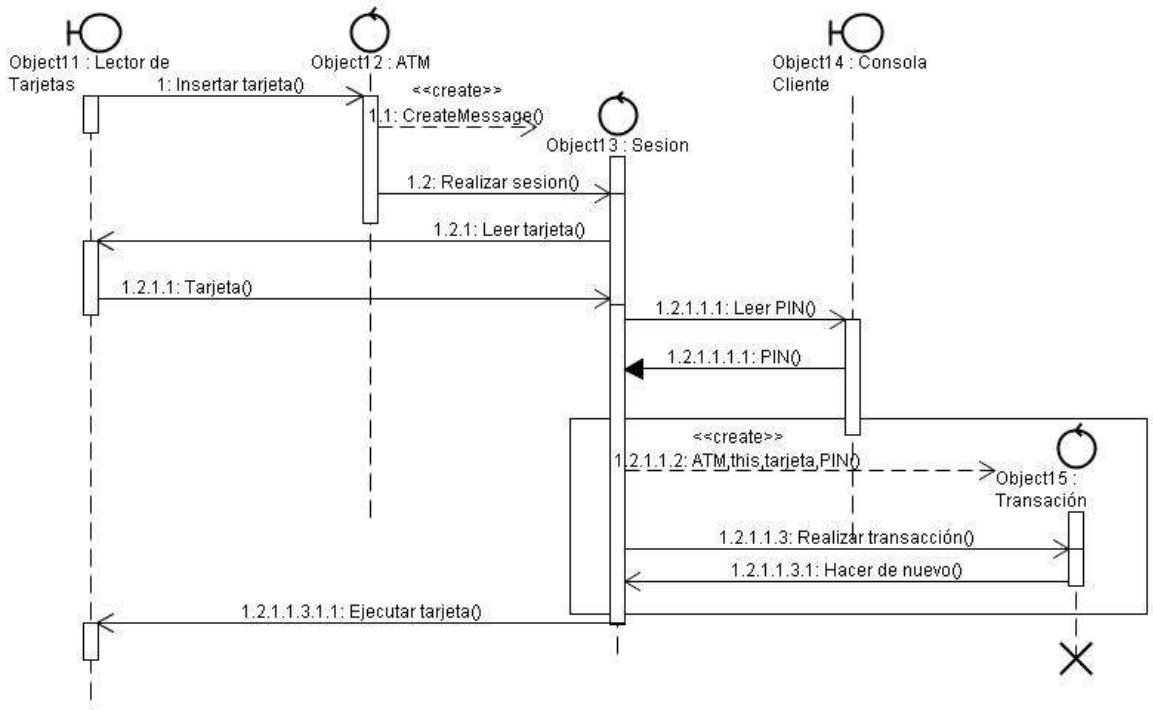
Figura 6. Retiro



Fuente: Autoras del proyecto

1.7 SESIÓN

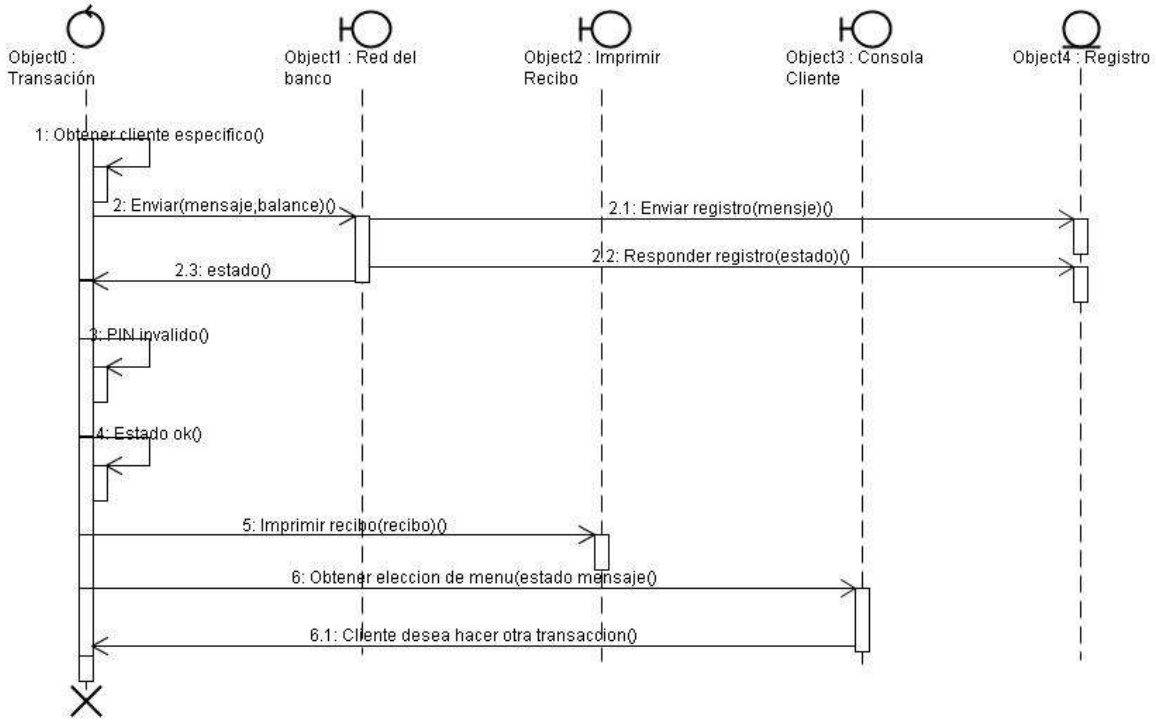
Figura 7. Sesión



Fuente: Autoras del proyecto

1.8 TRANSACCIÓN

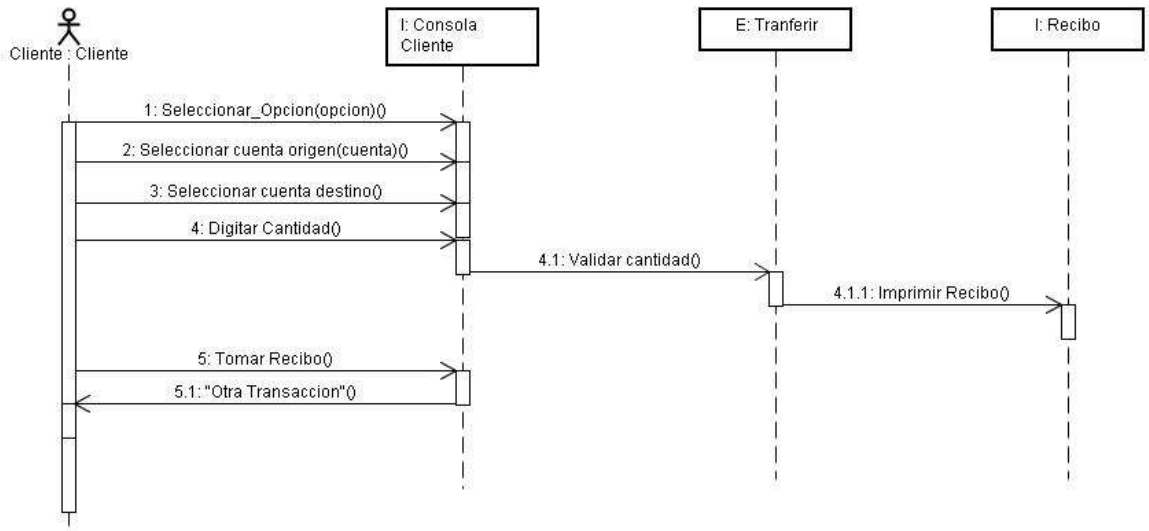
Figura 8. Transacción



Fuente: Autoras del proyecto

1.9 TRANSFERIR

Figura 9. Transferir

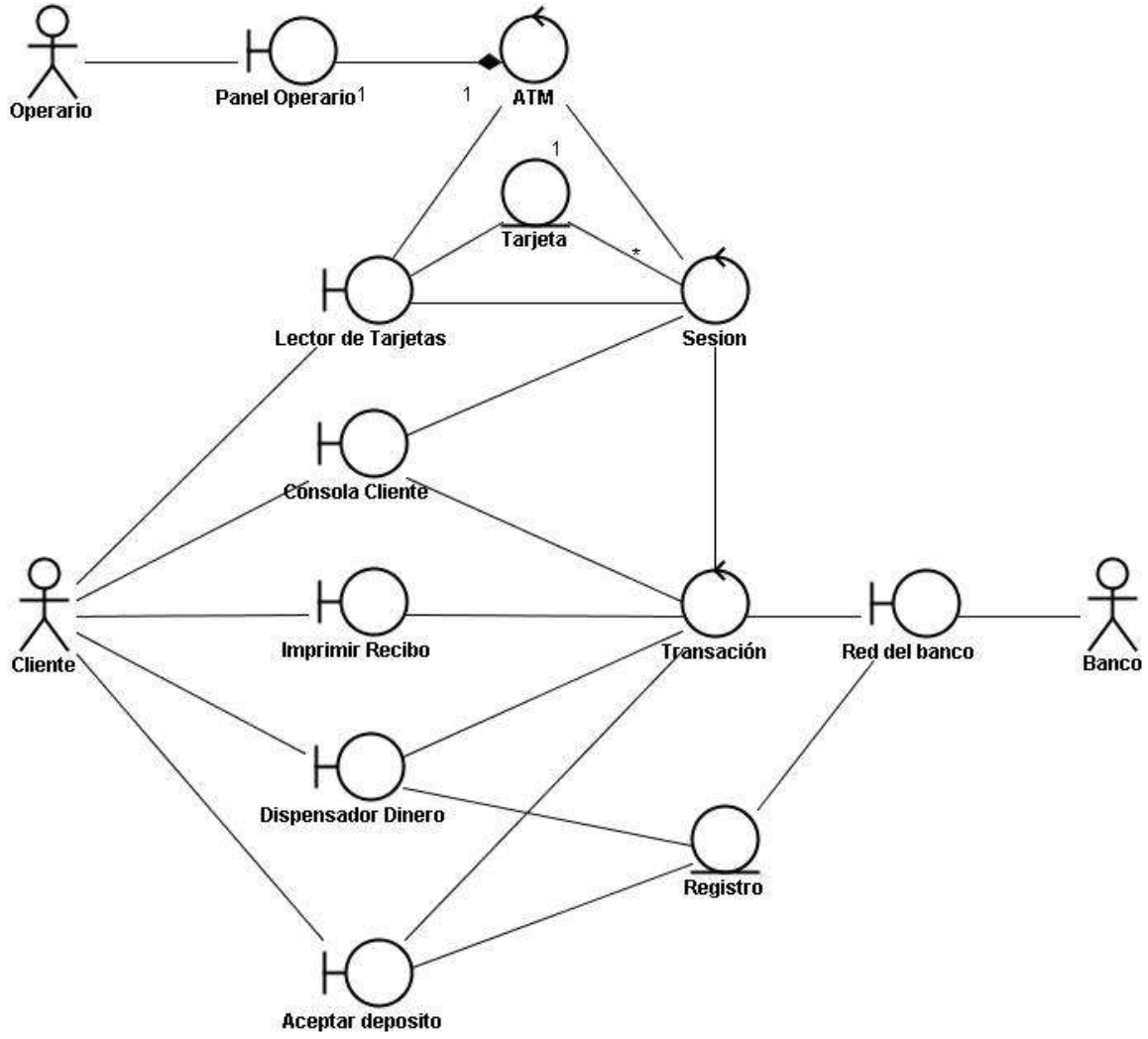


Fuente: Autoras del proyecto

2. DIAGRAMA DE CLASES ORIENTADO A ASPECTOS

2.1 DIAGRAMA DE CLASES GENERAL

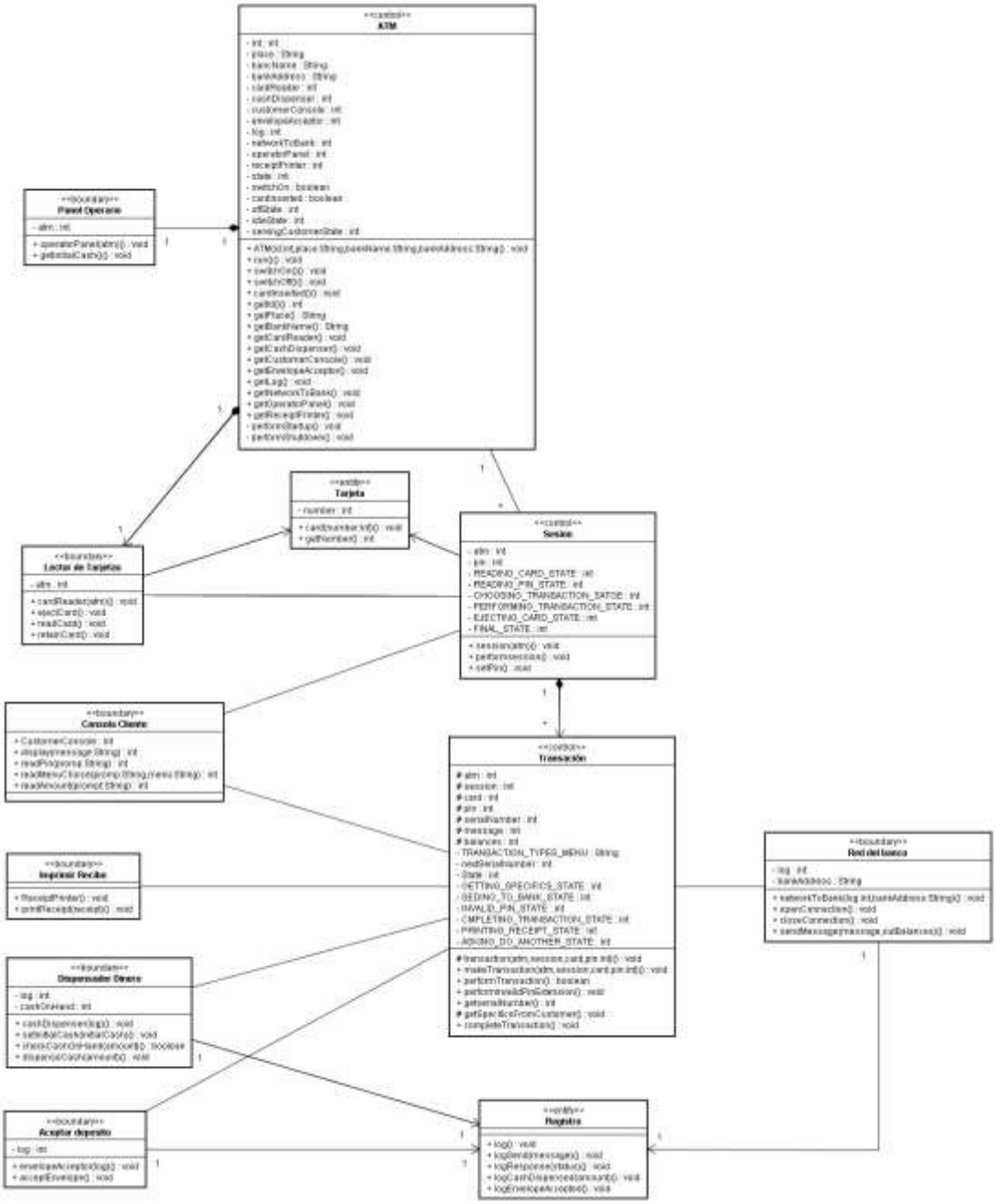
Figura 10. Diagrama de clases general



Fuente: Autoras del proyecto

2.2 DIAGRAMA DE CLASES

Figura 11. Diagrama de clases



Fuente: Autoras del proyecto

Anexo D

1. BASE DE DATOS DEL SISTEMA ATMUNAB

