“UIDRe”: Herramienta Case de UIDs para el proceso de análisis de requerimientos

Mario Matías Urbieta[1] **Carla Marina Vairetti**[2]

Trabajo final para obtener el grado de

Licenciado en Informática / Licenciatura en Sistemas

De la
Facultad de Informática,
Universidad Nacional de La Plata,
Argentina

Director: Dr. Gustavo Rossi

La Plata, Julio de 2006

[1] Nro. Alumno: 4209/6 – matiasurbieta@gmail.com
[2] Nro. Alumno: 3748/6 – cvairetti@gmail.com

Índice General

Índice de ilustraciones

1 Agradecimientos

En primer lugar; quiero agradecer a mi familia principalmente por todo el esfuerzo y la confianza dedicada; a ellos les debo más que estos años académicos.
A mi papá que siempre se preocupo por cada momento de mi carrera y sigue estando conmigo.
A mi mama por ser mi sostén en los momentos mas difíciles y a mi hermana por enseñarme a que todo se consigue si uno el pone esfuerzo.
A Rodrigo por darme el amor, el apoyo y las fuerzas por terminar esta etapa tan difícil.
A mis amigos que siempre estuvieron y compartieron todos estos años tan emocionantes.
A mi compañero de tesis por hacer que este trabajo fuese tan llevadero, divertido y a la vez constructivo.

Carla Vairetti

Empiezo los agradecimientos con la familia. A mis padres, Alicia y Mario, que me apoyaron siempre, tanto en los momentos de alegría como en los momentos difíciles. Dándome confianza para cada nuevo reto y ayudándome a alcanzarlo.
Mis hermanos son las personas con las que más tiempo conviví en toda mi vida y son los que mejor me conocen. En esta última etapa fueron con los que compartí todas mis experiencias y me ayudaron a superar cada una de las dificultades.
Amigos, ellos son mi segunda familia, con ellos no solo compartí las actividades académicas sino también las circunstancias de la vida apoyándonos y aprendiendo a compartir.
Por ultimo a Carla, por acompañándome en este ultimo trayecto de la carrera.

Matias

Queremos agradecer a nuestro director Gustavo por el tiempo y la predisposición que tuvo hacia nosotros.

Carla y Matías

2 Introducción

2.1 Introducción y motivación

En la actualidad, son muchos los procesos de desarrollo de software que existen. Con el pasar de los años, la Ingeniería de Software ha introducido y popularizado una serie de estándares para medir y certificar la calidad del proceso de desarrollo que concluye en el sistema desarrollado. Se han publicado muchos libros y artículos relacionados con este tema, con el modelado de procesos del negocio y la reingeniería. Un número creciente de herramientas automatizadas han surgido para ayudar a definir y aplicar un proceso de desarrollo de software efectivo. Hoy en día la economía global depende más de sistemas automatizados que en épocas pasadas; esto ha llevado a los equipos de desarrollo a enfrentarse con una nueva década de procesos y estándares de calidad.

La Ingeniería de Software y, en general, la Informática es una disciplina que está en constante evolución. Cada día surgen nuevas técnicas y metodologías que intentan mejorar la calidad y eficiencia de los productos de software.

La Ingeniería de Requerimientos cumple un papel primordial en el proceso de producción de software para obtener el producto deseado en una calidad óptima. Su principal tarea consiste en el relevamiento de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, el comportamiento del sistema; de esta manera, se pretende minimizar los problemas relacionados al desarrollo de sistemas.

Las aplicaciones de comercios electrónicos Web, requieren frecuentes modificaciones y adaptaciones rápidas a los cambios de negocio. Se necesitan técnicas y conceptos de modelado flexibles que permitan construir diseños reutilizables y genéricos.

Todas las tecnologías existentes no tienen suficiente grado de abstracción y en consecuencia expresividad para modelar un sistema.

El principal objetivo de la comunidad de Desarrollo de Software orientado a Aspectos (AOSD) consiste en la definición de técnicas, métodos y herramientas basados en UML. Esta comunidad esta tratando de alcanzar un nuevo grado de abstracción para el proceso de desarrollo de software. Los aspectos ya son parte del la implementación de los nuevos sistemas aunque aún no han logrado implantarse en las etapas anteriores a la de desarrollo.

En muchos documentos emitidos por esta comunidad se destaca la necesidad del modelado de propósitos en las primeras etapas del ciclo de vida del software.

2.2 Definición del problema

La separación de propósitos reduce la complejidad y facilita la compresión y seguimiento a lo largo del desarrollo de software. Esto minimiza el impacto de cambios por medio de la encapsulación de propósitos en diferentes módulos. Las diferentes tecnologías de desarrollo de software, como la programación orientada a objetos, fueron propuestas para alcanzar estos objetivos.

Sin embargo, hay ciertas propiedades que las tecnologías actuales no son capaces de modularizar. La razón de porque estas propiedades se modularizan con dificultad es porque afectan un gran número de otras propiedades que componen el sistema.

Estas propiedades, conocidas como propósitos horizontales, son las responsables de producir código esparcido donde la implementación de una propiedad dada no es localizable en un solo modulo, y de código disperso donde cada módulo contiene información de varías propiedades distintas. En consecuencia las aplicaciones son difíciles de entender, mantener y evolucionar.

El Desarrollo de Software orientado a Aspectos (AOSD) tiene como objetivo el análisis de los propósitos que se presentan horizontalmente en los sistemas utilizando medios semánticos para la identificación, separación, representación y composición de éstos.

Estos propósitos horizontales son encapsulados en módulos separados, conocidos como aspectos, que luego compuestos con el resto de los módulos.

Esto da lugar a una modularizacion que permite reducir los costes del desarrollo, mantenimiento y de la evolución de los sistemas.

Durante la ingeniería de requerimientos las propiedades que el software debe exhibir tienen que ser capturada y analizada. El análisis de esta información capturada y el contexto organizacional y operacional resulta en una síntesis de un conjunto de requerimientos. Estos requerimientos tienen que ser en lo posible, correctos, completos y factible. Para obtener requerimientos con estas características se necesita comunicación y negociación con los clientes.

Estos requerimientos son plasmados en un documento de especificación el cual comunica los requerimientos a las personas que desarrollan el software.

La tecnología comúnmente utilizada en la actualidad es UML. UML sugiere la utilización de casos de uso para capturar requerimientos y especificar la interacción entre el usuario y el sistema que esta siendo modelado. Los casos de uso son fáciles de entender por el usuario, pero la falta de precisión y consistencia alcanzada entre las otras herramientas de UML nos incentiva a utilizar una herramienta que facilite la integración entre estas.

Por otro lado no hay un método sistemático que ayude al diseñador obtener tal diagrama UML desde un conjunto de casos de uso. Para solucionar este problema nosotros proponemos el uso de UID (User Interaction Diagram).

Los UID han sido propuestos para ser una herramienta valiosa para obtener requerimientos, ya que esta describe el intercambio de información entre el sistema y el usuario en un alto nivel de abstracción; enfocándose solo en el intercambio de información y no en otros detalles como la interfase, que si lo vemos en otros diagramas de UML.

En UML la interacción es vista como el intercambio de información entre el usuario y los objetos del sistema, la cual difiere de UID en el cual el intercambio ocurre entre el usuario y el sistema.

Para la obtención de requerimientos UML sugiere la utilización de casos de usos para modelar la interacción de los usuarios en vez de utilizar los diagramas de secuencia, colaboración, etc.; que también provee UML.

Los UID son principalmente usados para la comunicación entre el diseñador y el usuario en la definición de los requerimientos del sistema. Además, los UIDs pueden ser utilizados para derivar un diagrama de clases de manera automática basándose en un conjunto de axiomas que mantienen la completitud y correctitud de éste diagrama.

Esta herramienta ha sido utilizada también para el desarrollo de aplicaciones Web ya que cada paso de interacción puede ser mapeado a un paso navegacional.

Utilizando esta técnica podemos localizar también propósitos horizontales en la primera etapa del ciclo de vida de una aplicación proyectando la separación futura de los propósitos en módulos independientes.

2.3 Contribuciones de la tesis

Esta tesis brinda un modelo de análisis de requerimientos en el que se intenta descubrir los cruces de requerimientos en las primeras etapas del desarrollo del software.

Básicamente el proceso propone facilitar la documentación de crosscutting concerns y dar una breve introducción a las consecuencias de su existencia en la etapa de diseño. Para ello en primer lugar se focaliza en mejorar relevamiento de requerimientos proponiendo la utilización de UIDs que facilita la comunicación con el cliente. Una vez definidos los requerimientos, comienza una etapa análisis donde se localizan los cruces de requerimientos; se brindan medios semánticos para señalar aspectos y composiciones navegacionales. Estos medios semánticos son una parte importante en la contribución porque extiende el modelo de análisis tradicional incursionando en Early Aspects a través de la identificación de aspectos en la etapa de análisis.

Entrándonos en la etapa de diseño, utilizamos heurísticas de derivación, obtenemos un modelo de objetos tentativo el cual nos sirve de prototipo. Aunque este prototipo es tentativo, es de gran ayuda yo que nos da un panorama de las entidades (objetos) participantes.

Finalmente, proponemos una base de análisis en el que se explora las relaciones encontradas en la identificación de aspectos para modelarlos y documentarlos en la etapa de diseño. El resultado obtenido es información tamizada, depurada, adecuada para un documento de especificación de la aplicación que el desarrollador utilizará para implemente los diferentes aspectos / Themes encontrados.

Para acompañar esta teoría tan importante, construimos una herramienta que es capaz de almacenar digitalmente en un documento universal como es XML los requerimientos obtenidos en la etapa de análisis.

Nuestra herramienta permite la creación de concerns con sus respectivos requerimientos, y adicionalmente la construcción gráfica de los Diagrama de Interacción de Usuario (UID). Una vez diseñados (graficados) estos diagramas se permite la posibilidad de generar reglas de composición que posteriormente se utilizarán generar un mapa integrador de interacción. En consecuencia el mapa refleja los diferentes cruces que se tiene entre los concerns.

Esta herramienta es de gran utilidad pues es el paso inicial de una plataforma de análisis. Esto nos permite hacer un análisis exhaustivo de las aplicaciones, reducir su complejidad, seguirla detalladamente y conseguir una mejor modularidad en la aplicación final.

2.4 Estructura del trabajo

Anteriormente dimos una introducción al problema de la evolución de la Ingeniería de Software, también mostramos cuáles son las motivaciones que nos llevan a su caso de estudio y posteriormente analizaremos el porqué de la utilización de los UID.

El trabajo desarrollado se focaliza en los UID que son principalmente usados para la comunicación entre el diseñador y el usuario en la definición de los requerimientos del sistema.

Para ello trabajaremos con casos de estudios reales, analizándolos para luego recopilar la información obtenida utilizando nuestra herramienta.

En el Capítulo 2, se dará una pequeña introducción de la historia de evolución el software para lograr integrar datos y funciones sin que el código se esparza por todos lados. Es decir, separar los componentes funcionales y los aspectos unos de otros, abstrayéndolos para realizar un análisis y diseño especifico del propósito (concern) y luego se define la composición de ellos, para formar todo el sistema.

Además se describen conceptos básicos que el lector deberán interiorizarse para la comprensión de la tesis en general. Describiremos: el concepto de concerns, clasificándolo en dos categorías: el "*core concern*" y los "*crosscutting concern*", analizando cuales son los problemas asociados a los mismos. Veremos también los conceptos de aspectos, mostrando que AOSD reduce las principales limitaciones, en las etapas de diseño e implementación, de la programación orientada a objetos con respecto al problema de modularidad y aumentando la capacidad de expresión permitiendo modularizar los crosscutting concerns; utilizando la programación orientada a aspectos (AOP).

Se describirá también la importancia de la ingeniería de requerimientos del software, los casos de uso como una herramienta para describir el uso del sistema una vez recopilados los requerimientos, y a continuación se citará una breve descripción de algunas técnicas de ingeniería de requerimientos agrupadas en dos grupos: orientadas a aspectos y no orientadas a aspectos. Además la iniciativa Early Aspects se concentra en el manejo de las propiedades crosscutting en las primeras etapas de la ingeniería de requerimientos y del diseño de la Arquitectura, con lo cual daremos cierre a este capítulo.

El Capítulo 3 describe conceptos relacionados los requerimientos en las aplicaciones Web, dando una breve introducción a las metodologías usadas para el desarrollo de aplicaciones Web.

Se explica la técnica de OOHDM que propone el desarrollo de aplicaciones hipermedia a través de un proceso compuesto por cuatro etapas: diseño conceptual, diseño navegacional, diseño de interfaces abstractas e implementación, dichas etapas son explicadas en este capítulo.

Otro tema importante que engloba este capítulo son los UID, dando su definición, el proceso de construcción, las heurísticas y derivaciones del diagrama de clases, el desarrollo de aplicaciones de acuerdo al Proceso Unificador, y la notación necesaria para representarlos.

Estaremos cerraremos el capítulo con una pequeña conclusión del uso de los UID.

El Capítulo 4 citamos como identificar y componer concerns navegacionales. Estos concerns navegacionales son de suma importancia para las aplicaciones Web porque generalmente la calidad de su estructura de navegación es una llave para el éxito de aplicación.

Además se explica un modelo propuesto para concerns navegacionales que propone un mecanismo de rehúso por medio de un catálogo de concerns definido de forma abstracta e independiente del sistema. Este catálogo debe ser usado para ayudar a obtener los concerns de dominio del problema. La idea es usar esos metaconcerns, para

ayudar a organizar los requerimientos del espacio del sistema de modo que los concerns del dominio del problema sean identificados.

Seguido a esto veremos como detectar aspectos en la etapa obtención de requerimientos por medio de la representación de una secuencia de pasos similar a los casos de usos y analizaremos algunas reglas de composición localizado los crosscutting.

El Capítulo 5 presenta la herramienta CASE en la que se ha trabajado, describiendo su funcionalidad, su diseño, las características técnicas (frameworks utilizados), la motivación de creación de esta herramienta.

En el Capítulo 6, damos una presentación al caso de estudio. Tomando en cuenta el sitio Web Amazon, realizamos un proceso de ingeniería inversa para luego definir un grupo de concerns con sus respectivos requerimientos. Una vez definido nuestro borrador de requerimientos, diseñamos los UIDs de aquellos requerimientos que tenían una interacción subyacente del sistema con el usuario.

Por ultimo, introducimos las extensiones semánticas de composición de Uids e identificación de aspectos. En el primer caso relacionamos dos o más interacciones para obtener un producto final más expresivo, completo e integrador de la aplicación. En el segundo caso, identificamos comportamiento transversal que no puede ser representado con UIDs.

En el Capítulo 7 comienza una etapa de análisis de las consecuencias de la etapa análisis de requerimiento sobre la etapa de diseño. Veremos casos particulares que pueden caer en ambigüedades o malas decisiones de diseño.

Por ultimo, se presentara un mapa conceptual de la solución tentativa del diseño conceptual habiendo utilizado sobre éste último las heurísticas citadas.

El Capitulo 8 defines los trabajos futuros a realizar, para ello completaremos el enfoque con un análisis más profundo, intentaremos hacer una derivación a partir de los UIDs. También veremos como se incrementara el comportamiento de la herramienta, utilizando motores de derivación del modelo y motores de vistas previas.

Finalmente concluiremos con el Capitulo 9, en conde daremos las conclusiones finales del trabajo de tesis.

3 Conceptos Básicos

3.1 Un poco de Historia

Desde los inicios de la Ingeniería de Software, uno de los principios fundamentales que se han aplicado para resolver problemas complejos ha sido el uso de la descomposición de un sistema en partes menores y más fáciles de manejar, aplicando el dicho popular de "divide y vencerás".

Así, con los primeros lenguajes de programación, el código no tenía separación de conceptos: datos y funcionalidad se mezclaban sin una línea divisoria clara. A esta etapa se la llama del *código spaghetti*, por cuanto el código estaba enmarañado a lo largo del programa, y los datos eran definidos indistintamente, sin separarse de la funcionalidad: éstos eran más bien un apoyo a ella.

En la siguiente etapa se pasó a aplicar la *descomposición funcional*, identificando dentro del dominio del problema, partes más manejables que se definían como funciones. Su ventaja era la integración de nuevas funciones, pero sin claridad, por cuanto se utilizaban datos compartidos, por un lado, y por otro, los datos quedaban también esparcidos por todo el código, con lo cual, el integrar un nuevo tipo de dato implicaba la modificación de nuevas funciones, haciendo más difícil el mantenimiento de los sistemas, parámetro de alta relevancia a la hora de medir la calidad de un producto de software.

Intentando resolver la separación de datos y funcionalidad, aparece la *programación orientada a objetos (POO)*, que permite integrar datos y funciones en un solo concepto llamado *objeto*. De esta manera, los sistemas se descomponen de manera más natural, permitiendo integrar nuevos datos fácilmente. Sin embargo, aspectos como el control de memoria, el manejo de errores, etc., no son controlados en una sola clase, sino que quedan esparcidos por todo el código, haciendo que la integración de funciones para manejar estos aspectos sea compleja, puesto que origina que se modifiquen varios objetos, produciendo un enmarañamiento de los objetos en funciones de alto nivel, que involucran a varias clases.

Uno de los principales problemas que se encuentran al agrupar estas descomposiciones es que muchas veces se tienen ejecuciones ineficientes puesto que las unidades de descomposición no siempre van acompañadas de un buen tratamiento de aspectos tales como la gestión de memoria, la coordinación, la sincronización, la distribución, la gestión de seguridad, etc.

Con el fin de enfrentar de mejor manera los problemas descritos, aparece la POA como una nueva metodología de programación que aspira a soportar la separación de propósitos (*concerns*), para los aspectos antes mencionados. Es decir, separa los componentes funcionales y los aspectos unos de otros, proporcionando mecanismos que hagan posible abstraerlos y componerlos para formar todo el sistema.

3.2 Concerns en Software

Un concern es un requerimiento o consideración especifica que debe ser alcanzada para satisfacer los objetivos de todo el sistema. Un sistema de software es la

concepción de un conjunto de concerns. Un sistema bancario, por ejemplo, esta compuesto por los siguientes concerns: administración de clientes y cuentas, computo de intereses, transacciones interbancarias, transacciones ATM, persistencia, etc. Además a los concerns del sistema, se requieren concerns de proyecto como: mantenimiento, seguimiento, fácil evolución.

Un concern puede ser clasificado en dos categorías: el *"core concern"* que captura la funcionalidad central del módulo y los *"crosscutting concern"* que capturan requerimientos que cruzan a otros subsistemas. Como por ejemplo: autentificación, logueo, administración, performance, persistencia de datos, seguridad, integridad transaccional, etc.

3.2.1 Separación

El principio de separación de propósitos (concerns) propone encapsular características en entidades separadas para poder localizar cambios de estos y trabajarlos de forma aislada reduciendo su complejidad de diseño e implementación. El primer paso es descomponer el conjunto de requerimientos separándolos en concern.

Identificar y separar los concerns en un sistema es un ejercicio importante en el desarrollo de un sistema de software, sin importar la metodología utilizada.

3.2.2 Crosscutting

Los concerns que entrecruzan otros concerns son denominados *"crosscutting concerns"*. Éstos son responsables de producir representaciones entrelazadas; las cuales son difíciles de entender y mantener. Ejemplos de este tipo de propósito en la implementación son los de distribución y sincronización de código que no puede ser encapsulado en una clase y en general se distribuye en varías clases.

La metodología POO fue desarrollada como respuesta a la necesidad de modularizar los concerns de un sistema de software. La realidad es que aunque POO es bueno a la hora de modularizar *"core concerns",* ésta falla cuando se tratan de *"crosscutting concerns"*.

En AOP (Programación orientada a aspectos) los crosscutting concerns son modularizados por su rol bien definido en el sistema, implementando cada rol en su propio módulo y debilitando el acoplamiento entre módulos.

Los crosscutting concern trabajan en contra de la modularización.

Existen dos tipos de crosscutting concern:

- **Código Tangling (enredado o entrelazado)**: es causado cuando un módulo implementa varios concerns simultáneamente. Un desarrollador usualmente considera concerns como: lógica de negocio, performance, sincronización, logueo, seguridad, etc. que son implementados en el mismo módulo.
- **Código Scattering (esparcido o disperso):** Es causado cuando una idea es implementada en múltiples módulos. Por ejemplo, en un sistema que usa una Base de datos, el concern de performance puede afectar todos los módulos que acceden a la BD.

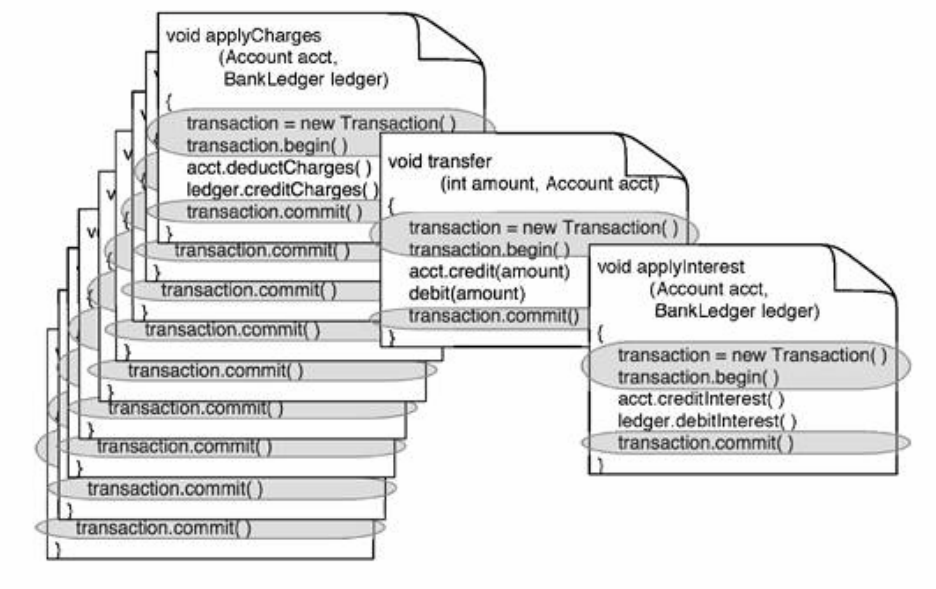


Ilustración 1: Código Scattered

La Ilustración 1 muestra un típico ejemplo en el cual existe código de acceso a una base de datos repetido en diferentes métodos de un objeto.

3.2.3 Problemas de modularizacion

Los códigos Tangling y Scattering juntos impactan en el diseño y desarrollo de software en las siguientes formas:

- **Pésima trazabilidad**: Implementaciones simultáneas de varios concerns acompleja la implementación de un concern; esto causa dificultad en el seguimiento de requerimientos a su implementación y viceversa.
- **Poca productividad**: La implementación simultánea de múltiples concerns mueve el foco del concern principal (el de la lógica de negocio) a los concerns secundarios. La falta de foco empeora la productividad porque los desarrolladores tiene que dejar de lado el objetivo primario para solucionar los crosscutting concern.
- **Bajo rehúso de código**: Si un módulo esta implementando múltiples concerns, otros sistemas que requieren funcionalidad similar no van a ser capaces de usar fácilmente el módulo.
- **Baja calidad**: el código tangling dificulta el examen de código y en consecuencia la localización de errores, y la corrección del código. Por ejemplo revisar el código de un módulo que implementa varios concerns va a requerir la participación de un experto en cada uno de los concerns.
- **Dificultad en la evolución**: Una perspectiva incompleta y limitada resulta en un diseño que solo alcanza los concerns actuales. Cuando un nuevo requerimiento llega este requiere una reimplementación. Esto se debe a que

la implementación no esta modularizada e implica la modificación de varios módulos. Modificar cada subsistema para tales cambios puede llevar a inconsistencias; lo cual requiere mucho esfuerzo en testeo para asegurar que los cambios no introducen errores.

3.3 Aspectos

Un aspecto es una unidad modular diseñada para implementar un concern. Una definición de aspectos puede contener algo de código y las instrucciones de donde, cuando, y cómo invocarlos. Dependiendo del lenguaje de aspectos, los aspectos pueden ser construidos jerárquicamente, y el lenguaje puede proveer mecanismos separados para definir un aspecto y especificar su interacción con el sistema subyacente.

3.3.1 Aspects Oriented Software Development (AOSD)

La comunidad de Desarrollo Software Orientado a Aspectos (Aspect Oriented Software Development, AOSD) ha estado estudiando durante más de una década cómo incrementar y mejorar la expresividad de los paradigmas OO. Las principales propuestas de AOSD se basan en la idea de desarrollar un software de mayor calidad mediante la separación de propósitos, especificando cada uno de ellos de forma separada y las relaciones existentes entre los mismos, para posteriormente, utilizando un mecanismo adecuado, componerlos formando un sistema completamente coherente.

Actualmente, la programación orientada a objetos constituye el paradigma dominante en el desarrollo de software. Así, se pueden encontrar distintas metodologías, herramientas de análisis y diseño y lenguajes de programación OO. La programación OO ha hecho posible el desarrollo de sistemas y aplicaciones de una cierta complejidad sin dejar de lado el mantenimiento de un código comprensible y estructurado. Sin embargo, la OO aún presenta algunas limitaciones. Los investigadores OO ven como ciertos aspectos de los sistemas que implementan no pueden ser separados de una manera clara en objetos; aunque se pueda encontrar un diseño orientado a objetos, esté sufre de subre-ingeniería. El código que implementa esos aspectos se encuentra esparcido y/o anidado en diferentes elementos o estructuras. Para los desarrolladores, resulta complicado centrarse en esos propósitos, de manera que el mantenimiento y la adaptabilidad del software de cierto tamaño se convierte en un proceso de una complejidad importante.

Básicamente, AOSD reduce las principales limitaciones de la orientación a objetos con respecto a este problema de modularidad. Desde el punto de vista del desarrollador, es evidente que algunas de las características más importantes del software son la facilidad de comprensión de su código y la flexibilidad del mismo para adaptarse a las extensiones y cambios que puedan producirse en los requisitos para los que fue diseñado. Para conseguir que el software presente estas características, es importante conseguir mantener una buena modularidad del mismo. Si se analiza un sistema OO, se pueden encontrar propósitos que entrecruzan el sistema, es decir, propósitos que no aparecen modelados en una sola entidad de primer nivel. Un ejemplo de este tipo de propósitos es el sistema de logging[3] utilizado en Tomcat[4], donde dicha funcionalidad se encuentra esparcida a través de todo el sistema y no localizada en una

[3] Loggin se refiere al proceso de bitácora, en el cual se asientan los pasos realizados.

[4] Apache Tomcat es un contenedor de Servlets. Los Servlets es generar páginas Web de forma dinámica a partir de los parámetros de la petición que envíe el navegador Web.

única entidad del lenguaje. Se puede comprobar, por tanto, cómo en los sistemas OO de una cierta complejidad se encuentran a menudo con problemas de modularidad.

3.3.2 Programación Orientada a Aspecto (AOP)

AOP complementa las metodologías existentes de programación como la programación orientada a objetos (POO) y la programación procedural, aumentando la capacidad de expresión permitiendo modularizar los crosscutting concerns. Con AOP, implementamos los core concerns usando la metodología base elegida, por ejemplo POO. Los Aspectos del sistema encapsulan los crosscutting concerns; estos estipulan como los módulos del sistema son ensamblados para conformar el sistema final.

3.3.2.1 Metodología AOP

De muchas formas, desarrollar un sistema usando AOP es similar a desarrollar un sistema usando otras metodologías: identificar los concerns, implementarlos, y formar el sistema final combinando éstos. Se definen tres pasos comunes:

- **Descomposición de Aspectos**: En este paso, se descomponen los requerimientos para identificar los crosscutting y los propósitos principales (core concerns).
- **Implementación de Concern**s: Se implementa cada concern de manera independiente. De esta forma, los desarrolladores pueden implementar, por ejemplo, las unidades de lógica de negocio, de logueo, de persistencia y de autorización.
- **Recomposición de Aspectos**: Finalmente, se especifican las reglas de recomposición para los módulos, o aspectos. Este proceso de recomposición, conocido como *weaving*, usa esta información para componer el sistema final.

El cambio fundamental que AOP provee es la preservación de la independencia de los concerns cuando son implementados. La implementación puede ser fácilmente ensamblada al correspondiente concern, resultando en un sistema que es más simple, fácil implementar y más adaptable a los cambios.

3.3.2.2 Beneficios

Las críticas de AOP usualmente hablan de la dificultad de entenderlo. Y en efecto AOP toma un poco de tiempo, paciencia y práctica para aprender. Sin embargo, la razón principal, detrás de la dificultad, es la novedad de la metodología. AOP demanda pensar el diseño e implementación del sistema de una forma nueva.

Algunos beneficios son:

- **Responsabilidades claras de los módulos**: AOP permite a un módulo responder sólo a los a sus requerimientos y no tener en cuenta a los crosscutting concern. Por ejemplo, un módulo que accede a la Base de Datos, ya no es responsable de gestionar una conexión.
- **Alta modularización**: AOP provee un mecanismo para obtener concerns con un mínimo acoplamiento entre ellos. En consecuencia, tenemos un sistema con mucho menos código duplicado. Las implementaciones modularizadas dan como resultado sistemas fáciles de entender y mantener.

- **Evolución simplificada**: AOP modularíza los aspectos y permite a los core concerns ignorar los aspectos. Agregar una funcionalidad es ahora una forma de incluir un nuevo aspecto y no requiere cambios en el modulo core. Además cuando agregamos un nuevo modulo core al sistema, los aspectos existentes lo cruzan, ayudando a una evolución coherente.
- **Retrazo de decisiones de diseño**: Los arquitectos pueden posponer la toma de decisiones de diseño para futuros requerimientos porque es posible implementarlos como aspectos separados. Ellos ahora pueden enfocarse en los requerimientos cores actuales del sistema. Los nuevos requerimientos de una naturaleza crosscuting pueden ser manejados con la creación de un nuevo aspecto.
- **Mayor rehúso de código**: La llave para un gran rehúso de código es un bajo nivel de acople en la implementación. Porque AOP implementa cada aspecto como un módulo separado, cada módulo es menos acoplado que su implementación equivalente convencional.
- **Rápida salida a producción**: Retrazar las decisiones de diseño permiten un ciclo de diseño mas rápido. Una clara separación de responsabilidades permite una mejor asignación de desarrolladores especializados a módulos. Mayor rehúso de código reduce el tiempo de desarrollo. Una fácil evolución permite una rápida respuesta a nuevos requerimientos. Todo esto conduce a un sistema que es fácil de desarrollar y distribuir.
- **Reducción de costo de implementación**: AOP reduce los costos de implementar las características de crosscutting, ya que evita la modificación de los módulos cada uno por separado. Además permite que cada desarrollador se enfoque en un concern de un módulo particular y haga uso de su especialidad, el costo de los requerimientos core es también reducido.

