UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA

FLUJO DE TRABAJO DEL ANÁLISIS Y DISEÑO DEL RUP

M O N O G R A F Í A

QUE PARA OBTENER EL TITULO DE LICENCIADO EN SISTEMAS COMPUTACIONALES

P R E S E N T A

SERGIO DEMIÁN GRACIÁN HERNÁNDEZ

ASESOR: LIC. LUIS ISLAS HERNÁNDEZ

MAYO/2006

Agradecimientos

Aunque en ocasiones me hizo un poco de gracia la parte de los agradecimientos en los libros, tesis o documentos, ahora, al redactar esta parte debo de reconocer que es muy importante dar honra al que la merece cuando se trata de las personas que nos ayudan a conseguir nuestras metas, es por esto que:

Agradezco sinceramente.

A Dios
Por darme los recursos suficientes para aprender.

A mis padres y hermanos
Por su cuidado, preocupaciones, desvelos y enseñanzas que han aportado para formar lo que soy, para bien o para mal.

A mis maestros
Por sus enseñanzas, vivencias y consejos, en especial a mi asesor Lic. Luis Islas Hernández por su apoyo para la realización de esta monografía, al Ing. Enrique Carrasco López y no por ser menos importante al Ing. Ciro Velásquez Jaén.

Índice

Introducción

La manera de desarrollar software ha cambiado sin duda desde el inicio de la era de la computación hasta estos días. Sin embargo, en México, la mayoría de las empresas sigue tomando un enfoque artesanal, es decir, los procesos se siguen haciendo manualmente, empezando con la captura de requerimientos hasta la implementación, y no se diga del mantenimiento del sistema.

En mayo del 2004 se realizó una auditoria de desarrollo de software al área de modernización de una empresa, y al preguntarle al director sobre cuál era el proceso que utilizaban para el desarrollo de sus sistemas, la respuesta fue que seguían usando el enfoque tradicional en cascada porque era el que venían usando desde siempre y tenían poco tiempo de planeación debido a lo apretado de su agenda de trabajo.

La respuesta del director puede ser válida, después de todo, porqué cambiar a un proceso como el RUP si lo que se tiene funciona. Sin embargo, en un mundo globalizado y tan competido, es una respuesta incorrecta, cada día el software se debe de entregar en menor tiempo y con más calidad, además, se debe aprender de los errores, el proceso debe ser bien planeado y sobre todo debe ser lo más automatizado como sea posible.

El RUP es una respuesta a las demandas de desarrollo de los sistemas actuales y al ser el proceso propietario de una compañía, es un producto que se está actualizando constantemente, además de proveer las herramientas necesarias que lo soporta.

El RUP es el resultado de la unificación de técnicas, conceptos y herramientas que convierten a éste en un proceso ideal para muchos tipos de proyectos, respondiendo a las demandas actuales de desarrollo de software.

Aun no se ha logrado hacer del todo un proceso automatizado, y tal vez nunca se hará, ya que la tarea del desarrollo de un sistema es igual a la tarea de resolver un problema, y hasta ahora esta tarea solo la hacen los humanos. Sin embargo lo que se puede hacer es automatizar las tareas monótonas, seguir una serie de pasos para resolver el problema (proceso) y sobre todo aprender de los errores.

Esta monografía pretende abarcar la disciplina de análisis y diseño, que es una de las nueve que tiene el RUP.

El **primer capítulo** da un resumen de la evolución, conceptos, filosofía y disciplinas del RUP, los cuales son necesarios para entender todo el proceso y posteriormente la disciplina de análisis y diseño.

En el **segundo capítulo** se explica el primer flujo de trabajo que realiza un bosquejo de la arquitectura, a fin de que se pueda hacer un estudio de la factibilidad del desarrollo del sistema.

El **tercer capítulo** explica el segundo flujo de trabajo en el que se diseña una arquitectura candidata que resuelva la funcionalidad principal del sistema.

En el **cuarto capítulo** se abordan los pasos necesarios para hacer un modelo de análisis, el cual abstrae el comportamiento del sistema de manera conceptual, que es el objetivo del tercer flujo de trabajo de la disciplina de análisis y diseño. Hasta aquí acaba de alguna manera el análisis del sistema, ya que es muy difusa la línea entre el diseño y el análisis.

En el **quinto capítulo** se tratan aspectos sobre cómo se puede refinar la arquitectura que se ha desarrollado.

Ya casi para finalizar, en el **sexto capítulo** se diseñan los componentes que se implementarán en disciplinas posteriores.

El **ultimo capítulo** trata sobre el diseño de la Base de Datos, un elemento indispensable en los sistemas de hoy en día.

Justificación

Cada día se va incrementando la complejidad del software y las partes dentro de la organización que se automatizan por medio de él. Debido a lo anterior, se necesita que los procesos de desarrollo de software produzcan sistemas de calidad, que se entregue lo más rápido que se pueda, y además que los pasos sean predecibles y administrables.

Ya que los procesos que se han usado desde siempre no cumplen con las características antes mencionadas, el RUP es una opción para hacer frente a las necesidades de los sistemas de hoy en día. Por lo tanto se precisa que los profesionales en las Tecnologías de la Información conozcan el RUP, para desarrollar software de calidad.

Al ser un proceso muy completo y complejo, resulta difícil analizar y exponer a detalle todas las partes que el tiene RUP.

Debido a todo lo anterior, la presente monografía se justifica en la necesidad de un conocimiento robusto del RUP, y da un panorama completo del flujo de trabajo de Análisis y Diseño que es uno de los nueve que componen este proceso. El Análisis y Diseño es uno de los flujos de trabajo más importantes, ya que es en donde se modela el sistema y donde se define la mayor parte de la arquitectura.

Objetivo General

- Documentar y explicar el flujo de trabajo del “Análisis y Diseño” del RUP con sus actividades, sus artefactos y los roles más importantes que guían ésta parte del proceso.

Objetivos Específicos

- Explicar de manera general los conceptos y la definición del Proceso Unificado de Rational (RUP)
- Documentar el flujo de trabajo del Análisis y Diseño que tiene el RUP.
- Documentar y explicar el propósito y la función de las actividades más importantes que se realizan en el Análisis y Diseño.
- Documentar y explicar el propósito y la función de los artefactos más importantes que se crean, modifican y utilizan en el Análisis y Diseño.
- Documentar y explicar el propósito y la función de los roles más importantes que se desenvuelven en el Análisis y Diseño.

Capítulo 1

Generalidades del RUP

"Los puentes romanos de la antigüedad eran estructuras muy ineficientes.
Para los estándares modernos, usaban muchísima piedra, y como resultado, necesitaban mucha labor para construirse."

Tom Clancy - The Sum of All Fears

Introducción

En este capítulo se da una explicación a grandes rasgos del Proceso Unificado de Rational (RUP), lo anterior con la finalidad de presentar una vista general antes de entrar al detalle del flujo de trabajo del análisis y diseño.

"Cada día el software entra en procesos más críticos dentro de las organizaciones, además, se requiere que este sea liberado mas rápidamente y con calidad" [20]. Sin embargo, lo anterior no se puede hacer con los procesos de desarrollo de software que se han venido utilizando desde hace mucho tiempo, se requiere de un proceso de desarrollo de software que pueda administrar lo requerimientos cambiantes, también se requiere de un proceso, que sea fácil de escalar o actualizar y que ataque los riesgos más críticos al inicio del proyecto ya que " el descubrimiento de defectos de diseño en fases tardías redundará en costos mayores a los inicialmente estimados y, el tiempo y dinero gastados en implementar un diseño incompleto o poco flexible NO son recuperables" [20], también se requiere de un proceso que entregue resultados lo más rápidamente posible y no tan solamente documentos, ya que el usuario tiende a desesperarse, se necesita que el proceso de desarrollo de software tenga al usuario como un componente importante porque a final de cuentas es el que va a utilizar el producto.

El RUP recolecta las mejores prácticas de ingeniería de software y las implementa con el fin de hacer un ciclo de vida eficaz, para que al final se entregue al cliente un producto de calidad.

1.1 Definición de un proceso de desarrollo de software

Un proceso en forma general es un conjunto de actividades ordenadas en las que se define:

- Responsabilidad de QUÍEN realizará las actividades.
- TIEMPOS en los que se realizarán las actividades.
- FORMAS de realizar las actividades.

Con el fin de alcanzar un determinado objetivo, los procesos de desarrollo además deben de ayudar a la administración del proyecto, obtención de recursos, medición del progreso y reducción de riesgos.

En el caso del RUP, el objetivo será el de desarrollar software que sea predecible, repetible y medible.

1.2 Evolución del RUP

El RUP es el resultado de más de 3 décadas de **conocimiento unificado**, la figura 1.1 muestra que su desarrollo ha recibido influencia de muchas fuentes, solo se citarán las más importantes.

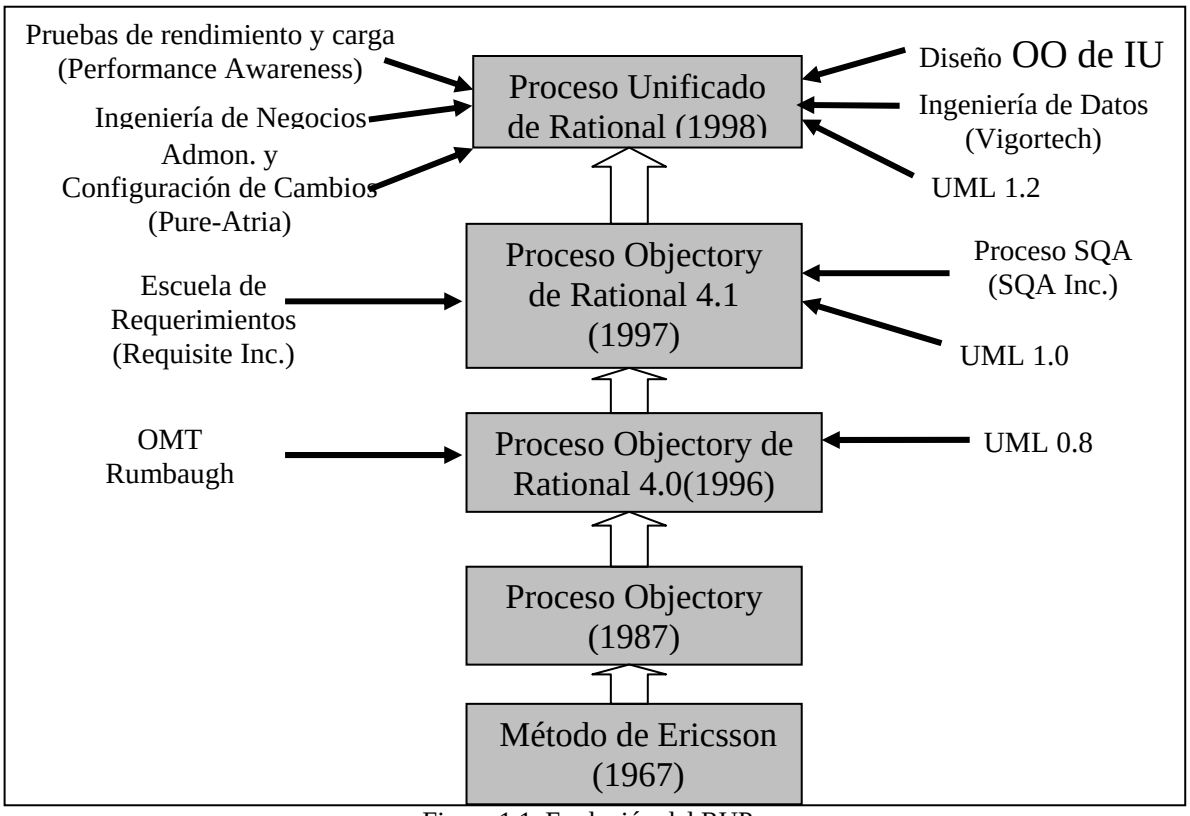

Figura 1.1. Evolución del RUP.

1.2.1 Método de Ericsson

El método de Ericsson marca el inicio de la evolución hacia el Proceso Unificado de Rational. “Ericsson modelaba el sistema entero como un conjunto de bloques interconectados” [13], es decir dividía la complejidad del sistema en varias partes para hacer el problema más manejable, en el método de Ericsson “los requerimientos impuestos por el problema… se agrupaban, clasificaban, y organizaban en problemas de menor complejidad que se resolvían concurrentemente” [25]

Los bloques se identificaban mediante un estudio de casos de negocio, lo que hoy se llaman casos de uso, que se realizaban previamente. La actividad de diseño producía un

conjunto de bloques que se comunicaban por medio de interfaces, estas se mandaban señales entre si, algo parecido a lo que hoy se conoce como diagrama de clases. “El método de diseñar a partir de bloques con interfaces bien definidas, fue la clave del éxito” [13]

1.2.2 Proceso Objectory

El proceso Objectory fue desarrollado entre 1987 y 1995 por la compañía Objectory AB, fundada por Ivar Jacobson, y aunque en el método de Ericsson estaban implícitamente los casos de uso, el proceso Objectory le dio su nombre formalmente, e hizo más clara la idea de un proceso guiado por casos de uso, es decir estos servían a los desarrolladores para realizar la arquitectura del sistema y a los usuarios como una comprensión del sistema en sus propias palabras.

“El desarrollo del proceso Objectory continuó en una serie de versiones, desde Objectory 1.0 hasta Objectory 3.8, en 1995.” [13]

En ésta época de evolución también se sentaron las bases para modelar procesos de negocios y no solo de software.

1.2.3 Proceso Objectory de Rational

Rational Software compro a Objectory AB en 1995, y aunque “Objectory 3.8 había demostrado que se puede crear y modelar un proceso de desarrollo de software como si fuera un producto” [13], todavía le faltaba desarrollar algunas partes del proceso tales como pruebas, administración de la configuración y aspectos sobre la calidad del software.

Es por eso que en la siguiente versión, ya como producto de Rational (Rational Objectory Process 4.1) se añadió de lleno el concepto de iteraciones y fases, además se incorporo UML. La figura 1.2 muestra el proceso Objectory.

Ésta etapa es conocida por prestarle la atención que debe a la **arquitectura**[1] como la parte de más peso en la organización del sistema.

[1] La arquitectura se define como la estructura de los componentes mas importantes de un sistema, en la sección 1.4.2 se muestra una discusión completa sobre éste termino

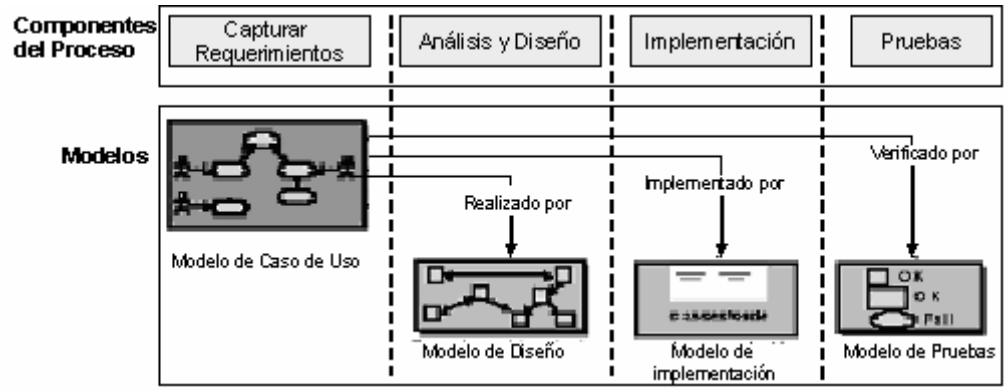

Figura 1.2. Relación de procesos y modelos del Proceso Objectory. Fuente. The Rational Objectory Process – A UML-based Software Engineering Process [14]

1.2.4 Proceso Unificado de Rational

La evolución siguió con la fusión de Rational Software con otras compañías que tenían experiencia en las áreas que aun no estaban del todo desarrolladas del proceso. Los siguientes puntos muestran los aportes de cada una de ellas al RUP.

