PROGRAMACIÓN ORIENTADA A ASPECTOS: MANEJO DE REQUISITOS CON ASPECTOS

CARLOS ANDRES OSPINA VALENCIA
CARLOS ANDRES PARRA DURAN

UNIVERSIDAD EAFIT
DEPARTAMENTO DE SISTEMAS
ESCUELA DE INGENIERÍA
MEDELLÍN
2007

PROGRAMACIÓN ORIENTADA A ASPECTOS: MANEJO DE REQUISITOS CON ASPECTOS

CARLOS ANDRES OSPINA VALENCIA
CARLOS ANDRES PARRA DURAN

TRABAJO DE GRADO PARA OPTAR AL TÍTULO DE
INGENIERO DE SISTEMAS

ASESOR
LUIS FERNANDO LONDOÑO LONDOÑO
GERENTE INVESTIGACIÓN Y DESARROLLO AVANSOFT S.A

UNIVERSIDAD EAFIT
DEPARTAMENTO DE SISTEMAS
ESCUELA DE INGENIERÍA
MEDELLÍN
2007

Nota de aceptación:

Firma del presidente del jurado

Firma del jurado

Firma del jurado

A Dios, a nuestras familias y a todas las personas que nos han apoyado para alcanzar las metas en nuestro proceso de formación.

AGRADECIMIENTOS

Expresamos nuestros agradecimientos a todas las personas que han permitido realizar el presente trabajo:

A Luis Fernando Londoño, por su dedicación, su apoyo y todos los aportes que brindó para la elaboración de esta investigación. A la Doctora Raquel Anaya, por sus valiosos aportes y el continuo interés en el trabajo realizado.

A AVANSOFT S.A., por brindarnos el espacio y los recursos requeridos para la ejecución de este trabajo. A nuestros compañeros del día a día, Cathalina Vallejo, Lenin Lozano, Jhon Jairo Díaz, entre muchos otros, por sus revisiones, aportes, críticas y en general por todo su apoyo.

Al grupo de interés de Ingeniería de Software de la universidad EAFIT, por brindarnos un espacio de discusión y por todas las críticas, ideas y aportes recibidos durante la ejecución del proyecto.

Al proyecto MMEDUSA -Marco Metodológico para Desarrollo de Aplicaciones Utilizando la Aproximación de Aspectos- y todas las personas involucradas en el desarrollo de este, que al igual que nosotros están trabajando por sacar adelante tan importante investigación.

Finalmente, nuestra más grande manifestación de gratitud para nuestras familias y demás allegados, quienes nos han dado todo su apoyo durante nuestro proceso de formación.

CONTENIDO

AGRADECIMIENTOS **5**

CONTENIDO **6**

LISTA DE FIGURAS **9**

LISTA DE TABLAS **11**

LISTA DE ANEXOS **12**

INTRODUCCION **13**

1. FUNDAMENTOS DEL PARADIGMA ORIENTADO A ASPECTOS **17**

1.1 La evolución de la ingeniería de software **19**

1.2 Antecedentes del Desarrollo Basado en Aspectos **20**
1.2.1 Modelos de programación aspectual 21
1.2.2 El paradigma Orientado a Objetos no es suficiente 24
1.2.2.1 Hechos y hallazgos 24
1.2.2.2 El problema: tangling y scattering 27

1.3 Así trabaja el paradigma orientado a aspectos **28**
1.3.1 La base del paradigma: Clases y Aspectos 29
1.3.1.1 Funcionalidad básica 29
1.3.1.2 Comportamiento aspectual 30
1.3.1.3 Puntos de enlace 31
1.3.2 Tejedor (weaver) 32
1.3.2.1 Tejido estático 32
1.3.2.2 Tejido dinámico 32
1.3.3 Aplicaciones bajo el paradigma orientado a aspectos 33

1.4 Aspectos en etapas tempranas **35**

1.5 Definiciones básicas en AOSD **36**
1.5.1 Concern (Interés) 37
1.5.2 Separation of concerns (Separación de interese) 38
1.5.3 Crosscutting concern (Interés de corte transversal) 39
1.5.4 Aspect (Aspecto) 40

2. METODOLOGÍAS DE MANEJO DE REQUISITOS BAJO EL ENFOQUE ORIENTADO A ASPECTOS. **41**

2.1 Algunas aproximaciones al manejo de requisitos aspectuales 42
2.1.1 Theme /Doc 43
2.1.2 Modularization and composition of aspectual requirements - AORE 45
2.1.3 Integrating the NFR framework in a RE model - INFR 46
2.1.4 COSMOS 47
2.1.5 Resumen metodologías para el manejo de requisitos aspectuales 49

2.2 Profundizando las metodologías: AORE y COSMOS....51
2.2.1 AORE....52
2.2.1.1 El modelo AORE 53
2.2.1.2 Espacio de concerns 53
2.2.1.3 Espacio de meta-concerns y espacio de sistema 54
2.2.1.4 Modelo de ingeniería de requisitos 55
2.2.1.5 Ejemplo de aplicación de AORE 68
2.2.1.6 Observaciones al modelo AORE....83
2.2.2 COSMOS....85
2.2.2.1 El modelo COSMOS 86
2.2.2.2 Elementos del modelo....87
2.2.2.3 Ejemplo de aplicación de COSMOS 94
2.2.2.4 Resumen y análisis del caso de estudio original 96
2.2.2.5 Análisis 100
2.2.2.6 Relaciones....103
2.2.2.7 Observaciones al modelo COSMOS....106

2.3 Ampliando el modelo AORE 108
2.3.1 Hacia un modelo integrado AORE - COSMOS 109
2.3.2 Modelo AORE extendido 110
2.3.2.1 Elementos de ampliación del modelo 112
2.3.2.2 Clasificar concerns 112
2.3.2.3 Complementar relaciones de grano grueso entre concerns 116
2.3.2.4 Generar vistas de concerns 117

3. ENTERPRISE ARCHITECT COMO HERRAMIENTA PARA SOPORTAR EL MODELO 119

3.1 Razones para utilizar Entreprise Architect - EA 120

3.2 Propuesta de modelado 121
3.2.1 El entorno de trabajo 121
3.2.2 Estructura de artefactos....123
3.2.3 Definición de concerns 125
3.2.4 Definición de requisitos de los concerns 126
3.2.5 Relaciones de grano grueso....127
3.2.6 Relaciones de grano fino....129
3.2.7 Relaciones de grano grueso refinadas....134
3.2.8 Manejo de conflictos....135
3.2.9 Vistas de concerns 136

3.3 Observaciones y consideraciones....137

4. APLICACIÓN DEL MODELO 139

4.1 Presentación del caso de aplicación 139

4.2 Aplicación del modelo 140
4.2.1 Identificación de concerns 140
4.2.2 Clasificar los concerns. 142
4.2.3 Identificación de relaciones de grano grueso 143
4.2.4 Identificación de relaciones de grano fino 147
4.2.5 Refinamiento de relaciones de grano grueso 150
4.2.6 Solución de conflictos. 152
4.2.7 Vistas de concerns 158
4.2.8 Dimensiones de concerns 163
Observaciones 164

5. CONCLUSIONES 166

GLOSARIO 169

BIBLIOGRAFIA 174

LISTA DE FIGURAS

Figura 1. Transición entre requerimientos y diseño / implementación. [19] 28
Figura 2. Construcción de aplicaciones bajo el esquema tradicional. [20].....33
Figura 3. Construcción de aplicaciones bajo el paradigma orientado a aspectos. [20].....34
Figura 4. Estructura de un programa orientado a aspectos. [20] 35
Figura 5. Relación entre aspectos tempranos, intermedios y finales. [13] 36
Figura 6. Espacio de concerns [27].....53
Figura 7. Espacio de meta concerns y espacio de sistema [14]. 55
Figura 8. Modelo de ingeniería de requisitos basado en tratamiento uniforme de concerns [27] 56
Figura 9. Plantillas para identificación de concerns. 57
Figura 10. Matriz de relaciones de grano grueso. 58
Figura 11. Plantillas para la especificación de proyecciones 60
Figura 12. Interpretación de las reglas de composición por los stakeholders. 63
Figura 13. Matriz de contribución entre concerns. 64
Figura 14. Tabla de contribuciones doblada por la diagonal. 65
Figura 15. Proyecciones reflejadas. 65
Figura 16. Ampliación del modelo AORE.....112
Figura 17. Clasificación de concerns 115
Figura 18. Vista del entorno de trabajo Enterprise Architect.....122
Figura 19. Vista de proyecto para el modelo AORE extendido. 124
Figura 20. Especificación de concerns 125
Figura 21. Especificación de los requisitos del concern.....127
Figura 22. Diagrama de relaciones de grano grueso. 128
Figura 23. Matriz de relación entre concerns en EA 129
Figura 24. Diagrama de relaciones de grano fino 130
Figura 25. Especificación de acción y operador de la composición. 131

Figura 26. Especificación del resultado de la composición. 132
Figura 27. Especificación de los requisitos afectados por el concern. 133
Figura 28. Diagrama de relaciones de grano grueso refinado. 134
Figura 29. Manejo de conflictos en EA. 136
Figura 30. Vista de concerns orientada a la plataforma. 137
Figura 31. Relaciones de concerns de grano grueso 144
Figura 32. Especificación de relaciones de grano fino para el concern Evaluaciones. 148
Figura 33. Especificación de relaciones de grano fino para el concern Competencias 149
Figura 34. Relaciones de grano fino para el concern Ergonomía. 150
Figura 35. Relaciones de grano grueso refinadas. 151
Figura 36. Vista de negocio. 159
Figura 37. Vista funcional 160
Figura 38. Vista QoS - Quality of service 161
Figura 39. Vista de plataforma 162
Figura 40. Vista de proyecto. 163

LISTA DE TABLAS

Tabla 1. Evaluación de aproximaciones a aspectos tempranos. [25] 50
Tabla 2. Acciones de restricción (Constraint actions) 61
Tabla 3. Operadores de restricción (Constraint operators) 62
Tabla 4. Acciones de resultado (Outcome actions).....62
Tabla 5. Especificación de las dimensiones de concerns. 67
Tabla 6. Matriz de relaciones entre concerns 78
Tabla 7. Matriz de contribución entre concerns 82
Tabla 8. Outline of the COSMOS concern-space modeling schema. Basada en [23] y [24].....93
Tabla 9. Elementos identificados en la aplicación de COSMOS.....99
Tabla 10. Clasificación de concerns en classes e instances. 101
Tabla 11. Clasificación de concerns topics. 102
Tabla 12. Otras clasificaciones de concerns. 102
Tabla 13. Relaciones identificadas en la aplicación de COSMOS. 106
Tabla 14. Clasificaciones propuestas para concerns. 114
Tabla 15. Relaciones de grano grueso 117
Tabla 16. Vistas de concerns. 118
Tabla 17. Clasificación de los concerns del sistema 143
Tabla 18. Matriz de relaciones entre concerns 146
Tabla 19. Matriz de contribución. 154
Tabla 20. Proyecciones reflejadas de los concerns. 157
Tabla 21. Especificación de las dimensiones de los concerns. 164

LISTA DE ANEXOS

Anexo A. Catalogo de requisitos del caso de aplicación: ESPECIFICACIÓN DE REQUISITOS GESCOM 1.0.

Anexo B. Archivos de Enterprise Architect con implementación del caso de aplicación.

Anexo C. Instaladores de Enterprise Architect Viewer.

INTRODUCCION

La ingeniería de software es un área en continuo desarrollo y en la cual día a día surgen nuevas propuestas y metodologías que pretenden ofrecer técnicas para desarrollar y mantener software de calidad que resuelve problemas de todo tipo. La investigación en el campo de la Ingeniería de Software nos enfrenta a una nueva aproximación para el desarrollo de software: El desarrollo basado en aspectos. Al igual que el desarrollo basado en objetos, el desarrollo basado en aspectos surge primero como propuesta a nivel de lenguaje de programación y luego empieza a tomar fuerza como aproximación en todas las fases del desarrollo, conocida como Aspect Oriented Software Development (AOSD). Sin embargo, dada su reciente aparición, buena parte de las experiencias de aplicación se limitan a ejercicios de tipo académico y por lo tanto existen expectativas a nivel mundial acerca del impacto real de la orientación a aspectos en la productividad y calidad de una solución software a escala industrial.

En este contexto, surge el proyecto MMEDUSA, un esfuerzo conjunto de la Universidad EAFIT y AVANSOFT S.A, el cual pretende brindar un *Marco Metodológico para Desarrollo de Aplicaciones Utilizando la Aproximación de Aspectos*, convirtiéndose en pionero del AOSD en el medio. Lograr posicionar un proyecto que divulgue guías y resultados alrededor de esta nueva aproximación metodológica, permitirá dar un paso de avanzada para disminuir la brecha de adopción tardía de tecnología y para capitalizar y fortalecer el contacto y apoyo de grupos reconocidos en el tema a nivel mundial.

Es importante resaltar la labor del grupo de Ingeniería de Software de la Universidad EAFIT, el cual tiene como misión “proponer ante la comunidad informática estándares, modelos, métodos y herramientas para el desarrollo de software apoyado en principios de calidad”, a través de sus dos líneas de trabajo

aseguramiento de la calidad y metodologías y modelos de desarrollo de software. Para lograr que la investigación tenga impacto en la práctica diaria del desarrollo de software de las empresas del sector, el grupo tiene como estrategia buscar alianzas de investigación con industrias de software reconocidas.

Paralelamente, AVANSOFT S.A. como una empresa del sector de desarrollo de Software, cuenta con un grupo de ingenieros que tienen como objetivo principal, evaluar, seleccionar y definir prácticas adecuadas de desarrollo de software y por tanto mantener un nivel alto de desarrollo e innovación en tecnologías de la información para la compañía. Este grupo de Ingenieros hace parte de la Gerencia de I+D de la organización y orienta sus esfuerzos en la investigación, desarrollo y adaptación de metodologías, modelos de desarrollo, modelos de evaluación arquitecturas de software, prácticas y procesos de ingeniería de software orientados a la calidad, definición de mecanismos de medida y análisis orientados, que permitan que los procesos de la cadena de valor sean herramientas para mantener un nivel de competitividad a nivel nacional e internacional basados en calidad y productividad.

A partir de esto, como estudiantes de pregrado de la universidad EAFIT y empleados de AVANSOFT S.A, con el apoyo de la gerencia de I+D, decidimos unirnos al proyecto MMEDUSA y apoyar con nuestro trabajo de grado esta investigación.

Es así, como el presente trabajo está enmarcado en el proyecto MMEDUSA y respaldado por el esfuerzo conjunto del grupo de ingeniería de software de la universidad EAFIT y AVANSOFT S.A, los cuales nos han brindado los espacios de debate y discusión de los cuales han surgido las ideas y fundamentos del presente trabajo.

El trabajo que se presenta a continuación busca analizar el impacto del paradigma orientado a aspectos en las fases tempranas del desarrollo de software, concretamente en ingeniería de requisitos. Para alcanzar este objetivo, el trabajo se encuentra organizado en cuatro secciones: la primera sección presenta los elementos más importantes que componen el paradigma aspectual, su historia y fundamentación.

En la segunda sección se realiza el análisis del modelo de ingeniería de requisitos bajo el enfoque aspectual AORE propuesto por la Universidad Nova de Lisboa. Se presentan los elementos y consideraciones más importantes del modelo y se realiza una propuesta de ampliación del modelo a partir de los hallazgos encontrados en el análisis de la propuesta y la aplicación de esta a un caso de estudio. La propuesta de expansión del modelo AORE, lleva a la exploración del modelo COSMOS, el cual es abordado y analizado a lo largo de este mismo capitulo, Adicionalmente, son presentadas otras propuestas para el manejo de aspectos en etapas tempranas, pero no se profundiza en estas.

La tercera sección, presenta una propuesta de aplicación del modelo AORE en la herramienta Enterprise Architect como alternativa para facilitar su aplicación. Se presentan las características de la herramienta y la forma como esta puede ser empleada para soportar el modelo.

Finalmente, la cuarta sección, presenta la aplicación del modelo a un caso de nivel industrial, el cual consiste en un sistema desarrollado por AVANSOFT S.A. para la gestión de las competencias del recurso humano.

Esperamos que el trabajo realizado y presentado a continuación, sea de gran utilidad tanto para las organizaciones educativas como para las empresas de

software del medio, permitiendo adoptar de una manera fácil y ágil este nuevo enfoque.

1. FUNDAMENTOS DEL PARADIGMA ORIENTADO A ASPECTOS

La ingeniería de software cuenta hoy día con uno de los paradigmas más aceptados y efectivos para dar solución a los problemas a los cuales se enfrenta. El paradigma orientado a objetos, se ha perfilado entre los estudiosos del software, como la estrategia más adecuada para abordar problemas de gran complejidad.

La existencia de técnicas y herramientas que apoyan los procesos de elicitación de requisitos, análisis, diseño y construcción bajo el enfoque orientado a objetos ha permitido la evolución de los procesos de desarrollo de software, propiciando la aparición de arquitecturas escalables y robustas que han logrado llevar todo el proceso de desarrollo de software a niveles de modularidad y descomposición funcional que facilitan significativamente la evolución y crecimiento de los sistemas. Si bien este enfoque parece solucionar eficientemente los problemas que se ha propuesto la ingeniería de software, la realidad es que el paradigma orientado a objetos da solución adecuada al comportamiento funcional, pero la solución ofrecida para el comportamiento no funcional no es la más apropiada, como se evidenciará en otros apartados de este trabajo (ver sección 1.3). De igual manera, el paradigma objetual permite realizar una separación adecuada del comportamiento no transversal, pero el comportamiento transversal no es tratado de manera apropiada.

Los comportamientos funcionales hacen referencia a lo que conocemos como la lógica del negocio, lo que el sistema debe hacer. Los comportamientos no funcionales, o ***atributos de calidad*** como serán llamados en adelante, hacen referencia a aquellas características con las que el software –resultado final del proceso- debe contar, pero que en sí, no afectan su comportamiento. Dentro de estas características podemos encontrar el manejo de errores, la seguridad, la sincronización, la gestión de memoria, la distribución de componentes, restricciones de tiempo real y otros.

La forma como la programación orientada a objetos, ha dado solución a tales necesidades no funcionales, ha generado que los sistemas tengan un fuerte acoplamiento de dichas características con el comportamiento que es propio del problema que se está solucionando ya que estas se encuentran dispersas por todo el sistema.

La propuesta de orientación *a aspectos* se ha posicionado como un enfoque en la ingeniería de software que puede mejorar significativamente el diseño y construcción de grandes sistemas de software. La orientación a aspectos se presenta entonces, como un modelo que soporta la separación de los comportamientos que representan la funcionalidad base de aquellos comportamientos que pueden ser transversales a mas de un módulo; es decir, intenta separar los componentes y los aspectos unos de otros, facilitando el desarrollo de los sistemas.

La orientación a aspectos como propuesta a nivel de lenguaje de programación se encuentra soportada [1], [44] y es aceptada por la comunidad de desarrolladores, existen investigaciones y trabajos en curso encaminadas a proponer metodologías que soporten y estandaricen todo el proceso de desarrollo de software, abarcando las correspondientes fases: elicitación de requisitos, análisis, diseño y arquitectura e implementación.

1.1 La evolución de la ingeniería de software

En [20] se exponen las generaciones por las que ha atravesado la ingeniería de software a través de su historia:

1ª Generación: *Código spaghetti*.: Esta generación se caracterizó por que no existía una clara separación entre datos y funcionalidad. Como consecuencias de esta falta de separación se tiene un código resultante es complejo y una alta dificultad para realizar cambios

2ª y *3ª* Generación: Descomposición funcional: En estas generaciones se hace visible una clara división entre los datos y la funcionalidad, identificando las partes más manejables como funciones que se definen en el dominio del problema. La facilidad de integración de nuevas funciones, el hecho que las funciones queden algunas veces poco claras debido a la utilización de datos compartidos y que los datos quedan esparcidos por todo el código, son algunas de las características que se observan en esta generación.

4ª Generación: Orientación a objetos: En esta generación aparece el modelo de objetos, el cual utiliza el principio de descomposición funcional, ajustándose mejor al dominio del problema. Algunas de las características que se observan en esta generación son:

- Provee una fácil la integración de nuevos datos.
- Las funciones quedan esparcidas por todo el código.
- Con frecuencia, para realizar la integración de nuevas funciones hay que modificar varios objetos.
- Enmarañamiento de los objetos en funciones de alto nivel que involucran a varias clases.

5ª Generación: ¿Orientación a aspectos?: Separa los componentes funcionales y los atributos de calidad unos de otros, proporcionando mecanismos que hagan posible abstraerlos y componerlos para formar todo el sistema.

El paradigma orientado a aspectos es una nueva propuesta de desarrollo de software que pretende mejorar la separación entre los componentes que conforman un sistema, brindando nuevos mecanismos que permitan aumentar los niveles de abstracción, un mayor entendimiento del sistema y sus componentes, al tiempo que aumenta la eficiencia y facilita su evolución.

En lo que sigue de este capítulo conoceremos cuáles han sido los hechos que han motivado el surgimiento de este nuevo paradigma, sus principales características y la forma como opera.

1.2 Antecedentes del Desarrollo Basado en Aspectos

El enfoque basado en aspectos, al igual que el enfoque Orientado a Objetos, surge primero como propuesta a nivel de lenguaje de programación. Tradicionalmente, las aproximaciones de análisis y diseño para los paradigmas de la ingeniería de software han surgido posteriormente al esfuerzo de muchas personas que han explorado ideas a nivel de lenguajes de programación [1]. Es complicado establecer claramente donde empieza la historia de la orientación a aspectos ya que han sido numerosos los trabajos que se han realizado y han emergido a partir de la Aspect Oriented Programming (AOP). Lo que ha sido común a todos ellos ha sido la continua búsqueda de lo que conoceremos como *separation of concerns*.

El principio de *separation of concerns*, que fue identificado desde la década de 1970, plantea que un problema dado involucra varios intereses (*concerns)* que deben ser identificados y separados. *Separation of concerns* ayuda a disminuir la complejidad a la hora de tratarlos y se puede cumplir con requerimientos

relacionados con la calidad como adaptabilidad, mantenibilidad, extensibilidad y reusabilidad [2].

Tal como afirma ALFEREZ [4], "...los progresos más significativos en la historia de la Ingeniería de Software se han obtenido gracias a la aplicación de la descomposición de un sistema complejo en partes que sean más fáciles de manejar, conocido como *descomposición funcional*, que pone en práctica el principio de "divide y vencerás" al identificar las partes más manejables como funciones que se definen en el dominio del problema. La ventaja principal que aporta esta descomposición es la facilidad de integración de nuevas funciones, aunque también tiene grandes inconvenientes, como es el hecho de que estas quedan algunas veces poco claras debido a la utilización de datos compartidos y esparcidos por todo el código, con lo cual, normalmente el integrar un nuevo tipo de dato implica que se tengan que modificar varias funciones." La aparición del paradigma Orientado a Objetos parecía en principio ser una solución efectiva para el desarrollo de software basado en el principio de descomposición funcional, este se ve afectado por los fenómenos de *scattering* y *tangling* los cuales abordaremos mas adelante.

1.2.1 Modelos de programación aspectual

Es importante tener en cuenta que muchos de los trabajos e investigaciones desarrollados se dieron de forma independientemente, todos motivados por la necesidad de encontrar nuevos mecanismos que dieran solución a inconvenientes presentados con los paradigmas tradicionales y que finalmente convergieron en lo que ahora es el paradigma aspectual. Un resumen de los trabajos más importantes se presenta a continuación:

Programación basada en filtros: En 1990, en la Universidad de Twente en Holanda, un equipo bajo el mando de Mehmet Aksit trabajó en *Composición de*

Filtros. La composición de filtros busca modularizar el comportamiento a través de *filtros,* los cuales pueden ser utilizados para capturar y mejorar la ejecución y el comportamiento de los objetos. El concepto principal detrás de los filtros es el *aumento* de los objetos convencionales a través de la manipulación de todos los mensajes enviados y recibidos [9], permitiendo así extender el comportamiento de un sistema basado en objetos incrementando el conjunto de comportamientos visibles. Cada filtro especifica una inspección y manipulación particular de mensajes. Los filtros de entrada y de salida pueden manipular mensajes que son respectivamente enviados y recibidos por un objeto.

Programación adaptativa: Paralelamente, en la Universidad de Northeastern, en Estados Unidos Kart Lieberher definió el *Método Demeter*, el cual proveía una abstracción de la estructura de las clases y su navegación para soportar de una forma más adecuada el conocimiento del comportamiento de las operaciones [1]. *En su esencia es el "principio de menos conocimiento" con respecto a las instancias de los objetos que se utilizan en un método [3].*

El concepto principal detrás del trabajo del grupo Demeter es la programación adaptativa (PA), la cual puede verse como una instancia temprana de la POA. El término programación adaptativa se introdujo recién en 1991. Basada en el uso de autómatas finitos y una teoría formal de lenguaje para expresar concisamente y procesar eficientemente conjuntos de caminos en un grafo arquitectónico.

La definición formal de PA puede considerarse como una definición inicial y general de la POA: en PA los programas se descomponen en varios bloques constructores de corte (crosscutting building blocks). Inicialmente separaron la representación de los objetos como un bloque constructor por separado. Luego agregaron comportamiento de estructura (structure-shy behavior) y estructuras de clase como bloques constructores de corte.

Al crecer la POA como paradigma fue posible definir la PA como un caso especial de la POA, esto es, la PA es igual a la POA con grafos y estrategias transversales. Las estrategias transversales se consideran expresiones regulares que especifican un recorrido en un grafo [6].

Programación orientada al sujeto: En 1993, un equipo de IBM T.J Research Watson Center, liderado por William Harrixon y Harold Ossher publicaron un trabajo sobre "*programación orientada al sujeto*" (Subject-oriented programming). La programación orientada al sujeto presentaba una descomposición y composición de módulos de software basados en diferentes *dimensiones de concerns [1].*

El principal objetivo de la programación orientada al sujeto es facilitar el desarrollo y evolución de suites de aplicaciones cooperativas. Las aplicaciones cooperan compartiendo objetos y contribuyendo con la ejecución de operaciones [10]

En la programación orientada al sujeto, las unidades son clases, métodos y variables de instancia. Los sistemas son construidos como composiciones de sujetos, cada uno de los cuales es una jerarquía de clases que modela un dominio desde un punto de vista particular [9].

Aspectos como extensión al modelo de programación orientada a objetos: Los trabajos mas conocidos fueron los realizados en Xerox PARC en 1997 los cuales dieron como primer desarrollo el popularizado Aspecto. Liderados por Gregor Kiczales, el equipo trabajó en protocolos de metaobjects y reflection, con ideas que evolucionaron posteriormente a la modularización de crusscutting concerns. A partir de las investigaciones desarrolladas en los laboratorios del Xerox PARC Research Center de Palo Alto se generó una herramienta para la programación de aspectos en Java, AspectJ.

Si bien los anteriores no han sido los únicos trabajos y aproximaciones que han encaminado la evolución de la orientación a aspectos, sí han sido los más sobresalientes y popularizados por lo que bien podemos considerarlos como los orígenes del Desarrollo de Software Orientado a Aspectos. (AOSD).

1.2.2 El paradigma Orientado a Objetos no es suficiente

La programación orientada a aspectos surgió como una metodología basada en el principio de descomposición funcional y con la cual se ha logrado de forma exitosa la construcción de sistemas complejos gracias al uso del principio de descomposición funcional y la facilidad para integrar nuevos datos y funcionalidades [4]. Esto ha motivado a que hoy día, el paradigma objetual cuente con gran acogida en la comunidad de desarrollo de software. Dicho esto, se hace inevitable preguntarnos por que dicho paradigma no es suficiente para enfrentar los nuevos retos de la ingeniería de software. En [2] se plantean dos interesantes preguntas que pueden ayudarnos a ver la necesidad de un nuevo paradigma: "*¿Podemos ubicar siempre lo que una funcionalidad debe hacer en un sólo lugar? ¿Alguna vez se ha notado que existen algunas partes de procesamiento que no encajan bien en una clase particular y que de hecho se tiene la sensación de que dicho procesamiento no puede encapsularse en una sóla clase?"* Las debilidades del paradigma orientado a objetos empezaron ha hacerse evidentes desde 1978 como se evidencia en [12] y han estado presentes a lo largo del ciclo de vida del software.

1.2.2.1 Hechos y hallazgos

Algunos de los hechos y hallazgos que demuestran la necesidad de un nuevo paradigma se presentan a continuación.

- El paradigma orientado a objetos está basado sobre los conceptos de herencia, encapsulamiento y polimorfismo. Estas características y/o mecanismos del paradigma son abordadas en [5] donde se establece el encapsulamiento como un mecanismo que permite soportar el estado y comportamiento de un concern, mientras que la herencia y la agregación son vistos como mecanismos que soportan la composición de concerns. La programación orientada a objetos funciona bien si el problema puede ser descrito en términos de interfaces simples por medio de objetos, sin embargo en sistemas complejos, tales como sistemas concurrentes y sistemas distribuidos, al aplicar la orientación a objetos, se hace más difícil la reutilización, aumenta la complejidad y se complica la validación del sistema [5]. Estos problemas se presentan porque las interacciones entre los componentes del sistema están basadas en un conjunto de propiedades que no pueden ser localizadas en una sola unidad modular pero que tienden a estar esparcidas a través de un conjunto de componentes funcionales (tangling). Estas propiedades son dependientes del dominio del sistema y pueden ser requisitos funcionales o no funcionales.

- El tangling y su consecuente alto acoplamiento entre los componentes funcionales de un sistema acaban con la modularidad del código, limitando su reutilización y hace los programas más propensos a errores. El tangling acaba con la calidad del software, y los beneficios asociados con la programación orientada a objetos no pueden ser utilizados completamente [5].

- Los casos de uso han sido universalmente adoptados para la especificación de requisitos. Los casos de uso comienzan en requisitos, son traducidos a colaboraciones en análisis y diseño, y a casos de prueba. La codificación de un componente o una clase requiere de nosotros para mezclar el código derivado de varios casos de uso, los mecanismos de extensión soportados cuando se trabaja con casos de uso, con el UML actual no es soportado entre los elementos de análisis y diseño tampoco en colaboraciones, componentes, clases ni en la implementación en ambientes como JAVA o C#. El principal problema es la

limitación de los lenguajes usados actualmente. La programación orientada a aspectos es el enlace faltante, esto permitirá cortar un sistema claramente, caso de uso por caso de uso sobre muchos modelos y los casos de uso permanecen separados, logrando así, la separación de intereses [12].

- En [11] se argumenta como los métodos de diseño de la orientación a objetos producen artefactos que se encuentran desalineados entre el código resultante y los requerimientos del sistema. Esta falta de alineamiento se debe a que las metodologías de requisitos, análisis y diseño usadas en este paradigma, hacen especial énfasis en el dominio del usuario, omitiendo otras características del sistema tales como sincronización, persistencia, manejo de fallos, entre otras, las cuales no están debidamente soportadas por la orientación a objetos. Este desalineamiento a través del ciclo de vida del software lleva a la generación de diseños de requerimientos individuales a través de múltiples clases (scattering) y clases individuales que contienen implementaciones de muchos requisitos (tangling). Estos fenómenos acaban con la trazabilidad del sistema, terminando en grandes problemas e impidiendo analizar el impacto de los cambios y evolución del mismo, al tiempo que se aumenta la complejidad y la falta de comprensión de los diversos artefactos de diseño y código.

- En [17] se argumenta la incapacidad del paradigma orientado a objetos para lograr completamente la separación de concerns ya que este paradigma provee débiles mecanismos de composición y descomposición, permitiendo sólo una dimensión en la cual los concerns pueden ser separados. Esto es llamado “La tiranía de la descomposición dominante.” Las aproximaciones que sufren de esta tiranía, proveen una base de descomposición, siendo ésta comúnmente los requisitos funcionales, a partir de la cual se realiza la separación de intereses, lo cual limita la capacidad para descomponer sistemas complejos.

1.2.2.2 El problema: tangling y scattering

Si tratamos de resumir los principales inconvenientes que nos presenta el paradigma orientado a objetos, encontraremos los fenómenos de *tangling* y *scattering* como factor común. Estos fenómenos se presentan por la incapacidad del paradigma orientado a objetos para soportar la separación de intereses, no sólo en las etapas finales, sino desde las etapas tempranas del ciclo de vida del software.

Las definiciones que consideramos adecuadas para los términos de *tangling* y *scattering* se presentan más adelante en este capítulo, pero para los intereses que nos atañen en este momento podemos entender el *scattering* como la necesidad de *diseminar* un conjunto de requisitos a través de muchos componentes del sistema. El *tangling*, por el contrario, consiste en la necesidad de hacer que un sólo componente del sistema tenga que *realizar* todo un conjunto de requisitos. Estos dos fenómenos, que se hacen presentes en el paradigma orientado a objetos, dificultan la reutilización, aumentan el acoplamiento entre componentes al tiempo que aumentan la complejidad del sistema y hacen que el software pierda capacidad para evolucionar.

En pocas palabras, *tangling* y *scattering,* atentan contra los atributos de calidad a los que apunta la ingeniería de software: aumentar la calidad del software, reducir sus costos y facilitar su mantenimiento y evolución [7].

Si bien se ha identificado al *tangling* y al *scattering* como principales fallas del paradigma orientado a objetos, se hace necesario identificar cuándo aparecen estos problemas.

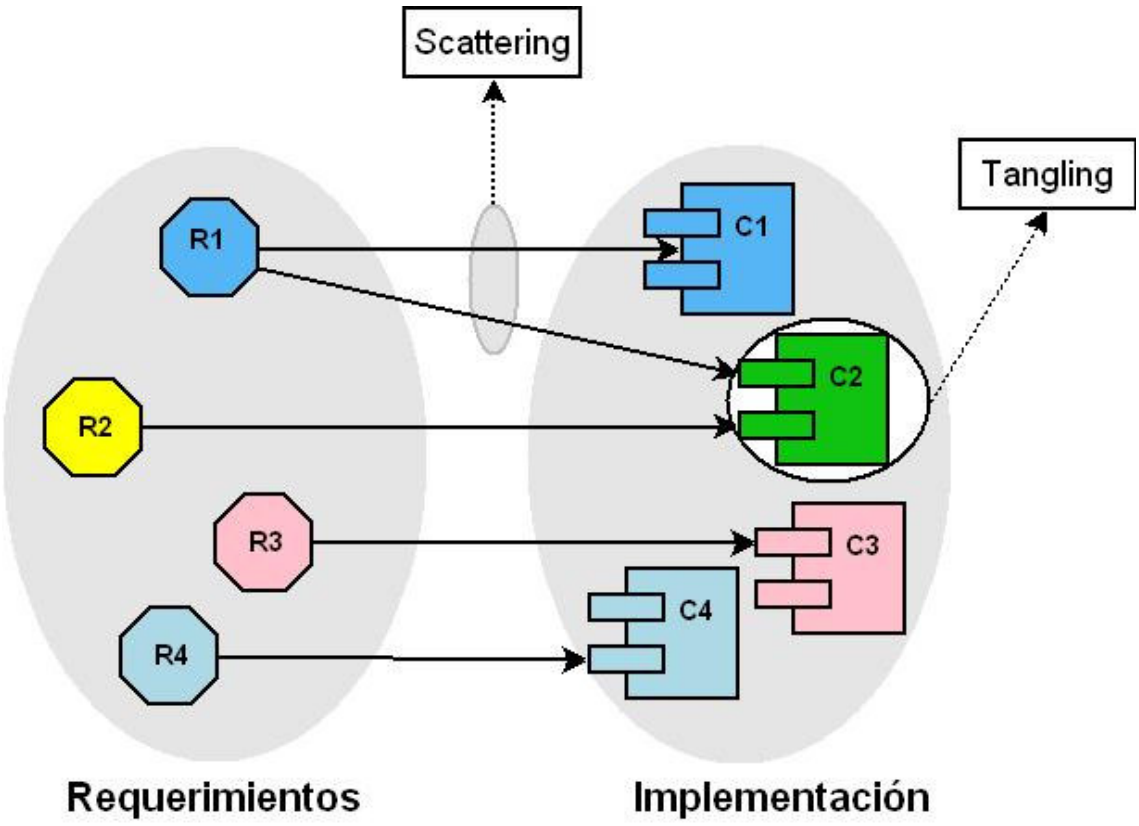

Figura 1. Transición entre requerimientos y diseño / implementación. [19]

La figura 1 muestra los problemas que se presentan al tratar de mapear el dominio del problema con el espacio de la solución ya que no siempre es posible tener un mapeo uno a uno. Algunos requerimientos son llevados a codificaciones dispersas y/o enredadas por todo el sistema. En la figura podemos observar como el requisito R1 tiene que ser satisfecho por los componentes C1 y C2, dando así lugar al *scattering*. También es posible observar como el componente C2 se encuentra satisfaciendo los requisitos R1 y R2, dando lugar al *tangling.*

1.3 Así trabaja el paradigma orientado a aspectos

Hasta el momento se ha dado un vistazo histórico de lo que ha sido la evolución del paradigma orientado a aspectos y las falencias de la orientación a objetos que

han motivado el surgimiento de este. En esta sección se presentan los elementos que permiten entender cómo funciona el paradigma aspectual y cuáles son sus componentes y principios fundamentales.