1.1 SCRIPT BASE DE DATOS

```
-- MySQL Administrator dump 1.4
--
-- ------------------------------------------------------
-- Server version      4.0.18-max-debug


/*!40101 SET
@OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET
@OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET
@OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS,
UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;


--
-- Create schema atmunab
```

```
--
CREATE DATABASE /*!32312 IF NOT EXISTS*/ atmunab;
USE atmunab;

--
-- Table structure for table `atmunab`.`cajero`
--

DROP TABLE IF EXISTS `cajero`;
CREATE TABLE `cajero` (
  `capacidad` int(15) NOT NULL default '0',
  `capacidadUtilizada` int(15) NOT NULL default '0',
```

```
  `dineroactual` double NOT NULL default '0',
  `estado` int(1) NOT NULL default '0',
  PRIMARY KEY  (`capacidad`)
) TYPE=InnoDB ROW_FORMAT=FIXED;

--
-- Dumping data for table `atmunab`.`cajero`
--

/*!40000 ALTER TABLE `cajero` DISABLE KEYS */;
INSERT INTO `cajero` (`capacidad`,`capacidadUtilizada`,`dineroactual`,`estado`)
VALUES
 (100,0,6800000,0);
/*!40000 ALTER TABLE `cajero` ENABLE KEYS */;



--
-- Table structure for table `atmunab`.`cuenta`
--

DROP TABLE IF EXISTS `cuenta`;
CREATE TABLE `cuenta` (
  `codigo_cuenta` int(10) NOT NULL default '0',
  `nombre` varchar(10) NOT NULL default '',
  PRIMARY KEY  (`codigo_cuenta`)
) TYPE=InnoDB ROW_FORMAT=DYNAMIC;

--
-- Dumping data for table `atmunab`.`cuenta`
--
```

```
/*!40000 ALTER TABLE `cuenta` DISABLE KEYS */;
INSERT INTO `cuenta` (`codigo_cuenta`,`nombre`) VALUES
 (1,'AHORRO'),
 (2,'CORRIENTE'),
 (3,'EMPRESARIA');
/*!40000 ALTER TABLE `cuenta` ENABLE KEYS */;


--
-- Table structure for table `atmunab`.`log`
--

DROP TABLE IF EXISTS `log`;
CREATE TABLE `log` (
  `codigo_cuenta` int(11) NOT NULL default '0',
  `codigo_tarjeta` varchar(45) NOT NULL default '',
  `fecha_movimiento` datetime NOT NULL default '0000-00-00 00:00:00',
  PRIMARY KEY  (`codigo_cuenta`,`codigo_tarjeta`,`fecha_movimiento`)
) TYPE=InnoDB;

--
-- Dumping data for table `atmunab`.`log`
--

/*!40000 ALTER TABLE `log` DISABLE KEYS */;
/*!40000 ALTER TABLE `log` ENABLE KEYS */;


--
-- Table structure for table `atmunab`.`usuario`
```