- Requisite: Gestión de requisitos.
- SQA: Procedimientos de pruebas.
- Pure-Atria: Gestión de la configuración.
- Performance Awareness: Pruebas de rendimiento y de carga.
- Vigortech: Ingeniería de datos.

Además, se fueron incorporando las nuevas versiones de UML, y se agregó el flujo de modelado de negocio que se encarga de entender la organización en la que trabaja el sistema. “El cambio del nombre (de Objectory a RUP) refleja el hecho de que la unificación ha tenido lugar en muchas dimensiones: unificación de técnicas de desarrollo a través del Lenguaje Unificado de Modelado y unificación del trabajo de muchos metodologistas” [13]

1.3 Definición del RUP

El RUP es: “un proceso de negocios genérico para la ingeniería de software orientada a objetos. Describe una familia de procesos de ingeniería de software relacionados, que comparten una estructura y arquitectura común de procesos. El RUP provee un método disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su meta es asegurar la producción de software de alta calidad, que conozca las necesidades de los usuarios, dentro de un calendario y presupuesto predecible.” [22]

1.4 Características del RUP

Las características principales del RUP son la de ser un proceso iterativo e incremental que es guiado por casos de uso, se centra en la arquitectura e implementa las mejores practicas de ingeniería de software, en cuanto a esta ultima característica cabe señalar el papel importante que juega el Lenguaje Unificado de Modelado o UML como un agente crítico, ya que no solo es un conjunto de diagramas, sino un "lenguaje para capturar conocimiento (semántica) acerca de una materia (en este caso un sistema) y que además tiene la capacidad de comunicarlo a quien conozca su sintaxis" [25].

Así que UML complementa al RUP ya que: UML aporta el modelado y el transporte del conocimiento a los distintos equipos de trabajo y el RUP el proceso o la logística para el desarrollo, entre los modelos mas importantes que aporta el UML están: el modelo de caso de uso, el modelo de análisis, el modelo de diseño y el modelo de distribución.

1.4.1 Un proceso dirigido por casos de uso

La razón de ser de un sistema de información es en primer lugar satisfacer las necesidades de los usuarios[2], en el RUP se les llaman requerimientos de software.

El RUP define requerimiento como "Una condición o capacidad que el sistema debe conformar" [22]; UML lo define como "comportamiento, propiedad o característica deseada de un sistema" [17]

Los requerimientos se pueden categorizar de la siguiente manera:

- **Requerimientos funcionales:** Especifican acciones que un sistema debe ser capaz de realizar, sin considerar restricciones físicas (estéticas, documentación, tolerancia a fallos, velocidad, etc.)
- **Requerimientos no funcionales:** Son aquellos requerimientos que no entran en la categoría de requerimientos funcionales. La mayoría de los requerimientos que están en este tipo tienen que ver con: Usabilidad, Fiabilidad, Desempeño y Soportabilidad (escalabilidad, mantenibilidad, configurabilidad e instalabilidad).

Los casos de uso especifican los requerimientos funcionales y los no-funcionales que aplican única y exclusivamente a el caso de uso que se está elaborando, todos los casos de uso sumados dan como resultado la funcionalidad total que se requiere del sistema, por esta

[2] El termino usuario no solo hace referencia a usuarios humanos sino también a otros sistemas.

causa el modelo de casos de uso sirve como un contrato entre el cliente y los desarrolladores.

Los casos de uso deben ser especificados en lenguaje natural y por esta causa los pueden leer todos los involucrados en el desarrollo (analistas, diseñadores, implementadores, testers) y por gente que esta afuera de la organización (clientes).

Los casos de uso proveen un hilo para el desarrollo del sistema ya que: Los diseñadores modelan un sistema que permita a los usuarios hacer las tareas especificadas por los casos de uso, los testers[3] prueban que el sistema es capaz de realizar todos los casos de uso, la documentación debe enseñar cómo hacer todas las tareas en los casos de uso, etc.

Un modelo de caso de uso se compone de:

- **Actores:** Son los usuarios del sistema y sus descripciones, la sintaxis de un actor se indica en la figura 1.3.
- **Caso de uso:** Define una secuencia de acciones que un sistema desempeña, es iniciado por un actor para invocar una funcionalidad específica del sistema. La sintaxis de un caso de uso se indica en la figura 1.3.
- **Diagramas de caso de uso:** Muestran las relaciones de los actores y los casos de uso.

Figura 1.3. Sintaxis de un caso de uso y un actor.

Como se indica en la figura 1.4, ya que se tiene elaborado el modelo de caso de uso: “los desarrolladores analizan y desarrollan el sistema para cumplir los casos de uso, creando en primer lugar un modelo de diseño, después se elabora un modelo de implementación, que contiene el código. Por ultimo los desarrolladores preparan un modelo de prueba que permite verificar que el sistema proporciona la funcionalidad descrita en los casos de uso” [13]

[3] Los testers son los encargados de la tarea de probar el sistema.

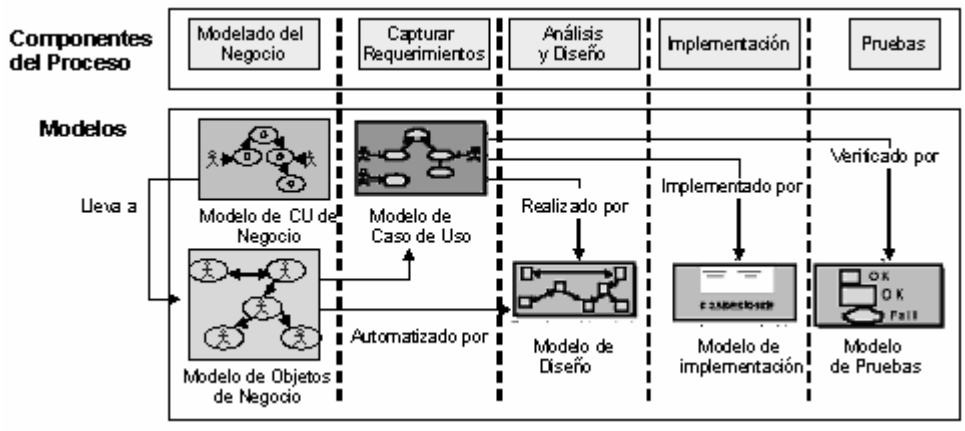

Figura 1.4. Los casos de uso guían todo el proceso.

La figura 1.5 muestra un diagrama de casos de uso para un sistema de cajero automático, el actor Cliente de Banco utiliza el sistema de cajero automático para retirar, ingresar y transferir el dinero de su cuenta. Lo anterior se representa por los tres casos de uso que se muestran y que poseen asociaciones con el actor para indicar su interacción.

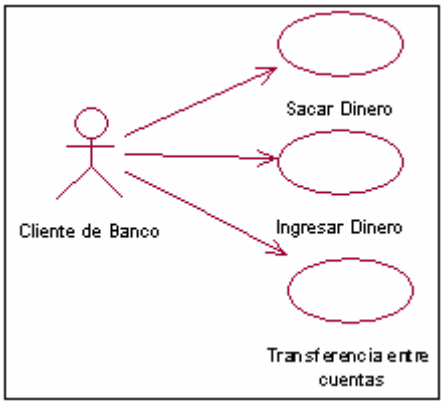

Figura 1.5 Ejemplo de Caso de Uso.

1.4.2 Un proceso centrado en la arquitectura

El RUP define la **arquitectura** como: "**la organización o la estructura de los componentes importantes de un sistema interactuando mediante interfases**" [22]

Pero, ¿Cuáles son estos componentes?, la respuesta varía, para algunos es el diagrama de clases, para el cliente o usuario el modelo de casos de uso. Esta idea se asemeja un poco al concepto de vistas en las bases de datos en la que se crean varias vistas, una para el administrador, una para el diseñador, una para la secretaria, etc. La arquitectura en el RUP se representa por medio de lo que se llama **vistas arquitectónicas**.

Philippe Kruchten en un documento llamado "The 4+1 view model of architecture" plasmó la idea de vistas arquitectónicas, como se indica en la figura 1.6. Estas vistas

arquitectónicas se representan por medio de conjuntos de diagramas, que se representan en el Lenguaje Unificado de Modelado (UML).

UML no solo sirve para documentar sistemas de software, sino sistemas en general, en términos de problemas y soluciones, para lo cual los requerimientos son el conjunto de problemas que tiene el usuario y a los productos o servicios que resuelven estos requerimientos se les llama soluciones.

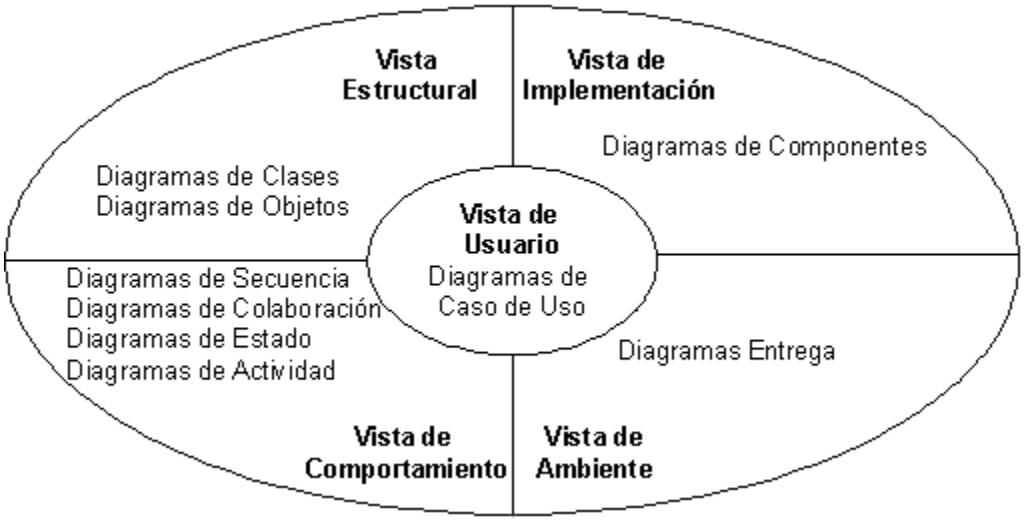

Figura 1.6. Vistas Arquitectónicas.

A continuación se presenta una explicación de cada vista en términos de UML:

- **Vista de Usuario:** Abarca el problema y solución desde la perspectiva de aquellos individuos cuyos problemas son resueltos por medio de la solución.
- **Vista Estructural:** Abarca los aspectos estáticos o estructurales de un problema y su solución
- **Vista de Comportamiento:** Abarca la realización de la solución en aspectos de comportamiento y estructura, estos diagramas describen la organización y las dependencias entre los componentes de implementación del software.
- **Vista de Ambiente:** Abarca los aspectos estructurales y de comportamiento del dominio en el que la solución debe ser realizada [25]

Para cada tipo de diagrama no quiere decir que es todo el modelo en su conjunto, sino los componentes arquitectónicos más significativos, los cuales se discriminan por medio de: los casos de uso que ayudan a mitigar los riesgos más críticos, los CU más importantes para los usuarios, las partes del sistema que se pueden reutilizar, las políticas y estándares

de la organización y los requisitos no funcionales generales, entre otros. Así pues, la idea de un proceso centrado en la arquitectura significa que la arquitectura es usada para:

- Tener una abstracción común del problema y su solución sin detenerse en aspectos poco significativos.
- Especificar, realizar y validar metodológica y sistemáticamente una solución a un problema.
- Capturar, permitir e influenciar la comunicación de conocimiento.
- Administrar la complejidad y mantener la integridad del sistema[25]

1.4.3 Un proceso iterativo e incremental

El software desde hace mucho tiempo es desarrollado como una secuencia de disciplinas encadenadas una detrás de otra, iniciando con los requerimientos, siguiendo con el diseño, codificación, pruebas, entrega y mantenimiento. Siendo el criterio para pasar de una disciplina a otra los documentos generados.

Sin embargo, por lo general no todos los requerimientos son comprendidos por el analista o expresados por el cliente, a veces sucede que el analista o el cliente creen que algunos aspectos son obvios y no los expresan, también es frecuente que a la mitad del desarrollo, el cliente adiciona requerimientos, entonces el costo de implementación de tal requerimiento sube en el orden de 100 a 1000, dependiendo del tipo de necesidad, que si se hubiera desarrollado desde el inicio.

El RUP implementa la idea del desarrollo iterativo, en el cual se pasa varias veces por todas las disciplinas (requerimientos, análisis y diseño,…), mejorando el entendimiento de los requerimientos gradualmente, construyendo una arquitectura robusta y haciendo entregas parciales que son cada vez más completas. Cada vez que se recorre el conjunto de disciplinas se le conoce como **iteración**. En cada iteración se hace énfasis en una o varias disciplinas, por ejemplo en las primeras iteraciones el énfasis se hace en la captura de requerimientos, modelado del negocio, análisis y diseño, mientras que en las últimas iteraciones se hace más énfasis en la implementación, pruebas y distribución como se indica en la figura 1.7.

Desde la perspectiva de desarrollo, este ciclo de vida es una sucesión de iteraciones, es decir, el software se desarrolla iterativamente. Cada iteración concluye con un liberable, el cual es un ejecutable que tiene algunas características que el sistema debe tener.

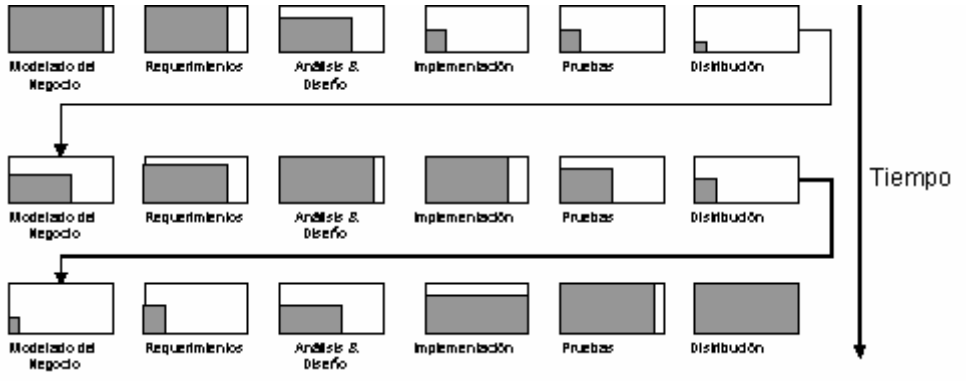

Figura 1.7. Desarrollo iterativo.

Ya que se ha explicado el desarrollo iterativo será más fácil de entender el concepto de "**incremental**", el cual se define como sigue: "La estrategia incremental determina las necesidades del usuario, y define los requerimientos del sistema, después realiza el resto del desarrollo en una secuencia de builds[4]. El primer build incorpora una parte de las capacidades (del sistema) planeadas, el siguiente build adiciona más capacidades y así hasta que el sistema está completo" [4]

1.5 Fases del RUP

Desde una perspectiva, el RUP es un proceso que se descompone en el tiempo en cuatro partes o fases secuénciales, cada fase concluye por una milestone mayor que no es más que el conjunto de criterios para pasar a la siguiente fase.

"Cada vez que se pasa por las cuatro fases es un **ciclo de desarrollo** se produce una **generación de software**. Si el producto no se deja de usar, lo más seguro es que se perfeccione en su próxima generación, repitiendo la misma secuencia de fases (inicio, elaboración, construcción y transición), pero con un énfasis diferente en cada fase. Estos ciclos subsecuentes son llamados **ciclos de evolución**. Los ciclos de evolución pueden ser provocados por actualización en tecnologías, nuevos requerimientos de usuarios ó reacción a la competencia, por mencionar algunos. Los ciclos de evolución típicamente tienen fases de inception y elaboration mucho más cortas, ya que la definición del producto y su arquitectura fueron determinadas en ciclos de desarrollo anteriores" [22]

[4] Un build es una versión operacional de un sistema o parte de un sistema

1.5.1 Fase Inception

En la fase inception se trata de lograr la unidad de todos los stakeholders[5], en cuanto a los objetivos del ciclo de vida para el proyecto. Esta es una fase importante para sistemas nuevos, en los que hay riesgos de negocio y de requerimientos que deben ser resueltos antes de que el proyecto pueda continuar.