Como hemos mencionado a lo largo de este capítulo, el paradigma orientado a objetos ofrece una solución eficiente cuando se hace referencia al comportamiento funcional del sistema. Este es un hecho que el paradigma aspectual no ignora, por el contrario, hace uso de la orientación a objetos.

El paradigma orientado a aspectos sigue al paradigma de la orientación a objetos, y como tal, soporta la descomposición orientada a objetos. Pero, a pesar de esto, el paradigma aspectual no se puede considerar como una extensión de la orientación a objetos, ya que puede utilizarse con los diferentes estilos de programación [20].

A continuación se presentan los elementos que conforman el paradigma orientado a aspectos y la forma como éste opera.

1.3.1 La base del paradigma: Clases y Aspectos

Un sistema orientado a aspectos puede verse como una combinación de la *funcionalidad básica y el comportamiento aspectual,* los cuales pueden ser combinados por medio de *puntos de enlace.* Estos conceptos son tratados a continuación.

1.3.1.1 Funcionalidad básica

El paradigma orientado a aspectos utiliza toda la potencia de la orientación a objetos para soportar la funcionalidad base del sistema. Por medio de objetos se implementa la funcionalidad principal del sistema, tal como puede ser la gestión de

inventarios, el pago de una nomina, entre otros. La funcionalidad básica está determinada por el dominio del problema.

1.3.1.2 Comportamiento aspectual

El enfoque original con el cual surgió la orientación a aspectos identificaba el comportamiento aspectual sólo con conceptos técnicos tales como la persistencia, la gestión de errores, la sincronización o la comunicación de procesos [20]. Hoy, el comportamiento aspectual es llevado más allá de las características técnicas del sistema y empiezan a identificarse conceptos del dominio del problema que tiene comportamiento aspectual y que *cruzan el sistema.*

Los lenguajes orientados a aspectos definen una nueva unidad de programación de software para encapsular las funcionalidades que cruzan todo el código [20].

Los lenguajes orientados a aspectos pueden ser de propósito general o de dominio especifico:

Lenguajes de dominio específico: Soportan uno o más de los aspectos considerados (distribución, manejo de errores, etc.), pero no pueden soportar otros aspectos distintos de aquellos para los que fueron diseñados. Los lenguajes de aspectos de dominio específico normalmente tienen un nivel de abstracción mayor que el lenguaje base y, por tanto, expresan los conceptos del dominio específico del aspecto en un nivel de representación más alto.

Estos lenguajes normalmente imponen restricciones en la utilización del lenguaje base. Esto se hace para garantizar que los conceptos del dominio del aspecto se programen utilizando el lenguaje diseñado para este fin y evitar así interferencias entre ambos. Como ejemplo de este tipo de lenguaje de aspectos se puede nombrar a COOL (COOrdination Language), de Xerox, un lenguaje para

sincronización de hilos concurrentes. En COOL, el lenguaje base es una restricción de Java, ya que se han eliminado los métodos wait, notify, y notifyAll, y la palabra reservada synchronized para evitar que el lenguaje base sea capaz de expresar sincronización. Se obtiene un mayor nivel de abstracción que en Java al especificar la sincronización de hilos de manera declarativa. Un segundo ejemplo es RIDL(Remote Interaction and Data transfers Language), para el aspecto de distribución: invocación remota y transferencia de datos [45].

Lenguajes de propósito general: se diseñaron para ser utilizados con cualquier clase de aspecto, no solamente con aspectos específicos. Por lo tanto, no pueden imponer restricciones en el lenguaje base. Principalmente soportan la definición separada de los aspectos proporcionando unidades de aspectos. Normalmente tienen el mismo nivel de abstracción que el lenguaje base y también el mismo conjunto de instrucciones, ya que debería ser posible expresar cualquier código en las unidades de aspectos. Un ejemplo de este tipo de lenguajes es AspectJ, que utiliza Java como base, y las instrucciones de los aspectos también se escriben en Java [4].

En [20] se encuentra una comparación entre los lenguajes de aspectos de propósito general y los de dominio específico y se presentan algunos ejemplos de estos lenguajes.

1.3.1.3 Puntos de enlace

Para poder combinar la *funcionalidad base* con el *comportamiento aspectual* se requiere determinar algunos puntos comunes, que son los que se conocen como *puntos de enlace (join points)*.

Los puntos de enlace son una clase especial de interfaz entre los aspectos y los módulos del lenguaje de componentes. Son los lugares del código en los que éste

se puede aumentar con comportamientos adicionales. Estos comportamientos se especifican en los aspectos.

1.3.2 Tejedor (weaver)

La interacción entre clases y aspectos se hace posible a través de un tercer componente, el cual conocemos como *tejedor.* El tejedor es el encargado de realizar la mezcla de la *funcionalidad base* con el *comportamiento aspectual*. Las clases y los aspectos se pueden mezclar de dos formas distintas: de manera estática y de manera dinámica.

1.3.2.1 Tejido estático

Implica modificar el código fuente de una clase insertando sentencias en estos puntos de enlace. Es decir, que el código del aspecto se introduce en el de la clase. En [22] se destacan dos características importantes del tejido estático, se evita un impacto negativo en el rendimiento de las aplicaciones, pero se hace difícil identificar los aspectos en el código una vez ya se ha tejido.

1.3.2.2 Tejido dinámico

El entrelazado dinámico requiere que los aspectos existan y estén presentes de forma explícita tanto en tiempo de compilación como en tiempo de ejecución. A partir de una interfaz de reflexión, el tejedor es capaz de añadir, adaptar y borrar aspectos de forma dinámica, si así se desea, durante la ejecución.

Ambos esquemas de tejido presentan ventajas y desventajas como se detalla en [20] y [6].

1.3.3 Aplicaciones bajo el paradigma orientado a aspectos

Como vemos, para construir un sistema orientado a aspectos tenemos tres elementos principales:

- Clases, las cuales modelan la funcionalidad base
- Aspectos
- Tejedor

En el esquema tradicional, la construcción de la aplicación se realiza a través de un compilador o intérprete, el cual simplemente toma el código de la funcionalidad base y lo traduce a un lenguaje entendible por la maquina. La construcción de aplicaciones bajo este esquema se muestra en la figura 2.

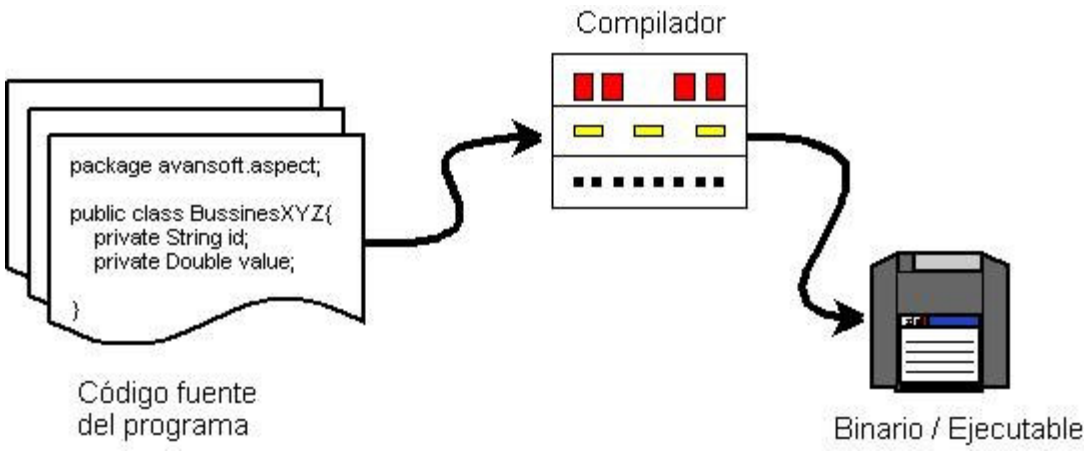

Figura 2. Construcción de aplicaciones bajo el esquema tradicional. [20]

La construcción de una aplicación bajo el enfoque orientado a aspectos requiere la combinación del código de la funcionalidad base con los distintos módulos que implementan los aspectos. Esta combinación es realizada por el *tejedor.* Cada aspecto codificado con un lenguaje distinto. La construcción de aplicaciones bajo este esquema se muestra en la figura 3.

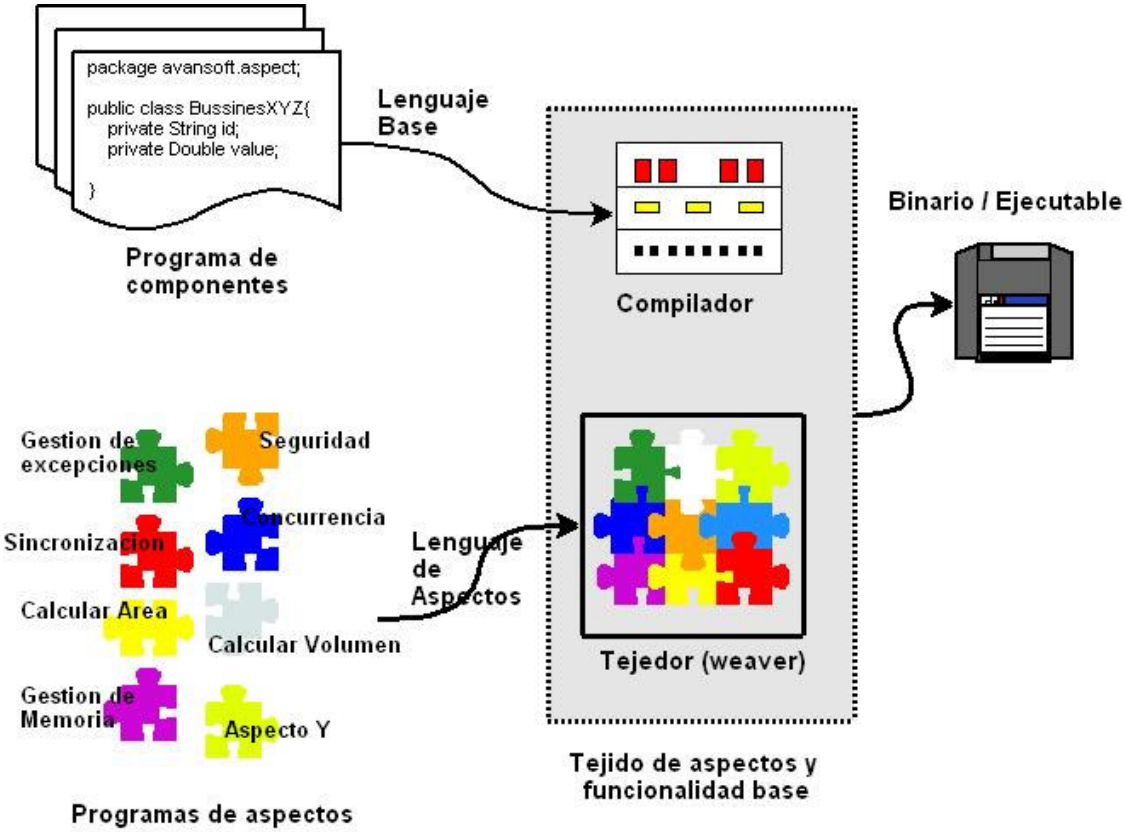

Figura 3. Construcción de aplicaciones bajo el paradigma orientado a aspectos. [20]

El paradigma aspectual nos permite ver el sistema como un conjunto conformado por un modelo de objetos y varios modelos de aspectos. El modelo de objetos es el encargado de *implementar* la funcionalidad base del sistema, mientras que el modelo de aspectos nos permite implementar las características no funcionales (distribución, manejo de errores, sincronización entre otras) y según los enfoques más recientes [14], también nos permite implementar *aspectos del dominio del problema.* La figura 4 presenta la estructura de un programa orientado a aspectos.

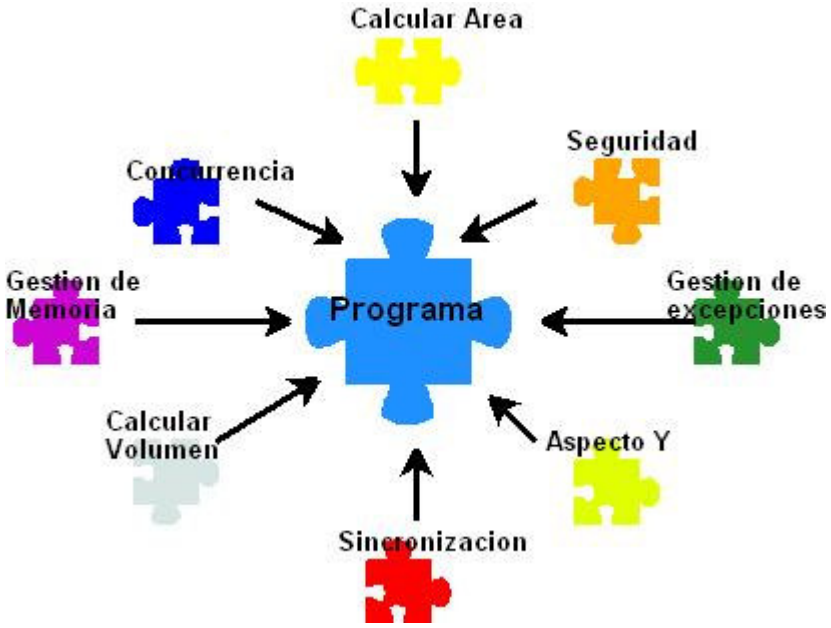

Figura 4. Estructura de un programa orientado a aspectos. [20]

1.4 Aspectos en etapas tempranas

Todos los sistemas, sin importar el paradigma bajo el cual se encuentren necesitan estar bien diseñados, aplicando *buenas prácticas de ingeniería de software.* El análisis y el diseño de un sistema son tan importantes como la misma implementación, tanto así que estas fases son consideradas como las que más contribuyen al éxito de un proyecto.

Considerar las etapas tempranas del ciclo de vida del software permite tener clara la estructura final del sistema, de igual forma que reduce el esfuerzo de los programadores ya que desde el principio se tienen definidos los caminos y procesos que se deben seguir.

La necesidad de brindar un adecuado soporte a través de todo el ciclo de vida del software puede verse claramente en [13], donde se establece una relación entre

aspectos tempranos, intermedios y finales, dando lugar a una propuesta de desarrollo en la que los aspectos son elementos relevantes a lo largo del proceso de desarrollo y que se conoce como AOSD (Aspect Oriented Software Development). La figura 5 presenta la relación entre estos aspectos.

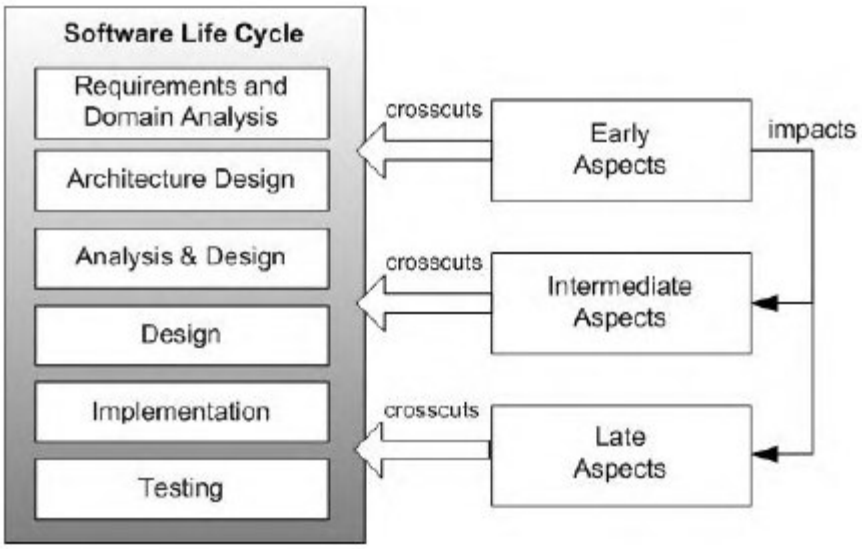

Figura 5. Relación entre aspectos tempranos, intermedios y finales. [13]

Igualmente, en [13] se establece el impacto que tienen los aspectos tempranos en el ciclo de desarrollo de software.

En [23] se presentan las potenciase aplicaciones del modelado de concerns en etapas tempranas. Estos pueden conllevar a un mejor entendimiento y un mejor tratamiento de los concerns desde las primeras etapas del desarrollo y además pueden ayudar a identificar elementos reutilizables durante el análisis de requisitos. Esto facilita la formulación de vistas alternativas y representaciones de requisitos.

1.5 Definiciones básicas en AOSD

Como ya hemos visto, el surgimiento del paradigma orientado a aspectos ha estado rodeado de un sin número de esfuerzos encaminados a aplicar de manera apropiada el principio de separación de intereses. Esta diversidad de trabajos ha ocasionado que en muchos casos las definiciones puedan presentarse ambiguas, o como en el caso de los concerns, sean tan abundantes que en principio sea un término que genera confusión. A continuación presentamos las definiciones que encontramos más acordes con nuestros objetivos de trabajo y que desde nuestro punto de vista ofrecen una mayor claridad**.**

1.5.1 Concern (Interés)

Este es uno de los términos mas documentados en el paradigma orientado a aspectos, sin embargo, ¿cual es la definición de concern?. En [13] se establece una discusión a cerca de cuál es la definición más acertada para este término, surgiendo así las siguientes preguntas:

¿Cual es la definición de concern?

- ¿Es un requisito?
- ¿Es un concern algo que los stakeholders consideran que es importante?
- ¿Es un concern algo que los desarrolladores deben considerar para que el sistema sea exitoso?
- ¿Es un concern una expectativa?
- ¿Es cualquier cosa?

Finalmente se llega a un acuerdo sobre algunas posibles definiciones [13]:

- Es una propiedad observable deseada.
- Es una característica.
- Es alguna responsabilidad.
- Es un sub-problema.

Otras definiciones y/o características que consideramos adecuadas:

- Es una propiedad de interés para algún stakeholder [6].

- Una colección coherente de requisitos [14].
- Un aspecto de un problema que requiere la atención de los desarrolladores [15].
- Concerns son intereses que pertenecen al desarrollo del sistema, operaciones o cualquier otro aspecto crítico o importante para uno o más de los stakeholders. Los concerns incluyen consideraciones del sistema tales como desempeño, disponibilidad, seguridad, distribución y capacidad de evolución [16].
- Algunos concerns son localizados en un lugar particular en un sistema emergente, algunos se refieren a una propiedad medible de un sistema como un todo, otros son estéticos y otros son comportamientos sistemáticos [9].

1.5.2 Separation of concerns (Separación de interese)

El principio de "*separation of concerns"* hace referencia a la comprensión de los conceptos del sistema como unidades de software independientes. Este principio trae asociados múltiples beneficios para el desarrollo de software, incluyendo un mejor análisis, mejor entendimiento del sistema, mayor adaptabilidad, un alto grado de reusabilidad y mantenibilidad.

En [5] se exponen los beneficios que el principio de separación de intereses presenta en las fases de análisis y diseño para el desarrollo de software, al tiempo que se establece que los criterios de descomposición son establecidos por el paradigma dominante.

El objetivo de la separación de intereses es dar representaciones independientes a intereses independientes. La efectiva separación de intereses obtiene ventajas a través del ciclo de vida del software, pero es especialmente importante extender la vida útil de un sistema con respecto a evolución y otros pequeños cambios.

Los intereses que tienen representaciones independientes pueden ser cambiados de manera independiente, simplificando los cambios y minimizando los costos, esfuerzo, e impacto.

Para el proceso software la separación de intereses puede dar flexibilidad en términos de asignación y programación de tareas. Adicionalmente, puede proveer una base para gestión de la configuración y la familia de productos de gestión. Separación de intereses es una técnica clave para la ingeniería de software para la creación de componentes de software evolutivos. [18]

1.5.3 Crosscutting concern (Interés de corte transversal)

Los intereses de corte transversal son aquellas propiedades o conceptos de un sistema que tienden a estar presentes en varios componentes funcionales [34]. La distribución y sincronización son ejemplos típicos de estos concerns, sin embargo existen concerns de naturaleza funcional que también son de corte transversal.

En [16] se hace referencia a los intereses de corte transversal como aspectos críticos del problema de composición de software. Se destacan los intereses de corte transversal como la principal causa de muchas complicaciones en la arquitectura de software; complicaciones como carencia de abstracción para la separación y combinación de intereses de varios tipos en especificaciones de arquitectura.

En [19] se destacan las implicaciones de los intereses de corte transversal al ser tratados desde los paradigmas no aspectuales:

- Bajos niveles de trazabilidad entre los artefactos de la fase de requisitos y la fase de implementación.
- Baja productividad en el proceso de desarrollo de software.
- Bajo grado de reutilización de código fuente.
- Bajos niveles de adaptabilidad y evolución de los sistemas.
- Baja calidad en el software.

1.5.4 Aspect (Aspecto)

Como hemos visto, la historia del paradigma aspectual ha estado influenciada por diversas corrientes y grupos de investigación, lo que conlleva a que podamos encontrar diversas definiciones para el concepto de aspecto.

La definición de aspecto dada por el grupo Demeter en 1995 fue:
“Un aspecto es una unidad que se define en términos de información parcial de otras unidades” [7].

Gregor Kiczales define un aspecto como:
“Un aspecto es una unidad modular que se dispersa por la estructura de otras unidades funcionales. Los aspectos existen tanto en la etapa de diseño como en la de implementación. Un aspecto de diseño es una unidad modular del diseño que se entremezcla en la estructura de otras partes del diseño. Un aspecto de programa o de código es una unidad modular del programa que aparece en otras unidades modulares del programa” [7].

Un aspecto es una unidad modular diseñada para implementar un concern. La definición de un aspecto puede contener algún código y las instrucciones dónde cuándo y cómo ser invocado [9].

2. METODOLOGÍAS DE MANEJO DE REQUISITOS BAJO EL ENFOQUE ORIENTADO A ASPECTOS.

Las técnicas de la ingeniería de software orientada a aspectos –AOSD- proveen mecanismos para la identificación, modularización, representación y composición de intereses de corte transversal, tales como la seguridad, movilidad y restricciones de tiempo real. Las iniciativas del tratamiento de aspectos tempranos están enfocadas en el manejo de las propiedades de corte transversal en las etapas tempranas de la ingeniería de requisitos y el diseño de la arquitectura [36].

El tratamiento temprano de los concerns que componen un sistema se ha convertido en uno de los campos que ha generado mayor interés en los últimos años. En [36] se manifiestan las razones por las cuales el tratamiento de los aspectos debe ser llevado a cabo desde las primeras etapas del ciclo de vida del software.

Dadas las diversas corrientes que han contribuido al surgimiento del paradigma orientado a aspectos, se hace natural que esta misma diversidad se haga presente al momento de desarrollar metodologías que pretendan dotar al paradigma aspectual de los mecanismos necesarios para realizar el tratamiento temprano de los aspectos. La primera parte de este capítulo nos permitirá

conocer las metodologías mas relevantes para el manejo temprano de aspectos, posteriormente se profundizará en el modelo AORE como metodología de referencia para el manejo de aspectos a nivel de requisitos y su aplicación en un caso real será planteada en los siguientes capítulos.

2.1 Algunas aproximaciones al manejo de requisitos aspectuales

Basados en [25], en esta sección pretendemos resaltar los principales trabajos que se están realizando en el mundo asociados al tema de Aspect Oriented Software Development específicamente a la identificación y modelado de aspectos en etapas tempranas. Podremos ver los diferentes trabajos, analizarlos, compararlos y ver como se traslapan y/o como se complementan entre ellos.

En [25] se propone la utilización de un framework que permita realizar el estudio de cada una de las aproximaciones seleccionadas, el cual se basa en los siguientes criterios:

a) Fase

¿Qué fase del ciclo de vida es considerada por la metodología? Por ejemplo: análisis de requisitos, análisis del dominio y/o arquitectura.

b) Objetivo

¿Cuál es el objetivo de la metodología? Ejemplos son: identificación, especificación y/o evaluación de aspectos.

c) Artefactos

¿Cuálles son los artefactos adoptados que son usados para la identificación y/o especificación de aspectos tempranos?

d) Reglas heurísticas y procesos

¿La metodología usa reglas heurísticas explícitamente definidas o procesos para identificar y especificar aspectos tempranos?

e) Modelo de aplicación

¿Qué tipos de modelos/artefactos de aplicación es entregado?

f) Herramientas de soporte

¿Cualquier herramienta de soporte?

g) Soporte a la trazabilidad

¿En caso de que la herramienta soporte más de una fase del proceso de desarrollo, esto significa que son trazados/mapeados los artefactos o modelos?

2.1.1 Theme /Doc

a) Fase

Análisis de requisitos.

b) Objetivo

Identificar y especificar aspectos tempranos.

c) Artefactos

Requisitos textuales, acciones, entidades y temas. Son distinguidos dos tipos de temas, temas base y temas de corte transversal.

d) Reglas heurísticas y procesos

Theme/Doc consta de tres sub-procesos: identificación de acciones y entidad de los requisitos textuales (Identification of actions and entities from the textual

requirements), categorización de acciones dentro de temas e identificación de temas de corte transversal. Cada sub-proceso incluye reglas heurísticas. La decisión si un tema es de corte transversal es explícitamente definida, mientras que las reglas para identificar acciones y entidades, y la decisión de cuáles acciones son suficientemente importantes para ser un tema permanece implícita ("Altamente intuitiva").

e) Modelo de aplicación

Theme/Doc provee vistas que exponen cuáles comportamientos son reubicados en requisitos. La ***vista de acción principal*** es usada para mostrar gráficamente la relación entre requisitos y acciones. La ***vista de acción de corte*** es usada para representar los temas de corte transversal. Junto a estas vistas, la ***vista de tema*** es usada para planear el diseño y modelado de los temas identificados. Las vistas de temas difieren de las vistas de acciones en que estas no solamente muestran requisitos y acciones, sino que también muestran elementos claves del sistema que necesitaran ser considerados para el diseño de cada tema.

f) Herramientas de soporte

La herramienta Theme/Doc está bajo desarrollo. La herramienta toma requisitos textuales, concerns, grupos de concerns, entidades, *adjuntos,* asociaciones y *aplazamientos* como entradas. Esta herramienta crea varias vistas y soporta la identificación y especificación de concerns de corte transversal en la fase de análisis de requisitos.

g) Soporte a la trazabilidad

Theme/Doc está enfatizado solamente en la fase de análisis de requisitos. Esta metodología no soporta la trazabilidad al análisis del dominio o la fase de diseño de la arquitectura.

2.1.2 Modularization and composition of aspectual requirements - AORE

a) Fase

Análisis de requisitos.

b) Objetivo

Identificar concerns de corte transversal al nivel de análisis de requisitos, especificando y evaluando estos concerns.

c) Artefactos

Concerns, requisitos de los participantes, matriz de relación entre concerns y requisitos de los participantes, reglas de composición y una matriz de contribución.

d) Reglas heurísticas y procesos

AORE provee los siguientes siete pasos del proceso que son ejecutados de manera secuencial: identificar y especificar concerns y requisitos de los participantes, identificar granularidad de las relaciones de los requisitos de los participantes, identificar aspectos candidatos, definir reglas de composición para un aspecto candidato y requisitos de los participantes, identificar y resolver conflictos entre aspectos candidatos, redefinir requisitos y especificar dimensión de los aspectos.

De [25] se extrae, *"si bien el proceso está bien definido, no es posible decir lo mismo con respecto a las reglas y heurísticas. Algunas reglas heurísticas son explícitas, tales como, la identificación de aspectos candidatos, pero muchas otras reglas permanecen implícitas".*

e) Modelo de aplicación

El resultado del proceso es un conjunto de aspectos candidatos, los cuales son concerns que cruzan múltiples requisitos.

f) Herramienta de soporte

ARCADE (Aspectual Requirements Composition and Decision). Esta herramienta está actualmente bajo desarrollo. La herramienta hace posible definir los puntos de vista de los requisitos, requisitos aspectuales y reglas de composición usando plantillas predefinidas.

g) Soporte a la trazabilidad

AORE soporta la identificación y especificación en la fase de análisis de requisitos. Esta metodología no soporta la trazabilidad a otras fases del ciclo.

2.1.3 Integrating the NFR framework in a RE model - INFR

a) Fase

Análisis de requisitos.

b) Objetivo

Identificar concerns de corte transversal.

c) Artefactos

Documentación de participantes, catálogos, plantillas, concerns, y puntos de corte. Las plantillas son usadas para especificar un concern. Un punto de encuentro especifica cuáles concerns de corte transversal deben ser integrados con un concern dado. Reglas de composición definen el orden en el cual los concerns serán aplicados en un punto de encuentro particular.

d) Reglas heurísticas y procesos

La metodología INFR consiste de cuatro tareas: identificar concerns, especificar concerns, identificar concerns de corte transversal e integrar concerns.

e) Modelo de aplicación

Aspectos candidatos, los cuales son concerns que atraviesan transversalmente múltiples requisitos.

f) Herramienta de soporte

Actualmente esta metodología no dispone de una herramienta que la soporte.

g) Soporte a la trazabilidad

La metodología INFR se enfoca en la fase de requisitos, y no tiene soporte a la trazabilidad a otras fases tempranas del ciclo de desarrollo de software.

2.1.4 COSMOS

a) Fase

La metodología COSMOS abstrae el ciclo de vida del software. Por lo tanto este modelo puede ser usado en cada fase del ciclo de software.

b) Objetivo

Modelar concerns y una plataforma para el desarrollo alternativo de modelado de concerns.

c) Artefactos

El espacio de concerns consiste de: unidades, concerns, relaciones y restricciones. Unidades son elementos del modelo de concerns cuyo propósito es representar, en el modelo de concerns, los productos de trabajo del desarrollo de software. En el espacio de concerns, cada unidad hace referencia a un artefacto,

el cual representa los detalles del software. En COSMOS nuevos tipos de artefactos, posiblemente envuelvan lenguajes no manejados previamente puede ser introducido. Esto incluye software (tales como requisitos, diseño o programación) lenguajes como también artefactos como directorios, archivos .JAR, XML y otros documentos.

d) Reglas heurísticas y procesos

El modelo COSMOS no provee un proceso de soporte, reglas o heurísticas por medio de las cuales se pueda realizar la identificación y clasificación de los concerns y sus relaciones.

e) Modelo de aplicación

COSMOS maneja un espacio de concerns en el cual se modelan las partes del software y los intereses de éste. Las partes del software pueden ser: un proyecto simple, un componente o un sistema completo.

f) Herramienta de soporte

La herramienta Concern Manipulation Enviroment (CME) ofrece un conjunto de componentes para crear y componer concerns. Actualmente CME soporta identificación y modelado de concerns en un nivel detallado de diseño. Sin embargo, en el futuro la herramienta puede ser usada para identificación y modelado de concerns en todas las fases del ciclo de vida de software.

g) Soporte a la trazabilidad

No hay un soporte a la trazabilidad explícito porque esta metodología abstrae el ciclo de vida del software.

2.1.5 Resumen metodologías para el manejo de requisitos aspectuales

La tabla 1 presenta un resumen de las principales metodologías para el manejo de los requisitos bajo el paradigma aspectual.

	Fases del ciclo de desarrollo	Objetivo	Artefactos	Reglas heurísticas	Modelo de aplicación	Herramienta de soporte	Soporte a la trazabilidad
Theme/Doc	Análisis de requisitos	Identificar	Requisitos textuales, acciones, entidades y temas	Vistas de acción, soporte a reglas y soporte del proceso para identificación de aspectos.	Vista de tema	Actualmente en desarrollo	No
AORE	Análisis de requisitos	Identificar y especificar	Concerns, requisitos de los participantes, matriz relacionando concerns y requisitos de los participantes, una matriz de contribución	Reglas de composición, soporte del proceso para identificación y especificación de aspectos	Aspectos candidatos	ARCADE, actualmente en desarrollo	No
INFR	Análisis de requisitos	Identificar	Documentación, catálogos, plantillas, concerns y puntos de encuentro	Soporte del proceso identificación de aspectos, reglas heurísticas y composición de reglas	Aspectos candidatos, Modelo del sistema	Actualmente en desarrollo	No
COSMOS	Abstracto	Especificar aspectos	Concerns, relaciones, restricciones, unidades	-	Modelo de concerns	CME	No

Tabla 1. Evaluación de aproximaciones a aspectos tempranos. [25]

2.2 Profundizando las metodologías: AORE y COSMOS

Las instancias iniciales de este trabajo basado en las metodologías de tratamiento temprano de aspectos, llevaron a descubrir en AORE un modelo, que además de proponer la identificación y tratamiento temprano de aspectos, establece mecanismos para la identificación y negociación de conflictos desde la misma ingeniería de requisitos. Si bien AORE propone los mecanismos mencionados anteriormente, a medida que se avanza en el estudio de esta metodología, se ven evidenciadas dos falencias en el modelo:

- No se propone una *clasificación* para los concerns.

- Las relaciones (de grano grueso) propuestas por el modelo se limitan a indicar que existe una afección, positiva o negativa, entre los concerns. Hace falta en el modelo que las relaciones precisen la semántica específica de la influencia que tiene un concern sobre el otro concern involucrado en la relación.

En busca de proveer elementos que ayuden a enriquecer y fortalecer el modelo AORE se plantea el estudio de otra propuesta de tratamiento temprano de aspectos: COSMOS. Con este modelo, se busca encontrar una clasificación adecuada de los concerns que componen un sistema y cuales son las relaciones existentes entre estos.

En esta sección se presenta en detalle los elementos de cada uno de estos dos modelos y un pequeño caso de estudio reflejando su aplicación. A partir de esto, se pretende encontrar elementos que nos permitan proponer una metodología que conserve las características básicas de AORE y que brinde clasificación para concerns y las relaciones entre estos aporten mayor información la cual pueda ser usada en los demás elementos del modelo.

2.2.1 AORE

El modelo AORE es un modelo propuesto por la Universidad Nova de Lisboa, el cual se encuentra en estado de continua madurez y refinamiento. En este modelo se basa nuestro trabajo, por lo que en esta sección lo presentaremos detalladamente.

El modelo AORE ha sido motivado por la necesidad de brindar una aproximación para la ingeniería de requisitos que no sufra de la "tiranía de la descomposición dominante", tal como sucede con las aproximaciones tradicionales. Se propone un modelo que realice una descomposición de los requerimientos de una manera *uniforme*, sin importar su naturaleza funcional, o no funcional.

Estudios posteriores de los autores muestran que el fenómeno de *crosscutting* se presenta no sólo en los requisitos no funcionales, sino también en los requisitos funcionales [26], sin embargo, otras aproximaciones a un modelo de ingeniería de requisitos basado en aspectos, no tienen en cuenta este hecho, por lo que los mecanismos ofrecidos por dichas aproximaciones, no permiten realizar una manejo adecuado de los requisitos funcionales, lo cual se refleja en la carencia en la identificación y caracterización de la *influencia* de estos requisitos en otros *concerns* del sistema [27]. Dado que AORE propone un tratamiento uniforme para manejar la descomposición de los requisitos, sin importar su naturaleza (funcional o no funcional), se hace posible *proyectar* la influencia de cualquier conjunto de requisitos en un rango de otros requisitos, soportando así una separación multidimensional de los concerns. Una proyección especifica la influencia de un concern en otros concerns y es realizada a través de las *reglas de composición* propuestas por el modelo.

El modelo AORE propone un tratamiento *multidimensional* de los concerns del sistema. El enfoque multidimensional apunta a eliminar los problemas resultantes

de utilizar una descomposición dominante, lo cual se logra a través del tratamiento uniforme de los diferentes tipos de requisitos en el sistema.

El modelo soporta la detección temprana de conflictos y requisitos *sobrepuestos,* facilitando la negociación y la toma de decisiones entre los stakeholders.

2.2.1.1 El modelo AORE

El modelo propuesto realiza un tratamiento uniforme de los concerns del sistema con el objetivo de poder determinar la influencia de cualquier conjunto de requerimientos en el sistema.

Bajo este modelo, los concerns implican un *conjunto coherente* de requisitos. Sobre estos no se realiza ninguna clasificación, bien sea en casos de uso, puntos de vista o aspectos. Los concerns mantienen *encapsulados* conjuntos coherentes de requisitos tanto funcionales como no funcionales.

2.2.1.2 Espacio de concerns

Se percibe el espacio de los concerns, al nivel de requisitos, como un hipercubo. Cada cara del hipercubo representa un concern particular de interés, tal como se presenta en la figura 6.

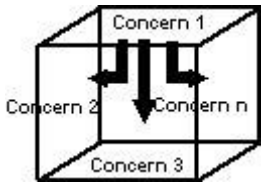

Figura 6. Espacio de concerns [27]

Al realizar un tratamiento uniforme de los requisitos, cualquiera de ellos puede ser tomado como base para proyectar la influencia de cualquier otro conjunto de concerns en esa base. La flexibilidad de una vista multidimensional hace posible manejar efectivamente los requisitos de corte transversal (crosscutting concerns) funcionales y no funcionales [27].

2.2.1.3 Espacio de meta-concerns y espacio de sistema

La idea de proponer *catálogos de concerns* se propone en [37]. Este planteamiento es retomado en [14], en el cual se propone extender la noción de los catálogos a concerns funcionales, tales como facturación, registro, reservación y otros. A partir de esta observación, se divide el espacio de los concerns en dos espacios separados:

- **Espacio del sistema**: El cual comprende los diferentes tipos de sistemas que los desarrolladores pretenden realizar.

- **Espacio de meta concerns**: Comprende un conjunto abstracto de concerns funcionales y no funcionales, los cuales repetidamente se manifiestan en diferentes sistemas.

