PROYECTO DE GRADO

CAPACIDAD DE SOPORTE Y CONTROL DEL TRAZADO DE ASUNTOS TRANSVERSALES EN ETAPAS TEMPRANAS DEL CICLO DE VIDA

REQUISITO PARA OPTAR AL GRADO DE INGENIERIA DE SISTEMAS

IVONNE MELISA CALLEJAS GALVIS
199920021010

MARIANA LUJAN SANSON
200310081010

ASESOR
JOSE ARLEY MEJIA

UNIVERSIDAD EAFIT
DEPARTAMENTO DE INGENIERA DE SISTEMAS
MEDELLIN
2009

Dedicatoria:

Este trabajo se lo dedico a Marta Silvia Tabares, Nohelia Galvis y Juan David Gaviria.

Ivonne

Este trabajo se lo dedico a mis padres: Gabriel Lujan y Gloria Sanson, a mi mayor apoyo en EAFIT: Juan Luis Mejía Arango y a mi apoyo incondicional: Juan David Sánchez Vargas.

Mariana

Agradecimientos

A Dios por permitirme llegar al final de este proyecto, a Mariana porque sin ella no lo hubiese logrado, a mi madre por todo su apoyo durante el transcurso no solo de mi carrera sino de toda mi loca vida, a mi novio por su apoyo y en especial un agradecimiento a Marta Silvia Tabares por toda su paciencia durante el tiempo que estuvo conmigo.

Ivonne

Dios gracias por haberme permitido ver este día y este trabajo, a Ivonne por ser mi amiguita y compartir conmigo este trabajo, a mis papás por su ejemplo y apoyo, a mi novio por ser la persona más maravillosa e incondicional y que me apoyo en toda esta locura y a todas las personas que me ayudaron y apoyaron a lo largo de mi carrera, a quienes compartieron todo este camino, Gracias!!.

Mariana

Tabla de Contenido

Tabla de Figuras

Índice de Tablas

Tabla de anexos

GENERALIDADES

En este capítulo se presenta el problema a ser abordado por este trabajo de grado y se especifican los objetivos.

1.1 DESCRIPCIÓN DE LA PROBLEMÁTICA

En algunos de los procesos de software, establecer la trazabilidad surge de la necesidad de poder hacer un seguimiento al cumplimiento de los requisitos a lo largo del ciclo de vida. Esto debido al hecho de ver con frecuencia que, a medida que el desarrollador avanza en sus tareas de diseño y codificación, se va dificultando la ubicación de los elementos del sistema que deben ser modificados ante una necesidad de cambio en los requerimientos de un sistema que ya se encuentra en operación.

Desde el punto de vista de la orientación a objetos, no siempre es posible mantener la traza claramente para capturar de forma aislada la transformación de asuntos que no son ortogonales o que cortan transversalmente otros módulos. Esto lleva a inferir que, un cambio sobre un asunto que corta a otros asuntos se realizaría de forma invasiva en la descripción de ellos y estará altamente acoplado. Además habrá una alta probabilidad de eliminación o adición descontrolada de elementos o artefactos de software al descomponer y componer modularmente, no pudiendo controlar situaciones tales como descartar elementos de los módulos considerados “poco significativos” en el nivel de diseño, sin haber sido contrarrestados con estructuras de módulos altamente cohesivos pero con comportamiento semejante.

Con base en lo anterior, tratar la trazabilidad desde el enfoque AOSD, pretende dar fiabilidad a sistemas de software, orientando al desarrollador a identificar la traza desde los asuntos transversales cuando estos afectan otros asuntos del sistema. Así, se hace necesario identificar el impacto del cambio que motiva la evolución del sistema validando algunas situaciones tales como:

- ¿Qué asuntos transversales son afectados por un escenario de cambio?

- ¿Qué modelos y artefactos están o estarán asociados a los asuntos afectados?
- ¿Cuál es el nivel de granularidad donde los asuntos y sus artefactos son afectados directamente?
- ¿Qué reglas de composición se hace necesario modificar, crear o eliminar?
- ¿En qué sentido, una traza hacia atrás puede proveer elementos aspectuales que obliguen a la construcción de asuntos o artefactos no identificados en etapas tempranas?
- ¿A qué asuntos transversales candidatos sólo es posible llevarles la traza en algunas etapas del ciclo de vida?
- ¿Existen semánticas apropiadas para llevar la trazabilidad de los asuntos a lo largo del ciclo de vida sin importar los modelos de descomposición y composición utilizados en la construcción del sistema?
- ¿Puede afectar la trazabilidad el hecho de que nuevos artefactos se consideren subsistemas del sistema de software actual?

1.2 OBJETIVO GENERAL

Analizar la capacidad de soporte y control de trazado de asuntos transversales desde las aproximaciones de asuntos transversales en etapas tempranas de desarrollo.

1.3 OBJETIVOS ESPECÍFICOS

- Estudiar los modelos de trazabilidad más representativos en la ingeniería de software.
- Determinar las características generales que se deben de tener en cuenta para la trazabilidad de requisitos en las diferentes etapas del ciclo de vida de desarrollo.
- Seleccionar las aproximaciones de Aspectos tempranos en el contexto del desarrollo de software orientado a aspectos [1], sobre las cuales se analizará la práctica de la trazabilidad.
- Desarrollar un caso de estudio para mostrar la capacidad de trazado de las aproximaciones de asuntos transversales en etapas tempranas de desarrollo por medio de las aproximaciones seleccionadas para estudio de forma que podamos conocer cómo son aplicadas las características de soporte y control al trazado de asuntos transversales en etapas tempranas del ciclo de vida.

1.4 JUSTIFICACION

Poder soportar y controlar trazabilidad como un asunto transversal en etapas tempranas de desarrollo, abre posibilidades prometedoras para garantizar la consistencia del proceso de modelado, analizar el impacto de los cambios y facilitar las pruebas del software. Un manejo homogéneo de la trazabilidad desde los requisitos transversales permitirá identificar posibles efectos colaterales en las etapas de diseño e implementación en características como rendimiento, persistencia, disponibilidad, entre otros.

Por esto, se hace necesario estudiar y analizar los modelos y artefactos (qué y cómo) que el desarrollo de software orientado por aspectos (AOSD)[1] pone a disposición para seguir la traza de los asuntos transversales en las etapas tempranas de desarrollo (análisis y diseño).

1.5 ALCANCE

El alcance este proyecto está determinado por la entrega de los siguientes productos:

- Análisis del estado del arte orientado a la trazabilidad, de algunas aproximaciones, de asuntos transversales en etapas tempranas de desarrollo contenidas en [2]. Una (1) de requisitos y una (1) de diseño.
- Caso de estudio desarrollado sobre dichas aproximaciones.
- Propuestas de soporte y control del trazado en las aproximaciones elegidas (Análisis de la Trazabilidad).
- Conclusiones finales.

2 CONCEPTOS GENERALES DE TRAZABILIDAD

La trazabilidad de los requisitos es una práctica que no se encuentra presente de forma rigurosa en los desarrollos de software. La mayoría de los desarrolladores realizan modificaciones sin tener en cuenta el impacto del cambio, debido a que comúnmente el desarrollador tiene plena confianza de su conocimiento total de la aplicación; es él quién la está implementando y este valor de confianza lo impulsa a creer que una modificación puede ser tratada sin necesidad de realizar cambios en la documentación que se tiene para sustentar el sistema.

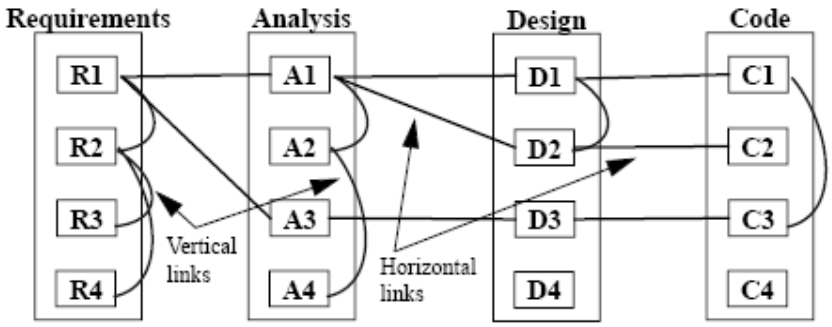
Es necesario entonces resaltar la importancia de una buena trazabilidad en los requisitos. El problema de no realizar esta práctica se verá reflejado en el futuro de la aplicación, cuando se presenten errores y no sea posible localizarlos fácilmente, cuando la mantenibilidad de la aplicación no sea tan sencilla pues se tiene desconocimiento del total de cambios realizados en la aplicación y en una situación más crítica cuando el desarrollador de la aplicación no se encuentre y sea necesario asignar a alguien más para darle soporte a la aplicación.

Por las razones planteadas anteriormente, para que un producto sea altamente mantenible es necesario que el analista de desarrollo conozca a profundidad todos los aspectos o artefactos que componen el producto y deje rastro de ellos en un documento que sirva como soporte para modificaciones venideras de la aplicación, a esto le llamamos Trazabilidad.

Se define como trazabilidad la “habilidad de crear o realizar trazas entre los componentes de un producto ya sea en un mismo sistema o entre varios sistemas” [3]

Lindvall dice que la trazabilidad se puede representar como grafos dirigidos utilizando nodos y arcos. Cada componente, cada requisito o parte de código puede ser representado como un nodo y cada vínculo o dependencia entre

ellos es denotado por un arco. Dadas las relaciones que existen entre los diferentes artefactos el resultado de estos grafos se asemeja a una red compleja en la que se pueden visualizar los diferentes caminos o trazas requeridas para llegar a otros nodos y son estas trazas las que indican la dependencia o no entre ellos [4]. En la figura 1 se muestra un grafo que representa nodos como productos en los diferentes niveles del ciclo de vida de desarrollo y arcos para indicar el trazado entre ellos.



.

Figura 1 Gráfico de la trazabilidad
Tomada de (Pfleeger.Bohner, 1990) y una extensión realizada.

Generar trazas hace posible el seguimiento por parte del desarrollador de la forma en cómo está diseñada y estructurada una aplicación, es así como se genera confianza a la hora de realizar cambios que puedan alterar el comportamiento de la aplicación, ya que el analista tiene la posibilidad de saber qué partes del producto se verán afectadas por los cambios que se realicen.

Para los productos de software en general se hace necesario una trazabilidad profunda entre los artefactos que componen la aplicación, un ejemplo claro de esto y es una situación que se presenta de forma muy común en las empresas desarrolladoras de software, es el momento de la elicitación de los requisitos. El cliente en las reuniones iníciales solicita lo que él desea que realice el sistema de forma que solucione por medio de tecnología el problema con el que cuenta actualmente. Una vez identificados los requisitos, el analista procede al desarrollo, sin embargo el cliente puede en cualquier momento cambiar de opinión y solicitar un nuevo requisito, restricción o regla sobre algo que ya esta especificado, diseñado o implementado. Un cambio como estos

implica tener conocimiento del impacto sobre los productos desarrollados, refiriéndonos con esto no sólo a qué parte del sistema se ve afectada y que tan grande es el cambio, sino también la necesidad de verificar el impacto en tiempo y dinero, que son los dos factores más importantes y determinantes a la hora de realizar un cambio de último momento solicitado por el cliente.

Si se cuenta entonces con una buena trazabilidad es posible realizar un seguimiento al requisito antiguo verificando qué artefactos se ven afectados por el cambio y poder así estimar el impacto del mismo en el sistema. La trazabilidad también permite que un producto sea mantenible en su totalidad aun cuando aquella persona que lo diseñó y desarrolló no se encuentra ya para darle el soporte requerido, esto permite que cualquier otro analista pueda realizar cualquier cambio o realizar algún mantenimiento basándose únicamente en las trazas definidas para el sistema.

La trazabilidad tiene varios stakeholders o participantes que indican el flujo y la validez de cada uno de los requisitos, sin embargo es importante tener en cuenta que estas opiniones pueden llegar a generar confusión sobre los requisitos del sistema. Así, es importante la realización de una buena elicitación con buenas técnicas que ayuden a la creación de las trazas desde el mismo instante en que se especifican los requisitos con el cliente para así poder generar un producto con un porcentaje de mantenibilidad alto.

En el momento de realizar pruebas a un producto la trazabilidad se convierte en un elemento sumamente importante ya que se puede visualizar los posibles problemas que afectan el producto desde el principio hasta el fin y adicionalmente se puede realizar un seguimiento completo de los cambios y el impacto de los mismos en el producto desarrollado.

Ver la trazabilidad como una actividad que ayuda a disminuir costos y tiempo es una de las más grandes ventajas que se tiene, ya que esto contribuye con la mejora del proceso de desarrollo dando como resultado un producto de buena calidad y que satisface las solicitudes realizadas por el cliente.

2.1 DIMENSIONES DE TRAZABILIDAD

Existen dos tipos de trazas, horizontales y verticales. Las trazas horizontales se dan desde la fase de elicitación, a partir de los requisitos presentados por el cliente, se puede generar trazas para el diseño hasta la implementación de forma que se cumpla con el requisito especificado. Por otra parte las trazas verticales se dan al interior del producto para trazar la relación entre los requisitos de diferentes niveles.

La trazabilidad tiene diferentes niveles, esto implica que se puede realizar entre diferentes modelos a cierto nivel de detalle, pero no a un nivel bastante detallado. Un ejemplo de esto se puede observar en los sistemas orientados a objetos donde se hace posible realizar trazas a nivel objetual, pero no permiten la captura completa de las trazas que involucran varios módulos, esto implica que puede existir la posibilidad de eliminar artefactos de forma no controlada dando como resultado la posibilidad de desarrollar elementos que no son necesarios para el sistema pues fueron identificados en el diseño como elementos poco significativos [4]

La trazabilidad tiene también dos formas: la implícita y la explícita. Por medio de la trazabilidad explícita se pueden generar trazas usando vínculos de trazabilidad predefinidos entre ítems de dos modelos. La trazabilidad implícita es lo opuesto a la explícita y ocurre precisamente cuando ésta última no se da, la trazabilidad implícita contribuye por ejemplo a verificar que los modelos no se contradigan entre sí.

De acuerdo con el modelado de Objetos, un sistema inicia su construcción con la especificación de los requerimientos siendo este uno de los más importantes artefactos para el desarrollo de un sistema, para esto el modelado de casos de uso es una de las varias herramientas que se pueden utilizar al momento de la elicitación. Con base en esto entonces podemos decir que inicialmente construimos un modelo basado en un análisis de requisitos, después de esto realizamos un refinamiento a los requisitos y finalmente implementamos el sistema. El resultado de este proceso son los siguientes modelos:

- Modelo de Requisitos
- Modelo de Análisis
- Modelo de Diseño
- Modelo de Implementación.

Cada modelo describe y resuelve diferentes problemas durante las diferentes fases del ciclo de vida de desarrollo, el modelo y las reglas para la transformación de un modelo en otro pueden ser descritos en un esquema de trazabilidad.

La figura 2 muestra la forma de un esquema de trazabilidad objetual:

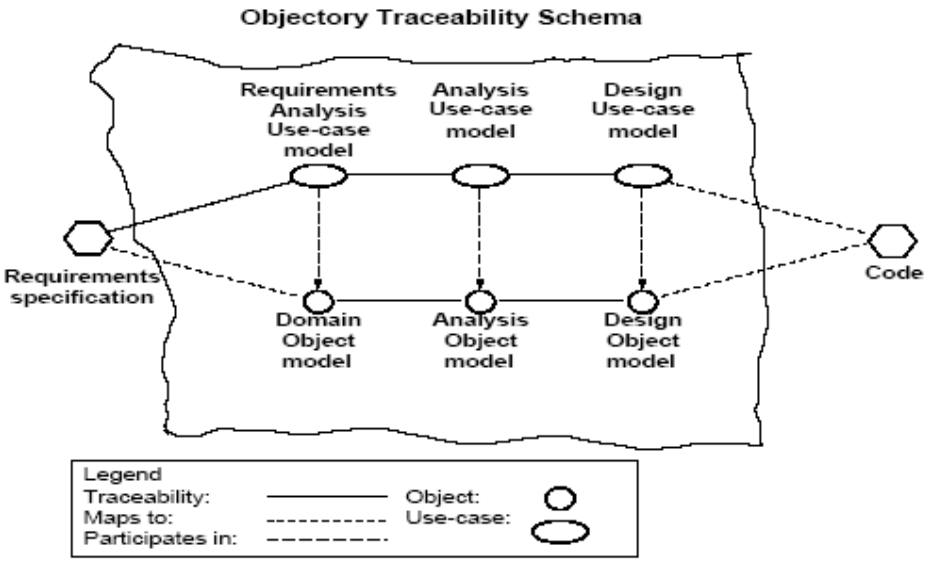


Figura 2.Esquema de trazabilidad de Objectory
Muestra cómo los modelos están relacionados unos con otros. La figura también muestra cómo los modelos no manejados por Objectory, tales como la especificación de los requerimientos y la implementación del sistema son relacionados.

En general podemos decir que la trazabilidad entre artefactos de un sistema es la forma básica para el soporte, mantenimiento y entendimiento de un desarrollo de software. Existen distintas formas de modelar sistemas pero lo más importante es conservar el conocimiento de las relaciones que existen entre los diferentes elementos que lo componen, éste es obtenido mediante

una buena elicitación y creación de trazas desde el inicio del ciclo de vida de desarrollo de un software.

Existen en la actualidad varias orientaciones que son importantes a la hora de realizar una correcta trazabilidad entre los diferentes artefactos de un sistema, Lindvall por ejemplo plantea en uno de sus artículos la orientación en dos dimensiones [4], por medio de la primera dimensión se puede distinguir entre los ítems trazados de forma específica clasificándolos así:

- Trazabilidad Objeto – Objeto
- Trazabilidad Asociación – Asociación
- Trazabilidad Caso de Uso – Caso de Uso
- Trazabilidad de dos dimensiones Objeto – Objeto

La segunda dimensión de trazabilidad se refiere a la forma como las trazas actúan:

- Trazabilidad vía links explícitos los cuales son parte del ambiente de desarrollo.
- Trazabilidad usando referencias los cuales definen referencias textuales de diferentes documentos.
- "Name tracing" asume una estrategia consistente y es usada cuando se construyen modelos, ésta actúa buscando ítems con nombres similares a los que se encuentran en el modelo inicial.
- Conocimiento del Sistema y Dominio del Sistema pueden ser usados por un desarrollador de software experimentado, trazando conceptos con su conocimiento acerca de cómo están interrelacionados los diferentes ítems.

Se presentan también cuatro ejemplos que muestran distintos aspectos de la trazabilidad. En el primero se muestra un ejemplo simple de una trazabilidad uno a uno, Objeto – Objeto, en el segundo se hace énfasis en el rol de la trazabilidad asociación – asociación, en el tercero se crean trazas de un caso

de uso a los objetos de análisis participativos y a los objetos de diseño participativos y adicionalmente esto se realiza también entre los dos modelos.

A continuación una explicación breve de los distintos aspectos de la trazabilidad.

2.1.1 Trazabilidad Objeto – Objeto

Se presentan acá la forma como son trazados cinco importantes conceptos a través de varios modelos objeto, para esto se realiza una división en 4 pasos:

1. Trazar los conceptos relacionados
2. Trazar los objetos del dominio
3. Trazar los objetos de análisis
4. Trazar los objetos de diseño

Por medio de estos pasos se puede entender como los objetos de diseño del sistema están relacionados al concepto conexión. Se puede observar también que la trazabilidad se facilita a medida que los ítems del modelo correspondan de forma clara y se utiliza una estrategia de nombramiento adecuada.

2.1.2 Trazabilidad Asociación - Asociación

Se presenta la necesidad de una trazabilidad asociación – asociación cuando se comparan diferentes modelos para propósitos de trazabilidad. Si dos modelos difieren en estructura y nivel de detalle esto implica dificultades para verificar su actual correspondencia o no.

Para esto es necesario realizar una división en cinco pasos:

1. Trazabilidad de los objetos de análisis y las asociaciones
2. Trazabilidad de las asociaciones de los objetos de diseño

3. Trazabilidad de las asociaciones
4. Asociación con Dirección Invertida.
5. Comparación de “object reachability”

Se muestra entonces la necesidad de expresar trazabilidad entre asociaciones así como también entre objetos.

2.1.3 Trazabilidad Caso de Uso – Objeto

Para este tipo de trazabilidad se realizan trazas de un caso de uso a los objetos que participan en él. Los diferentes modelos de casos de uso usan diferentes tipos de objetos, sin embargo es posible crear trazas de un caso de uso específico a sus objetos de análisis y a sus objetos de diseño. Los últimos pasos de este ejemplo son una comparación del set de objetos trazados desde un mismo caso de uso.

Para esto se realiza una división en cinco pasos:

1. Trazar un caso de uso
2. Trazar un objeto de análisis
3. Trazar objetos de diseño
4. Trazar objetos de análisis a objetos de diseño
5. Comparar los dos modelos de objetos

2.1.4 Trazabilidad en dos dimensiones

Se examinan acá los link de trazabilidad Muchos a Muchos (n-n). En el análisis del modelo objetual se encontró que muchos objetos tienen relación con “Recording”.

El resultado de las trazas fueron dos objetos de diseño: PMR_CollectorElement y PRM_OrderElement. Por medio de esto se llegó a la conclusión que la

técnica de modelamiento utilizada para el análisis fue muy distinta a la utilizada durante el proceso de diseño.

Durante el análisis los desarrolladores construyeron modelos con la intención de que éstos se basaran en el fonema real que es el centro de la aplicación de acuerdo con los principios generales de varias metodologías de modelamiento orientado a objetos. Durante el diseño fue adoptada una metodología diferente, acá se modeló el fonema real y su estructura, sus roles en el PMR-system fueron modelados también.