Los objetivos primarios de la fase inception son:

- Establecer el alcance del proyecto y las condiciones de frontera del sistema.
- Determinar los casos de uso y escenarios primarios que conducirán las decisiones de diseño más importantes.
- Demostrar una arquitectura candidata contra algunos de los escenarios primarios del sistema.
- Estimar el costo y calendarización global del proyecto.
- Identificar riesgos potenciales [20]

"Al final de la fase inception está la primera milestone mayor del proyecto; la Milestone de Objetivos sobre el Ciclo de Vida. En este punto se examinan los objetivos del ciclo de vida del proyecto y se decide si se sigue con el desarrollo del sistema o se cancela" [22]

1.5.2 Fase Elaboration

"En la fase elaboration se elabora el baseline[6] de la arquitectura del sistema para proveer una base estable para el esfuerzo de diseño e implementación en la siguiente fase. La estabilidad de la arquitectura se evalúa mediante uno o más prototipos arquitectónicos" [22]

Los objetivos primarios de la fase elaboration son:

- Elaborar el Baseline de la arquitectura.
- Establecer el Baseline de la Visión del Proyecto.
- Establecer el baseline para el plan detallado de construcción.
- Demostrar que el baseline de la arquitectura soportará la visión del producto a un costo y tiempo razonables [20]

[5] Los stakeholders son todas las personas o entidades organizacionales que están involucradas en el proyecto, a veces se les llama así a los involucrados en el proyecto pero que no están dentro de la organización de desarrollo, tal como es el caso de los clientes.

[6] Baseline: Un entregable aprobado y revisado de artefactos que constituye una base de acuerdo para evoluciones o desarrollo posteriores y que pueden ser cambiados solo a través de un procedimiento formal, como administración de cambios y control de la configuración

La milestone de arquitectura sobre ciclo de vida establece una baseline para la arquitectura del sistema. En este punto se examinan los objetivos detallados del sistema y el alcance, la selección que se hizo de la arquitectura y la resolución de los riesgos mayores, si no se cumplen los objetivos primarios aun se puede abortar el desarrollo.

1.5.3 Fase Construction

La meta de la fase construction es aclarar todos los requerimientos que pudieran faltar, y completar el desarrollo del sistema basándose en el baseline de la arquitectura. Esta fase de alguna manera es la de manufactura, donde se hace énfasis en administrar los recursos y controlar las operaciones para optimizar los costos, calendarios y calidad.

Los objetivos primarios de la fase construction son:

- Minimizar los costos de desarrollo evitando desechos, retrabajo y optimizando los recursos.
- Alcanzar la calidad adecuada lo más rápido posible
- Alcanzar versiones útiles del sistema tan rápido como sea posible.

La milestone que está al final de la fase construction se llama de “Capacidad Operativa Inicial”, el criterio para pasar a la siguiente fase es: “determinar si el producto esta listo para ser entregado en un ambiente de pruebas” [22]

1.5.4 Fase Transition

En la fase transition se asegura que el software está disponible para los usuarios finales, principalmente incluye pruebas, la preparación del producto para su entrega y ajustes menores debido a la retroalimentación del usuario. Los ajustes que se realizan en la fase transition sirven para afinar el producto, enfocándose en aspectos de usabilidad, instalación, configuración y documentación del usuario.

Los objetivos primarios de la fase de la fase transition son:

- Lograr que el usuario pueda soportarse a si mismo al operar el sistema.
- Completar la satisfacción de los requerimientos de concurrencia y distribución.
- Alcanzar un baseline de producto final con un costo razonable y tan rápido como sea posible [20]

Al final de la fase transition está la milestone de entrega de producto, el criterio es cuestionarse si se cumplieron los objetivos del proyecto. Aquí acaba toda una generación de

producto, terminando con la actividad de Revisión de aceptación del proyecto, donde el cliente acepta formalmente los entregables que se hicieron.

1.6 Disciplinas del RUP

Antes de definir lo que es una disciplina se dará una explicación de lo que es un “flujo de trabajo” que se define como: “una secuencia de actividades que producen un resultado de valor observable” [22]

Explicado lo anterior una disciplina es como un flujo de trabajo que agrupa actividades que se relacionan en torno a una "área de interés mayor” dentro del proyecto total.

La representación de un flujo de trabajo se expresa por medio de un diagrama de actividades.

Como otros flujos de trabajo, un flujo de trabajo de disciplina es una secuencia semi-ordenada de actividades que son realizadas para lograr un resultado particular. Esta naturaleza semi-ordenada de los flujos de trabajo de disciplina enfatiza que no se pueden presentar los aspectos reales de calendarización de un proyecto real, porque de esta forma no se puede representar la opcionalidad de las actividades o la naturaleza iterativa de los proyectos reales. Sin embargo, las disciplinas aun tienen el valor de ayudar a entender el proceso, partiéndolo en diferentes “áreas de interés”.

Como se indica en la figura 1.8, la fase de inception se enfoca más en las disciplinas de modelado del negocio y requerimientos. En la fase de elaboración, el enfoque es en el análisis y diseño.

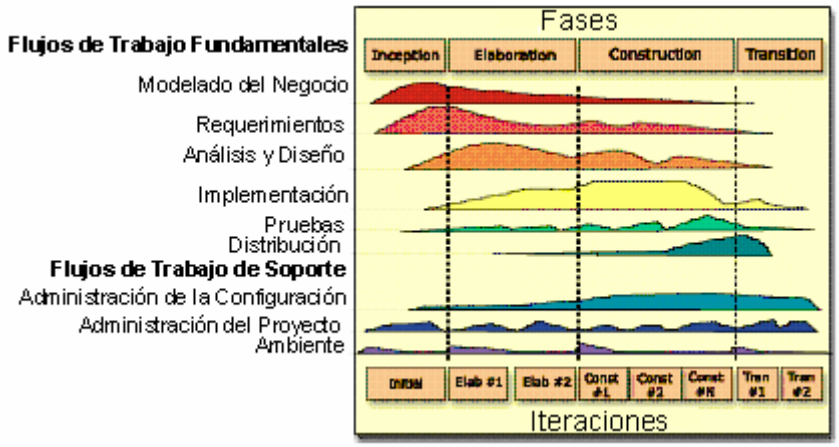

Figura 1.8. El Proceso Unificado de Desarrollo de Software.

Como se indica en la figura 1.8, hay diversas disciplinas que se dividen en flujos de trabajos fundamentales y de soporte, estos últimos, como su nombre lo dice, ayudan a los

fundamentales en actividades de administración, por lo que quedan fuera del alcance de la monografía, solo se explicarán los flujos de trabajo fundamentales que son los que llevan la carga del desarrollo en sí.

1.6.1 Modelado del Negocio

El RUP inicia el esfuerzo de desarrollo entendiendo y representando la dinámica de la organización en la que se pretende realizar el sistema, tal es el caso de la disciplina de modelado del negocio, este es uno de los aportes más importantes del proceso a la ingeniería de software.

La disciplina arranca con el desarrollo de la visión de la empresa lo que implica razonar acerca de cómo se quiere ver la organización a futuro.

Con base en la visión se identifican los procesos actuales de la organización para después refinarlos y automatizarlos, con lo que se mejora la dinámica de los mismos, con base en este perfeccionamiento se definen los roles y responsabilidades. La información anterior se plasma en el modelo de caso de uso de negocio y en el modelo de objetos de negocio.

Además se crean los siguientes artefactos:

- **Glosario:** Define términos importantes en el esfuerzo del modelado del negocio
- **Especificaciones Suplementarias del Negocio:** Presenta cualquier definición del negocio que no se explica en el modelo de caso de uso de negocio y en el modelo de objetos de negocio.

1.6.2 Requerimientos

La siguiente disciplina del RUP llamada de “Requisitos” o “Requerimientos” toma el modelo de objetos de negocio y el modelo de caso de uso de negocio como entrada principal y trata de entender el problema que se quiere resolver por medio del sistema que se va a desarrollar.

En el siguiente paso el analista junto con los interesados externos en el desarrollo del sistema (stakeholders) definen las necesidades de los clientes las cuales se plasman en lenguaje natural en los **Diagramas de caso de uso** que capturan los requerimientos funcionales y en las **Especificaciones Suplementarias** que capturan los requerimientos no

funcionales de este modo los stakeholders y los desarrolladores logran un entendimiento común de los que se quiere realizar.

1.6.3 Análisis y Diseño

En la siguiente disciplina llamada de "Análisis y diseño", se transforman los requerimientos en un diseño de sistema, se desarrolla una arquitectura robusta y se adapta el diseño para concordar con el ambiente de implementación.

Esta disciplina se caracteriza por su esfuerzo en el modelado, ya que se transforman los requerimientos en clases, subsistemas, paquetes y relaciones, además se añaden las condiciones de los requerimientos No-funcionales, se diseña de la Base de Datos y se hace la identificación de componentes.

Los principales artefactos que se relacionan con esta disciplina son:

- **Modelo de diseño:** Es un modelo de objetos describiendo la realización de casos de uso y sirve como una abstracción del modelo de implementación y su código fuente.
- **Modelo de datos:** Es un subconjunto del modelo de implementación que describe la representación lógica y física de los datos persistentes en el sistema. También incluye cualquier comportamiento de la Base de Datos tal como Triggers, Procedimientos almacenados, constraints, etc.
- **Documento de la arquitectura de software:** Es un resumen arquitectónico del sistema, usa el conjunto de vistas arquitectónicas.

1.6.4 Implementación

En la disciplina de implementación se toman los artefactos y esfuerzos hechos en el diseño para transformar las clases, subsistemas y paquetes en archivos fuente, binarios y ejecutables, entre otros. Después se hace una integración de todos los elementos para lograr un sistema ejecutable.

1.6.5 Pruebas

La disciplina de pruebas es en alguna forma un proveedor de servicios para otras disciplinas. Las pruebas se enfocan primariamente en la evaluación de la calidad del producto que se realiza por las siguientes prácticas.

- Encontrar y documentar defectos

- Asesorar acerca de la calidad del software
- Validar las funciones del producto de software
- Validar que los requerimientos han sido implementados apropiadamente.

Una diferencia interesante entre las pruebas y otras disciplinas es que sus tareas están encaminadas principalmente a encontrar debilidades, la diferencia es que las otras disciplinas se centran en la construcción del software.

El artefacto principal de esta disciplina es el modelo de pruebas, las entradas principales para realizar este artefacto son el modelo de casos de uso y las especificaciones suplementarias, es decir, los requerimientos del sistema.

1.6.6 Distribución

La disciplina de distribución agrupa las actividades para poner el software a disposición de los usuarios finales.

Las actividades se pueden desarrollar de diversas formas, depende el tipo de software que sea, algunas de estas formas son:

- Instalación a usuarios (Aplicaciones personalizadas)
- Distribución a puntos de venta (Software Comercial)
- Acceso al software desde internet.
- Proporcionar ayuda y asistencia al usuario (Guías de Usuario, Ayuda en línea, Demos, Tutoriales, Materiales de entrenamiento)
- Migrar el software o los datos existentes
- Aceptación Formal por parte del Usuario

1.7 Conceptos generales del RUP

Los siguientes puntos explican los conceptos generales del RUP que sirven para entender mejor el resto de la monografía.

1.7.1 Trabajador

Un trabajador o rol define el comportamiento y las responsabilidades de un individuo o un grupo de individuos en una organización de desarrollo de software como se indica en la figura 1.9, sin embargo no quiere decir que si un individuo tiene el trabajo o el rol de

"especificador de requerimientos", no puede cumplir otras actividades de otros roles, al contrario, un individuo puede tener varios roles dentro de la organización.

"El comportamiento se refiere a que tiene que realizar un grupo de actividades que se relacionan, las responsabilidades usualmente están definidas y relacionadas a ciertos artifacts" [20]

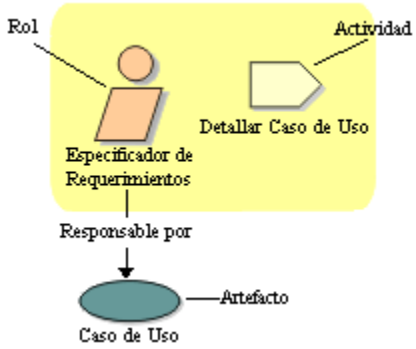

Figura 1.9. Actividades, trabajadores y artefactos.

1.7.2 Actividad

Las actividades son esfuerzos o conjuntos de tareas, las cuales son dirigidas para producir artefactos. La actividad "es una pieza de trabajo susceptible a ser solicitada a un trabajador para desarrollar, puede tardar en realizarse de unas horas a algunos días y puede repetirse de ser necesario en cada iteración" [20]

Alguna de las características de las actividades es que:

- Son especificadas y ordenadas por métodos
- Se realizan dentro de un proceso
- Tienen asociadas a trabajadores que conduzcan o realicen las tareas.
- Deben tener artefactos asociados, los cuales se van a crear, modificar o utilizar conforme se desarrolla la actividad.
- Deben ser tratados como **recomendaciones** las cuales se pueden modificar dependiendo de la organización, el proyecto, los recursos,…

1.7.3 Artefactos

Los artefactos o artifacts son una pieza de información que puede ser creada, modificada o utilizada por un trabajador dentro de una actividad. Los artifacts pueden ser:

- Modelos (Modelo de diseño, Modelo de casos de uso,…)
- Elementos de un modelo (clases, objetos,…)

- Documentos (Documento de arquitectura de software, visión,…)

Ya que el RUP no es un proceso dirigido por documentos, se pueden usar únicamente los artefactos que se necesiten, en el whitepaper “The Ten Essentials of RUP - The Essence of an Effective Development Process” [18], se recomienda empezar solo por los elementos mas esenciales y de ahí, ir adicionando los artefactos que se crean convenientes.

1.7.4 Detalle de Flujo de Trabajo

Las disciplinas son flujos de trabajo que se pueden representar por medio de diagramas de actividades, este diagrama de actividades está compuesto por detalles de flujos de trabajo.

Los flujos de trabajo muestran grupos de actividades que se realizan unidas y muestran los roles involucrados, los artefactos de entrada y salida y las actividades a realizar. En la figura 1.10 se muestra el ejemplo de un detalle flujo de trabajo de la disciplina de requerimientos, en el que participan Roles, Artifacts y Actividades.

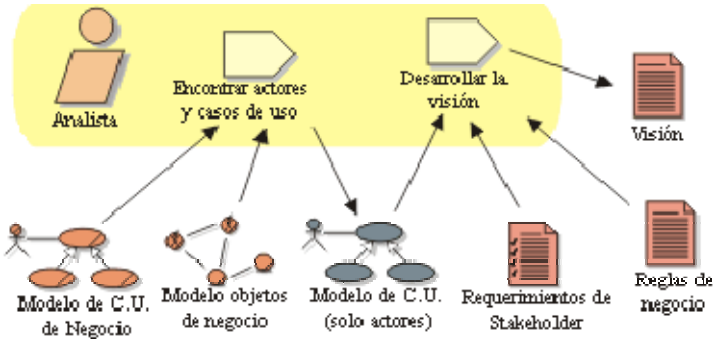

Figura 1.10. Detalle de Flujo de Trabajo.

“Los detalles de flujo de trabajo en el RUP siempre representan una sola iteración en el proyecto. Ya que cada detalle de flujo de trabajo se realiza en el lapso de una iteración” [3]

1.8 Breve explicación del análisis y diseño

En la disciplina del análisis y diseño se transforman los requerimientos en un diseño de sistema, se desarrolla una arquitectura robusta y se adapta el diseño para concordar con el ambiente de implementación.