A partir de los meta-concerns, cada sistema puede definir y especificar los concerns de su espacio de acuerdo a su domino, al tiempo que permite iterativa e incrementalmente ampliar el espacio y definición de los meta-concerns. La figura 7 presenta la especificación de los concerns del espacio del sistema a partir de los concerns del meta espacio de concerns.

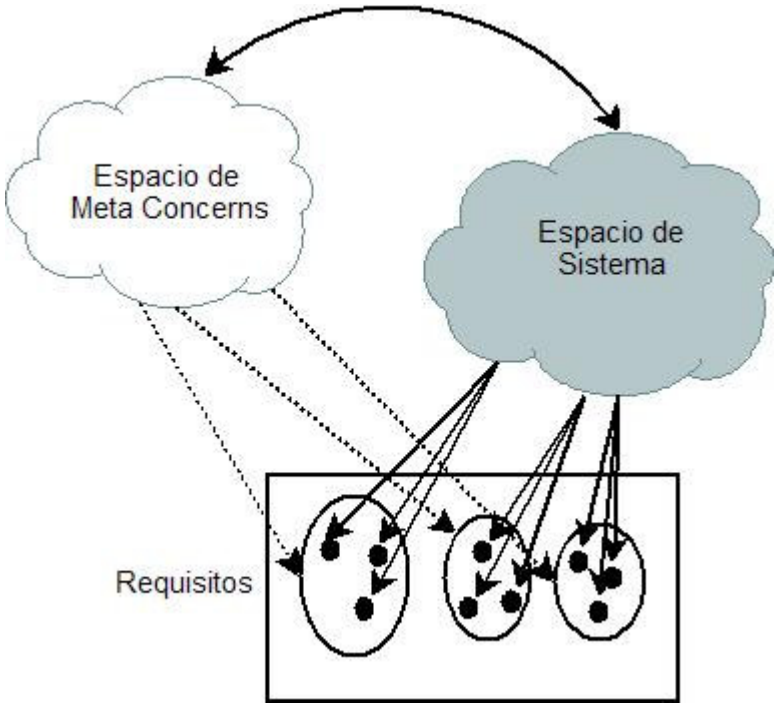

Figura 7. Espacio de meta concerns y espacio de sistema [14].

La idea de contar con espacios de meta concerns, a partir de los cuales cada sistema pueda especificar sus concerns en su espacio de sistema, puede resultar de gran utilidad al brindar al ingeniero de requisitos una base a partir de la cual pueda guiar la elicitación de los requisitos, al tiempo que disminuye la posibilidad de dejar de lado concerns y requisitos que no sean tan evidentes en el proceso de levantamiento. De igual forma, un espacio de meta concerns especifica las posibles relaciones que puedan presentarse entre los diversos concerns que componen un sistema.

2.2.1.4 Modelo de ingeniería de requisitos

El modelo de ingeniería de requisitos de AORE pude ser esquematizado como se muestra en la figura 8.

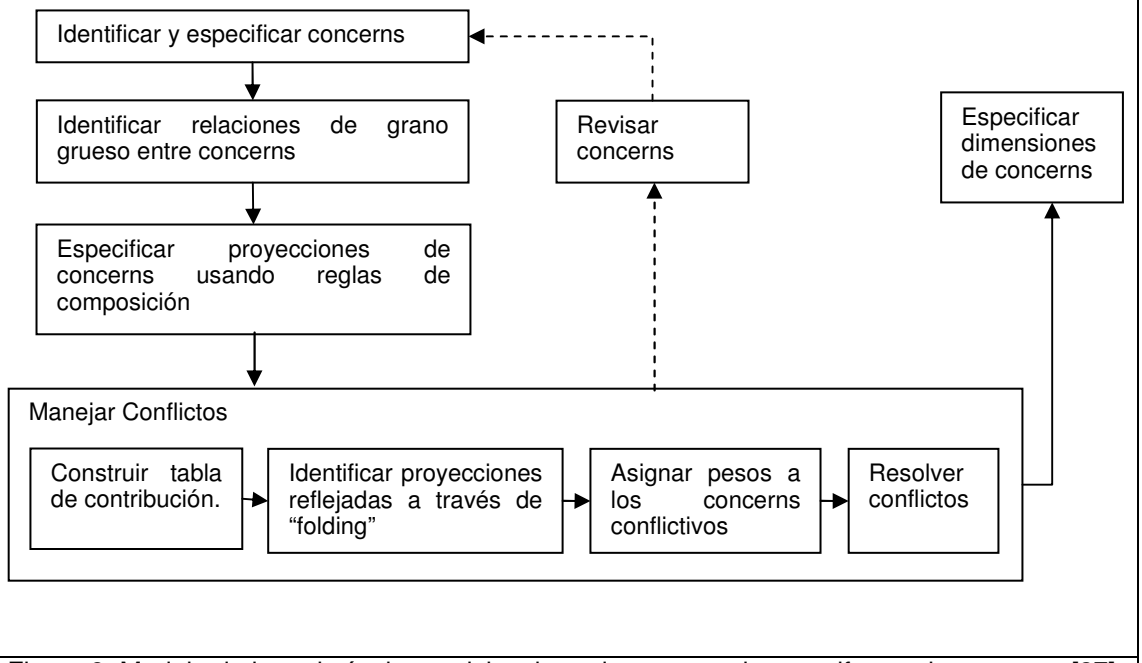

Figura 8. Modelo de ingeniería de requisitos basado en tratamiento uniforme de concerns [27]

Identificar y especificar concerns

El primer paso del modelo consiste en la identificación y especificación de los concerns. La identificación de concerns puede ser llevada a cabo a través de los mecanismos existentes de requisitos.

El modelo proporciona plantillas basadas en XML para la identificación de los concerns como se muestran en la figura 9.

```
<?xml version="1.0" encoding="UTF-8"?>
<Concern name="Visitante">
      <Requirement id="1">
            El visitante podrá recuperar la información del
            sistema.
            <Requirement id="1.1">
                  El visitante podrá acceder a la información sobre
                  las atracciones.
```

```
            </Requirement>
            <Requirement id="1.2">
                  El visitante podrá acceder información sobre su
                  ubicación.
                  <Requirement id="1.2.1">
                        El visitante podrá validar la información
                        sobre la situación si no corresponde a lo que
                        él ve.
                  </Requirement>
            </Requirement>
            <Requirement id="1.3">
                  El visitante podrá obtener una lista de tours
                  predeterminados que se encuentran disponibles.
            </Requirement>
      </Requirement>
      <Requirement id="2">
            El visitante podrá crear tours personalizados.
            <Requirement id="2.1">
                  El visitante podrá especificar preferencias.
            </Requirement>
      </Requirement>
      <Requirement id="3">
            El visitante podrá seguir un tour.
            <Requirement id="1.3">
                  El visitante podrá reconfigurar un tour.
            </Requirement>
      </Requirement>
      <Requirement id="4">
            El visitante podrá acceder a servicios externos.
            <Requirement id="4.1">
                  El visitante podrá acceder a las reservaciones del
                  hotel.
            </Requirement>
            <Requirement id="4.2">
                  El visitante podrá acceder a las reservaciones de
                  boletas de teatro.
            </Requirement>
      </Requirement>
</Concern>
```

Figura 9. Plantillas para identificación de concerns.

2.2.1.4.1.1. Identificar relaciones de grano grueso.

El siguiente paso en el modelo consiste en la identificación de las relaciones de grano grueso entre los concerns. Se relaciona cada concern con todos los otros concerns a través de una matriz. Una relación de grano grueso, es una relación

en la cual no se detallan los requisitos que están implicados en la relación de los concerns.

AORE propone diferentes técnicas para establecer las relaciones entre concerns, tales como análisis de dominio [30], ethnography [31], y procesamiento de lenguaje natural [32].

La matriz permite observar cuáles concerns influencian a otros y en qué puntos existen influencias recíprocas o bidireccionales. La figura 10 presenta la matriz de relaciones de grano grueso entre concerns.

	Concern 1	Concern 2	…	Concern n
Concern 1		✓		
Concern 2				✓
…			...	
Concern n		✓		

Figura 10. Matriz de relaciones de grano grueso.

Especificar proyecciones de concerns.

Una vez se han identificado las relaciones de grano grueso entre los concerns, se procede a especificar las posibles proyecciones de cada concern en otros concerns. Para especificar las proyecciones se utilizan las reglas de composición.

Las reglas de composición, a diferencia de las relaciones de grano grueso, operan a la *granularidad* de los requisitos individuales. Esta granularidad permite especificar como un requisito dentro del concern, influye o restringe el

comportamiento de un conjunto de requisitos de otros concerns. También se hace posible observar conflictos entre concerns a un nivel de *granularidad fina.*

El hecho de poder observar conflictos a un nivel fino de granularidad es de gran importancia para el proceso de manejo de conflictos, ya que pueden evitarse *negociaciones* innecesarias entre stakeholders para casos en los cuales pueden aparecer *aparentes* conflictos entre dos o más concerns, cuando en realidad son requerimientos diferentes o aislados los que están siendo influenciados.

De manera similar a la identificación de concerns, el modelo brinda unas plantillas por medio de las cuales se pueden especificar las proyecciones de concerns, tal como se muestra en la figura 11.

```
<?xml version="1.0" encoding="UTF-8"?>
<Composition>
      <Requirement concern="Visitor" id="all">
            <Constraint action="ensure" operation="during">
                  <Requirement concern="Mobility" id="1"
                              children="include" />
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>

      <Requirement concern="Visitor" id="all">
            <Constraint action="provide" operation="by">
            <Requirement concern="ElectronicDevice" id="all" />
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>
</Composition>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Composition>
      <Requirement concern="Mobility" id="1" children="include">
            <Constraint action="provide" operation="for">
                  <Requirement concern="Visitor" id="all"/>
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>
```

```
      <Requirement concern="Mobility" id="1" children="exclude">
            <Constraint action="enforce" operation="for">
                  <Requirement concern="Portability" id="1" />
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>

      <Requirement concern="Mobility" id="1" children="include">
            <Constraint action="affect" operation="on">
                  <Requirement concern="Context" id="1"
                              children="include" />
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>

      <Requirement concern="Mobility" id="2">
            <Constraint action="affect" operation="on">
                  <Requirement concern="Availability" id="all" />
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>
</Composition>
```

Figura 11. Plantillas para la especificación de proyecciones

Las reglas de composición, como se presentan en la figura 11, se especifican a partir del tag *Composition,* el cual encapsula un conjunto coherente de reglas de composición. El tag *Composition* está conformado por un conjunto de requisitos, los cuales tienen una estructura diferente a la presentada en la figura 9. Si el requisito del concern tiene un sub-conjunto de requisitos, estos deben ser *incluidos* o *excluidos* explícitamente en la restricción (*Constraint*) impuesta por un requisito del concern, lo cual se logra con el uso del atributo "*children".* Por otra parte, si al atributo "*id"* del requisito se le asigna el valor "*all*", indica que todos los requisitos especificados en el concern serán afectados por el *Constraint,* en otro caso, se debe especificar el identificador del requisito que se verá afectado.

El tag *Constraint* especifica una acción y un operador, los cuales definen cómo los requisitos del concern serán *afectados* por otro requisito. Las acciones establecidos por el modelo se presentan en la tabla 2.

Tipo	Descripción
enforce	Usado para imponer una condición adicional sobre un conjunto de requisitos del concern.
ensure	Usado para asegurar que una condición que debe existir para un conjunto de requisitos del concern actualmente existentes.
provide	Usado para especificar características adicionales a ser incorporadas para un conjunto de requerimientos del concern.
applied	Usado para describir reglas que aplican a un conjunto de requisitos del concern y deben alterar su resultado.
exclude	Usado para excluir algunos concerns o requisitos cuando el valor *all* es especificado.
affect	Usado para especificar que un conjunto de requisitos del concern alterarán el estado de otro concern.

Tabla 2. Acciones de restricción (Constraint actions)

Los operadores establecidos por el modelo se presentan en la tabla 3.

Tipo	Descripción
during	Describe el intervalo temporal durante el cual un conjunto de requisitos está siendo satisfecho.
between	Describe el intervalo temporal que trascurre entre la satisfacción de dos requisitos. El intervalo inicia cuando el primer requisito es satisfecho y termina cuando el segundo requisito empieza a satisfacerse.
on	Describe el punto temporal después de que un conjunto de requisitos han sido satisfechos.
for	Describe características adiciónales que complementarán los requisitos del concern.
with	Describe una condición que se sostendrá para dos conjuntos de requisitos, uno dependiente del otro.

in	Describe una condición que se sostendrá para un conjunto de requisitos que han sido cumplidos.
AND, OR, XOR	Conjunción, disyunción y disyunción exclusiva.

Tabla 3. Operadores de restricción (Constraint operators)

El tag *Outcome* especifica el resultado después de afectar el requisito del concern con otro requisito. El valor de la acción describe si otro requisito del concern, o un conjunto de requisitos de éste, deben ser satisfechos o si simplemente la restricción especificada debe ser cumplida. Los valores de acción se presentan en la tabla 4.

Tipo	Descripción
satisfied	Utilizado para asegurar que los diferentes puntos de vista de un requisito estarán satisfechos después de que las restricciones de un requisito de un concern han sido aplicadas.
fulfilled	Usado para asegurar que las restricciones de un requisito de un concern han sido realizadas con éxito.
observe	Usado para indicar que después de que se hayan aplicado los constraints, el resultado será observado para ver si la restricción del constraint se ha mantenido sobre la base o no.

Tabla 4. Acciones de resultado (Outcome actions)

Las reglas de composición buscan establecer un lenguaje que permita definir la especificación de las relaciones, al tiempo que ofrece una estructura que puede ser *legible y entendible* por los stakeholders, lo cual es de gran importancia en la ingeniería de requisitos. Un ejemplo de cómo un stakeholder puede lograr la comprensión de las reglas de composición puede verse al analizar las reglas de composición de la figura 12.

```
<?xml version="1.0" encoding="UTF-8"?>
<Composition>
      <Requirement concern="Visitor" id="all">
            <Constraint action="ensure" operation="during">
                  <Requirement concern="Mobility" id="1"
                        children="include" />
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>

      <Requirement concern="Visitor" id="all">
            <Constraint action="provide" operation="by">
            <Requirement concern="ElectronicDevice" id="all" />
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>
</Composition>
```

Figura 12. Interpretación de las reglas de composición por los stakeholders.

Si se observan las palabras resaltadas, en la figura 12, es posible realizar una interpretación como se sigue: "**All Visitor** requirements must be **ensure during** requirement **1** of **Mobility, includ**ing its children, whit the outcome that the Visitor's requirements are **fulfilled**". (**Todos** los requisitos del **Visitante** deberán estar **garantizados durante** el requisito **1** de **Movilidad**, **incluyendo** sus hijos, esta composición garantiza que los requisitos del visitante están **cumplidos**)

Manejar conflictos

Teniendo especificadas las proyecciones de los concerns a través de las reglas de composición, es posible realizar la identificación y resolución de conflictos entre los concerns.

El manejo de los conflictos se entiende como una macro-tarea del proceso que es comprendida por las siguientes actividades:

a) Construir matriz de contribución

La construcción de la matriz de contribución consiste en identificar cómo cada concern contribuye con los otros concerns con los cuales se encuentra relacionado. Una contribución puede ser positiva (+), negativa (-) o bien, no tener contribución.

La decisión a cerca del tipo de contribución para cada caso particular es comúnmente difícil de tomar. Para esto puede recurrirse al uso de catálogos existentes de concerns o bien a la experiencia y al conocimiento empírico del dominio [26].

La figura 13 presenta el ejemplo de una matriz de contribución entre concerns.

	Concern 1	Concern 2	…	Concern n
Concern 1		+		
Concern 2				-
…			...	+
Concern n		-		

Figura 13. Matriz de contribución entre concerns.

b) Folding

El *folding* consiste en *doblar* la matriz de contribuciones a lo largo de la diagonal. Este permite obtener el *efecto acumulativo* de las contribuciones para las situaciones en las cuales dos concerns se influencian directamente el uno al otro. Al doblar la matriz, obtenemos las proyecciones reflejadas, las cuales dan la influencia combinada de un conjunto de requisitos en un concern particular.

La figura 14 presenta el efecto de realizar el *folding* sobre la matriz de contribuciones.

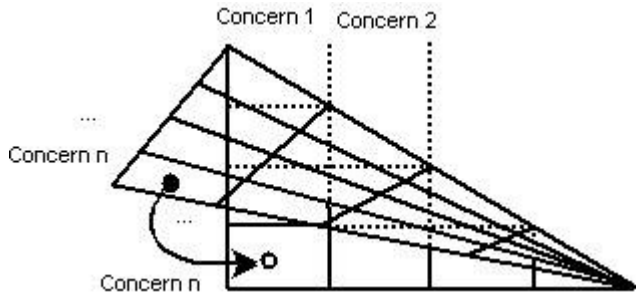

Figura 14. Tabla de contribuciones doblada por la diagonal.

Al momento de realizar el doblado de la matriz, los concerns que tienen una proyección simétrica sobre otros, obtienen sus efectos acumulados.

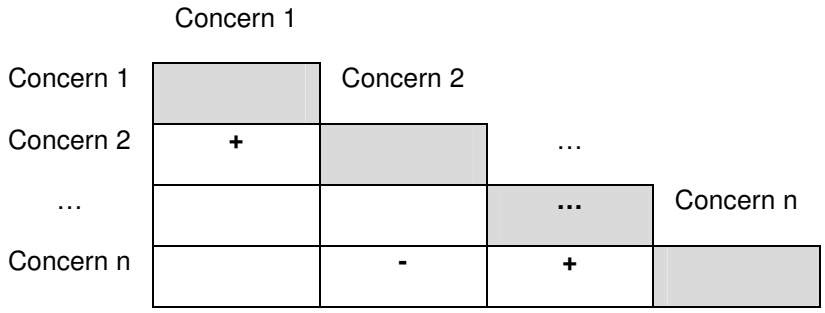

Figura 15. Proyecciones reflejadas.

La figura 15 muestra que el concern n contribuye negativamente a concern 2. Para realizar la solución de estos conflictos, se asignan los pesos a los concerns involucrados en la situación conflictiva.

c) Asignar pesos

Ya identificados los concerns que tienen una contribución negativa con otros concerns, se procede a asignar pesos. Cada peso consiste en un número en el intervalo [0 .. 1] y representa la prioridad de un concern con respecto de los concerns en los cuales se proyecta.

Los valores de los pesos son basados en las ideas de la lógica difusa y pueden ser interpretados de la siguiente manera:

- Muy importante: Toma los valores en el intervalo (0.8 .. 1.0]
- Importante: Toma valores en el intervalo (0.5 .. 0.8]
- Medio: Toma valores en el intervalo (0.3 .. 0.5]
- Poco importante: Toma valores en el intervalo (0.1 .. 0.3]
- No importante: Toma valores en el intervalo [0 .. 0.1]

El uso de valores de la lógica difusa facilita a los stakeholders la tarea de atribuir los pesos a los concerns conflictivos. En caso de presentarse concerns conflictivos con el mismo peso, los stakeholders deben definir cuál de estos deberá ser rebajado para solucionar los conflictos. Los pesos se asignarán a los concerns con respecto al concern para el cual se especificó la regla de composición.

d) Resolver conflictos

La solución de conflictos se realiza por medio de la asignación de prioridades especificada en la sección C del numeral 0. Ésta puede conllevar a una revisión de la especificación de requisitos (identificación de concerns y/o reglas de composición). Si esto sucede, entonces las proyecciones son revisadas y cualquier conflicto resultante puede ser resuelto. El ciclo debe repetirse hasta que todos los conflictos sean resueltos.

Especificar dimensiones de concerns

La última actividad del modelo es la identificación de las dimensiones de los concerns. Según se argumenta en [27], los concerns en etapas tempranas pueden tener impacto en los artefactos de etapas posteriores del ciclo de desarrollo y que pueden ser descritas en términos de 2 dimensiones:

- **Mapping**: Un concern se puede *mapear* en una característica o función del sistema (un método simple, un objeto o componente), decisión (decisión para la selección de arquitectura) o aspecto de diseño (y por tanto implementación).

Se debe notar que a pesar de que algunos concerns pueden tener una naturaleza "crosscutting" en esta etapa, algunos de estos pueden no mapearse en un aspecto en etapas posteriores.

- **Influence:** Un concern puede *influenciar* diferentes puntos en un ciclo de desarrollo, ejemplo: *disponibilidad* influencia la arquitectura del sistema, mientras que *movilidad* influencia la especificación, la arquitectura, el diseño e implementación.

La tabla 5 presenta un ejemplo de la especificación de las dimensiones de concerns.

Concern	Influence	Mapping
Visitante	Especificación, diseño, evolución	Función
Movilidad	Especificación, arquitectura, diseño, implementación	Aspecto
Disponibilidad	Arquitectura	Decisión
Contexto	Arquitectura, diseño, implementación	Aspecto

Tabla 5. Especificación de las dimensiones de concerns.

2.2.1.5 Ejemplo de aplicación de AORE

Con el fin de realizar una aplicación práctica del modelo AORE que nos ayude a entender claramente los elementos y metodología propuesta, nos remitimos al ejemplo propuesto en [23]. El caso de estudio y la aplicación del modelo AORE se presentan a continuación.

Por razones prácticas, y dado que en otra sección se realiza una aplicación completa del modelo, en este caso de estudio sólo se llega hasta la construcción de las tablas de contribución. Consideramos que hasta dicho punto, se han presentado y aplicado los elementos más importantes del modelo.

ACME Warehouse Magnament Inc. [23]

El sistema soportará el manejo de almacén. El sistema de órdenes de la compañía ACME Worehouse Magnament Inc, se especializa en dar soporte a sus clientes con espacios de almacén en toda la nación. Ejemplos de clientes son compañías que necesitan espacio para almacenar sus productos antes de que estos sean enviados, o por compañías que necesitan almacenes locales sin tener oficinas locales. ACME ya es una especialista en el almacenamiento de diferentes tipos de ítems y en el uso de camiones para redistribuir los ítems. ACME planea crecer y ahora necesita un sistema de información con el cual pueda crecer. La idea es ofrecer a los clientes espacios de almacén y servicios de distribución entre diferentes almacenes con un soporte completamente computarizado. El servicio incluye redistribución al interior del almacén y redistribución entre almacenes, lo que significa que es importante diferenciar entre ciertos tipos de ítems; por ejemplo, algunos ítems no deben entrar en contacto con otros ítems (como químicos industriales y comida). Las siguientes personas estarán usando el sistema de una u otra forma:

- Foreman: Responsable de un almacén.
- Warehouse worker: Trabaja en un almacén, cargando y descargando.
- Truck driver: Maneja un camión entre diferentes almacenes.
- Forkfilt operator: Maneja un montacargas en un almacén.
- Office personnel: Recibe órdenes y peticiones de los clientes.
- Clientes: Son dueños de los ítems en el almacén y dan instrucciones de donde y cuando ellos requieren los ítems.

Es fundamental para el sistema ACME que este pueda ser descentralizado como sea posible y

que todas las personas involucradas puedan alcanzarlo en todo momento Por tanto los conductores de camión deben tener dispositivos de comunicaciones para obtener sus ordenes y ellos podrán comunicarse con el foreman de la oficina. Esto significa que también necesitaremos una red de radio comunicación, un sistema que no debe ser desarrollado por nosotros pero será comprado separadamente. Los trabajadores del almacén, para cargar y descargar deben usar un lector de código de barras cuando manipulan los ítems, en orden a ser eficientes como sea posible. Esto significa que todos los ítems deben ser marcados cuando sean insertados en el sistema del almacén por el trabajador del almacén; esta marca debe, al mismo tiempo, dar información a cerca del ítem al sistema de información.

El foreman podrá trabajar con varios ítems al mismo tiempo; por lo que probablemente necesitara una Terminal basada en ventanas. Ellos son los responsables de efectuar la redistribución de las órdenes desde la oficina.

Cuando un cliente quiere hacer algo con sus ítems, el contactará la oficina, la cual se sucesivamente enviara ordenes de redistribución al sistema. Eventualmente, si el sistema trabaja bien, ACME planea dar a sus clientes terminales para que ellos puedan interactuar directamente con el sistema.

El sistema deberá usar una base de datos relacional (puesto que al momento ACME tiene toda su información en bases de datos y no desea cambiarlo) y la aplicación deberá ser codificada en C++. El sistema deberá tener una implementación distribuida.

A partir del texto presentado se obtiene el siguiente catalogo de requisitos.

Catalogo de requisitos identificados para el caso de estudio.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Requirement SYSTEM "DTDConcern.dtd" >
<Requirement id="0">Catalogo de requisitos
      <Requirement id="1">Se debe permitir definir la información de los
                        tipos de items del almacén.
            <Requirement id="1'1">Se podrá ingresar los tipos de items
                        que se tendrán en el almacén.</Requirement>
            <Requirement id="1'2">Se podrá modificar la información de
                        los tipos items.</Requirement>
            <Requirement id="1'3">Se podrá consultar los tipos de
                        items.</Requirement>
```

```
            <Requirement id="1'4">Se podrá dar de baja un tipo de item
                        existente.</Requirement>
            <Requirement id="1.5">Se podrá definir los tipos de items que
                        pueden almacenarse juntos.</Requirement>
            <Requirement id="1.6">Se podrá definir los tipos de items que
                        pueden transportarse juntos.</Requirement>
            <Requirement id="1.7">El sistema deberá considerar
                        información de los tipos de items que se manejan
                        en el almacén.</Requirement>
      </Requirement>
      <Requirement id="2">Se debe permitir definir lotes de ingreso de
                        items al almacén.
            <Requirement id="2.1.">El sistema deberá considerar la
                  información de los lotes de ingreso al almacén. En
                  concreto:
                        Código de lote,
                        Cliente,
                        Item,
                        Fecha de ingreso,
                        Operario recibe,
                        Cantidad recibida,
                        Cantidad disponible,
                        Almacén origen,
                        Vehiculo ingreso
            </Requirement>
            <Requirement id="2.2">Un lote sólo puede tener un item
                  asociado.</Requirement>
            <Requirement id="2.3">Se debe permitir consultar información
                  de los lotes.
                  <Requirement id="2.3.1">Se debe permitir imprimir la
                        información de los lotes.</Requirement>
                  <Requirement id="2.3.2">Se debe permitir obtener los
                        lotes para un cliente especifico.</Requirement>
                  <Requirement id="2.3.3">Se debe permitir obtener los
                        lotes por rangos de fechas.</Requirement>
                  <Requirement id="2.3.4">Se debe permitir obtener los
                        lotes por tipo de item.</Requirement>
                  <Requirement id="2.3.5">Se debe permitir obtener los
                        lotes por item.</Requirement>
                  <Requirement id="2.3.6">Se debe permitir obtener las
                        ordenes por la combinación de cualquiera de los
                        filtros del requisitos 2.3.*.</Requirement>
      </Requirement>
      </Requirement>
      <Requirement id="3">Se debe permitir definir la información de los
                        items que se almacenan.
            <Requirement id="3.1">Se podrá registrar la información de
                  los items que se ingresan al almacén.</Requirement>
            <Requirement id="3.2">El sistema deberá considerar la
                  información de los items que se ingresan al almacén. En
                  concreto:
                        Código (código de barras)
                        Descripción del item,
                        Tipo de item,
```

```
                  Lote
            </Requirement>
      </Requirement>
      <Requirement id="4">Se debe permitir la generación de ordenes de
            distribución de items.
            <Requirement id="4.1">Se debe permitir ingresar las ordenes
                  de distribución de los clientes.</Requirement>
            <Requirement id="4.2">Se debe permitir recuperar la
                  información de las ordenes de distribución de los
                  clientes.
                  <Requirement id="4.2.1">Se debe permitir imprimir las
                        ordenes.</Requirement>
                  <Requirement id="4.2.2">Se debe permitir obtener las
                        ordenes para un cliente especifico.</Requirement>
                  <Requirement id="4.2.3">Se debe permitir obtener las
                        ordenes por rangos de fechas.</Requirement>
                  <Requirement id="4.2.4">Se debe permitir obtener las
                        ordenes por tipo de item.</Requirement>
                  <Requirement id="4.2.5">Se debe permitir obtener las
                        ordenes por ítem.</Requirement>
                  <Requirement id="4.2.6">Se debe permitir obtener las
                        ordenes asignadas a un camión.</Requirement>
                  <Requirement id="4.2.7">Se debe permitir obtener las
                        ordenes por la combinación de cualquiera de los
                        filtros del requisitos 3.2.*.</Requirement>
            </Requirement>
            <Requirement id="4.3">Se debe permitir cancelar una orden de
                  distribución.</Requirement>
            <Requirement id="4.4">Se debe permitir modificar una orden de
                  distribución.</Requirement>
            <Requirement id="4.5">Una orden de distribución no se puede
                  cancelar o modificar si su fecha de despacho es
                  anterior a la fecha actual.</Requirement>
            <Requirement id="4.6.">El sistema deberá considerar la
                  información de la órdenes de distribución. En concreto:
                        Código de orden,
                        Cliente,
                        Ítem,
                        Fecha de salida,
                        Operario salida,
                        Cantidad salida,
                        Almacén destino,
                        Vehiculo envío
            </Requirement>
      </Requirement>
      <Requirement id="5">El sistema deberá utilizar la base de datos
            relacional que la empresa utiliza actualmente.</Requirement>
      <Requirement id="6">El sistema deberá ser codificado en el lenguaje
            C++.</Requirement>
      <Requirement id="7">El sistema deberá tener una implementación
            distribuida.¿Como hacer mas conciso este requisito de acuerdo
            al texto?</Requirement>
      <Requirement id="8">La interfase de usuario deberá ser basada en
            ventanas.
```

```
            <Requirement id="8.1">La interfase de usuario debe permitir
            manipular varios ítems al tiempo.</Requirement>
      </Requirement>
      <Requirement id="9">El sistema deberá permitir conectividad con
            dispositivos móviles.</Requirement>
      <Requirement id="10">El sistema deberá, a futuro, permitir
            conectividad con extranet para ser accedido por los
            clientes.</Requirement>
      <Requirement id="11">El sistema deberá permitir el uso de lectores
            de código de barras para manipular la información de los
            ítems.
            <Requirement id="11.1">El sistema deberá permitir consultar
                  la información de un item a partir de la lectura por un
                  lector de código de barras</Requirement>
            <Requirement id="11.2">El sistema deberá permitir el ingreso
                  de ítems al almacén por medio de la lectura de su
                  código de barras.</Requirement>
            <Requirement id="11.3">El sistema deberá permitir el retiro
                  de ítems al almacén por medio de la lectura de su
                  código de barras.</Requirement>
      </Requirement>
      <Requirement id="12">Las labores de administración del sistema sólo
            pueden ser realizadas por el foreman.</Requirement>
      <Requirement id="13">Las labores de ingreso lotes sólo pueden ser
            realizadas por warehouse workers.</Requirement>
      <Requirement id="14">Las labores de generación de ordenes sólo
            pueden ser realizadas por office personnel.</Requirement>
      <Requirement id="15">A futuro, los clientes podrán acceder al
            sistema directamente para generar ordenes de
            salida.</Requirement>
<Requirement id="16">La información de ordenes de salida de ítems podrá
            ser consultada por los truck drivers por medio de
            dispositivos móviles.</Requirement>
</Requirement>
```

2.2.1.5.1. Identificar y especificar concerns

A continuación se presentan los concerns identificados a partir de catalogo de requisitos.

Concerns Identificados para el caso de estudio

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Concern SYSTEM "DTDConcern.dtd" >
<Concern name="Cliente">
      <Requirement id="15">A futuro, los clientes podrán acceder al
```

```
            sistema directamente para generar ordenes de
            salida.</Requirement>
</Concern>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Concern SYSTEM "DTDConcern.dtd" >
<Concern name="Condiciones de desarrollo">
      <Requirement id="5">El sistema deberá utilizar la base de datos
            relacional que la empresa utiliza actualmente.</Requirement>
      <Requirement id="6">El sistema deberá ser codificado en el lenguaje
            C++.</Requirement>
</Concern>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Concern SYSTEM "DTDConcern.dtd" >
<Concern name="Conectividad">
      <Requirement id="10">El sistema deberá, a futuro, permitir c
            conectividad con extranet para ser accedido por los
            clientes.</Requirement>

</Concern>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Concern SYSTEM "DTDConcern.dtd" >
<Concern name="Dispositivos">
      <Requirement id="11">
            El sistema deberá permitir el uso de lectores de código de
            barras para manipular la información de los ítems.
            <Requirement id="11.1">El sistema deberá permitir consultar
                  la información de un ítem a partir de la lectura por un
                  lector de código de barras</Requirement>
            <Requirement id="11.2">El sistema deberá permitir el ingreso
                  de ítems al almacén por medio de la lectura de su
                  código de barras.</Requirement>
            <Requirement id="11.3">El sistema deberá permitir el retiro
                  de ítems al almacén por medio de la lectura de su
                  código de barras.</Requirement>
      </Requirement>
</Concern>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Concern SYSTEM "DTDConcern.dtd" >
<Concern name="Distribucion">
      <Requirement id="7">El sistema deberá tener una implementación
            distribuida</Requirement>
</Concern>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Concern SYSTEM "DTDConcern.dtd" >
```

```
<Concern name="Ergonomía">
      <Requirement id="8">
            La interface de usuario deberá ser basada en ventanas.
            <Requirement id="8.1">La interface de usuario debe permitir
                  manipular varios items al tiempo.
            </Requirement>
      </Requirement>
</Concern>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Concern SYSTEM "DTDConcern.dtd" >
<Concern name="Parametrizacion">
      <Requirement id="12">Las labores de administración del sistema sólo
            pueden ser realizadas por el foreman.</Requirement>
      <Requirement id="1">
            Se debe permitir definir la información de los tipos de ítems
            del almacén.
            <Requirement id="1'1">Se podrá ingresar los tipos de ítems
                  que se tendrán en el almacén.</Requirement>
            <Requirement id="1'2">Se podrá modificar la información de
                  los tipos de ítems.</Requirement>
            <Requirement id="1'3">Se podrá consultar los tipos de
                  items.</Requirement>
            <Requirement id="1'4">Se podrá dar de baja un tipo de ítem
                  existente.</Requirement>
            <Requirement id="1.5">Se podrá definir los tipos de ítems que
                  pueden almacenarse juntos.</Requirement>
            <Requirement id="1.6">Se podrá definir los tipos de ítems que
                  pueden transportarse juntos.</Requirement>
            <Requirement id="1.7">El sistema deberá considerar
                  información de los tipos de ítems que se manejan en el
                  almacén.</Requirement>
      </Requirement>
</Concern>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Concern SYSTEM "DTDConcern.dtd" >
<Concern name="Manejo de lotes>
      <Requirement id="2">Se debe permitir definir lotes de ingreso de
            ítems al almacén.
            <Requirement id="2.1.">El sistema deberá considerar la
                  información de los lotes de ingreso al almacén. En
                  concreto:
                        Código de lote,
                        Cliente,
                        Ítem,
                        Fecha de ingreso,
                        Operario recibe,
                        Cantidad recibida,
                        Cantidad disponible,
                        Almacén origen,
                        vehiculo ingreso
```

```
            </Requirement>
            <Requirement id="2.2">Un lote sólo puede tener un ítem
                  asociado.</Requirement>
            <Requirement id="2.3">Se debe permitir consultar información
                  de los lotes.
                  <Requirement id="2.3.1">Se debe permitir imprimir la
                        información de los lotes.</Requirement>
                  <Requirement id="2.3.2">Se debe permitir obtener los
                        lotes para un cliente especifico.</Requirement>
                  <Requirement id="2.3.3">Se debe permitir obtener los
                        lotes por rangos de fechas.</Requirement>
                  <Requirement id="2.3.4">Se debe permitir obtener los
                        lotes por tipo de item.</Requirement>
                  <Requirement id="2.3.5">Se debe permitir obtener los
                        lotes por ítem.</Requirement>
                  <Requirement id="2.3.6">Se debe permitir obtener las
                        ordenes por la combinación de cualquiera de los
                        filtros del requisitos 2.3.*.</Requirement>
            </Requirement>
      </Requirement>
            <Requirement id="3">Se debe permitir definir la información
                  de los ítems que se almacenan.
            <Requirement id="3.1">Se podrá registrar la información de
                  los ítems que se ingresan al almacén.</Requirement>
            <Requirement id="3.2">El sistema deberá considerar la
                  información de los ítems que se ingresan al almacén. En
                  concreto:
                        Código (código de barras)
                        Descripción del ítem,
                        Tipo de ítem,
                        Lote
            </Requirement>
      </Requirement>
      <Requirement id="4">
            Se debe permitir la generación de órdenes de distribución de
            ítems.
            <Requirement id="4.1">Se debe permitir ingresar las ordenes
                  de distribución de los clientes.</Requirement>
            <Requirement id="4.2">
                  Se debe permitir recuperar la información de las
                  órdenes de distribución de los clientes.
                  <Requirement id="4.2.1">Se debe permitir imprimir las
                        ordenes.</Requirement>
                  <Requirement id="4.2.2">Se debe permitir obtener las
                        ordenes para un cliente especifico.</Requirement>
                  <Requirement id="4.2.3">Se debe permitir obtener las
                        ordenes por rangos de fechas.</Requirement>
                  <Requirement id="4.2.4">Se debe permitir obtener las
                        ordenes por tipo de item.</Requirement>
                  <Requirement id="4.2.5">Se debe permitir obtener las
                        ordenes por ítem.</Requirement>
                  <Requirement id="4.2.6">Se debe permitir obtener las
                        ordenes asignadas a un camión.</Requirement>
                  <Requirement id="4.2.7">Se debe permitir obtener las o
```

```
                        ordenes por la combinación de cualquiera de los
                        filtros del requisitos 3.2.*.</Requirement>
            </Requirement>
            <Requirement id="4.3">Se debe permitir cancelar una orden de
                  distribución.</Requirement>
            <Requirement id="4.4">Se debe permitir modificar una orden de
                  distribución.</Requirement>
            <Requirement id="4.5">Una orden de distribución no se puede
                  cancelar o modificar si su fecha de despacho es
                  anterior a la fecha actual.</Requirement>
            <Requirement id="4.6.">El sistema deberá considerar la
                  información de las órdenes de distribución. En
                  concreto:
                        Código de orden,
                        Cliente,
                        Ítem,
                        Fecha de salida,
                        Operario salida,
                        Cantidad salida,
                        Almacén destino,
                        Vehiculo envío
            </Requirement>
      </Requirement>
      <Requirement id="14">Las labores de generación de ordenes sólo
            pueden ser realizadas por office personnel.
      </Requirement>
</Concern>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Concern SYSTEM "DTDConcern.dtd" >
<Concern name="Generación de ordenes">
      <Requirement id="14">Las labores de generación de ordenes sólo
            pueden ser realizadas por office personnel.</Requirement>
</Concern>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Concern SYSTEM "DTDConcern.dtd" >
<Concern name="Movilidad">
            <Requirement id="16">La información de ordenes de salida de
                  ítems podrá ser consultada por los truck drivers por
                  medio de dispositivos móviles.</Requirement>
</Concern>
```

2.2.1.5.2. Identificar relaciones de grano grueso entre concerns.

La tabla 6 presenta las relaciones de grano grueso identificadas para el caso de estudio.