3.4 Ingeniería de requerimientos

3.4.1 Introducción

Según la definición de Boehm[29] decimos que: "*Ingeniería de Requerimientos es la disciplina para desarrollar una especificación completa, consistente y no ambigua, la cual servirá como base para acuerdos comunes entre todas las partes involucradas y en dónde se describen las funciones que realizará el sistema*"

La ingeniería de requerimientos del software es un proceso de descubrimiento, refinamiento, modelado y especificación. Se refinan en detalle los requerimientos del sistema y el papel asignado al software -inicialmente asignado por el ingeniero del sistema-. Se crean modelos de los requerimientos de datos, flujo de información y control, y del comportamiento operativo.

Tanto el analista como el cliente tienen un papel activo en la ingeniería de requerimientos del software -un conjunto de actividades que son denominadas análisis-. El cliente intenta replantear un sistema confuso, a nivel de descripción de datos, funciones y comportamiento, en detalles concretos. El analista actúa como un interrogador, como consultor, como persona que resuelve problemas y como negociador.

El análisis y la especificación de los requerimientos puede parecer una tarea relativamente sencilla, pero las apariencias engañan. El contenido de comunicación es muy denso. Abundan las ocasiones para las malas interpretaciones o falta de información.

3.4.2 Análisis de requerimientos

El análisis de los requerimientos es una tarea de ingeniería del software que cubre el hueco entre la definición del software a nivel sistema y el diseño del software. El análisis de requerimientos permite al ingeniero de sistemas especificar las características operacionales del software (función, datos y rendimientos), indica la interfaz del software con otros elementos del sistema y establece las restricciones que debe cumplir el software.

El análisis de requerimientos del software puede dividirse en cinco áreas de esfuerzo que se encuentran en todas las técnicas de desarrollo: (1) reconocimiento del problema, (2) evaluación y síntesis, (3) modelado, (4) especificación y (5) revisión. Inicialmente, el analista estudia la Especificación del Sistema (si existe alguna) y el Plan del Proyecto de Software. Es importante entender el software en el contexto de un sistema y revisar el ámbito del software que se empleó para generar las estimaciones de la planificación. A continuación, se debe establecer la comunicación para el análisis de manera que nos garantice un correcto reconocimiento del problema.

El objetivo del analista es el reconocimiento de los elementos básicos del problema tal y como los percibe el cliente/usuario.

La evaluación del problema y la síntesis de la solución es la siguiente área principal de esfuerzo en el análisis. El analista debe definir todos los objetos de datos observables externamente, evaluar el flujo y contenido de la información, definir y elaborar todas las funciones del software, entender el comportamiento del software en el contexto de acontecimientos que afectan al sistema, establecer las características de la interfaz del sistema y descubrir restricciones adicionales del diseño. Cada una de estas tareas sirve para describir el problema de manera que se pueda sintetizar un enfoque o solución global.

Por ejemplo, un mayorista de recambios de automóviles necesita un sistema de control de inventario. El analista averigua que los problemas del sistema manual que se emplea actualmente son: (1) incapacidad de obtener rápidamente el estado de un componente; (2) dos o tres días de media para actualizar un archivo a base de tarjetas; (3) múltiples órdenes repetidas para el mismo vendedor debido a que no hay manera de asociar a los vendedores con los componentes, etc.

Una vez que se han identificado los problemas, el analista determina qué información va a producir el nuevo sistema y qué información se le proporcionará al sistema. Por ejemplo, el cliente desea un informe diario que indique qué piezas se han tomado del inventario y cuántas piezas similares quedan. El cliente indica que los encargados del inventario registrarán el número de identificación de cada pieza cuando salga del inventario.

Una vez evaluados los problemas actuales y la información deseada (entrada y salida), el analista empieza a sintetizar una o más soluciones. Para empezar, se definen en detalle los datos, las funciones de tratamiento y el comportamiento del sistema. Una vez que se ha establecido esta información, se consideran las arquitecturas básicas para la implementación. Un enfoque cliente/servidor parecería apropiada, pero ¿está dentro del ámbito esbozado en el Plan del Software? Parece que sería necesario un sistema de gestión de bases de datos, pero, ¿está justificada la necesidad de asociación del usuario/cliente? El proceso de evaluación y síntesis continúa hasta que ambos, el analista y el cliente, se sienten seguros de que se puede especificar adecuadamente el software para posteriores fases de desarrollo. A lo largo de la evaluación y síntesis de la solución, el enfoque primario del analista está en el «qué» y no en el «cómo». ¿Qué datos produce y consume el sistema, qué funciones debe realizar el sistema, qué

interfaces se definen y qué restricciones son aplicables?'. , Durante la actividad de evaluación y síntesis de la solución, el analista crea modelos del sistema en un esfuerzo de entender mejor el flujo de datos y de control, el tratamiento funcional y el comportamiento operativo y el contenido de la información. El modelo sirve como fundamento para el diseño del software y como base para la creación de una especificación del software.

3.4.3 Casos de uso

Una vez recopilados los requerimientos, el ingeniero del software (analista) puede crear un conjunto de escenarios que identifiquen una línea de utilización para el sistema que va a ser construido. Los escenarios, algunas veces llamados casos de uso, facilitan una descripción de cómo el sistema se usará.

Todo sistema de software ofrece a su entorno una serie de servicios. Un caso de uso es una forma de expresar cómo alguien o algo externo a un sistema lo usa. Cuando decimos "*alguien o algo*" hacemos referencia a que los sistemas son usados no sólo por personas, sino también por otros sistemas de hardware y software.

Para crear un caso de uso, el analista debe primero identificar los diferentes tipos de personas (o dispositivos) que utiliza el sistema o producto. Estos actores actualmente representan papeles que la gente (o dispositivos) juegan como impulsores del sistema. Definido más formalmente, un actor es algo que comunica con el sistema o producto y que es externo al sistema en sí mismo.

Es importante indicar que un actor y un usuario no son la misma cosa. Un usuario normal puede jugar un número de papeles diferentes cuando utiliza un sistema, por lo tanto un actor representa una clase de entidades externas (a veces, pero no siempre personas) que lleva a cabo un papel.

Porque la obtención de requerimientos es una actividad de evolución, no todos los actores se identifican durante la primera iteración. Es posible identificar actores iniciales durante la primera iteración y actores secundarios cuando más se aprende del sistema. Los actores iniciales interactúan para conseguir funciones del sistema y derivar el beneficio deseado del sistema.

Ellos trabajan directa y frecuentemente con el software. Los actores secundarios existen para dar soporte al sistema que los actores iniciales han dado forma con su trabajo.

Una vez que se han identificado los actores, se pueden desarrollar los casos de uso. El caso de uso describe la manera en que los actores interactúan con el sistema. Jacobson [JAC93] sugiere un número de preguntas que deberán responderse por el caso de uso:

- ¿Cuáles son las principales tareas o funciones que serán realizadas por el actor?
- ¿Cuál es el sistema de información que el actor adquiere, produce o cambia?
- ¿Qué actor informará al sistema de los cambios en el entorno externo?
- ¿Qué información necesita el actor sobre el sistema?

En general, un caso de uso es, simplemente, un texto escrito que describe el papel de un actor que interactúa con el acontecer del sistema.

3.4.4 Técnicas de Ingeniería de requerimientos

Durante la ingeniería de requerimientos, a grande rasgos, las propiedades que el software debe exhibir tienen que ser capturadas. El análisis de información extraída y

el contexto organizacional y operacional asociado da lugar a la síntesis de un conjunto de requerimientos. Estos requerimientos deberían ser, en lo posible, correctos, completos y factibles. Alcanzar estas características típicamente requiere negociaciones y acuerdos con/entre usuarios y otros clientes. El conjunto de requerimiento que emerge de la actividad de análisis necesita ser plasmada en un documento de especificación que comunica los requerimientos a las personas que lo utilizarán para el desarrollo de software. Los requerimientos documentados necesitan ser validados para asegurar que el software que especifican va a satisfacer las necesidades de las personas de las cuales fueron obtenidos los requerimientos.

En el proceso de desarrollo, los requerimientos necesitan ser administrados para controlar los cambios.

Por lo tanto, el proceso de ingeniería de requerimientos fundamentalmente guía el descubrimiento, entendimiento, publicación, verificación, comunicación y administración de requerimientos.

A continuación se citará una breve descripción de algunas técnicas de ingeniería de requerimientos agrupadas en dos grupos: orientadas a aspectos y no orientadas a aspectos.

3.4.4.1 Técnicas de RE (no-AO) contemperarías

3.4.4.1.1 PREview

PREview es una técnica Orientada a Punto de Vista (VOA), como toda técnica VOA, considera el problema desde diferentes perspectivas (llamadas *Puntos de Vista*) poniendo en evidencia responsabilidades, roles, objetivos, etc. de las fuentes de información.

PREview complementa la noción estándar de punto de vista con el de concerns organizacionales: generalización de la noción de objetivo que incluye objetivos organizacionales y restricciones del sistema o del proceso.

Los concerns PREview son identificados al principio del proceso de RE y descompuestos en preguntas, restricciones o requerimientos. Los *puntos de vista* son identificados y registrados usando plantillas provistas de *puntos de vista*.

Durante el análisis de requerimientos, preguntas asociadas con concerns deben ser asociadas a todos los *puntos de vista* y respondidas por las fuentes de los *puntos de vista*. Se continúa con la detección de interacción y la resolución de inconsistencias entre *puntos de vista.* De esta forma PREview revela como los concerns afectan los *puntos de vista* y requerimientos. Los concerns discriminados son utilizados para la toma de decisiones en el mapeo de concerns a módulos funcionales, arquitectura u otras decisiones.

PREview sugiere que la funcionalidad siempre es negociable, en oposición a los concerns (ej. Seguridad en un sistema crítico).

Por lo tanto, PREview usas concern como medio en el descubrimiento de requerimientos y *puntos de vista* para el descubrimiento actual de requerimientos.

3.4.4.1.2 Non-Functional Requirements (NFR) Framework

El concepto central de NFR Framework es objetivo-liviano: un objetivo que no tiene una definición clara ni un criterio preciso o para determinar cuando este ha sido

satisfecho. Esta definición se ajusta bien a los requerimientos no funcionales (los cuales son representados como *objetivos-livianos*); como tal un requerimiento puede tener diferentes significados y criterios de satisfacción para personas diferentes, y aún para la misma persona trabajando en diferentes proyectos.

En el NFR framework los *objetivos-livianos* representan tres tipos de entidades: restricciones en el sistema, soluciones concretas de diseño o implementación para restricciones, análisis razonado o explicaciones para las decisiones (llamadas demanda).

Los *objetivos-livianos* son descompuestos (o refinados) en una descendencia de objetivos que es relativa a sus padres a través de una relación es-un. Los descendientes también contribuyen a sus padres tanto positivamente como negativamente. Los métodos de refinamiento son patrones y guías para la descomposición basada en las experiencias pasadas y de los ingenieros y en el conocimiento del dominio.

El framework también provee un procedimiento de evaluación usado para determinar el grado en el cual el objetivo-liviano inicial es satisfecho. La evaluación depende de satisfacción de los descendientes y de la contribución a sus padres. Los descendientes peden tener diferentes prioridades, las cuales pueden ser usadas para la toma de decisiones de los *trade-off* y para la resolución de conflictos.

Las relaciones de conflicto/soporte entre requerimientos no funcionales (ej. Costo vs. Calidad / Disponibilidad vs. Fiabilidad), también llamadas correlaciones, son colectadas y catalogadas.

Este catalogo es utilizado para examinar el impacto del cruce de los objetivos-livianos durante el análisis de *trade-off* y la selección de soluciones alternativas.

3.4.4.1.3 Problem Frames

La técnica de propone descomponer problemas complejos en conjuntos estructurados de clases de sub-problemas más sencillos; clases de subproblemas comunes. (Las clases de los problemas comunes deberían ser identificadas desde el cuerpo del análisis del problema de forma similar a los *patrones de diseño*)

La combinación descripción/solución para los sub-problemas sirve para la descripción/solución del problema original. Las amplias características de las clases de problemas comunes y las interacciones de los dominios del problema real con el sistema previsto son extraídos en *marcos de problema*. PF sugiere que una vez que los marcos principales son conocidos para los desarrolladores, es más fácil reconocer y guiar las variaciones de estas clases de problemas y anticipar las dificultades así como también producir soluciones eficientes para estas.

Un marco del problema es representado por un diagrama del problema que consiste en la máquina (software), unos o más dominios, el requisito y los fenómenos compartidos entre ellos. Cada marco del problema se suple con un marco de concern que diga qué clases de descripciones son necesarias para entender adecuadamente el problema y cuales son los pasos lógicos para su solución.

Los tipos de problemas guiados con PF son los usualmente llamados requerimientos funcionales. Sin embargo, PF también reconoce la importancia de concerns no funcionales, lo más importante para PF es la composición de concerns.

Es necesario combinar marcos simples del problema en los marcos compuestos que representan un análisis de problema real y complejo.

En consecuencia, es deseable reconocer concerns relacionado a cada marco de problema y construir un repertorio de concerns similares. PF incluso demuestra como conseguir fiabilidad como un marco de problema separado.

3.4.4.2 Técnicas Orientadas a Aspectos

AOSD anima a extender las técnicas tradicionales de desarrollo de software. Debajo presentamos dos técnicas AORE y describimos como ellas extienden el trabajo de RE.

3.4.4.2.1 AORE con Arcade

En un modelo de proceso general de RE se desarrolla para separar requerimientos aspectuales y no aspectuales, como así también sus reglas de composición. Se realiza una instanciación concreta del modelo, usando puntos de vista y un lenguaje XML de composición, junto con una herramienta de apoyo, llamada Arcade.

En la técnica Arcade, los requerimientos aspectuales son similares a los concerns de PREview. La noción de punto de vista de PREview es también utilizada para la obtención de requerimientos. Los requerimientos aspectuales se presentan simultáneamente (crosscut) en los puntos de vista. Ambos son representados utilizando un framework basado en XML. El documento XML es también utilizado para definir las reglas de composición que emplea acciones informales (especificas de concern) y operaciones reflejando como los requerimientos aspectuales afectan grupos de puntos de vista de requerimientos que cortan (crosscut).

El conjunto de reglas de composición es extensible; reglas específicas al nuevo problema pueden ser creadas cuando sea necesario.

La validación de composición de relaciones, y proceso de detección de puntos de interacción y trade-off es asistida vía la herramienta Arcade. Finalmente, los requerimientos aspectuales son mapeados a decisiones, funciones o aspectos de diseño.

La contribución más valorada y novedosa de esta técnica es la habilidad de separar y luego componer requerimientos crosscutting y no crosscutting, un conjunto de operadores de composición y la muy original representación de requerimientos en formato XML. También proporciona un cuidadoso procedimiento para la identificación y resolución de conflictos.

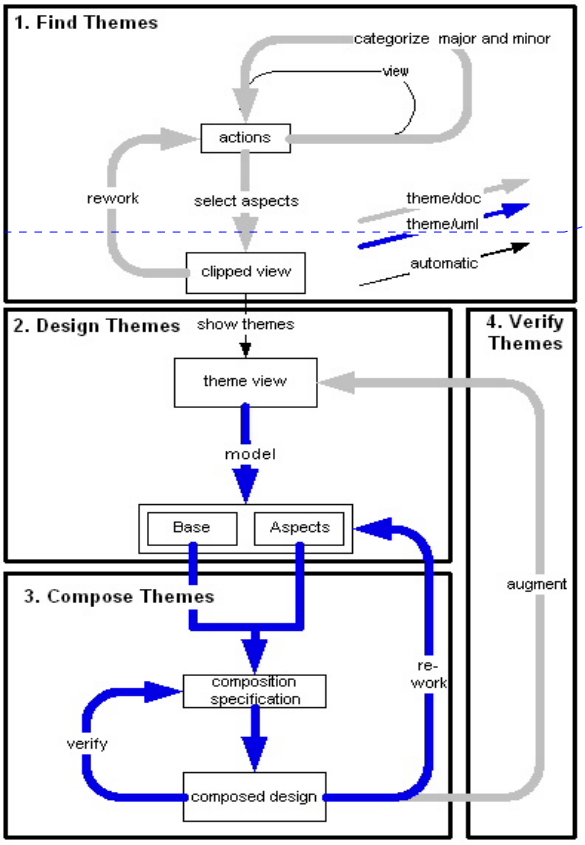
El soporte para el mapeo de requerimientos, a pesar de ser muy útil, carece el rigor y la minuciosidad de la inspección del análisis de concern PREview o NFR.

Por otro lado, trabajos recientes sobre Arcade permiten un trazado y verificación de requerimientos y trade-offde las etapas posteriores del ciclo de vida del software.

Así, utilizando la terminología AO, en Arcade los concerns sirven como aspectos que cruzan (crosscut) puntos de vista; los puntos de vista sirven como joinpoints; y los operadores de composición permiten la cuantificación sobre los puntos de vista de los requerimientos durante la composición.

3.4.4.2.2 Theme/Doc

Theme/Doc es la parte RE de la técnica Theme. Theme/Doc soporta identificación y "análisis de aspectos en la documentación de requerimientos" donde los aspectos se manifiestan a si mismos como descripciones de comportamiento que son



Así, esta orientada a las etapas posteriores del RE, donde por lo menos un documento inicial de requerimiento está disponible para el análisis léxico.

La técnica es soportada por la herramienta Theme/Doc la cual es absolutamente central al método porque su análisis y pasos se basan en las visualizaciones desde la herramienta. La herramienta recibe como entrada el documento de requerimientos junto con una lista de palabras de acciones seleccionadas por el ingeniero. Las palabras de acción y requerimientos son presentados en la vista de acciones como cajas; las acciones se ligan con una línea a los requisitos en los cuales aparecen. Si una acción es ligada a más de un requerimiento puede reflejar la presencia de un potencial crosscutting. El ingeniero de los requisitos revisita los lazos, reagrupa las palabras y los requisitos y poda los lazos entre las palabras de la acción y los requisitos a los cuales proporcionan funcionalidad secundaria. Los lazos podados son reemplazados por otros decorados. Este proceso conduce a agrupar la funcionalidad de requerimientos en funcionalidad principal (base) y secundaria (aspectual), llamados themes. Los themes son organizados de acuerdo al orden de composición: el *theme* base forma el primer nivel, los *themes* que solo se ligan a éste forman el segundo nivel; los *themes* que cruzan al *Theme* base y el segundo nivel forman el tercer nivel, y así sucesivamente.

La herramienta provee varias vistas para ayudar al diseño y mapeo de la vistas de requerimientos a *Theme*/UML.

En consecuencia, algunos *themes*, que encapsulan funcionalidad aspectual, cruzan otros *themes* donde los requerimientos individuales en los *theme* sirven como *joinpoint*; el orden de composición para los *theme* es reflejado por el nivel de los *themes* en la vista de acción.

Los trabajos actuales en la técnica Theme/Doc están enfocados en características de escalabilidad de la técnica.

En la figura se muestra la secuencia de pasos necesarios en el paradigma:

Análisis: El primer paso en la técnica es realizar el análisis de los requerimientos para identificar los *themes*. Esta operación consiste en mapear

Comentario [11]: descriptions of behaviours that are intertwined, and woven throughout" [6].

requerimientos a concerns. Theme/Doc permite ver la relación de comportamientos. Estas relaciones exponen la diseminación de concerns y permite identificar aspectos.

Diseño: Luego se diseñan los *themes* usando Theme/UML. Utilice los temas que usted encontró con Theme/Doc para identificar clases y métodos potenciales, después complete los detalles del diseño y realize los cambios que son necesarios beneficiar el diseño. Otros aspectos emergerán durante diseño técnico detallado.

Composición: Luego se especifican como los modelos Theme/UML deberían ser recombinados. En algunos casos, algunas de las vistas Theme/Doc van a ayudar a determinar como los *themes* se relacionan con otros.

3.5 Early Aspects

Las técnicas AOSD proveen medios semánticos para la identificación, modularización, representación y composición de crosscutting concerns tal como la seguridad, movilidad y restricciones de la aplicación. Estas propiedades tienen un efecto amplio sobre otros requerimientos o componentes de arquitectura.

AOSD ha proveído un entendimiento mejorado y complementario del principio de separación de concerns por la separación y especificación explicita de concerns; por ejemplo: aspectos, que tiende a presentarse sobre múltiples componentes. Como resultado, un número creciente de proyectos de software explícitamente consideran la separación de aspectos en la aplicación para alcanzar una mejor modularidad. Un análisis de los sistemas muestra que muy seguido los mismos aspectos son implementados una y otra vez. En el diseño de sistemas distribuidos, por ejemplo, concerns como sincronización, recuperación, seguridad, logging, y monitoreo son implementados; al parecer muchos de estos concerns son crosscutting y deberían ser implementados como aspectos. A pesar del hecho que muchos aspectos re aparecen en varias aplicaciones ninguna tentativa semántica parece haber sido propuesta para capturar y reutilizar estos aspectos. Los mismos aspectos son implementados para el caso en particular o en el mejor de los casos son reutilizados oportunamente adaptando las especificaciones existentes de los aspectos.

La reutilización oportuna puede funcionar en un modo limitado para programadores individuales o pequeños grupos. Sin embargo, ésta metodología no es fácilmente escalable en aplicaciones grandes y por lo tanto se requiere alguna tentativa semántica para la reutilización de software. La motivación generalmente reconocida para la reutilización sistemática son (válidas también para aspectos): disminuir el tiempo del ciclo de desarrollo, incrementar la calidad, y disminuir el costo de desarrollo.

Desafortunadamente, las metodologías AOSD convencionales se han enfocado principalmente en identificar aspectos a nivel de desarrollo (programación) y no se ha prestado atención en el impacto de los crosscutting concerns en las primeras fases del desarrollo de software.

La iniciativa *Early Aspects* se concentra en el manejo de las propiedades crosscutting en las primeras etapas de la ingeniería de requerimientos y del diseño de la Arquitectura.

Si los *Early Aspects* no son efectivamente modularizados, no es posible deducir sus efectos en el sistema. Además, la ausencia de modularizacion de tales propiedades puede resultar en una larga onda de efectos sobre otros requerimientos o componentes de arquitectura en su evolución.

Ha habido un trabajo significativo en la separación de concerns en las comunidades de ingeniería de requerimientos y diseño de arquitectura. Por ejemplo, viewpoints, use case, goals y modelos de análisis de arquitectura. Sin embargo, estas

técnicas no es concentran explícitamente en crosscutting concerns. El trabajo en Early Aspects complementa estas técnicas completándolas con medios semánticos para manejar tales concerns.

3.5.1 Análisis del Dominio

La reutilización de aspectos requiere sobre todo la identificación y la explotación sistemática de especificaciones comunes relacionadas de un aspecto.

Uno de los procesos comúnmente adoptados para la reutilización sistemática es el *análisis del dominio* el cual es adoptado ampliamente en la comunidad. El *análisis del dominio* puede ser definido como el proceso de identificación, captura y organización del conocimiento del dominio de un problema en estudio con el propósito de hacerlo reutilizable cuando se crea un nuevo sistema. El resultado de un análisis de dominio es un modelo de dominio el cual puede ser reutilizado para implementar varias aplicaciones.

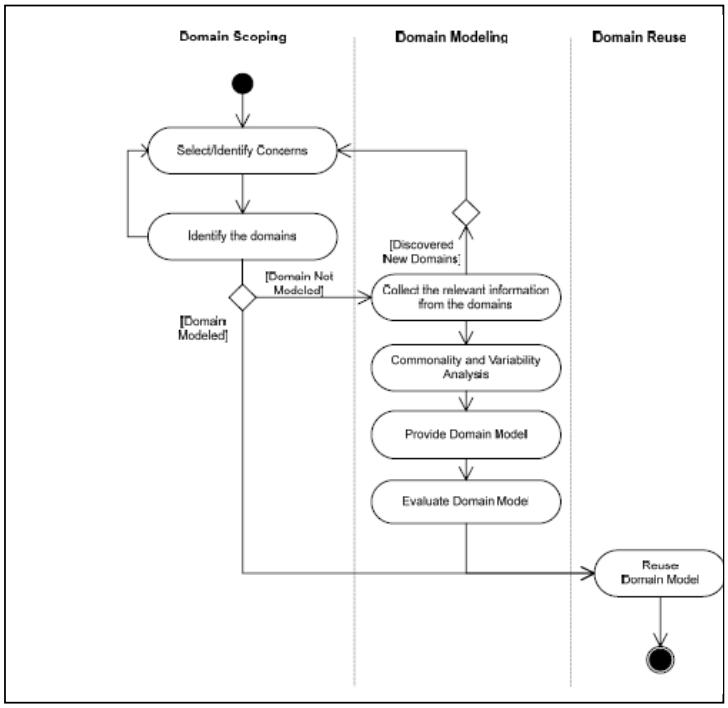


Ilustración 2: Estructura común de los análisis de dominio

Procesos convencionales de análisis de dominio consisten generalmente de los procesos de *definición de dominio* y de *modelado de dominio*. La *definición del dominio* identifica el interés del dominio, los clientes y sus objetivos, y el alcance del dominio. El *modelado del dominio* es la actividad para representar el dominio. El modelo del dominio puede ser representado en diferentes formas tales como lenguajes orientados a objetos, especificaciones algebraicas, reglas, modelos conceptuales, etc. Un modelo de dominio es usado como base para la ingeniería de componentes con la intención de usarlos en múltiples aplicaciones dentro del dominio.

3.5.1.1 Motivaciones para el Análisis de Dominio Orientado a Aspectos

Podemos encontrar dos razones básicas para le análisis de dominio en el desarrollo orientado a aspectos:

Identificar aspectos desde la solución del dominio: La definición general de aspectos, concerns que tienden a presentarse en diferentes módulos, puede ser aplicada a través de todo el ciclo de vida. Para evitar problemas relacionados al *crosscutting*, como mantenimiento reducido, estos crosscutting concerns deben ser identificados y especificados separadamente.

La identificación de aspectos durante el análisis de requerimientos no siempre será suficiente para cubrir todos los aspectos. Esto es porque el análisis de requerimiento se concentra en el dominio del problema en vez del dominio de la solución.

Por otro lado, identificar aspectos en el diseño de la arquitectura puede ser demasiado tarde. Para completar el proceso de identificación de aspectos nosotros pensamos que es necesario también considerar aspectos en el dominio de la solución. El dominio de la solución ha sido definido como el dominio que incluye la solución para los requerimientos que son parte del dominio del problema.

Implementar aspectos basados en los modelos del dominio de la solución: Antes de implementar aspectos es necesario un total entendimiento de su estructura, propiedades y comportamiento. Caso contrario, no habrán bases sólidas para garantizar que los aspectos son definidos e implementados de manera adecuada. Sin embargo, en prácticas actuales de AOSD el análisis de aspectos no es explícito y la implementación de ellos depende sobre todo de la experiencia y del conocimiento del programador.

3.5.2 Ingeniería de requerimientos y Early Aspects

La Ingeniería de Requerimientos Orientado a Aspectos tiene como objetivo tratar las propiedades crosscutting que se encuentran distribuidas de una manera sistemática. De esta manera intenta facilitar el razonamiento eficaz acerca de su impacto en el dominio del problema como así la modularización de tales propiedades en la especificación de requerimientos para que ellas puedan ser efectivamente mapeadas a la solución.

Un aspecto en el nivel de requerimientos es una propiedad distribuida, representada por un requerimiento simple o un conjunto de requerimientos coherentes (requerimientos de seguridad), que afectan múltiples requerimientos en el sistema de modo que:

- Puede condicionar el comportamiento de los requerimientos afectados
- Puede influenciar los requerimientos afectados de modo que alteen su comportamiento específico.

Por ejemplo, el requerimiento de seguridad puede condicionar un requerimiento brindando acceso a ciertos tipos de datos en el sistema de manera que sólo un conjunto de usuarios certificados puedan tener acceso a tales datos.

Similarmente, otro requerimiento de seguridad puede influenciar los requerimientos de comunicación alterando el comportamiento usual imponiendo restricciones de encriptación.

El tratamiento de aspectos en este nivel, requiere cuatro pasos básicos:

Identificación

No es simple la identificación de aspectos. A veces acudimos a nuestro conocimiento del dominio del problema para determinarlos, pero un documento de requerimiento puede describir algunos aspectos atípicos.

Este paso intenta localizarlos, para poder manejarlos efectiva y consistentemente.

En el momento de buscar aspectos, miramos:

- **Términos aspectuales**: Estos son atributos de calidad (seguridad, consiste disponibilidad, etc.). Por ejemplo: "una vez completo". Los ingenieros de requerimientos pueden identificar estos términos utilizando técnicas de búsqueda simple o con herramientas hechas especialmente para el análisis de requerimientos orientado a aspectos, ej AORE y AE-Miner.
- **Impacto de requerimientos**: esto describe la influencia de un concen sobre otro. Técnicas de visualización (Theme/Doc, EA-miner, etc) pueden resaltar tales requerimientos porque ellos muestran cual requerimiento describe tal concern, revelando cuales concerns se solapan.
- **Distribución de concern**: Estos son términos, conceptos, o comportamiento que aparecen en múltiples (bien organizados) requerimientos. Tales requerimientos tienen impacto global.

Captura

En este paso, ingenieros en requerimientos organizan de tal forma que cada requerimiento describe un solo concern.

Composición

Formalmente se indica el estado del impacto de los requerimientos para especificar como los concerns deben se compuestos.

Análisis

Se analizan los concerns compuestos para localizar los trade-off e identificar conflictos e inconsistencias.

3.5.3 Aspectos y el diseño de la arquitectura

Una arquitectura de Software son las estructuras del sistema de Software; estas estructuras comprometen elementos, las propiedades de estos elementos y las relaciones entre ellos.

Vistas

Las técnicas modernas de arquitectura representan estructuras y otra información estructural como un conjunto de vistas, donde cada una muestra un tipo particular de elementos y las correspondientes relaciones entre los elementos. Los tres tipos básicos de vistas son los módulos, componente y conector, y asignación.

Algunas de las vistas son híbridas, cubriendo información perteneciente a mas de una de estas familias.

Un arquitecto diseña las estructuras nativas para cada vista para alcanzar la calidad del sistema deseada y los objetivos propuestos de comportamiento. Un proceso de arquitectura basado en vistas revelará similitudes entre vistas que el arquitecto puede capturar como aspectos.

En consecuencia, consideramos que las vistas en si mismas, más que mostrar elementos y relaciones de software, forman la descomposición dominante de una arquitectura. La idea de la técnica arquitectónica del vista múltiples es alejarse de la idea de que los sistemas tienen solamente un estructura o estilo que determina la descomposición. La descomposición es jerárquica, pues las vistas tienen subestructuras. El arquitecto puede descomponer vista en paquetes que muestran pequeñas partes del sistema; que según la granularidad, nos dan un panorama del sistema.