La figura 1.11 muestra el flujo de trabajo de análisis y diseño con todos sus detalles de flujo de trabajo, en ella se indica que los esfuerzos de esta disciplina se pueden empezar en la fase inception elaborando una primera aproximación a la solución del problema (realización de una síntesis arquitectónica), después, ya en las primeras iteraciones de la

fase elaboration, se crea una arquitectura inicial para el sistema (definición de una arquitectura candidata). Si la arquitectura ya existe es decir, se produjo en iteraciones o proyectos previos, el enfoque está en refinar la arquitectura (refinación de la arquitectura), analizar el comportamiento y crear un conjunto inicial de elementos que suministren un comportamiento apropiado (análisis del comportamiento de los casos de uso).

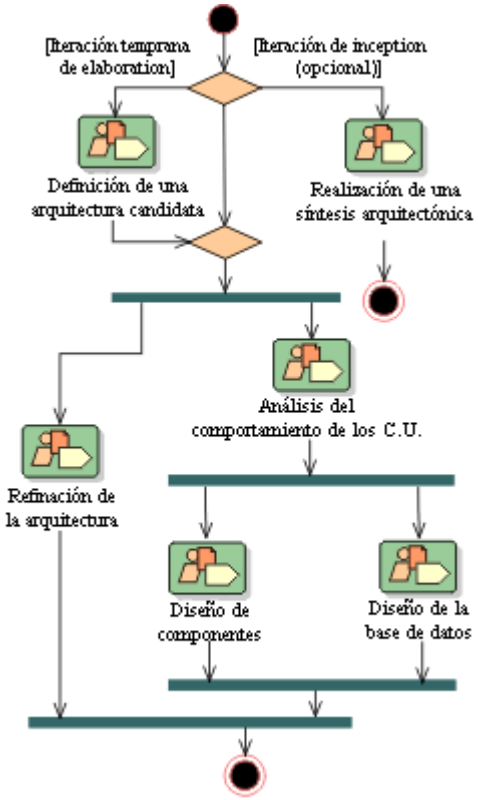

Figura 1.11. Disciplina del análisis y diseño.

Después que los elementos iniciales son identificados, se refinan. El detalle de flujo de trabajo “Diseño de componentes” produce un conjunto de componentes que provee el comportamiento adecuado para satisfacer los requerimientos del sistema. En paralelo con estos esfuerzos, los aspectos de persistencia son tratados en el detalle de flujo de trabajo “Diseño de la base de datos”. El resultado del análisis y diseño es un conjunto inicial de componentes que son la entrada principal de la disciplina de implementación.

Los artefactos de entrada principales para el análisis y diseño son: **el modelo de caso de uso, el glosario y las especificaciones suplementarias de la disciplina de requerimientos**. El resultado del análisis y diseño es un “modelo de diseño “que sirve como una abstracción del código fuente. Este modelo consiste de clases listas para

implementarse, estructuradas en paquetes de diseño, también contiene descripciones de cómo los objetos de estas clases colaboran para realizar los casos de uso (realizaciones de caso de uso).

Capítulo 2

Realización de una síntesis arquitectónica

"Ustedes saben que en una carrera todos corren, pero solamente uno recibe el premio. Pues bien, corran ustedes de tal modo que reciban el premio."

1 Corintios 9:24

Introducción

Cuando se habla de “correr” de tal modo que se reciba el premio quiere decir que el proceso se debe de planear y ejecutar de tal modo que al final se reciban los resultados esperados, el premio: un sistema robusto. En la disciplina de análisis y diseño, se inicia con la elaboración de un estudio de factibilidad que sirve para razonar acerca de si el proyecto se puede realizar o no. El estudio de factibilidad se realiza en una serie de actividades que se juntan en un detalle de flujo de trabajo llamado: “Realización de una síntesis arquitectónica”. En la figura 2.1 se indica que la realización de una síntesis arquitectónica se realiza en la fase inception.

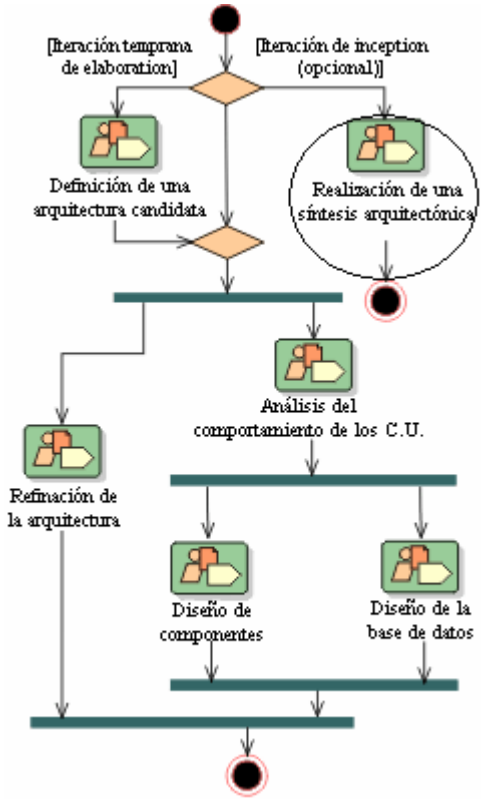

Figura 2.1. Detalle de flujo de trabajo “Realización de una síntesis arquitectónica”.

Como se indica en la figura 2.2, para realizar este flujo de trabajo se requieren realizar tres actividades principales:

- **Análisis arquitectónico:** En este paso se recopilan los requerimientos significativos desde el punto de vista de la arquitectura.
- **Construcción de la demostración conceptual de la arquitectura:** Se construye una solución conceptual que dé respuesta a los requerimientos.
- **Evaluación de viabilidad de la demostración conceptual de la arquitectura**

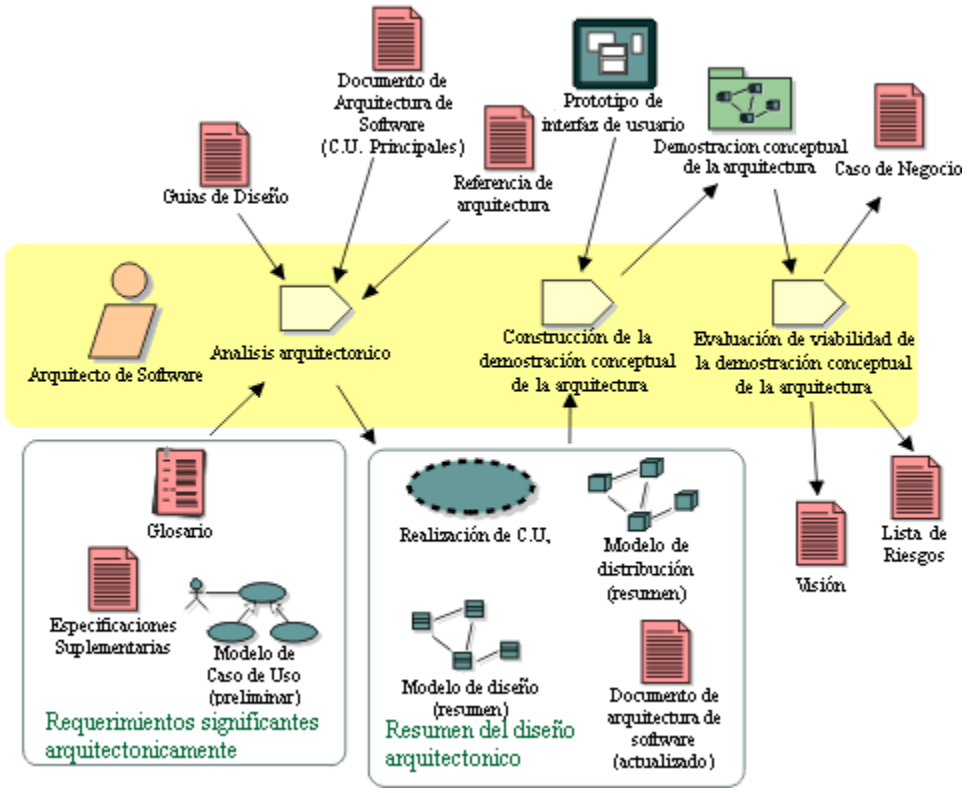

Figura 2.2. Actividades, Artefactos y Roles del detalle de flujo de trabajo "Realización de una síntesis arquitectónica".

En la figura 2.2 se indica que todas las actividades son realizadas por el arquitecto de software, una descripción de este trabajador se encuentra en el apéndice B.

Los siguientes puntos muestran los artefactos más críticos en la realización de una síntesis arquitectónica.

- Documento de Arquitectura de software.
- Demostración conceptual de la arquitectura.
- Realización de casos de uso.

2.1 Propósito del flujo de trabajo "Realización de una síntesis arquitectónica"

El propósito de la realización de una síntesis arquitectónica es construir y evaluar una "Demostración conceptual de la arquitectura". Lo anterior con el objetivo de mostrar que existe o puede existir, una solución que satisface los requerimientos significativos del sistema

El objetivo principal no el de construir el sistema en si, sino un análisis de viabilidad del proyecto, así es que puede ser limitado a una o dos iteraciones.

2.2 Análisis arquitectónico

El análisis arquitectónico es la primera actividad que se realiza en el detalle de flujo de trabajo “realización de una síntesis arquitectónica”. En este caso se trata de definir una arquitectura candidata “conceptualmente”, basándose en la experiencia del desarrollo en sistemas anteriores o dominios de problema parecidos.

Esta actividad puede ser omitida en sistemas que tengan una arquitectura bien definida. Por lo tanto se recomienda realizarla solo en sistemas nuevos, o en los que no se tiene experiencia de dominios similares.

2.2.1 Desarrollo de un resumen de la arquitectura

El primer paso de la actividad “Análisis arquitectónico” es el desarrollo de un resumen de la arquitectura”, su propósito es facilitar una visión conjunta; explorando y evaluando opciones de arquitectura de alto nivel, además de transmitir a los clientes, equipos de desarrollo y otros stakeholders un entendimiento anticipado de la estructura de alto nivel del sistema planeado.

El resumen de la arquitectura se crea al inicio del proyecto, en la fase inception. Este resumen refleja las decisiones iniciales y los cálculos del trabajo, además de las decisiones de la arquitectura física y lógica y los requerimientos no-funcionales del sistema.

La ejecución se realiza en conjunto con los clientes, por esta razón, el resumen de la arquitectura puede ser informal tal como un story-board, una imagen o graficas con símbolos para que se puedan entender mejor.

Al resumen se le llama conceptual, porque refleja las partes naturales de la solución propuesta, transmitiendo las ideas críticas e incluyendo los bloques de construcción de mayor abstracción. El nivel de formalidad depende del proyecto, en un proyecto de alto riesgo se puede llegar a poner el resumen de la arquitectura en el documento de arquitectura de software para que se revise formalmente.

No se deben hacer compromisos basándose en el resumen de la arquitectura hasta que se hayan desarrollado y validado los modelos de distribución y de implementación. El resumen de la arquitectura se debe de refinar cada vez más para mejorar su efectividad como vehículo de comunicación.

2.2.2 Búsqueda de activos de valor para el proyecto

El segundo paso de la actividad "Análisis arquitectónico" es identificar los activos que pueden ser de valor relevante para el proyecto, analizar los ajustes entre los activos y los requerimientos del proyecto, además de encontrar y listar activos potenciales para ser reusables en el proyecto.

La primera tarea es hacer una búsqueda de activos o elementos que puedan ayudar a satisfacer los requerimientos del usuario. Antes de esto se deben entender los requerimientos del ambiente en el que se pretende incorporar el activo.

Después se realiza una búsqueda de activos en donde se crea que hay más posibilidad de encontrar elementos de ayuda. Los activos o elementos de valor pueden ser de diferentes tipos, por ejemplo, modelos industriales, frameworks[1] , clases reusables y experiencia.

Ya que se encontraron los activos candidatos, se hace un análisis para verificar si los elementos disponibles pueden contribuir a resolver los principales desafíos del proyecto actual y son compatibles con las limitaciones del proyecto.

Cuando se encontraron y analizaron los elementos de valor para el proyecto se procede con un análisis sobre el alcance que tendrá el ajustar los activos a los requerimientos del cliente y se considera una negociación, a fin de que el elemento se pueda usar.

Otro punto importante es el grado en el que el activo se puede modificar o extender para su uso en el proyecto, y el costo, riesgo y funcionalidad al adoptar el elemento

Al final de deben documentar las decisiones y los razonamientos que se hicieron para decidir si se adopta o no cada activo de valor.

2.2.3 Identificación de las principales abstracciones

El siguiente paso para la realización del análisis arquitectónico es la identificación de las principales abstracciones, su propósito es iniciar los esfuerzos para el análisis, identificando los principales conceptos o abstracciones que el sistema debe manejar.

Lo anterior se realiza solo para orientar al arquitecto de software en la selección de activos para la construcción del artefacto "demostración conceptual de la arquitectura" y permitir la construcción de las interacciones entre abstracciones.

[1] Un framework o esqueleto define un método general para resolver un problema, estos difieren de un patrón en su escala y su alcance, los frameworks describen un solución estructural a un problema particular que puede omitir varios detalles.

Las actividades de las disciplinas de requerimientos y modelado del negocio comúnmente descubren conceptos clave que el sistema debe ser capaz de manejar; estos conceptos son en si mismos abstracciones principales de diseño.

Las fuentes para encontrar las abstracciones son: el glosario, el modelo de objetos de negocio, los requerimientos, etc. Mientras que se están definiendo las abstracciones principales también se deben definir las relaciones entre estas. Las abstracciones principales se deben presentar en forma de diagramas de clase y crear una corta descripción de cada una.

El fin de este paso no es encontrar un conjunto de abstracciones que sobrevivan hasta el diseño, sino encontrar un conjunto de conceptos principales que el sistema debe manejar. Las abstracciones se representan como una clase de UML y no se debe de gastar mucho tiempo en describir las abstracciones a detalle porque hay muchas relaciones que existen en las clases pero que los casos de uso no necesitan, por lo tanto se esta trabajando de más, en esta etapa inicial se deben modelar solo las relaciones principales ir progresando conforme se van revisando los casos de uso.

2.2.4 Desarrollo de un modelo de distribución de alto nivel

El propósito del desarrollo de un modelo de distribución de alto nivel, es proveer una base para evaluar la viabilidad de la implementación del sistema, adquirir el conocimiento de la distribución geográfica y la complejidad operacional del sistema, además de proveer las bases de una estimación temprana de esfuerzo y costo.

En este paso se toma el resumen de arquitectura para desarrollar un modelo de distribución de alto nivel, es decir, a un nivel conceptual, ya que no se puede detallar mucho porque aun no está avanzado el análisis ni mucho menos el diseño.

Las partes principales del modelo de distribución son las siguientes:

- **Usuarios** (en sus locaciones), los cuales están definidos en la parte de “perfiles de usuario” de la visión, casos de uso y modelos de caso de uso.
- **Datos del negocio** (en el modelo de objetos de negocio y en el modelo de diseño)
- **Requerimientos de niveles de servicio** (en las especificaciones suplementarias)
- **Limites** (en las especificaciones suplementarias)

El modelo de distribución debe ser capaz de soportar a los usuarios realizando los casos de uso, los cuales acceden a componentes. Estos componentes se ponen provisionalmente

en nodos. Se debe validar que las conexiones entre los nodos son adecuadas para soportar la interacción entre los componentes.

Solo se debe hacer una descripción de nodos detallada si es importante para la evaluación de viabilidad.

Aunque este es el primer modelo de distribución que se produce en el proyecto, y además de que se desarrolla rápidamente por ser de alto nivel, se pueden identificar componentes de hardware y software si se conoce esta información, o si es importante tomar la decisión, de definir el hardware y software en el que se debe de correr el sistema, en este momento del proyecto debido a que es una base para la estimación de costos.

2.2.5 Identificación de interacciones entre abstracciones

El siguiente paso del análisis arquitectónico es identificar aquellas interacciones entre las abstracciones principales, que caracterizan, es decir son representativas de tipos significativos de actividad en el sistema. Las interacciones se modelan por medio de un diagrama de colaboración, el cual por lo general se encuentra en el artefacto “realización de caso de uso”.