Una técnica de modelamiento que enfatiza en las correspondencias del modelo con el mundo real puede resultar en un modelo compuesto por un núcleo y sus partes físicas. Cuando se aplica el modelamiento por medio de roles es importante considerar en qué contexto está construido el modelo.

Es necesario también conocer que la trazabilidad implica mejora de los procesos, sin embargo es conveniente reconocer que un buen análisis implica una buena elicitación de requisitos y una buena trazabilidad de los mismos, para esto Orlena C. Z. Gotel & Anthony C. W. Finkelstein [5] proponen una división de la trazabilidad que implica un seguimiento hacia delante y hacia atrás en el ciclo de vida de desarrollo de un producto. Estas divisiones son llamadas así: Pre-RS traceability y Post-RS Traceability, la primera se refiere a los aspectos de la vida útil de un requerimiento antes de ser incluido en la especificación de los requisitos, y el segundo implica aquellos requisitos que se presentan como resultado de la inclusión en la especificación de los mismos.

Adicionalmente otros autores [6] presentan herramientas que facilitan el trazado de requisitos donde se especifica la funcionalidad de una herramienta llamada TOOR la cual permite realizar trazados de requisitos implicando el hecho que no sólo los factores técnicos juegan un papel crucial en el desarrollo de software, sino que también factores sociales pueden cambiar de forma significativa el desarrollo de un producto [5]. La herramienta propuesta expone tres modelos de trazabilidad:

- Trazado selectivo: restringe la traza a ciertos patrones seleccionados de objetos y relaciones

- Trazado interactivo: permite una búsqueda interactiva a través de los objetos relacionados con el objeto seleccionado hacia atrás y hacia adelante del mismo.

- Trazado no guiado: permite al usuario ir de un objeto a otro inspeccionando los contenidos de la forma en que lo requiera.

A la fecha existe una cantidad considerable de documentos y herramientas que tratan de dar una solución confiable al problema de la trazabilidad en los requisitos, pero pocas logran su cometido, pues usan una técnica similar y su única diferencia radica en las interfaces, el tiempo y el esfuerzo que se emplea para generar los requisitos.

Se puede observar por ejemplo que la mayoría de la técnicas ofrecen herramientas que facilitan el desarrollo de la trazabilidad, algunas de ellas como: PROBE, TOOR y ARCADE. Es necesario tener claro que en algunos de los modelos se hacen importantes ciertos artefactos del análisis y el diseño, en [3] el actor o usuario final es la base de la fundamentación teórica y es este quién identifica y documenta la información que es relevante al proyecto, sin embargo sigue siendo compleja la forma de localizar y acceder a la fuente de los requerimientos, importante también es detallar por ejemplo el elemento base para una trazabilidad, según [6] las relaciones entre los artefactos son básicas para el entendimiento apropiado de los requisitos y a diferencia de muchos otros modelos estas relaciones no son simples links, sino que son referenciadas por axiomas y esto permite la obtención de una mayor flexibilidad en los requisitos que se tienen para el desarrollo.

Se habla entonces de muchos artefactos obtenidos durante la etapa de análisis y diseño, su definición y su importancia radica según el modelo que se tome pero por lo general siempre la base de todos ellos son las relaciones y los actores involucrados en el sistema, pues son ellos lo que a la final definen el

comportamiento que debe tener el desarrollo que se va a implementar. Es necesario tener en cuenta también la importancia de una buena toma de requisitos [5]. La trazabilidad de las pre-especificaciones y pos-especificaciones de requisitos son esenciales a la hora de un buen diseño y una buena implementación.

3 TRATAMIENTO DE LA TRAZABILIDAD DESDE EL DESARROLLO DE SOFTWARE ORIENTADO POR ASPECTOS

En este capítulo presentan diferentes teorías que fundamentan el desarrollo de software orientado por aspectos. A partir de eso, se identifican las diferentes aproximaciones AOSD, en las fases de requisitos, diseño y arquitectura para estudiar la forma como hacen la trazabilidad de requisitos y sus artefactos en diferentes fases. En el estudio de cada una de ellas se identifica la forma como manejan los artefactos que posiblemente pueden ser trazados en una o varias fases del ciclo de vida. Para lograr esto se presenta de forma corta el objetivo logrado por cada aproximación, se ilustra un mapa conceptual que permite identificar los artefactos o elementos de modelo que pueden generar vínculos de trazo y se analiza la forma como entre ellas pueden crearse vínculos de trazado

3.1 La Orientación a Aspectos

Durante varios años ya se ha venido introduciendo la separación de concerns de un sistema, de forma que se pueda hacer más manejable la complejidad de un sistema, estos elementos pueden ser características funcionales o no funcionales. Habitualmente la aplicación final se construye con el código de propósito específico y el código de las funcionalidades principales o concerns [i]

Teniendo claro lo anterior, definimos un aspecto como una abstracción con la que se modela un concern y que puede ser utilizada en uno o varios de ellos.

3.1.1 La Programación Orientada por Aspectos

Es una metodología nueva cuya idea fundamental es utilizar la separación de concerns en todas las etapas del ciclo de desarrollo de software, de esta forma se permite un mejor manejo y mantenimiento del software desarrollado.

3.1.2 El Desarrollo de Software orientado por Aspectos

El desarrollo de software orientado a aspectos (AOSD) provee un conjunto de enfoques para identificar, modularizar e implementar intereses o propiedades del sistema que pueden cruzar otros intereses del sistema, también busca mejorar el entendimiento de cada interés del sistema de forma clara y separada desde las primeras etapas del ciclo de vida de software. Este se orienta a la obtención de productos de software de calidad con partes más reutilizables y que evolucionen fácilmente en el tiempo. [7]

3.2 Análisis de la Trazabilidad en las Aproximaciones Orientadas a Aspectos [2]

3.2.1 Identificación de las Trazas en el ciclo de vida

A continuación se presenta una tabla en la que se indica el origen de los artefactos de cada una de las aproximaciones y la forma en cómo se realiza el rastreo de la trazabilidad.

Modelo de Trazabilidad (Aproximación)	Origen del artefacto	Tracking del artefacto a través del ciclo de vida
Use cases	Actor en el diagrama de Casos de Uso.	Diagramas de Colaboración
AORE with ARCADE	Viewpoint de requisitos.	Mapping decisions, PROBE
ARGM	Softgoal Interdependency Graph para los sub goals de los softgoals.	Desing decision y operationalisation
AOSD/UC	Actores en el diagrama de casos de Uso.	Diagramas de Colaboración, use case slices y modules
SMA	Actores en el diagrama de casos de Uso, para NFR la fuente está en los Templates.	
AUCDA	Actores en el diagrama de casos de Uso, para NFR la fuente está en los Templates.	Diagramas de Colaboración
CORE	Abstract concerns en el espacio de los meta concerns.	Mapping decisions

AOREC		Aspectos en el nivel de diseño
Theme/Doc		Directamente entre los requisitos y su modelo de diseño

Tabla 1. Origen de los artefactos
Fuente: [2]

Un criterio que es básico para el desarrollo de una trazabilidad completa es la forma en cómo se componen los artefactos de un sistema, para esto entonces debemos tener claro que Composición o *Composability* es la habilidad de componer artefactos y consecuentemente ver y entender un set completo de artefactos y sus interrelaciones, así como la percepción del sistema como una unidad de artefactos integrados [4]

Modelo de Trazabilidad (Aproximación)	Artefactos que soportan la Composición
Use cases	Relaciones de Extend e Include.
AORE with ARCADE	Reglas de composición flexibles y extensibles y operadores, unicidad de Ids para los viewpoints, requisitos y sub-requisitos.
ARGM	Tópicos comunes de goals y task.
AOSD/UC	Extension pointcuts, relaciones extend e include en los casos de uso
SMA	Binding definitions, eventos de entrada y salida, estados y parámetros de rol
AUCDA	Relaciones de Collaborate, damage, constrain, extend e include.
CORE	Reglas de composición flexibles y extensibles y operadores, unicidad de Ids para los viewpoints, requisitos, sub-requisitos, concern projections y compositional intersections.
AOREC	Pre-component provided/required spect details y provided/required links.

Theme/Doc	Orden para la composición de los temas en los Clipped action view.

Tabla 2. Modelos y artefactos

Fuente: [2]

Adicionalmente para el seguimiento correcto de los artefactos de un sistema debemos también tener presente que estos a medida que se avanza dentro del ciclo de vida pueden sufrir transformaciones o evoluciones, éstas también deben ser trazadas dentro del proceso y para esto es necesario conocer en que evolucionan o se transforman los artefactos iníciales de forma que los podamos reconocer durante el camino de su cambio y poder rastrearlos a cualquier nivel.

A continuación una tabla donde se presentan las distintas aproximaciones estudiadas previamente y las características que en ellas soportan la evolución o cambio de los artefactos.

Modelo de Trazabilidad (Aproximación)	Evolución o refinamiento de los artefactos
Use cases	Use case decomposition, notación minimalista para diagramas de caso de uso
AORE with ARCADE	Separación de concerns y composición de concerns, tablas de referencias cruzadas.
ARGM	catálogo de correlación descomposición de sub-goals, link entre goal-goal-task
AOSD/UC	Use case decomposition, notación minimalista para diagramas de caso de uso, use case slices y modules
SMA	Elementos de rol genéricos, bindings.
AUCDA	NFR templates, elementos de rol genérico, bindings.
CORE	Separación y composición de concerns, tablas de referencia cruzada.
AOREC	Aspect details.
Theme/Doc	Automatización otorgada por la herramienta.

Tabla 3 Aproximaciones – Evolución de los artefactos

Fuente: [2]

Para tener un conocimiento certero de cómo están moviéndose a través del ciclo de vida los distintos artefactos que componen un sistema, se hace necesario también realizar un mapeo de los mismos de forma que tengamos oportunidad de reconocerlos y de conocer su funcionalidad dentro del sistema.

A continuación se presenta una tabla en la que se muestran que características de las distintas aproximaciones soportan el mapeo.

Modelo de Trazabilidad (Aproximación)	Características que soportan el mapeo
Use cases	Diagramas de Colaboración y algunos guidelines.
AORE with ARCADE	Guidelines.
ARGM	Tipos y catálogos de descomposición, operacionalizaciones, links de requisitos funcionales
AOSD/UC	Diagramas de Colaboración y algunos guidelines, pointcuts, aspectos.
SMA	
AUCDA	Diagramas de colaboración y algunos guidelines.
CORE	Guidelines
AOREC	Uso de los mismos aspectos y detalles de los requisitos y los distintos niveles de diseño.
Theme/Doc	Acercamiento de requisitos y diseño de modelos, Theme view de la herramienta de Theme/Doc tool.

Tabla 4. Características de las distintas aproximaciones que soportan mapeo.
Fuente: [2]

Con base en lo anterior, a continuación se hace estudio de algunas de las aproximaciones más representativas para cumplir nuestro objetivo y en cada

una de ellas se ilustra su modelo conceptual y la forma como se relacionan los artefactos para lograr trazabilidad.

3.2.2 AO-REQUIREMENTS ENGINEERING

A continuación se presentarán algunos de los modelos que pueden ser aplicados a la etapa de elicitación de requisitos

3.2.2.1. AORE with Arcade

Es una de las primeras aproximaciones que introduce el concepto de aspectos en el nivel de requisitos, es decir dentro de la etapa de análisis al momento de la elicitación. Esta aproximación propone una técnica para separar requisitos aspectuales y no aspectuales, así como sus reglas de composición.

El objetivo de esta aproximación es modularizar y componer los concerns a nivel de requisitos, produciendo no solamente un documento de especificación sino asegurando la consistencia del mismo, por medio de la identificación de conflictos basado en la composición de requisitos.

Esta aproximación permite entonces satisfacer la necesidad recurrente de realizar un balance entre formalizar los concerns, viewpoints, requisitos, operadores de composición y reglas de negocio y mantener los artefactos simples de forma que sean asequibles al usuario, para esto AORE utiliza XML para la representación de artefactos, esto permite mantenerlos estructurados, semi-formalizados, simples y entendibles.

Adicionalmente AORE utiliza un framework desarrollado para establecer lazos entre requisitos y sus artefactos en las etapas posteriores del ciclo de desarrollo, dicho framework es llamado PROBE y recibe como parámetros de entrada, el documento de requisitos y la resolución de conflictos generada por la aproximación, esto genera Proof Obligations para los artefactos y soporta el

trazado de éstas a través de todas las etapas del ciclo de vida de desarrollo de software.

3.2.2.1.1 *Cómo se puede tratar la trazabilidad desde esta aproximación*

Los elementos de modelo disponibles para el trazado son:

- Viewpoints
- Concerns
- Requisitos
- Composition Rules: Requirements Viewpoints, outcome action, constraint action (elemento de composición)

De acuerdo al análisis realizado por [2] con respecto a esta aproximación, se observa que existen vínculos de trazado con otras aproximaciones que son no aspectuales. Estos posibles vínculos son:

- Requirement viewpoint *es trazado a* un use case
- Outcome action *Contribuye a* Una operación de una clase

También en este mismo análisis se definen vínculos de trazado a otras aproximaciones que si son aspectuales las cuales son:

- En la Arquitectura
- En el Diseño

Ver Anexo 1: Mapa Conceptual Método AORE

3.2.2.2. Aspects in Requirements Goal Models (ARGM)

Por medio de esta aproximación los aspectos se obtienen de las relaciones dadas entre “goals” funcionales y no funcionales, por medio de la descomposición de estos en “sub-goals” y “sub-softgoals” pudiendo identificar finalmente los aspectos.

La descomposición comienza con un procedimiento llamado “Aspect finder” el cual toma un grupo de nodos de “goals” y “soft-goals” e iterativamente los

descompone, luego los relaciona y finalmente encuentra los conflictos entre ellos.

Se utilizan en esta aproximación Gráficos de Interdependencia para detectar los aspectos, en este caso serán llamados tareas. El gráfico comúnmente utilizado es V-Graph usado para representar las relaciones entre goal-softgoal-aspect.

Ver Anexo 2: Mapa conceptual ARGM

3.2.2.3. Aspect-Oriented Software Development with Use Cases (AOSD/UC)

Esta aproximación es una extensión de la aproximación por Casos de Uso, utiliza dos elementos principales:

- ***Pointcuts:*** Se agrupan casos de uso representados por *extensión points y* los elementos como clases, operaciones etc., son representados por casos de uso. Los *extensión points* facilitan la identificación de los pointcuts en la etapa de diseño.
- ***Use Case Slices:*** contienen detalles de un caso de uso en una etapa del desarrollo de software dada y un *use case module* contiene todos los detalles relacionados con el caso de uso, a través de todas las fases del desarrollo de software.

Adicionalmente identifica dos tipos de casos de uso:

Peer: *son distintos e independientes unos de otros, es decir que pueden ser utilizados de forma separada*

Extensión: *Son las características adicionales de los casos de uso, y pueden ser definidos de forma independiente.*

Ver Anexo 3: Mapa conceptual AOSD/UC

3.2.2.4. Scenario Modeling with Aspect Artifacts

Esta aproximación se enfoca en el efecto producido por los aspectos en el comportamiento del modelamiento, tomando como base la visión de UML, de esta forma se ayuda a mejorar la forma de desarrollo de tal manera que sea más completa y con escenarios más consistentes evitando así el hecho de tener que lidiar con escenarios que contiene fallos y excepciones.

Los escenarios aspectuales son representados por medio de Interaction Pattern Specification (IPS) los cuales describen un patrón de interacción entre sus participantes en términos de roles que el participante debe cumplir, para esto se define dentro de la aproximación un pipeline () el cual indica que el rol necesita ser llenado por los participantes instanciados en el patrón.

Los escenarios aspectuales son traducidos de IPS a una máquina de estados de especificación de patrones (SMPS) la cual se comporta de forma similar a una maquina de estados de UML pero ésta tiene en cuenta los elementos de rol que deben ser llenados, SMPS es usada únicamente para escenarios aspectuales, las máquinas de estado finito sencillas (FMS) son utilizadas para escenarios no aspectuales. Es importante tener presente que los roles que sean definidos en SMPS necesariamente deben ser mapeados a elementos de FMS.

Ver Anexo 4: Mapa conceptual Scenario Modeling with aspect artifacts

3.2.2.5. Aspectual Use Case Driven Approach

Para esta aproximación se presentan similitudes con AOSD/UC, dado que ésta también realiza una separación de los crosscutting funcionales, dividiéndolos en: Inclusión y Extension use cases. Una vez separados los casos de uso estos deben ser integrados en un modelo completo usando los mecanismos de composición.

La aproximación también sugiere que modelar como parte del sistema los atributos de calidad que son tan importantes en el desarrollo del software, dichos atributos de todas formas, no deben obligatoriamente estar ligados a un caso de uso de infraestructura.

Ver Anexo 5: Mapa conceptual Aspectual Use Case Driven Approach

3.2.2.6. Concern-Oriented Requirements Engineering (CORE)

Esta aproximación es una adaptación de AORE, la cual presenta la descomposición de los requisitos de acuerdo a su naturaleza funcional o no funcional.

CORE es representado por medio de un hipercubo, cada cara refleja un concern en particular, todos los concerns son tratados de igual manera lo que permite que cualquier set de concerns puede ser seleccionado como base del proyecto, de esta forma entonces se puede ver que la aproximación soporta separación multidimensional de concerns, el establecimiento de Trade-offs sobre crosscutting y requisitos traslapados.

Adicionalmente CORE define dos nuevas nociones:

- ***Meta Concern Space:*** comprime todos los concerns (funcionales y no funcionales) que se presentan en varios sistemas, de esta forma entonces tenemos que un meta concern space es un catalogo de concerns que aparecen una o varias veces en varios desarrollos de software, que pueden ser clasificados así como también es posible definir las relaciones entre ellos y sus formas de utilización.
- ***Compositional intersection:*** es un concepto muy poderoso dentro del modelo, este provee una restricción a un set de concerns que puede ser usado como base para la proyección de concerns durante la interacción y el análisis de trade-off.

Es importante resaltar que en CORE los concerns implican una colección coherente de requisitos que son identificados empleando perspectivas multidimensionales. Una vez identificados todos estos concerns se inicia con la construcción de una matriz de contribución por medio de la cual se detectan conflictos se asignan prioridades y se toman decisiones para solucionar los problemas encontrados.

Ver Anexo 6: Mapa conceptual CORE

3.2.2.7. Aspect-Oriented Requirements Engineering for Component-Based Software Systems (AOREC)

Esta aproximación se enfoca en identificar y especificar los requisitos relacionados a los principales aspectos sistémicos, un ejemplo de dichos aspectos pueden ser por ejemplo interfaces de usuario, persistencia y trabajo colaborativo. Para la refinación de los aspectos sistémicos, esta aproximación utiliza sets de “Aspect Details”.

Adicionalmente podemos observar en esta aproximación que no se cuenta con un método general para soportar la identificación de concerns. Los aspectos y los “aspect details” son identificados uno a uno con base en la ingeniería de requisitos.

Ver Anexo 7: Mapa conceptual AOREC

3.2.2.8. Theme – Doc

3.2.2.8.1 *Método*

Es una metodología diseñada para identificar los aspectos de un sistema en su etapa más temprana, es decir en el momento en que se crea el documento de Requisitos. Esta aproximación permite entender un requisito como la descripción de los comportamientos que están entrelazados o entretejidos, por

medio de largos procesos de elicitación de requisitos que darán como resultado un análisis léxico.

Para ejecutar el proceso de identificación de aspectos existe una herramienta soportada por la aproximación la cual lleva su mismo nombre (Theme Doc Tool). Esta herramienta permite analizar eficazmente los requisitos del sistema por medio de un mapeo grafico, el cual es llevado a Theme/UML para ser representado en el momento en que se realiza el diseño de la aplicación.

3.2.2.8.2 *Artefactos*

El artefacto principal generado por esta aproximación es la grafica *action view*, la cual permite ver las acciones que van a ser realizadas por el sistema y las conexiones existentes dentro de ellas.

Cabe anotar que la mayoría de estos artefactos son obtenidos por medio de la herramienta, pero es importante tener los elementos de entrada los cuales son diseñados manualmente y se definen de acuerdo a sus características en *action words* y *entities*.

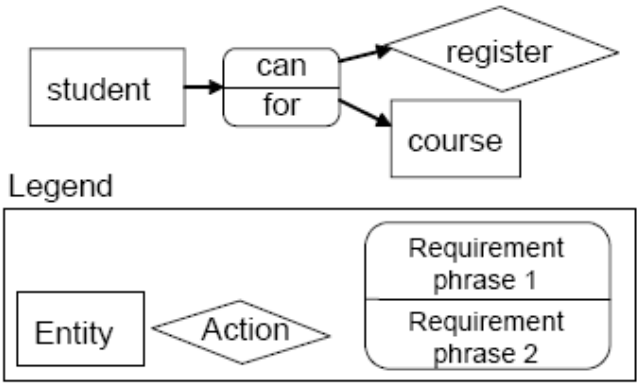


Figura 3Theme View from Theme/Doc Tool for a course of Registration System
Fuente: [2]

3.2.2.8.3 *Proceso Theme/Doc*

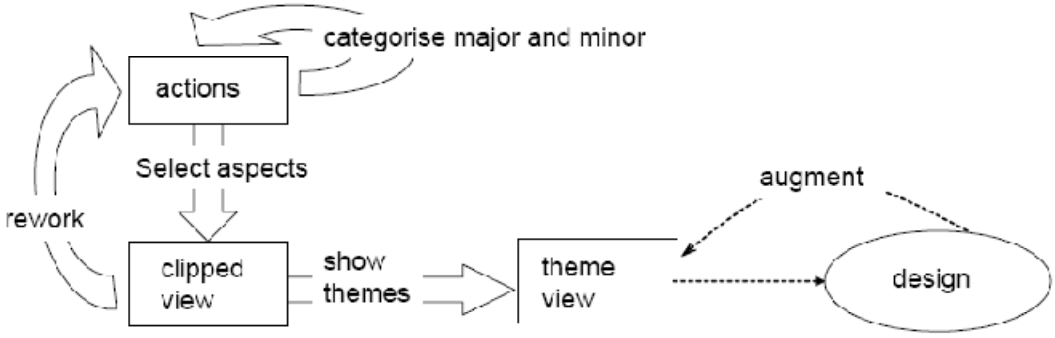


Figura 4.Proceso Theme/Doc
Fuente: [2]

La fase inicial se da con la identificación de las "*action words*" dentro del documento de requisitos, estos y el documento como tal serán los datos de entrada para la herramienta Theme/Doc. Una vez la herramienta procesa los datos ingresados genera los "*actions views*" por medio de los cuales se demuestra la relación entre los requisitos y las actions words.
Por medio de este proceso se pueden identificar los requisitos que se han especificado de forma incorrecta o se puede también localizar la presencia de un comportamiento "crosscutting".