Podemos utilizar aspectos para capturar conceptos que surgen durante el diseño arquitectural tal como:

- Asegurar que todos los elementos comparten comportamiento común, tal como protocolos de inicio, recuperación en caso de falla, reporte ante un error, etc.
- Especificar una estructura común.
- Dando interfaces a elementos que tienen partes en común, tal como un conjunto de métodos común por el cual la funcionalidad de los elementos puede ser invocada.

Actividades

En los requerimientos, las principales actividades concernientes a los aspectos arquitecturales son identificación, captura, composición y análisis.

Identificación

Aunque no existan técnicas de automatización para identificar aspectos arquitecturales, el arquitecto puede utilizar heurísticas para buscar crosscutting concerns arquitecturales durante las actividades normales del desarrollo basado en arquitectura.

Analizar los casos de negocio del sistema, el arquitecto utiliza los casos del sistema para identificar a los interesados y orientar los objetivos en la construcción del sistema. Mas allá de lo que una especificación de un requerimiento usualmente captura, estos objetivos pueden representar el desarrollo de las ambiciones de la organización subyacentes al sistema. Un ejemplo, es el deseo de una organización de re utilizar un subsistema en desarrollo dentro del actual proyecto en futuros sistemas. Otro ejemplo, es el deseo de utilizar la tecnología JEE porque el personal se encuentra capacitado en esta. Estos objetivos pueden eventualmente ser definidos dentro de atributos de calidad que restringirán la arquitectura, o solo serán representado en restricciones de diseño. El análisis de casos de negocio puede identificar requerimientos superpuestos, estos son candidatos a ser aspectos.

Seleccionar patrones y tácticas, muchas de las técnicas prescriptas al diseño de la arquitectura comienzan con la elección del patrón arquitectural apropiado. Los patrones arquitecturales representan concerns satisfechos usando un grupo de elementos arquitecturales.

Captura

El arquitecto captura aspectos en el documento de arquitectura, el cual podemos estructurar como un conjunto de vista mas información que liga las vistas. Si los aspectos arquitecturales se aplican a lo largo de un conjunto de elementos, en una vista

simple o sobre varias vistas, podemos factorearlos y documentarlos por separado- en una vista de aspecto. Se pueden documentar aspectos en la vista de aspectos utilizando el mismo lenguaje y notaciones que son usadas para describir las partes no aspectuales de arquitectura correspondiente.

- Si un aspecto captura comportamiento común a un conjunto de elementos, puede ser representado por medio de cualquier lenguaje que el arquitecto utiliza para expresar comportamiento no aspectual: maquina de estados, diagrama de secuencia, etc.
- Si un aspecto captura estructuras comunes, el arquitecto representa la subestructura de la misma manera que el resto de la información es capturada: diagrama de clases UM.
- Si un aspecto captura parte de una interfaz común a un conjunto de elementos, el arquitecto puede expresarlo de la misma forma que las interfaces restantes son capturadas: IDL, por ejemplo.

Las vistas convencionales tienen lugares (secciones en la documentación) para mostrar la estructura, comportamiento, interfaces, etc. Una vista de aspecto puede usar exactamente la misma organización. La única diferencia es que los aspectos van a ser abstractos con respecto a los elementos específicos del sistema. Por ejemplo, en una vista de aspecto, el comportamiento es especificado no para un elemento sino para un conjunto de elementos a los que le es ligado más adelante.

Composición

Además de producir vistas, un arquitecto debe producir documentación que explica como las vistas están relacionadas.

Utilizar aspectos requiere la definición de una relación (mapping) entre los elementos de una vista de aspecto y los elementos o subestructuras en las diferentes vistas a las cuales se aplican. Esta es una analogía al concepto de “one point”.

Análisis

El análisis arquitectural en el contexto de early-aspects incluye la evaluación de la conveniencia de los aspectos que el arquitecto ha identificado.

4 Requerimientos en aplicaciones Web

4.1 Introducción al diseño

El desarrollo de aplicaciones Web involucra decisiones no triviales de diseño e implementación que inevitablemente influyen en todo el proceso de desarrollo, afectando la división de tareas. Los problemas involucrados, como el diseño del modelo del dominio y la construcción de la interfaz de usuario, tienen requerimientos disjuntos que deben ser tratados por separado.

El alcance de la aplicación y el tipo de usuarios a los que estará dirigida son consideraciones tan importantes como las tecnologías elegidas para realizar la implementación. Así como las tecnologías pueden limitar la funcionalidad de la aplicación, decisiones de diseño equivocadas también pueden reducir su capacidad de extensión y rehúsabilidad. Es por ello que el uso de una metodología de diseño y de tecnologías que se adapten naturalmente a ésta, son de vital importancia para el desarrollo de aplicaciones complejas.

Existen en la actualidad tecnologías ampliamente usadas para el desarrollo de aplicaciones Web, pero muchas de ellas obligan al desarrollador a mezclar aspectos conceptuales y de presentación. Esto sucede principalmente con aquellas tecnologías no basadas en objetos.

La elección de tecnologías complejas demora el proceso e incrementa los costos, pero en ocasiones permite adecuarse a metodologías de diseño más fácilmente. Tal es el caso de las tecnologías orientadas a objetos, las cuales tienden a demorar el desarrollo en etapas tempranas. El tiempo de desarrollo en la actualidad es crítico, tanto por razones de marketing como por límites en el presupuesto y los recursos, pero la adopción de estas tecnologías hace que el mantenimiento se transforme en una actividad más simple, la división en capas sea tarea natural. Las aplicaciones hipermedia han evolucionado en los últimos años y se han concentrado mayormente en la Web. Las antiguas aplicaciones distribuidas en CD's dieron lugar a aplicaciones dinámicas, de constante actualización e incluso personalizables, capaces de adaptarse a los tipos de usuarios y en casos avanzados, a cada usuario en particular. Estas características encuentran el medio ideal en la Web, ya que de otra forma sería costoso su mantenimiento y evolución.

La complejidad del desarrollo ocurre a diferentes niveles: dominios de aplicación sofisticados (financieros, médicos, geográficos, etc.); la necesidad de proveer acceso de navegación simple a grandes cantidades de datos multimediales, y por último la aparición de nuevos dispositivos para los cuales se deben construir interfaces Web fáciles de usar. Esta complejidad en los desarrollos de software sólo puede ser alcanzada mediante la separación de concerns de modelización en forma clara y modular.

La metodología OOHDM (Object Oriented Hypermedia Design Method), ha sido utilizada para diseñar diferentes tipos de aplicaciones hipermedia como galerías interactivas, presentaciones multimedia y aplicaciones Web. El éxito de esta metodología es la clara identificación de los tres diferentes niveles de diseño en forma independiente de la implementación.

La justificación de tanto trabajo puede encontrarse en cualquier aplicación que requiera navegación: en términos de programación orientada a objetos, si los elementos por los que se navega son los del diseño conceptual se estaría mezclando la funcionalidad hipermedia con el comportamiento propio del objeto. Por otro lado, si los

nodos de la red de navegación tienen la capacidad de definir su apariencia, se estaría limitando la extensión de la aplicación para ofrecer nuevas presentaciones del mismo elemento y eventualmente se estaría dificultando la personalización de la interfaz.

Es necesario, entonces, mantener separadas las distintas decisiones de diseño según su naturaleza (conceptual, navegacional, de interfaz) y aplicar las tecnologías adecuadas a cada capa en el proceso de implementación.

4.1.1 Introducción a OOHDM

Las metodologías tradicionales de ingeniería de software, o las metodologías para sistemas de desarrollo de información, no contienen una buena abstracción capaz de facilitar la tarea de especificar aplicaciones hipermedia. El tamaño, la complejidad y el número de aplicaciones crecen en forma acelerada en la actualidad, por lo cual una metodología de diseño sistemática es necesaria para disminuir la complejidad y admitir evolución y rehúsabilidad.

Producir aplicaciones en las cuales el usuario pueda aprovechar el potencial del paradigma de la navegación de sitios Web, mientras ejecuta transacciones sobre bases de información, es una tarea muy difícil de lograr.

En primer lugar, la navegación posee algunos problemas. Una estructura de navegación robusta es una de las claves del éxito en las aplicaciones hipermedia. Si el usuario entiende dónde puede ir y cómo llegar al lugar deseado, es una buena señal de que la aplicación ha sido bien diseñada.

Construir la interfaz de una aplicación Web es también una tarea compleja; no sólo se necesita especificar cuáles son los objetos de la interfaz que deberían ser implementados, sino también la manera en la cual estos objetos interactuarán con el resto de la aplicación.

En hipermedia existen requerimientos que deben ser satisfechos en un entorno de desarrollo unificado (framework). Por un lado, la navegación y el comportamiento funcional de la aplicación deberían ser integrados. Por otro lado, durante el proceso de diseño se debería poder desacoplar las decisiones de diseño relacionadas con la estructura navegacional de la aplicación, de aquellas relacionadas con el modelo del dominio.

OOHDM propone el desarrollo de aplicaciones hipermedia a través de un proceso compuesto por cuatro etapas: diseño conceptual, diseño navegacional, diseño de interfaces abstractas e implementación.

4.1.2 Diseño Conceptual

Durante esta actividad se construye un esquema conceptual representado por los objetos del dominio, las relaciones y colaboraciones existentes establecidas entre ellos. En las aplicaciones hipermedia convencionales, cuyos componentes de hipermedia no son modificados durante la ejecución, se podría usar un modelo de datos semántico estructural (como el modelo de entidades y relaciones). De este modo, en los casos en que la información base pueda cambiar dinámicamente o se intenten ejecutar cálculos complejos, se necesitará enriquecer el comportamiento del modelo de objetos.

En OOHDM, el esquema conceptual está construido por clases, relaciones y subsistemas. Las clases son descritas como en los modelos orientados a objetos tradicionales. Sin embargo, los atributos pueden ser de múltiples tipos para representar perspectivas diferentes de las mismas entidades del mundo real.

Se usa notación similar a UML (Unified Modeling Language) y tarjetas de clases y relaciones similares a las tarjetas CRC (Class Responsibility Collaboration). El

esquema de las clases consiste en un conjunto de clases conectadas por relaciones. Los objetos son instancias de las clases. Las clases son usadas durante el diseño navegacional para derivar nodos, y las relaciones que son usadas para construir enlaces.

4.1.3 Diseño Navegacional

La primera generación de aplicaciones Web fue pensada para realizar navegación a través del espacio de información, utilizando un simple modelo de datos de hipermedia. En OOHDM, la navegación es considerada un paso crítico en el diseño aplicaciones. Un modelo navegacional es construido como una vista sobre un diseño conceptual, admitiendo la construcción de modelos diferentes de acuerdo con los diferentes perfiles de usuarios. Cada modelo navegacional provee una vista subjetiva del diseño conceptual.

El diseño de navegación es expresado en dos esquemas: el esquema de clases navegacionales y el esquema de contextos navegacionales. En OOHDM existe un conjunto de tipos predefinidos de clases navegacionales: nodos, enlaces y estructuras de acceso. La semántica de los nodos y los enlaces son las tradicionales de las aplicaciones hipermedia, y las estructuras de acceso, tales como índices o recorridos guiados, representan los posibles caminos de acceso a los nodos.

La principal estructura primitiva del espacio navegacional es la noción de contexto navegacional. Un contexto navegacional es un conjunto de nodos, enlaces, clases de contextos, y otros contextos navegacionales (contextos anidados). Pueden ser definidos por comprensión o extensión, o por enumeración de sus miembros.

Los contextos navegacionales juegan un rol similar a las colecciones y fueron inspirados sobre el concepto de contextos anidados. Organizan el espacio navegacional en conjuntos convenientes que pueden ser recorridos en un orden particular y que deberían ser definidos como caminos para ayudar al usuario a lograr la tarea deseada.

Los nodos son enriquecidos con un conjunto de clases especiales que permiten de un nodo observar y presentar atributos (incluidos los links), así como métodos (comportamiento) cuando se navega en un particular contexto.

4.1.4 Diseño de Interfaz Abstracta

Una vez que las estructuras navegacionales son definidas, se deben especificar los aspectos de interfaz. Esto significa definir la forma en la cual los objetos navegacionales pueden aparecer, cómo los objetos de interfaz activarán la navegación y el resto de la funcionalidad de la aplicación, qué transformaciones de la interfaz son pertinentes y cuándo es necesario realizarlas. Una clara separación entre diseño navegacional y diseño de interfaz abstracta permite construir diferentes interfaces para el mismo modelo navegacional, dejando un alto grado de independencia de la tecnología de interfaz de usuario.

El aspecto de la interfaz de usuario de aplicaciones interactivas (en particular las aplicaciones Web) es un punto crítico en el desarrollo que las modernas metodologías tienden a descuidar. En OOHDM se utiliza el diseño de interfaz abstracta para describir la interfaz del usuario de la aplicación de hipermedia.

El modelo de interfaz ADVs (Abstract Data View) especifica la organización y comportamiento de la interfaz, pero la apariencia física real o de los atributos, y la disposición de las propiedades de las ADVs en la pantalla real son hechas en la fase de implementación.

4.1.5 Implementación

En esta fase, el diseñador debe implementar el diseño. Hasta ahora, todos los modelos fueron construidos en forma independiente de la plataforma de implementación; en esta fase es tenido en cuenta el entorno particular en el cual se va a correr la aplicación.

Al llegar a esta fase, el primer paso que debe realizar el diseñador es definir los ítems de información que son parte del dominio del problema. Debe identificar también, cómo son organizados los ítems de acuerdo con el perfil del usuario y su tarea; decidir qué interfaz debería ver y cómo debería comportarse. A fin de implementar todo en un entorno Web, el diseñador debe decidir además qué información debe ser almacenada.

Comparación de OOHDM con otras metodologías La comparación de métodos de desarrollo de sistemas de software es una tarea difícil. El foco de cada metodología puede ser diferente, algunas tratan de concentrarse en varios aspectos del proceso de desarrollo, otras tratan de detallar en profundidad algún aspecto en particular.

	Proceso	Técnica de modelado	Representación gráfica	Notación	Herramienta de soporte
HDM	1.Desarrollo a largo plazo 2.Desarrollo a corto plazo	E-R[12]	1.-2.Diagrama E-R	1.E-R	
RMM	1.Diseño E-R 2.Diseño *Slice*[13] 3.Diseño de navegación 4.Diseño de protocolo de conversión 5.Diseño de UI[14] 6.Diseño de comportamiento en tiempo de ejecución 7.Prueba y construcción	E-R	1.Diagrama E-R 2.Diagrama *Slice* 3.Diagrama RMDM[15]	1.E-R 2.3.Propio	*RMCase*
EORM	1.Clases del entorno de desarrollo 2.Composición del entorno de desarrollo 3.Entorno de desarrollo de UI	OO[16]	1.Diagrama de clases 2.Diseño GUI[17]	1.OMT[18]	*ONTOS Studio*
OOHDM	1.Diseño conceptual 2.Diseño navegacional 3.Diseño abstracto de la UI 4.Implementación	OO	1.Diagrama de clases 2.Diagrama navegacional, clase + contexto 3.Diagrama de configuración de ADV + Diagrama ADV	1.OMT/ UML[19] 2.Propio 3.ADVs	*OOHDM-Web*
SOHDM	1.Análisis del dominio 2.Modelo en OO 3.Diseño de la vista 4.Diseño navegacional 5.Diseño implementación 6.Construcción	Escenarios Vistas-OO	1.Diagramas de escenarios de actividad 2.Diagrama de estructura de clase 3.Vista OO 4.Esquema de enlace navegacional 5.Esquema de páginas	1.-5.Propio	
WSDM	1.Modelado del usuario 2.Diseño conceptual 2.1.Modelo objetos 2.2.Diseño navegacional 3.Diseño implementación 4.Implementación	E-R/ OO	1.Diagrama de E-R o clase 2.Capas de navegación	1.E-R/ OMT 2.Propio	
WAE-Proceso *Conallen*	1.Manejo de proyecto 2.Captura de requerimientos 3.Análisis 4.Diseño 5.Implementación 6.Prueba 7.Desarrollo 8.Configuración y manejo de cambios	OO	2.-5.Diagramas UML	UML	*Rational Rose*

Ilustración 3 : Comparación de OOHDM con otras metodologías

En la Ilustración 3 : Comparación de OOHDM con otras metodologías se presenta una comparación de distintas metodologías, teniendo en cuenta los pasos que

componen el proceso, la técnica de modelado, la representación gráfica, la notación elegida para los modelos y la herramienta CASE de soporte proporcionada para el desarrollo.

Las metodologías comparadas son: HDM (Hypermedia Design Method), RMM (Relationship Management Methodology), EORM (Enhanced Object Relationship Methodology), OOHDM, SOHDM (Scenario-based Object-oriented Hypermedia Design Methodology), WSDM (Web Site Design Method), y WAE-Proceso Conallen (Web Application Extension for UML – Process Conallen).

	HDM	RMM	EORM	OOHDM	SOHDM	WSDM	WAE
Nivel Conceptual	Entidad Colección Perspectiva Relaciones	entidad relación	clases relación-OO -generalizada -definida por el usuario	clases perspectiva relación-OO	escenarios: -evento -actividad fujo de actividad	ojeto perspectiva relación	*case* relación-OO
Nivel Estructural	Enlace: -estructural -aplicación -perspectiva componente nodo Estructuras de acceso: -enlace colección -enlace índice -visita guiada	enlace: -unidireccional -bidireccional *Slices* primitivas de acceso: -agrupar (menú) -índice -visita guiada -visita guiada indexada	enlace: -simple -navegacional -nodo a nodo -tramo a nodo -estructural -conjunto -lista	enlace clase navegacional contexto navegacional estructuras de acceso: -índice -visita guiada	enlace navegacional vista-OO: -base -asociación -colaboración ASN[20]: -agrupar -índice -visita guiada	enlace componente -navegación -información -externo camino navegacional	enlace enlace dirigido redirigir construir enviar página *web* -página del cliente -página del servidor
Nivel Visible	Ranura Marco	*Slices*		ADV en contexto	componente UI: -elección -texto de entrada de búsqueda -botón -imagen -barra de desplazamiento - ancla HTML[21] -otros		conjunto de marcos formulario objetivo elemento de selección elemento de entrada elemento de área de texto

Ilustración 4: Comparación de conceptos de diseño de las metodologías de desarrollo Web

En la Ilustración 4: Comparación de conceptos de diseño de las metodologías de desarrollo , se presenta un segundo estudio comparativo, que relaciona los conceptos de diseño de los tres niveles típicos de diseño Web: conceptual, estructural y visible. La mayoría de estos métodos realizan una clara separación entre el análisis del dominio, la especificación de la estructura navegacional y el diseño de la interfaz de usuario.

4.1.6 Conclusiones

La metodología OOHDM propone dedicar un tiempo importante en las fases previas a la implementación.

Esta inversión de tiempo está ampliamente justificada no sólo porque simplifica el proceso de desarrollo, facilitando el trabajo del equipo encargado de cada capa de la aplicación, sino también durante su mantenimiento y eventual extensión. Son quizás estas últimas tareas las más difíciles de lograr con tecnologías tradicionales, y aún imposibles en muchos casos donde no existe diseño detallado y la implementación concentra conceptos heterogéneos muy difíciles de modificar.

OOHDM propone un conjunto de tareas que en principio pueden involucrar mayores costos de diseño, pero que a mediano y largo plazo reducen notablemente los tiempos de desarrollo al tener como objetivo principal la reusabilidad de diseño, y así simplificar la evolución y el mantenimiento.

4.2 Diagramas de Interacción (UIDs)

4.2.1 Introducción

UML sugiere el empleo de casos de uso para la captura de requerimientos y para especificar la interacción entre los usuarios y el sistema que esta siendo modelado. Los casos de usos son fáciles de entender por el usuario entonces ellos son una descripción textual esencial.

La interacción es la actividad de comunicación que tiene lugar entre el usuario y el sistema. Desde ésta interacción es posible identificar la información manipulada por el sistema y la funcionalidad que éste debe ofrecer. Entonces, la especificación de la interacción usuario-sistema es fundamental para el desarrollo de un sistema, principalmente para la obtención de requerimientos.

En UML, la interacción es vista como un intercambio de mensajes entre los objetos del sistema; el cual difiere del concepto que nosotros utilizamos que es entre el sistema y el usuario. De esta manera los diagramas de UML representan la interacción.

4.2.2 Definición

Los diagramas UID solo describen el intercambio de información entre el sistema y el usuario, sin considerar aspectos específicos de la interfase de usuario. Este representa la interacción de información que es textualmente descripta en un caso de uso.

Veamos un ejemplo basado en un típico negocio de ventas por Internet:

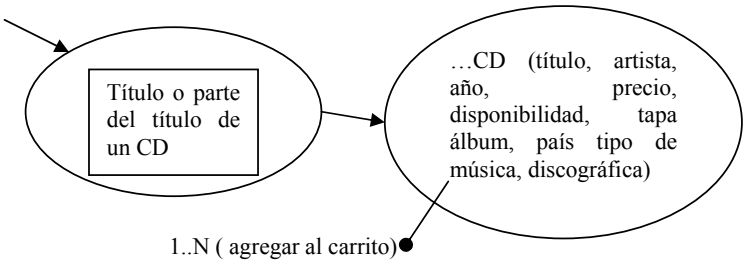


Ilustración 5 : Selección de un CD por un título determinado.

En la Ilustración 5 : Selección de un CD por un título determinado., la elipse representa la interacción entre el usuario y el sistema. La interacción comienza con una elipse que tiene una flecha sin origen. En este ejemplo, el usuario debe ingresar el titulo del CD, denotado por un rectángulo de bordes continuos, luego que el usuario haya ingresado la información, el sistema presenta el conjunto del CD que corresponda a la entrada del usuario. El resultado de cada interacción que cause procesamiento debe ser presentado como una elipse separada. Conectado a la interacción precedente por una flecha. En este ejemplo, la presentación del conjunto de CDs implica procesamiento del sistema, el conjunto resultante es presentado en otra interacción que esta conectado en la anterior por medio de una flecha. Un conjunto de CDs es representado por elipses en frente del nombre del conjunto (en el ejemplo "… CD"). Por cada CD que será mostrado por el sistema, se mostrará su información asociada: titulo, artista, año, precio, disponibilidad, tapa álbum, país tipo de música y discográfica; esta información se especifica entre paréntesis al lado del nombre del conjunto. Desde ésta última interacción es posible seleccionar desde 1 a N CDs y ejecutar la operación "agregar a carrito", esto esta representado a través de una línea conectada al texto "*1...N (agregar al carrito)*" con una circunferencia en este extremo.

4.2.3 Proceso de construcción de UID

De acuerdo con UML, los diagramas de secuencia, estado y colaboración permiten detallar los casos de uso. Sin embargo, la utilización de estas técnicas es engorrosa por no disponer de una herramienta apropiada y puede llevar a decisiones de diseño prematuras.

Nosotros proponemos el uso de los UIDs, como alternativa a los casos de usos, para representar de una forma más eficiente y expresiva la información.

Para obtener un UID desde un caso de uso[5], la secuencia de información intercambiada entre el usuario y el sistema debe ser identificada y organizada en Interacciones. Identificar el intercambio de información es crucial ya que es la base para la definición de UIDs.

A continuación presentaremos un método para mapear casos de uso a UID. Para ejemplificar la aplicación del método, mostraremos el mapeo del caso de uso "seleccionar un CD teniendo como base el nombre del artista".

[5] Nos referimos a caso de uso, como ejemplo práctico, no como un caso de uso de UML.

Caso de Uso: seleccionar un CD teniendo como base el nombre del artista.

El usuario ingresa el nombre del artista o parte de éste. El puede completar el criterio de búsqueda con el año o periodo del CD. El sistema retorna un conjunto de nombres de artistas que son compatibles con los datos ingresados. El usuario selecciona el artista deseado. El sistema retorna los CDs cuyo artista es el seleccionado. Por cada CD se muestra: título, nombre del artista, año, precio, disponibilidad, tapa álbum, país, y discográfica. Si el usuario desea adquirir uno o más CDs, él selecciona los CDs y los agrega al carrito de compras para concretar la compra más tarde (caso de uso Compra).

Paso 1. Comenzamos analizando el caso de uso para identificar que información que es intercambiada entre el usuario y el sistema. Generalmente "unidades de información" son representadas con sustantivos, llamados elementos de datos.

En el ejemplo, podemos identificar los siguientes flujos de información entre el usuario y el sistema: nombre del artista o parte de él (usuario), año o periodo del CD (usuario) ; una lista de nombres de artistas que son compatibles con los parámetros de búsqueda (sistema) , el artista de interés (usuario), un conjunto de CDs del artista seleccionado, por cada CD encontrado: el titulo, nombre del artista, año, precio, disponibilidad, tapa álbum, país, y discográfica (sistema).

Paso 2. Después de identificar la información intercambiada, debemos separar las unidades de información dentro de las interacciones. Estos elementos de datos son ubicados en la misma interacción, salvo que se requiera el procesamiento del sistema antes de ser presentada; en tal caso estos elementos son agrupados en otra interacción. Seguiremos con este criterio hasta definir correctamente todas las interacciones. Para poder seguir la secuencia de interacciones, a cada nueva interacción se le asigna temporalmente un número de secuencia.

En el ejemplo, comenzamos con el elemento de dato "*nombre del artista o parte de él*", el cual es ubicado en la primera interacción del UID. El siguiente elemento es el "a*ño del CD o periodo*" es opcional, es ingresado por el usuario, no requiere procesamiento del sistema, por lo tanto puede ser ubicado en la misma interacción que el elemento de dato anterior. Sin embargo, el siguiente elemento intercambiado, el conjunto de "*nombres del artistas*", es retornado por el sistema después de procesar una consulta en la base de datos situado en una segunda interacción. El próximo elemento de datos, "*el artista de interés"*, solicita al sistema verificar la selección del un artita indicado por el usuario. Lo cual será presentado en la tercer interacción. El ultimo elemento de dato (*el titulo, nombre del artista, año, precio, disponibilidad, tapa álbum, país, y discográfica*) son retornados por el sistema como consecuencia del procesamiento de la solicitud anterior; siendo presentados en la tercer interacción. Cabe destacar que en esta última interacción se presentará una lista de elementos de datos.

Paso 3. En este paso se discriminan cuales son los elementos de datos ingresados por el usuario y cuales son los retornados por el sistema. Los ingresados por el usuario pueden ser requeridos y se sitúan dentro de un rectángulo con bordes sólidos; en cambio los opcionales se sitúan dentro de un rectángulo con bordes discontinuos. Los elementos de datos retornados por el sistema son situados sin ninguna decoración.

En el ejemplo, en la primera interacción, como el "*nombre del artista o parte de él*" es requerido, este es situado dentro de un rectángulo con bordes sólidos. Por otro lado, "a*ño del CD o periodo*" es una entrada opcional, es situado en un

rectángulo de bordes discontinuos. En la segunda interacción encontramos un conjunto de "*nombres del artistas*" que es retornado por el sistema como respuesta a la interacción anterior. El único detalle de presentación es una elipsis antes del nombre del elemento de dato indicando que es un conjunto de elemento. En la tercer interacción, a pesar del hecho de que el elemento de dato "*artista de interés"* depende de la selección del usuario, este es presentado por el sistema directamente sobre la interacción. El conjunto de CDs es también situado en la interacción siendo precedido por una elipsis.

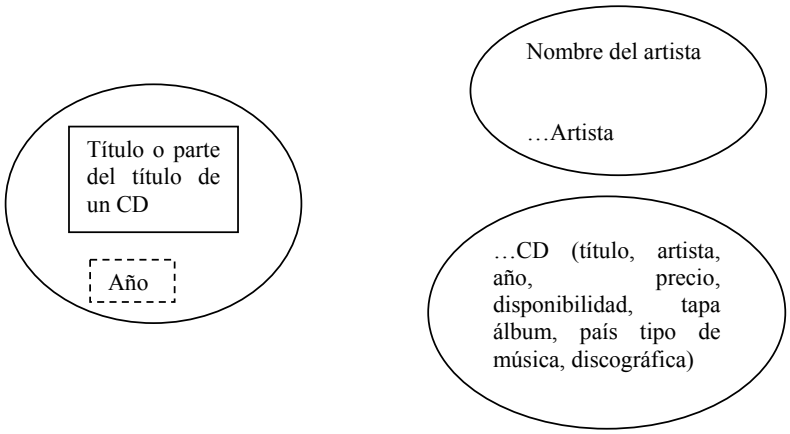


Ilustración 6: Caso de uso de seleccionar un CD basándose en el nombre del artista: interacciones y elementos de datos.

Paso 4. Las interacciones son conectadas por flechas. Cuando una interacción es conectada, la interacción destino debe tener un numero de secuencia mayor al de la fuente. Es posible conectar una interacción con dos o más interacciones representando varias alternativas de interacción. En este caso la interacción ingresada por el usuario determina cual de las interacciones disponibles será alcanzada. Si el cambio de la interacción es el resultado de una selección de un elemento, el número de elementos seleccionados es ligado a la flecha y el origen de la flecha es el conjunto (interacción) desde el cual fueron tomados los elementos. La interacción inicial es una flecha sin origen.

En el ejemplo, la segunda interacción ocurre cuando los elementos de datos de la primera interacción han sido ingresados, por lo tanto la flecha tiene origen en la primera interacción y como destino a la segunda interacción. La tercer interacción ocurre exactamente cuando el usuario selecciona un artista, en consecuencia la flecha conecta la segunda y tercer interacción con el número 1 ligado a la flecha y el origen es el conjunto de artista (y no a la interacción entera).

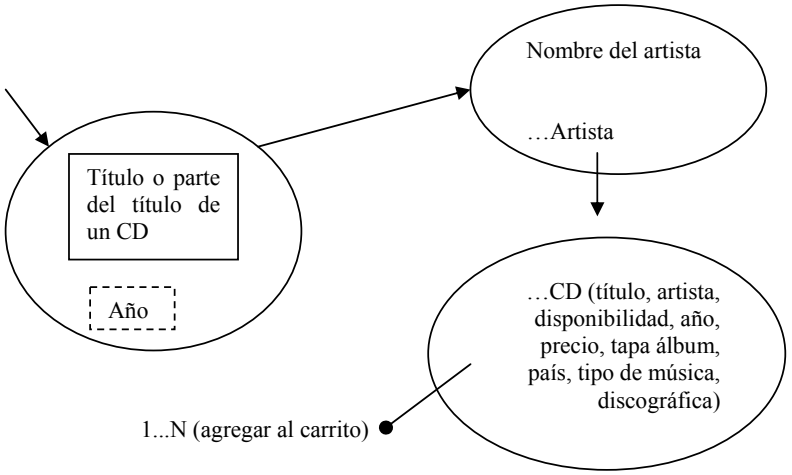


Ilustración 7: Caso de uso de selección de un CD en base al nombre de un artista: UID

Paso 5. Las operaciones realizadas sobre los elementos de datos deben también ser identificadas. Ellas son generalmente representadas en los casos de uso como verbos. Una operación es representada en un diagrama como una línea con una bala (o circunferencia). Si después de la ejecución de una operación otra interacción ocurre, entonces la operación es indicada con una flecha en vez de una línea.