2.2.6 Revisión de resultados

Ya que se ha concluido la actividad de análisis arquitectónico, se debe realizar una revisión de los elementos que se han identificado, para asegurarse de que estos son completos y consistentes. Como los resultados son preliminares y relativamente informales, las revisiones pueden ser informales también. Los escenarios de los casos de uso importantes pueden ser usados para validar las selecciones arquitectónicas del software, del hardware y las abstracciones.

2.3 Construcción de la demostración conceptual de la arquitectura

La segunda actividad del detalle de flujo de trabajo “Realización de una síntesis arquitectónica” es la construcción del artefacto “Demostración conceptual de la arquitectura”, su propósito es sintetizar al menos una solución que satisfaga los requerimientos críticos del sistema.

2.3.1 Toma de decisiones sobre el método de construcción

El primer paso para desarrollar la demostración conceptual es seleccionar las técnicas que serán usadas para la construcción del mismo, algunas de estas son:

- Realizar solo un modelo conceptual
- Prototipado “rápido”.
- Simulación.
- Translación automática de especificaciones a código.
- Especificaciones “ejecutables”

El arquitecto de software debe ser capaz de razonar sobre estos modelos, en el proceso de descubrir algo acerca de los espacios del problema y solución.

La siguiente tarea es seleccionar de las tecnologías y elementos identificados en el análisis arquitectónico, aquellos que serán usados en la construcción de la demostración conceptual de la arquitectura, entre ellos están el modelo de distribución, las principales abstracciones y sus interacciones, entre otros elementos.

2.3.2 Construcción de la demostración conceptual de la arquitectura.

Usando las técnicas preseleccionadas en el punto anterior, el arquitecto de software construye la demostración conceptual de la arquitectura usando las tecnologías y elementos seleccionados para satisfacer (dentro del alcance requerido de acuerdo al perfil del riesgo del proyecto) los requerimientos significativos arquitectónicamente como los capturados en el diseño del resumen, los modelos de distribución y el documento de arquitectura de software.

2.4 Evaluación de la viabilidad de la demostración conceptual de la arquitectura

La tercera y última actividad del detalle de flujo de trabajo de la realización de una síntesis arquitectónica es evaluar la viabilidad del artefacto “Demostración conceptual de la arquitectura”, con el propósito de determinar si los requerimientos más críticos del sistema son factibles y se pueden satisfacer.

Los siguientes puntos muestran los pasos para el desarrollo de esta actividad.

2.4.1 Determinación de los criterios de evaluación

El primer paso de la actividad de evaluación de la demostración conceptual de la arquitectura es determinar los criterios con los que se evaluará el artefacto, estos se obtienen de los requerimientos del sistema que están principalmente en el glosario, las especificaciones suplementarias y el modelo de casos de uso. Los tres artefactos anteriores se van desarrollando en la disciplina de requerimientos que es la que precede a la del análisis y diseño. Los criterios pueden variar de sistema a sistema pero al fin deben de asegurar que los requerimientos mas importantes del sistema se satisfagan.

2.4.2 Prueba de la demostración conceptual de la arquitectura

En el segundo paso de la actividad se prueba la demostración conceptual de la arquitectura contra los criterios de evaluación. La forma de esta prueba depende del modo en el que se desarrolló este artefacto, algunas de las formas que puede tomar este artefacto se muestran a continuación.

- Una lista de tecnologías conocidas que se sabe que son apropiadas para la solución.
- Un bosquejo de un modelo conceptual de una solución usando UML.
- Una simulación de la solución.
- Un prototipo ejecutable [22]

En el caso de un prototipo ejecutable, la prueba puede ser por medio de la demostración del prototipo; en el caso de un modelo conceptual, razonando e inspeccionando las partes y conexiones de los elementos del modelo; para una simulación, configurando el ambiente de modo que concuerde con datos de entrada que se deriven de los criterios de evaluación, para después, colectar los datos de salida y evaluarlos.

2.4.3 Evaluación de resultados

El tercer y último paso de la actividad es evaluar los resultados de la prueba para determinar si pueden ser satisfechos los requerimientos arquitectónicamente significativos. En este momento del desarrollo, los requerimientos aun pueden cambiar y no necesariamente son bien entendidos por los stakeholders. Por ejemplo, quizás existe la oportunidad de relajar requerimientos que han sido catalogados como de alto riesgo en la prueba de la demostración conceptual de la arquitectura. Se deben de explorar

completamente todos estos caminos en la evaluación de los resultados, en contraste con la situación en fases posteriores como la elaboración y la construcción donde hay menor disposición para cambiar o reinterpretar los requerimientos. Después de la evaluación, con un mejor entendimiento del alcance y la factibilidad por todos los involucrados en el proyecto, si es necesario se deben cambiar las propuestas del caso de negocio, la visión y la lista de riesgos.

2.5 Artifacts principales

Los siguientes puntos muestran los artifacts principales que se crean, modifican o utilizan en el detalle de flujo de trabajo “Realización de una síntesis arquitectónica”.

2.5.1 Documento de arquitectura de software

El documento de arquitectura de software provee de una descripción comprensible de la arquitectura del sistema usando las diferentes vistas arquitectónicas para representar diferentes aspectos del sistema. Este artefacto lo desarrolla el arquitecto de software.

El documento de arquitectura de software sirve como un medio de comunicación entre el arquitecto de software y los otros miembros de equipo del proyecto con respecto a decisiones significativas arquitectónicamente que se han hecho en el proyecto.

2.5.1.1 Partes importantes del documento de arquitectura de software

Los siguientes puntos muestran las partes importantes del documento de arquitectura de software:

- **Introducción:** La introducción del documento de arquitectura de software debe proveer un resumen de todo el documento. Debe incluir el propósito, alcance, definiciones, acrónimos, abreviaciones, referencias y un resumen.
- **Representación arquitectónica:** Esta sección describe lo que es la arquitectura para el sistema actual, y como se representa. Además se enumeran las vistas arquitectónicas que son necesarias para el proyecto y qué tipo de modelos contiene cada una.
- **Metas y limitantes arquitectónicas:** Esta sección describe los objetivos y los requerimientos del software que tienen algún impacto significativo en la arquitectura, por ejemplo, seguridad, privacidad, portabilidad, distribución y reuso. También captura las limitantes especiales que pueden aplicar: estrategias de diseño

e implementación, herramientas de desarrollo, estructura del equipo, calendarización, códigos legales, etcétera.

(continuación de las partes más importantes del documento de arquitectura de software)

- **Vista de caso de uso:** Esta sección lista los casos de uso o escenarios del modelo de caso de uso que representan elementos significativos de la funcionalidad central del sistema final.
- **Vista lógica:** Describe las partes del modelo de diseño que son significativas arquitectónicamente, tales como su descomposición en subsistemas y paquetes. También se deben incluir las clases significativas y describir sus responsabilidades, así como las relaciones, operaciones y atributos importantes. En esta sección debe haber una subsección llamada "Paquetes de diseño significativos arquitectónicamente" en la que para cada paquete se incluya una subsección con el nombre del paquete, una descripción breve y un diagrama con todas las clases y paquetes contenidos dentro del paquete. Para cada clase significativa en el paquete, se debe incluir nombre, una breve descripción y opcionalmente una descripción de algunas de sus responsabilidades, operaciones y atributos importantes.
- **Vista de proceso:** Esta sección describe la descomposición del sistema en procesos. Las secciones se organizan por grupos de procesos que se comunican o interactúan. También se describen los principales modos de comunicación entre procesos, tales como: paso de mensajes e interrupciones.
- **Vista de distribución:** Esta sección describe una o más configuraciones de redes en las que el software será ejecutado. Como mínimo, para cada configuración, se deben indicar los nodos físicos que ejecutan el software (computadoras, CPUs) y sus interconexiones (bus, LAN, punto a punto,…).
- **Vista de implementación:** Esta sección describe toda la estructura del modelo de implementación, la descomposición del software en capas y subsistemas, además de cualquier componente que sea significativo arquitectónicamente.
- **Vista de Datos:** Esta es una vista desde la perspectiva del almacenamiento de datos persistentes en el sistema.

2.5.2 Demostración conceptual de la arquitectura

La demostración conceptual de la arquitectura es una solución a los requerimientos significativos arquitectónicamente que son identificados durante etapas tempranas en la fase inception, particularmente en la disciplina de requerimientos.

El rol que realiza este artefacto es el arquitecto de software y se desarrolla en la fase inception para ayudar a determinar la factibilidad del proyecto y evaluar lo riesgos técnicos del desarrollo, además de refinar y reformular los requerimientos de la arquitectura.

En este artefacto se deben de atacar los riesgos más altos que son las necesidades en las que se pone mucho del esfuerzo total del sistema, con la expectativa de que los modelos que se evalúan y producen sean lo más realistas posibles, para que de esta forma todos los clientes estén convencidos de que la base para comprometer los fondos y continuar en la fase elaboración son creíbles. Sin embargo se tiene que saber que no todos los riesgos se pueden eliminar en esta fase.

Este artefacto tiene el propósito de determinar si existe, o es posible que exista, una solución que satisfaga los requerimientos significativos.

2.5.2.1 Representación

La demostración conceptual de la arquitectura puede tener muchas formas, entre las cuales están:

- Una lista de tecnologías conocidas (frameworks, patrones, arquitecturas ejecutables) que se sabe que son apropiadas para la solución.
- Un bosquejo de un modelo conceptual de una solución usando UML.
- Una simulación de la solución.
- Un prototipo ejecutable [22]

2.5.3 Realización de casos de uso

Una realización de caso de uso detalla cómo se realiza un caso de uso particular dentro del modelo de diseño, en términos de objetos de colaboración.

“La realización de caso de uso posee una descripción textual del flujo de sucesos de algún caso de uso por medio de diagramas de clases y diagramas de interacción que representan la realización de un flujo o escenario particular del caso de uso en términos de interacciones entre objetos” [13]

El propósito de una realización de caso de uso es separar los intereses de los especificadores del sistema (representados por el modelo de caso de uso y los requerimientos del sistema) de los intereses de los diseñadores del sistema. “Una realización de caso de uso representa la perspectiva de diseño de un caso de uso. Es una organización de elementos de modelos usados para agrupar artefactos relacionados al diseño de un caso de uso” [22]. Estos artefactos relacionados, típicamente consisten en los diagramas de colaboración y secuencia, los cuales expresan el comportamiento del caso de uso en términos de colaboración de objetos.

2.5.3.1 Partes importantes del artefacto

Los siguientes puntos muestran las partes importantes del artefacto “Realización de caso de uso”.

- **Descripción de la realización de caso de uso:** Se realiza una descripción textual de cómo se realiza el caso de uso en términos de colaboración de objetos. Su propósito principal es resumir los diagramas conectados al caso de uso y explicar cómo se relacionan.
- **Diagramas de interacción:** En esta parte se colocan los diagramas de interacción (de colaboración y secuencia) que describen como se realizan los casos de uso en términos de colaboración de objetos.
- **Diagramas de clase:** Los diagramas de clase describen como participan tanto las clases como sus relaciones en la realización de un caso de uso.

Capítulo 3

Definición de una arquitectura candidata

Ninguna persona que no es un gran escultor o pintor puede ser un arquitecto. Si no es un escultor o pintor, tan solo puede llegar a ser un constructor

John Ruskin - Lecturas de arquitectura y pintura

Introducción

Ahora que se sabe que el proyecto es viable, es tiempo de empezar a crear la arquitectura que sostendrá todo el sistema. En la línea del tiempo, el detalle de flujo de trabajo "Definición de una arquitectura candidata" se encuentra en las primeras iteraciones de la fase elaboration, como se indica en la figura 3.1.

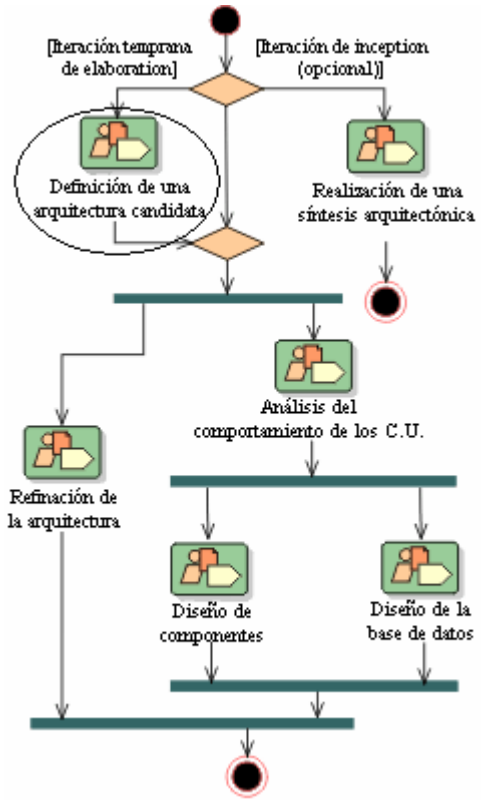

Figura 3.1. El detalle de flujo trabajo "Definición de una arquitectura candidata".

Este detalle de flujo de trabajo tiene dos actividades principales, como se indica en la figura 3.2, las cuales están mutuamente relacionadas, pues los artefactos de salida de la actividad "Análisis arquitectónico" son la entrada de la actividad "Análisis de caso de uso", primero se hace da un paso inicial para desarrollar la arquitectura, después se toma cada uno de los caso de uso significativos y por cada uno se realiza un "Análisis de caso de uso". Después de que se ha analizado cada caso de uso, se actualiza la arquitectura, de tal modo que refleje las adaptaciones requeridas para acomodar el nuevo comportamiento al sistema y para resolver los problemas arquitectónicos que se han identificado.

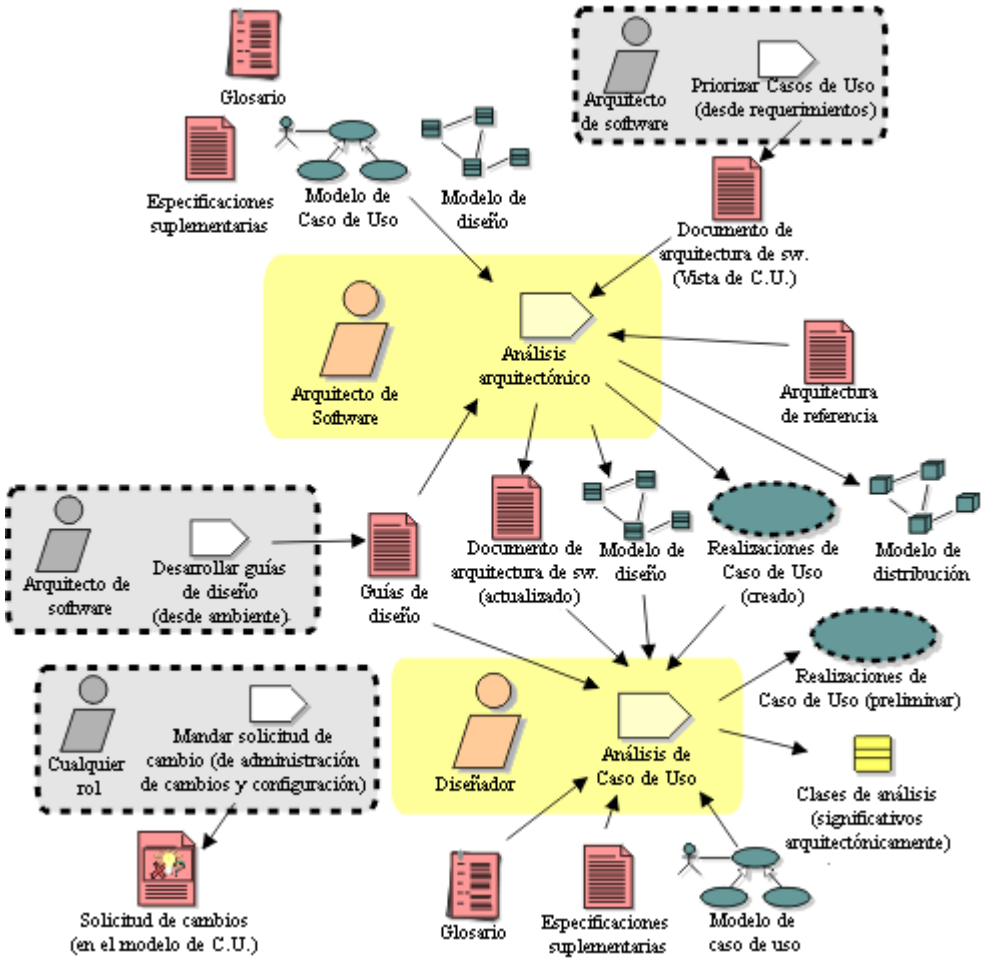

Figura 3.2. Actividades, Artefactos y Roles del detalle de flujo de trabajo “Definición de una arquitectura candidata”.