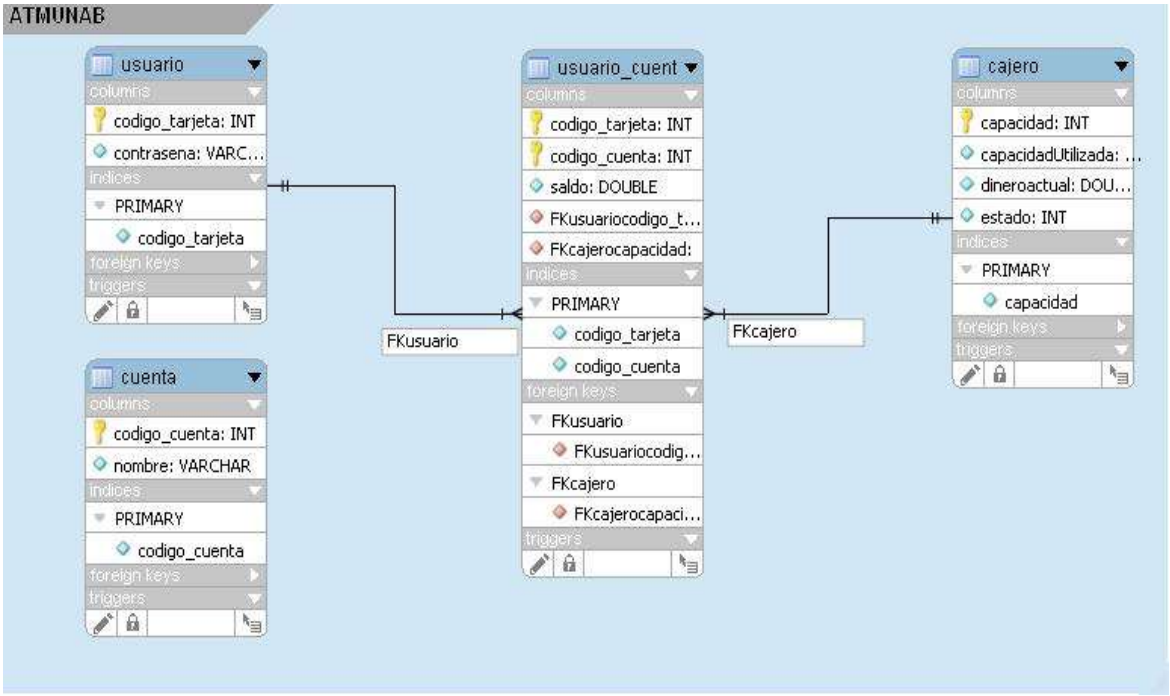
```
--

DROP TABLE IF EXISTS `usuario`;
CREATE TABLE `usuario` (
  `codigo_tarjeta` int(10) NOT NULL default '0',
  `contrasena` varchar(10) NOT NULL default '',
  PRIMARY KEY  (`codigo_tarjeta`)
) TYPE=InnoDB ROW_FORMAT=DYNAMIC;

--
-- Dumping data for table `atmunab`.`usuario`
--

/*!40000 ALTER TABLE `usuario` DISABLE KEYS */;
INSERT INTO `usuario` (`codigo_tarjeta`,`contrasena`) VALUES
 (852,'258'),
 (1234,'4321'),
 (4321,'1234');
/*!40000 ALTER TABLE `usuario` ENABLE KEYS */;

--
-- Table structure for table `atmunab`.`usuario_cuenta`
--

DROP TABLE IF EXISTS `usuario_cuenta`;
CREATE TABLE `usuario_cuenta` (
  `codigo_tarjeta` int(15) NOT NULL default '0',
  `codigo_cuenta` int(15) NOT NULL default '0',
  `saldo` double NOT NULL default '0',
  PRIMARY KEY  (`codigo_tarjeta`,`codigo_cuenta`)
```

```
) TYPE=InnoDB ROW_FORMAT=FIXED;

--
-- Dumping data for table `atmunab`.`usuario_cuenta`
--

/*!40000 ALTER TABLE `usuario_cuenta` DISABLE KEYS */;
INSERT INTO `usuario_cuenta` (`codigo_tarjeta`,`codigo_cuenta`,`saldo`)
VALUES
 (258,1,10020),
 (258,2,40020),
 (258,3,100020),
 (1234,1,9352060),
 (1234,2,1730020),
 (1234,3,970020),
 (4321,1,20020),
 (4321,2,500020),
 (4321,3,2000020);
/*!40000 ALTER TABLE `usuario_cuenta` ENABLE KEYS */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT
*/;
/*!40101 SET
CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION
*/;
```

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT
*/;