En el ejemplo, la última información presentada en el caso de uso es una operación que permite al usuario incluir los CDs seleccionados al carrito de compras. Notar que la línea que representa ésta operación esta conectada al conjunto de CDs indicando que el conjunto sirve como entrada a la operación. La cantidad de CDs seleccionados y la operación son posicionadas junto a la línea. El nombre de la operación es presentado entre paréntesis.

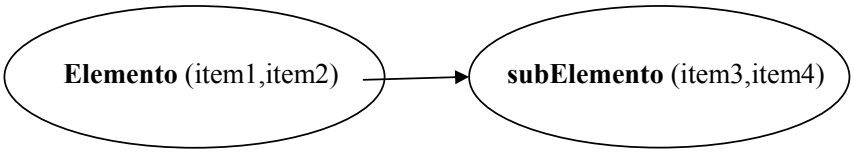
Paso 6. Los requerimientos no funcionales no son representados en UID. Sin embargo, los casos de uso no especifican estos requerimientos, algunas veces los requerimientos no funcionales pueden aparecer. En este caso deberían aparecer adjuntos al UID como un texto.

4.2.4 Heurísticas y Derivación del diagrama de clases

El diagrama de clase es derivado de acuerdo a reglas (heurísticas) que son aplicadas a los UID. Algunas de estas reglas son resultantes de las técnicas de normalización de esquemas. Las reglas para obtener un diagrama de clases desde UIDs son las siguientes:

1. Por cada UID, definir una clase para cada elemento y conjunto de elementos. Por cada clase indefinida, asumir la existencia de un atributo identificador OID.

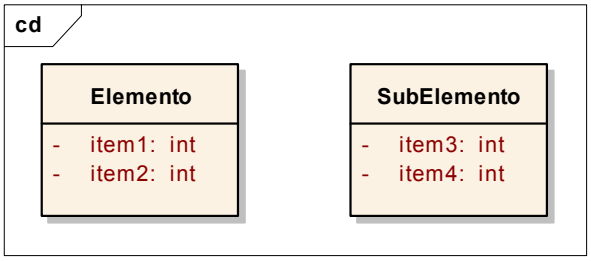
Ejemplo:



De esta regla, obtenemos las clases **Elemento** y **SubElemento**. El nombre de SubElemento es totalmente arbitrario no existe relación entre estos dos.

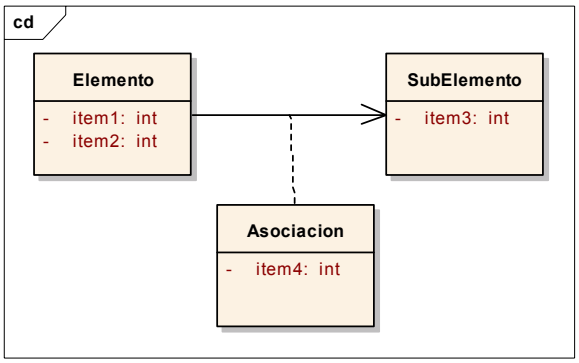
2. Por cada elemento de información de el conjunto de elementos que aparece en cada OID (Object IDentifier), definir un atributo de acuerdo a las siguientes pautas:
    a. Verificar si el item de dato es funcionalmente dependiente de el OID en cada clase. Ejemplo, si *OID → elemento de dato*. Verificar si el elemento de dato no es transitivamente dependiente en el OID (Ej, si *OID →OID2 → elemento de dato*). Si estas condiciones son satisfechas, entonces estos elementos de datos deben convertirse en atributos de la clase. Esta verificación debería preferiblemente comenzar con la clase que representa el conjunto, porque hay una alta probabilidad de que este elemento de dato pueda ser un atributo de esta clase.

       Ejemplo: Supongamos que en el diagrama anterior se cumplen las condiciones, obtendríamos los siguientes atributos.



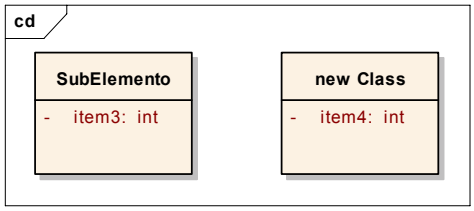
    b. Verificar si el ítem de información es funcionalmente dependiente sobre el OID de dos o más clases diferentes y si el elemento de dato no es transitivamente dependiente en estos OIDs. Si estas condiciones son satisfechas, entonces el elemento de dato debe convertirse en un atributo de una asociación entre las clases.

       Ejemplo: Si *Elemento.OID,SubElemento.OID* → item4

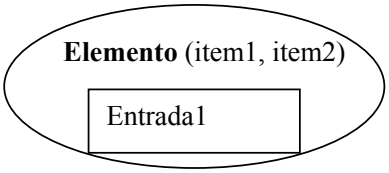


Obtenemos una clase asociación que se compone con el atributo Item4.

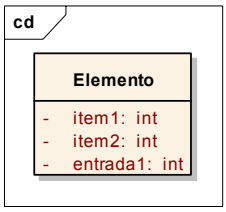
c. Si el ítem de dato no es funcionalmente dependiente sobre el OID de ninguna clase existente, o si este solamente es transitivamente dependiente de OIDs existentes, una nueva clase necesita ser definida con su propio OID. El elemento de dato debe ser agregado como un atributo de la nueva clase.
Ejemplo: En este caso, supongamos que item4 (sea *nombre_persona*) es transitivamente dependiente de *SubElemento*.OID (sea compra), y no es funcionalmente dependiente en ninguna de las clases existentes; entonces se crea una clase (sea *Persona*) que tiene como atributo *item4* (*nombre_persona*).



3. Por cada dato de entrada ingresado por el usuario, representado por un rectángulo simple, definimos un atributo de acuerdo a las siguientes pautas:
   a. Verificar si la entrada de dato es funcionalmente dependiente sobre el OID en cada clase, ejemplo si OID →elemento de dato. Verificar si la entrada de dato no es transitivamente dependiente en el OID. Si estas condiciones son satisfechas, entonces el elemento de datos debe convertirse en un atributo de la clase. La verificación debería comenzar preferiblemente con la clase que representa el conjunto porque hay una alta probabilidad de que este elemento de dato sea un atributo de esta clase.



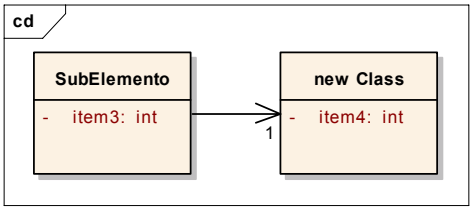
Ejemplo: En este caso, si Entrada1 es funcionalmente dependiente de *Elemento*. Obtenemos lo siguiente:



b. Verificar si la entrada de datos es funcionalmente dependiente sobre el OID de dos o más clases diferentes y si el elemento de datos no es transitivamente dependiente en estos OIDs. Si estas condiciones son satisfechas entonces el elemento de datos debe convertirse en un atributo de asociación entre las clases.
Ejemplo: Similar 2b.

c. Si la entrada de dato no es funcionalmente dependiente sobre el OID de cualquier clase existente o si este es solo transitivamente dependiente de OID existentes, una nueva clase necesita ser definida con su propio OID. El elemento de dato debe ser agregado como un atributo de la clase recientemente agregada.
Ejemplo: Similar 2c.

4. Por cada atributo *a* que aparece en un conjunto diferente de subclases, se proceden como sigue:
   a. Si el OID de la clase de *a* funcionalmente depende sobre el OID de la clase correspondiente al conjunto y el OID de la clase de *a* no es transitivamente dependiente de aquel OID, crear una asociación entre la clase que contiene a *a* y la clase que representa el conjunto. La máxima cardinalidad de la clase que contiene a *a* en la asociación es 1. La condición de transitividad es necesaria para que no se creen relaciones derivadas o puentes entre clases. Por ejemplo: A→B→C, es correcto una asociación ente A y C? No.
   Verificar si el resultado de la asociación es semánticamente correcta (por ejemplo, si tiene sentido en el dominio ser modelado).
   Ejemplo: En el ejemplo de la regla 2c, se obtuvo una nueva clase y en ese caso se cumple la regla actual; obteniendo el siguiente diagrama:



   b. Si el OID de la clase de *a* funcionalmente depende del OID de la clase de otro atributo *b* en el mismo conjunto, pero el OID de la clase de *a* no depende transitivamente en aquel OID, se crea una asociación entre la clase que contiene a *a* y la clase que contiene a *b*. la máxima cardinalidad

de la clase que contiene a *a* en la asociación es 1. Verificar si el resultado de la asociación es semánticamente correcta (por ejemplo, si tiene sentido en el dominio ser modelado).

5. Por cada flujo de interacción (representado por una flecha), si estos son de diferentes clases en la interacción origen y en la interacción destino, definir una asociación entre estas clases. Es también necesario verificar si el resultado de la asociación es semánticamente correcta (por ejemplo, si tiene sentido en el dominio ser modelado).
    a. Si n elementos pueden ser seleccionados en la interacción origen, entonces el máximo de cardinalidad de la clase de la interacción origen es n.
    b. Si el conjunto de elementos puede ser retornado en la interacción destino, entonces el máximo de cardinalidad de la clase interacción destino es n.
    c. Si el elemento o el conjunto de elementos retornados en la interacción destino es opcional (ejemplo, el elemento o el conjunto en la interacción destino puede ser vacío), entonces la cardinalidad mínima en la clase interacción destino es 0.
6. Por cada operación que aparece conectada a un elemento o conjunto, si este es una operación de la clase correspondiente, agregar éste a la clase.
7. En el final del proceso haremos los ajustes necesarios en el diagrama de clases resultante, por ejemplo:
    - Distinguir las asociaciones duplicadas, esto puede suceder porque podemos derivar una o más veces una misma asociación.
    - Identificar generalizaciones
    - Buscar cardinalidades perdidas de asociaciones.

4.2.5 Extensión de heurísticas

El proceso de derivación de clases a partir de diagramas de UIDs no es un proceso trivial. Este proceso esta sujeto, al igual que cualquier fase de análisis de requerimientos, al grado de completitud y correctitud con el que el Ingeniero en requerimiento realiza los diagramas. En consecuencia se aconseja un nivel de detalle satisfactorio que permita un funcionamiento de las reglas. Es deseable no omitir información (datos de entrada, ítems, secuencias, etc.) en los UIDs dado que cada unidad de información faltante indica que nuestro modelo es menos completo.

Basándonos en nuestra experiencia en desarrollo de sistemas, notamos que si definimos los UIDs de altas de entidades del sistema podemos obtener gran parte del modelo de clases. Sin embargo, es necesario tener todo el conjunto de UIDs porque los UIDs restantes nos completan el modelo con la lógica de negocio (clases y asociaciones).

Una vez completado el primer paso, vemos necesario una especificación de dependencias funcionales del conjunto de "Entidades" reconocidas. Este paso es necesario porque todo el conjunto de reglas se ven restringido por estas dependencias.

Si definimos adecuadamente toda la información, las reglas 2c y3c no deberían utilizarse.

Veremos más adelante que las reglas de composición de UIDs permiten determinar aspectos o concerns del sistema. No sólo permite definir el mapa navegacional del sitio, sino también nos insinúa posibles asociaciones entre clases.

Completamos la definición de reglas con el siguiente agregado:

"Si obtenemos de una interacción dos o más clases diferentes, es altamente probable que exista una asociación entre ellas, y su aridad depende de la forma en que fueron representados, elemento único o conjunto. Si uno de ellos es un conjunto entonces la aridad máxima será de N, caso contrario 1."

Nuevamente, las reglas dependen mucho la capacidad del Ingeniero de Requerimientos en generar diagramas completos y correctos; además de su forma de expresar las ideas.

4.2.6 UIDs y el Proceso Unificador (Unified Process)

El desarrollo de aplicaciones de acuerdo al Proceso Unificador es dividido en varios ciclos. Cada ciclo resulta en un producto y esta compuesto por cuatro fases: principio, elaboración, construcción y transición. Sobre estas fases, cinco flujos de trabajo tienen lugar: Requerimientos, Análisis, Diseño, Implementación y Testeo.

Durante los flujos de trabajo de Requerimientos, los actores son identificados y los casos de usos en los cuales cada actor participa están definidos. Los prototipos de interfaces de usuario pueden también ser usados en la obtención de requerimientos. Estas interfaces de usuario especifican la interacción entre los actores humanos y el sistema.

Se propone extender las etapas de Análisis y Requerimientos del Proceso Unificador introduciendo el uso de los UIDs. Durante los Requerimientos, luego de identificar los actores y especificar el caso de uso, un UID debe ser especificado para cada caso de uso. Estos UIDs son usados en conjunto con los casos de uso para validar los requerimientos. De esta forma no tenemos que definir una interfaz de usuario que puede ser modificada después de la etapa de Diseño, rompiendo con la expectativa que el usuario había creado sobre la interfaz del sistema.

Durante la etapa de Análisis, las clases de entidades son definidas usando las pautas de la sección anterior. El resultante diagrama de clases puede ser completado analizando los descriptores de casos de uso y definiendo los límites de las clases y las clases de control[6] como en el Proceso Unificador. Así, por cada actor humano, clases de entidad, y actores externos al sistema, una clase límite es definida. Para cada caso de uso, es necesario verificar si uno o mas clases de control son requeridas. Los atributos de límite y clases de control pueden ser obtenidos como en el Proceso Unificador, observando a las clases de entidad y sus atributos. Para completar la etapa de Análisis, el diagrama de colaboración es definido como en el Proceso Unificador.

4.2.7 Conclusiones

Hemos presentados a los UIDs como una notación gráfica que representa la interacción del usuario con el sistema. UIDs han sido propuestos como una herramienta de alto nivel de abstracción que puede ser usado sin preocuparse de las interfaces de

[6] El proceso unificador define clases de entidad, control y límite: Las clases de entidad como clases que describen la estructura y relaciones entre los datos que son eventualmente almacenados en la BD. Las clases límites son las que interactúan con el usuario. Y por ultimo, las clases de control que une los componentes de la aplicación.

usuario y detalles de diseño. Al mismo tiempo, UIDs pueden ser también usados para validar los casos de uso y contribuir a la bases para la definición del diagrama de clases.

También se han propuesto un proceso para definir el diagrama de clases desde los UIDs. Algunos de los pasos del proceso están basados en el bien conocido conceptos de dependencia funcional.

Aunque el diagrama resultante puede necesitar algunos ajustes, estos pueden ser aplicados en contados pasos.

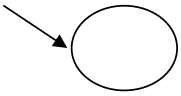
Los UIDs pueden ser usados para la obtención de requerimientos en cualquier metodología de software. En particular hemos citado como incorporar UIDs al Proceso Unificador.

Además los UIDs también son usados para el diseño navegacional en Object Oriented Hypermedia Design Method (OOHDM).

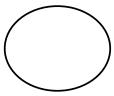
4.2.8 Notación

Para representar la interacción entre el usuario y el sistema usaremos la siguiente notación:

Interacción Inicial: representa el comienzo de la interacción entre el usuario y el sistema.

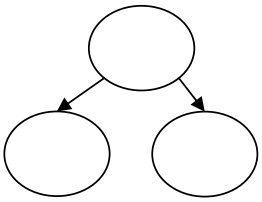


Interacción: representa una interacción entre el usuario y el sistema. La información dada por el usuario y devuelta por el sistema se ve dentro de la elipse.

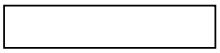


Interacción Opcional: representa la interacción que depende de otra interacción previa. Dependiendo de su resultado anterior puede ocurrir o no. Si no ocurre la salida de la interacción anterior será la entrada de la interacción siguiente.

Alternativas de interacción: esta representación es usada cuando existen dos alternativas de salida de una interacción. La interacción siguiente depende de los elementos o de la operación elegida por el usuario.



Entrada de dato: representa la entrada de dato obligatorio del usuario.



Entrada de datos opcional: representa la entrada de dato opcional del usuario.

Elemento y sus atributos: representa el elemento y sus atributos. Los atributos pueden ser opcionales.

Elemento (item1, item2,..)

Conjunto de elementos: Representa un conjunto de elementos. Los atributos asociados al elemento también se presentan.

… Elemento (item1, item2,..)

Elemento específico: Representa un elemento específico seleccionado o ingresado por el usuario en una iteración previa.

Elemento X

Texto: representa la información adicional del dato que participa en la interacción.

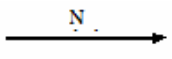
XXX

Texto opcional: representa la información opcional del dato que participa en la interacción.

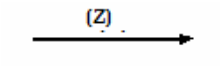
XXX*

Interacción nueva: representa una nueva interacción que ocurre después de que el usuario haya incorporado los datos requeridos y el sistema haya devuelto la información en la interacción anterior.

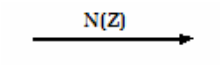
Selección de los N elementos y la nueva interacción: representa que los N elementos se deben seleccionar antes de la nueva interacción.



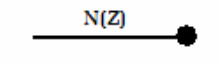
Invocación de la operación Z y nueva interacción: representa que la operación Z se debe llamar antes de la nueva interacción.



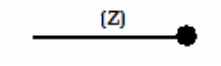
Selección de N elementos e invocación de la operación de Z y nueva interacción: representa que los N elementos deben ser seleccionados y la operación Z se debe llamar antes de la nueva interacción.



Selección de N elementos e invocación a la operación: representa que los N elementos deben ser seleccionados y la operación debe ser invocada.



Invocación de la operación Z: representa la operación Z puede ser invocada. La invocación es opcional.



5 Identificación y composición de concerns navegacionales

5.1 Concern Navegacionales

Un concern implica cualquier conjunto de requerimientos coherente, por ejemplo, todos los requerimientos se refieren a un tema particular o **característica de comportamiento de la aplicación**. Los concerns pueden se funcionales, tal como la validación y la auditoria, y no funcionales como la persistencia, la seguridad y la portabilidad. Los concerns pueden ser genéricos, cuando ellos aparecen en un extenso número de aplicaciones. (Por ejemplo la adaptabilidad, la utilidad), de dominio específico cuando ellos sólo se aplican a un conjunto de aplicaciones (por ejemplo el pago del e-commerce) o incluso en una aplicación especifica cuando ellas sólo se manifiestan en un tipo particular de software (por ejemplo el Mercado en Amazon.com).

Un concern navegacional es un concern de aplicación que impacta en el modo en el que el usuario navega la aplicación; en otras palabras un concern navegacional es reflejado en alguna estructura navegacional (por ejemplo en una pagina Web, un link), o comportamiento (por ejemplo, chequeando si un usuario puede acceder a esa página). Como consecuencia un concern navegacional puede impactar en el modelo navegacional de la aplicación, reflejándose tanto en un nodo como en una clase link. Estos concerns pueden ser concretados en un método, atributos (tales como información o enlace) o ambos.

Los concerns navegacionales son relevantes para las aplicaciones Web porque es generalmente aceptado que la calidad de su estructura de navegación (es decir la realización del conjunto de concerns navegacionales) es una llave para el éxito de aplicación; y en muchos casos del negocio en sí.

Las aplicaciones Web complejas pueden también incluir concerns que no son navegacionales, por ejemplo su realización no tiene la influencia en la navegación. Por ejemplo el concern de persistencia, aunque es relevante y complejo para implementar, es absolutamente transparente para el usuario final.

Nosotros decimos que los concerns navegacionales son crosscutting cuando ellos se esparcen a través de las clases de navegación (por ejemplo algunos concerns se muestran en diferentes clases nodos o links) o cuando ellos son combinados (por ejemplo algunas clases navegacionales soportan más de un concern). Los crosscutting concerns pueden aparecer como información o comportamiento que parecen no pertenecer completamente al nodo actual o como los links que “interrumpen” el flujo de la tarea actual.

Los concerns navegacionales puros tienen una característica importante: ellos no deben mostrarse como crosscutting en un modelo de aplicación (por ejemplo en le modelo conceptual OOHDM o UWE), pero ellos pueden aparecer como crosscutting en el modelo navegacional. Sin embargo, los crosscutting concerns funcionales pueden reflejarse en modificaciones en nodos navegacionales o en composiciones de vistas diferentes.

Notar que los crosscutting concerns en un concern navegacional puede ser originado en una interacción o en la navegación, por ejemplo nosotros queremos que el usuario tenga la posibilidad de vender sus productos (marketplace concern) mientras

explora un CD particular para comprarlo (concern vender, por ejemplo la información del producto y la selección del producto involucrado). Nosotros también queremos guardar un registro de la historia de navegación de usuario y mantenerla permanentemente disponible, etc.

Mientras algunos concerns crosscutting navegacionales pueden simplificar la tarea del usuario, estos impactan negativamente en la modularidad de la aplicación, su anidamiento dificulta su aislamiento (por ejemplo, para propósitos de mantenimiento).

Más concretamente, el anidamiento significa la misma clase de la navegación proporciona información y links relacionándolos a diferentes concerns (por ejemplo ventas y asesoramiento).

Cuando algunos de estos concerns evolucionan (por ejemplo aparecen nuevos requisitos) o se agregan nuevos concerns, todos los modelos de diseño, particularmente el modelo navegacional, podrían alterarse como consecuencia del anidamiento y distribución de concerns. Supongamos por ejemplo que nosotros decidimos cambiar nuestra política para recomendar los productos; este cambio de productos podría implicar la modificación de diferentes clases de navegación (por ejemplo el CD).

Analizar el crosscutting de los concerns navegacionales puede ayudar a identificar conflictos anticipadamente en el ciclo de vida del desarrollo. Por ejemplo alguna yuxtaposición de concerns navegacionales puede impactar negativamente en otros (por ejemplo los concerns no funcionales como la usabilidad).

Se argumento que los crosscuting concern navegacionales pueden ser identificados en la especificación de requerimientos; como fue dicho, la temprana identificación ayuda a los clientes a decidir acerca de potenciales Trade-offs, mejorar la modularidad de la aplicación y proveer seguimiento de requerimientos, para simplificar el mantenimiento.

5.2 Un modelo para concerns navegacionales

El modelo que soporte concerns navegacionales a nivel de requerimientos es el siguiente.

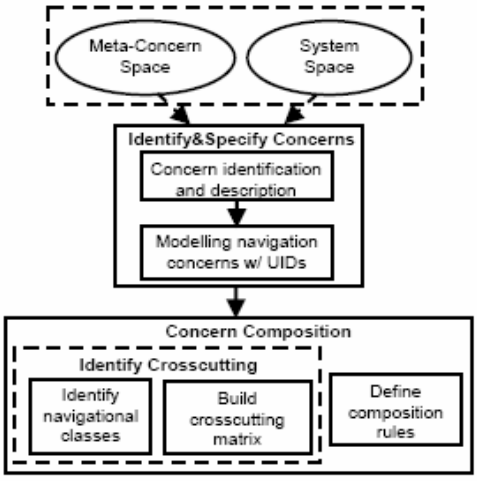


Ilustración 8: Visión general

Ciertos concerns, funcionales y no funcionales, aparecen varias veces durante el desarrollo del sistema. Esto fue el desencadenante para proponer un mecanismo de rehúso por medio de un catálogo de concerns definido de forma abstracta e independiente del sistema. Este catálogo (nombrado espacio del meta-concern) debe ser usado para ayudar a obtener los concerns de dominio del problema. La idea es usar esos metaconcerns, para ayudar a organizar los requerimientos del espacio del sistema (donde el dominio de la aplicación se describe) de modo que los concerns del dominio del problema sean identificados. Esta es una tarea dura para el analista ya que debe definir el espacio del sistema e instanciar los concerns. Estos concerns pueden verse como las especializaciones de concerns a nivel meta-espacio. Los colección meta-concerns crece y evoluciona con la experiencia del analista.

Los Meta-concerns y los espacios del sistema son representados por los ovalados en Figura 3.

Los concerns, abstractos del espacio del meta-concern así como sus realizaciones concretas podrán ser definidos usando un conjunto de plantillas bien definidas basadas en XML (eXtensible Markup Language).

Extendemos la noción de que un catálogo para concerns de navegación típico se encuentra en las aplicaciones Web, tales como Compras, publicidad, ventas, historial de navegación, etc.

En resumen, el espacio del sistema contiene la descripción del dominio del problema que nosotros queremos desarrollar. Esta descripción es obtenida desde fuentes diferentes (entrevistas, análisis de prácticas comerciales, etc.). Y el espacio del meta-concern contiene un conjunto de concerns abstractos típicos, los cuales se manifiestan repetidamente en varios dominios de la aplicación. La identificación de los concerns concretos puede ser alcanzada iterativa e incrementalmente, analizando partes pequeñas del dominio del problema en un momento dado. Notar que no todos los concerns en el espacio del meta-concern son necesariamente usados durante esta categorización; normalmente sólo un subconjunto de concerns de interés al problema del dominio es necesario.

Una vez que la descripción del dominio del problema ha sido organizada en especializaciones concretas de concerns desde el espacio del meta-concern, podemos modelar las interacciones relacionadas en cada concern. Para ello, esos requerimientos de cada concern de navegación que involucren interacciones de usuarios serán modelados usando UID.

Algunos concerns navegacionales, como la autenticación o la adaptatividad, abarcarán requerimientos que no tienen asociados una interacción con el usuario; por ejemplo requiriendo una validación del sistema contra la identidad del usuario (autenticación), y algún pre-procesamiento antes de que un link sea procesado y alcanzado (adaptatividad). En este caso, no va a haber un UID asociado.

La composición de concerns es realizada en el nivel de UID y es alcanzado en dos pasos:

1. identificar las unidades de navegación y construir una matriz crosscutting entre las unidades de navegación y los concerns navegacionales;
2. construir una regla de composición para cada unidad de navegación.

Una unidad de navegación es la homologo del requerimiento de un nodo en el modelo navegacional y refleja una estructura de información que surge de un estado[7] de la interacción en un UID (por ejemplo un CD, un carrito compra, etc.). Estas estructuras de información normalmente representan algunos elementos de datos

[7] Un estado es otras palabras una interacción (elipse) de un UID

presentados al usuario, y son generalmente la base para llevar a cabo la próxima interacción en un UID, por ejemplo porque alguna información es seleccionada.

Las unidades navegacionales corresponden a un concern dominante. Un concern dominante es el concern principal de una unidad navegacional, es decir representa la razón por qué la unidad fue alcanzada. Note sin embargo, que puede no ser necesario definir una unidad navegacional por concern. Por ejemplo, al acceder la información de un CD (concern principal), nosotros podemos tener un link a la promoción de las ventas (concern navegacional) o a otros CD recomendados, lo que significa que el concern de información de CD está entrelazado con los concerns representados por los links salientes. Y estos concerns no tienen unidades navegacionales porque son solo links.

La información del CD es el concern principal de una unidad particular (este define la unidad navegacional), y la promoción de las ventas es el concern navegacional crosscutting.

El segundo paso es definir reglas de la composición que mostrarán el impacto de los concerns crosscutting en las unidades de navegación.

Las unidades de navegación sirven como base para nuestras composiciones, esto significa que una regla de composición puede definirse para cada unidad de navegación. Como los concerns son descriptos en términos de UID, las reglas de composición definen composiciones específicas entre UIDs. En algunos casos la identificación de los concerns navegacionales crosscutting no implicará la composición de UIDs, pero esta información será usada en actividades de desarrollo posteriores.

Las reglas de composición pueden verse a dos niveles diferentes de abstracción:

(i) sólo mostrando presentación para los concerns navegacionales en la unidad de navegación, o
(ii) mostrando los UIDs en donde la unidad de navegación participa y los UIDs correspondiente a los concerns navegacionales en esta unidad compuesta formando un solo UID.

Para definir una regla de composición, necesitamos poder acceder los estados del UID. Para ello extendemos la notación de UID para incluir un número del identificador a cada estado que es accedido; usando la notación "." como:

<UID_nombre.id_estado>

Para acceder a una transición, usamos los operadores específicos *destino* y *origen* aplicado a estados (por ejemplo *Destino(UID_nombre.id_estado)*). Para acceder a un estado de un elemento también podemos usar la notación del punto (por ejemplo *UID_nombre.id_estado.nombreDelElemento*).

Encima de esto, necesitamos dos operadores, uno (AddConnection) para agregar un link a un estado de un UID y otro (AddOperation) para agregar una operación a un estado o un elemento de estado. La Operación AddConnection crea un nuevo estado (especial) nombrado en el UID compuesto y conecta (a través de una transición) éste estado con algún estado en el UID actual (base). El nombre de este nuevo estado es el nombre subrayado del concern navegacional que estamos componiendo.

Para componer UIDs por "extensión", donde el resultante UID explícitamente agrupan los UIDs, necesitamos agregar a nuestro idioma de la composición los operadores siguientes: Merge y AddTransition. El operador Merge se usa para

combinar dos estados (por ejemplo Merge UID1_nombre.id_estado con UID2_nombre.id_estado). Su semántica es equivalente a la unión entre conjuntos. El AddTransition crea una nueva transición, conectando dos estados de diferentes UIDs (por ejemplo AddTransition UID1_nombre.id_estado a UID2_nombre.id_estado).

La diferencia entre AddTransition y AddConnection es que mientras el primero crea una transición entre dos estados de UIDs, cada estado pertenece a un UID diferente y por lo tanto contribuye al engrosamiento del UID (base); agregando más estados y transiciones a él explícitamente. Mientras que el último agrega una transición de un estado UID base a un solo estado que representa la posibilidad de navegar a algún estado de la interacción definido externamente en otro UID. Los UIDs compuestos resultantes están en diferentes niveles de abstracción

La sintaxis propuesta es:

```
Compose <UID_Base> with <UID_NavC1, … UID_NavCk>
{<UID_Base, UID_Base.State>
        [Merge | AddTransion | AddConnection | AddOperation] [to | with]
                <UID_NavCi.State, UID_NavC.Operation, NavCi>}
}
```

Diferentes combinaciones de concerns pueden ser compuestas y mostradas al los clientes para verificación y, mas importante aún, para anticipar el estudio del "look and feel" de la estructura navegacional del futuro sistema. Al cliente se le ofrece la oportunidad de elegir cuando y donde los concerns navegacionales (alguno de los cuales están usualmente fuertemente ligados a las políticas del negocio) pueden aparecer en la estructura navegacional base del sistema. Esto significa, por otro lado, que diferentes funciones de composición, para brindar rutas diferentes de navegación, deben ser analizadas para elegir la que mejor encaja en los criterios establecidos tal como usabilidad.

Por otro lado, composiciones incrementales e iterativas sostienen mejoras de evolución, como nuevos concerns (ejemplo: Concerns volátiles) que pueden ser fácilmente añadidos y remover concerns no queridos del sistema existente sin tener que cambiar la definición de los concerns ya detallados e implementados.