	Cliente	Cond. Dllo	Conecti-vidad	Disposi-tivos	Distrib-ción	Manejo de lotes	Parame-trización	Generación de ordenes	Movilidad	Ergo-nomía
Cliente			x			x		x		
Cond. Dllo				x						x
Conectividad	x				x					
Dispositivos		x	x			x			x	
Distribución		x	x							
Manejo de lotes				X						
Parametrización						x				
Generación de ordenes	x					x				
Movilidad						x				
Ergonomía	x	x				X	x	x		

Tabla 6. Matriz de relaciones entre concerns

2.2.1.5.3. Especificar proyecciones de concerns

Las proyecciones de concerns se especifican con las siguientes reglas de composición.

Especificación de reglas de composición

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Composition SYSTEM "../DTDComposiciones.dtd" >
<Composition>
      <Requirement concern="Cliente" id="15">
            <Constraint action="provide" operation="for">
                  <Requirement concern="Conectividad" id="10" />
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>
      <Requirement concern="Cliente" id="15">
            <Constraint action="provide" operation="in">
                  <Requirement concern="Manejo lotes" id="4"
                        children="include"                  />
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>
      <Requirement concern="Cliente" id="15">
            <Constraint action="enforce" operation="for">
                  <Requirement concern="Oficce Personnel" id="14" />
            </Constraint>
            <Outcome action="fulfilled" />
      </Requirement>
</Composition>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Composition SYSTEM "../DTDComposiciones.dtd" >
<Composition>
      <Requirement concern="Condiciones de desarrollo" id="6">
            <Constraint action="enforce" operation="for">
                  <Requirement concern="Dispositivos" id="11"
                  children="include"/>
            </Constraint>
            <Outcome action="fulfilled"></Outcome>
      </Requirement>
      <Requirement concern="Condiciones de desarrollo" id="6">
            <Constraint action="enforce" operation="with">
                  <Requirement concern="Distribución" id="7"/>
            </Constraint>
            <Outcome action="fulfilled"></Outcome>
      </Requirement>
      <Requirement concern="Condiciones de desarrollo" id="6">
            <Constraint action="provide" operation="for">
```