1.2 DIAGRAMA ENTIDAD RELACION

Figura 1. Diagrama entidad relacion sistema ATMUNAB



Fuente: Autoras del proyecto

Anexo E

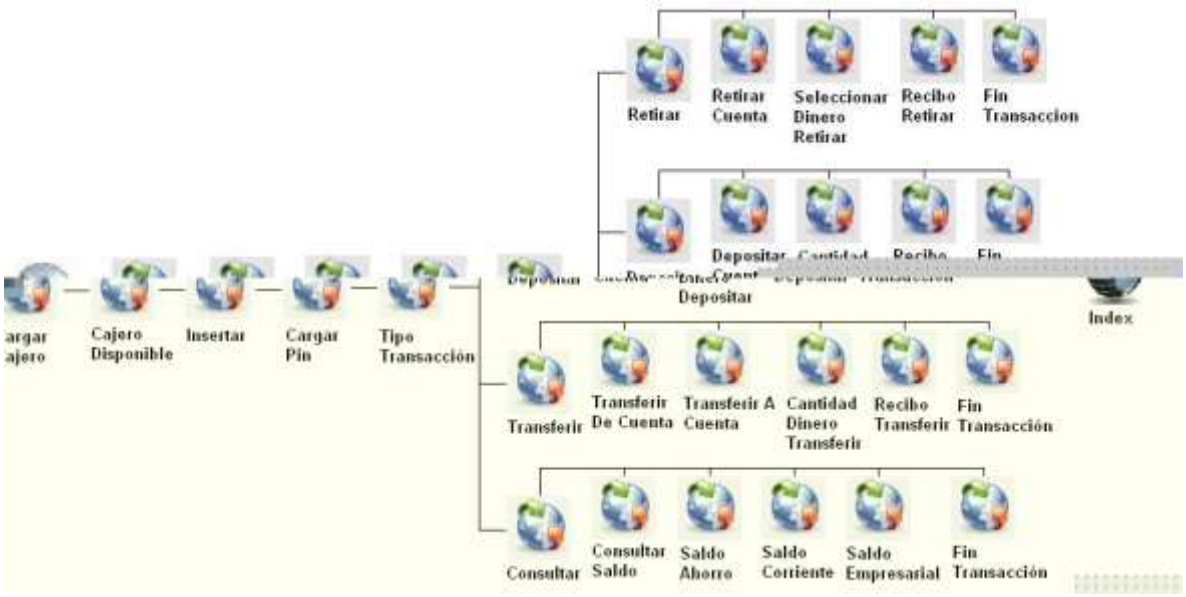
1. NAVEGACION DE CAJERO AUTOMÁTICO ATMUNAB

De acuerdo a la capa de Presentación de la arquitectura de tres capas para la POA (Programación Orientada a Aspectos), se llevó a cabo el diseño de un mapa de navegación y una nueva interfaz en Web, ya que la aplicación anterior era en escritorio y totalmente plana, esto es realizado tomando en cuenta los procesos de aspectos aplicados a la capa de datos y a la capa de procesos. Además, se realizó las nuevas clases partiendo de la aplicación en escritorio, los cuales podrán verse en la nueva estructura de funcionamiento de la Simulación del Cajero Automático ATMUNAB.

El mapa de navegación para la Simulación del Cajero Automático ATMUNAB, así como la interfaz propuesta se puede ver en las figura 1 y en las figuras de la 2 a la 25 respectivamente.

El mapa de navegación fue diseñado de acuerdo a cada una de las actividades que pueda realizar el usuario, en este caso Retiros, Depósito, Transferencia y Consulta de saldo. Para cada actividad, se especificaron las interfaces a las cuales tendrá acceso cuando hagan uso del sistema, lo cual facilita la realización de las diferentes actividades de las personas en el sistema

Figura 1. Mapa de Navegación



Fuente: Autoras del Proyecto

1.1 VISTA DE INTERFACES GENERAL

Index. El operario debe oprimir el botón ON para poner en funcionamiento el cajero ATMUNAB, se valida el funcionamiento On u OFF por medio de una consulta a la base de datos.

Figura 2. Index

Fuente: Autoras del Proyecto

1.1.2 cargar_cajero. El operario debe ingresar el número de billetes para cargar el cajero. CargaCajero.jsp, valida la cantidad máxima de billetes que se pueden ingresar, siendo esta de 100 billetes, esto se obtiene haciendo una consulta en la base de datos, en la cual se tiene en cuenta la cantidad máxima y la cantidad utilizada.

Figura 3. cargar_cajero

Fuente: Autoras del Proyecto

En caso de sobrepasar la cantidad máxima se enviara a una pagina de error CantidadBilletesIncorrecto.jsp, mostrando la cantidad actual de billetes en el cajero, se pedirá al operario ingresar nuevamente la cantidad de billetes en caso de ser menor a 100 o ingresar la cantidad de billetes faltantes.