3.1 Propósito del flujo de trabajo “Definición de una arquitectura candidata”

El propósito del detalle de flujo de trabajo “Definición de una arquitectura candidata” es

- Crear un bosquejo inicial de la arquitectura del sistema
- Identificar las clases de análisis de los casos de uso significativos arquitectónicamente.
- Actualizar las realizaciones de caso de uso (de acuerdo) con las interacciones de las clases de análisis.

3.2 Conceptos preliminares

Antes de comenzar a explicar las actividades del detalle de flujo de trabajo “Definición de una arquitectura candidata”, se deben explicar conceptos que son importantes para la comprensión del capítulo, los cuales se explican en los siguientes puntos.

3.2.1 Mecanismos arquitectónicos

Un mecanismo arquitectónico es una decisión estratégica en cuanto a estándares, políticas y prácticas del sistema. Es la definición de temas que deben ser estandarizados en el proyecto.

“Un mecanismo arquitectónico representa una solución común a un problema que se encuentra frecuentemente” [23]. Los mecanismos arquitectónicos son una parte importante de la unión entre la funcionalidad que se requiere del sistema y cómo se realiza esta funcionalidad cuando se definen las limitantes en el ambiente de implementación.

Hay tres categorías de mecanismos arquitectónicos los cuales varían solo del nivel de refinamiento, es decir, de ser un concepto a su implementación.

- **Mecanismos de Análisis:** Capturan los aspectos principales de una solución de tal manera que sea independiente de la implementación. Estos mecanismos proveen comportamientos específicos a clases o componentes de un dominio relacionado.
- **Mecanismos de diseño:** Estos mecanismos son más concretos que los mecanismos de análisis ya que abordan algunos detalles del ambiente de implementación, pero no están atados a una implementación específica.
- **Mecanismos de implementación:** Especifican una implementación concreta a los mecanismos. Estos deben hacer referencia a cierta tecnología, lenguaje de implementación, proveedores, etc.

Los siguientes tres puntos muestran un ejemplo de la evolución de los mecanismos:

- **Mecanismo de análisis:** Persistencia
- **Mecanismo de Diseño:** RDBMS(Relational Data Management System)
- **Mecanismo de Implementación:** Oracle

3.2.2 Mecanismos de análisis

Un mecanismo de análisis es un patrón que constituye una solución común a un problema común. Los mecanismos pueden mostrar patrones de estructura y/o

comportamiento. Estos mecanismos son usados durante el análisis para reducir su complejidad y para mejorar su consistencia ya que “permiten enfocar el esfuerzo del análisis en trasladar los requerimientos funcionales en conceptos de software sin atascarse en especificaciones de comportamiento relativamente compleja que se necesita para soportar la funcionalidad pero que no es parte central para ella” [22]

Por ejemplo en el análisis puede haber la necesidad de tener objetos que permanezcan en más de un caso de uso o aun que sobrevivan cuando el sistema inicia o termina de ejecutarse. En estos casos hay un requerimiento de persistencia, el cual es un mecanismo complejo y en el que por lo pronto, el equipo de desarrollo no se puede distraer. Lo anterior da lugar a un mecanismo de análisis de persistencia que permita al equipo de desarrollo hablar de objetos persistentes y capturar los requerimientos que tendrá el mecanismo de persistencia sin preocuparse acerca de qué hará o cómo trabajará exactamente tal mecanismo.

Algunos problemas arquitectónicos comunes que los integrantes del proyecto enfrentan son: la distribución, seguridad, interfaces con sistemas heredados, persistencia, detección y manejo de errores, entre otros.

3.2.3 Búsqueda y mapeo de mecanismos de análisis

El proceso para la búsqueda y mapeo de los mecanismos de análisis inicia con la recolección de mecanismos los cuales pueden aparecer bajo distintos nombres a lo largo de diferentes realizaciones de casos de uso o diferentes diseñadores. Por ejemplo, almacenamiento, persistencia, base de datos, y repositorio deben referirse todos a un mecanismo de persistencia; comunicación entre procesos, paso de mensajes, o invocación remota, son mecanismos de comunicación entre procesos.

Ya que se han recolectado todos los mecanismos de análisis se procede con la unificación de los mismos, lo anterior con el fin de eliminar la redundancia. Por ultimo se hace un mapeo o intersección de cada clase con el o los mecanismos que van a soportar la funcionalidad del sistema

3.2.4 Layering

Antes de definir el término layering se debe de comenzar por definir el término patrón arquitectónico. Un patrón arquitectónico expresa un esquema de organización estructural

fundamental para un sistema. Este provee un conjunto predefinido de subsistemas[1], especifica sus responsabilidades e incluye reglas y guías para organizar las relaciones entre los subsistemas. Existen diferentes tipos de patrones arquitectónicos, uno de ellos es el layering.

El termino layering se refiere al agrupamiento de funcionalidad por capas de acuerdo a los diferentes niveles de abstracción. La actividad se realiza con el fin de administrar la complejidad del sistema y para mejorar la reusabilidad.

Una estructura en capas empieza en el nivel de funcionalidad más general, el cual es la base de las demás capas y va creciendo conforme el nivel de funcionalidad es más específico.

"Las capas se usan para encapsular limites conceptuales entre diferentes tipos de servicios y proveen abstracciones útiles para hacer el diseño más fácil de entender" [23]. Cuando se hace y se calcula el número y descomposición de las capas se debe tener en cuenta la complejidad del sistema, el dominio del problema, y el espacio de la solución. La figura 3.3 muestra una forma típica de layering en un sistema.

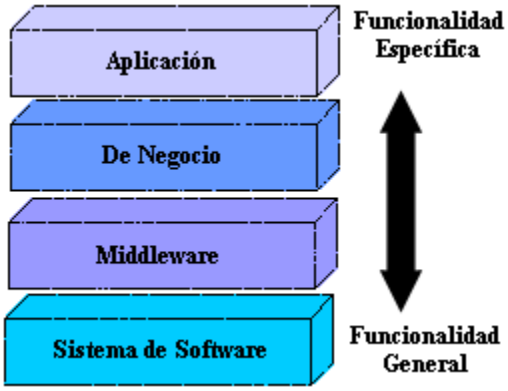

Figura 3.3. Layering.

Los siguientes puntos muestran la descripción de los elementos de cada capa.

- **Capa de aplicación:** Contiene los servicios específicos del sistema que se está desarrollando.
- **Capa de negocio:** Contiene un número de subsistemas reusables que son específicos al dominio en el que se desarrolla el sistema

[1] Los subsistemas sirven para agrupar elementos de un modelo, por lo general agrupan clases u otros subsistemas, estos sirven para administrar la complejidad del sistema, debido al gran número de clases que puede existir en un sistema.

- **Middleware:** Son subsistemas que ofrecen clases de utilidad y servicios de plataforma independiente
- **Sistema de software:** Contiene el software para la infraestructura actual tal como sistemas operativos, interfaces a software especifico, drivers de dispositivos, etc.

3.2.4.1 Guías para hacer el layering

El layering realiza un particionamiénto lógico de los subsistemas existentes en un determinado número de conjuntos o capas.

Este patrón arquitectónico también provee una forma de restringir las dependencias entre subsistemas para que el sistema esté mejor acoplado y sea más fácil de mantener, los siguientes puntos muestran los aspectos para la restricción de dependencias.

- **Visibilidad:** Los subsistemas pueden depender solamente de subsistemas que se encuentran en la misma capa o en la capa inferior siguiente.
- **Volatilidad:** Los siguientes tres puntos muestran los aspectos para la restricción de dependencias que tienen que ver con la volatilidad, es decir, la tendencia de un elemento a cambiar de lugar en el sistema.
    - En las capas más altas se deben poner elementos que se modifican cuando cambian los requerimientos del usuario.
    - En las capas bajas, se deben poner los elementos que se modifican cuando cambia la plataforma de implementación (Hardware, lenguaje, Sistema Operativo, base de datos, etc.).
    - En las capas intermedias se deben poner los elementos que son aplicables a un amplio rango de sistemas y ambientes de implementación.
- **Numero de capas:** Para un sistema pequeño, son suficientes 3 capas, si el sistema es complejo, generalmente se utilizan de 5 a 7 capas. La tabla 3.1 muestra un aproximado de capas conforme al número de clases del sistema.

Numero de Clases	Numero de Capas
0-10	No se necesita Layering
10-50	2 capas
25-150	3 capas
100-1000	4 capas

Tabla 3.1: Mapeo aprox. del numero de clases y de capas

3.2.4.2 Representación de las capas en UML

Antes de explicar la representación de las capas de un sistema en UML, se debe definir el término estereotipo y paquete:

"Los estereotipos son mecanismos para clasificar elementos y para introducir nuevos tipos de elementos en un modelo" [25]

"Un estereotipo es una extensión a la notación básica de UML que permite definir un nuevo elemento del modelo basado en un elemento de modelo estándar. La esencia de un estereotipo es que permite particularizar un elemento de modelo." [24]

En UML se pueden estereotipar las clases o los paquetes, para esto se pone el nombre del estereotipo entre los siguientes signos "<<>>" y abajo se coloca el nombre de la clase.

Un paquete es un mecanismo de propósito general para organizar elementos en grupos, en UML un paquete se representa por medio de un fólder (ver figura 3.4)

Las capas se representan en UML como paquetes, con el estereotipo de <<Layer>>. Debido a que una capa puede ser modelada como un paquete, esta puede aparecer en un diagrama de clases o un diagrama de caso de uso para mostrar las relaciones de las capas entre si. En UML los paquetes pueden estar relacionados unos con otros por medio de relaciones de dependencia como se indica en la figura 3.4. Estas relaciones entre paquetes pueden tener varias implicaciones, tal como que los cambios a un paquete origen pueden afectar al paquete cliente o que el paquete cliente no puede ser reusado independientemente pues el paquete cliente depende del paquete origen.

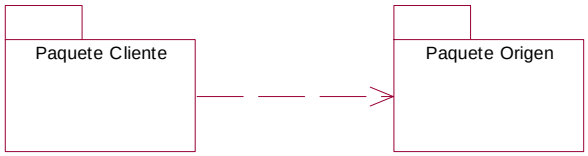

Figura 3.4. Relación de dependencia entre paquetes.

3.2.5 Clases de análisis

"Una clase es una colección de objetos con estructura, comportamiento, relaciones y semántica comunes" [19]. Una clase es una abstracción del mundo real, lo que quiere decir es que representa solo las cosas relevantes y suprime las que no tienen tanta importancia.

Una clase se representa en UML con un rectángulo dividido en tres partes; en la primera parte se coloca el nombre de la clase, en la segunda se colocan sus atributos (estructura) y

en la tercera se colocan las operaciones o el comportamiento, tal y como se indica en la figura 3.5.

Una clase de análisis es una “clase prototipo”, es decir, una clase candidata para llegar al modelo de diseño. “El primer paso en la transformación de los casos de uso a la descripción de cómo trabaja el sistema es justamente la búsqueda de un conjunto candidato de clases. Todas las clases de análisis juntas representan el primer modelo conceptual del sistema y son conjeturas tempranas de la composición del sistema” [24]

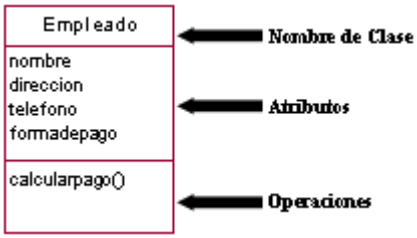

Figura 3.5. Representación de una clase en UML.

El modelo conceptual tendrá que evolucionar con el tiempo y se debe de tomar la decisión de seguirlo actualizando para seguir siendo un documento formal con las implicaciones en costos que esto conlleva. Mas adelante, las clases de análisis cambian en el diseño pudiendo llegar a ser subsistemas completos.

Las clases de análisis se pueden clasificar en tres, de acuerdo al rol que juegan en el sistema, estas clases de análisis son las siguientes: clases de interfaz (boundary), de control (control) y de entidad (entity); las tres son clases estereotipadas en UML y se coloca el tipo de clase entre los signos “<<>>” para identificarlas como clases estereotipadas. Otra forma de representación de clases de análisis es por medio de sus iconos característicos, las diferentes formas de cada clase de análisis se muestran en la figura 3.6.

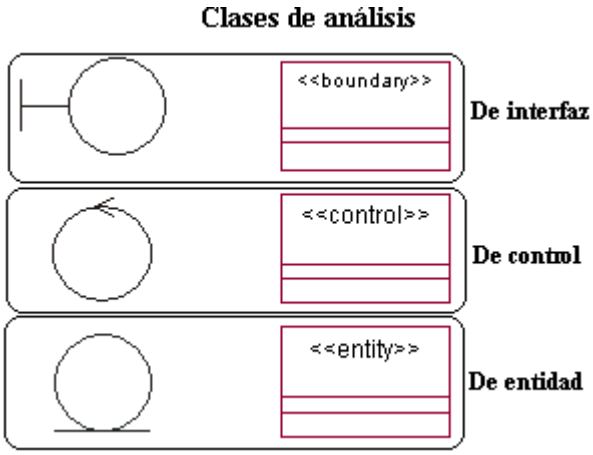

Figura 3.6. Representación de clases de análisis.

3.2.5.1 Clases de interfaz

"Las clases de interfaz se utilizan para modelar la interacción entre el sistema y sus actores… estas son partes del sistema que dependen de sus fronteras con el usuario" [13].

Para las clases de interfaz se capturan los siguientes aspectos:

- La entrada de información al sistema por medio de un actor
- La salida de información del sistema a un actor.
- La coordinación del comportamiento de los actores con la parte interna del sistema.

Las clases de interfaz pueden modelar las interfaces basadas en Windows, en este caso, el diseñador se debe concentrar en cuál va a ser la información que se presenta al usuario y no tanto en los detalles de la interfaz, los aspectos que se resaltan en esta clase son solo los que marca el caso de uso.

Si la clase de interfaz se refiere a un sistema o a un dispositivo, el enfoque se hace en los protocolos de la comunicación entre sistemas. De este párrafo y el anterior se saca una regla: "Concentrarse en las responsabilidades, no en los detalles" [24]. Las clases de interfaz hacen más fácil entender el sistema porque muestran las fronteras del mismo.

3.2.5.2 Clases de control

"Las clases de control representan coordinación, secuencia, transacciones y control de otros objetos" [13]. En otras palabras "Las clases de control coordinan el comportamiento del caso de uso" [24]

Las clases de control sirven también para realizar cálculos y coordinación de eventos que ocurren en el sistema. Debido a que las clases de control están en la parte interna del sistema, son independientes del ambiente en el sentido de que no interactúan con el usuario. Por lo general un objeto de una clase de control se destruye cuando el caso de uso al que encapsula el comportamiento acaba.

3.2.5.3 Clases de entidad

"Las clases de entidad se utilizan para modelar información que posee larga vida y que a menudo es persistente…estas suelen mostrar una estructura de datos lógica y contribuyen a comprender de qué información depende el sistema" [13]

Las clases de entidad modelan los conceptos principales del sistema y también son independientes del ambiente como las clases de control, la instancia de una determinada clase de entidad puede sobrevivir a varios casos de uso.