```
                  <Requirement concern="Ergonomía" id="8"
                  children="include"/>
            </Constraint>
            <Outcome action="fulfilled"></Outcome>
      </Requirement>
</Composition>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Composition SYSTEM "../DTDComposiciones.dtd" >
<Composition>
      <Requirement concern="Conectividad" id="10">
            <Constraint action="ensure" operation="on">
                  <Requirement concern="Cliente" id="15" />
            </Constraint>
            <Outcome action="satisfied">
                  <Requirement concern="Distribución" id="7" />
            </Outcome>
      </Requirement>
</Composition>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Composition SYSTEM "../DTDComposiciones.dtd" >
<Composition>
      <Requirement concern="Ergonomía" id="8" children="inlcude">
            <Constraint action="provide" operation="for">
                  <Requirement concern="Cliente" id="15"> </Requirement>
            </Constraint>
            <Outcome action="fulfilled"></Outcome>
      </Requirement>
      <Requirement concern="Ergonomía" id="8" children="inlcude">
            <Constraint action="enforce" operation="for">
                  <Requirement concern="Condiciones de desarrollo"
                        id="6"> </Requirement>
            </Constraint>
            <Outcome action="fulfilled"></Outcome>
      </Requirement>
      <Requirement concern="Ergonomía" id="8" children="inlcude">
            <Constraint action="provide" operation="for">
                  <Requirement concern="Parametrizacion" id="1"
                              children="include"> </Requirement>
                  <Requirement concern="Manejo de lotes" id="13"
                  children="include"> </Requirement>
                  <Requirement concern="Generación de ordenes" id="14"
                              children="include"> </Requirement>
            </Constraint>
            <Outcome action="fulfilled"></Outcome>
      </Requirement>
</Composition>
```

2.2.1.5.4. Manejo de conflictos

a) Construir tabla de contribución.

La tabla 7 presenta la matriz de contribución entre los concerns.

	Cliente	Cond. Dllo	Conectividad	Dispo-sitivos	Distrib.-ción	Manejo de lotes	Parame-trización	Generación ordenes	Movilidad	Ergonomía
Cliente			+			+		+		
Cond. Dllo				-	-					-
Conectividad	+			-	-					
Dispositivos		-	+			+			+	
Distribución		-	-							
Manejo de lotes				+						
Parametrización						+				
Generación ordenes	+					+				
Movilidad						+				
Ergonomía	+	-				+	+	+		

Tabla 7. Matriz de contribución entre concerns

2.2.1.6 Observaciones al modelo AORE

El modelo AORE ofrece un marco de trabajo bajo el cual podemos realizar la identificación de los concerns en etapas tempranas al tiempo que brinda elementos que permite la identificación y negociación de conflictos. El objetivo de este trabajo es analizar la viabilidad del modelo para ser utilizado a escala industrial. Para tal propósito hemos encontrado algunas oportunidades de mejora del modelo, las cuales presentamos a continuación.

- En el modelo existen diversos elementos que no se encuentran claramente definidos y contrario a lo que se espera, los ejemplos planteados por los autores no terminan de profundizar, dificultando de esta forma su entendimiento y aplicación. De igual forma, la falta de heurísticas para la identificación y definición de elementos tales como las reglas de composición, deja que esta labor quede al criterio del ingeniero de requisitos que esté aplicando el modelo.

- Basados en la experiencia como analistas de requisitos de una empresa del sector de desarrollo de software y respaldados en equipos interdisciplinarios de investigación y desarrollo tanto del sector educativo como privado, consideramos que para llegar a un mejor modelado de los concerns que aparecen en las etapas tempranas del ciclo de vida del software, se hace necesario proveer al modelo de nuevos elementos que nos permitan en primer lugar identificar y clasificar dichos concerns de una manera más amplia.

- El modelo parte de una especificación de requisitos, la cual puede ser obtenida aplicando cualquier técnica de elicitación existente. Este hecho tiene sus pro y sus contra: Como positivo, destacamos la facilidad que presenta el modelo para ser adoptado, dado que el proceso de elicitación de requisitos es un proceso complejo y dinámico que debe favorecer la utilización de diferentes técnicas de

acuerdo a las condiciones del problema y/o del contexto. Sin embargo, una desventaja observada, es el hecho de que el modelo utilizado para elicitación de requisitos, no entra a precisar la semántica inherente a los diferentes tipos de concern. Una caracterización de los diferentes tipos de concerns podría facilitar el análisis de las relaciones de grano grueso y la semántica de las reglas de composición (enforce, ensure, provide, applied, exclude, affect) que en algunos casos resultan un poco confusas.

Nuestra propuesta parte de la premisa de que, en la medida en que se puedan caracterizar los concerns de diferente naturaleza (requisitos de ley, requisitos de infraestructura, condiciones de entorno, riesgos y otros) será posible entender y analizar las relación que se suceden entre dichos concerns de tal manera que este análisis favorezca el proceso de negociación y resolución de conflictos con el cliente.

Para tal propósito, este trabajo se ocupó de estudiar la propuesta de COSMOS y su factibilidad de integración con el modelo AORE. Como veremos más adelante, la característica distintiva de COSMOS radica en la definición de un esquema de clasificación de los concerns en el que es posible distinguir la naturaleza de los mismos a partir de su estructura.

- El modelo AORE no define la forma como las relaciones de grano grueso deben ser establecidas, pare esto propone recurrir a otras técnicas, tal como se especifica en 2.2.1.4.1.1.

- La falta de una herramienta de soporte para el modelo hace que la aplicación del mismo sea compleja y engorrosa. El hecho de tener que construir archivos XML para especificar los concerns y las reglas de composición no es nada práctico al momento de aplicar la metodología en un caso real.

A partir de estas observaciones, se evidencia la necesidad de buscar un modelo que pueda fortalecer a AORE en cuanto clasificación de concerns, al tiempo que brinde una mayor claridad de las relaciones entre concerns que son propuestas. En esta búsqueda aparece COSMOS, un modelo que propone un esquema de modelado de concerns por medio de la utilización de vistas multidimensionales y que puede ser aplicada a través de todo el ciclo de vida del software. COSMOS se presenta como un modelo rico en clasificación de concerns y relaciones. Con el estudio de COSMOS pretendemos encontrar elementos que nos permitan realizar una integración entre ambos modelos.

2.2.2 COSMOS

El modelo COSMOS es un modelo propuesto por Stanley Sutton e Isabelle Rouvellou, del IBM T.J. Watson Research Center. El modelo busca proveer una aproximación a la separación de concerns avanzada que esté enfocada a la aplicación multidimensional de vistas de concerns y que pueda ser aplicada a través del ciclo de vida del software. En esta sección presentaremos los elementos fundamentales que conforman el modelo y realizaremos una aplicación práctica de éste con el fin de entenderlo y analizarlo claramente.

En [23] se indica que el modelo puede exhibir características relacionadas con la ocurrencia y distribución de concerns de diferentes dominios y tipos, concerns compartidos versus únicos, concerns explícitos versus implícitos. El modelo establece la forma en que los concerns son organizados, compartidos y vistos, al tiempo que exhibe sus características y múltiples dimensiones.

Según el autor, la aplicación del modelado de concerns en etapas tempranas conlleva a un mejor entendimiento y un mejor tratamiento de los concerns y además pueden ayudar a identificar elementos reutilizables durante el análisis de requisitos. Esto facilita la formulación de vistas alternativas y representaciones de requisitos. El modelado de concerns de requisitos propuesto puede brindar un

modelado más eficiente y sistemático de concerns y tecnologías orientadas a aspectos.

Entre las motivaciones del modelo, encontramos que los aspectos pueden aparecer en cualquier etapa del ciclo de vida y en cualquiera de los artefactos que se generan en éste. Un mejor entendimiento de aspectos en requisitos puede ayudar a identificar principios orientados a aspectos y técnicas en el ciclo de vida, y proveer un uso más sistemático durante todo el ciclo de vida.

2.2.2.1 El modelo COSMOS

COSMOS es un esquema diseñado para el modelado multidimensional de espacios de concerns. Este comprende tres tipos de elementos, *concerns, relaciones y predicados.* COSMOS divide los concerns básicos en dos categorías, lógicos y físicos.

En [23] y [24] se definen los concerns lógicos como las "materias de interés" conceptuales en un sistema de software: discusiones, problemas, "ilities" (functionality, reliability, usability, efficiency, maintainability, portability), entre otras. Por otro lado, los concerns físicos representan elementos de un sistema del "mundo real", potencialmente se incluye software, hardware, sistemas y servicios. Estos pueden representar implementos, soporte u otros concerns que afecten los concerns lógicos.

Los concerns lógicos se dividen en 5 categorías: ***classifications, classes, instances, properties y topics,*** mientras que los concerns físicos se dividen en 3 categorías: ***instances*****,** ***collections*** **y** ***attributes*****.**

2.2.2.2 Elementos del modelo

Los elementos que conforman la propuesta de COSMOS son presentados en [23] y [24]. En primer lugar se tienen los concerns lógicos, los cuales pueden ser:

I) ***Instances*** representa concerns específicos para el dominio del sistema. Representa la mínima expresión del concern.

II) ***Classes*** concerns que son introducidos para categorizar otros concerns. Las *classes* pueden incluir otras *classes* y pueden ser utilizadas para clasificar *instances* lógicas, *properties* y los diversos tipos de concerns físicos.

Los concerns o *classes* que hacen parte de una *class* no tienen que estar relacionados, el agrupamiento se realiza basado en características que son comunes a las *intances*, *properties* o *classes* que agrupa.

Las *classes* en COSMOS no deben ser confundidas con las clases en notaciones de diseño o lenguajes de programación.

III) ***Properties*** son concerns que caracterizan otros concerns. Ellos pueden ser aplicados a *classifications*, *classes* o *instances*. Las *properties* aplicadas a *classifications*, aplican a las *classes* contenidas a las *classifications*; *properties* aplicadas a *classes*, son aplicadas a los miembros de la *class*.

IV) ***Topics*** son colecciones arbitrarias de concerns. Los *topics* capturan concerns relacionados a un tema, que pueden pertenecer a varias *classes*. Los *topics* proveen una forma de organizar grupos de concerns que cruzan a través de otras categorizaciones de COSMOS.

Mientras las *classifications* contienen *classes* de un tipo particular y las *classes* contienen subclases y miembros de un tipo particular, los *topics* pueden contener elementos de cualquier tipo.

V) ***Classifications*** representan sistemas de *classes*. Ellas están identificadas con una *class* raíz y transitivamente se incluyen las subclases de esta. Desde una perspectiva de modelado de concerns, una clasificación corresponde a una dimensión en un espacio de concerns.

La siguiente clasificación corresponde a los concerns físicos, los cuales pueden ser:

I) ***Instances*** son elementos específicos de sistemas de software, incluyendo productos particulares de trabajo, elementos particulares de software, sistemas específicos y servicios particulares.

Las *instances* pueden ser modeladas y analizadas independientemente, pero no implica que sea requerido hacerlo. Las instancias físicas pueden ser incluidas en el análisis de las instancias lógicas.

II) ***Collections*** grupos de *instances* o *collections* físicas. Las collections en los concerns físicos son análogas a los *topics* en los concerns lógicos. Por ejemplo paquetes de archivos fuentes y estaciones de trabajo en una red de área local. Las *collections* pueden contener *sub-collections* y ser homogéneas o heterogéneas con respecto a los tipos de *instances* físicas incluidas.

III) ***Attributes*** son propiedades específicas de *instances* o *collections*.

Se definen cuatro categorías de relaciones: ***categorical, interpretive, mapping y physical***. El esquema de COSMOS define tipos específicos de relaciones

categorical, mientras los tipos específicos de otras categorías deben ser definidas de acuerdo al dominio del sistema.

Las relaciones de tipo categórico sirven para relacionar concerns basado en sus categorías. Estas relaciones reflejan la semántica estructural de los tipos de concerns y define la integridad del modelo de concerns. COSMOS define siete subcategorías para las relaciones *categorical*.

- ***Classification*** relaciona *classes* a un sistema de *classifications*.

- ***Generalization*** relaciona *classes* en una relación de herencia.

- ***Instantiation*** relaciona *instances* a *classes* a las que pertenece. Las relaciones de tipo *instantiation* pueden ser tanto para concerns lógicos o físicos, pero dada una *class* puede contener sólo uno o el otro exclusivamente.

- ***Characterization*** relaciona *properties* a las *classes* o *instances* a las cuales se pueden aplicar.

- **Membership** relaciona instances físicas a *collections* a las cuales ellas corresponden.

- ***Attribution*** relaciona *attributes* a instances físicas o *collections*.

- ***Topicality*** relaciona concerns de cualquier tipo a un *topic*.

Las relaciones interpretativas (***interpretive)***, reflejan asociaciones de interpretación semántica entre concerns lógicos. COSMOS no predefine tipos particulares de relaciones *interpretive*, estas deben ser adicionadas de acuerdo a las necesidades de modelado de concerns.

En [23] se definen las siguientes subcategorías para relaciones *interpretive*, basadas en el caso práctico de un sistema para la gestión de bodegas. *Admission*, *contribution*, *logical-implementation*, *logical- composition*, *motivation*.

En [24] se definen las siguientes subcategorías para relaciones interpretive. Basadas en el caso práctico GPS. *Contribution*, *motivation*, *logical implementation*, *admission*.

Las relaciones físicas (*physical)*, sirven para establecer relaciones entre concern físicos.

Las relaciones de mapeo (*mapping)*, permiten relacionar concerns físicos y lógicos. Por ejemplo *physical implementation* es una relación *mapping* donde un concern físico implementa un concern lógico.

IV) **Predicado y consistencia:** El modelo incorpora varias condiciones de consistencia relacionadas al esquema básico de integridad, algunos representan restricciones en las relaciones, por ejemplo, atributos aplican solo a *instances* físicas y collections. Otros predicados son menos importantes y pueden ser definidos como extensiones, por ejemplo, ¿puede un *instance* pertenecer a más de una *class*?, ¿puede un *instance* no pertenecer a una clase?

    Otros predicados pueden ser asociados con otras partes del esquema tales como relaciones *interpretive*, por ejemplo, motivación implica contribución (pero no viceversa).

La tabla 8 presenta un resumen de los elementos del modelo.

CORE					Extensiones
Concerns	Logical	Classifications	Representan sistemas de classes y permite múltiples alternativas para clasificación de concerns.	Unidades de código pueden ser clasificadas de acuerdo al lenguaje de programación, classes representadas, operaciones contenidas, métricas complejas.	
		Classes	Categorización de concerns.	Objetivos, responsabilidades, funcionabilidad, configurabilidad, documentos de diseño, comportamiento o estados.	
		Instances	Representan concerns particulares, usualmente de alguna class.	Funciones específicas tales como adicionar un objeto, especificar comportamientos tales como validar una entrada, especificar elementos de configurabilidad tales como configurabilidad del buffer.	
		Properties	Son características que pueden aplicar a classes e instances.	Rendimiento, configurabilidad, robustez	
		Topics	Grupos de concerns, generalmente de diferentes tipos de concerns, que son típicamente relacionados a un tema de interés para un participante. Puede incluir classes, intances y properties que están relacionados a un tema.	Manejo de errores, relaciones del consumidor.	
	Physical	Collections	Grupos de concerns físicos. Las	Paquetes de archivos	

			collections pueden contener sub-collections y ser homogéneas o heterogéneas con respecto a los tipos de instances físicas incluidas.	fuentes y estaciones de trabajo en una red de área local.	
		Instances	Son elementos específicos de sistemas de software.	Productos particulares de trabajo, elementos particulares de software, sistemas específicos y servicios particulares.	
		Attributes	Valores particulares que caracterizan instances físicas y collections. Generalmente estos reflejaran las propiedades de interés entre los concerns lógicos.		
Relaciones	Categorical	Classification	Relaciona classes a un sistema de clasificación.		
		Generalization	Relaciona classes en una relación de herencia.	Comportamiento de logging es una subclase de comportamiento.	
		Instantiation	Relaciona instances a classes a las cuales corresponden. Las relaciones de tipo instantiation pueden ser tanto para concerns lógicos o físicos, pero dada una class puede contener sólo uno o el otro exclusivamente.	Adicionar un objeto es una instancia de de core-funcionality.	
		Characterization	Relaciona propiedades a las classes o instances a las cuales se pueden aplicar.	Rendimiento es una propiedad de funcionalidad.	
		Topicality	Relaciona concerns de cualquier tipo a un topic.	Garbage-collection configurability (una propertie) y garbage-collection algorithms (una class) son ambos un topic de garbage-collection.	

		Membership	Relaciona instances físicas a collections las cuales ellas corresponden.	Cache_core.java es un miembro del paquete com.ibm.ws.abr.gps.	
		Attribution	Relaciona attributes a instances de tipo físico o collections.	Tamaño es un atributo de Cache_core.java.	
	Interpretive	Significance			Admite, Contribuye a, lógicamente implementa, motiva, lógicamente parte de.
	Mapping	Association			Afectado por, describe, modela, físicamente implementado por, representa.
	Physical	PhysicalRelation			Conecta, conecta a, físicamente afecta, físicamente parte de.
Predicados	<Sin subtipos definidos>				

Tabla 8. Outline of the COSMOS concern-space modeling schema. Basada en [23] y [24]

2.2.2.3 Ejemplo de aplicación de COSMOS

De [23] se toma el ejemplo “ACME Warehouse Management Inc.”. Haciendo uso de la interpretación del modelo expuesto en el apartado anterior, se realiza un análisis del ejemplo según la interpretación propuesta, se extrae y resalta cada uno de los elementos del modelo.

Para la identificación y clasificación de los elementos del modelo en el caso de estudio, seguiremos la metodología propuesta en [23], la cual consiste en subrayar los posibles concerns y demás elementos del modelo en varios colores:

- Amarillo: Instance concerns
- Verde: Class concerns
- Rosa: Property concerns
- Gris: Ítems Semanticos no estimados como concerns.
- Turquesa: Instances físicas
- Sin resaltar: Ítems no semánticos

Si se encuentra una selección con **palabras** en otros colores, el color de la palabra indica el tipo de concern o relación que se encuentra incluido en la **selección**.

ACME Warehouse Magnament Inc. [23]
El sistema soportará el manejo de almacén. El sistema de órdenes de la compañía ACME Worehouse Magnament Inc, se especializa en dar soporte a sus clientes con espacios de almacén en toda la nación. Ejemplos de clientes son compañías que necesitan espacio para almacenar sus productos antes de que estos sean enviados, o por compañías que necesitan almacenes locales sin tener oficinas locales. ACME ya es una especialista en el almacenamiento de

diferentes tipos de ítems y en el uso de camiones para redistribuir los ítems. ACME planea crecer y ahora necesita un sistema de información con el cual pueda crecer. La idea es ofrecer a los clientes espacios de almacén y servicios de distribución entre diferentes almacenes con un soporte completamente computarizado. El servicio incluye redistribución al interior del almacén y redistribución entre almacenes, lo que significa que es importante diferenciar entre ciertos tipos de ítems; por ejemplo, algunos ítems no deben entrar en contacto con otros ítems (como químicos industriales y comida). En la figura 13.1 la idea es ilustrada esquemáticamente.

Las siguientes personas estarán usando el sistema de una u otra forma:

- Foreman(1): Responsable de un almacén.
- Warehouse worker: Trabaja en un almacén, cargando y descargando.
- Truck driver: Maneja un camión entre diferentes almacenes.
- Forkfilt operator: Maneja un montacargas en un almacén.
- Office personnel: Recibe órdenes y peticiones de los clientes.
- Clientes: Son dueños de los ítems en el almacén y dan instrucciones de donde y cuando ellos requieren los ítems.

Es fundamental para el sistema ACME que este pueda ser descentralizado como sea posible y que todas las personas involucradas puedan alcanzarlo en todo momento Por tanto los conductores de camión deben tener dispositivos de comunicaciones para obtener sus ordenes y ellos podrán comunicarse con el foreman de la oficina. Esto significa que también necesitaremos una red de radio comunicación, un sistema que no debe ser desarrollado por nosotros pero será comprado separadamente. Los trabajadores del almacén, para cargar y descargar deben usar un lector de código de barras cuando manipulan los ítems, en orden a ser eficientes como sea posible. Esto significa que todos los ítems deben ser marcados cuando sean insertados en el sistema del almacén por el trabajador del almacén; esta marca debe, al mismo tiempo, dar información a cerca del ítem al sistema de información.

El foreman podrá trabajar con varios ítems al mismo tiempo; por lo que probablemente necesitara una Terminal basada en ventanas. Ellos son los responsables de efectuar la redistribución de las órdenes desde la oficina.

Cuando un cliente quiere hacer algo con sus ítems, el contactará la oficina, la cual se sucesivamente enviara ordenes de redistribución al sistema. Eventualmente, si el sistema trabaja bien, ACME planea dar a sus clientes terminales para que ellos puedan interactuar directamente con el sistema.

El sistema deberá usar una base de datos relacional (puesto que al momento ACME tiene toda su información en bases de datos y no desea cambiarlo) y la aplicación deberá ser codificada en C++. El sistema deberá tener una implementación distribuida.

2.2.2.4 Resumen y análisis del caso de estudio original

La tabla 9 presenta la clasificación de los concerns identificados en el caso de estudio propuesto en [23].

Tipo	Descripción Sutton - Rouvellou	Descripción
Instances concerns	El sistema soportará	El sistema soportará el manejo de almacén
	El sistema de órdenes de la compañía	Órdenes
	ACME Worehouse Magnament Inc	ACME Worehouse Magnament Inc
	se especializa en	dar soporte a sus clientes con espacios de almacén en toda la nación
	dar soporte a sus clientes	compañías que necesitan espacio para almacenar sus productos antes de que estos sean enviados
	con espacios de almacén	compañías que necesitan almacenes locales sin tener oficinas locales
	en toda	almacenamiento de diferentes tipos de ítems

	ACME	uso de camiones para redistribuir los ítems (física)
	almacenamiento de diferentes	ofrecer a los clientes espacios de almacén
	uso de	servicios de distribución entre diferentes almacenes con un soporte completamente computarizado
	Camiones	redistribución al interior del almacén
	para redistribuir	redistribución entre almacenes
	los ítems	diferenciar entre ciertos tipos de ítems
	planea crecer	algunos ítems no deben entrar en contacto con otros ítems
	un sistema de información	químicos industriales
	con el cual pueda crecer	comida
	La idea es ofrecer a los clientes espacios de almacén	Foreman
	diferentes almacenes	Warehouse worker
	soporte completamente computarizado	cargando
	redistribución dentro de un almacén y entre almacenes	descargando
	diferenciar entre ciertos tipos de ítems	Truck driver
	algunos ítems no deben entrar en contacto con otros ítems	Forkfilt operator
	químicos industriales	Office personnel
	Comida	Camión (físico)
	Foreman	Montacargas (físico)
	Responsable de un almacén.	órdenes
	Warehouse worker	peticiones
	Trabaja en un almacén	Dan instrucciones de dónde y cuándo ellos requieren los ítems.
	cargando	los conductores de camión
	Descargando	obtener sus ordenes
	Truck driver	podrán comunicarse con el foreman de la oficina
	Maneja un camión entre diferentes almacenes	red de radio comunicación (físico)
	Forkfilt operador	Los trabajadores del almacén, para cargar y descargar deben usar un lector de código de barras cuando manipulan los ítems
	Maneja un montacargas en un almacén	código de barras(físico)
	Office personnel	todos los ítems deben ser marcados cuando sean insertados en el sistema del almacén
	Recibe órdenes	esta marca debe, al mismo tiempo, dar información a cerca del ítem al sistema de información.
	pedidos de los clientes	necesitará una Terminal basada en ventanas
	Clientes	Terminal basada en ventanas(físico)
	~~Son dueños de los ítems~~	~~efectuar la redistribución de las~~

		órdenes desde la oficina
	en el almacén	Cuando un cliente quiere hacer algo con sus ítems, él contactará la oficina, la cual sucesivamente enviará ordenes de redistribución al sistema
	Dan instrucciones de dónde y cuándo ellos requieren los ítems	dar a sus clientes terminales para que ellos puedan interactuar directamente con el sistema
	todas las personas involucradas	El sistema deberá usar una base de datos relacional
	los conductores de camión	base de datos relacional(físico)
	foreman	la aplicación deberá ser codificada en C++
	Oficina	El foreman podrá trabajar con varios ítems al mismo tiempo
	red de radio comunicación	
	un sistema que no debe ser desarrollado por nosotros pero será comprado separadamente	
	Los trabajadores del almacén, cargando y descargando	
	lector de código de barras	
	todos los ítems deben ser marcados cuando sean insertados en el sistema del almacén por el trabajador del almacén	
	esta marca	
	dar información a cerca del ítem	
	sistema de información	
	Los foreman	
	Terminal basada en ventanas	
	responsables de efectuar la redistribución de las órdenes desde la oficina	
	él contactará la oficina	
	la cual sucesivamente enviara ordenes de redistribución	
	al sistema	
	Eventualmente, si el sistema trabaja bien	
	ACME	
	planea	
	dar a sus clientes terminales para que ellos puedan interactuar directamente con el sistema	
	El sistema	
	base de datos relacional	
	ACME	
	tiene toda su información en bases de datos	
	y la aplicación deberá ser codificada	
	C++	
	El sistema	

	implementación	
Class concerns	el manejo de almacén	tipos de ítems
	clientes	personas
	compañías que necesitan espacio para almacenar sus productos antes de que estos sean enviados	clientes
	compañías que necesitan almacenes locales sin tener oficinas locales	ítems
	tipos de ítems	personas
	servicios de distribución	dispositivos de comunicación
	personas estarán usando el sistema	
	dispositivos de comunicaciones	
Property concerns	es una especialista	necesita un sistema de información con el cual pueda crecer
	descentralizado como sea posible	descentralizado
	puedan alcanzarlo en todo momento	alcanzarlo en todo momento
	eficientes como sea posible	eficientes
	al mismo tiempo	al mismo tiempo
	trabajar con varios ítems al mismo tiempo	al mismo tiempo
	no desea cambiarlo	si el sistema trabaja bien
	Distribuida	no desea cambiarlo
		El sistema deberá tener una implementación distribuida
Relaciones semantic	y ahora necesita	
	lo que significa que es importante	
	Es fundamental para el sistema ACME	