5.3 Aspectos detectados

Nosotros proponemos una temprana identificación de los crosscutting concerns, lo cual notifica a los diseñadores de su existencia y esto impacta en las posteriores etapas del desarrollo de software notificando la existencia de los aspectos para un mejor mapeo en el modelo conceptual o la definición de un aspecto explicito.

Se especifica comportamiento aspectual detectado en la etapa obtención de requerimientos por medio de la representación de una secuencia de pasos similar a los casos de usos utilizando la técnica [27] .

De un UID se puede derivar un caso de uso sencillo enumerando los elementos de la ruta a navegar deseada. Utilizando esta información podemos indicar "join points" indicando:

- Concern (aspecto): al que pertenece
- Condición: ante que condición se bifurca el hilo de ejecución.
- Flujo: UID utilizado en la bifurcación cuando la condición se satisfaga

- Punto: Numero de paso de la enumeración en el que se debe observar el estado para realizar la bifurcación.

Veamos el siguiente ejemplo:

Pasos de ***Generar Orden Compra***:

1. Seleccionar Carrito
2. Seleccionar una dirección existente o indicar una en ese momento.
3. Oper: llevarADireccion
4. Seleccionar método de pago
5. Oper: comprar
6. Fin

Aspecto: Autenticación
Condición: No loqueado
Flujo:***Uid_login***
Punto: 2

Básicamente, en el ejemplo se muestra la incorporación de comportamiento deseado para el concern de autenticación del proceso de generar una orden de compra. Se quiere que el usuario esté logueado en el sistema para poder realizar la compra.
En la técnica Theme/Doc [28] , se puede expresar los crosscutting concern a través de un grafo. Es una manera diferente de indicar las dependencias entre requerimientos.

5.4 Tabla de Crosscutting

Habiendo definido las reglas de composición y localizado los crosscutting, nos disponemos a construir una tabla de cruces de concerns que brinda un mapa conceptual concreto sobre el tipo de relación que existe entre ellos.

Por el momento, hemos definido dos tipos de cruces:

- Navegacionales: Se deben indicar en el caso de que exista una regla de composición entre los dos concerns involucrados.
- Aspectuales: Nace de la definición de aspectos.

	C1	C2	C3	C4
C1		Asp	Nav	Asp/Nav*
C2				
C3				Asp/Nav
C4				

Esta tabla nos informa rápidamente le cruce de concern a modo de índice resumiendo los componentes del sistema en relaciones.

Este proceso podría ser automatizado, ya que no se necesita la mano del ingeniero en requerimientos.

6 UIDRe: Herramienta Case de UID para el proceso de análisis de requerimientos

6.1 Introducción

Para acompañar nuestra teoría, construimos una herramienta que es capaz de almacenar los requerimientos obtenidos en la etapa de análisis.

Nuestra herramienta permite la creación de concerns con sus respectivos requerimientos, y adicionalmente la construcción gráfica de los Diagrama de Interacción de Usuario (UID). Una vez diseñados (graficados) estos diagramas, se permite la posibilidad de generar reglas de composición que posteriormente se utilizarán generar un mapa integrador de interacción. En consecuencia el mapa refleja los diferentes cruces que se tiene entre los concerns.

Esta herramienta es de gran utilidad pues es el paso inicial de una plataforma de análisis. Esto nos permite hacer un análisis exhaustivo de las aplicaciones, reducir su complejidad, seguirla detalladamente y conseguir una mejor modularidad en la aplicación final.

El objetivo primario de nuestra herramienta es la captura de requerimientos para identificar los concerns. Por lo tanto ponemos énfasis en facilitar la carga de datos en UID. Deseamos no agregar parámetros que compliquen al usuario a la hora de expresar la información.

La herramienta tiene las siguientes funcionalidades básicas:

- Gestión de propósitos.
- Diseños de UIDs.
- Composición de reglas.
- Almacenamiento en un formato estándar de esta información.

Hemos visto que diferentes concerns pueden afectarse entre si. Y como resultado de estos crosscutting concerns, obtenemos aspectos dispersos en el sistema y enredados con otros concerns. En ésta herramienta presentamos una forma para modelar y componer concerns navegacionales en aplicaciones generalmente orientadas a las Web. Demostrando cómo construir “vistas” parciales de la navegación con los diagramas de la interacción del usuario, analizando y definiendo reglas correspondientes de la composición.

Tomaremos los requerimientos obtenidos de Amazon, y mostraremos algunos ejemplos de lo que genera la herramienta así como la creación del mismo.

6.2 Tecnologías y arquitectura

La herramienta esta construida sobre la plataforma Eclipse utilizando las tecnologías EMF (herramienta de modelado y de generación de código), el framework de edición grafica (GEF), y RCP (framework para la creación de clientes ricos) , las mismas serán detalladas en el Apéndice A: Frameworks Utilizados.

Este conjunto de herramientas permitieron utilizar el paradigma “Arquitera Basada en Modelo” (Model-Driven Architecture, MDA). MDA provee un conjunto de guías para la construcción de especificaciones expresadas en modelos. Utilizando la metodología MDA, la funcionalidad del sistema puede ser primero definida como modelo independiente del sistema.

Para ello, la primer etapa fue modelar el dominio y el resultado fue diagrama de la Ilustración 9.

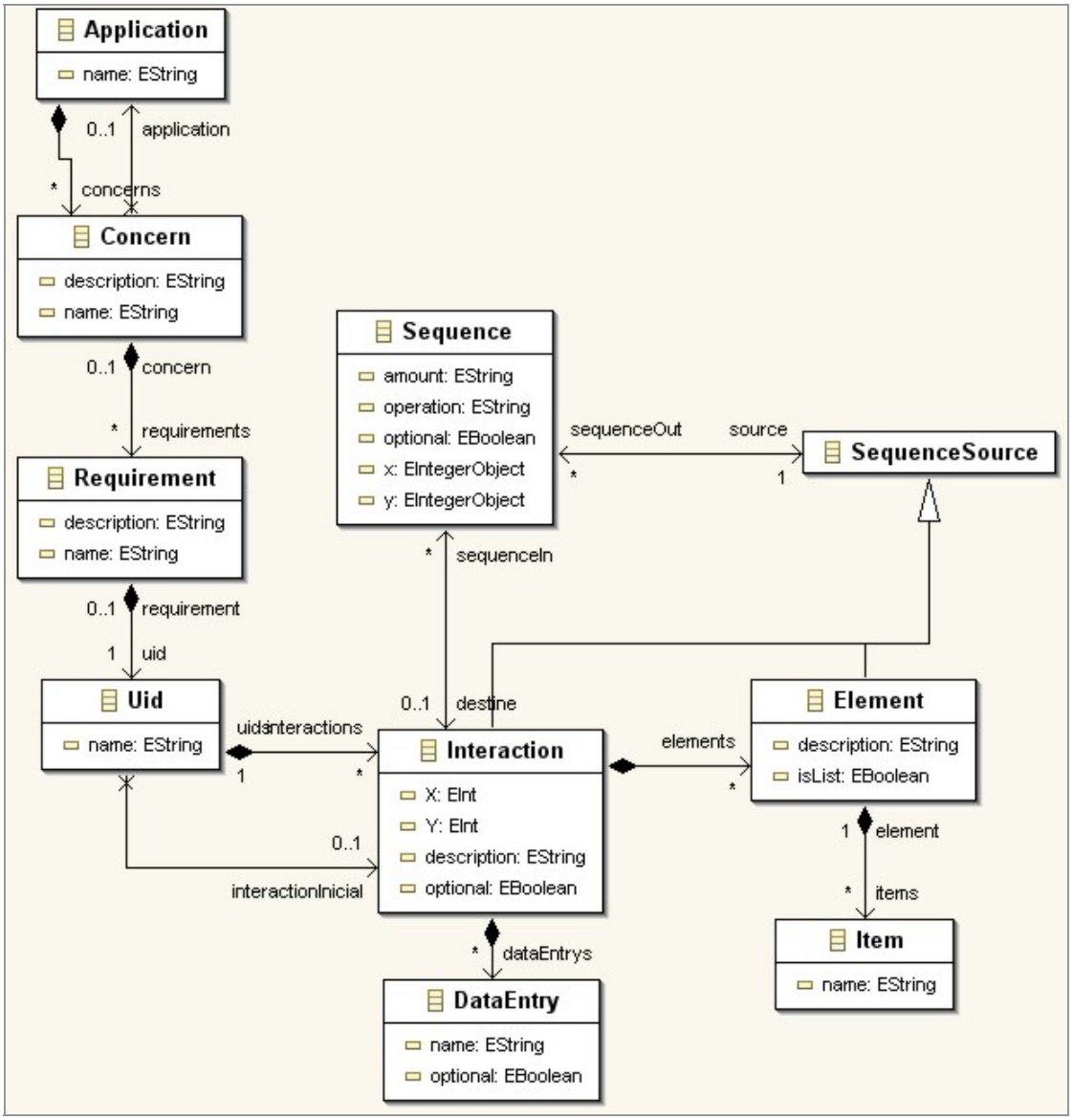


Ilustración 9: Modelo del dominio de la herramienta

Hemos utilizado el Framework de Modelado de Eclipse, este nos permitió describir el modelo anterior en Ecore Model, ya que este es su modelo de objetos. Este conjunto de clases forma nuestra API[8] principal, la cual fue empaquetada de forma

[8] (Application Program Interface). Conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación. Cuando se intenta estandarizar una plataforma, se estipulan unos APIs comunes a los que deben ajustarse todos los desarrolladores de aplicaciones. Herramientas de programación para rutinas, protocolos y software.

aislada para que pueda ser utilizada independientemente a la interfaz de usuario (en nuestro caso, una interfaz gráfica RCP).

Teniendo ya nuestro modelo almacenado en un documento digital, utilizamos las herramientas provistas por EMF para obtener un conjunto de clases Java derivado del modelo, soporte de persistencia para ellas y una API de acceso reflexivo al modelo. Este nueva API generada automáticamente nos permitió manipular y persistir objetos de nuestro modelo fácilmente.

El mecanismo de persistencia para almacenar la información de las instancias del modelo fue el estándar XMI. XMI (XML de Intercambio de Metadatos) es una especificación para el Intercambio de Diagramas, la especificación para el intercambio de diagramas fue escrita para proveer una manera de compartir modelos UML[9] entre diferentes herramientas de modelado. En versiones anteriores de UML se utilizaba un Schema XML para capturar los elementos utilizados en el diagrama; pero este Schema no decía nada acerca de cómo el modelo debía graficarse.

Luego teniendo en nuestras manos los mecanismos de manipulación del modelo, realizamos la construcción del editor gráfico. Para su construcción utilizamos el Framework de Edición Gráfica de Eclipse. Fácilmente y rápidamente construimos la capa de presentación de nuestra herramienta bajo la arquitectura de diseño MVC (Model View Controler).

Todos estos módulos citados fueron implementados en plugin diferentes lo cual permite un débil acople entre ellos y una fácil reutilización. Está metodología se adapta perfectamente a la arquitectura de la Plataforma Eclipse lo cual permite componer y re utilizar diferentes soluciones ya desarrolladas.

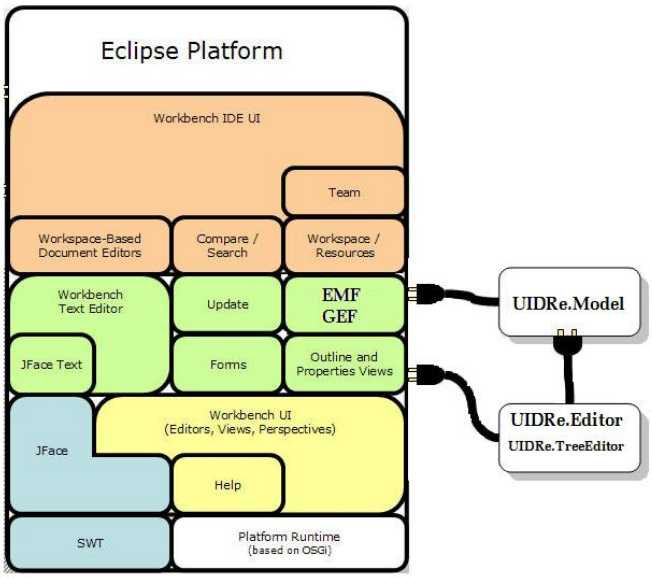


Ilustración 10: Arquitectura de herramienta UIDRe

[9] UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Como se puede apreciar en la Ilustración 10: Arquitectura de herramienta, hemos construido y ensamblado los siguientes s Plugin (módulos):

- UIDRe.Model: Como se comentó, incluye todo el modelo y los componentes adicionales para su utilización.
- UIDRe.Editor: Es el módulo de edición de UID, permite el diseño asistido por el computador de los diagramas.
- UIDRe.TreeEditor: Es el módulo de gestión de los Concerns y de los requerimientos, a diferencia del módulo anterior, éste se encarga de propósitos relacionados con los requerimientos, tales como administración y composición por medio de las reglas mencionadas. Este no se una herramienta CASE,

La herramienta fue construida con el patrón MVC[10]. En éste, el modelo utilizado fue enriquecido con el patrón Observador[11] para poder notificar al modelo que notifique al controlador sobre los cambios. Es importante comentar que el Framework EMF posee un excelente API de reflexión con lo cual se facilitó todo el desarrollo.

La capa de Controladores fue casi un espejo del modelo. Estos controladores eran los encargados de notificar a la vista de los cambios y permitir una interacción clara de información entre el modelo y la vista. Por ultimo, la vista encapsula la captura de todos los eventos y los propaga al modelo. Tanto la capa de controladores como los objetos de la vista se encuentran en el plugin UIDRe.Editor.

Como consecuencia, la información recopilada por nuestro entorno, con un previo análisis, puede ser inyectada a otros motores ya existentes como por ejemplo: generador de clases Java, componentes MVC (Struts o Tapestry), EJB, etc. De esta forma se podría conseguir un prototipo complejo del sistema relevado dado que se obtendrían las tres capas típicas de un sistema contemporáneo: capa de datos, capa de negocio y capa de presentación.

6.3 Definición

Nuestra herramienta UIDRe permite un sencillo relevamiento de requerimientos para ser analizado en un futuro.

Principalmente, permite crear grupos de requerimientos llamados concerns. Estos concerns representan un propósito bien definido del sistema. Estos son requisitos que el futuro sistema debería satisfacer al final de su desarrollo. En consecuencia, la herramienta brinda la alternativa de asociar a cada requisito un UID. Estos diagramas, a través de su sencillez y poder de expresión, facilitan la comunicación con el usuario y, por ende, disminuye la distancia entre el ingeniero de requerimientos y el cliente.

El utilitario permite gestionar altas, bajas y modificaciones de concerns y sus respectivos requerimientos. Además posee un editor de UID para aquellos diagramas asociados a requerimientos. Este último permite construir diagramas uniendo los siguientes conceptos:

- Interacciones

[10] Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

[11] El patrón Obervador define una dependencia del tipo uno-a-muchos entre objetos, de manera que cuando uno de los objetos cambia su estado, el observador de encarga de notificar este cambio a todos los otros dependientes. ([30])

- Secuencias
- Entradas de datos
- Elementos de información

El usuario intuitivamente puede representar gráficamente requerimientos de alta complejidad.

Una vez definidos los UIDs del sistema podemos componerlos utilizando reglas de composición alcanzar como resultado un mapa general de navegación. Este resultado permite confirmar los requerimientos con el cliente.

La herramienta permite la creación de concerns. Utilizaremos los concerns definidos en el caso de estudio mencionamos con anterioridad.

Luego se procede a descomponer el conjunto de requerimientos separándolos en estos concerns. Veamos algunos ejemplos de estudio aplicados a algunos de los concerns anteriormente mencionados:

Concern	**Requerimientos**
Buscar Producto	R1: Los productos van a ser organizados dependiendo del genero R2: Se debe permitir buscar productos mostrando información básica R3: Se debe permitir diferentes estrategias de búsqueda de un producto, por Ej. titulo
Comentarios	R4: Los comentarios pueden ser valuados por su información R5: Se pueden realizar comentarios de un producto
Foro	R6: Los foros pueden ser respondidos R7: Los productos pueden tener foros R8: Para iniciar un foro el usuario debe haber adquirido el producto
Historial	R9: Se debe registrar cada visita a un producto R10: Visualizar el historial de productos

Una vez que encontramos planteamos todos los requerimientos podemos generar los UID correspondientes a nuestro caso de estudio. Algunos UID son tan triviales que no plasman nada importante, es por ello que no es necesario hacer los diagramas, el usuario es el encargado de tomar estas decisiones de diseño.

Veamos un ejemplo de generación del UID para un requerimiento en especial. En esta sección analizaremos un requerimiento del concern de Foro.

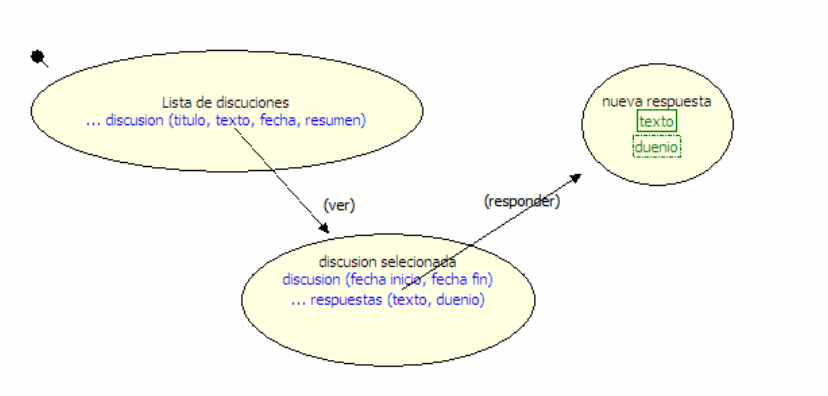


Ilustración 11: R6: Los foros pueden ser respondidos

Además la herramienta permite agregar, modificar o eliminar reglas de composición. Estas reglas de composición se establecen entre dos UID, la misma requiere que se elijan las interacciones principales para poder aplicar la regla, aquí un nuevo diagrama de interacción del usuario será generado, permitiendo al diseñador tomar las conclusiones deseadas acerca de la conveniencia o no de la aplicación de la regla.

6.4 Conclusiones de la herramienta

El uso de esta herramienta facilita al desarrollador a tener un esquema de cuales son los concerns principales, que requerimientos incluye cada uno y en caso de que decida su representación grafica posee el editor grafico de UIDs.

Esto le ayudara a tener una idea conceptual y gráfica de lo que espera de su aplicación. Podemos destacar que la composición de reglas le permite al desarrollador a tener un esquema en donde pueda tomar decisiones de diseño y evaluar la integridad del sistema.

El aporte inmediato es la facilidad de digitalización de la información recopilada en la etapa de análisis en un formato estandarizado derivado de XML. Bajo un entorno amigable podemos interactuar con un cliente rico que facilita la carga de requerimientos y de UID. En todo el proceso se evita trabajar con archivos planos de XML lo cual es de gran ayuda.

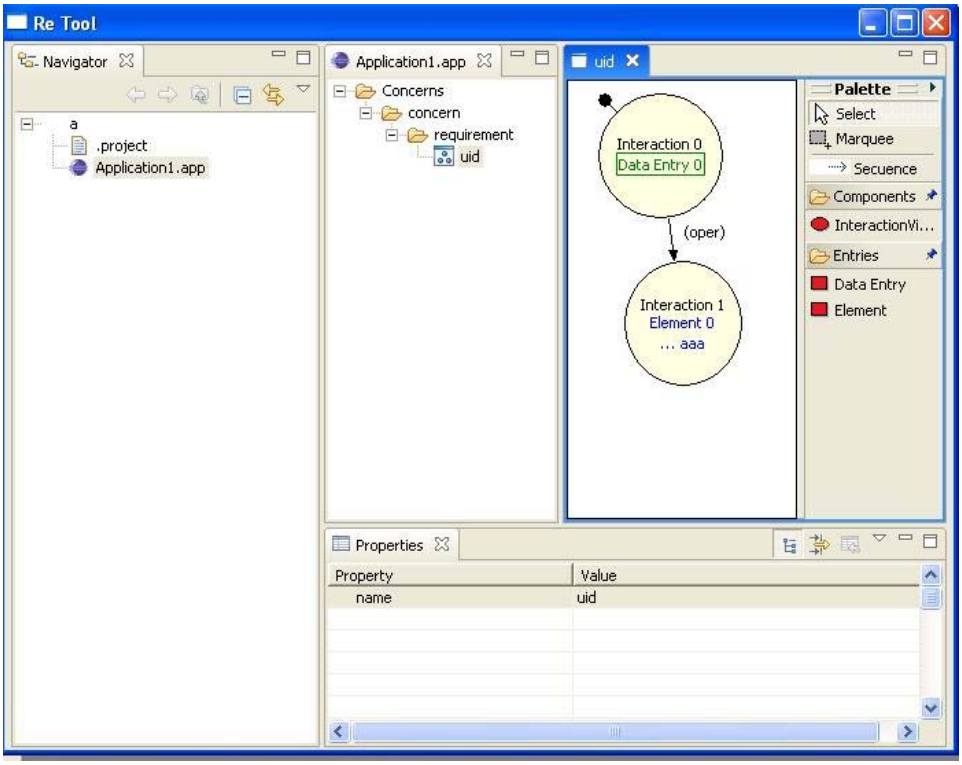
En la sección Introducción El futuro de la herramienta, comentaremos el futuro deseado para la herramienta

6.5 Ejemplo construido con la herramienta

Introduciremos nuestro producto, explicando en cada caso el uso adecuado.

Ilustración 12: Nuestro Producto

La siguiente es una descripción funcional de la herramienta. Se detalla a continuación cada uno de los pasos necesarios para interactuar con la misma.



Lo primero que veremos es una vista general del árbol de concerns, se detallan los mismos con sus respectivos requerimientos y uids asociados.

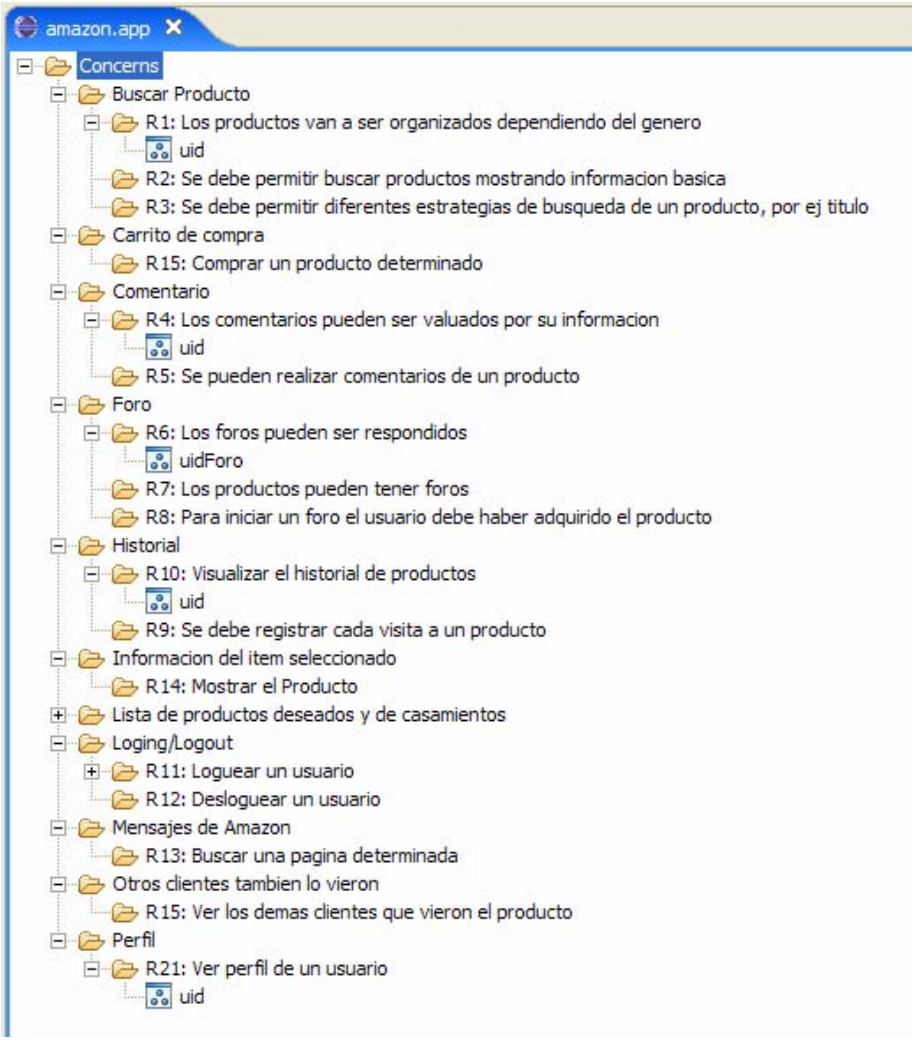


Ilustración 13: Vista del conjunto de concerns, requerimientos y UID

A continuación se muestra el menú de la aplicación (conjunto de concerns), entre ellos se encuentra: la creación de un concern y la visualización completa del árbol.
La creación de un concern se realiza simplemente haciendo clic sobre el botón derecho del mouse, previamente seleccionada la aplicación a la cual se quiere agregar uno.
La visualización del árbol mostrara la figura mencionada anteriormente: La vista del conjunto de concerns, requerimientos y UIDs.

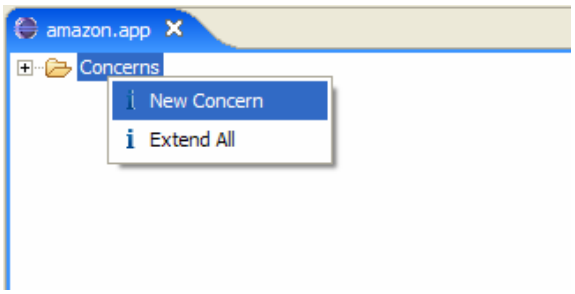


Ilustración 14: Menú de la aplicación principal

Para la creación de un concern es necesario ingresar el nombre del mismo.

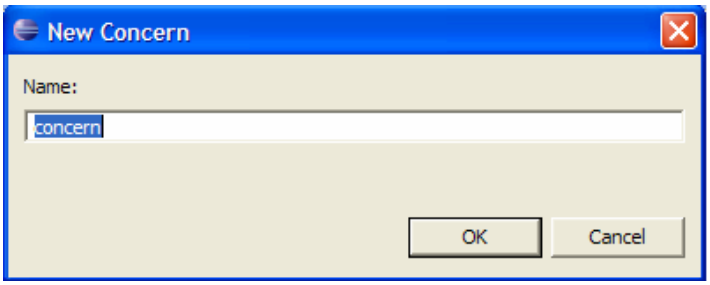


Ilustración 15: Nuevo concern

Luego tenemos el menú de cada concern, el cual permite crear un nuevo requerimiento, modificar el nombre del concern o borrarlo.

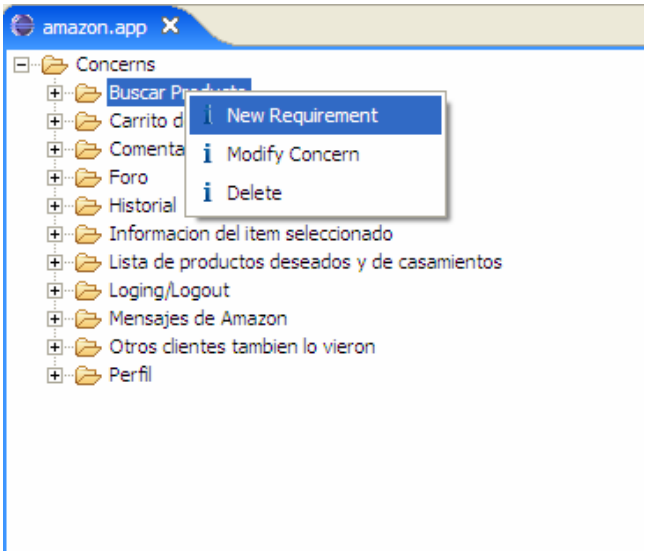


Ilustración 16: Menú de los concerns

Igualmente que para la creación de un concern, el requerimiento permite el ingreso del nombre del mismo. Así como la modificación presentara una misma pantalla grafica que permita modificar dicho nombre.

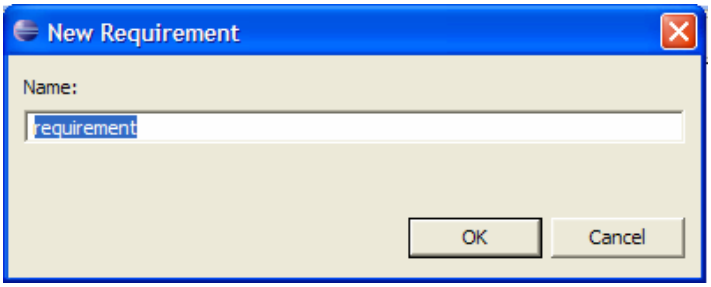


Ilustración 17: Nuevo requerimiento

A continuación veremos el menú de los requerimientos, el mismo permite la creación de un nuevo UID, la modificación y borrado de ese requerimiento.

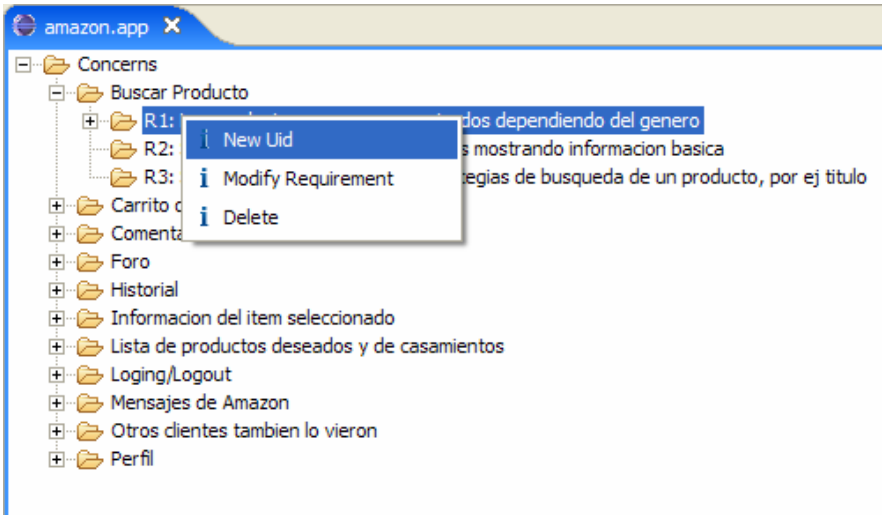


Ilustración 18: Menú de los requerimientos

Tato la creación como la modificación requiere el llenado del campo de nombre.