La "*Theme view*" es creada por medio de la identificación de entidades dentro del documento de requisitos que serán usadas en la etapa de diseño, adicionalmente a dichas entidades se ingresa también las action words y el documento de requisitos. La idea central con la Theme view es que se pueda utilizar en la etapa de diseño para producir Themes que serán luego mapeados en Theme/UML.

Ver Anexo 8: Mapa conceptual Theme/Doc

3.2.3. AO-ARCHITECTURE

A continuación se expondrán algunos de los modelos utilizados para generar el trazado de asuntos transversales en la parte de arquitectura.

3.2.3.1The Perspectival Concern-Space (PCS) Framework

Es una técnica utilizada para representar concerns de múltiples dimensiones en la parte de arquitectura que consta de uno o varios diagramas de modelos.

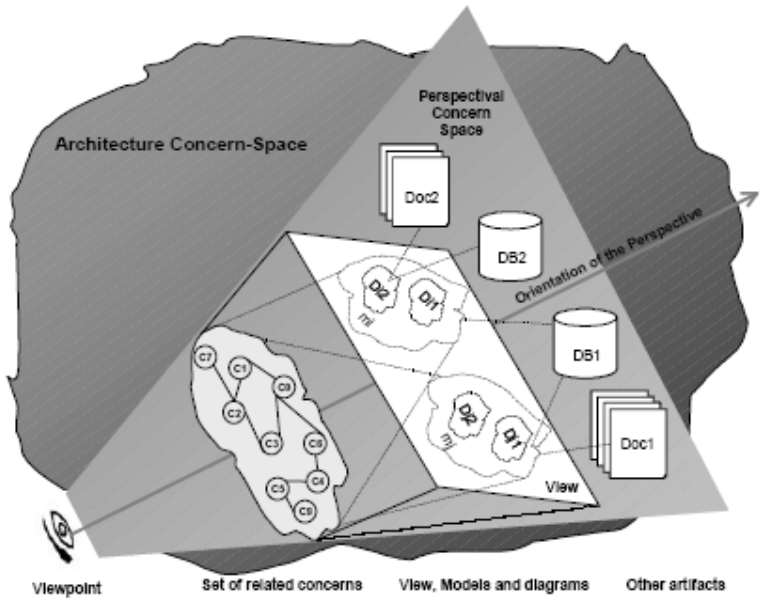


Figura 5 Visualización de la perspectiva del concern en los espacios definidos
Fuente: [2]

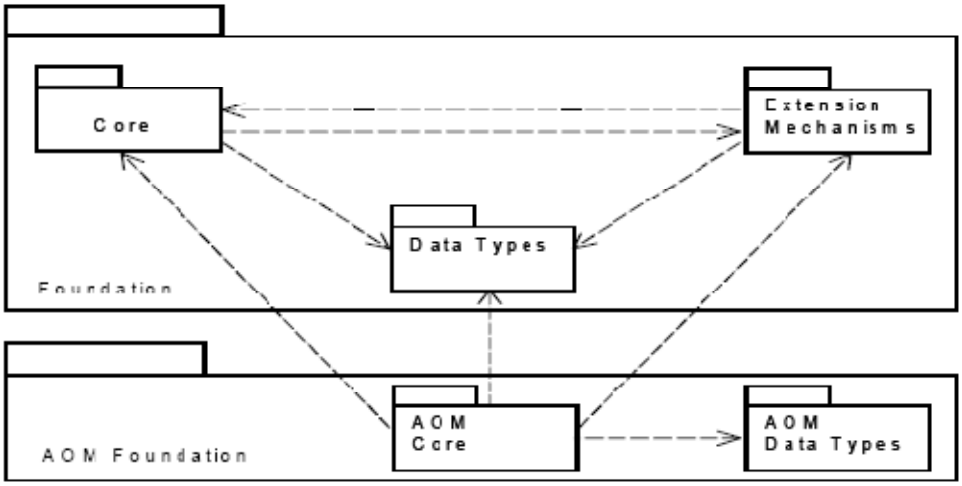


Figura 6. Vista de paquete de alto nivel del espacio UML para AOM

Fuente: [2]

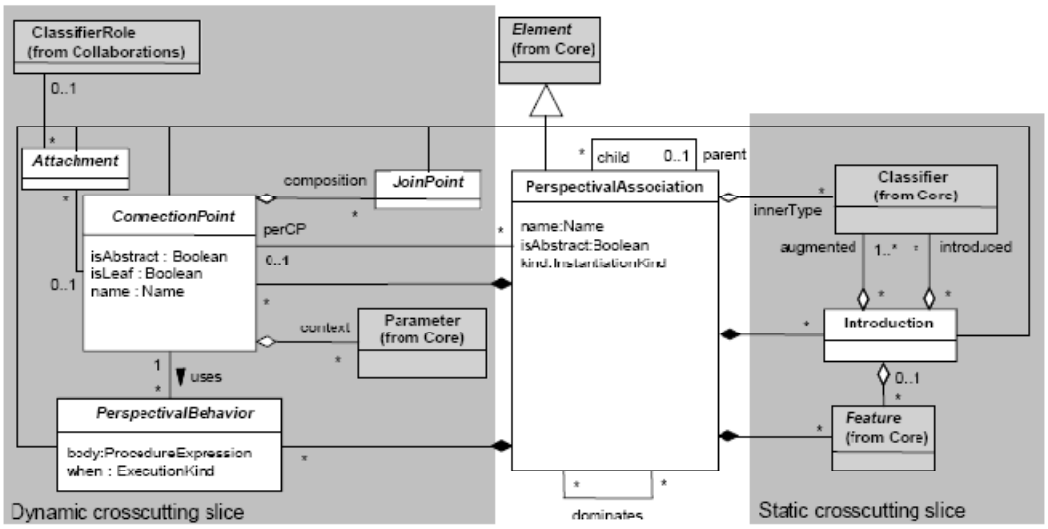


Figura 7. Espacio UML para AOM – Una vista a bajo nivel del core de AOM
Fuente: [2]

3.2.3.2 DAOP-ADL

Esta aproximación es basada en (ADL) un lenguaje de descripción de arquitectura basado en XML el cual describe una aplicación en término de sus componentes, sus aspectos y la conexión entre estos dos, dichas conexiones se presentan en dos tipos diferentes de composiciones:

- Component Composition Rules: Los cuales describen las reglas de negocio que guían la composición de componentes.
- Aspect Evolution Rules: Los cuales son equivalentes a los pointcuts en lenguajes de programación orientados a aspectos y presentan las reglas entre componentes y aspectos.

DAOP utiliza una herramienta llamada Component and Aspect Repository la cual es usada para registrar componentes COTS y aspectos. Dicha herramienta genera automáticamente la descripción de los componentes y aspectos usando la sintaxis de la aproximación.

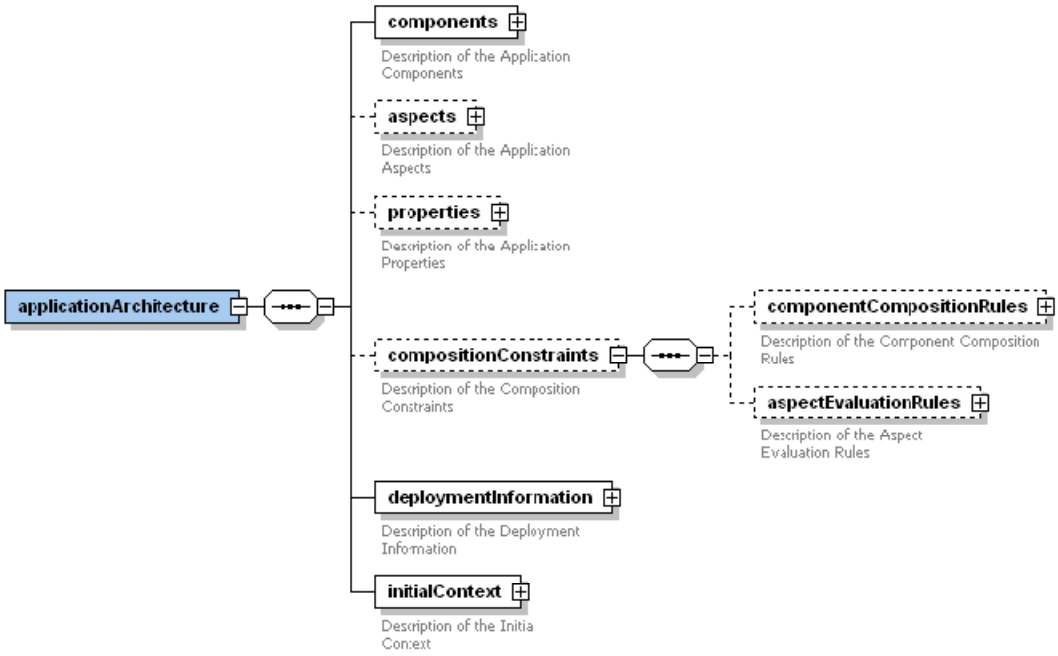


Figura 8. Estructura del lenguaje DAOP-ADL
Fuente: [2]

3.2.3.3 Aspect-Oriented Generative Approach (AOGA)

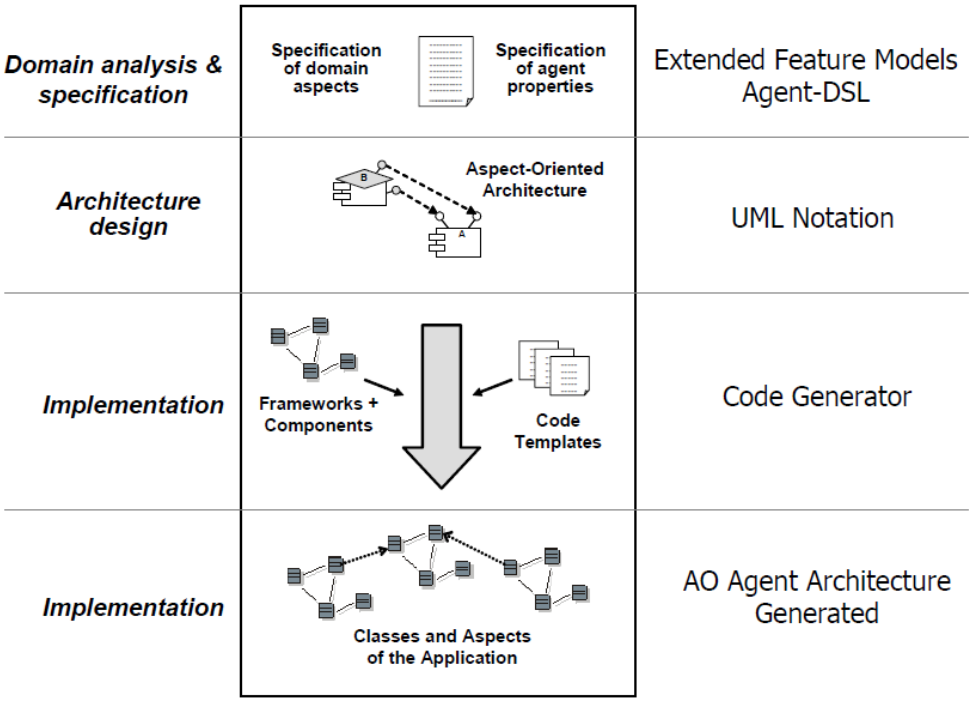


Figura 9. The development phases covered by the AOGA approach
Fuente: [2]

Es una aproximación centrada en la arquitectura y cuyo propósito es soportar el desarrollo de sistemas multiagente con un específico dominio de lenguajes, notaciones de modelado y herramientas de generación de código. AOGA cubre las siguientes fases del ciclo de vida de desarrollo de software: Análisis y especificación, diseño de arquitectura e implementación.

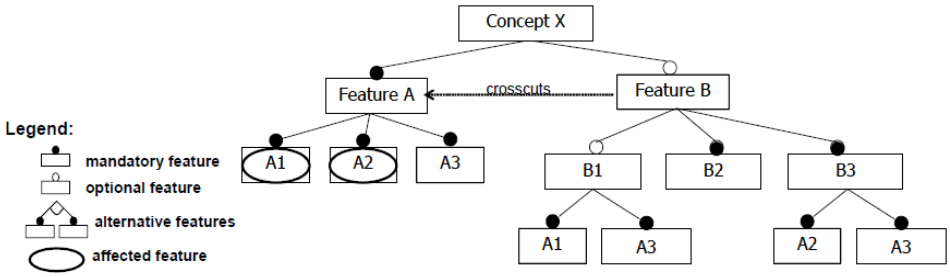


Figura 10. Crosscutting feature models
Fuente: [2]

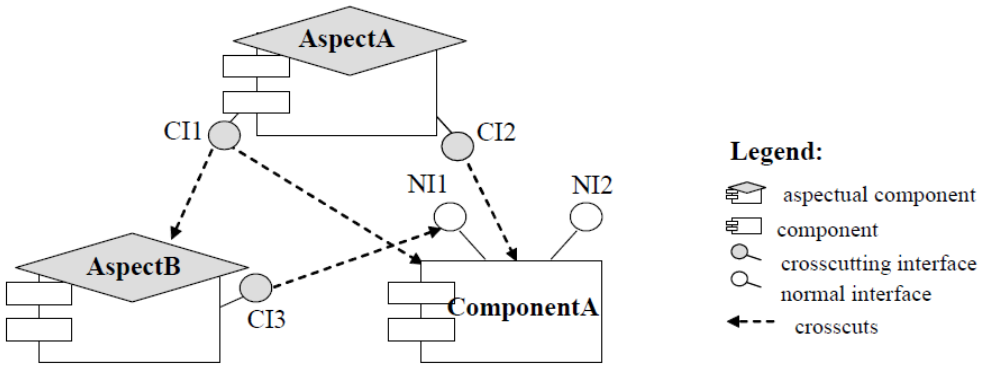


Figura 11. Architectural aspects and crosscutting interfaces
Fuente: [2]

3.2.3.4 TransSat

Es un framework para la especificación de la evolución del software, se enfoca en facilitar la evolución de la arquitectura a través de los principios de AOSD en el contexto de arquitectura. La especificación de la arquitectura es transformada por técnicas de integración de concerns dentro de la arquitectura.

Es importante resaltar que para esta aproximación es bastante importante mantener los concerns lo suficientemente genéricos de forma que estos puedan ser reusables en muchos contextos.

3.2.3.5 Aspectual Software Architecture Analysis Method (ASAAM)

Esta aproximación pretende identificar y especificar aspectos de arquitectura tempranos en el ciclo de vida del software [2]. Esta aproximación puede ser considerada un complemento a los métodos de análisis de arquitecturas y su mayor beneficio es el soporte sistemático para la administración de aspectos relacionados con la arquitectura.

Los artefactos que son base para el desarrollo de esta aproximación son dos:
1. Escenarios
2. Componentes de arquitectura.

Los escenarios son divididos en tres: directos, indirectos y aspectuales, el primero se desarrolla de forma directa como su nombre bien lo indica, el segundo necesita de cambios en los componentes y el tercero es una combinación de los dos primeros o uno de ellos pero éstos serán distribuidos a través de múltiples componentes. Así como los escenarios están divididos, los componentes también se dividen en cuatro:

- Componentes Cohesivos: son aquellos que están bien definidos y representan escenarios semánticamente cerrados.
- Ill-defined component: componente que consta de muchos subcomponentes el cual representa un set de escenarios semánticamente cerrados.
- Tangled component: un componente que representa un escenario aspectual el cual es también representado directa o indirectamente por el componente.
- Composite Component: componente que incluye escenarios semánticamente distintos pero que no pueden ser descompuestos semánticamente ó no incluyen un escenario aspectual.

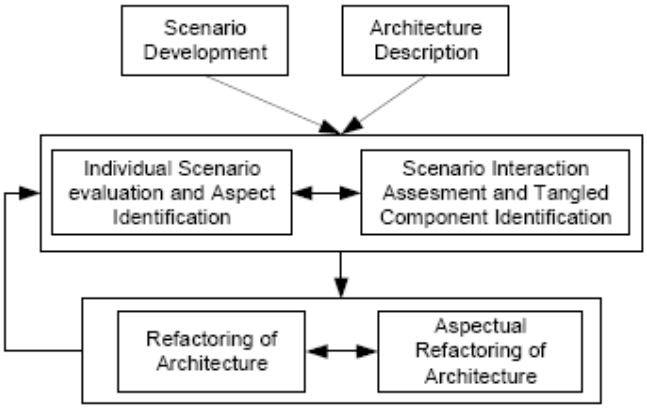


Figura 12. Actividades de ASAAM
Fuente: [2]

R0:
Develop SCENARIO artifacts based on PROBLEM DESCRIPTION

R1:
IF SCENARIO does not require any changes to architectural description
THEN SCENARIO becomes DIRECT SCENARIO

R2:
IF SCENARIO requires changes to one or more ARCHITECTURAL COMPONENTs
THEN SCENARIO becomes INDIRECT SCENARIO

R3:
IF INDIRECT SCENARIO can be resolved after refactoring
THEN INDIRECT SCENARIO is DIRECT SCENARIO

R4:
IF DIRECT SCENARIO is scattered and cannot be localized in one component
THEN DIRECT SCENARIO is ASPECTUAL SCENARIO

R5:
IF INDIRECT SCENARIO is scattered and cannot be localized in one component
THEN INDIRECT SCENARIO is ASPECTUAL SCENARIO

R6:
Derive ARCHITECTURAL ASPECT from ASPECTUAL SCENARIO

Figura 13. Reglas heurísticas para la evaluación de escenarios
Fuente: [2]

Modelo de Trazabilidad (Aproximación)	Características del Modelo que soportan trazabilidad en el ciclo de vida del artefacto.

PCS Framework	Puede ser mapeado por medio de AspectJ, lo cual es un proceso manual.
DAOP-ADL	La trazabilidad se realiza por medio de los artefactos identificados en el diseño de la aplicación por medio de CAM y son descritos en un XML usando el lenguaje DAOP-ADL
AOGA	Existe una traza directa entre un crosscutting en el análisis y el componente aspectual en el diseño
TransSat	Realiza un proceso iterativo por medio del cual soporta a trazabilidad a través del ciclo de vida de desarrollo
ASAAM	La especificación de requisitos tomada durante la evaluación de la arquitectura.

Tabla 5. Características del Modelo que soportan trazabilidad en el ciclo de vida del artefacto
Fuente: [2]

Ver Anexo 9: Mapa conceptual AO-Architecture

3.2.4. AO DESING

A continuación se presentan algunos de los modelos utilizados en la etapa de diseño del ciclo de vida de software.

3.2.4.1. Aspect-Oriented Design Modeling (AODM)

3.2.4.1.1 *Lenguaje AOD*

Desde AODM fue originalmente diseñado para modelar el estilo AspectJ AOP, sus elementos de diseño siguen siendo en gran medida relacionados con AspectJ. Nos basamos en la terminología de AspectJ cuando discutimos sobre AODM incluso cuando se habla de diseño de modelos que se dirigen a diferentes objetivos.

3.2.4.1.2 *Especificación de los aspectos*

En AODM, los aspectos se representan como clases con el estereotipo <<aspect>>. Esto se hizo así puesto las similitudes estructurales entre aspectos y clases.
Como las clases, los aspectos actúan como containers y namespaces para los atributos, operaciones, pointcuts, asesoramiento y declaraciones intertype. Los aspectos también pueden tener relaciones de asociación y de generalización como las clases.

AspectJ difiere de las clases en sus mecanismos de instanciación y de herencia. En AspectJ las declaraciones pueden contener clausulas de instanciación que especifiquen la manera en que los aspectos deben ser instanciados. Los aspectos hijos pueden heredar todas las características de su aspecto padre pero únicamente los pointcuts abstractos y las operaciones de java pueden ser sobrescritos, es decir no heredar las características de su padre e implementar las propias.

3.2.4.1.3 *Especificación de crosscutting*

AODM soporta la especificación de crosscutting estructurales y de comportamiento.la estructura crosscutting es expresada en diagramas de clase con una plantilla parametrizada de un diagrama de colaboración.

3.2.4.1.4 *Especificación de integración*

AODM soporta la especificación de integración tanto estructural como de comportamiento a través de la integración de modelos de diseño de crosscutting. Los crosscutting estructurales afectan el tipo de estructura dada al modelo de diseño y que puede ocurrir en alguna instancia de la jerarquía de clase especificada. Ciertos comportamientos de los crosscutting afectan la

especificación de otros o del mismo comportamiento. Esto ocurre en ciertos puntos en lo que se ejecuta la especificación.

3.2.4.1.5 *Semántica de la composición*

Una semántica detallada de la composición no es provista por AODM en su literatura. Debido a los estrechos vínculos con AspectJ permite inferir que la composición semántica seguida por AODM es muy similar a AspectJ.