	Por tanto	
	para obtener sus órdenes	
	podrán comunicarse	
	Esto significa que también necesitaremos	
	en orden a ser	
	Esto significa que	
	probablemente necesitará	
	deberá usar	
	al momento	
Ítems semánticos no estimados como concerns	la nación	diferenciar
	Cuando un cliente quiere hacer algo con sus ítems	almacén

Tabla 9. Elementos identificados en la aplicación de COSMOS

2.2.2.5 Análisis

Una vez se han identificado los diferentes concerns del texto, se procede a realizar la clasificación de los mismos según las categorías propuestas por el modelo. La tabla 10 presenta la clasificación de los concerns identificados como *classes* e *instances.*

Classes	Subclasses (Nivel 1)	Subclasses (Nivel 2)	Instances
Tipos de ítems			Productos químicos industriales
			Comida
			La marca de los ítems debe dar información al sistema de información.
Personas			Foreman
			Warehouse worker
			Truck driver
			Forkfilt operator
			Office personnel
Empresas	Clientes		Compañías que necesitan espacios para almacenar sus productos antes de que estos sean enviados.
			Compañías que necesitan almacenes locales sin tener oficinas locales
			ACME
Equipamiento	Dispositivos de comunicación		Red de comunicación
	Dispositivos Electrónicos		Lector de códigos de barras
		Red de terminales (collection)	Terminales basadas en ventanas
			Terminales de clientes
	Maquinaria (No aparece en la tabla de relaciones)	Flotilla de almacén (collection)	Camiones
			Montacargas
Servicios			Dar soporte a clientes con espacios de almacén en toda la nación.
			Ofrecer a los clientes espacios de almacén
			Servicios de distribución entre diferentes almacenes con un soporte completamente computarizado.
Almacenamiento			Almacenamiento de diferentes tipos de ítems
			Diferenciar entre ciertos tipos de ítems
			Algunos ítems no deben entrar en contacto con otros ítems
			Todos los ítems deben ser marcados cuando sean insertados en el sistema del almacén.
Distribución de ítems			Redistribución al interior del almacén

		Redistribución entre almacenes
		Diferenciar entre ciertos tipos de ítems
		Uso de camiones para redistribuir los ítems (física)
		Manejo de pedidos (órdenes).
Consideraciones de tecnología		El sistema deberá usar una base de datos relacional
		La aplicación deberá ser codificada en C++
Funcionalidades por rol	Rol Foreman	Efectuar la redistribución de las órdenes desde la oficina.
		El foreman podrá trabajar con varios ítems al mismo tiempo
	Rol Trabajadores de almacén	Permitir a los trabajadores usar un lector de código de barras cuando cargan y descargan los ítems.
		Obtener órdenes en línea.
		Permitir comunicarse con el foreman de la oficina
	Rol Personal de oficina	Atender las necesidades que las compañías clientes, cuando quieren hacer algo con sus ítems, él contactará la oficina, la cual sucesivamente enviara ordenes de redistribución al sistema
	Rol Clientes	Dar instrucciones de dónde y cuándo ellos requieren los ítems.
		Proporcionar a sus clientes terminales para que ellos puedan interactuar directamente con el sistema.

Tabla 10. Clasificación de concerns en classes e instances.

La tabla 11 presenta una clasificación de los concerns identificados por *topics* y las clases que conforman cada uno de ellos.

Topics	Classes
Almacenamiento de ítems	Tipos de ítems
	Warehouse worker
	Forkfilt operador
	Foreman
	Clientes
	Dispositivos de comunicación
	Dispositivos Electrónicos
	Montacargas
	Dar soporte a clientes con espacios de almacén en toda la nación.
	Ofrecer a los clientes espacios de almacén Almacenamiento
	El sistema deberá usar una base de datos relacional
	Rol Foreman

	Rol Trabajadores de almacén
Atención de clientes	Foreman
	Office personnel
	ACME
	Clientes
	Terminales de clientes
	Servicios
	Manejo de pedidos (órdenes).
	Rol Personal de oficina
	Rol clientes
Distribución de pedidos	Foreman
	Truck driver
	Office personnel
	Compañías que necesitan espacios para almacenar sus productos antes de que estos sean enviados.
	Flotilla de almacén (Collection)
	Servicios de distribución entre diferentes almacenes con un soporte completamente computarizado
	Algunos ítems no deben entrar en contacto con otros ítems
	Terminales basadas en ventanas
	Distribución de ítems
	Rol Foreman
	Rol Personal de oficina
	Rol clientes

Tabla 11. Clasificación de concerns topics.

Finalmente, en la tabla 12 se observan las clasificaciones de concerns restantes.

Properties
Crecimiento
Descentralizado (distribuido)
Disponibilidad
Eficiencia
Concurrencia
Robustez
Continuidad

Attributes
Modelo
Frecuencia
Versión de sistema operativo
Capacidad

Tabla 12. Otras clasificaciones de concerns.

2.2.2.6 Relaciones

Las relaciones identificadas entre los concerns se presentan en la tabla 13.

Concern (Dominio en COSMOS)	Relación	Rango
Productos químicos industriales	InstanceOf	*Tipos de ítems*
Comida	InstanceOf	
La marca de los ítems debe dar información al sistema de información	InstanceOf	
Foreman	InstanceOf	*Personas*
Warehouse worker	InstanceOf	
Truck driver	InstanceOf	
Forkfilt operador	InstanceOf	
Office personnel	InstanceOf	
ACME	InstanceOf	*Empresas*
Clientes	SubclassOf	
Compañías que necesitan espacios para almacenar sus productos antes de que estos sean enviados.	InstanceOf	*Clientes*
Compañías que necesitan almacenes locales sin tener oficinas locales	InstanceOf	
Dar soporte a clientes con espacios de almacén en toda la nación.	InstanceOf	*Servicios*
Ofrecer a los clientes espacios de almacén	InstanceOf	
servicios de distribución entre diferentes almacenes con un soporte completamente computarizado	InstanceOf	
Almacenamiento de diferentes tipos de ítems	InstanceOf	*Almacenamiento*
Diferenciar entre ciertos tipos de ítems	InstanceOf	
Algunos ítems no deben entrar en contacto con otros ítems	InstanceOf	
Todos los ítems deben ser marcados cuando sean insertados en el sistema del almacén.	InstanceOf	
Redistribución al interior del almacén	InstanceOf	*Distribución de ítems*
Redistribución entre almacenes	InstanceOf	
Diferenciar entre ciertos tipos de ítems	InstanceOf	
Uso de camiones para redistribuir los ítems (física)	InstanceOf	
Manejo de pedidos (órdenes).	InstanceOf	
El sistema deberá usar una base de datos relacional	InstanceOf	*Consideraciones de tecnología*
La aplicación deberá ser codificada en C++	InstanceOf	
Rol Foreman	SubclassOf	*Funcionalidades por rol*
Efectuar la redistribución de las órdenes desde la oficina.	InstanceOf	*Rol Foreman*
El foreman podrá trabajar con varios ítems al mismo tiempo	InstanceOf	
Rol Trabajadores de almacén	SubclassOf	*Funcionalidades por rol*
Permitir a los trabajadores usar un lector de código de barras cuando cargan y descargan los	InstanceOf	*Rol Trabajadores de almacén*

ítems.		
Obtener órdenes en línea.	InstanceOf	
Permitir comunicarse con el foreman de la oficina	InstanceOf	
Rol Personal de oficina	SubclassOf	*Funcionalidades por rol*
Atender las necesidades que las compañías clientes, cuando quieren hacer algo con sus ítems, él contactará la oficina, la cual sucesivamente enviará órdenes de redistribución al sistema.	InstanceOf	*Rol Personal de oficina*
Rol clientes	SubclassOf	*Funcionalidades por rol*
Dar instrucciones de donde y cuando ellos requieren los ítems.	InstanceOf	*Rol clientes*
Proporcionar a sus clientes terminales para que ellos puedan interactuar directamente con el sistema.	InstanceOf	
Dispositivos de comunicación	SubClassOf	*Equipamiento*
Red de comunicación	InstanceOf	*Dispositivos de comunicación*
Dispositivos Electrónicos	SubClassOf	*Equipamiento*
Lector de códigos de barras	InstanceOf	*Dispositivos Electrónicos*
Red de *terminales (Collection)*	InstanceOf	*Dispositivos Electrónicos*
Terminales basadas en ventanas	MemberOf	*Red de terminales*
Terminales de clientes	MemberOf	
Flotilla de almacén (Collection)	InstanceOf	*Maquinaria*
Camiones	MemberOf	*Flotilla de almacén*
Montacargas	MemberOf	*Flotilla de almacén*
Modelo	AttributeOf	Lector de códigos de barras
		Camiones
		Montacargas
Frecuencia	AttributeOf	Red de comunicación
Versión de sistema operativo	AttributeOf	Terminales basadas en ventanas
		Terminales de clientes
Capacidad	AttributeOf	Camiones
		Montacargas
Crecimiento	PropertyOf	*Empresas*
		Equipamiento
		Todos los ítems deben ser marcados cuando sean insertados en el sistema del almacén
		Distribución de ítems
		Consideraciones de tecnología
Descentralizado (distribuido)	PropertyOf	Red de comunicación
		Red de terminales (Collection)
		El sistema deberá usar una base de datos relacional

Disponibilidad	PropertyOf	*Equipamiento*
		El sistema deberá usar una base de datos relacional
Eficiencia	PropertyOf	*Dispositivos de comunicación*
		Red de *terminales* (Collection)
		Consideraciones de tecnología
		Permitir a los trabajadores usar un lector de código de barras cuando cargan y descargan los ítems.
		Obtener órdenes en línea.
Concurrencia	PropertyOf	*Consideraciones de tecnología*
		Red de terminales (Collection)
		Red de comunicación
Robustez	PropertyOf	*Consideraciones de tecnología*
		Proporcionar a sus clientes terminales para que ellos puedan interactuar directamente con el sistema
Continuidad	PropertyOf	El sistema deberá usar una base de datos relacional
Tipos de ítems	TopicOf	Almacenamiento de ítems
Warehouse worker		
Forkfilt operador		
Foreman		
Clientes		
Dispositivos de comunicación		
Dispositivos Electrónicos		
Montacargas		
Dar soporte a clientes con espacios de almacén en toda la nación.		
Ofrecer a los clientes espacios de almacén		
Almacenamiento		
El sistema deberá usar una base de datos relacional		
Rol Foreman		
Rol Trabajadores de almacén		
Foreman	TopicOf	Atención de clientes
Office personnel		
ACME		
Clientes		
Terminales de clientes		
Servicios		

Manejo de pedidos (órdenes).		
Rol Personal de oficina		
Rol clientes		
Foreman		
Truck driver		
Office personnel		
Compañías que necesitan espacios para almacenar sus productos antes de que esos sean enviados.		
Flotilla de almacén (Collection)		
Servicios de distribución entre diferentes almacenes con un soporte completamente computarizado	TopicOf	Distribución de pedidos
Algunos ítems no deben entrar en contacto con otros ítems		
Terminales basadas en ventanas		
Distribución de ítems		
Rol Foreman		
Rol Personal de oficina		
Rol clientes		

Tabla 13. Relaciones identificadas en la aplicación de COSMOS.

2.2.2.7 Observaciones al modelo COSMOS

COSMOS ofrece un esquema de modelado bajo el cual se pueden modelar los concerns de un sistema en cualquier etapa del ciclo de vida. A continuación se presentan algunas observaciones a cerca del modelo.

- COSMOS parte de una definición en prosa de las necesidades de un sistema, la cual puede ser ambigua y contener elementos "irrelevantes". Esto hace que el análisis propuesto sea arduo y complejo. Además, si en algún momento se llegasen a cometer errores en el *marcado* de los ítems esto puede ocasionar confusiones y reprocesos en la aplicación del modelo, más aun cuando se tienen definiciones de sistemas extensos.

- La definición dada para los concerns del tipo *Classifications* y la forma como estos son abordados en los ejemplos propuestos por el autor, no hacen un aporte

considerable a la ingeniería de requisitos, ni al modelo mismo, se queda sólo en un modelado para el cual no se propone una aplicación puntual.

- El autor realiza una distinción clara entre instances físicas y lógicas al momento de dar la definición, pero en el momento de realizar la selección de los concerns tipo *classes*, se incluyen tanto *instances* físicas como lógicas. Esto implica que se tenga que realizar un esfuerzo en la clasificación inicial de los concerns, para posteriormente *ignorar* esta clasificación y tratarlas de manera uniforme.

- La forma en que COSMOS presenta las relaciones entre concerns por medio de una matriz no es la más práctica ni fácil de realizar, más aún si en el momento no se cuenta con una herramienta adecuada. Usar técnicas de modelado puede ayudar considerablemente a identificar y analizar las relaciones entre los concerns.

- Al discutir esta propuesta con otras personas ajenas el paradigma aspectual, pero con experiencia en ingeniería de requisitos, se presentan críticas por la forma *manual* como debe realizarse la selección y clasificación de los concerns. Para el caso propuesto se cuenta con una descripción breve del problema y una complejidad mínima, aun así el proceso selección y clasificación es complicado; se deducen entonces que, los casos reales cuentan con definiciones de sistema mucho más amplias y complejas, resulta en una tarea ardua aplicar este modelo en casos reales.

- COSMOS es un modelo que ofrece una visión y clasificación para los concerns y contempla elementos que *comúnmente* no son tenidos en cuenta en los procesos de ingeniería de requisitos. Sin embargo, y aunque en la definición del modelo se plantea la posibilidad de usarse en cualquier etapa del proceso de desarrollo, en el estudio de éste no se observa que al aplicar el modelo, los artefactos obtenidos de una etapa puedan servir como insumo a una etapa posterior en el ciclo de vida.

Las fortalezas del modelo COSMOS se presentan en la clasificación de los concerns y en las relaciones, sin embargo consideramos que es poco viable que el modelo pueda ser utilizado en etapas posteriores del ciclo de vida del software ya que dichas etapas requieren elementos que van mas allá que clasificación y relaciones entre concerns. Adicionalmente, existen modelos mucho mas refinados para el manejo de los concerns en etapas posteriores por ejemplo: Assam [40], A Concern-Oriented Approach To Software Architecture [41], Identifying Aspects Using Architectural Reasoning [25].

- **ASSAM**: Metodología cuyo objetivo es identificar y especificar aspectos arquitectónicos, y hacerlos *transparentes* desde etapas tempranas del ciclo de vida del desarrollo de software.

- **A Concern-Oriented Approach To Software Architecture**: Acercamiento para desarrollar y documentar arquitecturas de software, provee mecanismos para encapsular concerns de manera individual dentro de construcciones arquitectónicas independientes.

- **Identifying Aspects Using Architectural Reasoning**: El objetivo de esta metodología es derivar arquitecturas de software de los atributos de calidad de los requisitos a través del entendimiento adquirido de los modelos de atributos de calidad creados usando tácticas arquitectónicas.

2.3 Ampliando el modelo AORE

AORE se presenta como un modelo para el tratamiento de concerns en etapas tempranas del ciclo de vida del software. Este modelo, además de proponer la identificación de los concerns desde la etapa de requisitos, propone mecanismos de resolución temprana de conflictos y mapeo de los concerns a artefactos en etapas posteriores del ciclo de vida. Sin embargo, en AORE se identificó la

necesidad de extender el modelo con el fin de proveer un mecanismo de clasificación de concerns y análisis de relaciones entre estos. Esta sección presenta la propuesta de extensión del modelo.

2.3.1 Hacia un modelo integrado AORE - COSMOS

Como ya fue mencionado en el estudio del modelo AORE, se observa que el modelo no es muy amplio en cuanto a la clasificación de los concerns. Este problema surge por la misma concepción del modelo al proponer un tratamiento uniforme de los concerns sin importar la naturaleza de los requisitos que le componen.

La necesidad de extender el modelo con el fin de proveer un mecanismo de clasificación de concerns y análisis de relaciones entre estos, nos llevo a explorar el modelo COSMOS, el cual se propone un esquema de modelado de concerns y que exhibe un amplio universo de clasificaciones de concerns y relaciones entre estos.

En la exploración inicial de COSMOS, no fue fácil comprender el esquema de composición y recursividad de los diferentes elementos del modelo, más si se tiene en cuenta que los casos de estudio propuestos en [23], [24] y [28] no presentan ni profundizan en todos los elementos del modelo.

Basados en el análisis del modelo AORE y COSMOS y teniendo en cuenta sus fortalezas y debilidades ya mencionadas, este trabajo busca, robustecer el modelo AORE. Si bien encontramos que no es posible realizar una fusión completa entre ambos modelos, es posible articular buena parte de las ideas básicas en las que se sustenta COSMOS para complementar la propuesta de AORE en lo que respecta a la caracterización y clasificación de los concerns. La exploración de COSMOS brindó una visión más amplia de la que se tenía al momento de modelar los requisitos y concerns en un sistema. Esto nos lleva destacar la importancia de

intereses que están presentes en todos los sistemas y que generalmente son *obviados* en las etapas iniciales del proceso de desarrollo. La inclusión en etapas tardías de algunos intereses, generan impactos de mayor envergadura y los cuales se podían mitigar si se identificaran desde las etapas iniciales del proceso.

En busca de un modelo que nos provea estos elementos, en la sección 2.3 proponemos una expansión del modelo AORE usando ideas y fundamentos del modelo COSMOS y complementado por nuestra experiencia en la ingeniería de requisitos y el desarrollo de software.

2.3.2 Modelo AORE extendido

Proponer un modelo integrado a partir de AORE y de COSMOS, tal como se pretendía inicialmente, no fue posible; esto se percibe desde la definición misma que cada modelo presenta al concepto de *concern*, sin embargo, se logran ampliar algunos de los elementos de AORE a partir de ideas y fundamentos ofrecidos por COSMOS. Uno de los objetivos de este trabajo es fortalecer los diversos tipos de relaciones que pueden darse entre concerns en el espacio del problema. COSMOS enfatiza su trabajo en las relaciones de tipo categórico las cuales definen las características estructurales de los concerns. Nos interesa en este trabajo entrar a precisar las relaciones de tipo interpretativo que el modelo COSMOS deja abierto. El interés de una caracterización de concerns basado en la naturaleza de éstos, tiene como fin proveer al modelo de diversas vistas que permitan a cada stakeholder ver los concerns que son relevantes de acuerdo a su rol dentro del sistema, así por ejemplo los arquitectos podrán centrar su atención en vistas que le proporcionen los concerns de contexto, de calidad, físicos, sistemas existentes y servicios, mientras que un analista podrá centrar su atención en la vista de negocio.

A partir de esto se propone una extensión del modelo AORE que nos brinde la clasificación y relaciones de forma que nos permita abordar el análisis de un caso

de estudio real haciendo uso de una metodología de ingeniería de requisitos basada en un enfoque aspectual. La ampliación propuesta para el modelo consiste en la inclusión de tres nuevas actividades y la introducción de nuevos elementos que pueden ser entendidos como concerns, tales como objetivos, riesgos, atributos de calidad y modelos de referencia. Estas actividades adicionales tienen como objetivo hacer que el modelo AORE brinde mayores elementos de clasificación de concerns, mejorar la definición de relaciones de grano grueso y proveer vistas que estén orientadas a los diferentes stakeholders involucrados en el sistema. Las actividades adicionales propuestas son: (a) Clasificar concerns, (b) Complementar relaciones de grano grueso y (C) Generar vistas de concerns.

La propuesta de ampliación del modelo AORE puede ser vista en la figura 16 y el detalle de las actividades propuestas es abordado en la siguiente sección.

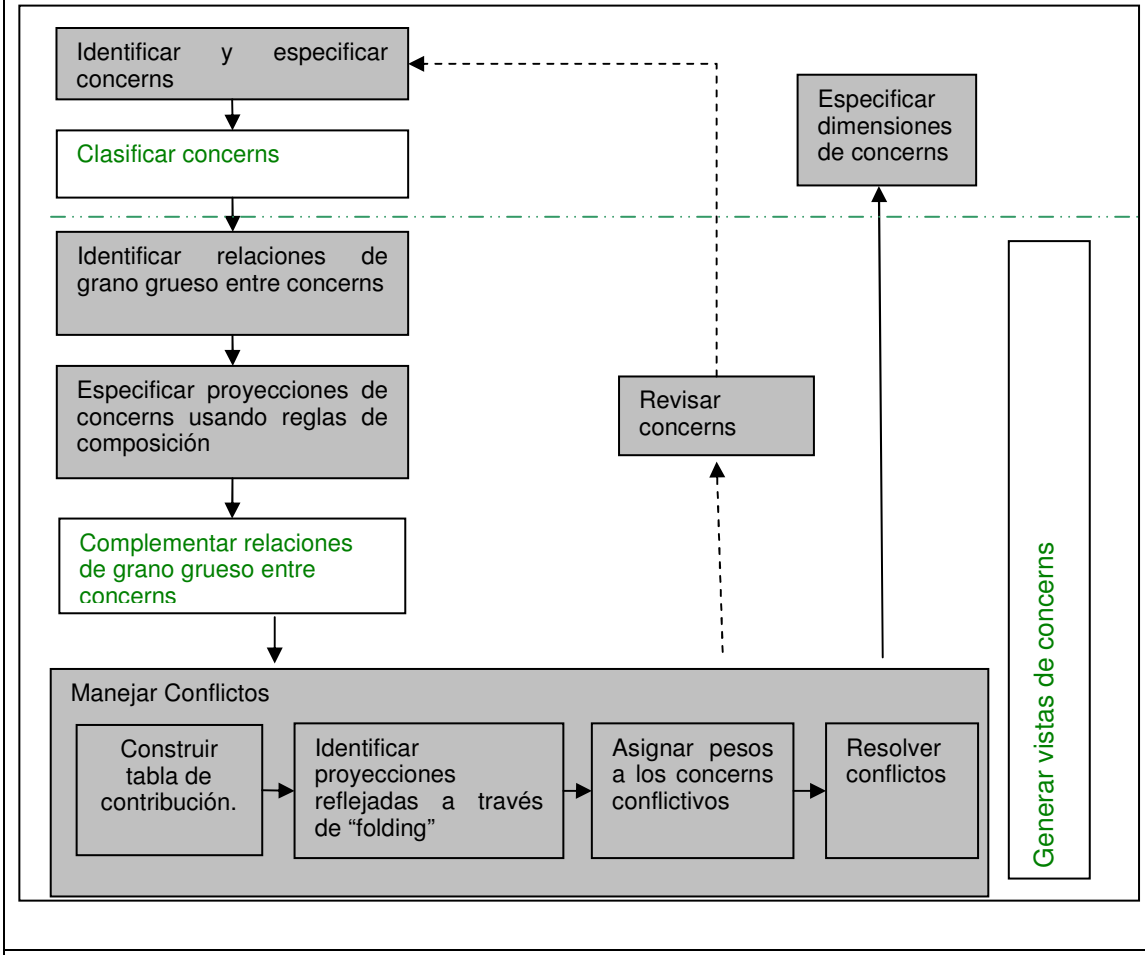

Figura 16. Ampliación del modelo AORE

2.3.2.1 Elementos de ampliación del modelo

La figura 16 presenta los elementos propuestos para realizar una ampliación del modelo AORE orientados a complementar la propuesta AORE.

2.3.2.2 Clasificar concerns

Esta actividad está orientada a proveer la clasificación de los concerns que componen un sistema y de la cual carece la propuesta inicial. Como se argumentaba anteriormente, la clasificación de los concerns está orientada a

proveer vistas que permitan a cada stakeholder centrarse en los concerns que son de su interés.

La clasificación que se propone de los concerns puede ser vista en la tabla 14. Estas clasificaciones surgen a partir de diversas motivaciones, entre las cuales se encuentran los trabajos realizados en [39], la norma ISO 9126 [35], COSMOS [9], [23], [24], la experiencia y aportes realizados por el grupo de investigación de Ingeniería de Software liderados por la Universidad EAFIT y AVANSOFT S.A.

Clasificación	Descripción	Motivación / Justificación
Objetivo	Objetivos del sistema, los cuales pueden estar agrupados o ser vistos de manera individual como un concern.	Propuesta realizada por el grupo de investigación de Ingeniería de Software y apartir de la experiencia en proyectos en la industria del software. Soportada en propuestas como VGraph [46]
Negocio	Basado en la definiciónb de concern propuesta por AORE, los concerns de negocio son una colección coherente de requisitos, los cuales pueden ser funcionales, de información y reglas de negocio.	Basados en la metodología propuesta por Amador Durán [39] y guiados por la necesidad de clasificar los concerns que plasman la funcionalidad principal de un sistema.
Riesgos	Son concerns que representan posibles dificultades a las cuales se expone el sistema en	Propuesta realizada por el grupo de investigación de

	desarrollo.	Ingeniería de Software y apartir de la experiencia en proyectos en la industria del software.
Calidad	Generalmente son concerns asociados a características no funcionales del software ("ilities" ISO 9126) y requisitos no funcionales del sistema que se tenga en cuestión. Estas características pueden afectar todo el sistema o solo partes de éste.	Basados en la norma ISO 9126 [35] y en las propuestas de requisitos como NFR [47]
Físicos	Estos concerns comprenden un conjunto amplio de elementos que no son propios del sistema, pero los cuales pueden afectarlo considerablemente en cualquiera de las etapas. Estos concerns comprenden características tales como infraestructura, plataforma (Sistema Operativo), componentes externos, entorno de desarrollo, dispositivos, equipos y frameworks.	Motivados por la propuesta encontrada, estudiada y aplicada en el modelo COSMOS [9], [23] y [24].
Contexto	Este tipo de concerns comprende elementos del sistema que no son susceptibles a cambios y el sistema los debe considerar. Ejemplos de elementos que se consideran concerns de contexto son: restricciones de ley, servicios externos, sistemas existentes, modelos de referencia.	Propuesta realizada por el grupo de investigación de Ingeniería de Software e identificados en la experiencia en proyectos en la industria del software.

Tabla 14. Clasificaciones propuestas para concerns.

La figura 17 presenta el esquema de clasificación propuesto para los concerns.

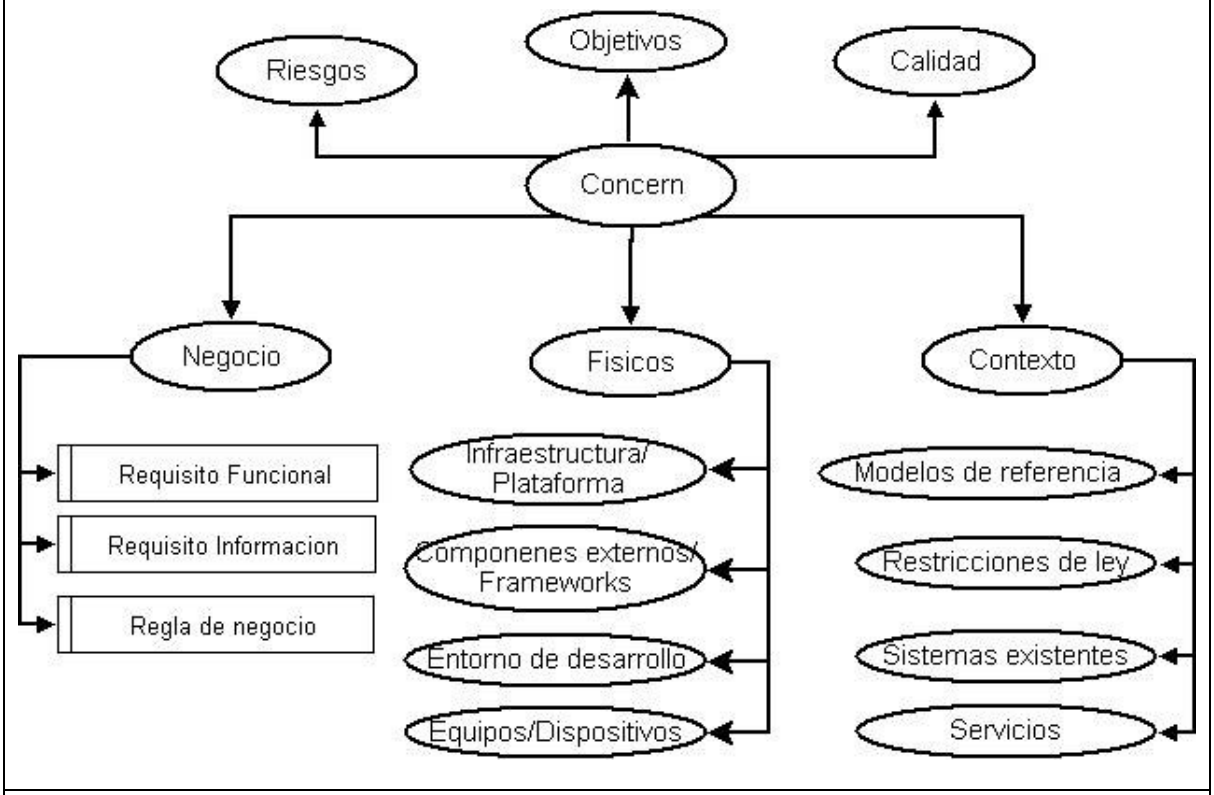

Figura 17. Clasificación de concerns

Es importante anotar que para los concerns introducidos al modelo y que no pueden entenderse como una colección coherente de requisitos (objetivos, riesgos y concerns de contexto), no tiene sentido realizar las proyecciones sobre otros concerns por medio de reglas de composición. La influencia de estos concerns se ve directamente en las relaciones de grano grueso refinadas.

2.3.2.3 Complementar relaciones de grano grueso entre concerns

Esta actividad se realiza posterior a la actividad de especificación de relaciones de grano fino, luego de que haya adquirido una visión más clara de las relaciones de grano grueso realmente existentes entre los concerns. Esto permite replantear y complementar las relaciones de grano grueso definidas anteriormente.

La actividad de definición de relaciones de grano fino implica establecer relaciones a nivel de los requisitos que componen los concerns; en este proceso es posible encontrar que no existe una relación entre los requisitos de los concerns lo cual implica que la relación de grano grueso debe desaparecer. En caso de encontrar relación entre los requisitos de los concerns, estas ayudan a identificar el tipo de relaciones que existen entre los concerns y de acuerdo al dominio del problema es posible asignar un nombre a esta relación. Las posibles relaciones de grano grueso se describen en la tabla 15.

Relación	Descripción
Contribución	Un concern contribuye a otro cuando los requisitos del concern origen ayudan al cumplimiento de los requisitos del concern destino. Esta relación es vista como la relación recíproca a la afección. La contribución se da cuando el concern origen afecta de manera positiva al concern destino.
Restricción	Un concern restringe a otro cuando los requisitos del concern origen restringen los requisitos del concern destino.
Condición	Un concern condiciona a otro cuando los requisitos del concern origen condicionan los requisitos del concern destino. Se diferencia de la restricción porque las condiciones sólo se aplican bajo ciertas reglas o circunstancias.
Uso	Un concern usa a otro concern cuando los requisitos del concern origen usan requisitos del concern destino.
Extensión	Un concern extiende a otro concern cual los requisitos del concern origen extienden los requisitos del concern destino
Afección	Un concern afecta a otro concern cuando los requisitos del concern origen

	afectan los requisitos del concern destino. La afección es considerada como un aporte negativo entre los concerns. Esta relación se considera como la relación reciproca a la contribución.
Admisión	Un concern admite a otro concern cuando los requisitos del concern origen admiten la aplicación de los requisitos del concern destino.
Se aplica a	Un concern se aplica a otro concern cuando los requisitos del concern origen son aplicados a los requisitos del concern destino sin condicionarlos. Puede entenderse como la relación reciproca a la admisión.

Tabla 15. Relaciones de grano grueso

Las relaciones entre concerns tienen interpretación de acuerdo al sentido de la relación y pueden darse de manera bidireccional. Se puede observar que las relaciones propuestas son diferentes a las acciones y operadores de restricción del modelo AORE (ver tablas 2 y 3). Estas relaciones surgen de un análisis, de cómo un conjunto de requisitos de un concern puede afectar a otro conjunto de requisitos en otro concern. Las relaciones *contribución* y *admisión* están basadas en las relaciones interpretativas propuestas en [24].

2.3.2.4 Generar vistas de concerns

La generación de las vistas de los concerns se propone como la etapa final del proceso. Esta etapa es opcional y tiene como objetivo permitir a los diferentes stakeholders obtener visiones del sistema desde diferentes perspectivas.

Las vistas principales se presentan en la tabla 16. Cualquier vista adicional y de interés para el sistema puede ser incluida a criterio de los stakeholders.

Vista	Concerns	Descripción	Stakeholders
Negocio	Objetivos,	Vista orientada a mostrar los	Analistas.

	negocio, calidad y restricciones de ley	concerns que brinden la globalidad del sistema, reflejando la funcionalidad básica y las características de calidad que éste poseerá.	desarrolladores, SQA
Funcional	Objetivos, negocio.	Proporciona una visión de las funcionalidades de negocio que el sistema proveerá.	Usuario final.
QoS	Contexto y calidad.	Proporciona una visión de los elementos invariantes del sistema y características de calidad a tener en cuenta para el diseño del sistema.	Arquitectos, desarrolladores, SQA
Plataforma	Físicos, sistemas existentes y servicios.	Proporciona una visión de los concerns que determinan el entorno en el cual operará el sistema.	Operadores, integradores sistemas y arquitectos, desarrolladores
Proyecto	Objetivos, negocio, riesgos.	Proporciona una vista, de los objetivos del sistema, funcionalidades, riesgos y atributos de calidad que el sistema debe satisfacer.	Gerentes de proyecto, SQA.

Tabla 16. Vistas de concerns.

3. ENTERPRISE ARCHITECT COMO HERRAMIENTA PARA SOPORTAR EL MODELO