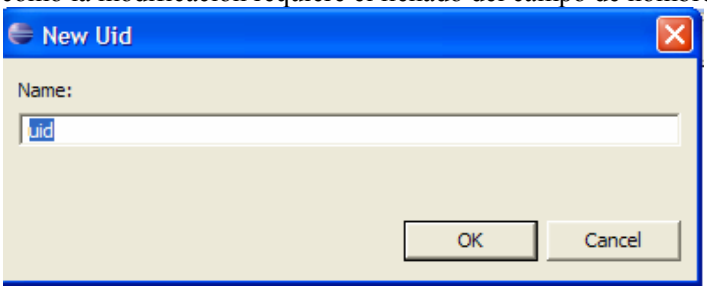


Ilustración 19: Nuevo UID

Además existe el menú de cada UID, el mismo contiene las operaciones de mostrar la vista del editor grafico y la modificación y borrado del mismo.

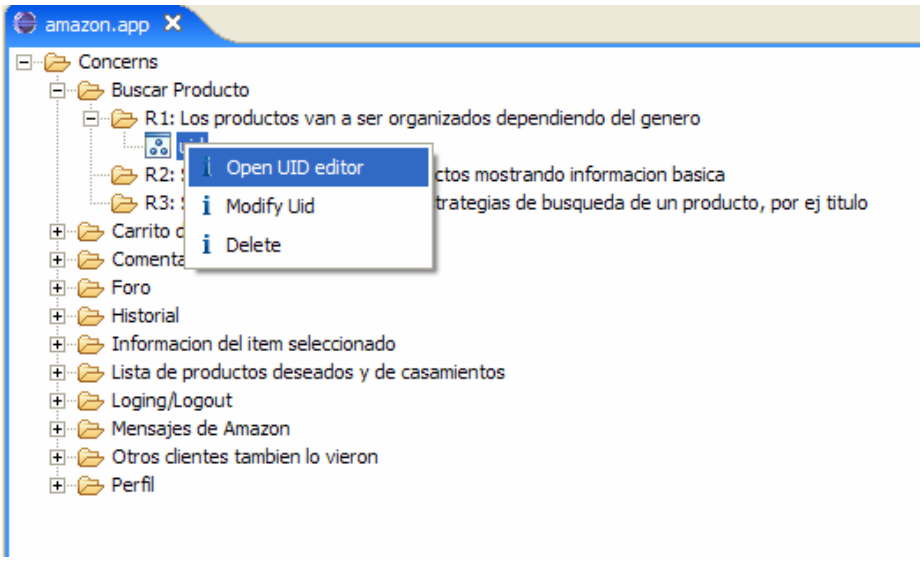


Ilustración 20: Menú de los UIDs

Cuando se selecciona la opción de abrir el editor, se podrá visualizar el esquema gráfico del UID, permitiendo editarlo: crear interacciones, secuencias, datos de entrada y elementos (los cuales pueden ser tanto simples como compuestos, por ejemplo lista de elementos)

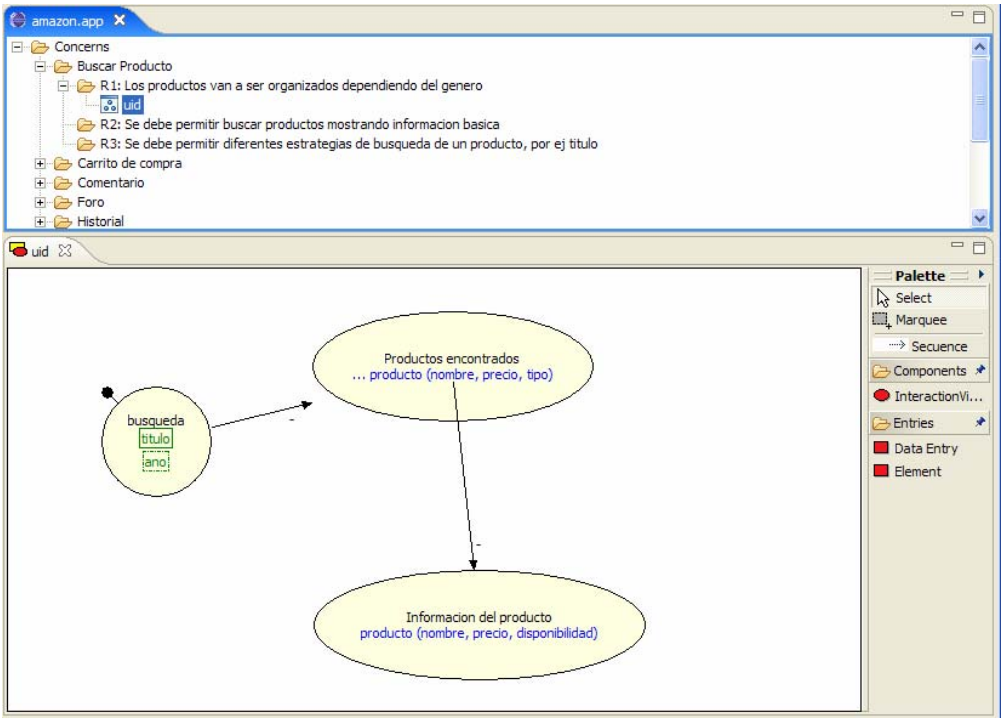


Ilustración 21: Vista del editor de UID

Cuando se seleccionan dos UIDs, además de visualizar en el menú las operaciones previamente vistas, se permite la composición de los mismos.

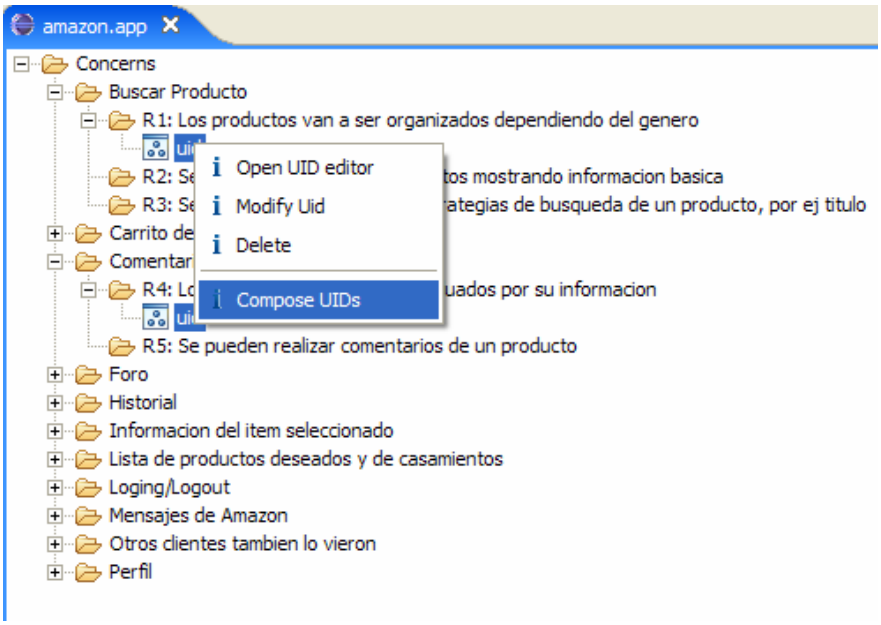


Ilustración 22: Menú de composición de UIDs

Esta acción genera la apertura de una nueva ventana que permite: agregar reglas, modificarlas, borrarlas y obtener una vista previa de la composición.
En la lista se muestran las composiciones creadas. Notar que la misma posee, la descripción de la operación realizada y las interacciones principales que fueron utilizadas para la composición.

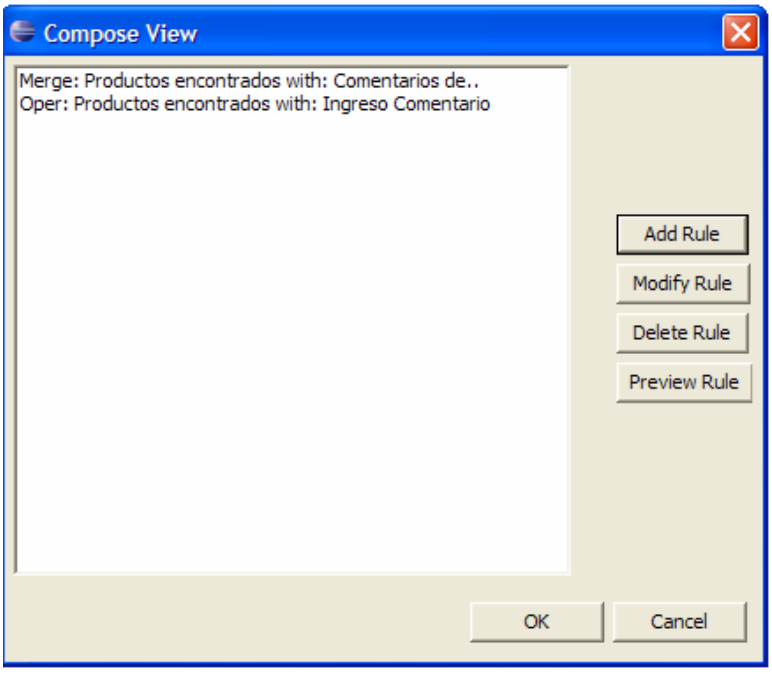


Ilustración 23: Ventana de Composición de UIDs

En el caso de que queramos agregar una nueva regla, debemos seleccionar la interacción fuente y la interacción destino, así como también la operación de composición a realizar.
En el caso de la modificación de la regla ocurre lo mismo, se permite cambiar una o ambas iteraciones y la operación asociada a la misma.

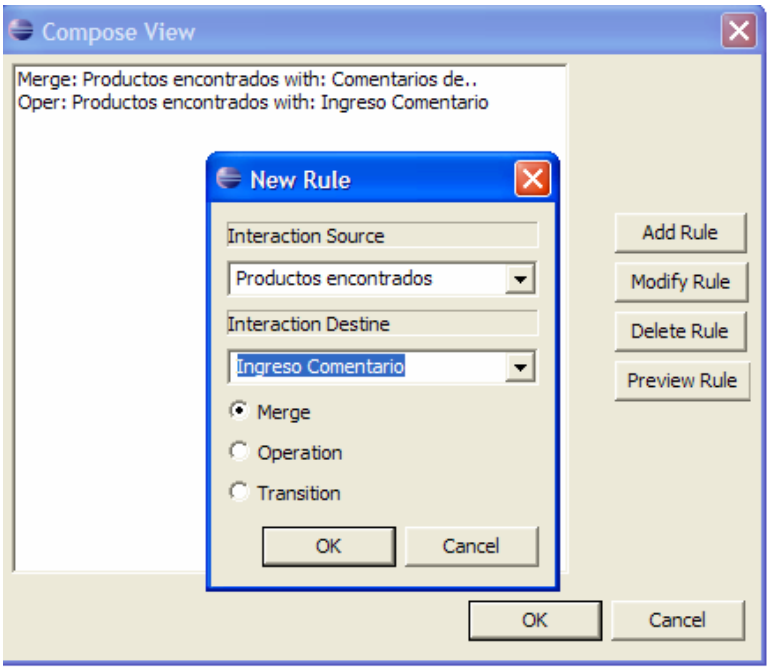


Ilustración 24: Agregar una nueva regla de composición

Como se menciono anteriormente, existe una visualización de cada una de las reglas generadas. La misma será mostrada en un editor gráfico de UIDs.

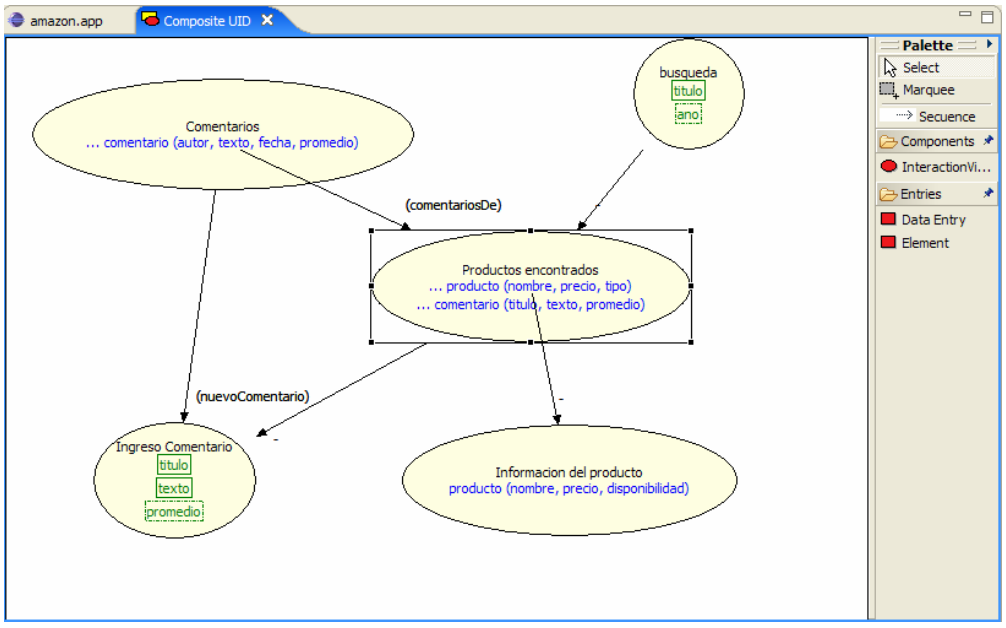


Ilustración 25: Vista del editor, luego de la composición de dos UIDs

Por ultimo es importante contemplas el código XML que se genera. En nuestro caso solo veremos una reducción del mismo, es decir, mostraremos el XML generado de la aplicación que solo tiene el concern de búsqueda del producto, con sus respectivos requerimientos y UIDs asociados.

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:ar.edu.unlp.info.model="http://ar.edu.unlp.info.model">
  <ar.edu.unlp.info.model:Secuence/>
  <ar.edu.unlp.info.model:Secuence
destine="/11/@concerns.0/@requirements.2/@uid/@interactions.1"
source="/11/@concerns.0/@requirements.2/@uid/@interactions.0"/>
  <ar.edu.unlp.info.model:Secuence
destine="/11/@concerns.0/@requirements.2/@uid/@interactions.2"
source="/11/@concerns.0/@requirements.2/@uid/@interactions.1/@elements.0"/>
  <ar.edu.unlp.info.model:Secuence operation="comentariosDe"/>
  <ar.edu.unlp.info.model:Secuence operation="nuevoComentario"/>
  <ar.edu.unlp.info.model:Secuence operation="responder"/>
  <ar.edu.unlp.info.model:Secuence operation="ver"/>
  <ar.edu.unlp.info.model:Secuence/>
  <ar.edu.unlp.info.model:Secuence operation="ok" y="126" x="239"/>
  <ar.edu.unlp.info.model:Secuence operation="editar" y="161" x="376"/>
  <ar.edu.unlp.info.model:Secuence operation="registrarse" y="284"
x="185"/>
  <ar.edu.unlp.info.model:Application name="Concerns">
    <concerns name="Buscar Producto">
      <requirements name="R2: Se debe permitir buscar productos mostrando
informacion basica"/>
      <requirements name="R3: Se debe permitir diferentes estrategias de
busqueda de un producto, por ej titulo"/>
      <requirements name="R1: Los productos van a ser organizados
dependiendo del genero">
        <uid name="uid"
interactionInicial="/11/@concerns.0/@requirements.2/@uid/@interactions.0">
          <interactions secuencesOut="/1" description="busqueda" Y="98"
```

```
X="78">
            <dataEntrys name="titulo"/>
            <dataEntrys name="ano" optional="true"/>
          </interactions>
          <interactions description="Productos encontrados" Y="36" X="252"
secuencesIn="/1">
            <elements secuencesOut="/2" description="producto"
isList="true">
              <items name="nombre"/>
              <items name="precio"/>
              <items name="tipo"/>
            </elements>
          </interactions>
          <interactions description="Informacion del producto" Y="249"
X="253" secuencesIn="/2">
            <elements description="producto">
              <items name="nombre"/>
              <items name="precio"/>
              <items name="disponibilidad"/>
            </elements>
          </interactions>
        </uid>
      </requirements>
    </concerns>
  </ar.edu.unlp.info.model:Application>
</xmi:XMI>
```

7 Caso de estudio

Tomaremos el siguiente caso de estudio para analizar el uso de la herramienta, podemos observar que cada imagen de diagrama de interacción de usuario fue tomada de la herramienta.

7.1 Modalidad de trabajo

Como caso de estudio proponemos el sitio Amazon dado que es uno de los más completos y dinámicos que existen en la actualidad. Amazon es el sitio más volátil que hemos encontrado, constantemente define nuevos servicios y reglas de negocios.

> Amazon.com es una compañía Americana de comercio electrónico. Esta fue una de la primer compañía de venta de bienes sobre Internet y fue uno de los iconos de la burbuja de Internet de los 90′s.

Partiremos de un proceso de ingeniería inversa para poder identificar requerimientos que serán útiles para ejemplificar la metodología propuesta. Agruparemos los requerimientos en concerns permitiendo de esta forma localizar diferentes intereses de la aplicación.

Una vez obtenido los requerimientos, definiremos los UIDs correspondientes a los diferentes requerimientos. En esta etapa se identificaran los diferentes flujos de información, información requerida en cada paso y las alternativas de navegación.

Posteriormente utilizaremos los diagramas en el proceso de derivación de un modelo de clases, se completará el diagrama y se analizará las diferentes vistas obtenidas.

7.2 Requerimientos obtenidos del proceso de ingeniería inversa

Los requerimientos obtenidos de Amazon serán agrupados en concerns. Para cada uno definiremos un conjunto de requerimientos.

No haremos un análisis exhaustivo de todos los requerimientos, sino aquellos que destaquen comportamiento crosscuting.

Ilustración 26: Concern Producto

Concern producto

- Por cada producto, se conocerá su nombre, descripción e imagen.
- Además, se conocerán características especiales que corresponden al producto. En el caso de un producto electrónico se tendrán las especificaciones técnicas, pero en el caso de un producto de la índole textil, se conocerá su composición.
- Se debe permitir conocer la información relativa a cada producto, en el caso de un libro a su autor, en el caso de un TV a su marca, etc.

Visitas recientes

- Se llevará un control de todos los productos accedidos por el usuario.
- Se le dará acceso a los últimos 5 productos visitados al usuario.

Clientes que lo compraron también compraron…

- Se le dará acceso a cinco productos que hayan adquirido clientes que también compraron el producto que es visitado.

Clientes que lo visitaron también visitaron…

- Se llevará un control de todos los productos accedidos por los usuarios.

- Se dará acceso a los productos visitados por otros clientes que visitaron el producto actual.

Ventas

- Se dará a conocer la disponibilidad de cada producto, el precio, los descuentos, y disponibilidad de usado
- Cada producto puede ser agregado a un carrito de compras, donde se almacenará la compra tentativa del usuario.
- Una vez completa la selección de productos, el usuario podrá generar la orden de compra. El usuario deberá autenticarse en caso de que no lo haya hecho, y luego deberá: seleccionar un destino, definir las opciones de compra, y por ultimo elegir la forma de pago.

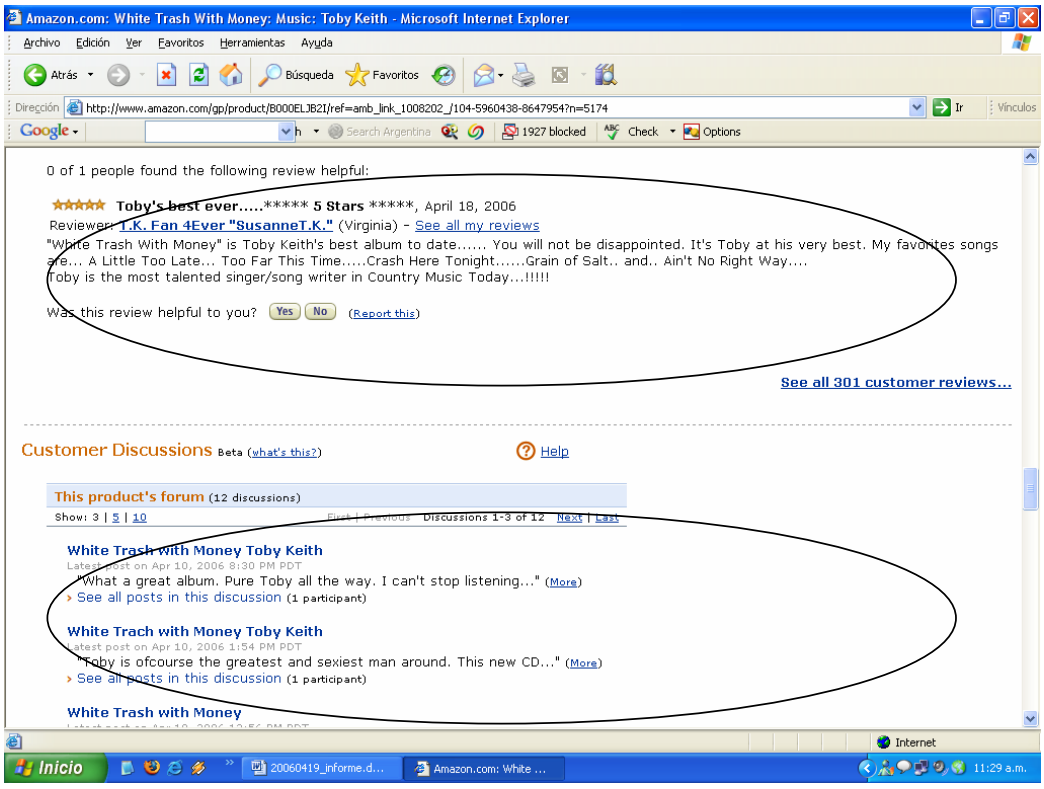


Ilustración 27: Vista de la página de Amazon, encontrando otros concerns

Valorización y Comentarios

- Los usuarios podrán hacer comentarios sobre los productos.

- Los diferentes usuarios podrán valorar un comentario indicando que tan útil le resulto. El promedio de valoraciones será visualizado.
- Se podrá valorar el producto directamente.

Foro

- Los usuarios podrán hacer crear líneas de discusión en base a tópicos.
- Una vez iniciada la discusión, los diferentes usuarios podrán responder a las respuestas de otros usuarios.
- Para cada operación se requiere que el usuario, este logueado y haya adquirido el libro.

7.3 UIDs de los requerimientos

En este punto completamos la definición de los requerimientos con los Diagramas de Interacción de Usuario. Estos diagramas son sumamente expresivos; permiten expresar fácilmente requerimientos complejos gráficamente facilitando la comunicación con el cliente.

Esta etapa consta de dos partes, la primera consiste en el diseño de los UIDs, donde se definen las entradas de datos del usuario y los elementos de información involucrados. En la segunda etapa, se definen las reglas de composición que permiten definir nuevas relaciones entre los componente de los UIDs.

7.3.1 Diseño de UID

A continuación describimos algunos UIDs del los diferentes concern el sistema. Estos Diagramas tienen la cualidad de representar gráficamente un caso de uso, incluyendo los elementos de información involucrados y entradas de datos del usuario.

1. ***Busqueda***: UID para representar una búsqueda en base a un titulo y opcionalmente a un año. De los productos encontrados se debe visualizar información básica.

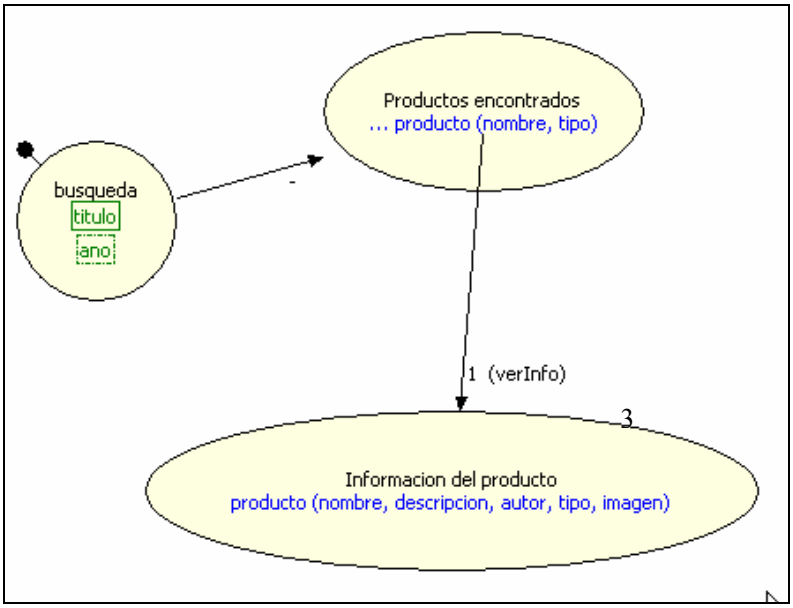


Ilustración 28: UID de búsqueda

2. ***Agregar producto a Carrito***: UID para agregar un producto al carrito. En este caso partiendo de una vista de un producto podemos agregar el producto actual al carrito ingresando el dato ***cantidad.*** Opcionalmente, se da la posibilidad de ver los productos dentro del carrito, permitiendo administrar la lista dando de baja productos del carrito o marcándolos para una futura compra.

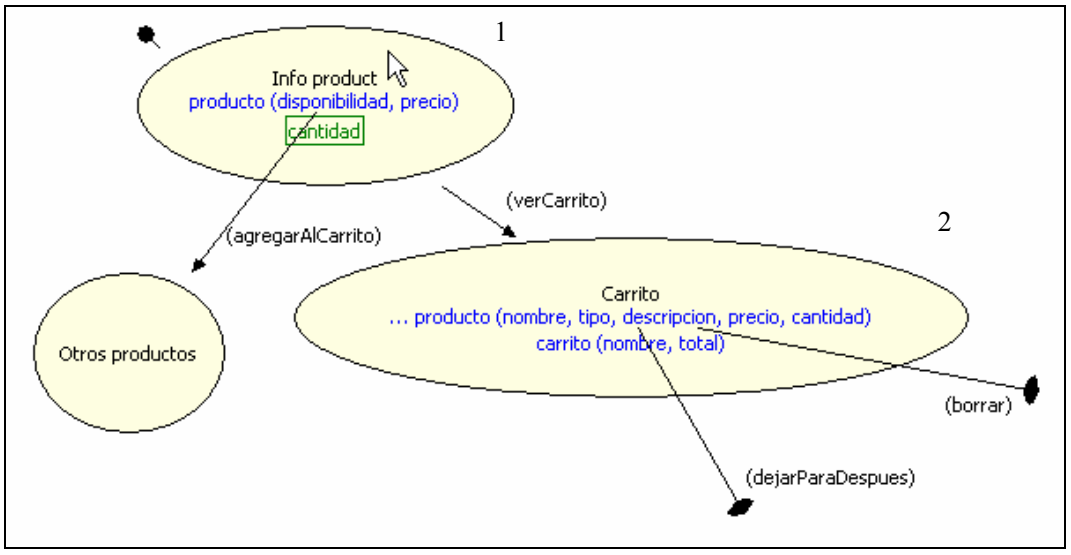


Ilustración 29: UID para agregar un producto a un carrito

3. ***Generar orden de compra (checkout)***: Este diagrama modela un conjunto de requerimientos complejos en donde se refleja el proceso de generar una orden de

compra. Partiendo desde la interacción Carrito comenzamos el proceso de checkout seleccionando una dirección existente (que ya haya sido ingresada) o ingresando una nueva. Una vez indicada la dirección continuamos con las opciones de entrega; se señalan tipo de envío (Express o normal), si se deben esperar la disponibilidad de todos los ítems para enviarlos o a medida que se dispone de ellos son enviados y si alguno de los productos es un regalo. Por ultimo indicamos el método de pago, para ello se solicita el tipo de pago, numero de cuenta, nombre del propietario y código de seguridad. Si todo es correcto la venta es concretada.

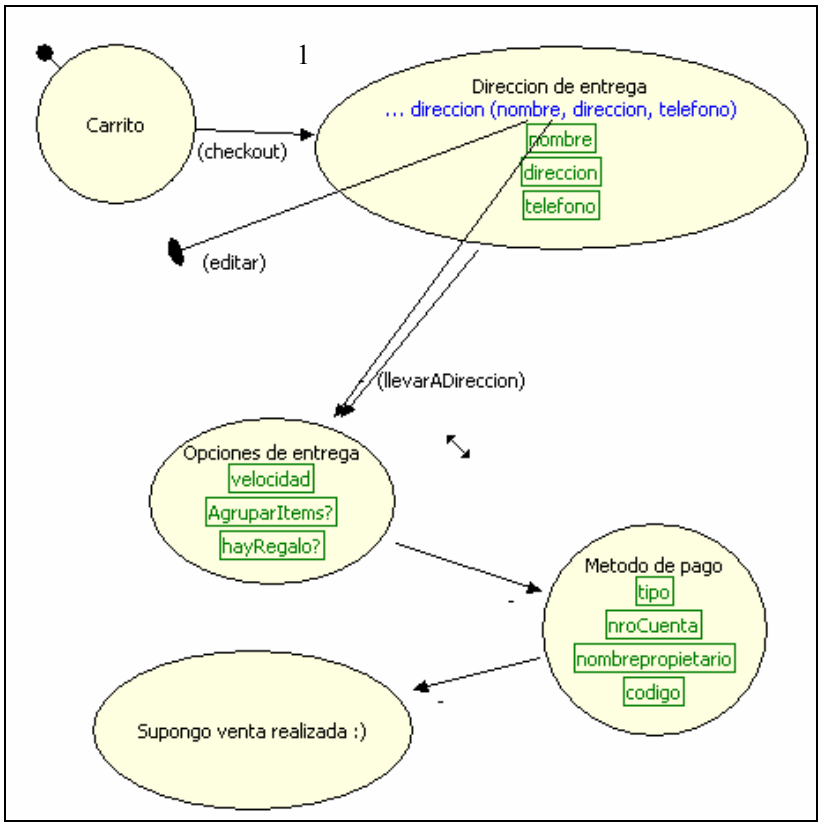


Ilustración 30: UID de checkout

4. ***Comentario***: Se debe dar la posibilidad de navegar sobre una lista de comentarios. Por cada uno de los ítems de elle debemos mostrar el autor, parte del texto, fecha y calificación. Desde este punto de partida podemos tanto agregar un nuevo comentario como navegar sobre un único comentario disponiendo de todo el texto.

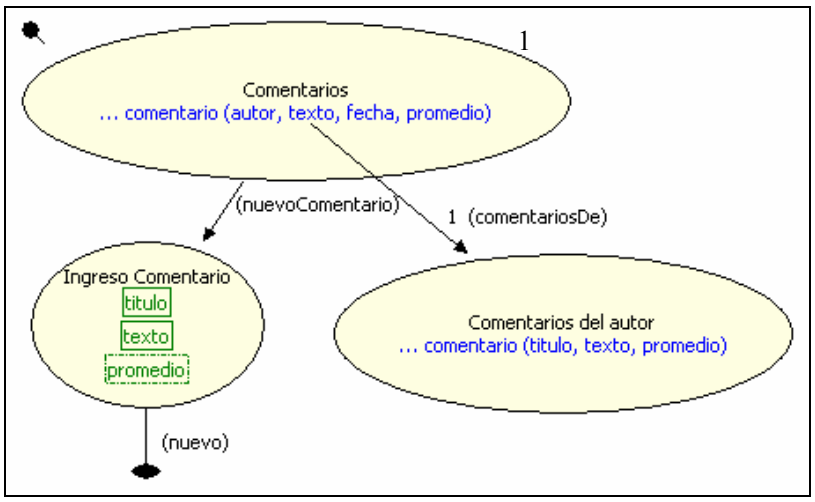


Ilustración 31: UID de comentarios

7.3.2 Reglas de composición

Las reglas de composición son de gran ayuda ya que nos permiten relacionar los diferentes concerns de nuestro sistema. Como consecuencia, analizándolo, podemos encontrar relaciones no explicitas y además podemos obtener un mapa total del sistema.

Aquí se declararán algunas reglas de composición:

```
Compose {
        Extends Busqueda.3.producto with  Carrito_agregar.1.producto
        Transition Carrito_agregar.2 to  OrdenCompra_generar.1
        Merge Busqueda.3 with  Comentario.1.producto
}
```

Como consecuencia tenemos el siguiente UID integrador:

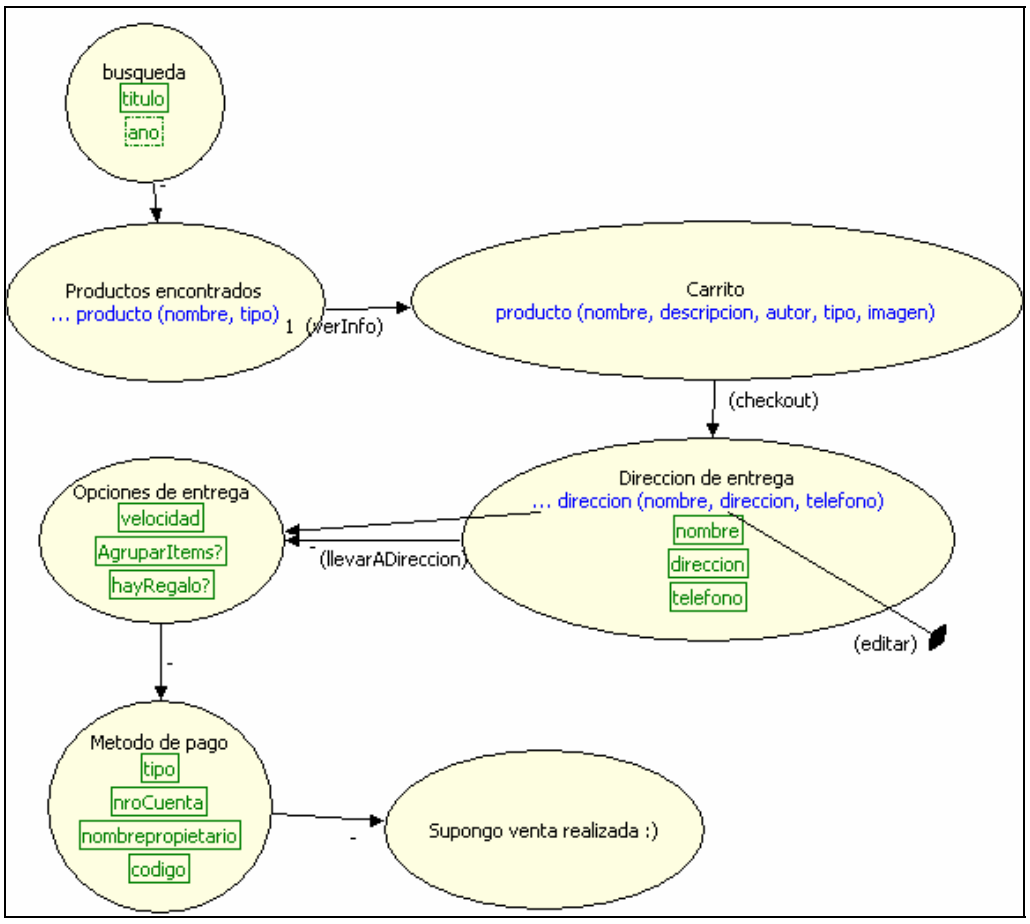


Ilustración 32: UID integrador

Hemos integrado el UID de búsqueda y de generación de orden de compra.

7.4 Identificando los aspectos

Utilizando la herramienta definida en el capítulo anterior, daremos un ejemplo de cómo especificar comportamiento no navegacional en un caso de uso derivado de un uid.

El siguiente ejemplo representa al caso de uso de "generar una orden de compra", en éste se enumeran todos los pasos realizados en el workflow.

Veamos el siguiente ejemplo:

Pasos de ***Generar Orden Compra***:

1. Seleccionar Carrito
2. Seleccionar una dirección existente o indicar una en ese momento.
3. Oper: llevarADireccion
4. Seleccionar método de pago
5. Oper: comprar
6. Fin

Aspecto: Autenticación
Condición: No loqueado
Flujo:***Uid_login***
Punto:2

En el ejemplo, hemos identificado un requerimiento crosscutting de Autenticación. Si el usuario no se encuentra logueado en el momento de indicar una dirección de envio, se rompe el flujo normal de navegación bifurcando al UID de login.

7.5 Tabla de Crosscutting

En este punto, ya hemos definido la composición de UIDs y publicado los aspectos encontrados, por ende producimos una tabla de crosscutting para tener un mapa conceptual de todos los cruces y sus tipos utilizando la técnica citada en el capitulo anterior.

Veamos la tabla.

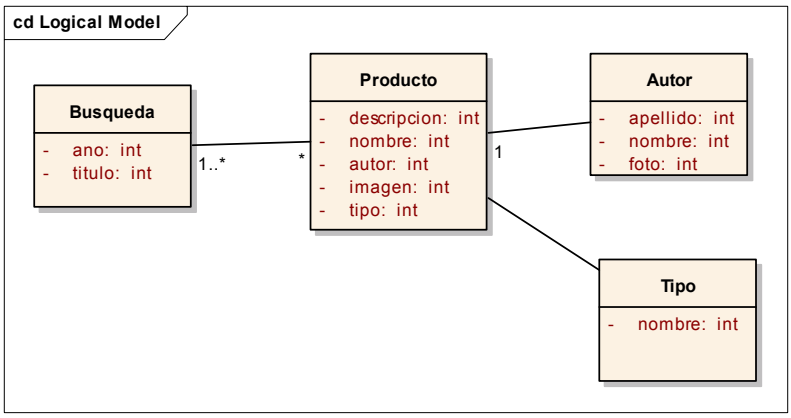
	Carrito	Autenticación	Producto	Comentario
Carrito		Asp	Nav	Asp/Nav*
Autenticación				
Producto				Asp/Nav
Comentario				

7.6 Derivación modelo

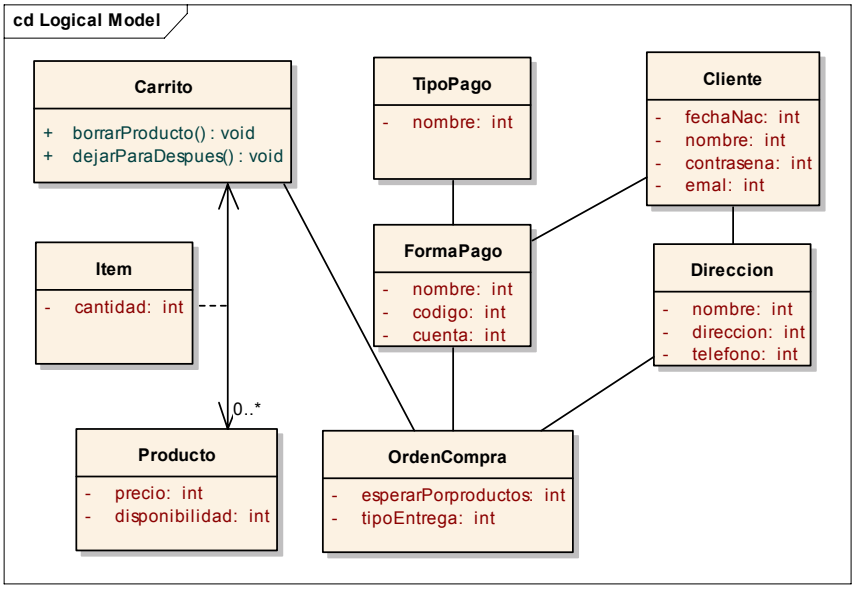
El resultado de este procedimiento es un modelo tentativo, que no necesariamente es correcto y mucho menos el producto final. El objetivo de este diagrama es dar una primera versión del modelo de clases (conceptual) libre de patrones donde s intentar revelar las entidades participantes.

A continuación mostraremos los modelos derivados a partir de UIDs utilizando las reglas definidas en (4.2.4).

1. Modelo derivado del concern producto. En este punto solo nos encontramos con un modelo elemental en donde solo se cubren requerimientos que giran alrededor de un Producto.



2. Aquí derivamos un modelo en el cual se revela un modelo de complejidad media donde participan las entidades necesarias para el concern de Venta.



Podemos notar que estas dos modelos representan dos islas sin relación entre ellas. En el capitulo siguiente se analizarán a estos y revelaremos relaciones no evidentes y de gran utilidad así como también su mejor implementación.

8 Análisis de diseño del caso de uso

Hasta este momento hemos determinado los requerimientos del futuro sistema, documentando aspectos y diseñando UID's. Además se espera que esta información ya se encuentre verificada por el cliente y confirmada para las etapas futuras del desarrollo de software.

Continuando con el caso de estudio realizado con nuestra herramienta, introduciremos heurísticas de análisis para ser aplicados en la etapa de diseño.

8.1 Análisis de Secuencias

Al momento de analizar las secuencias de un UID debemos estudiar su semántica, la intención subyacente de la secuencia.

Veamos el siguiente ejemplo:

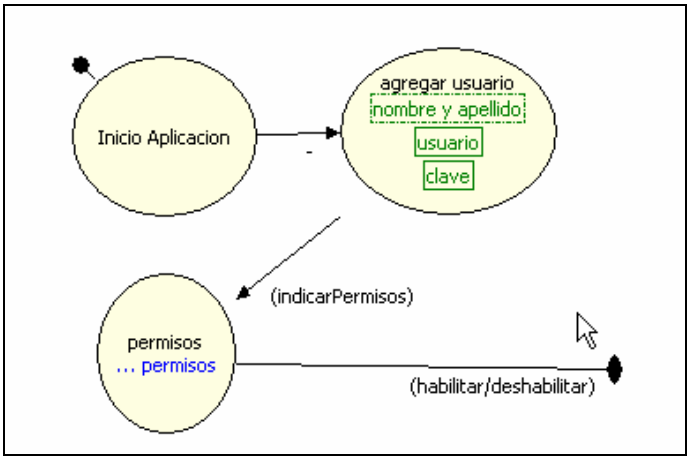


Ilustración 33: Estudio de un UID

En este ejemplo encontramos dos secuencias:

- la primera, desde inicio de aplicación a "agregar usuario"
- "indicarPermisos", de "agregar usuario" a "permisos".

Claramente, la primer secuencia no tiene sentido que sea mapeada a una asociación entre clases debido a que ésta es una facilidad de navegación.

Pero en cambio la segunda, sí corresponde, podría indicar una asociación entre las clases derivadas usuario y permiso (siendo estas clases obtenidas del diagrama).

A partir de este ejemplo podríamos caracterizar las secuencias de la siguiente forma:

- Sin semántica, por ejemplo también los links de navegación.
  - Solución: descartarla
- Con semántica de inspección simple, donde la inspección es sobre el objeto que representa la interacción fuente de la secuencia. Este caso representa asociaciones (colecciones), asociaciones especiales (algún filtro básico sobre alguna asociación), etc.

- Solución: referencia o método (que eventualmente utiliza una referencia).

- Con semántica de consulta compleja, no concluyen en relaciones porque son consultas sobre el sistema. Por ejemplo: "productos que fueron visitados por otros usuarios que han visitado el producto actual". Claramente es una operación que involucrará a varias entidades del sistema (suponiendo una buena encapsulación).
  - Solución: método.

Este análisis debe ser realizado en cada secuencia, inclusive en las que son generadas por una *regla de composición*.

Si señalamos el sentido de cada secuencia (de forma sencilla) en el momento del diseño de los UIDs, este análisis se vería simplificado. Pero como el elevamiento de requerimientos puede ser realizado por diferentes personas, se debe concensuar el medio para hacerlo.

8.2 Análisis de Reglas de composición

La composición de interacciones no necesariamente indica una relación entre objetos (entidades), por ejemplo composición con interacciones de "facilidades de navegación". Pero existen casos en lo que existe una relación y se necesitaría un pequeño análisis para encontrar la mejor interpretación. Veremos algunos casos a continuación donde además se propondrá una alternativa de diseño elegante.

Supongamos que tenemos el siguiente UID que representa un conjunto de requerimientos del concern de Producto:

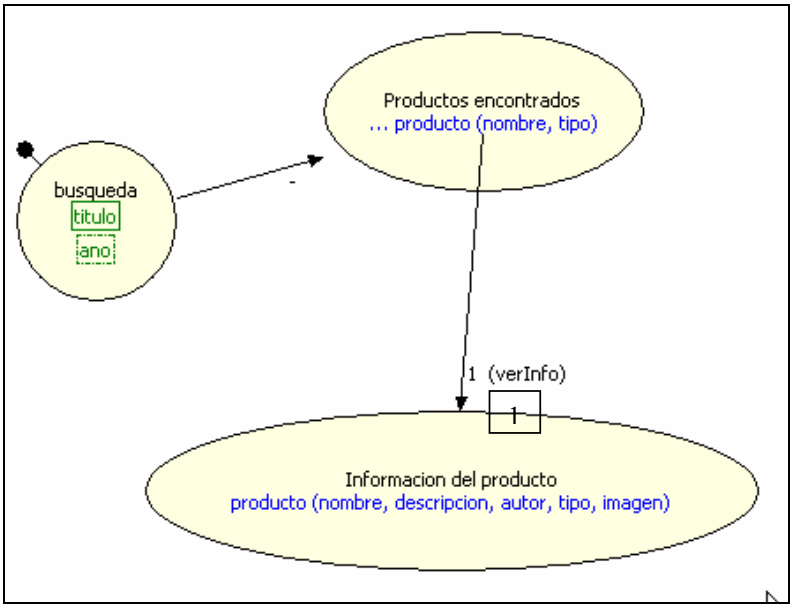


Ilustración 34 : Analizando una regla de composición

Y luego tenemos el siguiente UID que representa ciertos requerimientos del concern "Clientes que compraron, también compraron…":

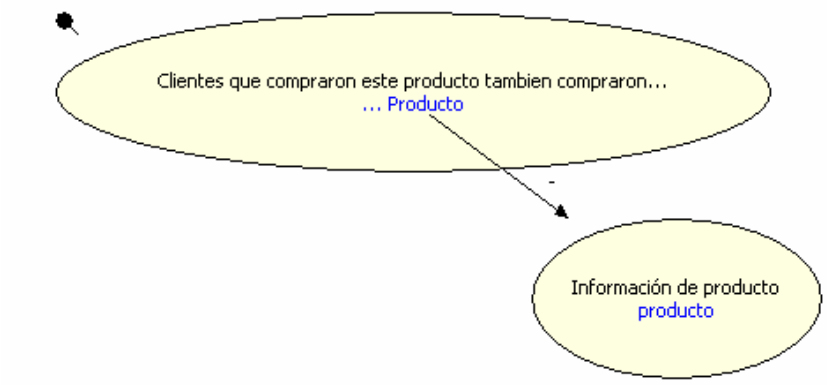


Ilustración 35: UID, clientes que compraron también compraron…

Si además tenemos la siguiente regla de composición:

```
Compose {
        Merge UID1.1 with UID2.1
}
```

En este caso, podríamos proponer una simple asociación pero no sería correcto si tomamos en cuenta la semántica de la composición. Estudiando los requerimientos relacionados al UID revelamos que el conjunto de elementos que se deben visualizar en las Recomendaciones ("Clientes que compraron, también compraron…") es un conjunto que debe satisfacer ciertas restricciones que están ligadas al producto pero no es parte del "producto" principal porque nace de otro conjunto de requerimientos (concern). Por lo anterior, podemos concluir que la mejor solución sería modelarlo como un nuevo método.

Es natural pensar que este comportamiento corresponda al objeto Producto, pero que pasaría si se siguen agregando requerimientos (concerns) que concluyen en el mismo tipo de solución. ¿Debemos agregar al nuevo método también a la clase? ¿Qué pasaría si éste nuevo comportamiento es temporal? ¿Debemos modificar el código existente que ya ha pasado las etapas de testing?

En el caso anterior la mejor solución sería modelarlo como un aspecto de la clase producto; permitiéndonos decidir que aspectos serán compuestos (weaving) según los requerimientos actuales del sistema. Cabe aclarar que este tipo de aspectos son claramente estructurales, es decir definen variables de instancia o métodos a la clase base.

Este es una buena solución para aquellos concern volátiles; estos son conjuntos de requerimiento que son temporales y caducan en un tiempo determinado.

Los concerns de Foro y de Comentario, son similares y por lo tanto podemos aplicar el mismo análisis.

Ahora que pasaría si tenemos diferentes vistas, obtenidas desde varios concern, del modelo como en Ilustración 36. En este caso podemos ver diferentes perspectivas del objeto Producto: producto, ***productoVendible*** y ***productoAdquirible***. Ante esta situación podemos proponer una solución como en el ejemplo anterior, pero no sería la mejor ya que diríamos que todos los productos son ***vendibles*** y ***adquiribles.*** Aunque estos roles pueden convivir juntos, existe una alternativa de mayor categoría: roles.

Utilizando roles ([26]), podemos dar comportamiento a cada objeto en particular sin tener generalizar los roles a todos los productos.

Muchas veces los roles son mas claros, por ejemplo cuando la información mostrada varía; por ejemplo a veces se muestra el promedio de calificación que recibió el producto por los usuarios o se muestran planes especiales de extensión de garantía.

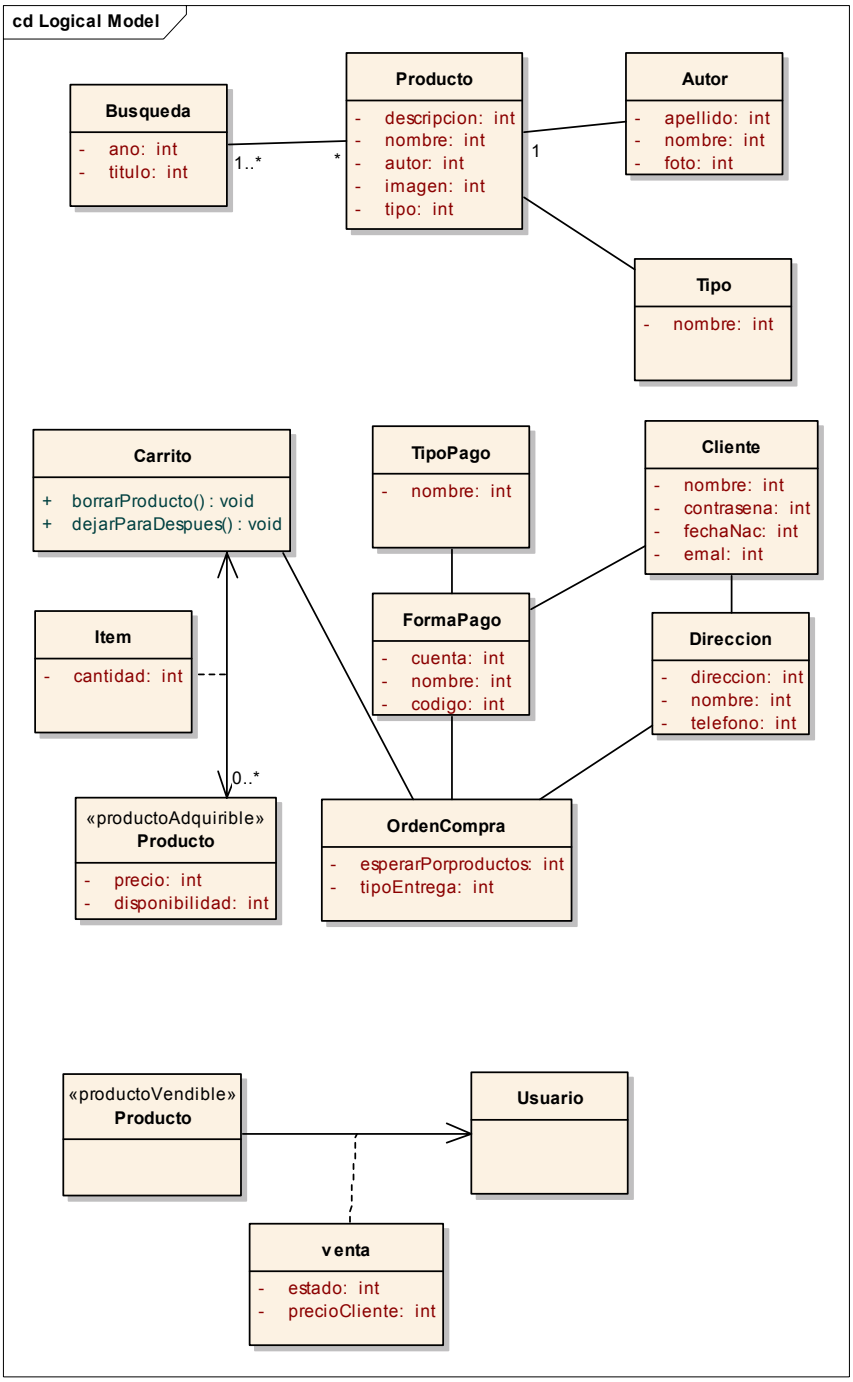


Ilustración 36: Partes del modelo

8.3 Conclusión del análisis

A continuación definiremos una tabla que enumera los concerns identificados previamente y plantea las posibles soluciones. La primer columna regencia al nombre del concern; la segunda columna da una breve descripción de la intención del concern; la tercer columna indica el impacto del concern en las diferentes capas; la cuarta columna sintetiza su mejor implementación; y por ultimo, la quinta columna describe los criterios que se tuvieron en cuenta para la solución.

Debemos destacar que en Ilustración 37: Tabla de análisis de crosscuttings. Que se comienza con dos concerns de diferente índole: el concern Producto e Historial. Estos dos concerns tienen impactos en diferentes modelos, el primer concern se ve reflejado en el modelo conceptual, y en cambio el segundo involucra al Modelo Navegacional.

Debemos destacar que el Concern de Autenticación, al igual que el concern Historial, solo involucra al Modelo Navegacional; pero además este concern implica comportamiento crosscutting sobre otros concerns. Parte de este comportamiento crosscutting esta altamente ligado con los contextos navegacionales (ver OOHDM [17]).

Posteriormente ofrecemos el diagrama de clases completo en el que ya se ha implementado las diferentes soluciones propuestas:

- Implementación de roles para los diferentes perfiles del Producto
- La inserción del nuevo comportamiento al Producto por consecuencia del cruce con:
  - Foro
  - Comentario
  - Tags
- Inyectado de comportamiento al Carrito de Compras por el cruce con el concern Checkout.

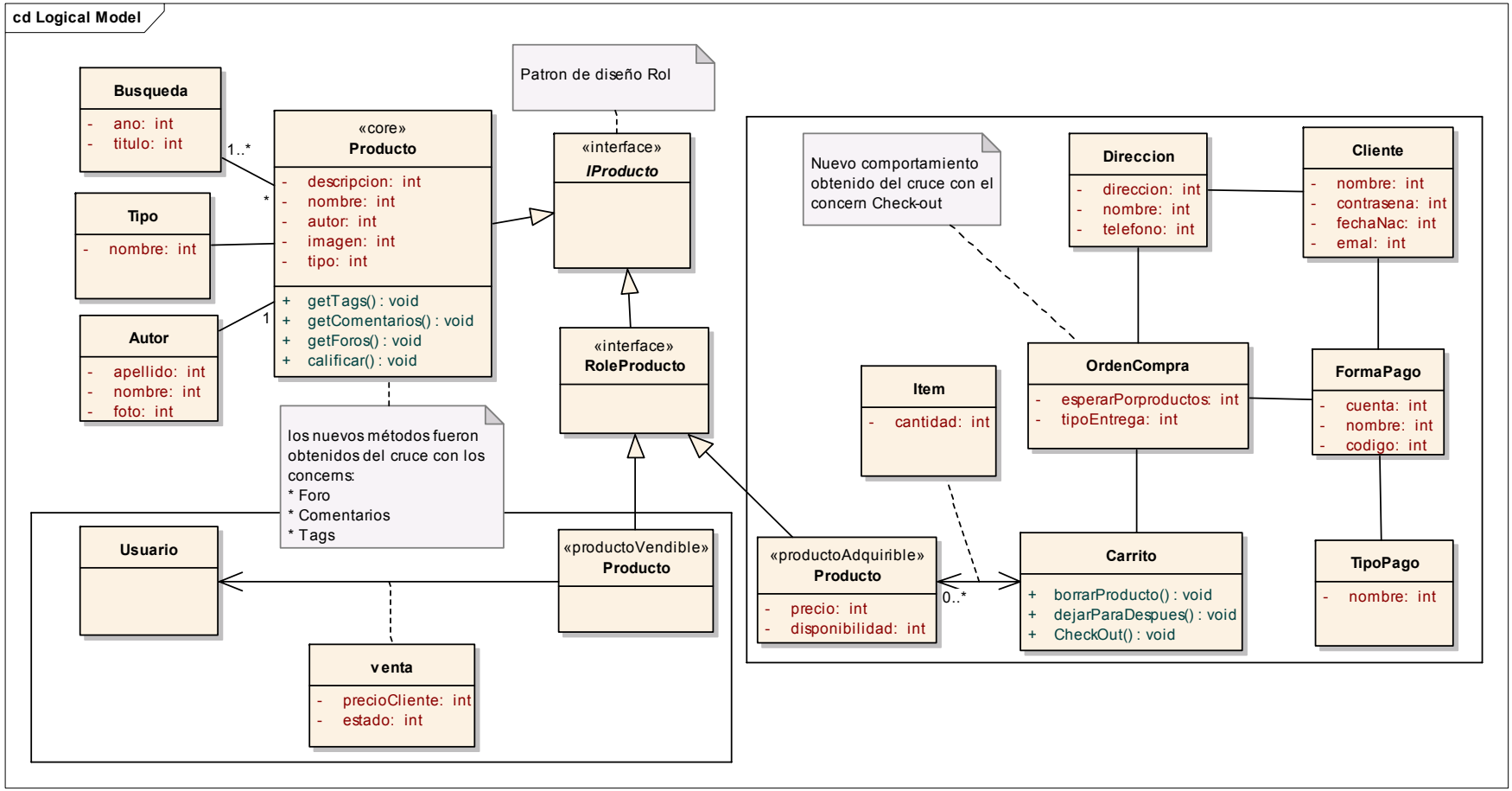
Ilustración 37: Tabla de análisis de crosscuttings.

Concern	Propósito	Capas involucradas	Solución	Comentario
Producto	Este concern muestra la información general del elemento.	Modelo conceptual	Modelo Core: captura la funcionalidad central del módulo.	Se debe destacar que además de este pequeño análisis, existe otro análisis en la búsqueda de patrones de diseño como por ejemplo TypeObject (para que diferente información varié dependiendo del tipo de producto, seria el caso de: un libro tiene autor y una notebook tiene un fabricante) o Roles (cuando un producto tiene elementos multimedia que se deseen mostrar).
Historial	Hacer el tracking de los productos visualizados.	Modelo navegacional	Joint point a nivel navegacional	
Autentificación	Requiere que para realizar ciertas operaciones en el sistema el usuario se encuentre logueado.	Modelo navegacional	Filtrado de eventos de la capa navegacional + contextos navegacionales	En este concern tenemos todos los requerimientos de autenticación. Se nombran todos eventos a ser capturados en los que se inicia el proceso de autenticación.
Foro	Representa las discusiones de los diferentes usuarios.	Modelo conceptual	Se crea un Modelo Conceptual sólo para los foros + se requieren Joint point en el modelo conceptual del core producto para adjuntar el nuevo comportamiento	Este concern tiene como parte de su definición: "El usuario debe haber comprado algún artículo para comenzar una discusión.", este requerimiento puede ser modelado tanto como parte de las reglas de negocios como un aspecto.
Comentarios	Permite agregar comentarios al producto	Modelo Conceptual	Modelo conceptual de Comentarios + se requieren Joint point en el modelo conceptual del core producto para adjuntar el nuevo comportamiento	Este concern tiene un crosscutting del Concern de autenticación: "el usuario debe estar logueado para poder agregar un comentario".

Concern	Propósito	Capas involucradas	Solución	Comentario
Tags	Permite agregar etiquetas al producto, para que sean más fáciles de recordar.	Modelo Conceptual	Modelo conceptual de Tag + se requieren Joint point en el modelo conceptual del core producto para adjuntar el nuevo comportamiento.	Este concern tiene un crosscutting del Concern de autenticación: "el usuario debe estar logueado para poder agregar un tag".
Listas (Casado, cumpleaños, etc.)	Nos deja llevar un listado de los productos que nos interesan.	Modelo Conceptual	Modelo conceptual de Listas + se requieren Joint point en el modelo conceptual del core producto para adjuntar el nuevo comportamiento.	Este concern tiene un crosscutting del Concern de autenticación: "el usuario debe estar logueado para poder agregar un producto a la lista".
Publicidad	Permite visualizar la publicidad del sitio.	Modelo navegacional	Composición (Merge) de vistas	La publicidad no es referente al producto seleccionado, en el caso de Amazon, por ello es navegacional. Si el nivel es mas complejo estilo Google se debe prestar más atención y representarlo en el modelo.
Recomendaciones	Constantemente Amazon adapta el sitio para alcanzar productos que se adapten al interés del usuario. También permite aconsejar productos según la relación que existe con el producto que se visita.	Modelo conceptual	Modelo conceptual de Clientes... + se requieren Joint point en el modelo conceptual del core producto para adjuntar el nuevo comportamiento.	Hace referencia a los requerimientos del tipo: **"Clientes que compraron este producto también compraron"**

Concern	Propósito	Capas involucradas	Solución	Comentario
Carrito de Compras	Este concern permite agregar cierta cantidad de productos al carrito de comparas.	Modelo conceptual	Se crea un modelo conceptual para soportar la compra de productos (toda la información subyacente) y agrega un Rol “vendible” al Producto.	Se propuso el uso de roles porque no todos los productos son “vendribles” (rol)
Venta del Producto	El usuario puede vender un producto determinado.	Modelo conceptual	Se crea un modelo conceptual para soportar la venta de productos (toda la información subyacente) y agrega un Rol “comprable” al Producto.	Se propuso el uso de roles porque no todos los productos son “comprables” (rol). Por ejemplo, un servicio de Internet no podría ser transferido de un usuario a otro o una entrada a un concierto puede no se revender una vez que ocurra el evento.
Check out	Es el proceso que permite representar las órdenes de compras.	Modelo conceptual	Modelo conceptual de Check Out + se requieren Joint point en el modelo conceptual del core carrito de compras para adjuntar el nuevo comportamiento. En el modelo navegacional , se agrega al nodo de Carrito de Compra un link que dispara el proceso de checkout.	Este concern tiene un crosscutting del Concern de autenticación: “el usuario debe estar logueado para hacer el check out”.

Ilustración 38: Modelo final



9 Trabajos futuros

En este capítulo, clasificaremos los trabajos futuros en las posibles ramas de investigación.

9.1 El futuro del enfoque

El futuro del enfoque básicamente consta de profundizar cada uno de los conceptos para obtener mejores resultados.

Completaremos el enfoque con un análisis más profundo en la clasificación de los requerimientos navegacionales y no navegacionales estudiando sus relaciones.

La idea general aun necesita ser pulida, además de ser aceptada por la comunidad.

El primer paso en la investigación debería ser adjuntar los conceptos necesarios para darle un nuevo perfil MDA a la técnica.

En paralelo, profundizar el análisis de requerimientos capturados en nuestro modelo antológico (documento de requerimientos) para obtener automáticamente modelos tentativos para la etapa de diseños.

A continuación se comentarán las alternativas posibles

9.1.1 Heurísticas de Derivación

El proceso de derivación, no es un proceso fácil. Existen varías ramas en la ingeniería de software que ya han estudiado este tema y que deberíamos adjuntar a este enfoque.

No solo deberíamos indagar en la derivación a partir de los UIDs sino que también a partir de los requerimientos puros. En la técnica [28] se hace un análisis léxico de los requerimientos para obtener información a partir de ciertas palabras claves. Aunque no es una tarea fácil, un camino interesante sería investiga y automatizar el análisis de la información de los requerimientos en este formato tan particular como lo es el documento de especificación. Esta variante requeriría investigación en análisis léxico (ya que sería necesario conocer la correctitud del texto), análisis de semántico (para interpretar y localizar acciones, entidades y relaciones), y por ultimo una investigación en el área de la lógica para poder inferir información que no se encuentra en la superficie. Dada la magnitud y variedad de áreas, esta empresa no es para nada sencilla.

9.1.2 Continuación del análisis de los diferentes crosscutting

Para esta rama, intentaremos realizar casos de estudios más sofisticados con casos reales en los que la aplicación está por ser desarrollada. Siguiendo todo su ciclo de vida de desarrollo y medir las contribuciones reales del enfoque.

En este trabajo de investigación, no se pudo verificar esto debido a la disponibilidad de recursos. Como nombramos anteriormente definimos nuestro caso de uso a partir de requerimientos obtenidos mediante la ingeniería inversa. Y tomamos como válidos las características de los UIDs ya que fue una técnica consensuada en varios congresos internacionales.

9.2 El futuro de la herramienta

9.2.1 Implementación del motor de derivación del modelo

Una vez obtenida la información del sistema, que en nuestro caso se manifiesta en UIDs. Se puede pensar en diseñar un sistema de plug-in para derivar código, donde cada plug-in sería capaz de interpretar la información del sistema y generar la estructura de archivos que corresponda.

Por ejemplo, podríamos tener los siguientes plug-in de inferencia de código:

Capa de Presentación:

En esta capa, las interacciones[12] son mapeadas directamente a nodos navegacionales. En tecnología Web es mas sencillo que en tecnología Desktop debido a que el mapeo resulta más fácil por su patrón request/response.

Por otro lado, en la tecnología desktop tradicional, el plug-in deberá construir las ventanas (como nodo navegacional) y linkearlas. Esto ya requiere generar más código estático que en los casos anteriores, es más pesado pero no imposible.

Algunos plug-in podrían derivar código: Struts, Tapestry, Swing, etc.

Capa de Negocio:

Hemos visto como generar un modelo de clases (objetos planos) a través UIDs. Además, se podrían construir EJB (Enterprise Java Beans) que den soporte distribuido y transaccional; o archivos de configuración de Spring.

Capa de datos:

Una vez obtenido el diagrama de clases para la capa de negocio podemos derivar archivos de mapeo para ORM como Hibernate y también los correspondientes esquemas de BD.

Siguiendo el hilo de la aplicación, la tecnología Eclipse brinda soporte para la manipulación de clases java, php y otros lenguajes; permitiendo inferir código de cualquier lenguaje.

El resultado de cada operación debería ser un prototipo que permita ver la evolución del análisis de requerimiento.

El problema fundamental de esta empresa es lo extensa que es con lo cual tiene que tener muy claro los objetivos.

Por ultimo, un aporte mucho más importante a lo anterior sería poder expresar crosscutting concerns navegacionales, para poder expresar tracking del usuario, publicidad personalizada, etc. Por ejemplo, esta información puede servir para derivar controladores de la capa Web especiales o aspectos.

Nuevamente son muchas ideas y hay que ver cuales valen la pena…

9.2.2 Implementación de motor de Vista previa con facilidades de edición

Actualmente la herramienta no brinda un soporte gráfico para indicar la forma de composición y distribución de las unidades navegacionales.

[12] Nos referimos a las interacciones de los UIDs

Por ende, un punto en el cual la herramienta debería indagar es en el prototipado navegacional y la distribución de los nodos navegacionales.

Esto, favorecerá el "*feedback*" del cliente al ingeniero en requerimiento por medio de la pre-visualización del producto final.

Está rama de investigación, podría orientarse a la identificación explicita de nodos navegacionales (al estilo OOHDM [19]) y a sus relaciones con los objetos del modelo conceptual. De esta forma focalizaríamos el diseño a una técnica top-down (de lo general a lo particular).

10 Conclusiones

La ingeniería de software es ha demostrado ser el eje de la calidad del software pues define normas para cada una de las etapas de desarrollo del software, permitiéndonos alcanzar software correcto, completo y de calidad.

A pesar de la importancia que tiene la Ingeniería de Requerimientos, ha costado mucho que se le preste la atención adecuada a esta actividad. Aún quedan muchos aspectos que deben ser mejorados, siendo un desafío para la ingeniería de software, tales como la integración de requerimientos funcionales y no funcionales, entre otros.

Cada actividad y técnica de la IR utilizada individualmente, dará diferentes soluciones para diferentes proyectos, incluyendo aquellos casos en los que el dominio y el área del problema son el mismo. Por esta razón, consideramos que no existe un modelo de proceso ideal para la IR; encontrar el método o la técnica perfecta es una ilusión, pues cada método y técnica ofrece diferentes soluciones ante un problema.

Es importante tomarse el tiempo necesario para conocer a nuestros clientes y usuarios, así como su ambiente de trabajo. Esto, también ayuda a establecer una buena relación de trabajo y comunicación entre el equipo de desarrollo y los clientes. Es realmente necesario que los clientes y usuarios participen en la definición de sus requerimientos, pues ellos son los que deciden el destino de nuestro proyecto, deciden si les gustamos o no y además financian el proyecto.

En cuanto a la investigación realizada de la técnica de Casos de Uso para la Ingeniería de Requerimientos, puede decirse que los casos de uso son independientes del método de diseño que se utilice, y por lo tanto, del método de programación. Luego de documentar los requerimientos de un sistema con casos de uso, se puede diseñar un sistema "estructurado" (manteniendo una separación entre datos y funciones), o un sistema Orientado a Objetos, sin que la técnica sea de mayor o menor utilidad en alguno de los dos casos. Esto da más flexibilidad al método, y probablemente contribuya a su éxito.

Asimismo, no basta con utilizar una sola herramienta/técnica para lograr nuestros fines, cada una de ellas aporta diferente y vital información sobre el problema en cuestión. Por ende nuestro aporte obtiene información adicional para las próximas etapas del desarrollo del Software. No sólo permite validar los requerimientos indagados con el cliente sino que además discrimina información vital para la etapa de diseño extendiendo a un nuevo nivel de modularización. Este nuevo grado es alcanzado por medio de los lazos lógicos definidos en las reglas de composición y la especificación de aspectos a través del caso de uso derivado de un UID. Como consecuencia obtenemos sistemas más fáciles de entender y mantener, lo cual conlleva a una evolución simplificada; cada nuevo módulo surgido de un conjunto de nuevos requerimiento no impacta (drásticamente) en nuestra aplicación. Como cada módulo es definido e implementado de forma modular e independiente y hay un bajo nivel de acople, no existe redundancia de código. Y además el módulo puede ser fácilmente reutilizado. Por otro lado, una fácil evolución nos facilita la salida a producción disminuyendo radicalmente la cantidad de horas de desarrollador y por ende los costos totales del sistema con la calidad de Software deseada.

Al enfoque tradicional de localización y aislamiento de concerns, ahora agregamos la localización de aspectos, requerimientos que cruzan otros requerimientos, que pueden tanto condicionar los requerimientos cruzados como también influenciarlos alterando su definición.

Claramente la dificultad más grande de la primera etapa del desarrollo de Software yace en el completo entendimiento del dominio para localizar este tipo particular de requerimientos.

Por estas razones creímos importante investigar este punto del proceso. Sin embargo, no negamos que es un área de investigación difícil porque está sometida al contacto con el cliente y todo lo que esto implica. El factor más importante es el aprendizaje del dominio de la aplicación. Pero existe el factor cultural; no siempre el desarrollo tiene alcance local o regional, muchas veces es internacional y sobre todo intercultural.

Creemos que un buen entendimiento del proceso de análisis de requerimientos es mucho más importante que las tecnologías subyacentes ya que ellas son volátiles y el análisis no, este trasciende en el tiempo. Y el proceso propuesto es fácil implantar en cualquier contexto.

11 Referencias

[1] Peri L. Tarr, Harold Ossher, William H. Harrison, Stanley M. Sutton Jr.: *N* Degrees of Separation: Multi-Dimensional Separation of Concerns. ICSE 1999: 107-119

[2] Aspect-Oriented Requirements Engineering, Isabel Brito

[3] Patricia Vilain, Daniel Schwabe, Clarisse Sieckenius de Souza: A Diagrammatic Tool for Representing User Interaction in UML. UML 2000: 133-147

[4] Ruzanna Chitchyan, Awais Rashid, Peter Sawyer: Comparing Requirement Engineering Approaches for Handling Crosscutting Concerns. WER 2005: 1-12

[5] Ingeniería del Software: Un enfoque práctico, Pressman MCGRAW-HILL / INTERAMERICANA DE ESPAÑA, S.A.

[6] Modelando Procesos de Negocio Web desde una Perspectiva Orientada a Aspectos; Roberto Rodríguez-Echeverría, Fernando Sánchez, José M. Conejero, Javier Pedrero

[7] http://help.eclipse.org

[8] AspectJ in Action, Practical Aspect-Oriented Programming; Manning, Ramnivas Laddad; ISBN: 1930110936

[9] Aspect-Oriented Software Development; Addison Wesley; Filman, Elrad, Clarke y Aksit; Septiembre 2004. ISBN: 0-321-21976-7

[10] Identifying and Composising Navegational Concerns in Web Applications, Silvia Gordillo y Gustavo Rossi

[11] www.eclipse.org/emf

[12] www.eclipse.org/gef

[13] Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework, IBM, ISBN 0738453161

[14] H. Lee; C. Lee and C. Yoo. A scenario-based object-oriented methodology for developing hypermedia information systems. En: Proceedings of 31st Annual Conference on Systems Science, Eds. Sprague R. 1998.

[15] O. De Troyer and C. Leune. WSDM: A user-centered design method for Web sites. En: Proceedings of the 7th International World Wide Web Conference. 1997.

[16] T. Isakowitz; E. Stohr and P. Balasubramanian. A methodology for the design of structured hypermedia applications. Communications of the ACM, 8(38), 34-44. 1995.

[17] G. Rossi; D. Schwave and Fernando Lyardet. Web application models are more than conceptual models. ER 1999: Paris, France - Workshops

[18] En: Proceedings of the First Internation Workshop on Conceptual Modeling and the WWW, Paris, France, November 1999.

[19] D. Schwave and G. Rossi. An Object Oriented Approach to Web-Based Application Desing. En: Theory and Practice of Object Systems (TAPOS), October 1998.

[20] Construyendo aplicaciones web con una metodología de diseño orientada a objetos; Darío Andrés Silva, Bárbara Mercerat

[21] Ruzanna Chitchyan, Awais Rashid, Peter Sawyer: Comparing Requirement Engineering Approaches for Handling Crosscutting Concerns. WER 2005: 1-12
[22] Early Aspects: The Current Landscape; Early Aspects and Domain Analysis.
[23] Reina Quintero, Antonia Ma., *Visión General de la Programación Orientada a Aspectos*, Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla, España, 2000.
[24] Crosscutting concerns in J2EE Aplications; Ali Mesbath, Arie van Deursen; Centrum voor Wiskunde en Informatica; P.O. Box 94079, 1090 GB Amsterdam, The Netherrlands.
[25] A Classification of Crosscutting Concerns; Maius Marin, Leon Moonen, Arie van Deursen
[26] The Role Object Pattern; Dirk Bäumer, Dirk Riehle, Wolf Siberski, and Martina Wolf; 1999; http://www.alejolp.com/blog/images/riehle.pdf
[27] Aspect-Oriented Software Development with Use Cases; Ivar Jacobson, Pan-Wei Ng; Addison Wesley Professional; ISBN 0-321-26888-1
[28] Aspect-Oriented Analysis and Design: The Theme Approach; Siobhán Clarke, Elisa Baniassad ;Addison Wesley Professional ; ISBN: 0-321-24674-8
[29] Barry Boehm, "Improving Software Productivity", IEEE Software, septiembre 1987
[30] Design Patterns: Elements of Reusable Object-Oriented Software; Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides; Addison-Wesley Professional Computing Series.

12 Apéndice A: Frameworks Utilizados

12.1 Introducción

El framework de edición grafica (GEF) provee un framework para crear editores gráficos. Estos editores no son fáciles usando los widget nativos de la plataforma Eclipse. En la mayoría de los casos, la gente utiliza sus propios modelos los cuales tienden a estar basados en Objetos Planos Java (Plain Old Java Objects, POJOs). Una alternativa a usar POJOs es la del Framework de Modelado de Eclipse (EMF), el cual provee algunas características para la manipulación de modelos que no se encuentran en los POJOs. A continuación se explicara brevemente cada framework.

Hay que comentar que estos frameworks generan código utilizando las mejores prácticas y patrones.

12.2 Eclipse

La plataforma eclipse define una arquitectura abierta de modo que cada equipo de desarrollo de componente pueda enfocarse en su área de especialidad.

La plataforma usa un modelo de área de trabajo común para integrar las herramientas desde el punto de vista de usuario final. Las herramientas que se desarrollan pueden integrarse al área de trabajo usando los ganchos llamados puntos de extensión.

La plataforma en si misma está construida sobre capas de componentes, cada uno acoplándose a los puntos de extensión de los componentes de menor nivel, y en su caso definiendo los puntos de extensión para futuras extensiones. Este modelo de extensión permite a los desarrolladores de componentes añadir una variedad de funciones a la plataforma básica. Los artefactos para cada herramienta, tal como archivos y otros datos, son coordinados por un modelo de recursos de plataforma común.

Cada subsistema en la plataforma está estructurado como un conjunto de componentes que implementan alguna función. Algunos componentes añaden características visibles a la plataforma usando el modelo de extensión. Otros proveen librerías de clases que pueden ser usadas para implementar extensiones del sistema.

El Eclipse SDK incluye la plataforma básica además de dos herramientas que son útiles para el desarrollo de componentes. Las Herramienta de Desarrollo Java (JDT) implementa un ambiente que brinda soporte a todas las características Java. El Ambiente de Desarrollo de Componentes (PDE) añade herramientas especializadas que optimizan el desarrollo de componentes y extensiones.

Estas herramientas no sólo son útiles por si mismo, sino que también proveen un gran ejemplo de cómo nuevas herramientas puedan ser añadidas a la plataforma para construir componentes que extiendan al sistema.

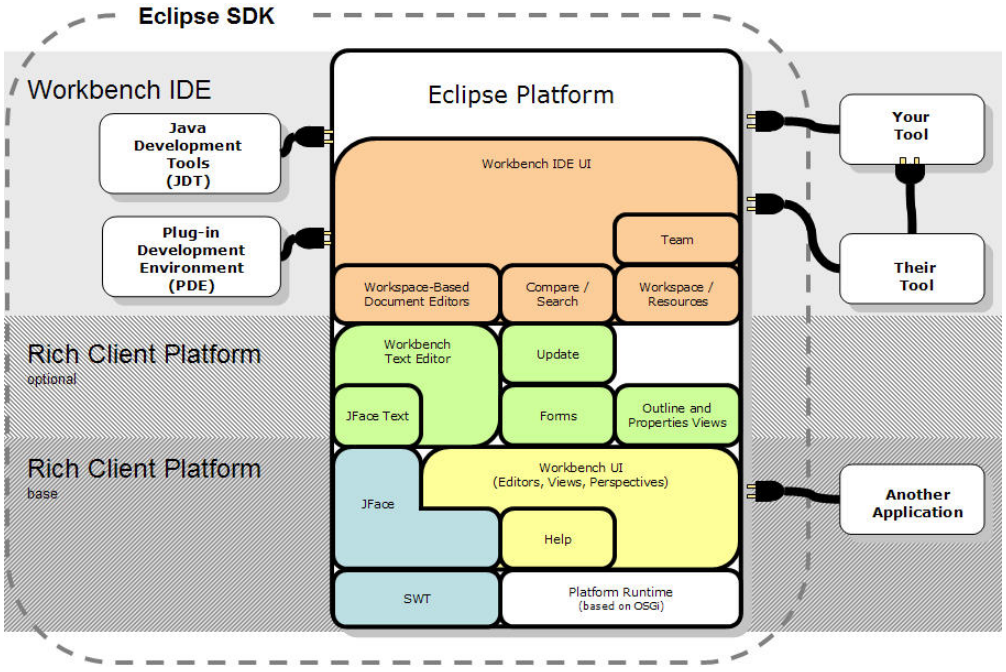


Ilustración 39: Arquitectura Eclipse

12.3 Eclipse Modeling Framework (EMF)

EMF es una herramienta de modelado y de generación de código para construir herramientas y otras aplicaciones basadas en un modelo de datos estructurado. Este framework esta diseñado para facilitar el diseño e implementación de un modelo. El framework Java (refiriéndose a JDT) provee la infraestructura para la generación de código permitiendo de esta forma enfocarse en el modelo y no en detalles de implementación.

El EMF es parte de la Arquitectura Orientada por el Modelo (Model Driven Architecture, MDA). Este es la implementación actual de una porción de la MDA en la familia de herramienta Eclipse.

La idea detrás de MDA es permitir desarrollo y administración de la totalidad del ciclo de vida de una aplicación poniendo el foco en el modelo.

Desde una especificación de modelo descripta en XMI. EMF provee herramientas y soporte para producir un conjunto de clases java para el modelo, un conjunto de clases "adapter"[13] que permiten la presentación y edición basada en comandos[14] y un editor básico. Los Modelos pueden ser especificados usando Anotaciones Java, documentos XML, o con herramientas de modelaje como Racional Rose, luego esta información es incorporada EMF.

EMF consiste en tres piezas fundamentalmente:

- ***EMF*** – El núcleo del framework incluye un metamodelo (ECore) para describir modelos y suporte para el modelo incluyendo: notificación de cambio, soporte

[13] Se refiere al patrón de diseño "*Adapter*",ver GOF
[14] Se refiere al patrón de diseño "*Commnad*", ver GOF

de persistencia con una serialización por defecto XMI, y una API reflexiva muy eficiente para manipular objetos genéricamente,

- ***EMF.Edit*** – este framework incluye clases genéricas reusables para la construcción de editores de modelos EMF. Se provee de:
    1. Clases que facilitan la presentación de modelos EMF usando JFace.
    2. Un framework de comandos, incluyendo un conjunto genérico de clases comando implementadas para la construcción de editores que soporten completamente automáticos ***deshacer*** (undo) y ***rehacer*** (redo), etc.

- ***EMF.Codgen*** – El módulo de generación de código es capz de generar todo lo necesario para construir un editor para un modelo EMF. Este incluye una interfaz gráfica (GUI) desde la cual se pueden configurar e invocar a los generadores.

Los niveles de generación de código soportados son:

1. **Modelo**: provee interfaces Java e implementaciones para todas las clases del modelos, además un Factory y otros.
2. **Adaptadores**: genera clases (llamadas ItemProviders) que adaptan las clases del modelo para su edición y presentación.
3. **Editor**: produce un editor propiamente estructurado que ajusta al estilo de los editores de modelo Eclipse EMF y sirve como un punto de partida desde el cual comenzar la especialización del editor.

Todos los generadores soportan regeneración de código preservando las modificaciones del usuario- Los generadores pueden ser invocados por medio de la GUI o desde la lía de comando.

12.3.1 Graphical Editing Framework (GEF)

El framework de edición gráfico (GEF) permite a los desarrolladores crear editores gráficos para un modelo un modelo de aplicación existente.

Con estos editores es posible realizar modificaciones simples de nuestro modelo, como cambiar propiedades de elementos u operaciones complejas como cambiar la estructura de nuestro modelo.

Toda la visualización es realizada por el framework Draw2D, el cual es un estándar de dibujo 2D basado en SWT de eclipse.org.

El componente *org.eclipse.draw2d* provee los módulos necesarios para la presentación de gráficos. Los desarrolladores pueden entonces tomar ventaja de algunas operaciones comunes que son provistas por GEF y/o extenderlos para un dominio específico.

GEF emplea una arquitectura MVC (model-view-controller) la cual permite cambios simples desde la vista al modelo, como un flujo de eventos ordenados.

GEF es una aplicación neutral y provee todas las herramientas para construir casi cualquier aplicación, algunos ejemplos son: diagramas de actividad, constructores GUI, editores de diagramas de clases, maquinas de estado, e incluso editores de texto WYSIWYG.

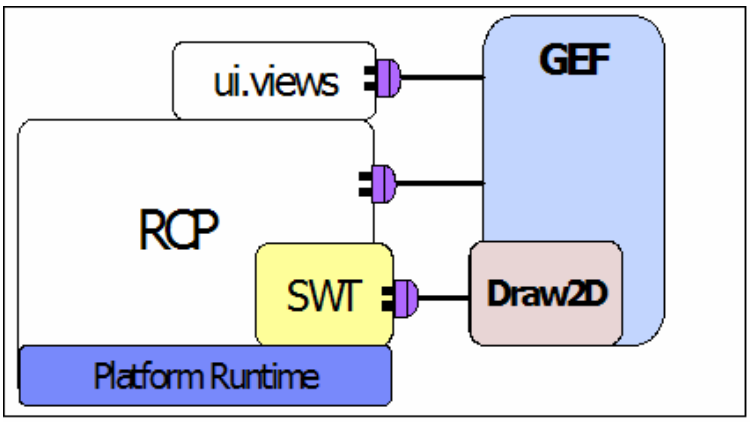


Ilustración 40: Arquitectura

El componente *draw2d* se enfoca en dibujar y localizar figuras. El componente GEF se añade sobre el componente *draw2d.* El propósito de este framework es:

1. Facilitar la presentación de cualquier modelo utilizando las figuras *draw2d.*
2. Soportar interacción desde el Mouse, Teclado y área de trabajo Eclipse.
3. Proveer componentes comunes relacionados a los puntos anteriores.

El framework provee un vínculo entre un modelo de aplicación y la vista. Este también brinda manejadores de entradas, como herramientas y acciones, que transforman eventos en requerimientos. Los Requerimientos y Comandos son usados para encapsular interacciones y sus efectos en el modelo.

En el diseño MVC, el controlador es a menudo la única conexión entre la vista y el modelo. El controlador es responsable de mantener la vista, e interpretar los eventos de la interfaz del usuario y transformarlos en operaciones sobre el modelo.

Los Roles son:

Modelo

El modelo es cualquier información que es persistida. Cualquier modelo puede ser usado con GEF. El modelo debe tener tipo de mecanismo de notificación.

Un comando representa algún tipo de modificación sobre el modelo de manera que pueda ser deshecha y rehecha por el usuario. En general, los modelos solo deberían funcionar sobre el modelo en si mismo.

Vista (Figures/Treeitems)

Es la parte visual que representa los elementos del modelo. La vista es cualquier cosa vista por el usuario.

Controlador (EditPart)

Editpart son los elementos centrales en las aplicaciones GEF. Ellos son los controladores que especifican como los elementos del modelo son apeados a las figuras y como estas figuras se comportan en diferentes situaciones.

Hay usualmente un controlador para cada objeto del modelo visualizado por lo tanto tendríamos casi la misma jerarquía de clases para los *EditPart*. Los EditPart son el vínculo entre el modelo y la vista.

Las *EditPolicies* son las partes del GEF que incorporan la funcionalidad de edición dentro de los *EditPart*.

Una *EditPolicies* define que puede ser hecho con una *EditPart*. Un *editPart* sin *EditPolicies* no va a hacer nada; ni siquiera van a ser seleccionable.

Las *EditPolicies* están categorizadas en roles y las *EditPart* están limitadas a tener solo una *EditPolicy* por rol.

Visualizadores

Un *EditPartView* es donde los *editpart* presentan sus vistas. Hay dos tipos de visualizadores en GEF. Un visualizador gráfico hospeda figuras mientras que un visualizador de árbol presenta elementos de árbol nativos (*TreeItem*).

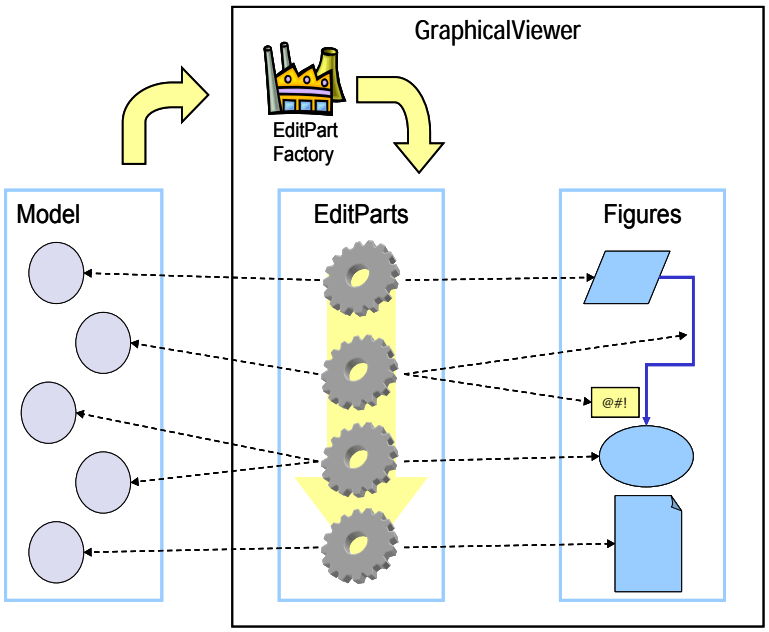


Ilustración 41: Interacción de componentes

12.3.2 Rich Client Platform (RCP)

Eclipse Rich Client Platform es un framework para la creación de clientes ricos, utilizando para ello una base genérica extraída del entorno de desarrollo Open Source Eclipse. Esta base genérica, ofrece una gran cantidad de servicios que ahorran gran cantidad de trabajo al desarrollador, y que por consiguiente permiten un desarrollo más rápido de aplicaciones.

Eclipse Rich Client Platform (RCP) es un framework genérico, basado en el entorno de desarrollo Eclipse, que permite el desarrollo rápido de aplicaciones basadas

en SWT y JFace. Este framework genérico consta de un conjunto básico de plugins que conformarían el núcleo genérico de la plataforma Eclipse.

La Plataforma de Clientes Ricos (RCP) de Eclipse permite construir muy rápidamente workbenches (blancos de trabajo) de Eclipse y compilarlos en aplicaciones que pueden ejecutarse sin Eclipse. Un cliente rico es aquel cliente que aprovecha las posibilidades del sistema operativo donde se ejecuta. El programador puede usar RCP para construir el armazón grafico de sus propias aplicaciones con la ventaja añadida de disponer de una interfase grafica.

La principal ventaja que ofrece es el ahorro de trabajo para el desarrollador., pues existe un conjunto de funciones ya desarrolladas, por lo tanto, el desarrollador se centra en la lógica de la aplicación.

Eclipse RCP es un framework de aplicaciones gráficas del tipo Cliente Rico y básicamente aporta dos grandes ventajas. Por un lado lógica embebida referente a funcionalidad de interfaces gráficas y capacidad de configuración. Y por otro, extensión dinámica e incremental de la funcionalidad de la aplicación.

El primer aspecto permite un desarrollo ágil de interfaces, ya que al disponer de mucha funcionalidad preprogramada al respecto, sólo se requieren algunas configuraciones y extensiones para adaptar esta funcionalidad genérica a las necesidades concretas de la aplicación. Esto posibilita al desarrollador concentrarse casi exclusivamente en los aspectos de interfaz inherentes al negocio, reduciendo o anulando el tiempo invertido en la implementación de detalles técnicos de bajo nivel. Por ejemplo, Eclipse RCP provee funcionalidad de perspectivas y wizards, entre otras muchas cosas, que puede incluirse en nuestra aplicación adaptándola mediante una simple configuración.

El segundo aspecto constituye la solución al problema planteado anteriormente, ya que permite el desarrollo y la separación de la funcionalidad total en pequeños módulos que pueden combinarse de acuerdo a las necesidades particulares. Esto permite poner en producción rápidamente un cliente a medida sólo mediante una combinación de tales módulos.

Técnicamente, cada uno de estos módulos se implementa en plug-ins. Por otro lado, se desarrolla una aplicación base (workbench) que contiene la funcionalidad elemental de todos los clientes ricos. De esta manera, la puesta en producción de un cliente a medida consiste en montar sobre el workbench los plug-ins que contienen la funcionalidad deseada para el mismo.

La extensión con nueva funcionalidad de un cliente en producción también resulta flexible y totalmente dinámica: un administrador del cliente selecciona, desde la misma aplicación, los nuevos plug-ins que desea instalar. El framework de Eclipse RCP se encargará de descargarlos desde un servidor remoto disponible a tal fin e instalarlos de manera automática, requiriendo ninguna o una mínima intervención del usuario.

Para entender el concepto de RCP sólo alcanza con ver el entorno de desarrollo Eclipse, el cual es en si una aplicación RCP. Para agregar funcionalidad adicional sólo se debe crear un nuevo plug in, definir que punto de extensión de la aplicación queremos extender e implementar las clases requeridas por el framework. Además, también se pueden definir nuevos punto de extensión para permitir que otros programadores agreguen funcionalidad a nuestros desarrollos.