Figura 4. CantidadBilletesIncorrecto

Fuente: Autoras del Proyecto

1.1.3 CajeroDisponible. Pide al usuario ingresar la tarjeta, para ello se debe oprimir el botón Tarjeta, para simular la entrada de la tarjeta.

Figura 4. CajeroDisponible

Fuente: Autoras del Proyecto

1.1.4 Insertar. Pide al usuario ingresar el número de la tarjeta. ValidarTarjeta.jsp valida si el número de la tarjeta es correcto y lo envía a ingresar el número del PIN.

Figura 5. Insertar

Fuente: Autoras del Proyecto

En caso de que el número de la tarjeta sea incorrecto, se enviara a una pagina de error TarjetaInvalida.jsp, donde pide al usuario ingresar nuevamente el número de la tarjeta.

Figura 6. TarjetaInvalida

Fuente: Autoras del Proyecto

1.1.5 cargar_pin. Pide al usuario ingresar el PIN respectivo a la tarjeta, ValidarPin.jsp valida que el número del PIN sea correcto y corresponda al número de la tarjeta, se encuentra en la base de datos.

Figura 6. cargar_pin

Fuente: Autoras del Proyecto

En caso de que el PIN sea incorrecto, se enviara a una página de error PinIncorrecto.jsp, donde pide al usuario ingresar nuevamente el PIN.

Figura 6. PinIncorrecto

Fuente: Autoras del Proyecto

En caso de que el PIN ingresado 3 veces incorrectamente, se enviará a una página de error TarjetaBloqueada, donde el usuario debe contactarse con el Banco para ser pide al usuario ingresar nuevamente el PIN.

Figura 6. TarjetaBloqueada

Fuente: Autoras del Proyecto

1.1.8 Tipo_Transaccion. El cliente puede seleccionar uno de los 4 tipos de transacciones: Retirar, Depositar, Transferir o Consultar Saldo

Figura 9. Tipo_Transaccion

Fuente: Autoras del Proyecto

1.2 VISTA DE INTERFACES RETIRAR

1.2.1 RetirarCuenta. El usuario puede seleccionar uno de los 3 tipos de cuenta que tiene: Ahorro, Corriente o Empresarial de la cual desee retirar el dinero

Figura 10. RetirarCuenta

Fuente: Autoras del Proyecto

1.2.1 SeleccionarDineroRetirar. El usuario puede seleccionar una de las 4 opciones de cantidad de dinero que puede retirar, 20.000, 60.000, 100.000 o 200.000, validarRetiro,jsp valida que el monto a retirar no sea superior a lo que el cliente tiene en su cuenta, esto se obtiene de la base de datos.

Figura 11. SeleccionarDineroRetirar

Fuente: Autoras del Proyecto

1.2.2 FondosInsuficientesRetirar. En caso de no poder hacer el retiro por fondos insuficientes en la cuenta, se pide al cliente que vuelva a seleccionar la cantidad de dinero a retirar.

Figura 12. FondosInsuficientesRetirar

Fuente: Autoras del Proyecto

1.2.3 ReciboRetirar. Muestra al usuario la cantidad retirada, el recibo de la transacción en donde se muestra la fecha, número de cuenta, tipo de cuenta y el saldo disponible, a si como la opción de realizar una nueva transacción o terminar.

Figura 13. ReciboRetirar

Fuente: Autoras del Proyecto

Si selecciona la opción Si, vuelve a la página de Tipo_Transaccion. (Ver Figura 9. Tipo_Transaccion)

Si selecciona la opción NO, va a la pagina FinTransacciones.

Figura 14. FinTransacciones

Fuente: Autoras del Proyecto

1.3 VISTA DE INTERFACES DEPOSITAR

1.3.1 DepositarCuenta. El usuario puede seleccionar uno de los 3 tipos de cuenta que tiene: Ahorro, Corriente o Empresarial a la cual desee depositar dinero

Figura 15. DepositarCuenta.

Fuente: Autoras del Proyecto

1.3.2 CantidadDineroDepositar. El usuario debe ingresar la cantidad de dinero que desee depositar, luego de esto debe oprimir el botón de Insertar dinero, ValidarDeposito.jsp captura la cantidad ingresada y la suma a la cantidad de la cuenta.

Figura 16. CantidadDineroDepositar

Fuente: Autoras del Proyecto

1.3.3 ReciboDepositar. Muestra al usuario la cantidad depositada, el recibo de la transacción en donde se muestra el nombre del cajero, la fecha, la transacción y el saldo disponible, a si como la opción de realizar una nueva transacción.

Figura 17. ReciboDepositar.