Contar con herramientas que soporten los procesos de desarrollo de software se ha convertido en un factor fundamental para el éxito de toda metodología. Hoy día, la complejidad y el tamaño de los sistemas de información hace impensable poder aplicar cualquier metodología sin contar con una herramienta que permita manejar ágilmente los elementos y relaciones propuestos. La tabla 1 presenta un resumen de las metodologías más relevantes para el tratamiento de aspectos en la ingeniería de requisitos, esta tabla permite observar que en su mayoría, las metodologías no cuentan con una herramienta adecuada para respaldar el proceso o ésta se encuentra en estado de desarrollo.

En el proceso de aplicar una metodología como AORE en un caso real, se hace indispensable contar con una herramienta que facilite el proceso de aplicar los elementos del modelo. ARCADE, la herramienta que soporta el modelo se encuentra en estado de desarrollo, por lo que en el momento en que se realizó este trabajo no fue posible utilizarla.

AORE proporciona un conjunto de plantillas XML por medio de las cuales es posible establecer la definición de los concerns y las relaciones entre ellos, sin embargo, el trabajo de generar descriptores XML manualmente para la aplicación

del modelo en un caso real implica un esfuerzo demasiado grande, al tiempo que realizar el mantenimiento y evolución de estos descriptores se hace inviable.

Entreprise Architect es una herramienta robusta y flexible, que brinda soporte para el desarrollo de software, administración de proyecto, administración de requerimientos y análisis de negocio. Entreprise Architect es una herramienta CASE UML orientada a objetos y que permite cubrir el ciclo de vida completo [38]. El desarrollo de este capítulo presenta una propuesta de modelado de los elementos de AORE.

3.1 Razones para utilizar Entreprise Architect - EA

Entreprise Architect no es una herramienta creada para soportar el paradigma aspectual, sin embargo ofrece características que permiten adaptarla para obtener un esquema de modelado que permita tratar aspectos tempranos, específicamente en la ingeniería de requisitos.

Algunas de las características de EA son:

- Importación y exportación de archivos XML: EA permite exportar y/o importar archivos XML de los artefactos generados en la herramienta. Esto presenta grandes posibilidades para extender el uso de esta herramienta con futuros desarrollos que soporten completamente el paradigma aspectual.

- Definición de estereotipos: EA permite la definición de estereotipos, lo que permite extender UML para adaptar los artefactos disponibles en la herramienta para trabajar con el paradigma aspectual.

- Shape Script: EA permite la creación de scripts personalizados, los cuales son aplicables a los artefactos estereotipados en un modelo y que ayudan a la diferenciación de los nuevos elementos.

- Definición de plantillas: EA permite establecer plantillas base, a partir de los cuales es posible generar nuevos documentos que siguen una guía durante todo el ciclo de vida del software.

- Documentación: EA permite la generación de documentación RTF y HTML de alta calidad y amigable de cara al usuario.

- Diversidad de diagramas: EA permite la generación de todos los diagramas establecidos por UML al tiempo que permite la definición de nuevos diagramas personalizados.

- Fácil de aprender: EA ofrece una interfaz de usuario intuitiva y rápida con patrones de modelo fáciles de usar.

Además de las características de EA, la experiencia de los integrantes del proyecto con esta herramienta ayuda considerablemente la adaptación de ésta al paradigma aspectual.

Las secciones posteriores de este capítulo presentan la propuesta de trabajo con EA y los elementos a considerar para el uso de la herramienta.

3.2 Propuesta de modelado

En esta sección se presentan lo elementos de la herramienta Enterprise architect que se proponen para extender su uso a la metodología AORE.

3.2.1 El entorno de trabajo

La figura 18 muestra el entorno de trabajo de EA. En esta imagen se presenta una visión general de los elementos que componen la herramienta, se observa

que básicamente el entorno de trabajo está constituido por una vista de herramientas (izquierda) en la cual se encuentran los artefactos de cada etapa del proceso de desarrollo, una vista de proyecto (derecha) en la cual se hace posible observar y navegar por el conjunto de artefactos existentes en el proyecto en el cual se está trabajando. También se cuenta con el espacio de modelado (centro), en el cual es posible diagramar y relacionar los diferentes artefactos de un sistema.

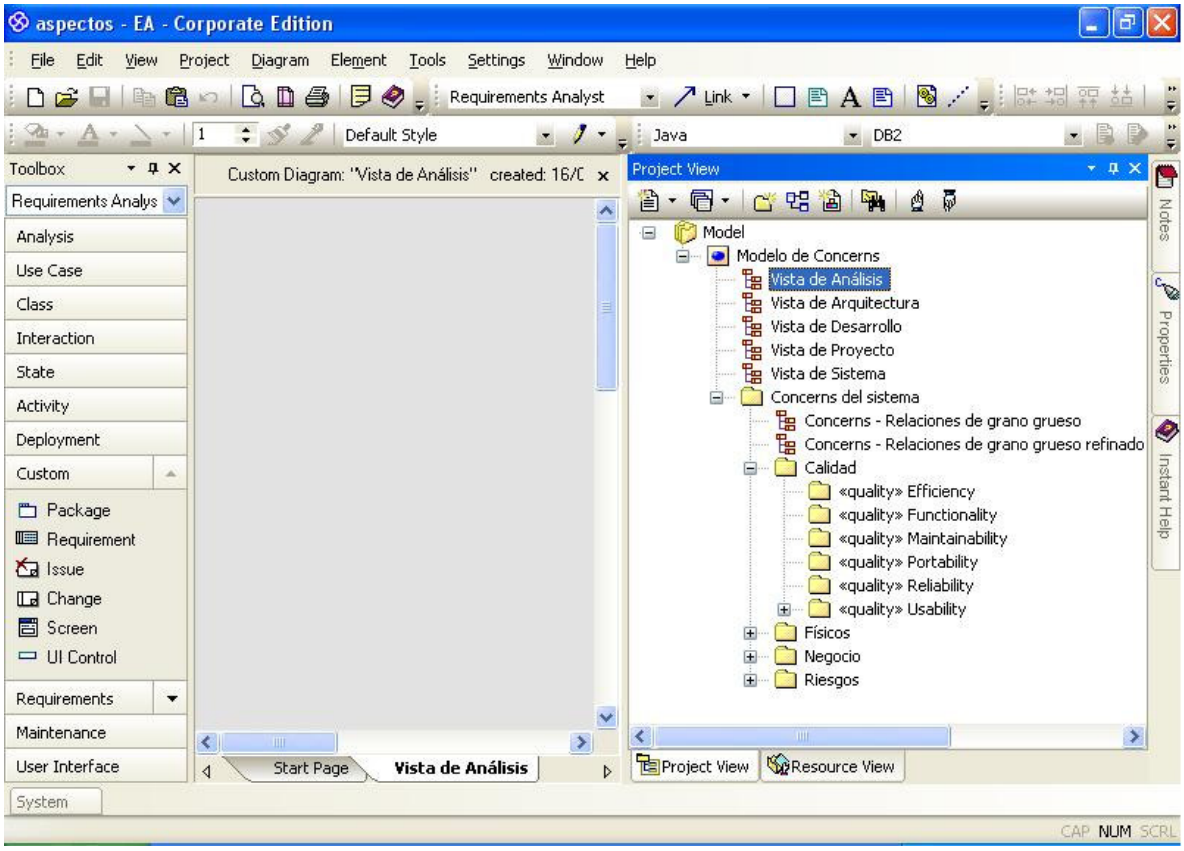

Figura 18. Vista del entorno de trabajo Enterprise Architect

Es posible observar que además de las vistas antes mencionadas, la herramienta cuenta con otro conjunto de vistas, tales como vista de recursos, propiedades, sistema y otros elementos que brindan apoyo al modelado de artefactos en las

diferentes etapas del ciclo de vida. Un mayor detalle de estos elementos de la herramienta esta fuera del alcance del presente trabajo, para más información remítase a [38].

3.2.2 Estructura de artefactos

En la figura 19 se presenta la estructura para los artefactos propuestos bajo el modelo AORE. Basados en el esquema de clasificación del modelo AORE propuesto en la sección 2.3., se observa que la organización del área del browser se estructuró siguiendo las categorías de concerns definidas. Cada paquete es estereotipado de acuerdo al tipo de concerns que define. De otra parte, las vistas de la arquitectura sirven para mostrar una visión de los concerns desde la perspectiva de los diferentes stakeholder. Estas vistas, son derivadas a partir de los concerns definidos en sus respectivas categorías y de las relaciones que dichos concerns establecen con otros. .

Un elemento a tener en cuenta, son las reglas de composición de los concerns las cuales son especificadas por medio de diagramas que están ubicados al interior del concern. Los diagramas de composición están detallados en la figura 24.

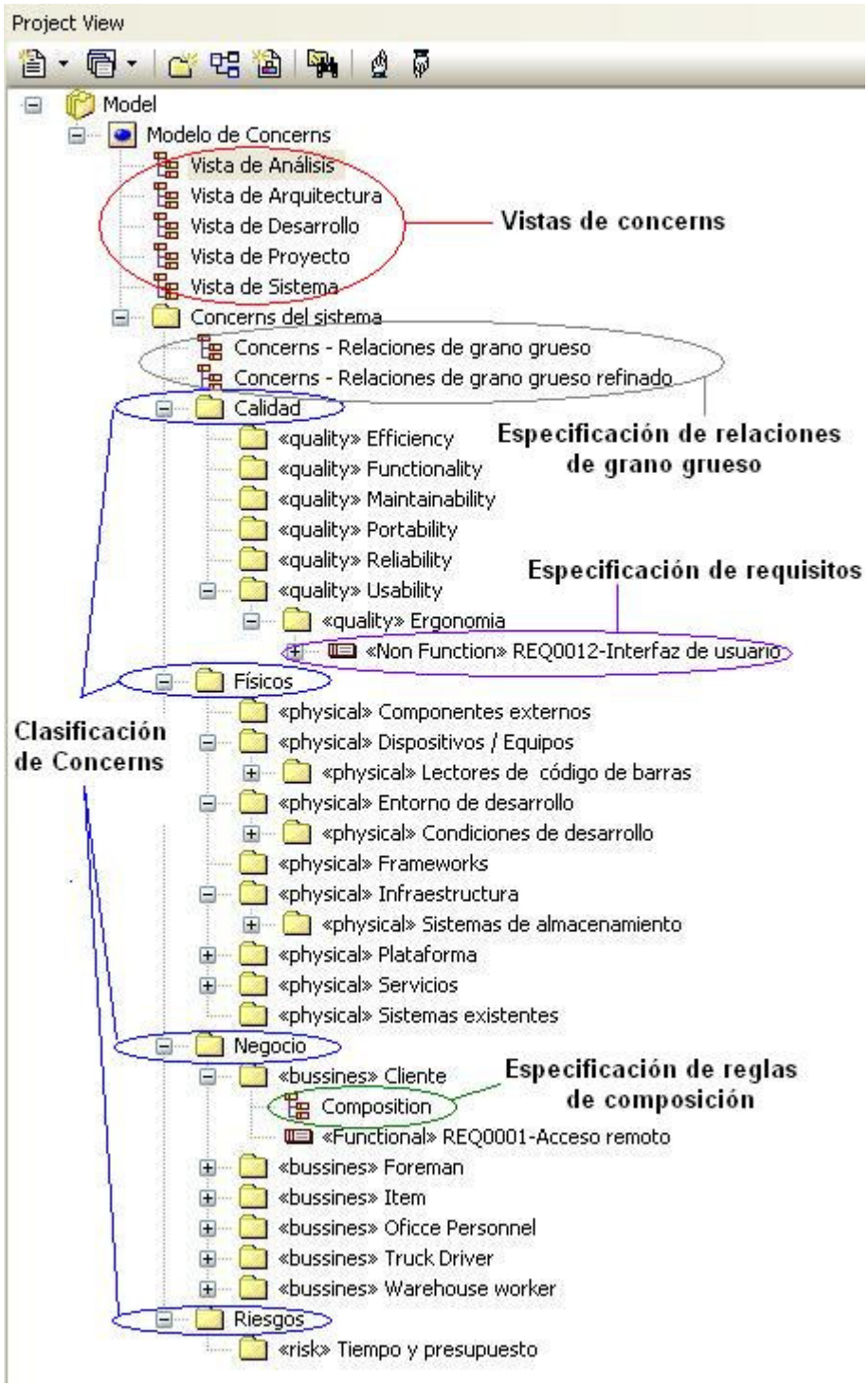

Figura 19. Vista de proyecto para el modelo AORE extendido.

3.2.3 Definición de concerns

La figura 20 presenta el esquema propuesto para el modelado de los concerns. Recordando la definición de AORE de concern: "una colección coherente de requisitos", a partir de esta definición se propone la utilización de "paquetes" en los cuales es posible agrupar el conjunto de requisitos que conforman el concern.

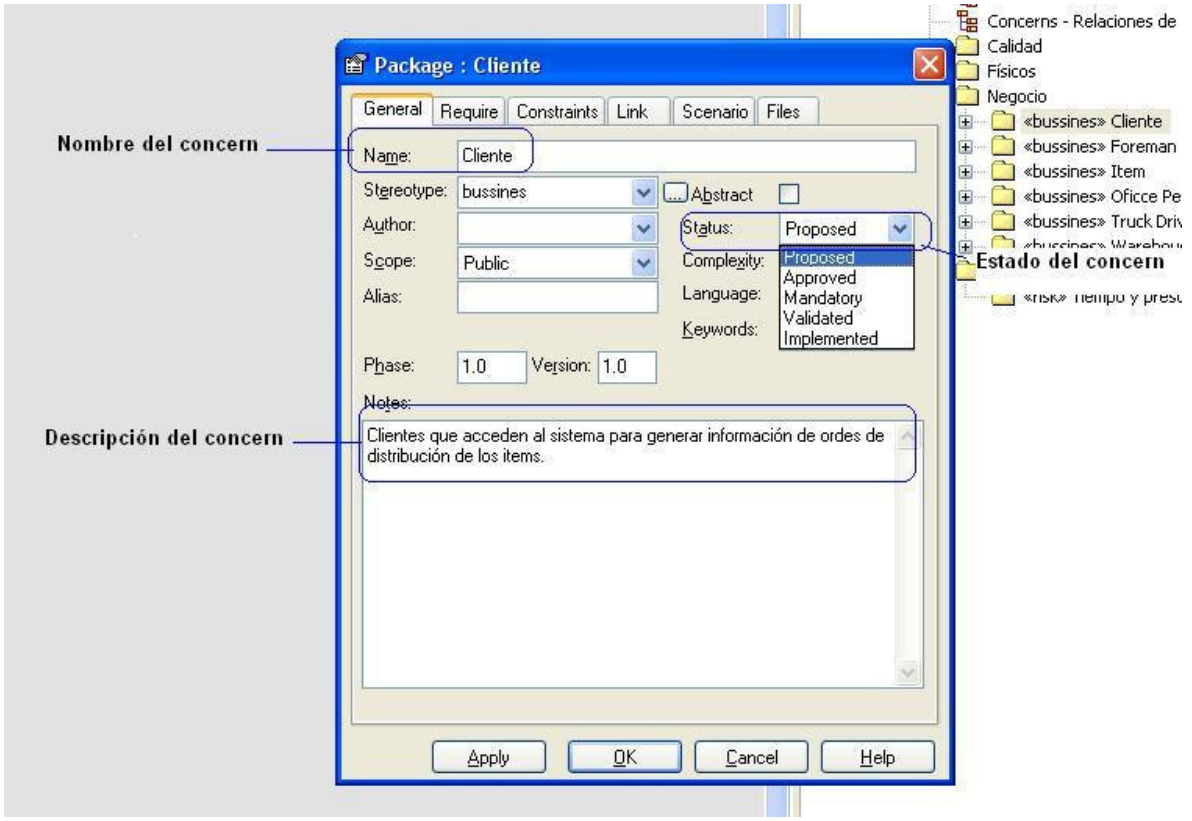

Figura 20. Especificación de concerns

Como se observa, para la definición del concern, es posible establecer el nombre y la descripción del concern. Para clasificar el concern se utiliza el estereotipo, para lo cual se encuentran definidos los estereotipos de acuerdo a la tabla 14. Adicionalmente, es posible controlar el estado en el cual se encuentra un concern e información adicional como autor, versión y otros. También es posible asociar

archivos a un concern, lo cual es útil cuando se requiere establecer relación con archivos externos, tales como leyes, artefactos software y otros.

3.2.4 Definición de requisitos de los concerns

EA es una herramienta diseñada para cubrir todas las fases de la ingeniería de software, por tal motivo el manejo de artefactos para la ingeniería de requisitos se encuentra bien soportado. La figura 21 muestra la forma como la herramienta permite realizar la especificación de los requisitos.

Como se observa, es posible establecer el nombre o descripción corta del requisito - el cual puede estar precedido por un código automático para facilitar la identificación -, la descripción o detalle del requisito, su naturaleza (funcional, no funcional, información, regla de negocio y restricción), el estado del requisito y otra información adicional (autor, versión, prioridad, versión, archivos asociados).

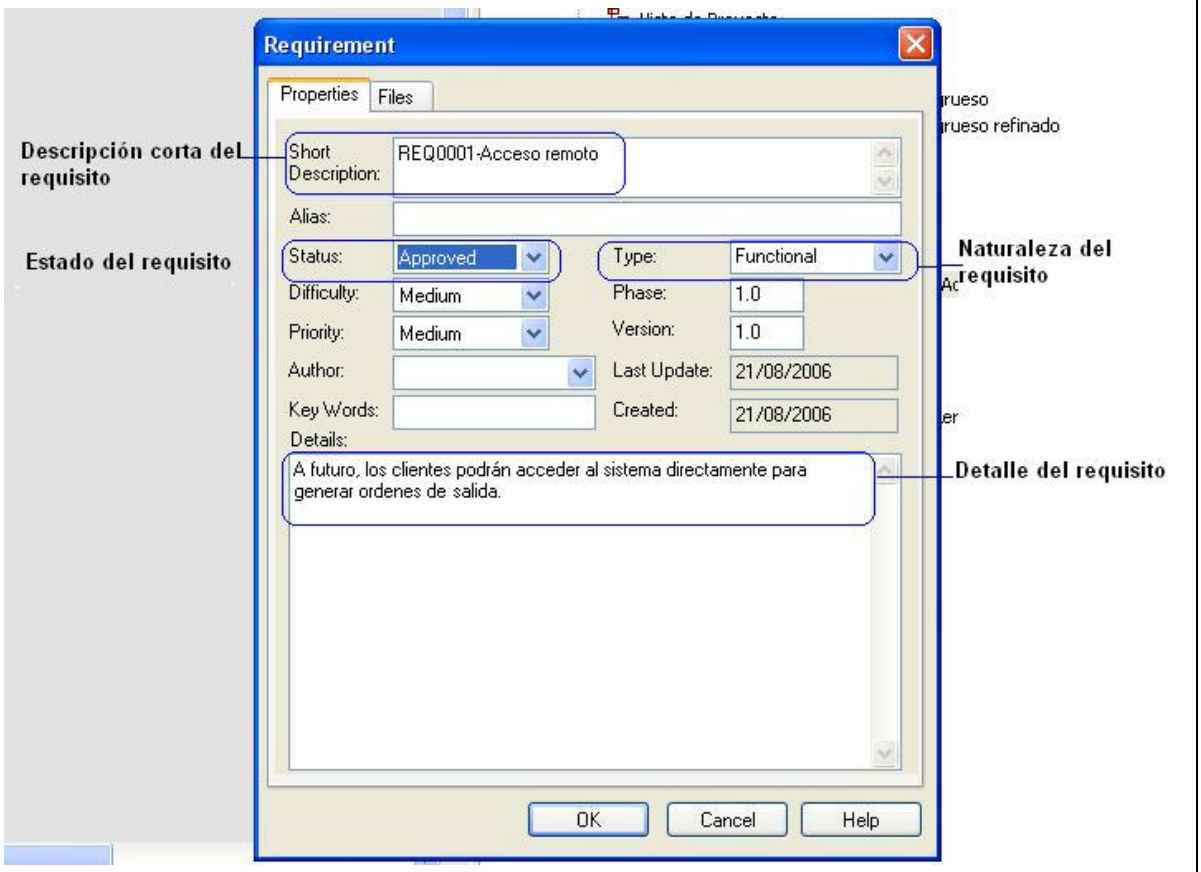

Figura 21. Especificación de los requisitos del concern

3.2.5 Relaciones de grano grueso

Las relaciones de grano grueso son establecidas por medio de diagramas tal como se observa en la figura 22.

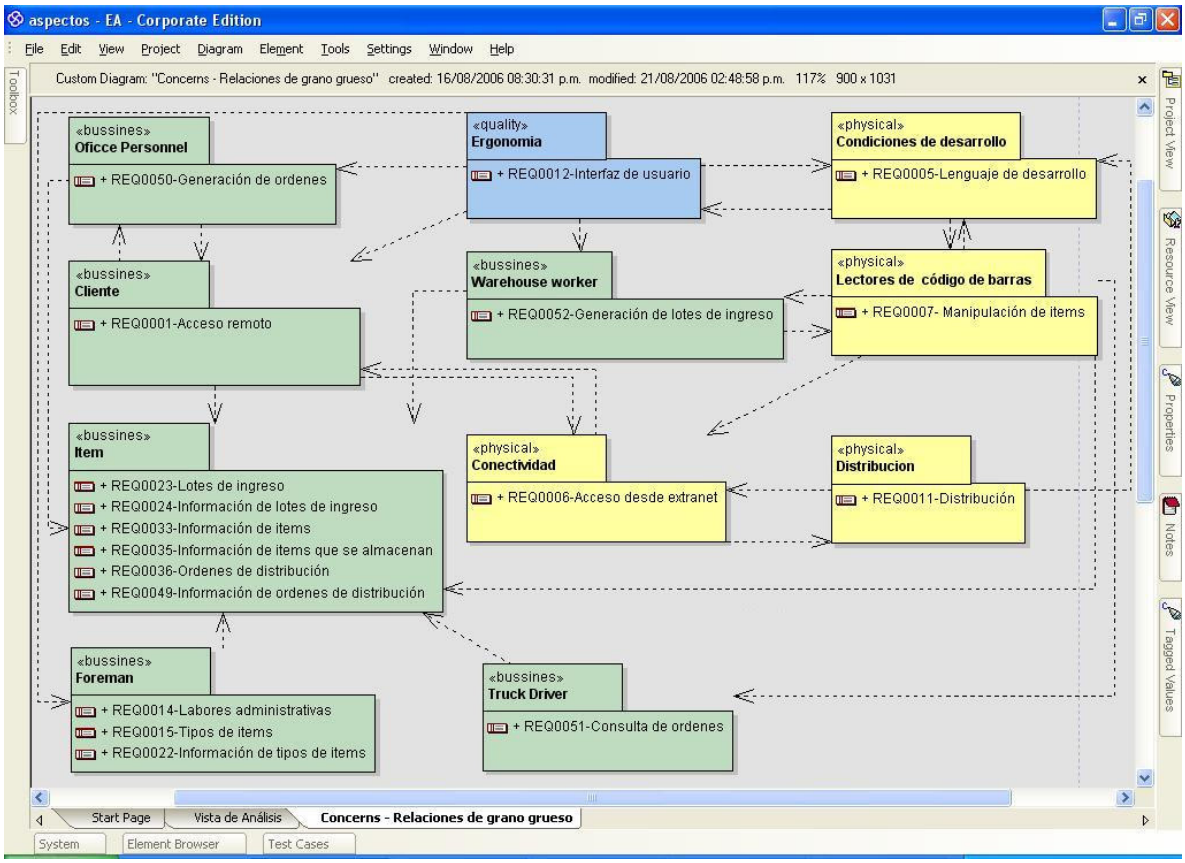

Figura 22. Diagrama de relaciones de grano grueso.

A partir de este diagrama es posible obtener la matriz de Relaciones de Grano Grueso, como se presenta en la figura 23.

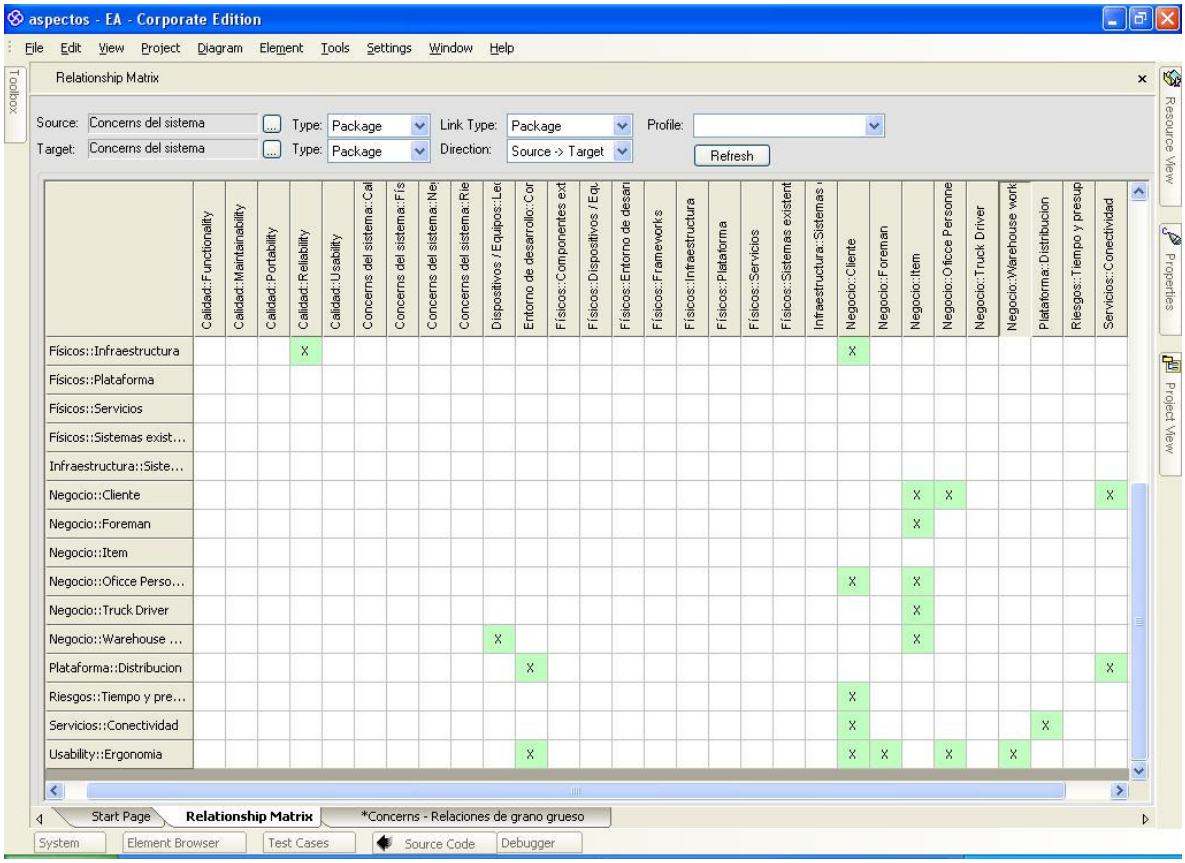

Figura 23. Matriz de relación entre concerns en EA

3.2.6 Relaciones de grano fino

EA ofrece más que un catálogo simple de requisitos, en esta herramienta es posible modelar las relaciones entre los requisitos por medio de diagramas. A partir de estos diagramas es posible definir las reglas de composición de un concern, ya que éstas se materializan a nivel de los requisitos. Los diagramas pueden observarse en la figura 24.

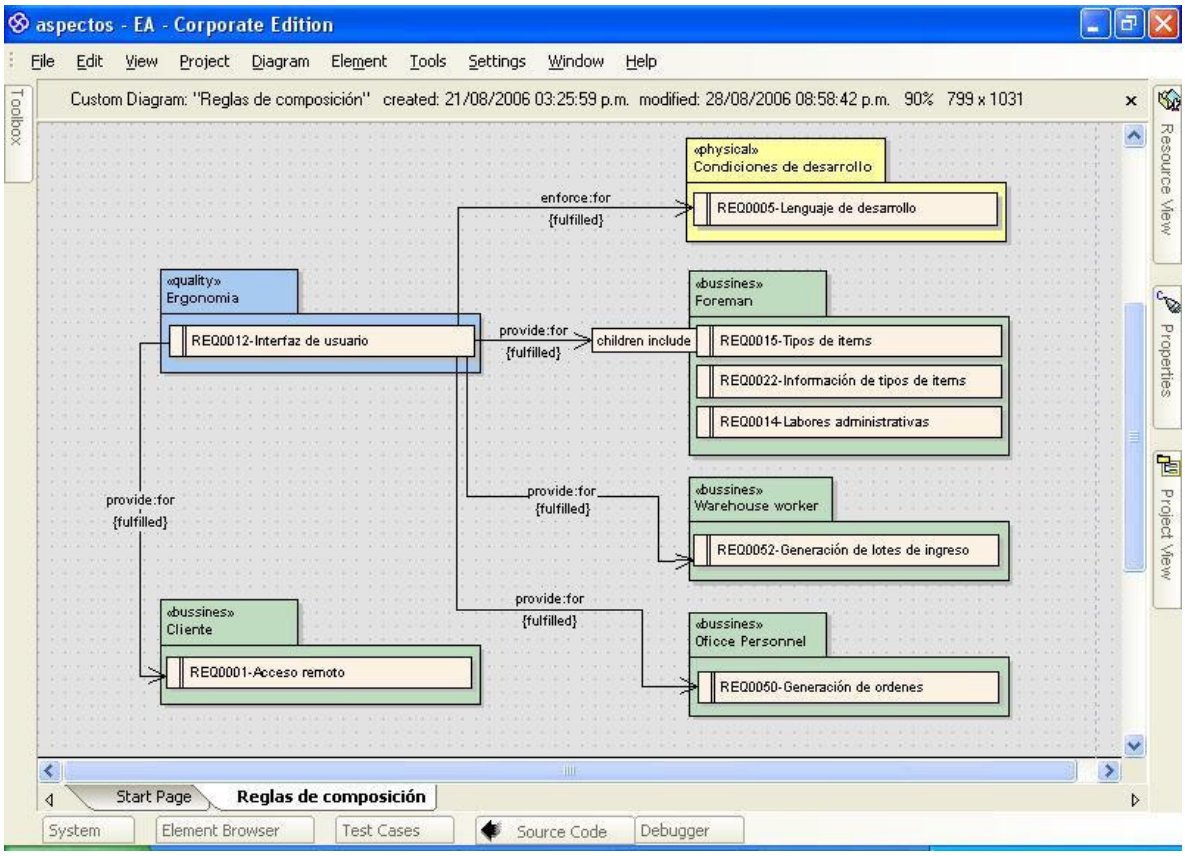

Figura 24. Diagrama de relaciones de grano fino

La especificación de las propiedades de una regla de composición, como establece el modelo AORE, consta de acciones, operadores, resultados y atributos adicionales.

La especificación de la acción y el operador que determinan la relación de grano fino entre los concerns se realiza por medio del nombre de la relación. Dado que es requerido especificar tanto la acción como el operador, se propone que esta sea especificada como *<accion>:<operador>.* La figura 25 presenta la forma como se especifica la composición en EA. Es posible observar que además de la acción y el operador, es posible especificar notas adicionales a la regla de

composición, lo que permite al ingeniero de requisitos documentar otras consideraciones de la relación.

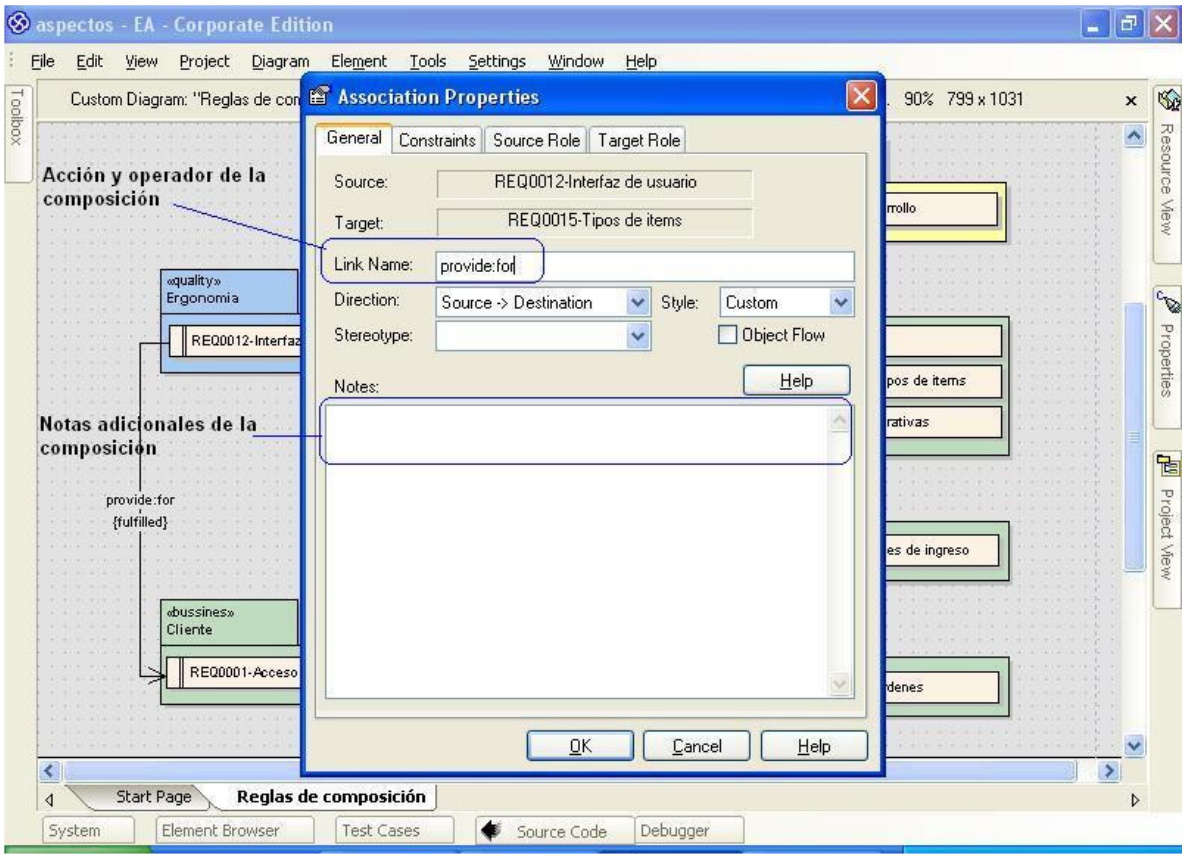

Figura 25. Especificación de acción y operador de la composición.

El resultado de la aplicación de la regla de composición es presentado en EA por medio de post condiciones. La figura 26 muestra la especificación del resultado de la regla de composición en la herramienta.

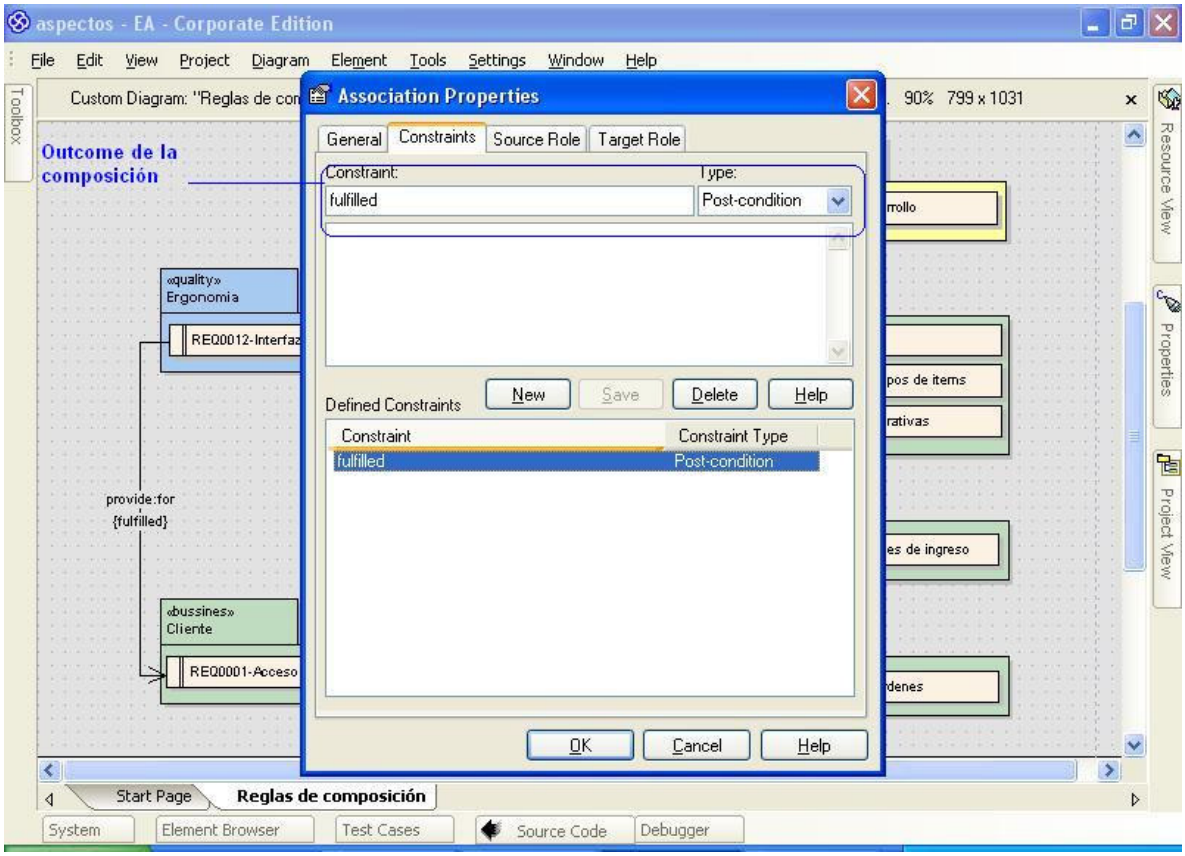

Figura 26. Especificación del resultado de la composición.

Si un requisito de un concern tiene un subconjunto de requisitos, estos deben ser explícitamente incluidos o excluidos de la regla de composición. Para realizar esto, el modelo AORE proporciona el atributo adicional *children,* el cual puede tomar los valores *include o exclude.* Para realizar el modelado de esta situación en EA, se hace uso de los calificadores de la relación, tal como se presenta en la figura 27.

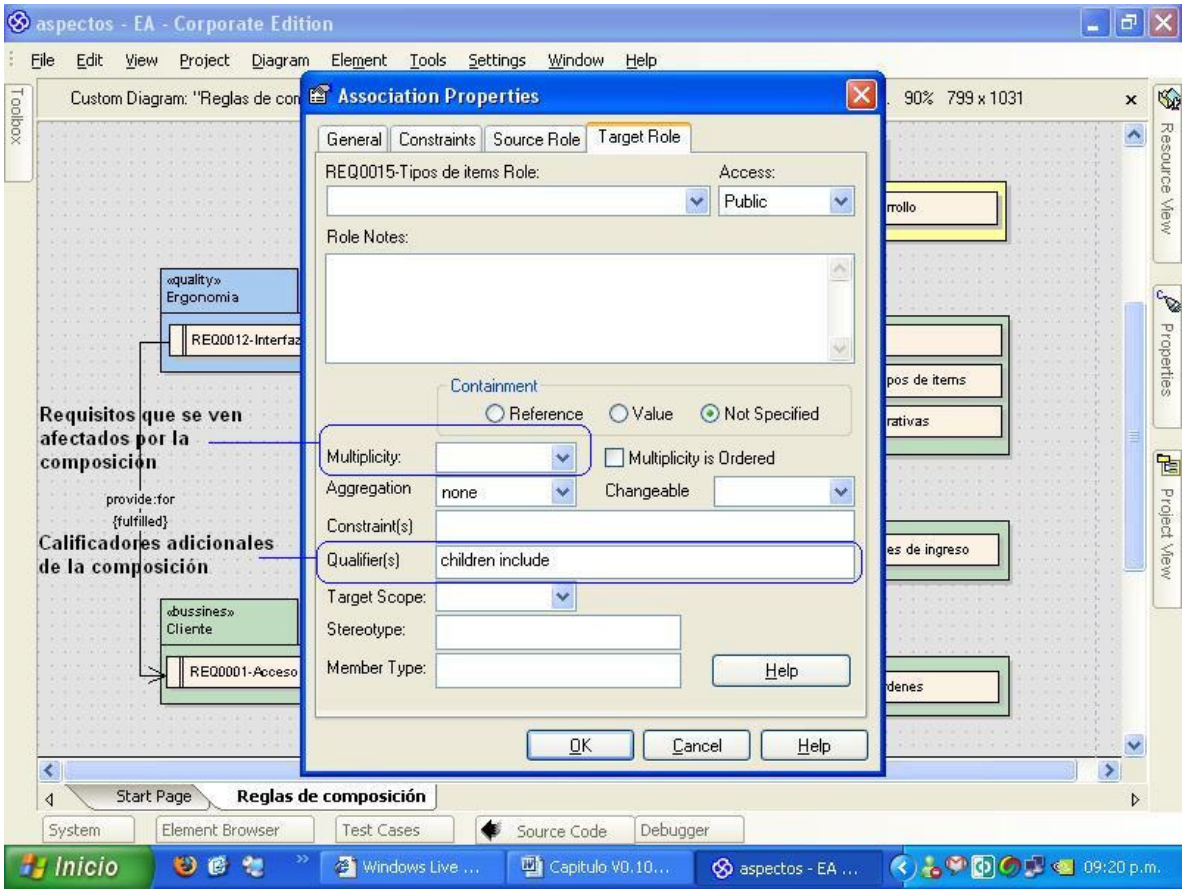

Figura 27. Especificación de los requisitos afectados por el concern.

El modelo AORE, en la especificación de las reglas de composición, propone que en la regla de composición se haga referencia al identificador de los requisitos afectados. Dado que el modelado propuesto en esta sección se realiza por medio de diagramas que relacionan directamente los requisitos, no se hace necesario contar con información explicita del identificador del requisito, sin embargo, en caso de ser requerido, es posible especificar explícitamente el identificador del requisito por medio del uso del campo *multiplicidad.*

3.2.7 Relaciones de grano grueso refinadas

De acuerdo a la propuesta de ampliación del modelo AORE en la sección 2.3, una vez se han definido las relaciones de grano fino entre los concerns, es posible para el ingeniero de requisitos refinar las relaciones de grano grueso. De acuerdo a la tabla 15 se definen como estereotipos, las posibles relaciones de grano grueso a establecer entre los concerns. La figura 28 presenta un diagrama de relaciones de grano grueso refinado.

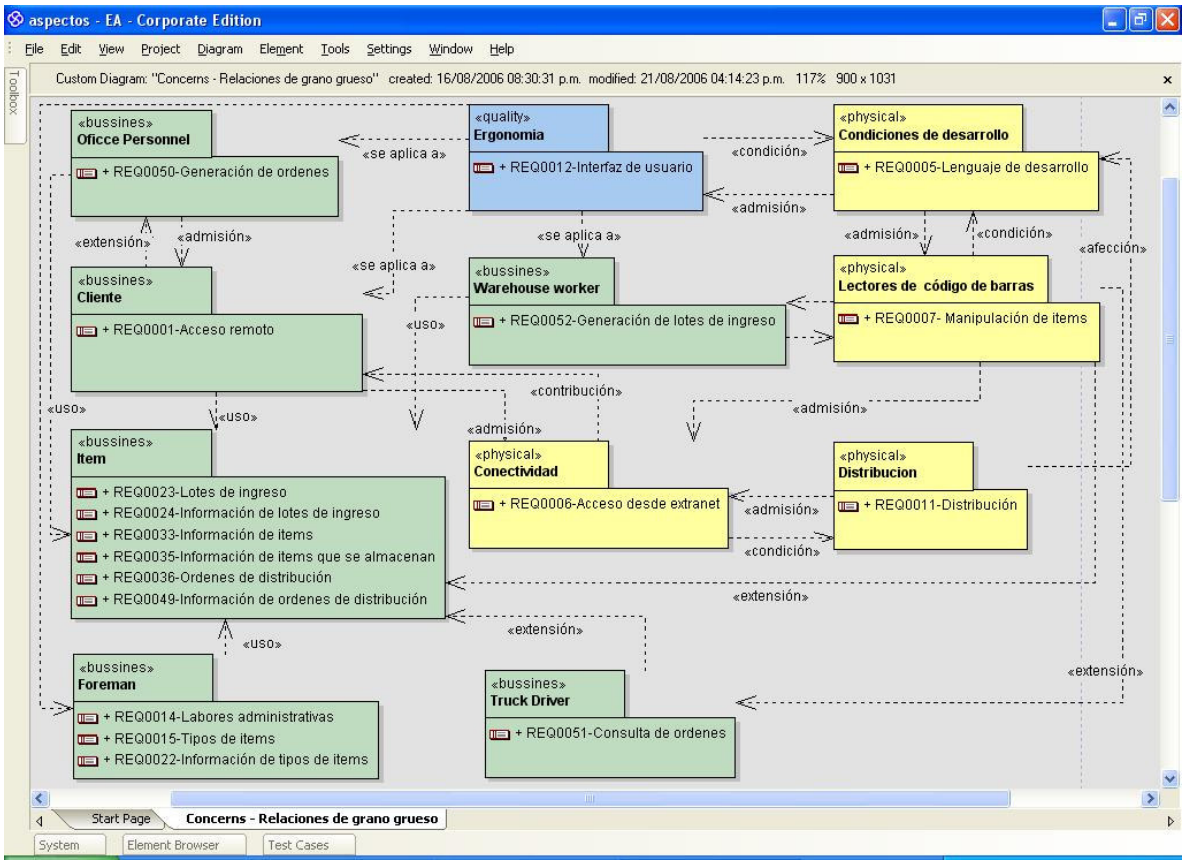

Figura 28. Diagrama de relaciones de grano grueso refinado.

3.2.8 Manejo de conflictos

El manejo de conflictos hace de AORE una de las metodologías más completas para el manejo de aspectos a nivel de ingeniería de requisitos, sin embargo no contar con una herramienta especializada que cubra todos los artefactos y procesos propuestos por dicha metodología dificulta su aplicación.

EA proporciona muchos elementos que permite extender su uso a la metodología AORE, sin embargo no puede pretenderse que de un soporte completo para ésta. El manejo de conflictos de acuerdo a la metodología se presenta como una de las dificultades, ya que no es posible encontrar los artefactos que permita agilizar este proceso.

AORE propone el manejo de los conflictos por medio de matrices de contribución como se presenta en la figura 13, sin embargo EA no brinda los mecanismos para obtener esta matriz y guiar la resolución de conflictos.

Pese a esta falencia, es posible utilizar EA para apoyar y seguir el proceso de resolución de conflictos. La figura 29 muestra la forma como EA permite definir conflictos, los cuales pueden ser trazados a los concerns afectados y definir información relevante al conflicto.

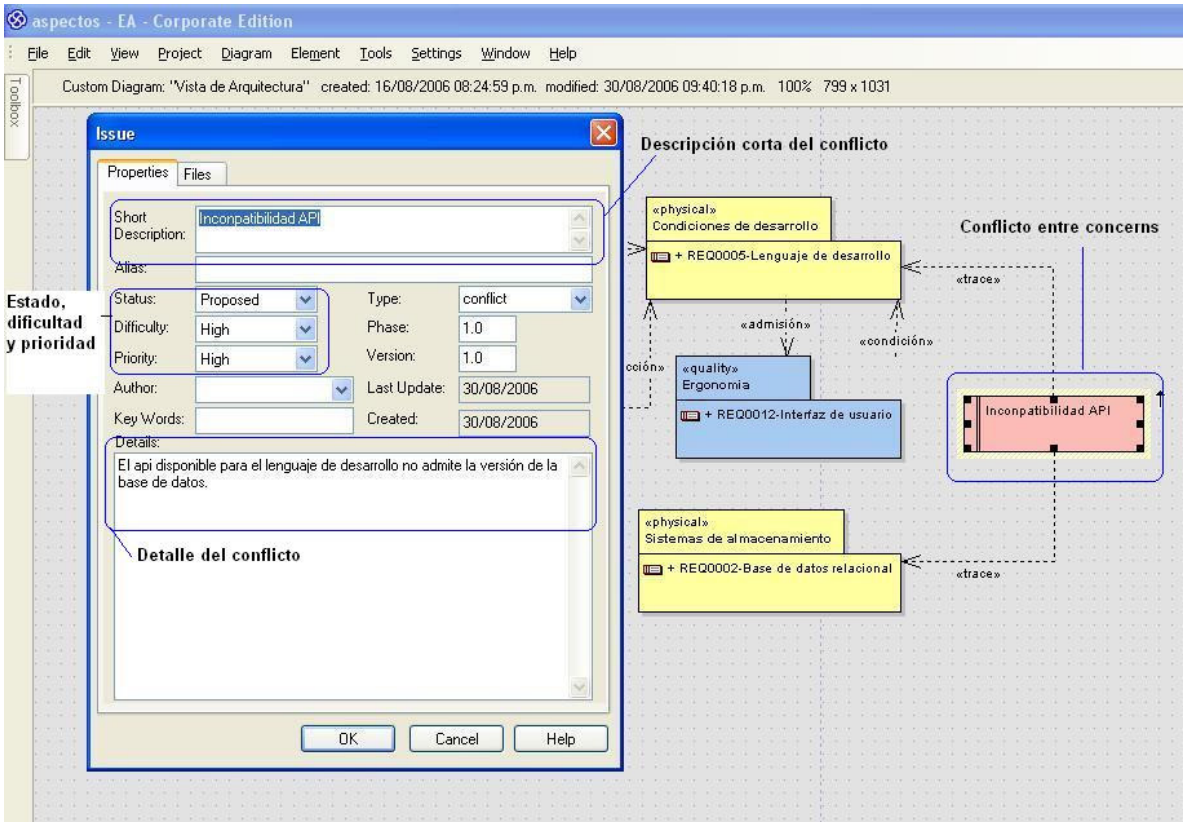

Figura 29. Manejo de conflictos en EA.

3.2.9 Vistas de concerns

Una vez se han refinado los concerns y se han resuelto los conflictos encontrados, es posible clasificar los concerns en vistas, las cuales, de acuerdo a lo propuesto en la sección 2.3, tienen por objeto permitir a los diferentes stakeholders obtener visiones del sistema desde diferentes perspectivas. La figura 30 presenta la vista de plataforma como ejemplo de la definición de las vistas del sistema. Idealmente se esperaría que estas vistas puedan ser generadas de manera automática por una herramienta CASE de acuerdo a una configuración dada por el usuario de la herramienta (desarrollador) con respecto a los tipos de concerns y tipos de relaciones que serían relevantes para una vista particular.

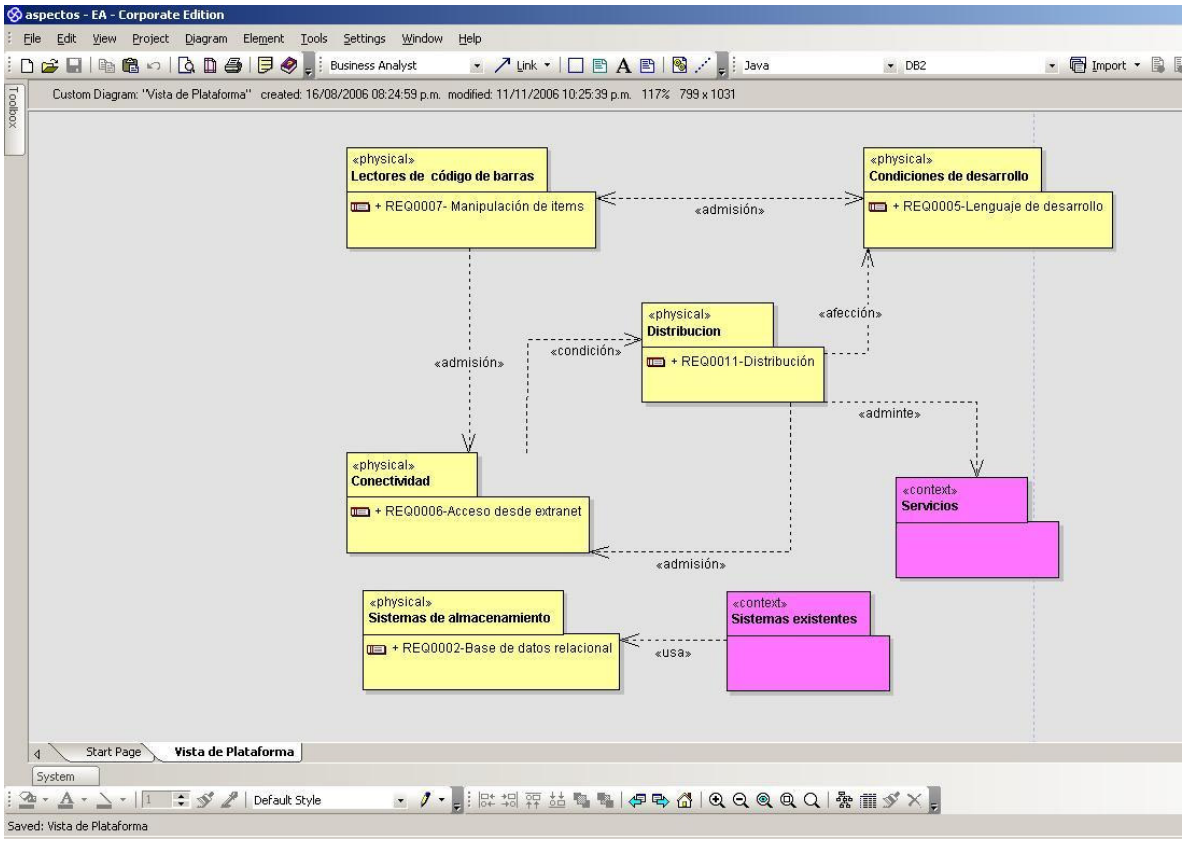

Figura 30. Vista de concerns orientada a la plataforma.

3.3 Observaciones y consideraciones

En la realización de la propuesta de modelado de los elementos de AORE que se presentó en este capítulo existen diversas observaciones y consideraciones que vale la pena considerar y que son presentadas a continuación.

- Enterprise Architect es una herramienta diseñada para soportar todas las fases del ciclo de vida del software bajo el paradigma orientado a objetos. Pretender extender su uso para soportar el paradigma orientado a aspectos resulta un tanto ambicioso, pero dadas las características y la flexibilidad de la herramienta, es

posible *adaptar* muchos de los artefactos y modelos para dar un soporte adecuado para el modelo AORE.

- Las propuestas de modelado que se han presentado en este capítulo han sido realizadas de acuerdo a la experiencia que en las labores del día a día hemos tenido como ingenieros de requisitos, por tanto creemos que el esquema de modelado propuesto ayudará a aplicar AORE en casos reales. Es importante tener en cuenta que AORE propone la especificación de los elementos del modelo por medio de XML, lo cual brinda la posibilidad de extender su uso a futuras herramientas, sin embargo generar dichos archivos sin la ayuda de una herramienta resulta poco práctico y podría adicionar complejidad a la aplicación del modelo.

- Como se mencionaba en secciones anteriores, contar con una herramienta de modelado que permita visualizar y manipular fácilmente los diferentes elementos del modelo facilita y agiliza la aplicación del modelo.

4. APLICACIÓN DEL MODELO

En la sección 2.3 del presente trabajo se mostró un modelo de ingeniería de requisitos con un enfoque aspectual, este modelo se basa en los modelos AORE, COSMOS y otros elementos adicionales para exponer un modelo AORE extendido.

El presente capítulo expone los resultados y conclusiones obtenidos de aplicar los elementos del modelo AORE extendido a un sistema ya desarrollado, en la industria del software específicamente en AVANSOFT S.A., bajo un enfoque objetual.

4.1 Presentación del caso de aplicación

GESCOM surge como un software cuyo objetivo es permitir a las empresas gestionar el talento del recurso humano que las componen y a los aspirantes a ingresar, basándose en el modelo por competencias. El modelo por compentencias usado en GESCOM es el creado por la empresa ALTA GESTIÓN EMPRESARIAL.

El software ofrece la posibilidad de generar gran cantidad de reportes con resultados cuantitativos, cuyo análisis posterior permita a las empresas ofrecer un

desarrollo integral a sus empleados y que estos aporten al cumplimiento de los objetivos organizacionales.

El documento de requisitos, anexo A, que surge del proceso de desarrollo del sistema GESCOM, es usado como insumo para la aplicación del modelo AORE extendido, el cual requiere de un catálogo de requisitos para la primera actividad del modelo cuyo objetivo es la identificación y especificación de concerns.

En la sección, 4.2, se expondrá paso a paso el proceso que se llevó a cabo para la aplicación del modelo y los resultados y conclusiones obtenidas.

Como se expuso en el capitulo 3, Enterprise Architech será usada como herramienta de soporte para el modelado del caso de aplicación.
Los diagramas resultantes de la aplicación del modelo, algunos expuestos como imágenes en la sección 4.2, y demás diagramas realizados, será posible consultarlos en detalle en el anexo B.

4.2 Aplicación del modelo

En esta sección se presenta la aplicación del modelo AORE ampliado al caso de estudio mostrado en la sección anterior.

4.2.1 Identificación de concerns

Como ya es conocido en el modelo, la primera actividad consiste en la identificación de los concerns que conforman el sistema. Bajo la definición de AORE, los concerns son un conjunto coherente de requisitos, por lo que esta tarea parte del anexo A, el cual es el catálogo de requisitos del sistema GESCOM. Los concens identificados bajo la definición de AORE son:

- Ergonomía.
- Restricciones técnicas.
- Componentes externos / frameworks.
- Herramientas de apoyo.
- Herramientas de desarrollo.
- Estaciones de desarrollo.
- Servidores.
- Infraestructura / plataforma.
- Cargos.
- Competencias.
- Configuración del sistema.
- Empresas.
- Evaluaciones.
- Experiencia.
- Necesidades de formación.
- Personas.
- Plan de desarrollo.

Otros concerns encontrados a partir de la propuesta de ampliación del modelo son:

- Configuración inicial de las empresas (Objetivo del sistema)
- Mostrar indicadores de forma gráfica (Objetivo del sistema)
- Permitir la configuración de la seguridad por roles (Objetivo del sistema)
- Permitir la evaluación por diferentes evaluadores (Objetivo del sistema)
- Permitir mostrar los resultados de las evaluaciones (Objetivo del sistema)
- Presentar informes imprimibles (Objetivo del sistema)
- Presentar informes técnicos (Objetivo del sistema)
- Disponibilidad de recursos (riesgo)
- Especificación (riesgo)
- Imprevistos técnicos (riesgo)
- Inexperiencia (riesgo)

- Tiempos y costos (riesgo)

Estos concerns se encuentran modelados Enterprise Architect, de acuerdo a lo propuesto en el capitulo 3.

4.2.2 Clasificar los concerns.

La clasificación de los concerns obtenida del caso de aplicación se presenta en la tabla 17.

Clasificación	Concerns
Calidad	Ergonomía
Contexto	Restricciones técnicas
Físico	Componentes externos / frameworks Herramientas de apoyo Herramientas de desarrollo Estaciones de desarrollo Servidores Infraestructura / plataforma
Negocio	Cargos Competencias Configuración del sistema Empresas Evaluaciones Experiencia Necesidades de formación Personas Plan de desarrollo
Objetivo	Configuración inicial de las empresas Mostrar indicadores de forma gráfica Permitir la configuración de la seguridad por roles Permitir la evaluación por diferentes evaluadores Permitir mostrar los resultados de las evaluaciones

	Presentar informes imprimibles Presentar informes técnicos
Riesgo	Disponibilidad de recursos Especificación Imprevistos técnicos Inexperiencia Tiempos y costos

Tabla 17. Clasificación de los concerns del sistema

A partir de estas clasificaciones es posible crear las vistas de concerns.

4.2.3 Identificación de relaciones de grano grueso

Las relaciones de grano grueso son establecidas de acuerdo a la especificación del sistema y modeladas en Enterprise Architect. La figura 31 presenta el diagrama de relaciones de grano grueso para los concerns del sistema.

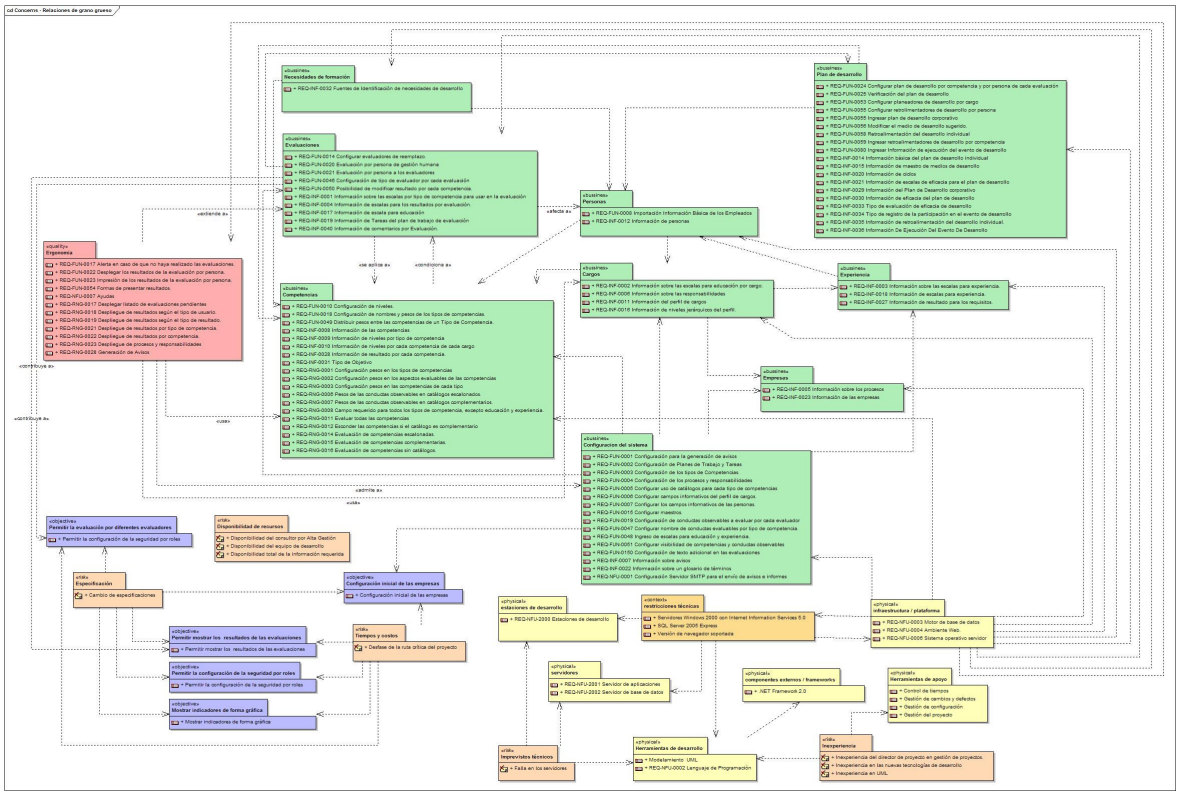

Figura 31. Relaciones de concerns de grano grueso.

El modelo AORE propone que las relaciones de grano grueso sean especificados por medio de una matriz, la cual se presenta en la tabla 18.

	«quality» Ergonomía	«context» restricciones técnicas	«physical» componentes externos / frameworks	«physical» Herramientas de apoyo	«physical» Herramientas de desarrollo	«physical» estaciones de desarrollo	«physical» servidores	«physical» infraestructura / plataforma	«bussines» Cargos	«bussines» Competencias	«bussines» Configuración del sistema	«bussines» Empresas	«bussines» Evaluaciones	«bussines» Experiencia	«bussines» Necesidades de formación	«bussines» Personas	«bussines» Plan de desarrollo	«objective» Configuración inicial de las empresas	«objective» Mostrar indicadores de forma gráfica	«objective» Permitir la configuración de la seguridad por roles	«objective» Permitir la evaluación por diferentes evaluadores	«objective» Permitir mostrar los resultados de las evaluaciones	«objective» Presentar informes imprimibles	«objective» Presentar informes técnicos	«risk» Disponibilidad de recursos	«risk» Especificación	«risk» Imprevistos técnicos	«risk» Inexperiencia	«risk» Tiempos y costos
«quality» Ergonomía									X	X	X		X																
«context» restricciones técnicas						X		X																					
«physical» componentes externos / frameworks																													
«physical» Herramientas de apoyo																													
«physical» Herramientas de desarrollo																													
«physical» estaciones de desarrollo																													
«physical» servidores																													
«physical» infraestructura / plataforma	X	X							X	X	X	X	X	X	X	X	X												
«bussines» Cargos										X		X		X															
«bussines» Competencias													X			X													
«bussines» Configuración del sistema									X			X	X	X															
«bussines» Empresas																													

«bussines» Evaluaciones										X						X	X				X	X															
«bussines» Experiencia																X																					
«bussines» Necesidades de formación										X						X																					
«bussines» Personas									X																												
«bussines» Plan de desarrollo																X																					
«objective» Configuración inicial de las empresas																																					
«objective» Mostrar indicadores de forma gráfica																																					
«objective» Permitir la configuración de la seguridad por roles																																					
«objective» Permitir la evaluación por diferentes evaluadores																																					
«objective» Permitir mostrar los resultados de las evaluaciones																																					
«objective» Presentar informes imprimibles																																					
«objective» Presentar informes técnicos																																					
«risk» Disponibilidad de recursos																																					
«risk» Especificación																		X	X	X		X															
«risk» Imprevistos técnicos					X	X	X																														
«risk» Inexperiencia				X	X																																
«risk» Tiempos y costos																		X	X	X	X	X															

Tabla 18. Matriz de relaciones entre concerns

4.2.4 Identificación de relaciones de grano fino

La identificación de las relaciones de grano fino implica el análisis de las relaciones de los requisitos de un concerns contra todos los requisitos de otros concerns. Esta es una tarea compleja y dispendiosa, lo que nos ha llevado a acortar esta fase para analizar sólo las relaciones de grano fino de los concerns Ergonomía, Evaluaciones y Competencias. La selección de estos concerns se realizó a partir de la observación de las potenciales relaciones de grano fino que se obtendrían.

El establecimiento de las relaciones de grano fino permitió observar algunas características, que si bien no es conveniente establecer como heurísticas, provee una guía de las posibles relaciones que pueden presentarse entre los requisitos de los concerns de acuerdo a su tipo. Estas observaciones son:

- Se presentan relaciones de grano fino entre requisitos funcionales y reglas de negocio con los requisitos de información, sin embargo, no se considera viable el establecimiento de relaciones entre estos concerns en sentido contrario.

- Las relaciones de grano fino entre requisitos funcionales y reglas de negocio con requisitos de información pueden tener acción *affect* con operador *on*, si la información plasmada en el requisito de información se ve alterada con la aplicación de la funcionalidad o regla de negocio. Si la información no se ve afectada, la relación tiene acción *ensure* con operador *on*

- Las relaciones que se observan entre una regla de negocio con un requisito funcional pueden tener acciones *applied, enforce o provide.*

El establecimiento de estas heurísticas está por fuera del alcance de este trabajo, por lo que no se profundizará más al respecto y queda abierto el tema a trabajos futuros.

Las figuras 32, 33 y 34 presentan las relaciones de grano fino obtenidas en la aplicación del modelo. Para realizar una mejor observación de esta aplicación se debe remitir al anexo B.

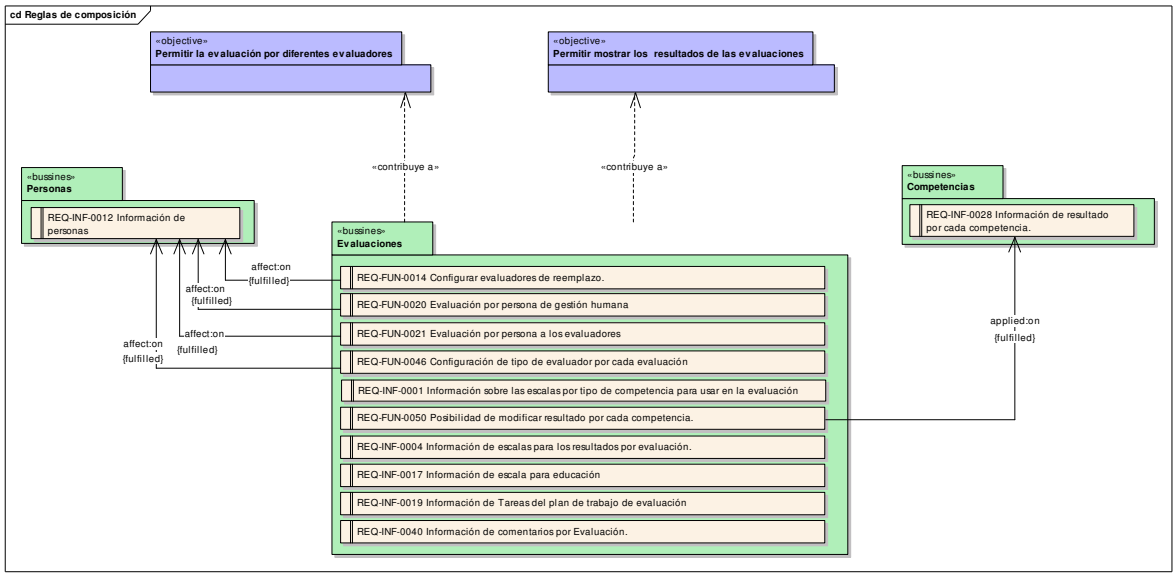

Figura 32. Especificación de relaciones de grano fino para el concern Evaluaciones.

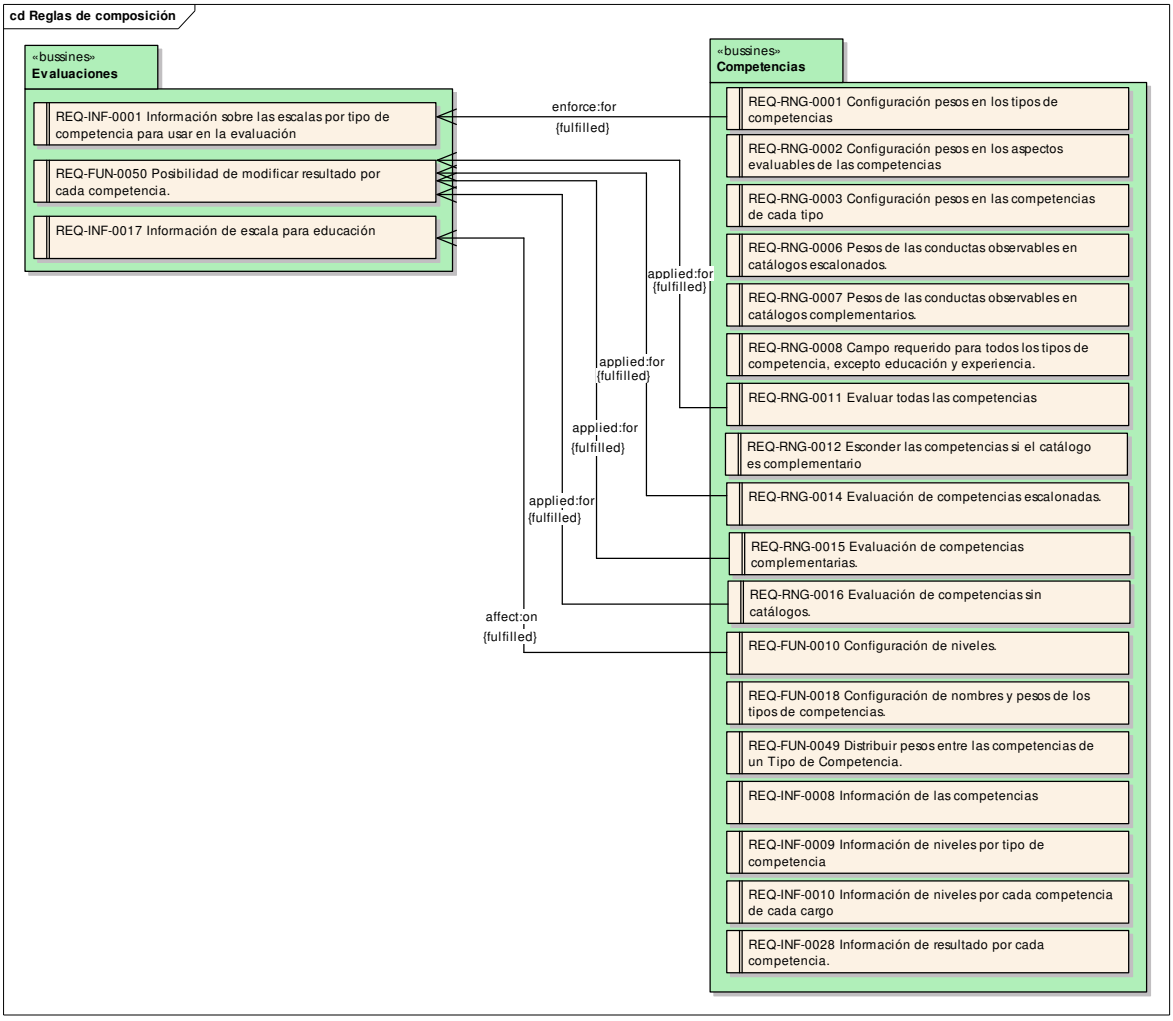

Figura 33. Especificación de relaciones de grano fino para el concern Competencias

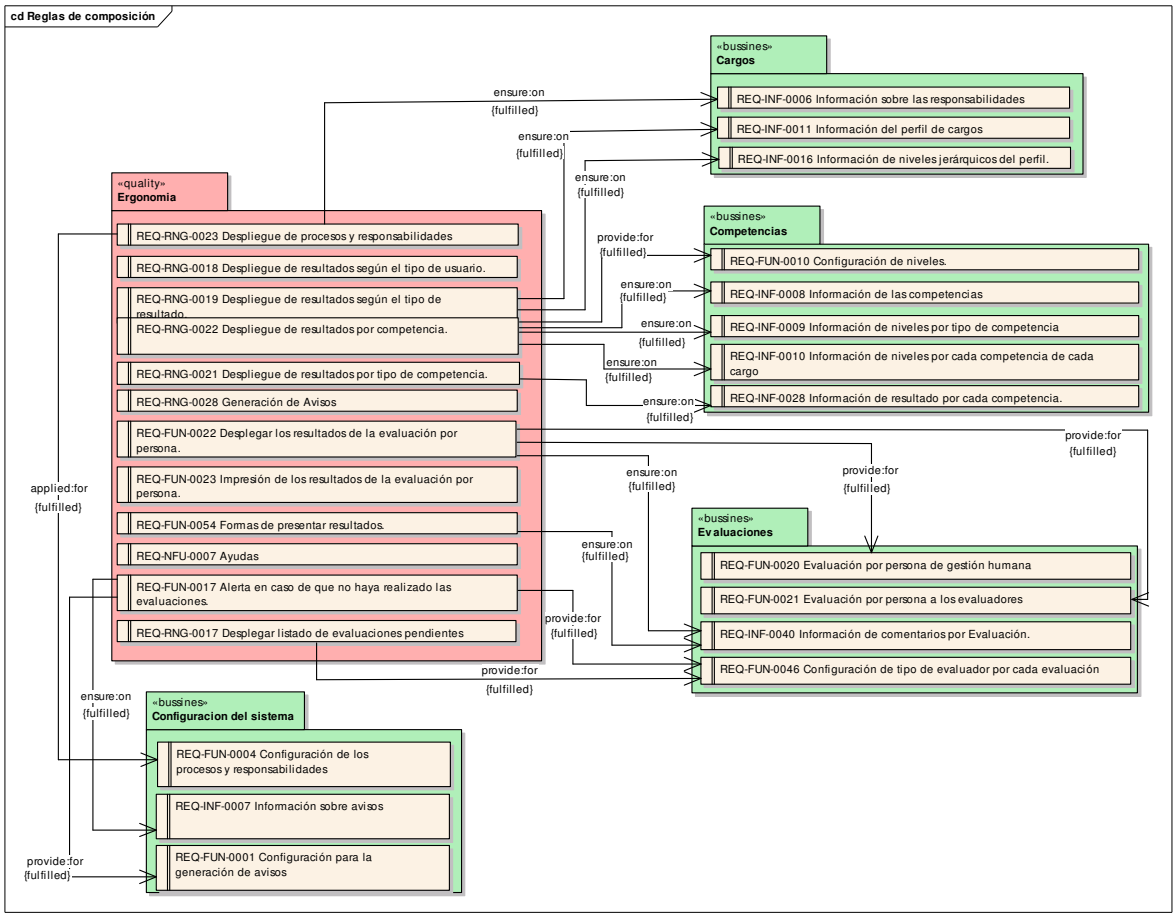

Figura 34. Relaciones de grano fino para el concern Ergonomía.

4.2.5 Refinamiento de relaciones de grano grueso

Una vez se han especificado las relaciones de grano fino, es posible realizar el refinamiento de las relaciones de grano grueso según lo propuesto en la sección 2.3.2.3.

Al refinar las relaciones de grano grueso a partir de las relaciones de grano fino, se puede observar que algunas de las relaciones desaparecen ya que no se encuentran relacionados los requisitos que conforman ambos concerns, tal como

es el caso de la relación inicialmente establecida entre los concerns *Evaluaciones y Plan de desarrollo.* También desaparece la relación entre los concerns Competencias y *Personas*.

La figura 35 presenta las relaciones de grano grueso refinadas para los concerns del sistema. Dado que no se realizó el análisis completo de las relaciones de grano fino, algunas relaciones de grano grueso no se presentan refinadas.

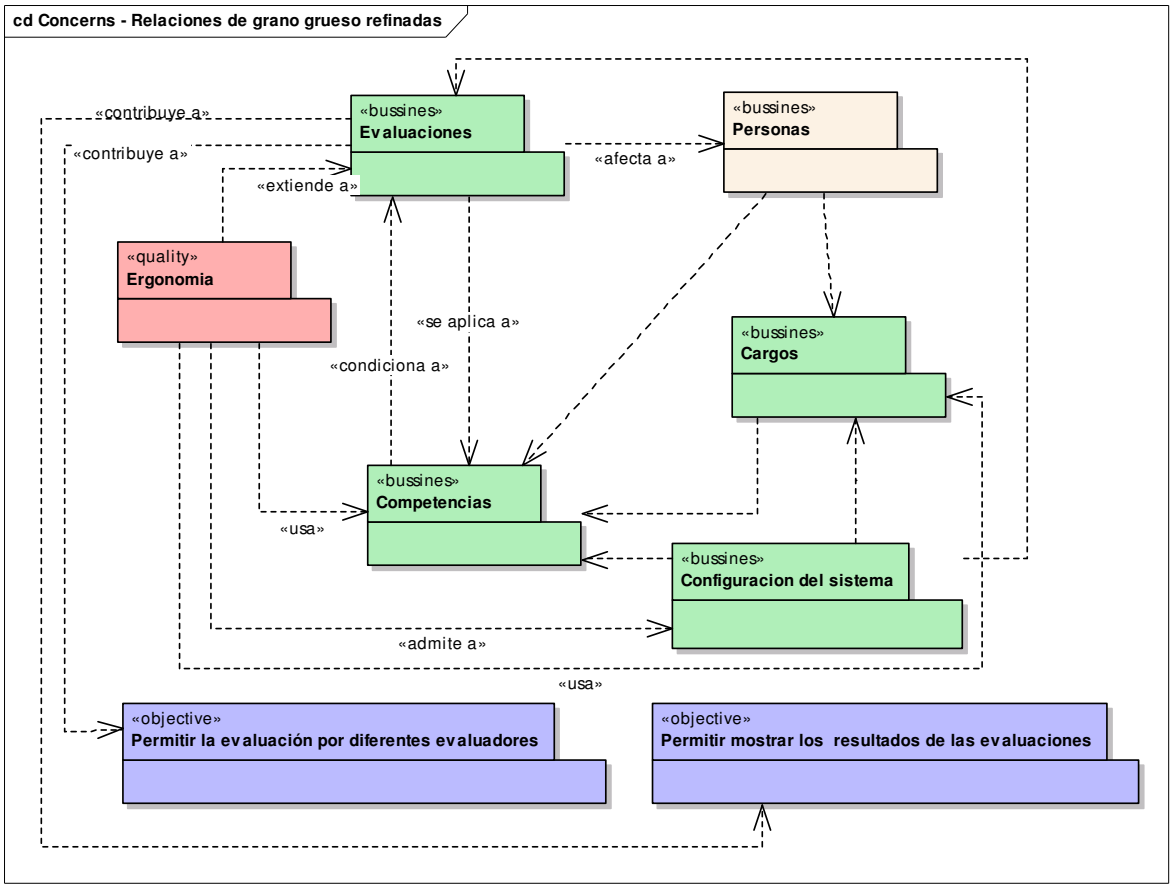

Figura 35. Relaciones de grano grueso refinadas.

Es importante anotar que para la selección de la relación de grano grueso se deben tener en cuenta las diferentes relaciones de grano fino que se presentan. Si bien no se presentan heurísticas que permitan identificar relaciones dominantes

que determinen el tipo de relación de grano grueso, las relaciones entre los requisitos funcionales (requisito funcional con requisito funcional) tienen una mayor relevancia que las relaciones entre reglas de negocio (requisito funcional con regla de negocio) y requisitos de información (requisito funcional con requisito de información o regla de negocio con requisito de información).

4.2.6 Solución de conflictos.

En la Tabla 19 se presenta la matriz de contribución de aplicación del modelo AORE extendido, en esta sólo se resalta la contribución negativa o positiva para los concerns que fueron analizados en la aplicación del modelo (ergonomía, evaluaciones y competencias), las demás contribuciones permanecerán marcadas con el símbolo **O** pero no serán analizadas en este trabajo.

	«quality» Ergonomía	«context» restricciones técnicas	«physical» componentes externos / frameworks	«physical» Herramientas de apoyo	«physical» Herramientas de desarrollo	«physical» estaciones de desarrollo	«physical» servidores	«physical» infraestructura / plataforma	«bussines» Cargos	«bussines» Competencias	«bussines» Configuración del sistema	«bussines» Empresas	«bussines» Evaluaciones	«bussines» Experiencia	«bussines» Necesidades de formación	«bussines» Personas	«bussines» Plan de desarrollo	«objective» Configuración inicial de las empresas	«objective» Mostrar indicadores de forma gráfica	«objective» Permitir la configuración de la seguridad por roles	«objective» Permitir la evaluación por diferentes evaluadores	«objective» Permitir mostrar los resultados de las evaluaciones	«objective» Presentar informes imprimibles	«objective» Presentar informes técnicos	«risk» Disponibilidad de recursos	«risk» Especificación	«risk» Imprevistos técnicos	«risk» Inexperiencia	«risk» Tiempos y costos