AODM soporta la representación de semánticas de composición a través de dos diagramas. El primer diagrama, identifica los joinpoints que se están cruzando transversalmente por un aspecto y el elemento de crosscutting en ese aspecto que afectara realmente ese joinpoint. El segundo diagrama representa un joinpoint actual y especifica la composición de dicho join point.

Los puntos que se están cruzando transversalmente son capturados dentro de los elementos del diagrama de clases para un crosscutting estructural.

Las declaraciones de intertype entrelazados son representadas en diagramas de casos de uso. El aspecto que contiene declaraciones intertype es representado como un caso de uso. Las declaraciones inter-type son refinadas primero desde el caso de uso aspectual hacia casos de uso que representen las característica actuales que serán introducidas en las clases base. Estos casos de uso son entonces incluidos en un caso de uso que representa la entidad entrelazada. La composición es representada como casos de uso. Estos están relacionados con la base y los elementos de crosscutting que están compuestos a través de la asociación <<include>>.

Ver Anexo 10: Mapa conceptual AODM

3.2.4.2 Theme/UML

3.2.4.2.1 *Método Theme/UML*

Theme es una aproximación de análisis y diseño que soporta la separación de concerns para las fases de análisis y diseño del ciclo de vida del software. La aproximación Theme también provee un UML basado en el lenguaje AOD llamado Theme/UML el cual extiende del meta modelo UML. La aproximación Theme expresa los concerns en construcciones conceptuales y de diseño llamadas Themes. Los Themes son más generales que los aspectos, y abarcan más de cerca los concerns con relación a la separación simétrica. Cualquier concern, sin importar si es o no crosscutting, puede ser encapsulado en un theme.

3.2.4.2.2 *Artefactos Theme*

3.2.4.2.3 *Lenguaje AOD*

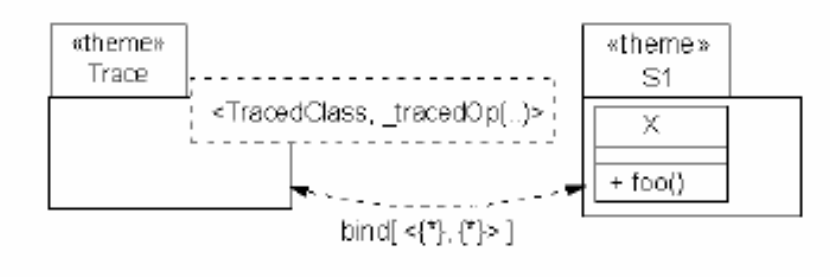


Figura 14. Themes e integración de Themes
Fuente: [2]

3.2.4.2.4 *Especificación de aspectos*

Siendo un lenguaje simétrico, Theme/UML soporta la representación de themes base y aspectuales. Los themes base y aspectuales son especificados en paquetes. Los paquetes theme están identificados como themes a través del estereotipo <<theme>> como se muestra en la figura 5-14. Los themes aspectuales y base contienen diagramas de clase que representan la estructura del aspecto. El diagrama de clase dentro del paquete del theme

representa los conceptos de diseño que se requieren para saber que requerimientos están relacionados con el theme.

Un theme aspectual difiere del theme base en que se trata en un paquete plantilla parametrizado. La figura 5.15 es un ejemplo de un theme aspectual. El cuadrado punteado en la parte superior derecha especifica los parámetros de la plantilla. Los parámetros representan los join points donde el theme hace crosscut. El parámetro, en este ejemplo, es un Type y el método y el parámetro se representan así <TracedClass tracedOp (...)>. Este ejemplo muestra el diseño de la funcionalidad de rastreo del crosscutting.

La relación estructural entre el parámetro y los elementos de crosscutting encapsulados dentro del theme son descritos en el diagrama de clases. Las relaciones de comportamiento entre el parámetro y los elementos de crosscutting están encapsulados dentro del theme descrito en el diagrama de secuencia.

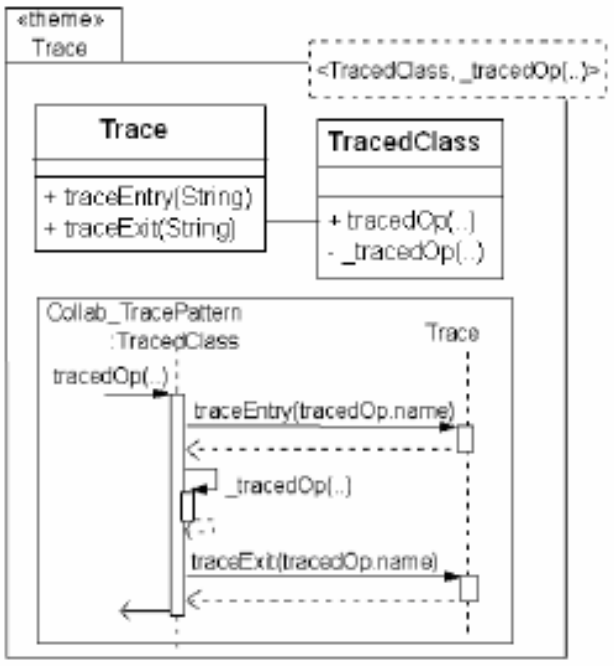


Figura 15 Aspect Theme
Fuente: [2]

3.2.4.2.5 *Especificación del crosscutting*

Ambos, themes aspectuales y base soportan un grado de crosscutting estructural debido a la posibilidad de superposición de conceptos de dominio entre los themes. Los themes aspectuales soportan crosscuting de comportamiento como crosscutting estructural.

En el diseño, los themes son visualizaciones parciales de los requerimientos. Los requerimientos expresan todo sobre los conceptos del dominio que deben ser modelados para crear un sistema. Una superposición entre los themes ocurre cuando los requerimientos para diferentes themes describen parcialmente un concepto de dominio. En el diseño, esta superposición se ve en las clases dentro de diferentes themes que representan el mismo concepto de dominio. Estas clases tienen miembros, métodos y atributos, como también relaciones con otras clases en un theme. Los métodos y atributos se pueden superponer como relaciones. La superposición de miembros de clase y relaciones entre clases es menos común. Durante la composición, estas clases superpuestas convergen a un modelo completo. Debido a la posible superposición entre themes, todos los themes son propensos a crosscutting estructural implícito. Únicamente las clases de aspectos soportan crosscutting de comportamiento, este se especifica en un diagrama de secuencia.

3.2.4.2.6 *Especificación de la integración*

Theme soporta la especificación de dos tipos de integración - override y merge. Debido al modelo simétrico que theme soporta, las relaciones de integración pueden ser especificadas como diferentes tipos de themes. La especificación de integración se denota con un arco y puede ser especificada en dos niveles.
Primero, la integración puede ser especificada en el nivel del theme. Esto indica que los elementos en los puntos de llegada del arco deben estar compuestos acorde a las reglas de integración especificadas por dicho arco.
Segundo, la integración puede ser especificada entre los elementos dentro de los encapsulamientos del theme. La integración por override y merge puede ser

especificada en ambos niveles. La integración override es una especificación de crosscutting estructural. Es usado para especificar que una superposición entre los elementos theme se resolverá mediante la sustitución de los elementos en los themes que se están superponiendo. Así, una integración de superposición puede ser considerada para que sea un crosscutting estructural. La integración de superposición se denota por un arco unidireccional.

La integración por merge es una especificación de crosscutting estructural. La integración merge es usada para especificar que hay una superposición entre los elementos theme y que será resuelta añadiendo las partes de los elementos que se superponen, uniéndolos. El resultado de un merge es la suma de las partes de aquellos elementos que se especifican para ser pasados por un merge. Este resultado es una estructura alterada dentro de las estructuras merge y son composiciones de estructuras. Una especificación de merge puede fusionar elementos en muchos themes. Por tanto, este puede ser considerado para ser una especificación para un crosscutting estructural. La integración merge se denota por un arco de doble dirección.

Ver Anexo 11: Mapa conceptual Theme/UML

3.2.4.3 CoCompose

CoCompose es un lenguaje de diseño no UML que soporta la representación de aspectos reusables de alto nivel como características. CoCompose introduce un elemento de construcción en el lenguaje AOD de CoCompose.
Una característica es un aspecto de alto nivel que atraviesa los límites de la aplicación.

3.2.4.3.1 *Lenguaje AOD*

CoCompose es un lenguaje de diseño grafico que puede ser usado para soportar AOD. CoCompose soporta diseños ejecutables mediante el diseño de elementos del lenguaje que sean bien definidos semánticamente.
Diseñar una algebra es una técnica para determinar y seleccionar construcciones concretas del lenguaje para implementar conceptos. Esta es la base para el lenguaje de diseño CoCompose para programar la traducción del lenguaje.

3.2.4.3.2 *Especificación de los aspectos*

Los aspectos reutilizables son especificados como características en CoCompose. Las características son construcciones abstractas que describen un patrón de diseño. Las características están bien definidas semánticamente. Una característica es una composición de conceptos y roles. Un rol es un template concept. Los conceptos son la construcción básica del lenguaje en CoCompose y son usados para modelar los conceptos del dominio.

Los conceptos pueden tener estrategias de implementaciones asociadas a ellos. Donde los conceptos no tienen estrategias asociadas implementadas, una adecuada aplicación de estrategia puede ser facilitada por la característica en la que se expresa el concepto.

3.2.4.3.3 *Especificación de los ejes transversales*

Los roles son plantillas que deben cumplirse para realizar funciones como entidades concretas. Los roles son especificaciones de ejes transversales. Se han asociado las limitaciones que restringen y validan los conceptos que se utilizan para coincidir con el rol.

3.2.4.3.4 *Especificación de la integración*

Solución de los patrones que construye CoCompose y especifica la integración de funciones de los roles y conceptos.

3.2.4.3.5 *Especificación de la composición*

Una vez que un patrón de solución integra los conceptos, un modelo completo de conceptos es el resultado. Ya que estrategias de implementación están relacionadas con los conceptos y características, una composición puede ser tratado como un sistema ejecutable. Una vez que la composición se completa el diseño del modelo puede ser traducido en una implementación.

3.2.4.3.6 *Proceso CoCompose*

CoCompose tiene por objeto automatizar el proceso de diseño hasta su aplicación. La automatización de generación de código se consigue mediante la definición de las estrategias de aplicación de elementos de diseño semánticamente bien definidos. Como tal, este enfoque se centra en automatizar la transformación de alto nivel de diseño de modelos a implementaciones especificas de plataforma.

Ver Anexo 12: Mapa conceptual CoCompose.

3.2.4.4 UML For Aspects (UFA)

En este modelo los paquetes UML son usados para encapsular partes de un sistema que contribuye a un comportamiento complejo, de forma que se puedan definir propiedades (Atributos y Métodos) de un paquete.

Los paquetes de aspectos contienen diagramas de clases que representan el diseño detallado de un comportamiento complejo que se ha diseñado. Las clases abstractas y los métodos definidos en los paquetes aspectuales son roles que necesitan ser completados para entender el aspecto dentro del contexto de la aplicación.

UFA por medio de la especialización permite que un aspecto se especialice para cortar transversalmente una aplicación específica, creando relaciones que definirán conectores. Un conector es un diseño de aspecto de nivel medio que es específico dentro de la aplicación.

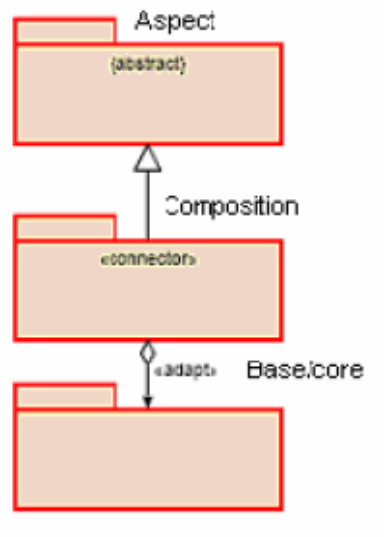


Figura 16. Nivel de Composición de un Paquete
Fuente: [2]

Ver Anexo 13: Mapa conceptual UFA

3.2.4.5 Aspect Modeling Language (AML)

AML así como UFA considera que los aspectos son constructores de más alto nivel que las clases, los aspectos entonces son especificados como paquetes de un <<aspecto>> estereotipado.

Los paquetes de aspectos encapsulan los crosscutting, su comportamiento y su estructura. Existe también para este modelo un conector, la asociación entre este conector y un paquete de aspectos es presentada por medio de una relación de dependencia estereotipada llamada <<*uses*>>. Así pues el aspecto usa el conector para especificar la estructura y el comportamiento del crosscutting que están encapsulados dentro de sí.

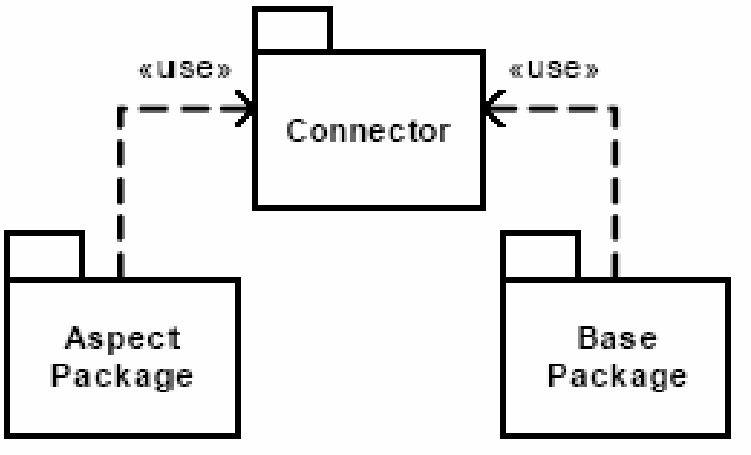


Figura 17. AML aspect, connector and base
Fuente: [8]

Ver Anexo 14: Mapa conceptual AML

3.2.4.6 Aspect Oriented Component Engineering (AOCE)

En este modelo los aspectos son definidos como componentes, dichos componentes están relacionados a través de los servicios que ellos proveen o requieren. Cabe anotar que la definición de un componente como aspecto es diferente de la definición de un componente en sí.

Los componentes están definidos en un modelo de componentes de AOCE el cual es extendido a un modelo aspectual. El modelo de diseño componente/aspecto es la representación de la implementación de un framework, dicho framework es extendido para crear el diseño específico de la aplicación.

Los aspectos pueden ser definidos también en diagramas de componentes, adicionalmente los aspectos componentes y sus interacciones pueden ser definidos en diagramas de colaboración.

Ver Anexo 15: Mapa conceptual AOCE

4 CASO DE ESTUDIO

En este capítulo presentamos un caso de estudio que nos permitirá analizar la trazabilidad en algunas aproximaciones en los requisitos, la arquitectura y el diseño.

4.1 Descripción Caso de Estudio Health Watcher

El propósito de este sistema es recoger y manejar las quejas y notificaciones que hacen las personas con relación a la salud. El sistema es también usado para notificar a las personas sobre información importante relacionada con el Sistema de Salud Pública.

El sistema "Health Watcher" debe también intercambiar información con el sistema SSVV (Sanitary Surveillance System). Inicialmente este intercambio involucrará la búsqueda de licencias de sanidad. Subsecuentemente, cuando el SSVS libere el modulo "Control de Quejas", las quejas relacionadas con el Sistema de Vigilancia Sanitaria serán intercambiadas entre los dos sistemas para mantener actualizada la información. [10]

4.2 Aspectos Tempranos y View Points obtenidos del caso de estudio

4.2.1 Early Aspects

4.2.1.1 Early aspect: Disponibilidad

Requerimientos	**1.** El sistema debe estar disponible 24 horas al día 7 días a la semana.
	2. Como en la naturaleza del sistema no se considera un sistema muy crítico, el sistema puede quedarse fuera de servicio hasta que la falla sea arreglada.

4.2.1.2 Early aspect: Seguridad

Requerimientos	1. Protocolo de seguridad (encriptación): el sistema debe usar un protocolo de seguridad cuando este enviando datos a través del internet.
	2. Control de acceso: para tener acceso a las características del registro de la queja, el acceso debe ser permitido a través del subsistema de control de acceso. Los empleados tienen un nombre de usuario y contraseña para usar el sistema (por ejemplo, la actualización de las quejas y tablas).

4.2.1.3 Early aspect: Desempeño

Requerimientos	1. El sistema debe ser capaz de manejar 20 usuarios de manera simultanea
	2. El tiempo de respuesta no debe exceder los 5 segundos

4.2.1.4 Early aspect: Concurrencia

Requerimientos	1. El sistema debe ser capaz de manejar 20 usuarios de manera simultanea
	2. El sistema deberá ser capaz de tratar con las respectivas peticiones de los usuarios concurrentes tales como base de datos de transacciones para evitar la corrupción de datos. Además, los usuarios concurrentes deberían disponer de un adecuado tiempo de respuesta.

4.2.1.5 Early aspect: Persistencia

Requerimientos	1. El mecanismo de persistencia debería almacenar datos sobre las quejas, los empleados, unidades de salud, enfermedades, especialidades y de los ciudadanos que se quejan.
	2. El sistema debe ser flexible en términos de formato de almacenamiento permitiendo el uso de arreglos o diferentes conjuntos de bases de datos (MySQL, Oracle, etc)

4.2.1.6 Early aspect: Distribución

Requerimientos	1. El sistema debería ser capaz de ejecutarse en diferentes máquinas. Por ejemplo, el sistema básico podría estar ejecutándose en una máquina y el otro sobre Servlets.

4.2.1.7 Early aspect: Manejo de errores y excepciones

Requerimientos	1. Varias funcionalidades pueden desencadenar errores mientras el usuario interactúa con el sistema y requieren diferentes técnicas de manipulación. 1.1. Los errores generales que se aplican a la mayoría de los casos se deben a la falta de información (por ejemplo, los usuarios no rellenan los campos necesarios en un formulario de inscripción) y el sistema señala el error y muestra los campos que deben proporcionarse. 1.2. Otros errores pueden ser relacionados con la entrada de datos no válidos y el mecanismo de gestión de errores debería tratar de evitar que suceda lanzar el error y sugerir la corrección.

4.2.1.8 Early aspect: Compatibilidad

Requerimientos	1. Como el sistema de Health Watcher intercambia información con el sistema SSVS (Sistema de Vigilancia Sanitaria), ellos deben cumplir con el mismo mecanismo de intercambio de datos. 1.1. Ambos sistemas deben ponerse de acuerdo sobre un protocolo para el intercambio de los datos para evitar incompatibilidades en la integración. 1.2. El intercambio de datos deben cumplir con un formato pre-definido y acordado (por ejemplo, XML). 1.3. Cuestiones de seguridad pudieran estar implicados para garantizar un intercambio seguro de información entre ambos sistemas.

4.2.1.9 Early aspect: Usabilidad

Requerimientos	1. El sistema debe tener una interfaz gráfica fácil de usar, ya que cualquier persona que tenga acceso a la Internet debería poder utilizar el sistema.
	2. El sistema debería tener una ayuda en línea para ser consultada por cualquier persona que la utilice. 2.1. Varias funcionalidades del sistema deben proporcionar ayuda sensible al contexto para el usuario (por ejemplo, explicar cuáles son los tipos de quejas)

4.2.1.10 Early aspect: Cuestiones jurídicas

Requerimientos	1. El sistema debe desarrollarse de acuerdo con las normas establecidas por la Empresa X, responsable de las normas y la estandarización de los sistemas del City Hall.

4.2.1.11 Early aspect: Ambiente de operación

Requerimientos	1. La interfaz de usuario debe ser implementada utilizando Servlets.
	2. Software: Una licencia para Microsoft Windows para la estación de trabajo.
	3. Hardware: un computador con: procesador Pentium 3, 256MB de memoria RAM, tarjeta de red 3com 10/100. Este equipo debe ser utilizado por los asistentes como estación de trabajo.

4.2.2 Viewpoints

4.2.2.1 Viewpoint: Empleado

Requerimientos	**1.** Existen tres tipos de empleados: **1.1.** Inspectores **1.2.** Asistentes **1.3.** Administradores
	2. El empleado se debe loguear para que así él/ella pueda acceder a varias operaciones del sistema, como son: **2.1.** Ingreso al sistema: esta operación permite a un empleado tener acceso a operaciones restringidas en el sistema Health Watcher. **2.2.** Registro de tablas: Esta operación permite el registro de las tablas del sistema. Las siguientes operaciones son posibles: insertar, actualizar, eliminar, buscar e imprimir. las tablas disponibles incluyen: **2.2.1.** **Unidad de salud** (código de unidad, descripción de unidad) **2.2.2.** **Especialidad** (código y descripción) **2.2.3.** **Unidad de salud** / **Especialidad** (unidad de salud y especialidades). **2.2.4.** **Empleado** (login, nombre y contraseña).

	2.2.5. Tipo de enfermedad (código, nombre, descripción, síntomas y la duración). **2.2.6. Síntoma** (código y descripción). **2.2.7. Tipo de enfermedades y síntomas** (tipo de enfermedad y los síntomas). **2.3. Actualización de la queja**: permite actualizar el estado de una queja **2.4. Registro de nuevos empleados:** permite registrar a los nuevos empleados que vayan a registrarse en el sistema. La información que debe ingresarse es: Nombre, Nombre de usuario, contraseña. **2.5. Actualización de los empleados:** cambios en el estado del empleado **2.6. Actualización de la unidad de salud:** actualización de los datos en la unidad de salud **2.7. Cambio empleado registrado:** cambiar los datos del empleado

4.2.2.2 Viewpoint: Ciudadano

Requerimientos	**1.** Un ciudadano es cualquier persona que desea interactuar con el sistema
	2. Un ciudadano puede interactuar con el sistema para solicitar información y para registrar una queja específica.
	3. Un ciudadano puede acceder al sistema a través de internet o marcando el 1520, y realizar su queja o pedir información sobre los servicios de salud.
	4. El ciudadano puede consultar: **4.1.** Unidades de salud que se hacen cargo de una especialidad. **4.2.** ¿Cuáles son las especialidades de una unidad de salud? **4.3.** Información acerca de una denuncia presentada por un ciudadano: **4.3.1.** Detalles de la queja **4.3.2.** Situación (abierto, suspendido o cerrado). **4.3.3.** Análisis técnico. **4.3.4.** Fecha del análisis **4.3.5.** Empleado que hizo el análisis. **4.4.** Información acerca de las enfermedades: **4.4.1.** Descripción. **4.4.2.** Síntomas. **4.4.3.** Duración.