Fuente: Autoras del Proyecto

Si selecciona la opción Si, vuelve a la página de Tipo_Transaccion. (Ver Figura 9. Tipo_Transaccion)

Si selecciona la opción NO, va a la pagina FinTransacciones. (Ver Figura 14. FinTransacciones)

1.4 VISTA DE INTERFACES TRANSFERIR

1.1.4 TransferirDeCuenta. El usuario puede seleccionar uno de los 3 tipos de cuenta que tiene: Ahorro, Corriente o Empresarial de la cual desea hacer la transferencia de dinero

Figura 18. TransferirDeCuenta

Fuente: Autoras del Proyecto

1.1.5 CuentaATranferir. El usuario puede seleccionar uno de los 3 tipos de cuenta que tiene: Ahorro, Corriente o Empresarial a la cual desea hacer la transferencia de dinero

Figura 19. CuentaATranferir

Fuente: Autoras del Proyecto

1.1.6 CantidadDineroTransferir. El usuario debe ingresar la cantidad de dinero que desee transferir

Figura 20. CantidadDineroTransferir

Fuente: Autoras del Proyecto

1.1.7 FondoInsuficienteTransaccion. En caso de no poder hacer la transferencia por fondos insuficientes, se pide al usuario que vuelva a ingresar la cantidad de dinero a transferir

Figura 21. FondoInsuficienteTransaccion

Fuente: Autoras del Proyecto

1.1.8 ReciboTransferir. Muestra al usuario la cantidad transferida, el recibo de la transacción en donde se muestra el nombre del cajero, la fecha, la transacción y el saldo disponible, a si como la opción de realizar una nueva transacción

Figura 22. ReciboTransferir

Fuente: Autoras del Proyecto

Si el usuario selecciona la opción Si, vuelve a la página de Tipo_Transaccion. (Ver Figura 9. Tipo_Transaccion)

Si el usuario selecciona la opción NO, va a la pagina FinTransacciones. (Ver Figura 14. FinTransacciones)

1.5 VISTA DE INTERFACES CONSULTAR SALDO

1.5.1 ConsultarSaldo. El usuario podrá consultar el saldo de cualquiera de los tres tipos de cuenta: Ahorro, Corriente o Empresarial.

Figura 23. ConsultarSaldo

Fuente: Autoras del Proyecto

1.5.2 SaldoAhorro. Se muestra al usuario el saldo actual de la cuenta ahorro.

Figura 24. SaldoAhorro

Fuente: Autoras del Proyecto

Si el usuario selecciona la opción Si, vuelve a la página de Tipo_Transaccion. (Ver Figura 9. Tipo_Transaccion)

Si el usuario selecciona la opción NO, va a la pagina FinTransacciones. (Ver Figura 14. FinTransacciones)

1.5.3 SaldoCorriente. Se muestra al usuario el saldo actual de la cuenta corriente.

Figura 25. SaldoCorriente

Fuente: Autoras del Proyecto

Si el usuario selecciona la opción Si, vuelve a la página de Tipo_Transaccion. (Ver Figura 9. Tipo_Transaccion)

Si el usuario selecciona la opción NO, va a la pagina FinTransacciones. (Ver Figura 14. FinTransacciones)

1.5.4 SaldoEmpresarial. Se muestra al usuario el saldo de la cuenta Empresarial.

Figura 25. SaldoEmpresarial

Fuente: Autoras del Proyecto

Si el usuario selecciona Si, vuelve a la página de Tipo_Transaccion. (Ver Figura 9. Tipo_Transaccion)

Si el usuario selecciona NO, va a la pagina FinTransacciones. (Ver Figura 14. FinTransacciones)

FUNCIONES DE LOS BOTONES

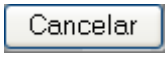
: El botón ON sirve para encender el cajero ATMUNAB, solo es usado por el operario, el cual lo envía a la pagina cargar_cajero.jsp.

: El botón OFF sirve para apagar el cajero ATMUNAB, solo es usado por el operario, el cual lo envía a la pagina index.jsp.

: Teclado numérico, funciona en todas las pantalla del cajero ATMUNAB, con restricciones para cada tipo de pantalla, dependiendo de las opciones o cantidades que se puedan ingresar.

: El botón Insertar Dinero, simula la entrada del dinero en caso de que se desee depositar dinero a una de las cuentas.

: El botón Enter permite enviar los datos capturados de lo que se digite en las cajas de texto.



: El botón Cancelar permite cancelar cualquier tipo de transacción cuando el usuario lo considere necesario, enviándolo a la pagina de FinTransacciones.

: El botón Limpiar permite borrar los datos de las cajas de texto en caso de que sean ingresados de forma incorrecta.

: El botón tarjeta simula el ingreso de la tarjeta al cajero ATMUNAB, enviándolo a una página de Insertar, donde pide al usuario insertar el numero de la tarjeta.

: El botón Log permite ver todos los registros realizados en el cajero ATMUNAB, como transacciones, cancelación de transacción, etc.