«quality» Ergonomía									+	+	+		+																
«context» restricciones técnicas						o		o																					
«physical» componentes externos / frameworks																													
«physical» Herramientas de apoyo																													
«physical» Herramientas de desarrollo																													
«physical» estaciones de desarrollo																													
«physical» servidores																													
«physical» infraestructura / plataforma	o	o							o	o	o	o	o	o	o	o	o												
«bussines» Cargos										o		o		o															
«bussines» Competencias													+																
«bussines» Configuración del sistema									o			o	o	o															

«bussines» Empresas																														
«bussines» Evaluaciones										X						X					X	X								
«bussines» Experiencia																X														
«bussines» Necesidades de formación										X						X														
«bussines» Personas									X																					
«bussines» Plan de desarrollo																X														
«objective» Configuración inicial de las empresas																														
«objective» Mostrar indicadores de forma gráfica																														
«objective» Permitir la configuración de la seguridad por roles																														
«objective» Permitir la evaluación por diferentes evaluadores																														
«objective» Permitir mostrar los resultados de las evaluaciones																														
«objective» Presentar informes imprimibles																														
«objective» Presentar informes técnicos																														
«risk» Disponibilidad de recursos																														
«risk» Especificación																		o	o	o		o								
«risk» Imprevistos técnicos					o	o	o																							
«risk» Inexperiencia				o	o																									
«risk» Tiempos y costos																		o	o	o	o	o								

Tabla 19. Matriz de contribución.

Como indica el modelo AORE, una vez elaborada la matriz de contribución, se realiza el proceso de *folding* y se obtienen las contribuciones acumuladas de los concerns, tal como se muestra en la Tabla 20.

	«quality» Ergonomía	«context» restricciones técnicas	«physical» componentes externos / frameworks	«physical» Herramientas de apoyo	«physical» Herramientas de desarrollo	«physical» estaciones de desarrollo	«physical» servidores	«physical» infraestructura / plataforma	«bussines» Cargos	«bussines» Competencias	«bussines» Configuración del sistema
«quality» Ergonomía									+	+	+
«context» restricciones técnicas											
«physical» componentes externos / frameworks											
«physical» Herramientas de apoyo											
«physical» Herramientas de desarrollo											
«physical» estaciones de desarrollo		o									
«physical» servidores											
«physical» infraestructura / plataforma		o									
«bussines» Cargos	+									o	o
«bussines» Competencias	+								o		o
«bussines» Configuración del sistema	+								o	o	
«bussines» Competencias											
«bussines» Configuración del sistema											
«bussines» Empresas											
«bussines» Evaluaciones											
«bussines» Experiencia											
«bussines» Necesidades de formación											
«bussines» Personas											
«bussines» Plan de desarrollo											
«objective» Configuración inicial de las empresas											
«objective» Mostrar indicadores de forma gráfica											
«objective» Permitir la configuración de la seguridad por roles											
«objective» Permitir la evaluación por diferentes evaluadores											
«objective» Permitir mostrar los resultados de las evaluaciones											
«objective» Presentar informes imprimibles											
«objective» Presentar informes técnicos											
«risk» Disponibilidad de recursos											
«risk» Especificación											
«risk» Imprevistos técnicos											
«risk» Inexperiencia											
«risk» Tiempos y costos											

«bussines» Evaluaciones	+							o		+	o																		
«bussines» Experiencia								o	o		o																		
«bussines» Necesidades de formación								o		o																			
«bussines» Personas								o	o				+	o	o														
«bussines» Plan de desarrollo								o								o													
«objective» Configuración inicial de las empresas																													
«objective» Mostrar indicadores de forma gráfica																													
«objective» Permitir la configuración de la seguridad por roles																													
«objective» Permitir la evaluación por diferentes evaluadores													+																
«objective» Permitir mostrar los resultados de las evaluaciones													+																
«objective» Presentar informes imprimibles																													
«objective» Presentar informes técnicos																													
«risk» Disponibilidad de recursos																													
«risk» Especificación																		o	o	o		o							
«risk» Imprevistos técnicos					o	o	o																						
«risk» Inexperiencia				o	o																								
«risk» Tiempos y costos																		o	o	o	o	o							

Tabla 20. Proyecciones reflejadas de los concerns.

Como se evidencia en la Tabla 20, una vez realizada la proyección reflejada de los concerns, no se detectó conflictos entre los concerns (Específicamente para los concerns estudiados ergonomía, evaluaciones y competencias) y los demás concerns identificados en el sistema.

4.2.7 Vistas de concerns

A continuación se exponen las vistas generadas a partir del caso de aplicación, como se propone en la sección 2.3.2.4 generar vistas de concerns, con los respectivos concerns involucrados, para un mayor detalle de los concerns y sus vistas remítase al anexo B.

- **Vista de negocio:** Como lo propone su definición en esta vista se muestran los concerns de tipo objetivos, negocio, calidad y restricciones de ley.

  En la **Figura 36** se muestra la vista de negocio para el caso de aplicación del modelo AORE extendido, en esta vista se encuentran los concerns de negocio (Necesidades de formación, evaluaciones, competencias, personas, cargos, configuración del sistema, empresas experiencia y plan de desarrollo), de calidad (Ergonomía) y los objetivos (Permitir la evaluación por diferentes evaluadores, configuración inicial de las empresas, permitir mostrar los resultados de las evaluaciones, permitir la configuración de la seguridad por roles, mostrar indicadores de forma gráfica). Para el sistema usado como caso de aplicación, GESCOM, en el catálogo de requisitos no se detectaron restricciones de ley.

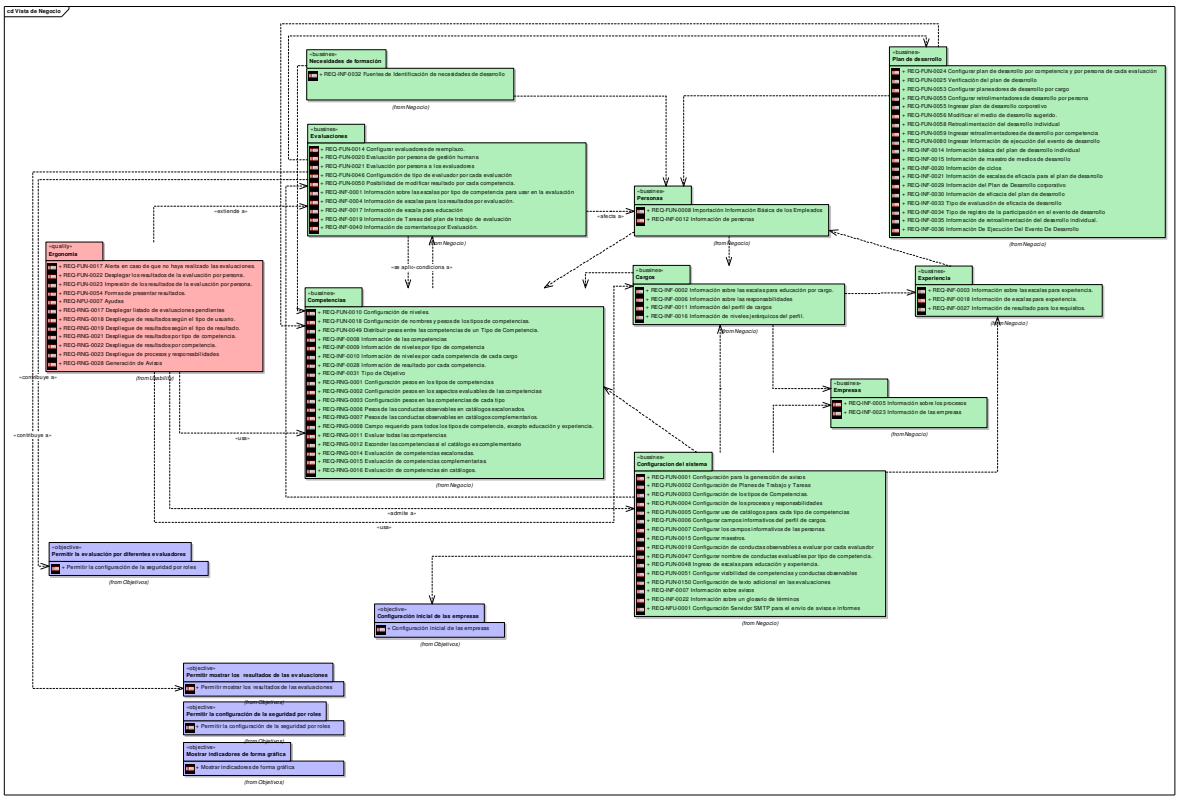

Figura 36. Vista de negocio.

- **Vista funcional:** La vista funcional está conformada por los concerns de tipo objetivos y concerns de tipo negocio.

En la **Figura 37** se muestra la vista funcional obtenida del caso de aplicación, en el cual se detectaron los concerns de tipo objetivo: permitir la evaluación por diferentes evaluadores, configuración inicial de las empresas, permitir mostrar los resultados de las evaluaciones, permitir la configuración de la seguridad por roles, mostrar indicadores de forma gráfica;, y concerns de tipo negocio: necesidades de formación, evaluaciones, competencias, personas, cargos, configuración del sistema, empresas, experiencia y plan de desarrollo.

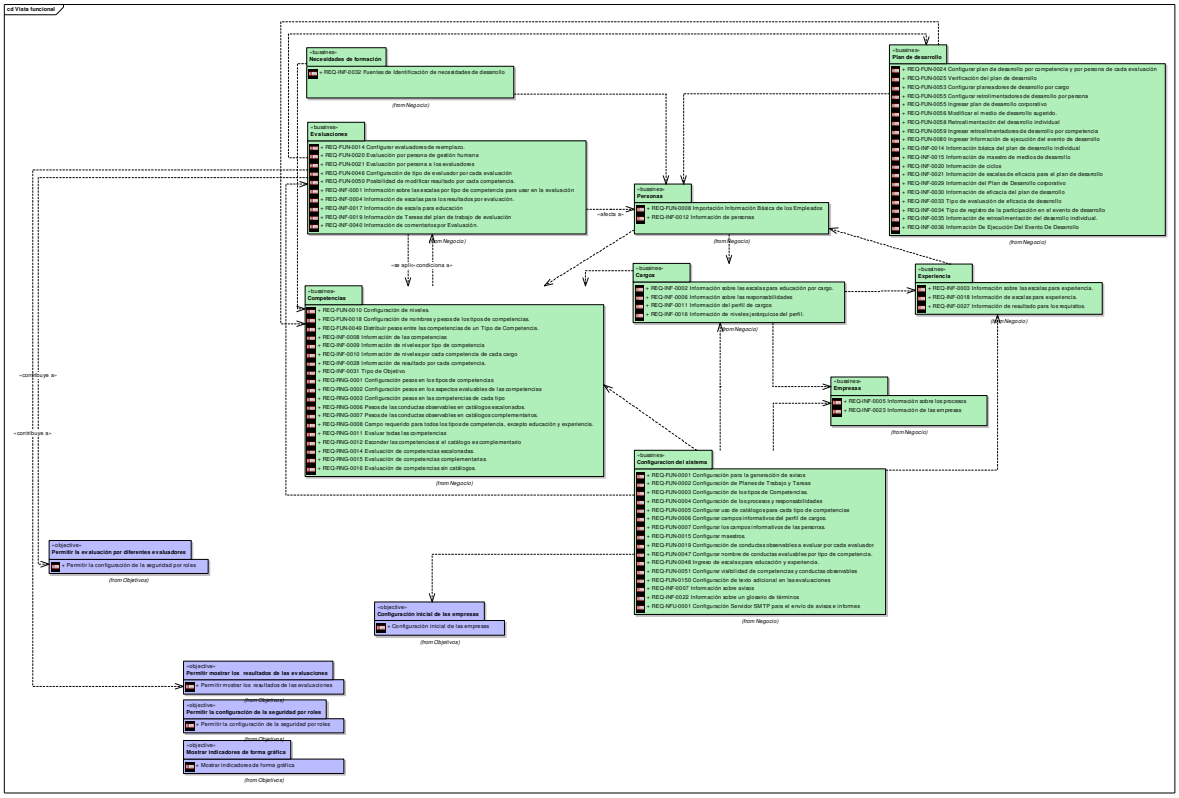

Figura 37. Vista funcional

- **Vista QoS – Quality of service:** En la Figura 38 se muestra la vista de QoS obtenida a partir del caso de aplicación, se extraen los concerns restricciones técnicas y ergonomía, los cuales pertenecen a los concerns de tipo contexto y calidad que conforman esta vista.

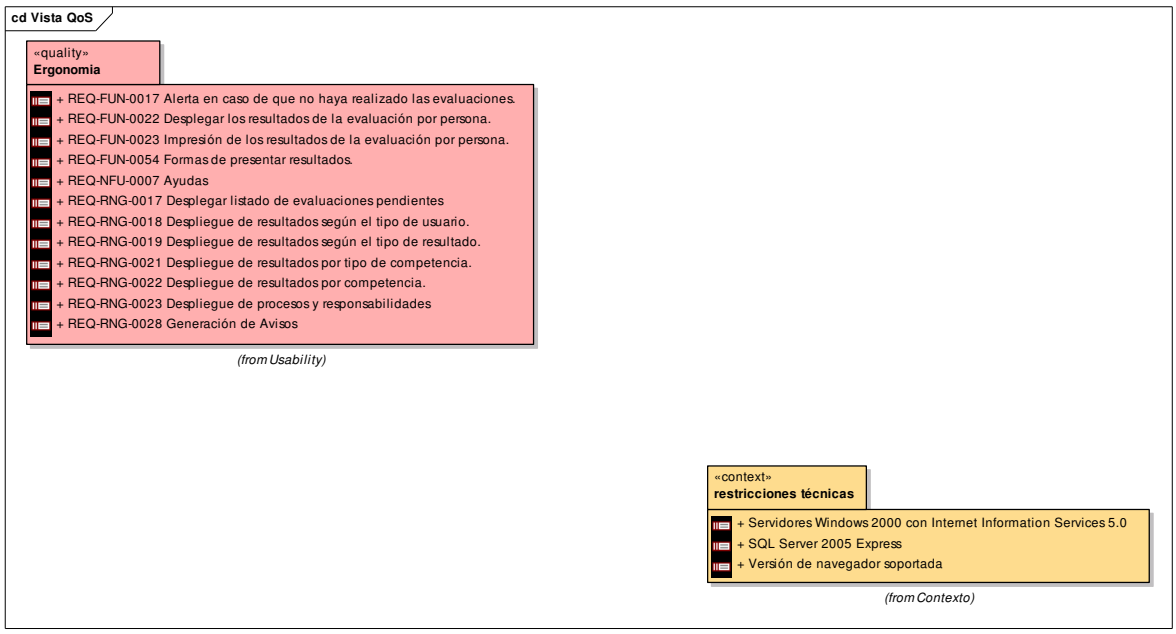

Figura 38. Vista QoS - Quality of service

- **Vista de plataforma:** En la Figura 39 se muestra la vista de plataforma, la cual está compuesta por los concerns de tipo físicos, sistemas existentes y servicios. En la aplicación del caso para el catálogo de requisitos de GESCOM sólo se detectaron concerns del tipo físico: estaciones de desarrollo, servidores, herramientas de desarrollo, componentes externos / frameworks, infraestructura / plataforma, herramientas de desarrollo.

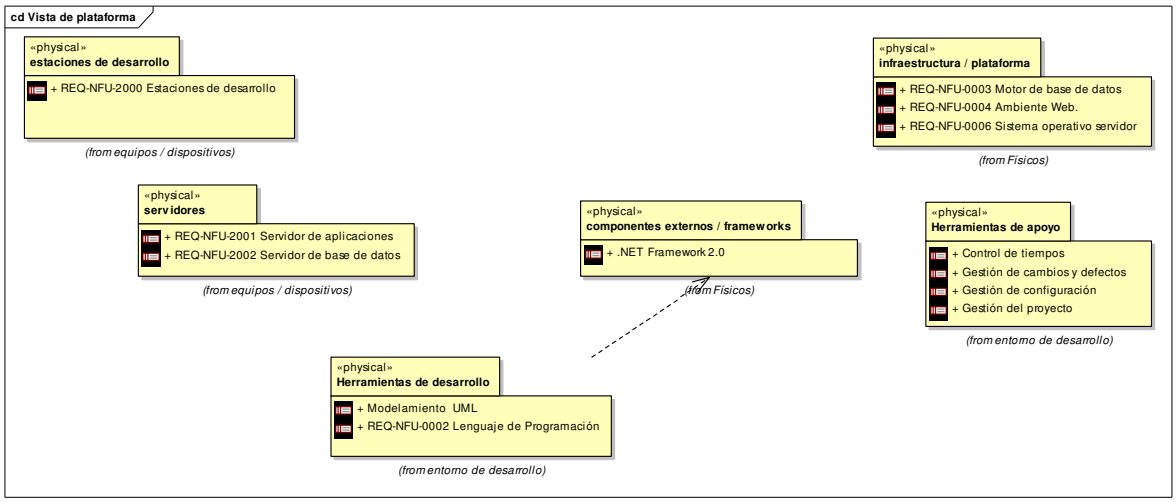

Figura 39. Vista de plataforma

- **Vista de proyecto:** En la Figura 40 se muestra la vista de proyecto la cual está compuesta por los concerns de tipo objetivos, negocio y riesgos. Del caso de aplicación se extrajeron los concerns objetivos: permitir la evaluación por diferentes evaluadores, configuración inicial de las empresas, permitir mostrar los resultados de las evaluaciones, permitir la configuración de la seguridad por roles, mostrar indicadores de forma gráfica; de negocio: necesidades de formación, evaluaciones, competencias, personas, cargos, configuración del sistema, empresas experiencia y plan de desarrollo; y los riesgos: especificación, disponibilidad de recursos, tiempos y costos, inexperiencia.

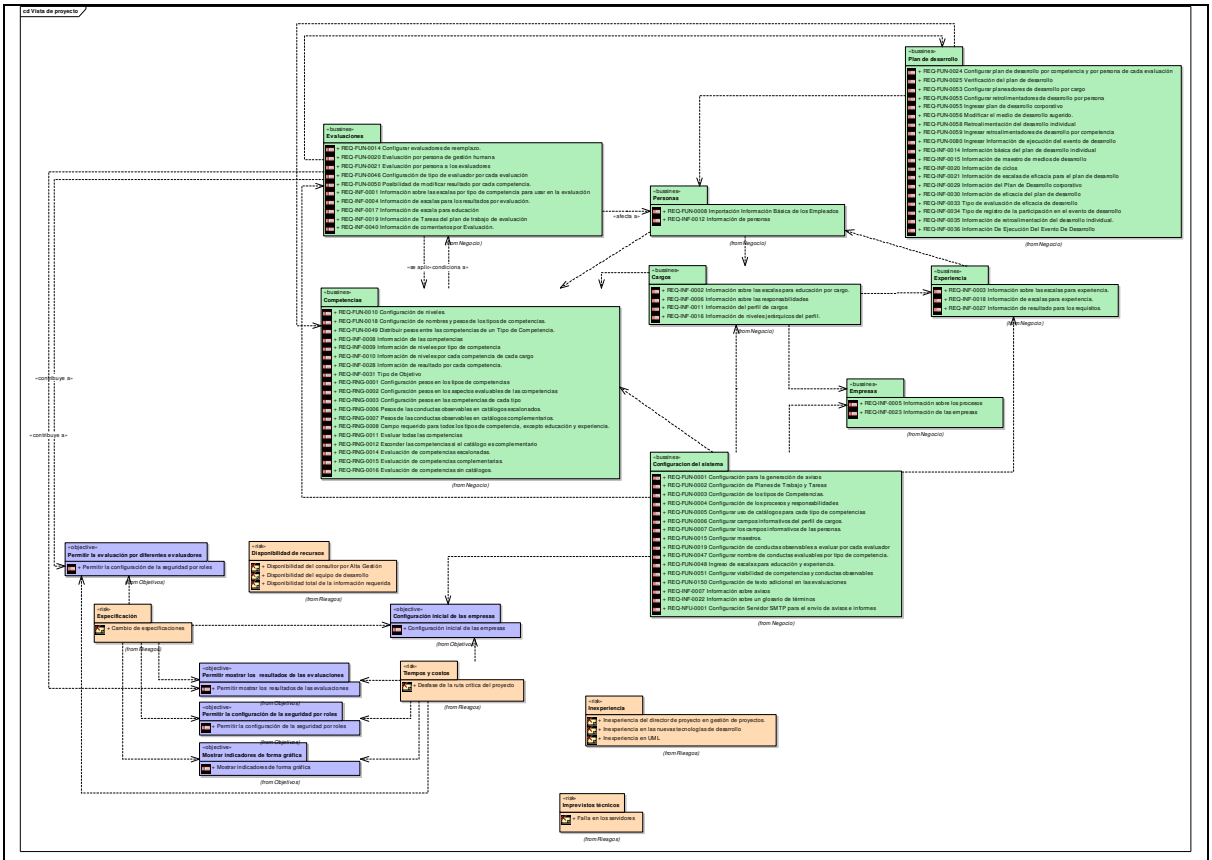

Figura 40. Vista de proyecto.

4.2.8 Dimensiones de concerns

Las dimensiones de los concerns identificados para el sistema se presentan en la tabla 21.

Concern	Influence	Mapping
Ergonomía	Diseño, evolución	Decisión
Restricciones técnicas	Diseño, construcción	Decisión
Componentes externos / frameworks	Diseño, arquitectura, construcción	Decisión
Herramientas de apoyo	Análisis, diseño, construcción, soporte y mantenimiento.	Decisión
Herramientas de desarrollo	Análisis, diseño, construcción	Decisión
Estaciones de desarrollo	Análisis, diseño, construcción	Decisión
Servidores	Arquitectura, instalación, soporte y mantenimiento	Decisión
Infraestructura / plataforma	Diseño, arquitectura	Aspecto
Cargos	Especificación, análisis.	Clase
Competencias	Especificación, análisis.	Clase
Configuración del sistema	Especificación, análisis.	Aspecto
Empresas	Especificación, análisis.	Clase
Evaluaciones	Especificación, análisis.	Clase
Experiencia	Especificación, análisis.	Clase
Necesidades de formación	Especificación, análisis.	Clase
Personas	Especificación, análisis.	Clase
Plan de desarrollo	Especificación, análisis.	Clase
Configuración inicial de las empresas	Especificación, análisis	Función
Mostrar indicadores de forma gráfica	Especificación, análisis	Función
Permitir la configuración de la seguridad por roles	Especificación, análisis	Función
Permitir la evaluación por diferentes evaluadores	Especificación, análisis	Función
Permitir mostrar los resultados de las evaluaciones	Especificación	Función
Presentar informes imprimibles	Especificación, construcción	Función
Presentar informes técnicos	Especificación, construcción	Función
Disponibilidad de recursos	Especificación, análisis	Decisión
Especificación	Especificación, análisis	Decisión
Imprevistos técnicos	Construcción, soporte y mantenimiento	Decisión
Inexperiencia	Especificación, análisis, diseño y construcción.	Decisión
Tiempos y costos	Análisis, diseño y construcción.	Decisión

Tabla 21. Especificación de las dimensiones de los concerns.

Observaciones

La aplicación del modelo AORE ampliado permite realizar algunas observaciones adicionales a las planteadas en la sección 2.2.1.6.

- Se hace difícil la aplicación de las relaciones de grano fino ya que no se cuenta con heurísticas que guíen el proceso. Las relaciones de grano fino propuestas por AORE son complejas, las acciones de restricción se hacen difíciles de aplicar, al tiempo que los operadores son confusos.

- Muchos requisitos de los presentados en la aplicación del modelo no presentan relaciones con otros requisitos de otros concerns. Esto puede deberse a fallas en el proceso de elicitación de los requisitos o en el análisis de estos.

5. CONCLUSIONES

La elaboración del trabajo presentado nos ha permitido tener un acercamiento al paradigma orientado a aspectos, el cual se presenta como la siguiente generación en el desarrollo de software.

Este trabajo ha sentado su esfuerzo en analizar la viabilidad de la aplicación del paradigma aspectual en las etapas tempranas del ciclo de vida del desarrollo de software, más específicamente en la ingeniería de requisitos a nivel industrial. Para esto se recurrió al modelo de ingeniería de requisitos AORE, propuesto por la Universidad Nova de Lisboa.

La exploración inicial del modelo estaba orientada a conocer y entender los elementos fundamentales de este, sin embargo en el transcurso de esta, se encontró que el modelo carecía de elementos que permitieran su aplicación a nivel industrial de forma que se facilitara el análisis de los concerns que componen un sistema de acuerdo a su naturaleza. En busca de estos elementos, se recurrió al modelo COSMOS, el cual presenta una gran cantidad de elementos como clasificaciones de concerns y relaciones entre estos.

El estudio del modelo COSMOS permitió conocer a detalle sus elementos, los cuales están determinados por características estructurales. Este hecho, difiere de los elementos que se pretendían buscar para complementar el modelo AORE, razón por la cual se decide realizar una propuesta de extensión del modelo, incluyendo elementos que permitan realizar una mejor clasificación de los concerns, ampliar el universo de los concerns propuestos y mejorar las relaciones entre concerns propuestas originalmente. En la propuesta de ampliación del modelo, se retoman ideas y elementos de COSMOS, el cual nos permitió tener una visión más amplia de los concerns que están inmersos en un sistema.

Además de la necesidad de extender el modelo AORE para incluir y reforzar los elementos mencionados, se hace evidente la necesidad de contar con una herramienta que permita soportar de manera adecuada el proceso. La propuesta original del modelo, establece el uso de descriptores XML para realizar la especificación de los elementos del modelo. Esta propuesta es bastante interesante dadas las fortalezas ofrecidas por un lenguaje de marcas como XML, sin embargo, si se tiene en cuenta la gran cantidad de elementos como requisitos y relaciones que se tienen a nivel industrial, la propuesta de usar descriptores XML deja de ser atractiva. Esto conlleva a proponer el uso de la herramienta Enterprise Architect para soportar el modelo.

A partir de la propuesta de ampliación del modelo AORE y de la propuesta de la herramienta Enterprise Architect para soportar el modelo, es posible realizar la aplicación del modelo extendido en un caso de nivel industrial. Para esto, se ha tomado el sistema GESCOM, un sistema de gestión de competencias desarrollado por AVANSOFT S.A. y a partir del cual se analiza la viabilidad de la aplicación del modelo.

Muchas de las observaciones y conclusiones realizadas al modelo de ingeniería de requisitos AORE y al modelo COSMOS se encuentran plasmadas a lo largo de este trabajo. Las conclusiones a continuación presentadas se centran en la aplicación del modelo AORE extendido a nivel industrial.

El modelo AORE, y por consiguiente el modelo extendido, parte de un catálogo existente de requisitos. Este hecho facilita su incorporación a los procesos de desarrollo de las empresas de desarrollo de software ya que las curvas de aprendizaje son menores al tener conocimientos de metodologías de elicitación de requisitos y experiencia en el manejo y tratamiento de estos.

Las relaciones de grano fino propuestas por el modelo AORE se presentaron como una de las mayores dificultades al momento de aplicar el modelo. Esto debido a la cantidad de requisitos que conforman un sistema de nivel industrial y a lo cual se suma la confusión que genera contar con relaciones conformadas por tres elementos: acciones, operadores y acciones de resultado. Estos tres elementos y los múltiples valores que pueden tomar en cada relación hacen que la ejecución de la tarea "Identificación de relaciones de grano fino", al no contar con heurísticas que guíen el proceso, sea compleja y ambigua.

La aplicación del modelo AORE ampliado a nivel industrial, requiere el uso de una herramienta que facilite la ejecución de las tareas propuestas. Enterprise Architect se presenta como una herramienta robusta que permite la adaptación de muchos de sus artefactos para la aplicación del modelo, sin embargo no da un soporte completo para todas las actividades de este.

Como se mencionó anteriormente, el modelo AORE parte de un catálogo de requisitos. Esto presenta una ventaja en cuanto a facilidad para adopción del modelo, sin embargo, la utilización de un catálogo de requisitos que no se encuentre bien especificado ocasionará que la aplicación del modelo se torne más compleja, al tiempo que no será eficaz. Esto nos lleva a concluir que la aplicación del modelo AORE (y su extensión) es viable a nivel industrial, siempre bajo la condición de contar con un proceso maduro de elicitación de requisitos. No satisfacer esta condición tiene como resultado la aplicación de un modelo que no realiza mayor aporte al tratamiento de los concerns y los requisitos que los componen, al tiempo que los resultados obtenidos en la actividad "especificación de las dimensiones de concerns" no será confiables y tendrán impacto en etapas posteriores del proceso de desarrollo de software.

GLOSARIO

AOSD: Aspect Oriented Software Development. Desarrollo de Software Orientado a Aspectos. Es una aproximación reciente para el desarrollo de software cuyo objetivo es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos

AP: Adaptive Programming. Programación Adaptativa. Modelo de programación aspectual que provee una abstracción de la estructura de las clases y su navegación para soportar de una forma más adecuada el conocimiento del comportamiento de las operaciones.

AOP: Aspect Oriented Programming. Ver POA. Programación Orientada a Aspectos.

Aspect (Aspecto): Un aspecto es una unidad modular diseñada para implementar un concern. La definición de un aspecto puede contener algún código y las instrucciones dónde cuándo y cómo ser invocado.

ASSAM: A Concern-Oriented Approach To Software Architecture. Metodología cuyo objetivo es identificar y especificar aspectos arquitectónicos, y hacerlos transparentes desde etapas tempranas del ciclo de vida del desarrollo de software.

Concern (Interés): Es una propiedad de interés en el sistema para algún stakeholder.

Constraint actions: Acciones de restricción. Elemento del modelo AORE que define cómo los requisitos del concern serán afectados por otro requisito.

Constraint operators: Operadores de restricción. Elemento del modelo AORE que define cómo los requisitos del concern serán afectados por otro requisito.

Crosscuting concern (Interés de corte transversal): Separación de intereses. Hace referencia a la comprensión de los conceptos del sistema como unidades de software independientes

Descomposición funcional: Tendencia en el ámbito de la programación, cuyo objetivo es plantear una clara división entre los datos y la funcionalidad, identificando las partes más manejables como funciones que se definen en el dominio del problema.

EA: Enterprise Architect. Herramienta CASE de diseño UML.

Elicitación de requisitos: Actividad en el proceso de desarrollo de software, en la cual se lleva a cabo una directa interacción con los usuarios y los clientes, en esta fase se identifica el dominio del problema, el contexto, el modelo de proceso de negocio, las necesidades de los usuarios y clientes, y se realiza una puesta en común de los stakeholders con respecto a los requisitos.

Encapsulamiento: Característica de la programación orientada a objetos. Se denomina "encapsulación" al ocultamiento del estado, es decir, de los datos miembro, de un objeto de manera que sólo se puede cambiar mediante las operaciones definidas para ese objeto.

Enmarañamiento: Consiste en la necesidad de hacer que un sólo componente del sistema tenga que realizar todo un conjunto de requisitos.

GESCOM: Sistema para la gestión del talento del recurso humano, basado en el modelo gestión por competencias, y desarrollado por AVANSOFT S.A. El modelo por compentencias usado en GESCOM es el creado por la empresa ALTA GESTIÓN EMPRESARIAL

Herencia: Es uno de los mecanismos de la programación orientada a objetos, por medio del cual una clase se deriva de otra de manera que extiende su funcionalidad.

Herramienta CASE: (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) herramientas que ayudan en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Ilities: Termino extraído de la norma ISO 9126, para hacer referencia a los atributos de calidad del sistema como lo son funcionabilidad, confiabilidad, usabilidad, eficiencia, mantenibilidad, portabilidad **(**functionality, reliability, usability, efficiency, maintainability, portability**).**

Join point: Ver Punto de enlace.

Objeto: Programación Orientada a Objetos – POO. Representación detallada, concreta y particular de una entidad del sistema. Tal representación determina su identidad, su estado y su comportamiento particular en un momento dado.

Outcome actions: Acciones de resultado. Especifica el resultado después de afectar el requisito del concern con otro requisito**.**

POA: Programación Orientada a Aspectos. Es un paradigma de programación cuyo objetivo es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de conceptos.

Polimorfismo: Capacidad que tienen objetos de diferentes clases de responder al mismo mensaje. Esto significa que puede haber muchos mensajes con el mismo nombre, en diferentes clases. Cada Clase responde al mensaje con su código propio.

Punto de enlace: Puntos comunes combina la funcionalidad base con el comportamiento aspectual.

QoS: Quality of Service. Calidades de servicio. Vista propuesta en el modelo AORE extendido.

Scattering: Necesidad de diseminar un conjunto de requisitos a través de muchos componentes del sistema.

Separation of concerns (Separación de intereses): Hace referencia a la comprensión de los conceptos del sistema como unidades de software independientes.

Stakeholder: Son todas las personas que tienen algún interés o relación con el sistema en cuestión.

Tangling: Ver Enmarañamiento.

Tejedor: En ingles Weaver. Es el encargado de realizar la mezcla de la funcionalidad base con el comportamiento aspectual. Sea en tiempo de compilación o en tiempo de ejecución.

BIBLIOGRAFIA

[1] CLARKE, Siobhán. BANIASSAD, Elisa. Aspect Oriented Analysis and Design, The Theme Aproach. Addison- Wesley, 2005.

[2] KICILLOF, Nicolás. Programación Orientada a Aspectos (AOP) http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art152.asp

[3] APPLETON, Brad. Introducing Demeter and its Laws. http://www.cmcrossroads.com/bradapp/docs/demeter-intro.html

[4] ALFEREZ, Mauricio. ALFEREZ, German. Análisis de un sistema de información bajo la aproximación de la orientación a aspectos caso práctico: manejo de solicitudes en la mesa de ayuda de servicios generales de una universidad. Universidad EAFIT, 2004.

[5] CONSTANTINIDES, Constantinos, ELRAD, Tzilla, y FAYAD, Mohamed. Extending the object model to provide explicit support for crosscutting concerns. s.l: John Wiley & Sons, 2002.

[6] ASTEASUAIN, Fernando. EZEQUIEL, Bernardo. POA: Análisis del Paradigma http://www.programemos.com/index.php?option=com_content&task= view&id=158&Itemid=27

[7] NIETO, Juan Manuel. Introducción a la programación orientada a aspectos. Escuela Técnica Superior de Ingeniería Informática. Universidad de Sevilla, España

[8] NUSEIBEH, Bashar. Crosscutting Requirements. The Open University, UKNuseibeh. 2004

[9] FILMAN, Robert. ELRAD, Tzilla. CLARKE, Siobhán. AKSIT, Mehmet. Aspect Oriented Software Development.

[10] HARRISON, William. OSSHER, Harold. "Subject-Oriented Programming (A Critique of Pure Objects)." IBM T.J. Watson Research Center

[11] AKSIT, Mehmet. "Composition and Separation of Concerns in the Object Oriented Model". University of Twente. 1996

[12] JACOBSON, Ivar. Use case and aspects-working seamlessly together. Journal of Object Technology. 2003

[13] TEKINERDOGAN, Bedir, MOREIRA, Ana, ARAÚJO, Joao, CLEMENTS, Paul. "Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design". 2004

[14] MOREIRA, Ana, RASHID, Awais, ARAUJO, Joao. "Multidimensional Separation of Concerns in Requirements Engineering". 2005.

[15] BAKKER, Jethro. "Traceability of Concerns". University of Twente. 2005.

[16] KANDÉ, Mohamed. "Concern Oriented Approach to Software Architecture" Doctoral Thesis. 2003

[17] TARR, Peri, OSSHER, Harold, SUTTON, Sanley, HARRISON, William. "N Degrees of Separation: Multi-Dimensional Separation of Concerns." ICSE 21. 1999.

[18] SUTTON, Stanley. ROUVELLOU, Isabelle. "Advanced Separation of Concerns for Component Evolution". 2001

[19] DESMONDS, Michael. "Aspect Oriented Programming". http://www.skynet.ie/~mscse2004/extras/aop.ppt

[20] REINA, Antonia. Visión general de Aspectos. Sevilla, España.2000.

[21] VAN DEN VERG, Klaas. CONEJERO, José Maria. "A conceptual Formalization of Crosscutting is AOSD". Trese Group, University of Twente.2005

[23] SUTTON, Stanley. "Concerns in a Requirements Model – A Small Case Study"

[24] SUTTON, Stanley. ROUVELLOU, Isabelle. "Modeling of Software Concerns in COSMOS". 1st Int' Conf. on Aspect-Oriented Software Development. 2002

[25] BAKKER, Jethro, TEKINERDOGAN, Bedir, AKSIT, Mehmet. "Characterization of Early Aspects Approaches"

[26] MOREIRA, Ana, RASHID, Awais, ARAUJO, Joao. "Modularization and Composition of Aspectual requirements", 2003.

[27] MOREIRA, Ana, RASHID, Awais, ARAUJO, Joao. "A Concern-Oriented Requirements Engineering Model", 2005.

[28] FILMAN, Robert. ELRAD, Tzilla, CLARKE, Siobhan. AKSIT, Mehmet. “Aspect-Oriented Software Development”. Addison-Wesley. 2005.

[29] MOREIRA, Ana, RASHID, Awais, SAWYER, Peter, ARAUJO, Joao. “Early Aspects: A model for Aspect-Oriented Requirements Engineering”. IEEE Computer Society Press. 2002.

[30] KANG, Kyo, SHOLOM, Cohen, HESS, James, NOVAK, William, PETERSON Spencer , "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Software Engineering Institute. 1990.

[31] VILLER, Stephen, SOMMERVILLE, Ian. "Social Analysis in the Requirements Engineering Process: From Ethnography to Method". EEE Computer Society. 1998.

[32] RAYSON, Paul, EMMET, Luke, GARSIDE, Roger, SAWYER, Pete. "The REVERE Project: Experiments with the application of probabilistic NLP to Systems Engineering". 2000

[33] AVANSOFT S.A. “LEVANTAMIENTO DE REQUISITOS”. Avansoft S.A. 2004.

[34] MOREIRA, Ana, FUENTES, Lidia, HERNANDEZ, Juan. “Desarrollo de software orientado a aspectos.” Universidad de Extremadura. 2003.

[35] ISO/IEC 9126-1: 2001 International Standard “Software Engineering”

[36] RASHID, Awais, MOREIRA, Ana, ARAÚJO, Joao, CLEMENTS, Paul. Elisa. Baniassad, TEKINERDOGAN, Bedir. "Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design" 2004.

[37] CHUNG, Lawrence, NIXON Brian, YU, Eric, MYLOPOULOS, John. "Non-Functional Requirements in Software Engineering". Kluwer Academic Publishers. 2000.

[38] SPARX SYSTEMS. "Enterprise Architect - UML Design Tools". Sparx Systems Pty Ltd. 2006.

[39] DURÁN, Amador. "Metodología para la elicitación de requisitos de sistemas de software". Escuela técnica superior de ingeniería informática. 2002.

[40] TEKINERDOGAN, Bedir. "ASAAM: Aspectual Software Architecture Analysis Method". 2004.

[41] KANDÉ, Mohamed. "A Concern-Oriented Approach To Software Architecture" 2003

[42] KRUTCHEN, Philippe. "The 4+1 View Model of Software Architecture". 1995.

[43] KONTIO, Mikko. "Designing software architectures. Introducing the 4+1 view model". 2005.
http://www-128.ibm.com/developerworks/wireless/library/wi-arch11/

[44] HUGUNIN, Jim. "The Next Steps For Aspect-Oriented Programming Languages (in Java)". Xerox Palo Alto Research Center. 2001.

[45] ASTEASUAIN, Fernando. “Lenguajes de aspectos específicos.” 2005. http://www.programemos.com/index.php?option=com_content&task=view&id=167&Itemid=77.

[46] YU, Yijun, SAMPAIO DO PRADO LEITE, Julio Cesar, MYLOPOULOS, John. “From Goals to Aspects: Discovering Aspects from Requirements Goal Models”. Department of Computer Science, University of Toronto. 2004.

[47] BRITO, Isabel, MOREIRA, Ana. “Integrating the NFR framework in a RE model”. 2004.