4.2.2.3 Viewpoint: Sistema de vigilancia sanitaria (SSVS)

Requerimientos	**1.** El sistema Health Watcher también debe intercambiar información con el sistema SSVS (Sistema de Vigilancia Sanitaria). **1.1.** Inicialmente, este intercambio supondrá la consulta de licencias sanitarias. **1.2.** Posteriormente, cuando el SSVS tenga el modulo de control de quejas desplegado, las quejas de la vigilancia sanitaria serán intercambiados entre los dos sistemas.

4.2.2.4 Viewpoint: Queja

Requerimientos	**1.** Los tipos de quejas pueden ser: **1.1. Queja animal – DVA** **1.1.1.** Animales enfermos. **1.1.2.** Infestaciones (roedores, escorpiones, murciélagos, etc.) **1.1.3.** Enfermedades relacionadas con los mosquitos (dengue, filariose). **1.1.4.** Maltrato animal. **1.2. Demanda de alimentos - DVISA** 1.2.1. Casos en los que se sospecha que los alimentos están infectados al ser ingeridos. **1.3. Queja Especial – DVISA** **1.3.1.** Los casos relacionados con diversas razones, que no se mencionan arriba (restaurantes con problemas de higiene, fugas de alcantarillado, agua sospechosas el transporte de camiones, etc.)
	2. Además de los datos anteriores, cada una de las quejas tiene su propio tipo de datos específicos, entre ellos: **2.1. Queja animal – DVA** **2.1.1.** Tipo de animal (obligatoria), la cantidad de animales (obligatorio), la fecha cuando se observó el problema (obligatorio). **2.1.2.** Problema de los datos de localización: calle, complemento, distrito, ciudad, estado o provincia,

	código postal y número de teléfono. Todos estos campos son opcionales **2.2. Demanda de alimentos – DVISA** **2.2.1.** Nombre de la víctima (obligatorio). **2.2.2.** Datos de la víctima: calle, complemento, distrito, ciudad (o más cercana), estado / provincia, código postal y número de teléfono. Todos estos campos son opcional. **2.2.3.** Cantidad de personas que comieron el alimento, la cantidad de personas enfermas, la cantidad de personas que fueron enviados a un hospital y la cantidad de personas fallecidas. Todos obligatorios. **2.2.4.** Lugar en donde los pacientes fueron tratados, comida sospechosa. Todo opcional. **2.3. Queja Especial – DVISA** **2.3.1.** Edad (obligatorio), títulos académicos (opcional), la ocupación (opcional). **2.3.2.** Calle, complemento, distrito, ciudad, estado o provincia, código postal y número de teléfono de la ubicación más cercana a la ubicación de la queja. Todo opcional.
	3. Los ciudadanos pueden registrar una queja, mientras que los empleados pueden registrar y actualizarlas.
	4. En el caso de que una queja se están realizando, esta será registrada en el sistema y direccionada por un departamento específico. **4.1. Este departamento será capaz** de manejar la queja de una manera apropiada y devolver una respuesta cuando la queja ha sido tratada. **4.2. Esta respuesta se registrará** en el sistema y estará disponible para ser consultada.

4.2.3 Modelo de casos de uso

Para el modelo de casos de uso existe ya un documento desarrollado para esto por lo cual lo citaremos y tomaremos de allí la información requerida para el desarrollo de este caso en los diferentes modelos que nos proponemos abordar. [12]

4.3 Representación del Problema en las aproximaciones AOSD

A continuación se presenta el análisis de dos de las metodologías estudiadas para la aplicación de la descomposición de aspectos, estas han sido elegidas con base en el conocimiento adquirido y adicionalmente en la idea fundamental de mostrar cómo se trabajan estas metodologías en dos etapas distintas.

AORE with Arcade fue seleccionada para la parte de Requisitos pues provee información suficiente para intervenir el proceso y generar las trazas que permitirán el entendimiento de la aplicación y sus relaciones hacia adelante. AML (Aspect Modeling Language) fue seleccionada para la parte de Diseño puesto que esta es una de las aproximaciones más usadas dentro del análisis de diseño para la parte aspectual. Además de que permite visualizar claramente el funcionamiento de aspectos dentro del caso de estudio debido a los múltiples artefactos que se pueden construir con este.

4.3.1 Modelo de Requisitos: AORE With Arcade

4.3.1.1 XML Definido para cada Viewpoint

```
<?xml version="1.0" ?>
<Viewpoint name="Empleado">
        <Requirement id="1">Existen 3 clases de empleados: inspectores, asistentes y
administradores</Requirement>
        <Requirement id="2">El empleado puede loguearse en el sistema de modo que el o ella
puedan acceder a las diferentes operaciones del sistema las cuales son:
                <Requirement id="2.1">Login: esta operación permite que un empleado tenga
acceso a operaciones restringidas en el Sistema Health Watcher </Requirement>
                <Requirement id="2.2">Registrar tablas: Esta operación permite el registro de
tablas del sistema. Las operaciones que pueden ser ejecutadas son: Insertar, modificar, eliminar,
buscar e imprimir. Las tablas disponibles incluyen:
                        <Requirement id="2.2.1">Unidad de Salud: código de unidad,
descripción de la unidad</Requirement>
                        <Requirement id="2.2.2">Especialidad: código y
descripción</Requirement>
                        <Requirement id="2.2.3">Unidad de Salud/Especialidad: unidad de
salud y especialidad</Requirement>
                        <Requirement id="2.2.4">Empleado: Login, nombre y
clave</Requirement>
```

<Requirement id="2.2.5">Tipo de Enfermedad: código, nombre, descripción, síntoma y duración</Requirement>
<Requirement id="2.2.6">Síntoma: código y descripción</Requirement>
<Requirement id="2.2.7">Tipo de enfermedad/Síntoma: tipo de enfermedad y síntoma</Requirement>
</Requirement>
<Requirement id="2.3">Actualizar queja: permite actualizar el estado de una queja</Requirement>
<Requirement id="2.4">Registrar nuevo empleado: permitir que los nuevos empleados puedan ser registrados en el sistema.
Información que debe ser guardada: Nombre, LoginID, Clave</Requirement>
<Requirement id="2.5">Actualizar empleado: cambios en el estado del empleado</Requirement>
<Requirement id="2.6">Actualizar Unidad de Salud: Actualizar los datos de la unidad de salud</Requirement>
<Requirement id="2.7">Cambiar los datos del empleado logueado: Cambiar los datos del empleado</Requirement>
</Requirement>
</Viewpoint>

<Viewpoint name="**Ciudadano**">
<Requirement id="1">Un ciudadano es cualquier persona que desee interactuar con el sistema</Requirement>
<Requirement id="2">Un ciudadano puede interactuar con el sistema para buscar información y registrar una queja</Requirement>
<Requirement id="3">Un ciudadano puede acceder al sistema a través de internet o marcando 1520. De esta forma realiza su queja o solicita
información de los servicios de salud</Requirement>
<Requirement id="4">Un ciudadano puede preguntar por:
<Requirement id="4.1">Que unidades de salud se encargan de una especialidad específica</Requirement>
<Requirement id="4.2">Cual es la especialidad de una unidad en particular</Requirement>
<Requirement id="4.3">Información acerca de una queja realizada por un ciudadano</Requirement>
<Requirement id="4.3.1">Detalles de la queja</Requirement>
<Requirement id="4.3.2">Estado: Abierta, Suspendida o Cerrada</Requirement>
<Requirement id="4.3.3">Análisis Técnico</Requirement>
<Requirement id="4.3.4">Fecha del Análisis</Requirement>
<Requirement id="4.3.5">Empleado que realizó el análisis</Requirement>
<Requirement id="4.4">Información acerca de enfermedades
<Requirement id="4.4.1">Descripción</Requirement>
<Requirement id="4.4.2">Síntomas</Requirement>
<Requirement id="4.4.3">Duración</Requirement>
</Requirement>
</Requirement>
</Viewpoint>

<Viewpoint name="**Sanitary Surveillance System (SSVS)**">
<Requirement id="1">El sistema Health Watcher debe también intercambiar información con el sistema SSVS

<Requirement id="1.1">Inicialmente este intercambio involucrará las preguntas sobre las licencias de sanidad</Requirement>
<Requirement id="1.2">Subsecuentemente, cuando el SSVS tenga desarrollado el módulo de "Manejo de quejas" Las quejas relacionadas con
Sanitary Surveillance serán intercambiadas entre los dos sistemas</Requirement>
</Requirement>
</Viewpoint>

<Viewpoint name="**Queja**">
<Requirement id="1">Los tipos de queja pueden ser:
<Requirement id="1.1">Queja por animales - DVA
<Requirement id="1.1.1">Animales enfermos</Requirement>
<Requirement id="1.1.2">Infestación por roedores, escorpiones, murciélagos etc.</Requirement>
<Requirement id="1.1.3">Enfermedades relacionadas con mosquitos</Requirement>
<Requirement id="1.1.4">Maltrato animal</Requirement>
</Requirement>
<Requirement id="1.2">Queja por alimentos - DVISA
<Requirement id="1.2.1">Casos en los cuales existe sospecha de que un alimento infectado fue consumido</Requirement>
</Requirement>
<Requirement id="1.3">Quejas especiales - DVISA
<Requirement id="1.3.1">Casos relacionados a varias situaciones, que no han sido mencionadas anteriormente como restaurantes con
problemas de higiene, falta de cloacas o desagües etc. </Requirement>
</Requirement>
</Requirement>
<Requirement id="2">Adicionalmente a los datos mencionados previamente, cada tipo de queja tiene su propio tipo de dato específico incluyendo:
<Requirement id="2.1">Quejas por Animales - DVA
<Requirement id="2.1.1">Tipo de animal(obligatorio), cantidad de animales(obligatorio), fecha en que el problema fue detectado
</Requirement>
<Requirement id="2.1.2">Datos de ubicación del problema: calle, distrito, ciudad, estado/provincia, Código Zip, Número de teléfono
</Requirement>
</Requirement>
<Requirement id="2.2">Quejas por Alimentos - DVISA
<Requirement id="2.2.1">Nombre de la victima</Requirement>
<Requirement id="2.2.2">Datos de la víctima: calle, distrito, ciudad, estado/provincia, Código Zip, Número de teléfono
(todos los campos son opcionales)</Requirement>
<Requirement id="2.2.3">Cantidad de personas que han consumido el alimento, cantidad de personas enfermas, cantidad de personas que
fueron enviadas al hospital y cantidad de personas fallecidas. Todos los campos son obligatorios</Requirement>
<Requirement id="2.2.4">Lugar donde los pacientes fueron tratados, alimento sospechoso. Los dos campos son requeridos</Requirement>
</Requirement>
<Requirement id="2.3">Quejas Especiales - DVISA
<Requirement id="2.3.1">Edad(obligatorio), Calificación académica(opcional), ocupación(opcional)</Requirement>

<Requirement id="2.3.2">calle, distrito, ciudad, estado/provincia, Código Zip, Número de teléfono del lugar más cercano al lugar de la queja. Todos los campos son opcionales</Requirement>
</Requirement>
</Requirement>
<Requirement id="3">Los ciudadanos pueden registrar una queja, mientras que los empleados pueden registrarla y actualizarla</Requirement>
<Requirement id="4">En el evento en que una queja está siendo interpuesta, será registrada en el sistema y direccionada al departamento específico</Requirement>
<Requirement id="4.1">Este departamento podrá manejar las quejas de una forma apropiada y devolverá una respuesta cuando la queja se haya tratado</Requirement>
<Requirement id="4.2">Esta respuesta será registrada en el sistema y estará disponible para ser buscada</Requirement>
</Requirement>
</Viewpoint>

4.3.1.2 XML Definido para cada Concern

```
<?xml version="1.0" ?>
    <Concern name="Disponibilidad">
        <Requirement id="1">El sistema deberá estar disponible las 24 horas los 7 días de la semana</Requirement>
        <Requirement id="2">El sistema por su naturaleza no será considerado un sistema crítico, por lo tanto este podrá estar apagado mientras se soluciona cualquier fallo que se presente</Requirement>
    </Concern>

<?xml version="1.0" ?>
    <Concern name="Seguridad">
        <Requirement id="1">Protocolo de Seguridad(encripción): El sistema deberá utilizar un protocolo de seguridad cuando envíe la información a través de internet</Requirement>
        <Requirement id="2">Control de Acceso: Para tener acceso a las características de el registro de quejas, el acceso debe ser controlado por un subsistema. Los empleados tienen un usuario y clave para utilizar el sistema.</Requirement>
    </Concern>

<?xml version="1.0" ?>
    <Concern name="Desempeño">
        <Requirement id="1">El sistema deberá ser capaz de manejar 20 usuarios simultáneamente</Requirement>
        <Requirement id="2">El tiempo de respuesta no debe exceder los 5 segundos</Requirement>
    </Concern>

<?xml version="1.0" ?>
    <Concern name="Concurrencia">
```

```
            <Requirement id="1">El sistema deberá ser capaz de manejar 20
usuarios simultáneamente</Requirement>
            <Requirement id="2">El sistema deberá estar en capacidad de
administrar los datos que se ingresen durante las sesiones que se
            encuentren activas evitando la corrupción de los datos. Sin embargo esto
no debe afectar el tiempo de respuesta para los usuarios
            </Requirement>
        </Concern>

<?xml version="1.0" ?>
        <Concern name="Persistencia">
            <Requirement id="1">El mecanismo de persistencia debe almacenar la
información relacionada con las quejas, los empleados, las unidades
            de salud, enfermedades, especialidades y ciudadanos</Requirement>
            <Requirement id="2">El sistema deberá ser flexible en términos de la
forma de almacenamiento permitiendo el uso de arreglos o
            diferentes bases de datos (MySQL, Oracle, etc.)</Requirement>
        </Concern>

<?xml version="1.0" ?>
        <Concern name="Distribución">
            <Requirement id="1">El sistema deberá permitir su ejecución en varias
máquinas. Por ejemplo, el core del sistema puede correr en una
            máquina y los Servlets en otra</Requirement>
        </Concern>

<?xml version="1.0" ?>
        <Concern name="Manejo de errors y excepciones">
            <Requirement id="1">Muchas de las funciones podrían generar errores
mientras que el usuario interactuar con el sistema, para esto se
            requieren diferentes técnicas de manejo:
                <Requirement id="1.1">Errores generales que aplican en la
mayoría de sus casos a información que no se ingresa(usuario no ingresa
                un campo requerido). El sistema muestra el error y señala los
campos que se requieren</Requirement>
                <Requirement id="1.2">Otros errores que pueden estar
relacionados al ingreso de información no válida al sistema. el mecanismo de
                manejo de errores debe evitar que esto suceda o mostrar el error y
sugerir la correción</Requirement>
            </Requirement>
        </Concern>

<?xml version="1.0" ?>
        <Concern name="Compatibilidad">
            <Requirement id="1">Como el Sistema Helath Watcher y el sistema
SSVS comparten información, los dos deben entonces utilizar el mismo
            mecanismo para el intercambio de datos.
                <Requirement id="1.1">Ambos sistemas deben utilizar el mismo
protocolo de intercambio de datos para evitar incompatibilidades en la
                integración</Requirement>
```

<Requirement id="1.2">La información intercambiada debe estar predefinida en el mismo tipo de formato(ejemplo: XML) </Requirement>
<Requirement id="1.3">Los issues de seguridad deberían estar orientados a garantizar el intercambio seguro de información entre
ambos sistemas </Requirement>
</Requirement>
</Concern>

<?xml version="1.0" ?>
<Concern name="**Usabilidad**">
<Requirement id="1">El sistema debe tener un fácil uso de la GUI, de forma tal que cualquier persona que tenga acceso a internet este
en capacidad de utilizarlo.</Requirement>
<Requirement id="2">El sistema deberá contar con una ayuda en línea que podrá ser consultada por cualquier usuario</Requirement>
<Requirement id="2.1">Muchas de las funcionalidades del sistema deben proveer tool tips u otras herramientas de ayuda que le
permitan al usuario conocer que información debe ser ingresada. </Requirement>
</Requirement>
</Concern>

<?xml version="1.0" ?>
<Concern name="**Cuestiones Legales**">
<Requirement id="1">El sistema debe ser desarrollado de acuerdo a loa estándares establecidos por la compañía X, responsable de las
normas de estandarización para los sistemas de la ciudad.</Requirement>
</Concern>

<?xml version="1.0" ?>
<Concern name="**Ambiente Operacional**">
<Requirement id="1">La interfaz de usuario debe ser implementada utilizando Servlets</Requirement>
<Requirement id="2">Software: Una licencia de Microsoft Windows para la estación de trabajo</Requirement>
<Requirement id="3">Hardware: One computer with: Pentiun III processor, 256 MB de memoria RAM, Tarjeta de red 3Com 10/100. Estos
equipos serán usados por los asistentes como máquinas de trabajo.</Requirement>
</Requirement>
</Concern>

4.3.2 Modelo de Diseño: Aspect Modeling Language (AML)

Para aplicar este modelo en el caso de estudio utilizaremos las siguientes convenciones

Aspect Package: Todos los early aspects definidos en 4.2.1

Base Package: Requisitos definidos en [12]

Connectores: Seran utilizados los Viewpoints definidos en 4.2.2

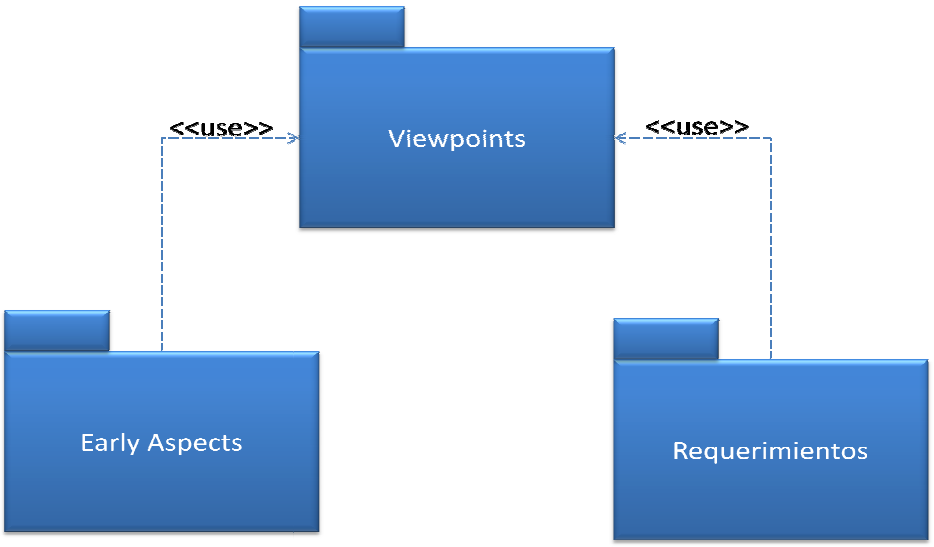


Figura 18.Modelo AML aplicado al caso de estudio Health Watcher.
Fuente: elaboración propia

4.4 Análisis de la trazabilidad

4.4.1 Relaciones de crosscutting Modelo AORE with Arcade

La tabla muestra los early aspects con los cuales los viewpoints hacen crosscut y cuales requerimientos son afectados por los early aspects.

Early Aspect	Viewpoints Crosscut(requisitos)
Disponibilidad	Empleado (requisito 2 y todos sus sub requisitos). Ciudadano (requisitos 2 al 4 y todos los sub requisitos)
Seguridad	Empleado (requisito 2 y todos sus sub requisitos). Ciudadano (requisitos 2 al 4 y todos los sub requisitos)

	Sistema SSVS (todos los requisitos)
Desempeño	Empleado (requisito 2 y todos sus sub requisitos). Ciudadano (requisitos 2 al 4 y todos los sub requisitos)
Concurrencia	Empleado (requisito 2 y todos sus sub requisitos). Ciudadano (requisitos 2 al 4 y todos los sub requisitos)
Persistencia	Empleado (todos los requisitos) Ciudadano (todos los requisitos) Queja (todos los requisitos)
Distribución	Empleado (requisito 2 y todos sus sub requisitos). Ciudadano (requisitos 2 al 4 y todos los sub requisitos)
Manejo de errores y excepciones	Empleado (requisito 2 y todos sus sub requisitos). Ciudadano (requisitos 2 al 4 y todos los sub requisitos)
Compatibilidad	Sistema SSVS (todos los requisitos)
Usabilidad	Empleado (requisito 2 y todos sus sub requisitos). Ciudadano (requisitos 2 al 4 y todos los sub requisitos)
Cuestiones jurídicas	Todos los viewpoints (excepto los del sistema SSVS) y todos los requisitos
Ambiente de operación	Todos los viewpoints (excepto los del sistema SSVS) y todos los requisitos

Tabla 6. Relaciones de crosscutting
Fuente: [11]

4.4.2 Trade off points y aspectos de interacción Modelo AORE with Arcade

Los trade off points son considerados para ser los requisitos de pertenencia a un viewpoint en que los diferentes early aspects aplican. Desde la perspectiva de Ingeniería de Requisitos estos puntos podrían plantear conflictos que deben investigarse más a fondo y tal vez involucren la negociación con las partes interesadas y como algunos de los early aspects podrían influir el uno al otro de manera negativa. La tabla muestra una discusión resumida de estas cuestiones de los trade off para cada viewpoint.