3.3 Análisis arquitectónico-definición de una arquitectura candidata

La primera actividad en el detalle de flujo de trabajo "Definición de una arquitectura candidata" es el complemento del análisis arquitectónico, se debe recordar que una actividad se puede realizar en muchas iteraciones o detalles de flujo de trabajo. Ahora el fin ya no es dar las bases para un estudio de factibilidad, sino definir una arquitectura candidata para el sistema, precisar los mecanismos principales y convenciones de modelado para el sistema

Un esfuerzo importante para definir la arquitectura es el layering en el que se particiona el sistema y se definen las dependencias entre las capas del software.

Los siguientes puntos muestran todos los pasos por orden de ejecución de la actividad "Análisis arquitectónico".

1. Desarrollo de un resumen de la arquitectura.
2. Búsqueda de activos de valor para el proyecto.
3. Definición de una organización de subsistemas de alto nivel (Solo aplica en este detalle de flujo de trabajo)
4. Identificación de las principales abstracciones
5. Desarrollo de un modelo de distribución de alto nivel
6. Identificación de mecanismos de análisis (Solo aplica en este detalle de flujo de trabajo)
7. Creación de las realizaciones de casos de uso (Solo aplica en este detalle de flujo de trabajo)
8. Identificación de interacciones entre abstracciones(NO aplica)
9. Revisión de resultados

Los pasos uno, dos y cinco no se desarrollan, porque ya se realizaron en el capítulo anterior, ni tampoco se desarrolla el punto ocho porque solo se solo se realiza como parte del detalle de flujo de trabajo "Realización de una síntesis arquitectónica".

Los pasos cuatro, cinco, y nueve se complementarán y se les dará el enfoque debido, los pasos tres, seis y siete se desarrollan completamente porque solo aplican a este detalle de flujo de trabajo.

3.3.1 Definición de una organización de subsistemas de alto nivel.

En este paso se ocupa el layering, es decir se descompone el sistema en capas, el layering es un patrón arquitectónico muy común para sistemas moderados y grandes. El número de capas no es específico, varía de situación a situación.

Durante el análisis arquitectónico, se debe hacer un enfoque a las dos capas más altas, que son la de aplicación y la de negocio. Lo anterior se logra por medio de una organización de subsistemas de alto nivel.

Ejemplo: La figura 3.7 muestra la organización de capas para el sistema que se plantea en el anexo A.

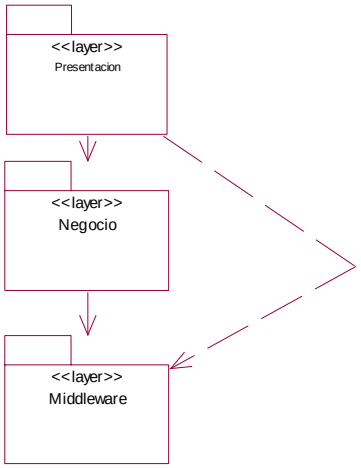

Figura 3.7. Capas arquitectónicas y sus dependencias.

A continuación se muestra la descripción de cada capa:

- **Presentación:** La capa de presentación contiene los elementos de diseño específicos de la aplicación.
- **Negocio:** La capa de negocio contiene los elementos específicos del negocio y que son usados en muchas aplicaciones
- **Middleware:** Provee utilidades y servicios independientes a plataforma

3.3.2 Identificación de las principales abstracciones.

Cuando se habla de abstracciones principales, se refiere a los conceptos primarios que debe manejar un sistema. Las fuentes para encontrar estas abstracciones incluyen: requerimientos, glosario, el modelo de dominio y el modelo de objetos de negocio.

Esta actividad provee un punto de inicio para el análisis de casos de uso, porque ya que se han identificado y modelado las abstracciones principales, se reduce la posibilidad de que diferentes términos en el modelo se refieran a las mismas abstracciones cuando las clases hayan sido identificadas para los diferentes casos de uso.

No se debe gastar mucho tiempo en definir las relaciones entre las abstracciones, esta tarea realiza en etapas posteriores.

Ejemplo: La figura 3.8 muestra las principales abstracciones del sistema de pago de nomina que se plantea en el anexo A.

Figura 3.8. Principales abstracciones.

- **SistemaDeBanco:** El Sistema externo al cual se mandan todos los depósitos de transacciones.
- **Empleado:** Un empleado que es pagado por hora
- **VerificadorDePagos:** Un registro de cuánto se le ha pagado a un empleado para un periodo dado
- **OrdenDeCompra:** Un registro de una venta hecha por un empleado
- **BDAdministracionDeProyecto:** Una BD heredada que contiene información de los proyectos y cambios de números
- **TarjetaDeRegistro:** Un tarjeta que contiene información de las horas trabajadas por un empleado para un periodo.

3.3.3 Desarrollo de un modelo de distribución de alto nivel

En este paso se desarrolla un modelo de distribución de alto nivel, el cual es un conjunto de nodos y conexiones que representan la repartición de los elementos físicos del sistema

en el ambiente de implementación. Cuando a este modelo se le agrega el calificativo de "alto nivel" significa que no tiene muchos detalles, solo los elementos más importantes.

El siguiente ejemplo muestra los nodos y conexiones que tendría el sistema de pago de nomina que se presenta en el anexo A.

Nodos y conexiones

- PCs Desktop: Estas son conectadas al servidor de nomina vía LAN o internet
- Servidor de Nomina: Este servidor esta conectado al sistema externo de Banco vía internet. El servidor de nomina también esta conectado a la BD de Administración de Proyecto vía LAN.
- Sistema de Banco.
- BD de Administración de Proyecto.
- Impresoras (Dispositivo): Las impresoras están conectadas al servidor de nomina vía LAN.

3.3.4 Identificación de mecanismos de análisis

El propósito de este paso es definir los mecanismos de análisis y los servicios usados por los diseñadores para darle "vida" a sus objetos.

Los mecanismos de análisis se pueden identificar por medio de la experiencia que guía al arquitecto de software para conocer los problemas que están presentes en el dominio, y que requieren cierto tipo de solución. Ejemplos de problemas arquitectónicos que pueden ser expresados como mecanismos son: persistencia, administración de transacciones, tratamiento de fallos, mensajes y motores de inferencia.

El aspecto común de cada uno de los conceptos anteriores es que tienen capacidad general para una amplia gama de sistemas y cada uno provee servicios que soportan la funcionalidad de la aplicación.

El mecanismo de análisis se crea cuando el arquitecto de software ve que un concepto común va emergiendo de un conjunto de soluciones a varios problemas. Por ejemplo que se necesita proveer una forma común para sincronizar los relojes de sus hilos de ejecución o que se necesita una forma común de destinar los recursos. De estos patrones emergen los mecanismos para simplificar el lenguaje del análisis.

Ya que se han identificado los principales mecanismos de análisis, se debe regresar a las principales abstracciones y hacer un mapeo de estas con sus mecanismos correspondientes.

La tabla 3.2 muestra los mecanismos de análisis mapeados a las principales abstracciones del ejemplo del sistema de nomina.

Abstracción	Mecanismo de Análisis
SistemaDeBanco	Interfaz Heredada
Empleado	Persistencia, Seguridad
VerificadorDePagos	Persistencia, Seguridad
OrdenDeCompra	Persistencia, Seguridad
BDAdministracionDeProyecto	Interfaz Heredada
TarjetaDeRegistro	Persistencia, Seguridad

Tabla 3.2 Mapeo de clases de análisis a mecanismos de análisis

3.3.5 Creación de las realizaciones de casos de uso

El propósito de éste punto es crear los artefactos del modelo de diseño que se usan para expresar el comportamiento de los casos de uso. En este paso solo se crea el artefacto, posteriormente en otros capítulos se desarrollará y se mantendrá para permanecer íntegro.

“Los casos de uso forman el foco central para la mayoría del trabajo inicial en el análisis y diseño. Sirven para permitir la transición de las actividades que se centran en los requerimientos a las actividades que se centran en el diseño. Las realizaciones sirven como un puente, para trazar (rastrear) el comportamiento del modelo de diseño al modelo de caso de uso. ” [22]

Anteriormente, en la disciplina de requerimientos, el arquitecto de software realizó la actividad “Priorizar casos de uso”, ésta actividad se realiza sobre el documento de arquitectura de software y tiene como fin hacer un subconjunto de todos los casos de uso existentes que tienen como característica el ser “arquitectónicamente significativos”, estos casos de uso se plasman en la vista de caso de uso del mismo artefacto.

Se debe crear una realización de caso de uso en el modelo de diseño por cada caso de uso identificado en la actividad “Priorizar casos de uso”. El nombre para la realización de caso de uso debe ser el mismo que el caso de uso que esta realizando y una relación del tipo “realiza” se debe de establecer entre la realización de caso de uso y el caso de uso asociado.

3.3.6 Revisión de resultados

En este paso se revisa que los resultados del análisis arquitectónico son completos y consistentes. Lo anterior se puede hacer por medio de los siguientes cuestionamientos:

- ¿Se han identificado los mecanismos de análisis necesarios?

- ¿Se han identificado las principales clases de entidad y sus relaciones principales?
- ¿El nombre de las clases o abstracciones principales y sus relaciones son consistentes y claras con el rol que juegan en el sistema?
- ¿Las clases o abstracciones principales y sus relaciones son consistentes con el modelo de negocio, el modelo de dominio, los requerimientos y el glosario?

3.4 Análisis de caso de uso

La actividad “Análisis de caso de uso” se realiza en dos partes complementarias, la primera se ejecuta en la definición de una arquitectura candidata que es el tema de este capítulo, la segunda parte de realiza en el detalle de flujo de trabajo “Análisis del comportamiento de los casos de uso”, que es el tema del siguiente capítulo.

El objetivo de la actividad “Análisis de caso de uso” en esta fase del análisis y diseño es:

- Suplementar los casos de uso para la mejor comprensión de los desarrolladores
- Identificar las clases que realizarán el flujo de eventos del caso de uso.
- Distribuir el comportamiento de los caso de uso a las clases identificadas en el punto anterior, usando las realizaciones de caso de uso.

Esta actividad se hace para cada realización de caso de uso creada anteriormente.

3.4.1 Complemento de las descripciones de casos de uso.

El primer paso es aclarar los puntos ambiguos o vagos que pudieran quedar de los requerimientos de los C.U. y acercar el lenguaje de los casos de uso a para la mejor comprensión del diseñador.

El que los casos de uso se escriban en lenguaje común tiene ventajas y desventajas, algunas de las desventajas es que la mayoría de las veces la información del caso de uso no es idónea para realizar las actividades de análisis y diseño, particularmente la actividad de análisis de caso de uso. Por lo anterior, se debe suplementar el flujo de los casos de uso para “entender el comportamiento interno del sistema, que puede estar perdido en la descripción del caso de uso escrito para el cliente” [24]

Cabe señalar que **los requerimientos no se deben de cambiar, tan solo suplementar**, enfocándose en mejorar el flujo del caso de uso con detalles internos de ejecución. Lo anterior quiere decir que no se debe de cambiar el orden del flujo de eventos y tampoco

agregar o eliminar pasos del flujo del caso de uso, porque en este caso si se estarían cambiando los requerimientos.

Por ejemplo:

Un paso en el flujo de eventos de un caso de uso podría ser el siguiente:

“El sistema muestra una lista de todas las materias que se encuentran disponibles para el alumno…”

Suplementando la descripción anterior quedaría:

“El sistema recupera y muestra una lista de las materias que se encuentran disponibles de la B.D. de catalogo de materias”.

3.4.2 Búsqueda de las clases que darán comportamiento al caso de uso

En este paso se deben encontrar clases que den comportamiento al caso de uso que se está analizando actualmente, se debe recordar que se hace una instancia de esta actividad en cada realización de caso de uso. “El comportamiento que se identifica en el análisis de caso de uso principalmente expresa comportamiento en las capas más altas del sistema. Las clases de control y las clases de interfaz frecuentemente evolucionan en elementos de diseño de la capa de aplicación. Las clases de entidad se convierten en elementos de diseño de la capa de negocio. Las capas más bajas del sistema son pobladas por elementos de diseño que evolucionan de los mecanismos de análisis” [22]

Ahora solo se realiza la búsqueda de clases de análisis y posteriormente en el siguiente capítulo se le dará estructura y comportamiento a las clases, complementando esta misma actividad. Las clases de análisis a buscar se dividen en tres: de interfaz, de control y de entidad, dependiendo de las características de cada una se hace la búsqueda, los siguientes puntos muestran la estrategia de búsqueda para cada tipo.

3.4.2.1 Búsqueda de clases de interfaz

El propósito del análisis es el de comprender mejor los requerimientos y acercarlos al lenguaje del desarrollador, por lo tanto la búsqueda de clases de interfaz se debe de hacer en los casos de uso, buscando qué clases de interfaz se pudieran ver en el flujo de eventos.

“Se debe considerar la fuente de los eventos externos y asegurarse de que hay una forma para que el sistema los detecte” [24]

Una recomendación que se hace es modelar una clase de interfaz por un par de actor/caso de uso. A esta clase se le puede dar la responsabilidad de coordinar la interacción entre el actor y el caso de uso. Por ejemplo, en la figura 3.9 se presenta un ejemplo de caso de uso del sistema de nomina, el administrador desea correr la nomina y así pagarle a los empleados por medio de una transacción a una cuenta bancaria. En este caso se crean dos clases de análisis de interfaz: la clase "FormaCorrerNomina" y la clase "InterfazSistemaBancario"

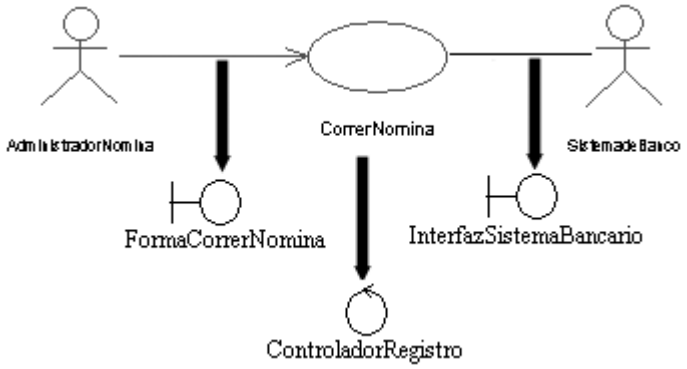

Figura 3.9. Ejemplo de búsqueda de clases de interfaz y de control.

3.4.2.2 Búsqueda de clases de control

Las clases de control sirven para coordinar a otros objetos y para ordenar el comportamiento del sistema, por lo general sus instancias se destruyen cuando acaba el caso de uso al que encapsula la clase. Por lo tanto se recomienda identificar una clase de control por caso de uso, a menos que este caso de uso tenga un comportamiento demasiado simple.

En la parte baja de la figura 3.9 se muestra una clase de control del caso de uso CorrerNomina llamada "ControladorRegistro", esta clase tiene la labor de administrar el proceso que implementa la funcionalidad para correr la nomina.

3.4.2.3 Búsqueda de clases de entidad

"Las clases de entidad administran y almacenan la información del sistema" [24]. Por lo general en esta información se encuentran los principales conceptos que maneja el sistema, así que algunas fuentes de información para encontrar estas clases son las siguientes: el glosario (requerimientos), el modelo de dominio del negocio (modelado del negocio) y también los casos de uso. Sobre estos últimos "la mejor manera de encontrar clases de

entidad candidatas es identificando y después filtrando los sustantivos de las descripciones de los casos de uso" [24]. Para filtrar estos sustantivos se deben de remover:

- Los candidatos redundantes
- Los candidatos que no son importantes para la solución del problema
- Los candidatos que no están bien definidos.
- Partes que se refieren a la implementación
- Sustantivos que se refieran a los atributos de una clase.

Antes de intentar que lo anterior sea una regla, los puntos son recomendaciones, por lo que también cuenta la experiencia del diseñador y la creatividad.