Viewpoint	Trade off points
Empleado	La mayoría los early aspects (disponibilidad, seguridad, rendimiento, concurrencia, persistencia, distribución, el error y de manejo de excepciones, facilidad de uso, las cuestiones jurídicas, el entorno operativo) se aplican a los viewpoints de empleados. El rendimiento contribuye generalmente negativamente a la seguridad y positivamente a la disponibilidad [1]. La distribución puede contribuir positivamente al rendimiento tanto como diferentes servidores pueden ser utilizados para el equilibrio de carga (aplicación y servidor web). La concurrencia puede contribuir negativamente a la persistencia como los usuarios simultáneos pueden corromper los datos, por lo que requiere una buena gestión de enfoque de transacciones. Manejo de errores, contribuye positivamente a la usabilidad ya que los usuarios pueden saber cuándo puede ocurrir errores y qué medidas tomar para corregirlos.
Ciudadano	Los mismos early aspects y explicación de empleado aplica.
Sistema SSVS	(Seguridad y Compatibilidad) principios de los aspectos aplicables a todos los requisitos de SSVS sistema. El vigilante de salud interactúa con el sistema SSVS sistema y la interacción debe respetar el mismo protocolo ser segura y al mismo tiempo. Garantizar la seguridad podría contribuir negativamente para la

	compatibilidad del mecanismo de seguridad, seleccione el protocolo para establecer el intercambio de datos entre los dos sistemas.
Queja	(Persistencia, cuestiones jurídicas y el funcionamiento del medio ambiente) aplicables al presente punto de vista. No hay restricción del entorno operativo en utilizar una base de datos específica y con respecto a las cuestiones jurídicas y normas de la empresa, estas deben ser analizadas para ver si alguna tiene limitaciones.

Tabla 7. Trade off points de los Viewpoints del caso de estudio
Fuente: [11]

4.4.3 AML aplicado al Caso de estudio: Health Watcher

Early aspect <<use>> View Point <<use>> Requisito

Disponibilidad <<use>> Ciudadano <<use>> FR01, FR02

Disponibilidad <<use>> Empleado <<use>> FR10, FR11, FR12, FR13, FR14, FR15, FR16

Disponibilidad <<use>> Queja <<use>> FR02, FR10, FR12

Disponibilidad <<use>> SSVS <<use>> FR11, FR15

Seguridad <<use>> Ciudadano <<use>> FR01, FR02, FR10

Seguridad <<use>> Empleado <<use>> FR10, FR11, FR12, FR13, FR14, FR15, FR16

Desempeño <<use>> SSVS <<use>> FR11, FR15

Desempeño <<use>> Ciudadano <<use>> FR01, FR02

Desempeño <<use>> Empleado <<use>> FR12, FR13, FR14

Concurrencia <<use>> Ciudadano <<use>> FR01, FR02, FR10

Concurrencia <<use>> Empleado <<use>> FR10, FR11, FR12, FR13, FR14, FR15, FR16

Persistencia <<use>> Empleado <<use>> FR11, FR14

Persistencia <<use>> Ciudadano <<use>> FR02

Manejo de Errores y excepciones <<use>> SSVS <<use>> FR02, FR13

Compatibilidad <<use>> SSVS <<use>> FR02, FR15

Usabilidad <<use>> Empleado <<use>> FR10, FR11, FR12, FR13, FR14, FR15, FR16

Usabilidad <<use>> Ciudadano <<use>> FR01, FR02

Cuestiones Jurídicas<<use>> SSVS <<use>> FR15

Ambiente de Operación <<use>> Empleado <<use>> FR10, FR11, FR12, FR13, FR14, FR15, FR16

5. CONCLUSIONES

- La trazabilidad permite que los analistas de desarrollo así como los ingenieros de requisitos tengan un pleno conocimiento de los artefactos del sistema y su relación entre ellos dentro de la aplicación como con otras aplicaciones.
- Las trazas en sentido horizontal y vertical permiten un mejor cubrimiento de los artefactos involucrados en un sistema de forma que se pueda hacer un seguimiento hacia adelante y hacia atrás dentro de un sistema.
- El uso del desarrollo orientado a aspectos desde el principio del ciclo de vida sirve para guiar todo el proceso de aspectos en todas las etapas del desarrollo.
- El diagrama de casos de uso es una herramienta fundamental y de vital importancia para el manejo de aspectos, pues estos son la materia prima para el diseño, modelado y construcción de los aspectos.
- Si se hace un correcto manejo de los crosscutting concerns, se puede llegar a obtener una mejor calidad en el software y un desarrollo ordenado de cada iteración en el desarrollo de software.
- Durante el desarrollo de este proyecto de grado se hizo visible la necesidad que existe en el medio de continuar profundizando en este tema, pues como se vio en el caso de estudio, un análisis somero de dos de los modelos bases generó una buena cantidad de artefactos que pueden y deberían ser utilizados dentro de un proceso formal de desarrollo para mejorar la trazabilidad interna de una aplicación.

6. GLOSARIO DE TERMINOS

Action Words

Son los verbos que se identifican en el documento de requisitos los cuales representan una actividad a ser ejecutada.

Advice

(*Consejo*) es la implementación del aspecto, es decir, contiene el código que implementa la nueva funcionalidad. Se insertan en la aplicación en los Puntos de Cruce.

AOSD

Abreviatura de "Aspect Oriented Software Development". Desarrollo de software orientado a aspectos. es una aproximación reciente para el desarrollo de software que permite la modularizacion del espacio del problema bajo una perspectiva dual, el punto de vista principal que define los módulos principales de la solución y la vista complementaria que separa la funcionalidad transversal en módulos independientes, llamados aspectos, con el fin de obtener un sistema más manejable y reusable.

Aspect (Aspecto)

Es la funcionalidad que se cruza a lo largo de la aplicación (crosscutting) que se va a implementar de forma modular y separada del resto del sistema. El ejemplo más común y simple de un aspecto es el logging (registro de sucesos) dentro del sistema, ya que necesariamente afecta a todas las partes del sistema que generan un suceso.

Asunto

Ver Concern.

Concern

Un concern (asunto) es un área de interés o un foco del sistema. Los concerns son el criterio primario para la descomposición de un software en partes más pequeñas, más manejables y más comprensibles. Ejemplos de concerns son: requisitos, casos de uso, características, estructuras de datos, issues de la calidad del servicio, variantes y patrones. Existen muchas formulaciones que han sido usadas para la captura de concerns, las cuales permiten una buena definición de los mismos como unidades separadas, los aspectos son uno de los varios mecanismos para separarlos.

Crosscutting

Es una relación estructural entre las representaciones de un concern. Así definido un concern es similar a otras clases de estructuras como la estructura jerárquica y la estructura en bloque. Crosscutting es un concepto diferente de scattering y tangling.

Crosscutting concern

Interés de corte transversal. El término "crosscutting" se utiliza para caracterizar un interés (concern) que atraviesa múltiples unidades de modularidad orientada a objetos (clases y objetos). Los "crosscutting concerns" son resistentes a la modularizacion utilizando constructores normales de orientación a objetos, pero los programas orientados a Aspectos pueden modularizar crosscutting concerns.

Entities

Son los “sustantivos” que están relacionados con las acciones de forma directa o indirecta.

Introduction (Introducción)

Permite añadir métodos o atributos a clases ya existentes. Un ejemplo en el que resultaría útil es la creación de un Consejo de Auditoría que mantenga la fecha de la última modificación de un objeto, mediante una variable y un método setUltimaModificacion(fecha), que podrían ser

introducidos en todas las clases (o sólo en algunas) para proporcionarlas esta nueva funcionalidad.

Join point (Punto de Cruce o de Unión)

Es un punto de ejecución dentro del sistema donde un aspecto puede ser conectado, como una llamada a un método, el lanzamiento de una excepción o la modificación de un campo. El código del aspecto será insertado en el flujo de ejecución de la aplicación para añadir su funcionalidad.

Modularity

Modularity es otro termina para hablar acerca de separation of concerns. El término *modularity* es usado cuando se habla de artefactos formales como el código por ejemplo y se refiere a alcanzar un alto nivel de claridad en la separación entre las unidades individuales (módulos). El código que implementa un concern es modular (es un módulo) si:

- Es textualmente local
- Existe una interfaz bien definida que describe como esta interactúa con el resto del sistema.
- La interface es una abstracción de la implementación, en que es posible hacer cambios materiales en la implementación sin violar la interfaz.
- Un mecanismo automático obliga a que cada modulo satisfaga sus propia interface y respete la interface de todos los otros módulos.
- El modulo puede ser automáticamente compuesto – por un compilador, loader, linker, etc. – en varas configuraciones con otros modulos para producir el sistema completo

Ortogonal

Dos o más cosas (de software) son ortogonales si los cambios en una de ellas no afectan a ninguna de las otras. La ortogonalidad, además de

eliminar efectos entre cosas no relacionadas, facilita los desarrollos con equipos grandes.

Pointcut (Puntos de Corte)

Define los Consejos que se aplicarán a cada Punto de Cruce. Se especifica mediante Expresiones Regulares o mediante patrones de nombres (de clases, métodos o campos), e incluso dinámicamente en tiempo de ejecución según el valor de ciertos parámetros.

Proxy (Resultante)

Es el objeto creado después de aplicar el Consejo al Objeto Destinatario. El resto de la aplicación únicamente tendrá que soportar al Objeto Destinatario (pre-AOP) y no al Objeto Resultante (post-AOP).

Separation of Concerns

En su nota de apertura en la primera conferencia de AOSD, Michael Jackson señaló que Djikstra es el hombre acreditado para recordarnos el principio Romano Divide y Conquistaras. Usándolo de otra manera la separación de concerns SOC es la simple idea que implica que un gran problema es más sencillo de manejar si puede ser dividido por partes; particularmente si la solución de los sub-problemas puede ser combinada para formar la solución del problema mayor.

SOC puede ser soportada en muchas formas: por procesos, por notación por organización, por mecanismos de lenguaje entre otros. Dentro del marco del tema de SOC, AOSD es distinguido por proveer nuevas formas de ver la separación of crosscutting concerns, en particular la idea de que las estructuras jerárquicas indivudiales están muy limitadas para obtener una separación efectiva de concerns en sistemas complejos..

Scattering

La representación de un concern es scattered (dispersa) sobre un artefacto si esta está dispersa en vez de localizada. Las representaciones de concerns son "tangled" dentro de un artefacto si estos están entremezclados en vez de separados. Scattering y tangling van de la mano, sin embargo son conceptos diferentes.

Tangling

Ver Scattering.

Target (Destinatario)

Es la clase aconsejada, la clase que es objeto de un consejo. Sin AOP, esta clase debería contener su lógica, además de la lógica del aspecto.

Trazabilidad

Se entiende como trazabilidad aquellos procedimientos preestablecidos y autosuficientes que permiten conocer el histórico, la ubicación y la trayectoria de un producto o lote de productos a lo largo de la cadena de suministros en un momento dado, a través de unas herramientas determinadas. En software es el grado de relación entre dos o más productos del proceso de desarrollo, especialmente productos que tienen una relación de predecesor – sucesor o de superior – subordinado con otro.

Weaving (Tejido)

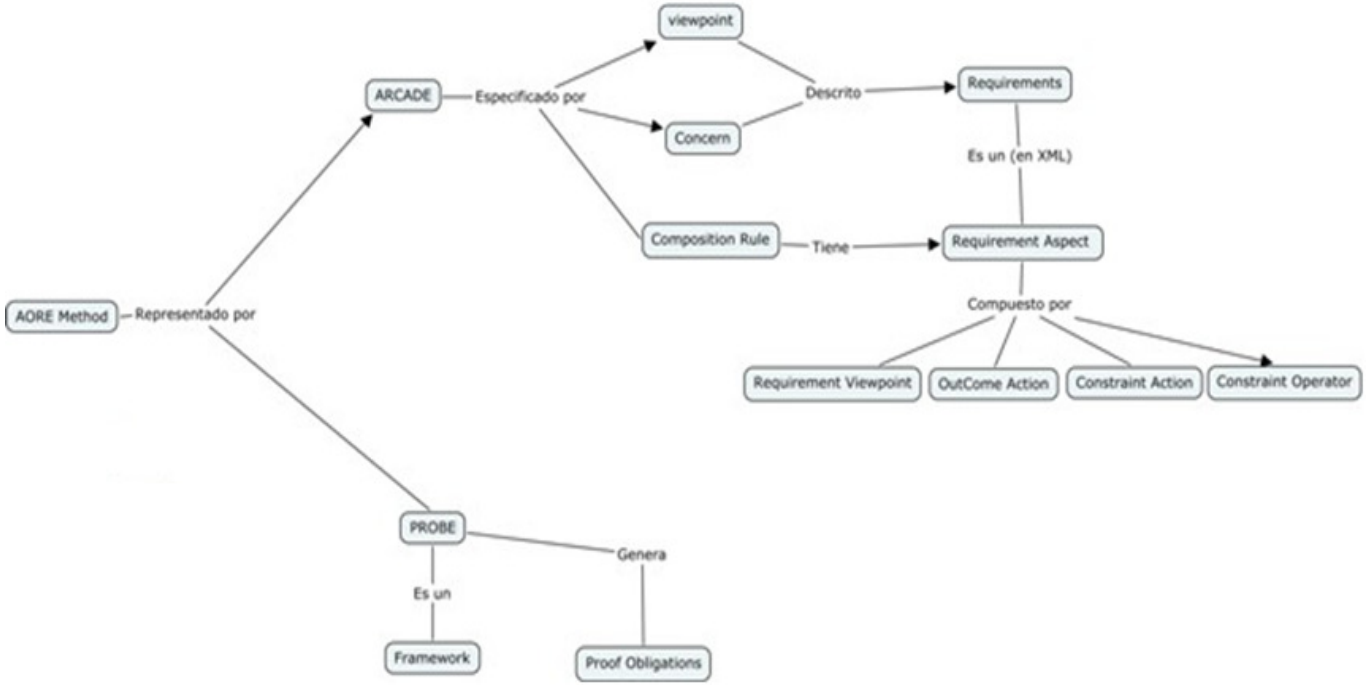
Es el proceso de aplicar Aspectos a los Objetos Destinatarios para crear los nuevos Objetos Resultantes en los especificados Puntos de Cruce. Este proceso puede ocurrir a lo largo del ciclo de vida del Objeto Destinatario:

- Aspectos en Tiempo de Compilación, que necesita un compilador especial.

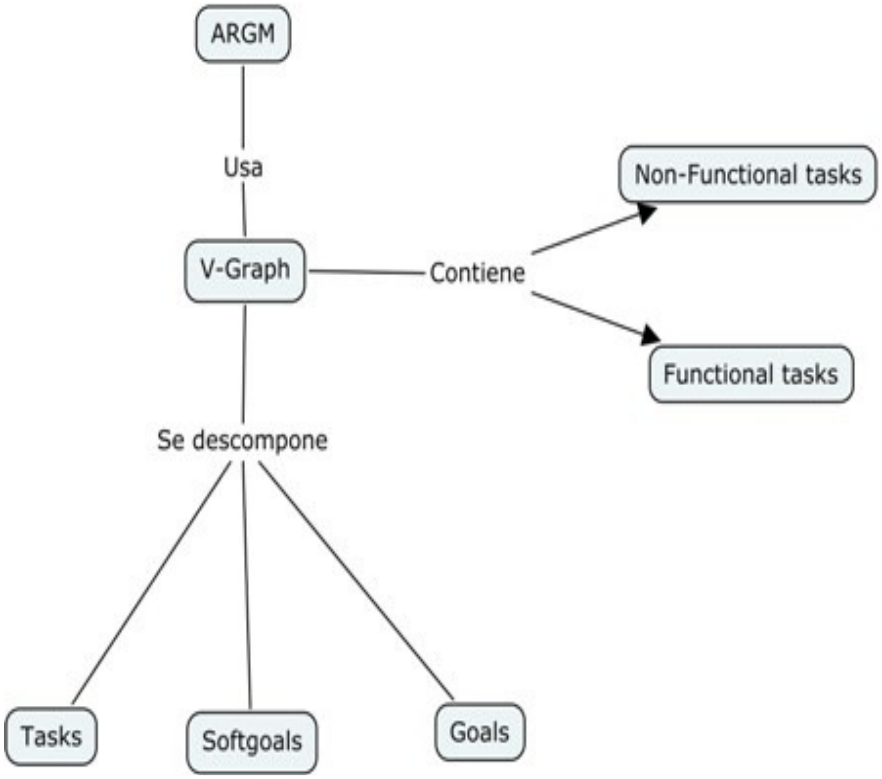
- Aspectos en Tiempo de Carga, los Aspectos se implementan cuando el Objeto Destinatario es cargado. Requiere un ClassLoader especial.
- Aspectos en Tiempo de Ejecución.

7. ANEXOS

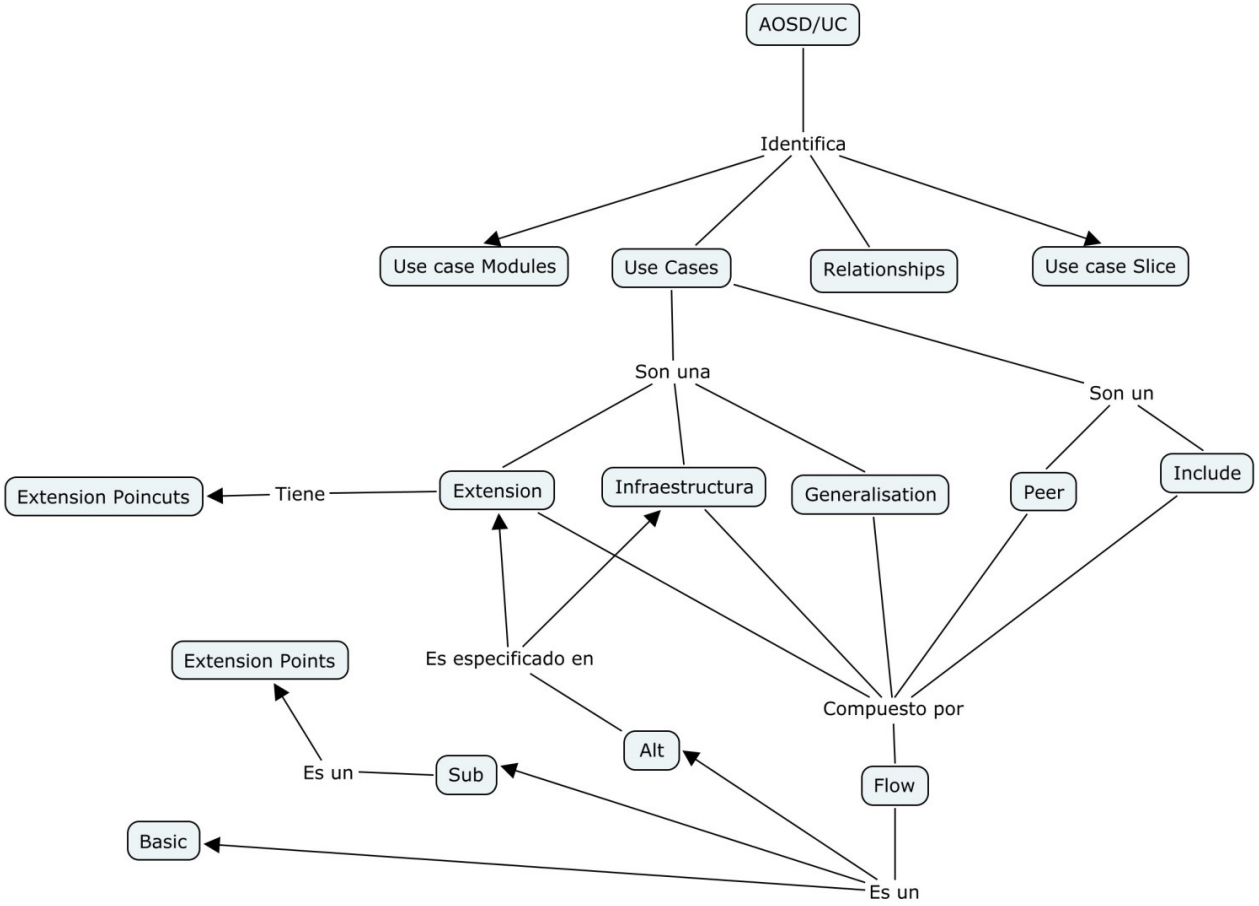
En esta sección se encuentran los mapas conceptuales elaborados por nosotros para cada uno de los métodos y aproximaciones tratados en el proyecto de grado. Se elaboraron para que el lector tenga una mayor claridad acerca del tema en estudio.



Anexo 1. Mapa conceptual Método AORE.
Fuente: elaboración propia

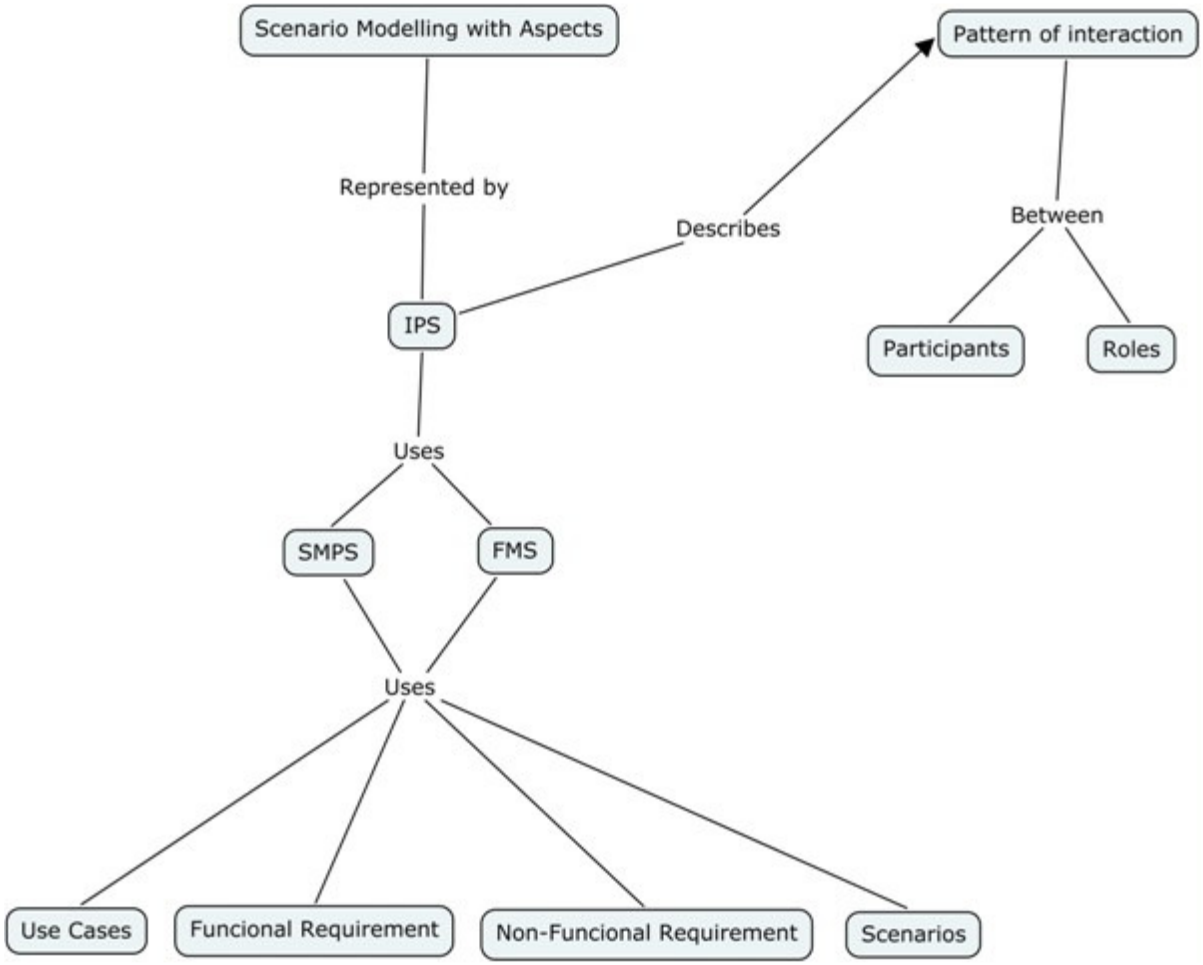


Anexo 2. Mapa conceptual ARGM.
Fuente: Elaboración propia



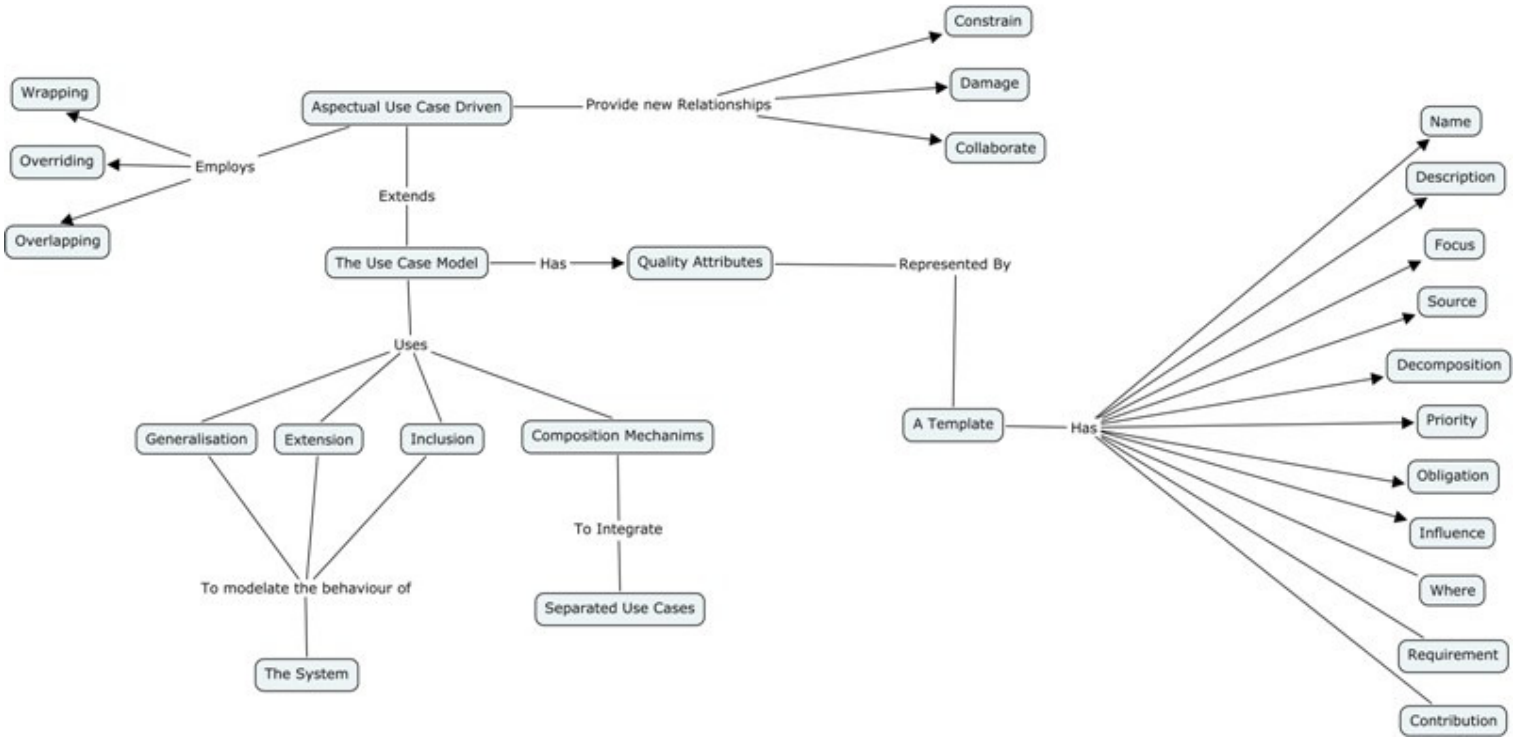
Anexo 3. Mapa conceptual AOSD/UC

Fuente: Elaboración propia



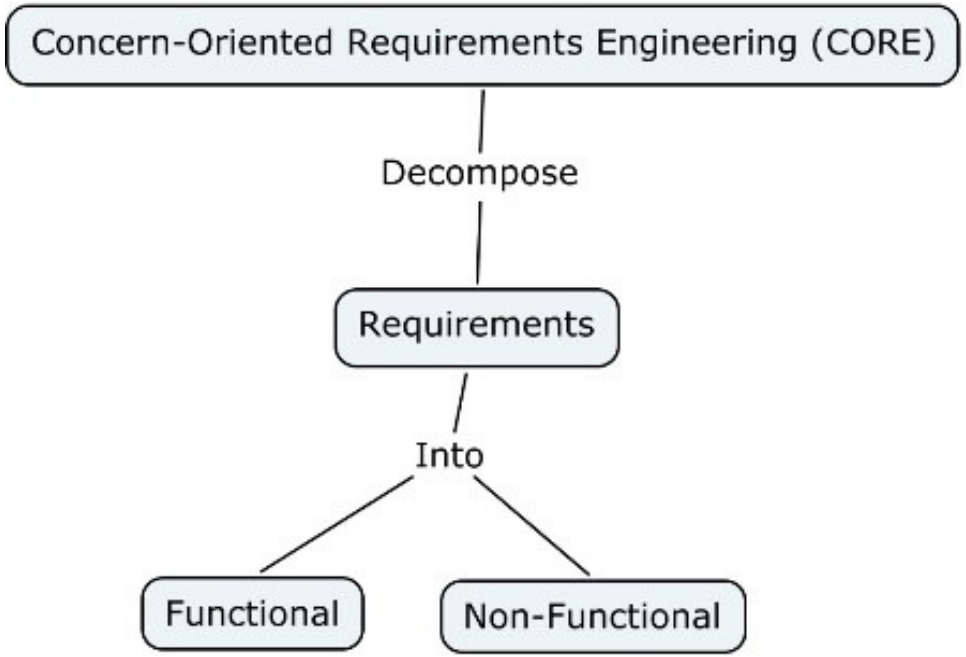
Anexo 4. Mapa conceptual Scenerio Modeling with aspects artifacts

Fuente: elaboración propia

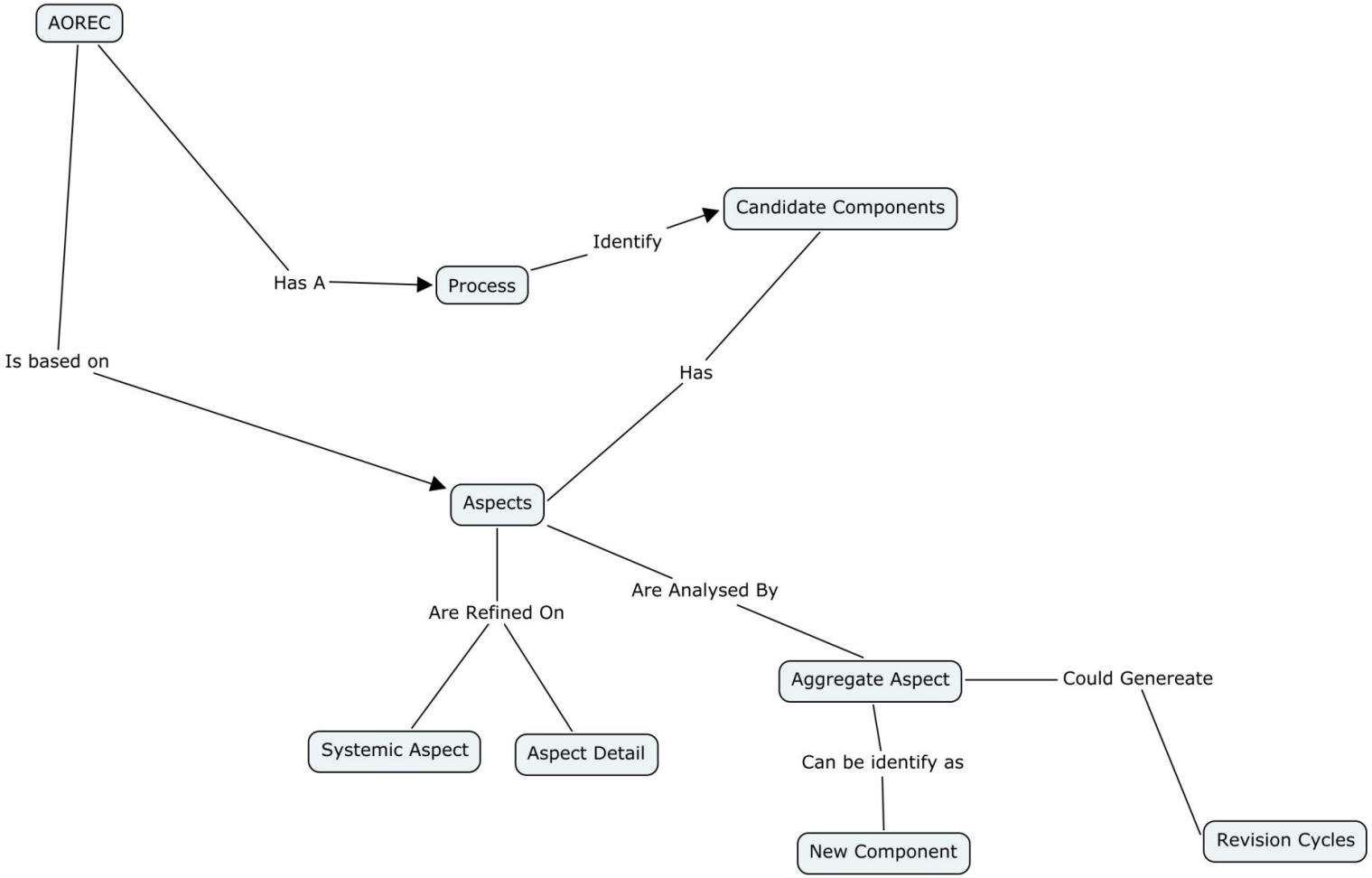


Anexo 5. Mapa conceptual Aspectual Use Case Driven Approach

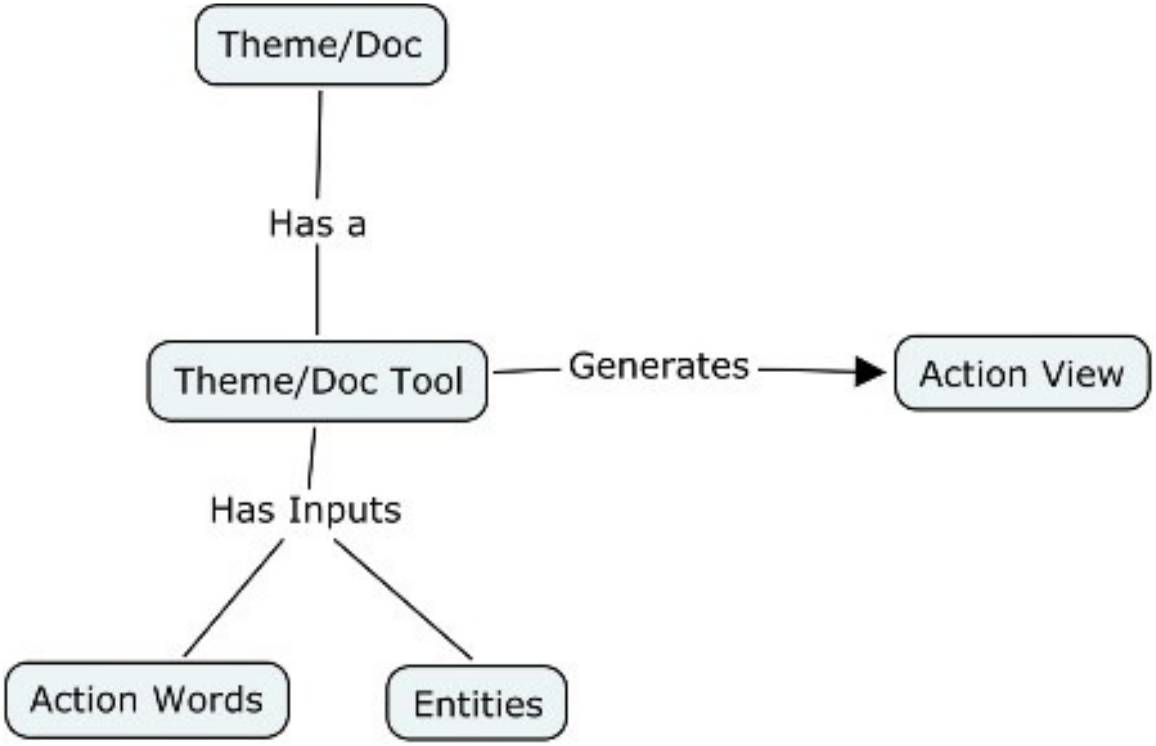
Fuente: Elaboración propia



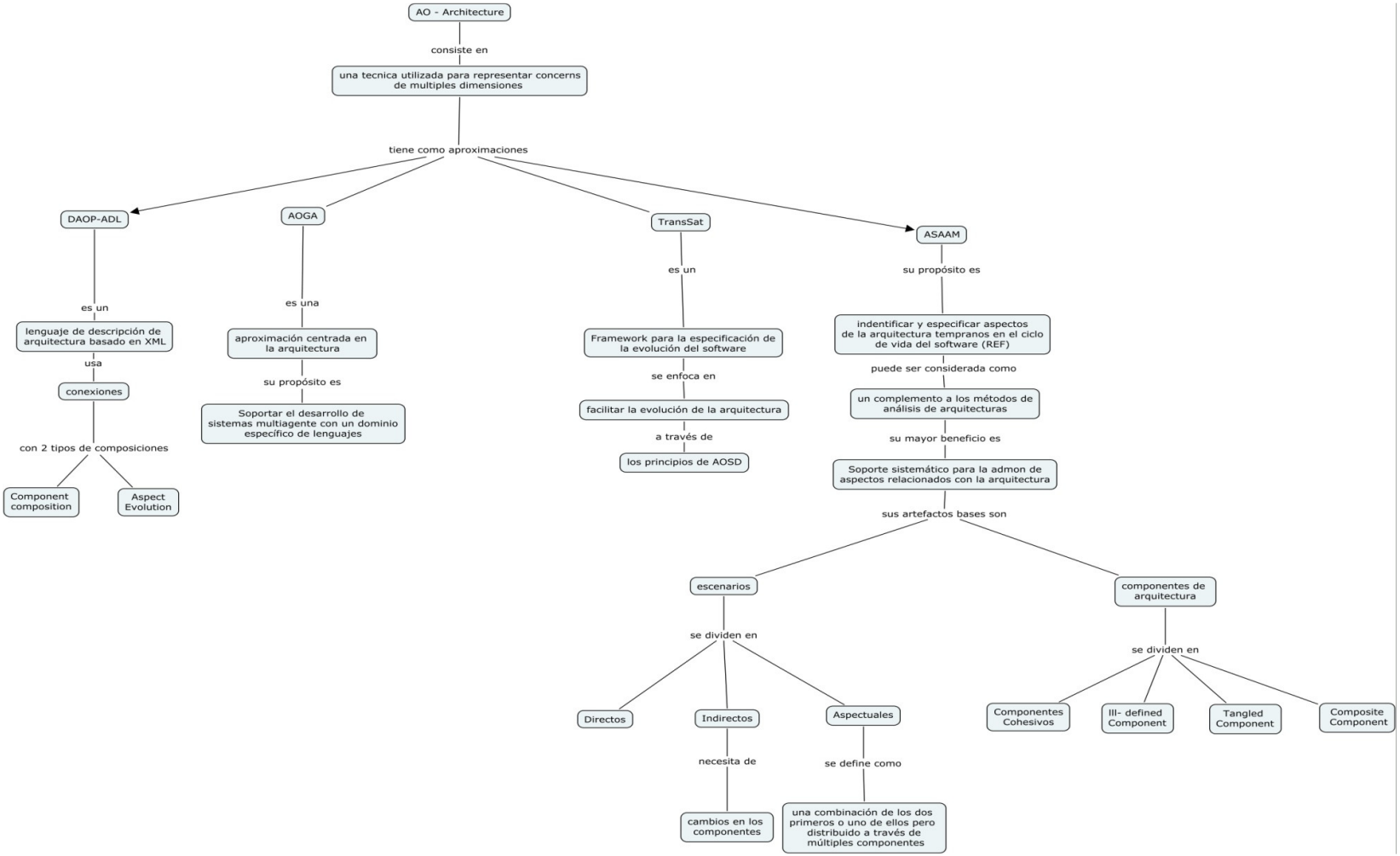
Anexo 6. Mapa conceptual CORE
Fuente: Elaboración propia



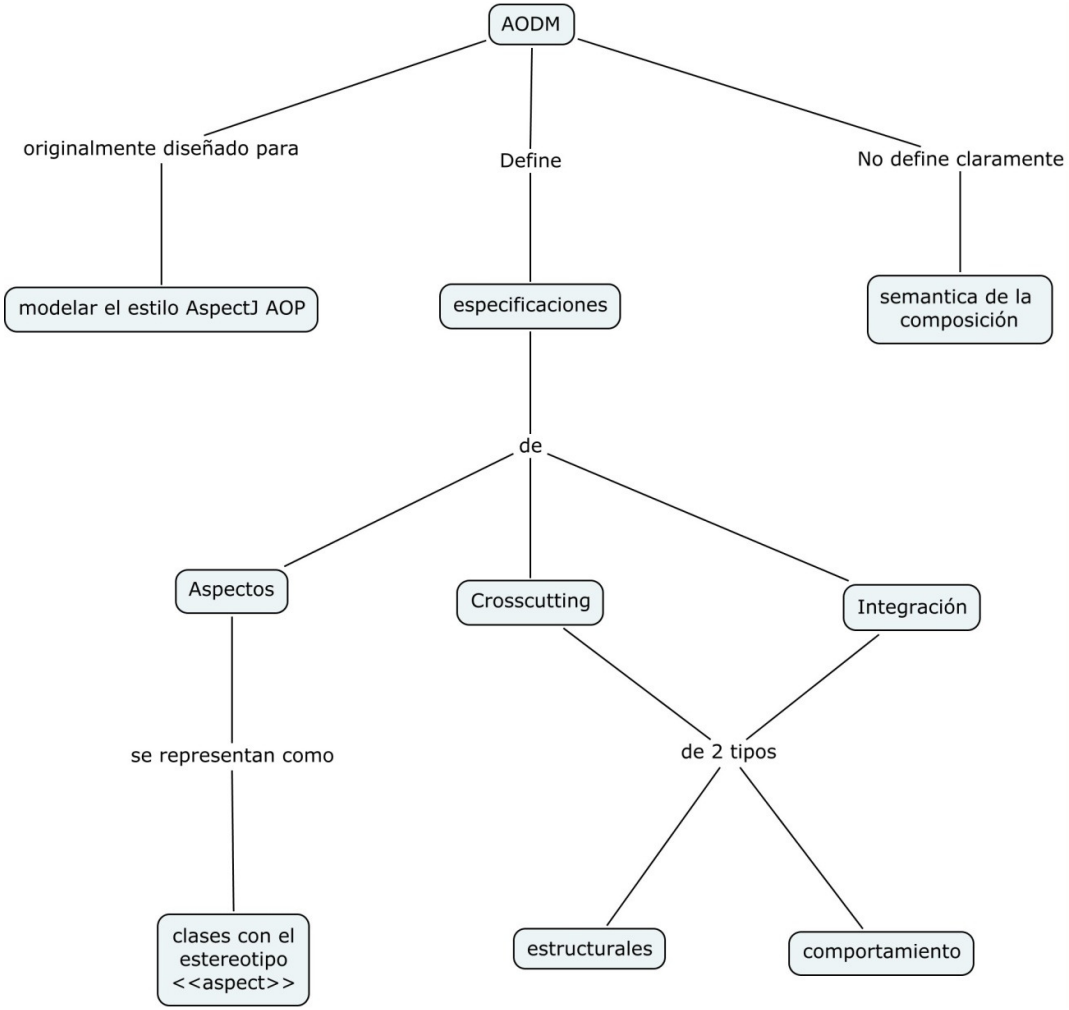
Anexo 7. Mapa conceptual AOREC
Fuente: elaboración propia



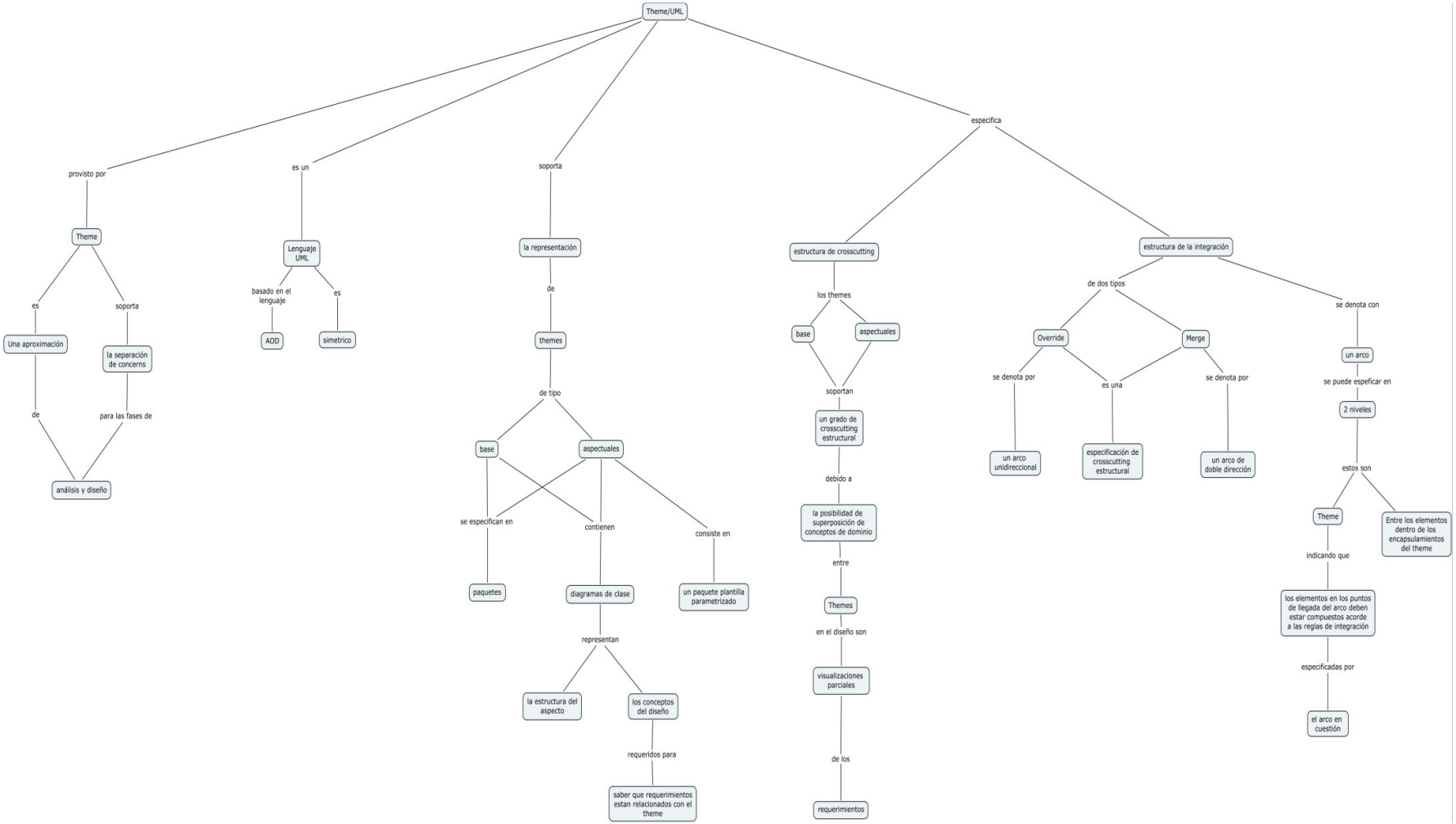
Anexo 8. Mapa conceptual Theme/Doc
Fuente: Elaboración propia



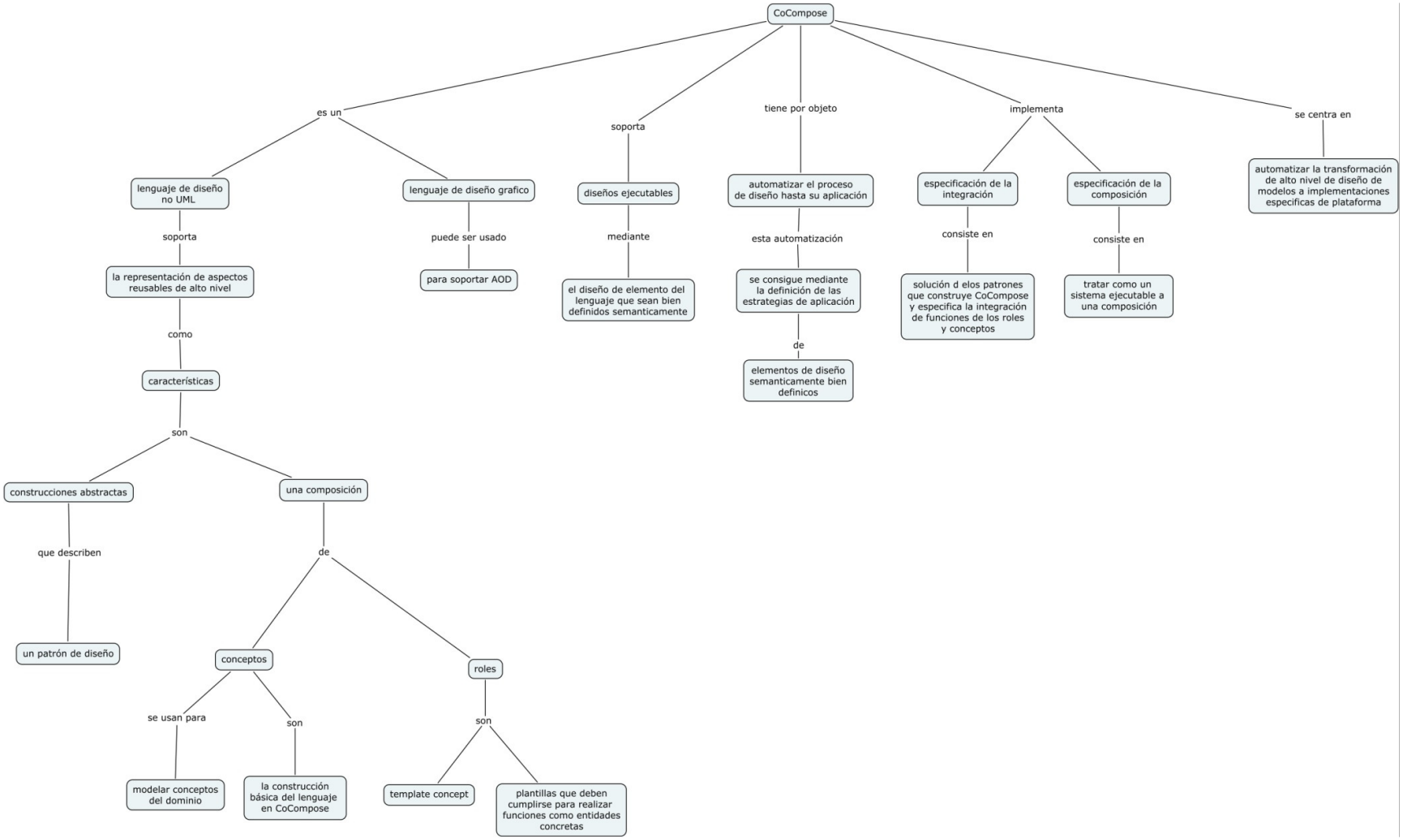
Anexo 9. Mapa conceptual AO Architecture
Fuente: Elaboración propia



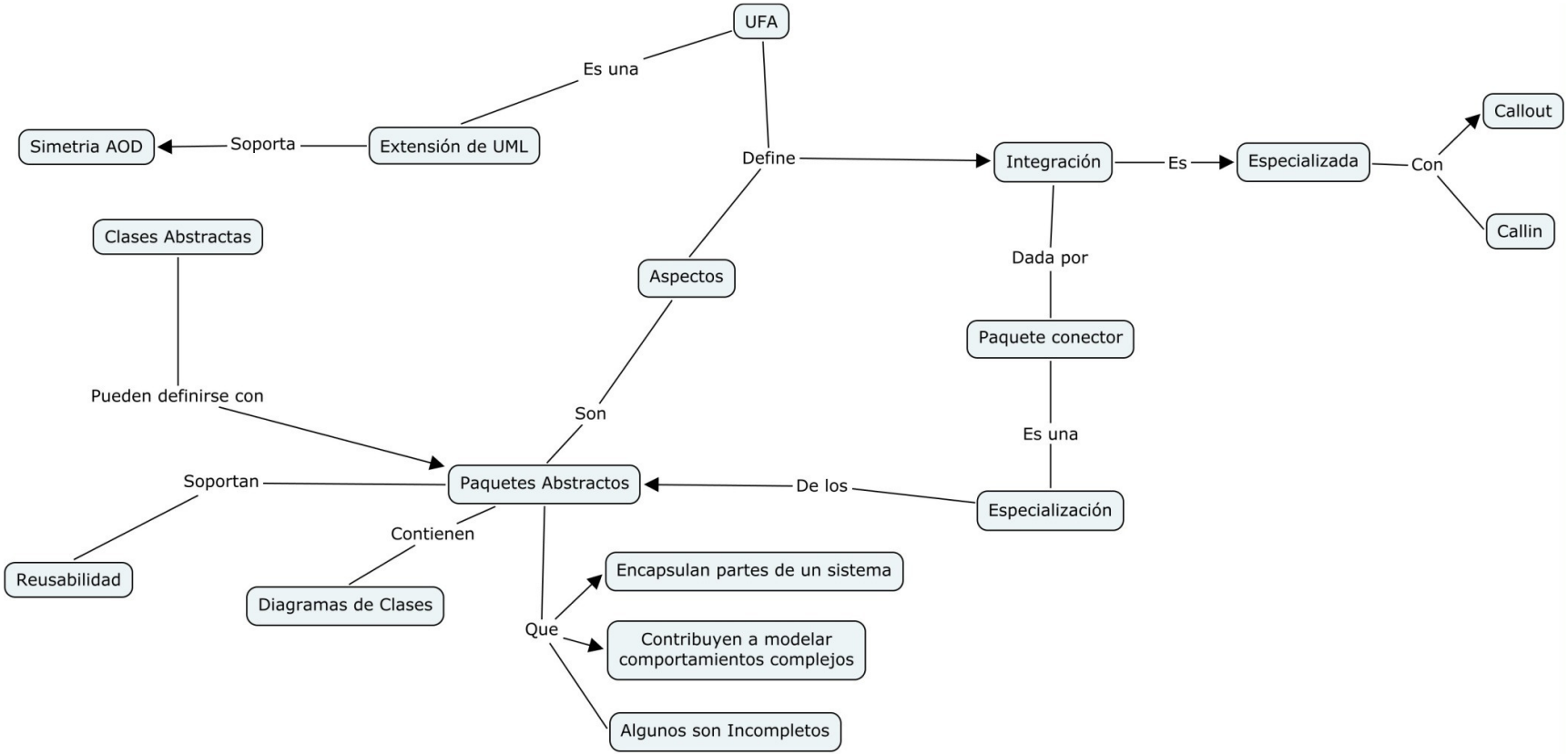
Anexo 10. Mapa conceptual AODM
Fuente: elaboración propia



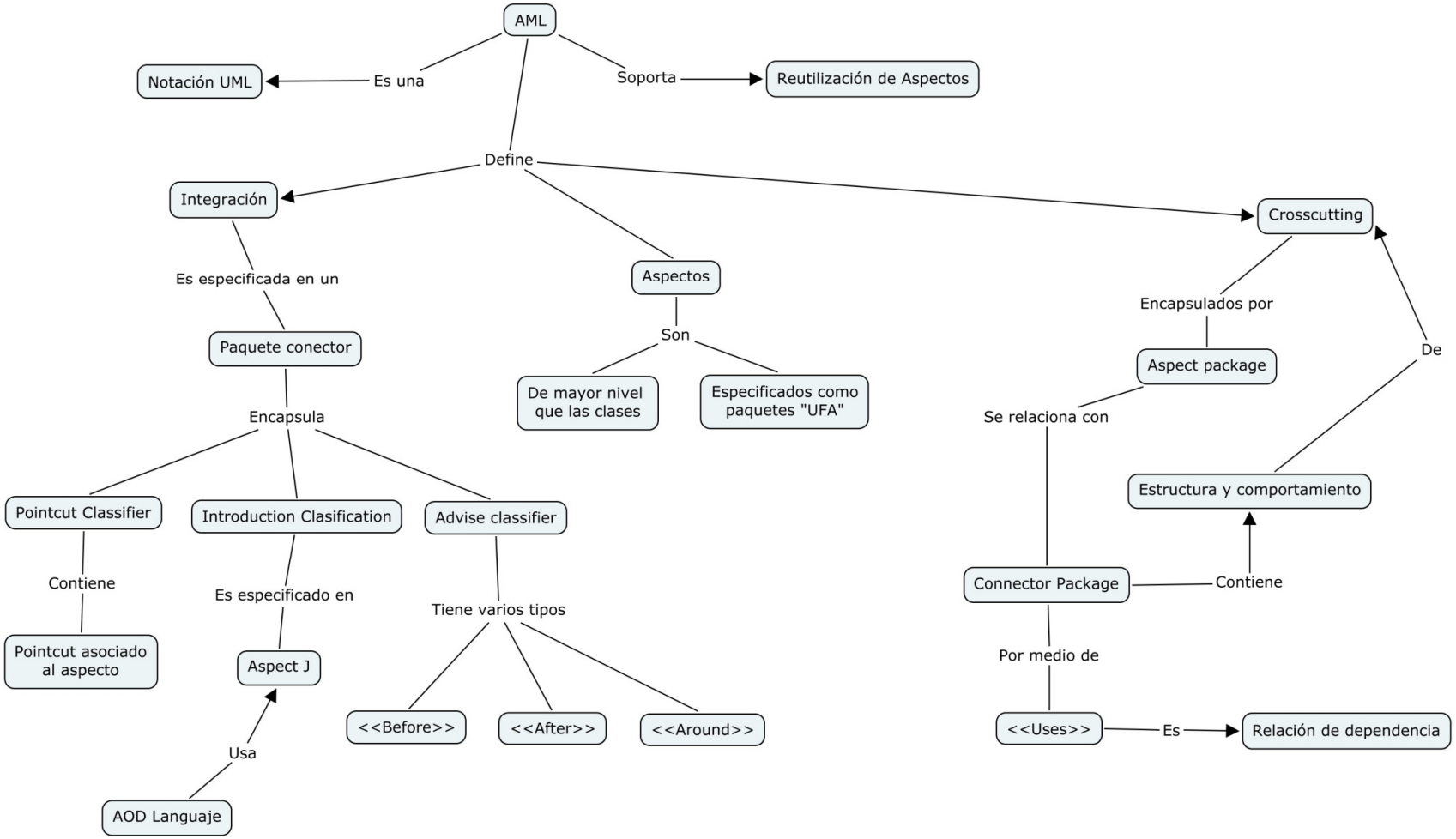
Anexo 11. Mapa conceptual Theme/UML .
Fuente: Elaboración propia



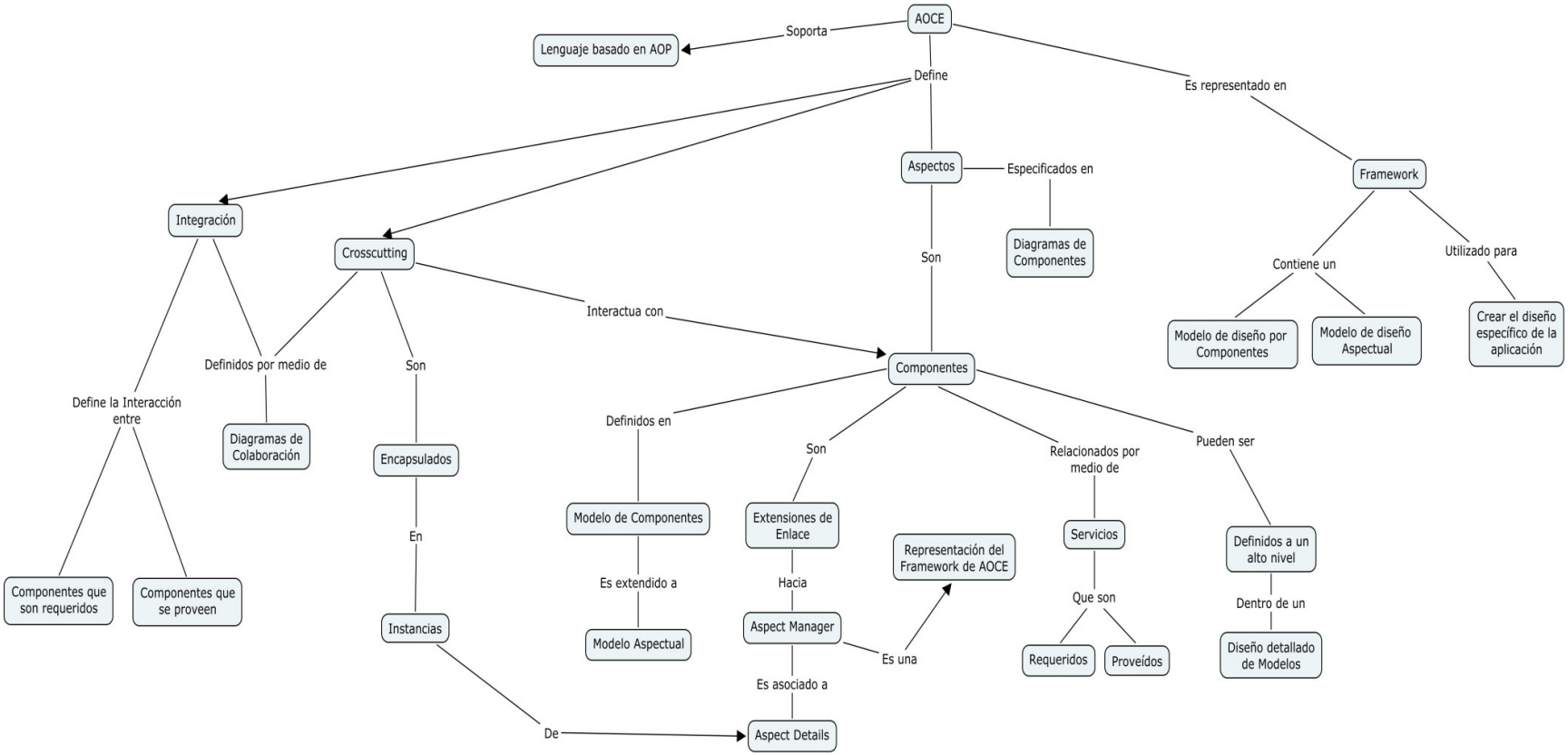
Anexo 12. Mapa conceptual CoCompose. Fuente: Elaboración propia



Anexo 13. Mapa conceptual UFA.
Fuente: Elaboración propia



Anexo 14. Mapa conceptual AML. Fuente: Elaboración propia



Anexo 15. Mapa conceptual AOCE.
Fuente: Elaboración propia

8. REFERENCIAS

[1] www.aosd.net, www.early-aspects.net . Visitado: mayo 6 de 2009

[2] Chitchyan, R; Rashid, A; Sawyer, P (2005). A Survey of aspect- and design approaches.

[3] Pfleeger, S; Bohner, S (1990). A framework for software maintenance metrics.

[4] Lindvall, M; Sandahl, K (1996). "Practical implications of traceability," Software practice and experience

[5] Gotel, Orlena C; Finkelstein, Z; Anthony,C (1993).An analysis of the requirements traceability problem.

[6] Pinheiro, F; Goguen; Joseph, A (1996). An object-oriented tool for tracing requirements

[7] Tabares, M; Alferez ,G H; Alferez, E (2008).Desarrollo de software orientado a aspectos: un caso práctico para un sistema de ayuda en línea.

[8] Groher, I; Schulze (2003). "Generating aspect code from UML models."

[9] Haak, B; Díaz, M; Marcos, C; Pryor, J (2005). Identificacion temprana de aspectos.

[10] Soares,S; Laureano, E; Borba, P (2002)."Implementing distribution and persistence aspects with AspectJ."

[11] Sampaio, A (2007).Analysis of health watcher system using viewpoint-based AORE and the EA-Miner tool.

[12] Massoni, T; Soares, S; Borba, P (2006).Health_watcher_requirements_v2_1.

[13] http://www.aosd.net/wiki/index.php?title=Glossary. Visitado: mayo 6 de 2009