Capítulo 4

Análisis del comportamiento de los casos de uso

Tercera llamada…

Introducción

Hasta este punto, se ha hecho un esfuerzo inicial para especificar la arquitectura, es decir, se han definido las capas más altas de la arquitectura, las principales abstracciones y algunos mecanismos de análisis. Todo lo anterior, junto con los requerimientos del sistema sirven como entrada principal para el detalle de flujo de trabajo "Análisis de comportamiento de los casos de uso" en el cual se modela el sistema por medio del artefacto "modelo de análisis", la figura 4.1 muestra en qué parte de la disciplina de análisis y diseño se encuentra este detalle de flujo de trabajo.

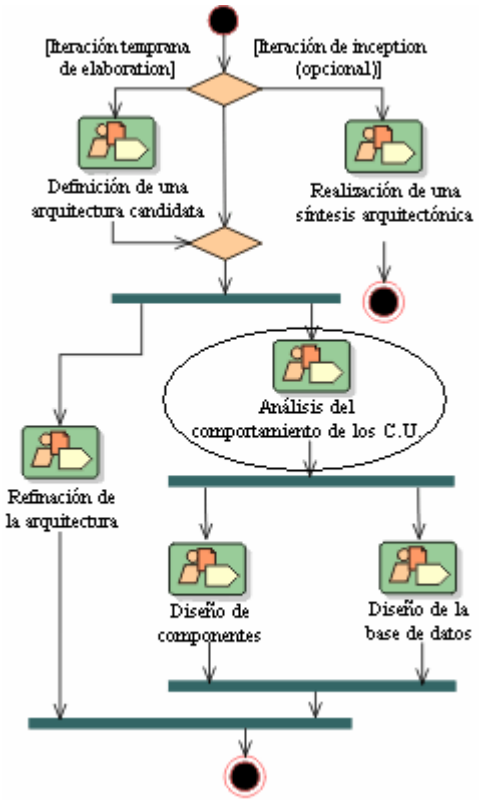

Figura 4.1. Detalle de flujo trabajo "Análisis del comportamiento de los C.U.".

"El modelo de análisis es el artefacto principal del detalle de flujo de trabajo en el que se analiza el comportamiento de los casos de uso. Este es un modelo de plataforma independiente, lo que significa que no contiene decisiones del tecnología especifica" [3]

Se realiza una instancia de la actividad por cada caso de uso y ya que se tienen los resultados del análisis de los casos de uso se procede con la identificación de los elementos de diseño. Por ultimo se hace una revisión del trabajo que se realizó en este detalle de flujo de trabajo.

La figura 4.2 muestra los artefactos, las actividades los roles y el proceso del detalle de flujo de trabajo “Análisis del comportamiento de los C.U.”

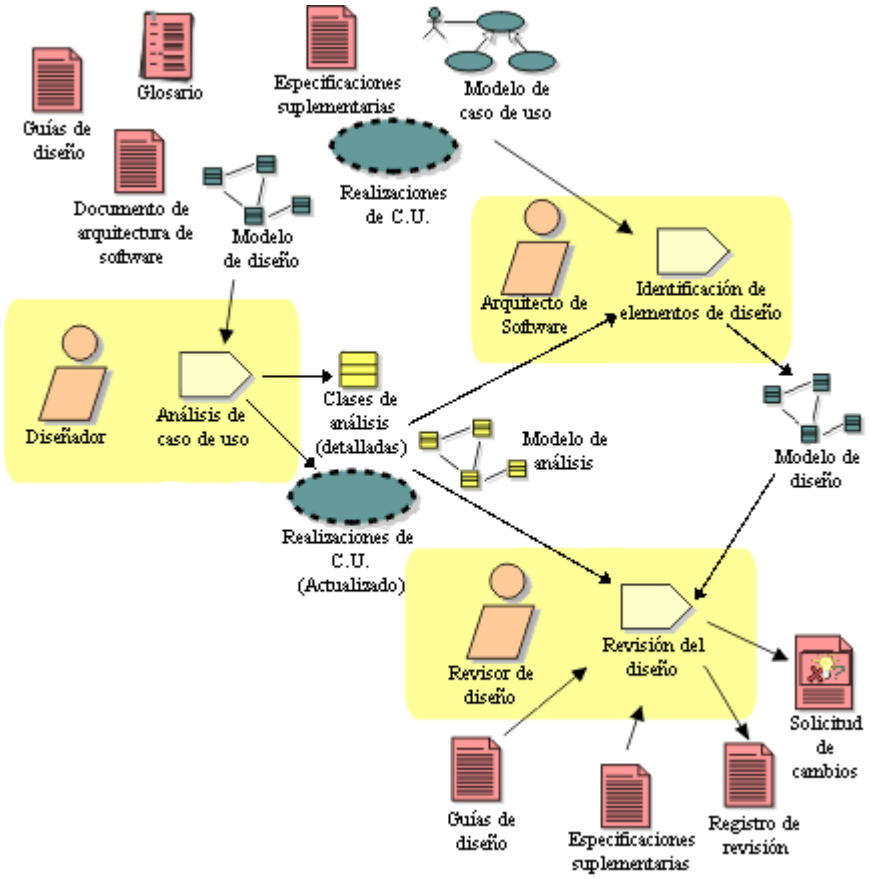

Figura 4.2. Actividades, Artefactos y Roles del detalle de flujo de trabajo “Análisis del comportamiento de los C.U.”

4.1 Propósito del detalle de flujo de trabajo “Análisis del comportamiento de los casos de uso”

El propósito del detalle de flujo de trabajo del análisis de los casos de uso es transformar las descripciones de comportamiento descritas en los casos de uso en un conjunto de elementos sobre los que se pueda basar el diseño del sistema

4.2 Conceptos preliminares

Los siguientes puntos muestran los conceptos preliminares que son necesarios para comprender mejor el capítulo, entre los que están los diagramas de secuencia y de colaboración, los subsistemas, el acoplamiento y la cohesión del sistema.

4.2.1 Diagrama de secuencia

Los diagramas de secuencia y de colaboración son usados dentro de las realizaciones de caso de uso para modelar las interacciones entre los actores y el sistema, por ello se le llaman a ambos diagramas de interacción.

"Un diagrama de secuencia muestra un patrón de interacción ordenada de objetos en un orden cronológico" [24]. Otro concepto dice que "los diagramas de secuencia se usan a lo largo del proceso de análisis y diseño para demostrar las interacciones internas entre actores y objetos dentro del sistema a través del tiempo" [16]

La idea anterior aporta una de las características mas importantes de lo que muestra el diagrama de secuencia: "la interacción en un orden cronológico y a un tiempo preciso".

Los objetos en un diagrama de secuencia se comunican mediante mensajes, los cuales son un mecanismo para transmitir información de un objeto a otro. Los objetos por sí solos no tienen ninguna funcionalidad, sino que se tienen que comunicar entre ellos para realizar algún comportamiento esperado.

La figura 4.3 muestra la anatomía de un diagrama de secuencia

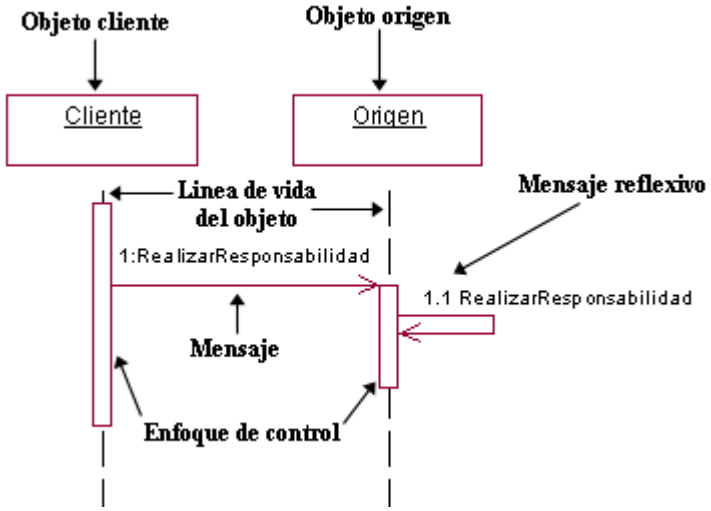

Figura 4.3. Diagrama de secuencia.

A continuación se explica la semántica y sintaxis de cada parte de este diagrama:

- **Un objeto** se representa mediante un rectángulo y en el interior se coloca el nombre del objeto, abajo se coloca una línea punteada a la cual se le llama "línea de vida". La línea de vida de objeto representa la existencia de un objeto en un tiempo determinado.

- **Un mensaje** es una comunicación entre objetos que transmiten información, se representa mediante una flecha que inicia del objeto que quiere mandar el mensaje hacia el objeto destino. En un diagrama de secuencia puede haber mensajes reflexivos en los cuales la flecha inicia y termina en el mismo objeto.
- **El enfoque de control** representa el tiempo en el que el flujo se enfoca en un objeto. Éste se representa mediante un rectángulo angosto sobre la línea de vida del objeto.

4.2.2 Diagrama de colaboración

Un diagrama de colaboración describe las interacciones entre objetos y sus asociaciones. De la misma manera que los diagramas de secuencia, los diagramas de colaboración realizan interacciones entre objetos, solo que en este caso estas interacciones no se dan en un tiempo definido ni en un orden cronológico directo.

En un diagrama de colaboración para que un objeto le mande un mensaje a otro, tiene que haber un "enlace" entre los dos, lo que significa que los objetos están asociados y por lo tanto que se pueden mandar mensajes entre ellos.

La figura 4.4 muestra las partes de un diagrama de colaboración, en donde el objeto cliente le manda el mensaje "RealizarResponsabilidad" al objeto destino, en esta figura se indica que el mensaje tiene un número, por medio del cual se puede determinar la secuencia del mensaje en toda la interacción.

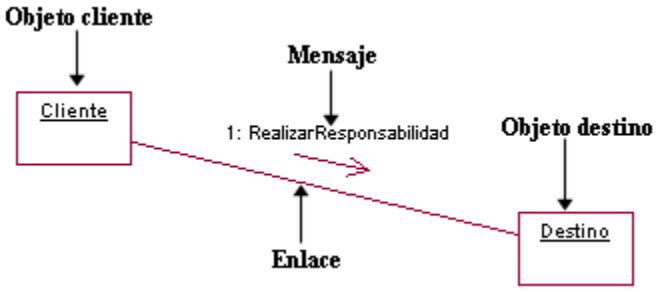

Figura 4.4. Diagrama de colaboración.

4.2.3 Subsistemas

El concepto de subsistema se encuentra entre la idea de un paquete y el de una clase. Los subsistemas se comportan como paquetes porque agrupan a otros elementos de modelo. En cambio se parecen a las clases porque los subsistemas tienen comportamiento.

"Un subsistema tiene una vista interna y una vista externa. Internamente el subsistema es un conjunto de otros elementos de modelo que colaboran para cumplir el comportamiento del que es responsable el subsistema. Externamente no necesariamente son visibles estos elementos y es una buena idea, ya que el subsistema se trata como una unidad." [9]

El comportamiento en un subsistema se muestra por medio de sus interfaces externas al subsistema. En este sentido difiere el concepto de subsistema con el de paquete y se asemeja con el de la clase porque cuando se requiere un comportamiento, el subsistema "cliente" hace el requerimiento al subsistema "servidor", sin embargo, el cliente no tiene idea de cómo trabaja internamente el servidor, tal como una clase. En el caso de un paquete, la petición de comportamiento se hace directamente a las partes internas que agrupa.

En UML un subsistema se representa por medio de un paquete estereotipado con la leyenda "<<subsystem>>". Una convención (forma) útil es nombrar las interfaces empezando con la letra mayúscula "I".

"El objetivo principal de un subsistema es encapsular el comportamiento. Cuando el subsistema cliente solicita que otro subsistema haga un servicio, no debe estar conectado directamente con la implementación de tal servicio. En vez de eso, los clientes deben acceder al subsistema a través de una interfaz" [9]

Para la representación de las interfaces del subsistema existen dos formas, tal y como se indica en la figura 4.5. En la forma canónica, la interfaz se ve como una clase estereotipada con la leyenda "<<Interface>>", y la relación de realización entre el subsistema y su interface se representa como una flecha de realización de caso de uso. En la forma abreviada la interface se muestra como un icono y la relación de realización se muestra con una línea sólida.

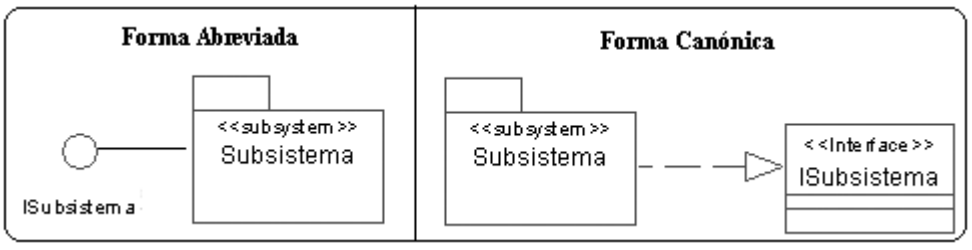

Figura 4.5. Representación de subsistemas e interfaces.

4.2.4 Conceptos de acoplamiento y cohesión de paquetes

En esta sección se presentan dos aspectos muy importantes que tienen que ver con la forma en que se unen los elementos en un sistema. Estos conceptos son el acoplamiento y la cohesión.

El **acoplamiento** se refiere a qué tan fuerte o que nivel de relación hay entre dos elementos de diseño (paquetes, clases, objetos,…). La meta en el análisis y diseño es lograr un "acoplamiento holgado".

El acoplamiento holgado es una relación entre dos elementos que sea acompleja, directa, visible y flexible.

Lo anterior se puede ilustrar por medio de un tren que tiene varios vagones. Estos vagones están diseñados de tal forma que es fácil conectarlos y una vez que se conectan no se desprenden, a su vez, la unión de los vagones no hace que el tren esté rígido, sino que en una curva estas uniones de vagones permiten la flexibilidad en el movimiento.

El ejemplo anterior bien se podría aplicar a la conexión de elementos, esta debería ser lo más simple posible, y también debería ser sólida y flexible.

La **cohesión** entre los elementos (paquetes, clases, objetos,…) de un sistema se refiere al nivel de distribución de las responsabilidades en el modelo. Un modelo con clases perfectamente cohesivas es aquel en el que cada clase realiza una sola responsabilidad. La meta es que una clase haga una cosa y que la haga bien, en vez de hacer muchas pero mal.

4.2.5 Concurrencia y sistemas concurrentes

Concurrencia es la tendencia de que en un sistema las cosas pasen al mismo tiempo, esto es un fenómeno normal en el mundo real, en cualquier tiempo dado, muchas cosas están ocurriendo simultáneamente.

Cuando se hace frente a los problemas que genera la concurrencia en los sistemas, hay dos aspectos importantes: detectar y responder a eventos externos ocurriendo en un orden aleatorio y asegurarse que esos eventos se responden de manera idónea.

Si las actividades se desarrollan independientes unas de otras, la concurrencia es fácil de tratar, tal y como en una autopista de dos carriles, en donde los autos viajan en direcciones opuestas y los autos no chocan porque cada uno viaja en un sentido.

Cuando las actividades concurrentes interactúan, se requiere algún tipo de coordinación, estos son los problemas reales de la concurrencia.

Diseñar un sistema procedural convencional para tratar con estas situaciones es extremadamente complejo. Puede ser mucho más simple particionar el sistema en elementos de software concurrentes para tratar con cada uno de los eventos.

A fin de diseñar sistemas concurrentes de manera efectiva primero se debe pensar en el papel que juega la concurrencia en el sistema, y después abstraerla.

La concurrencia trata con actividades que se realizan al mismo tiempo y los bloques de construcción de un programa son procedimientos y estructuras de datos, estos son inadecuados para pensar en la concurrencia.

Razonando sobre sistemas de procedimientos, un procesador ejecuta un procedimiento y este sigue una ruta dependiendo de diversas condiciones. Esta ruta se le llama “hilo de ejecución” o “hilo de control” el cual puede tomar diferentes ramas o ciclos dependiendo de las condiciones en un tiempo dado.

Lo anterior lleva a la idea que desde el punto de vista del diseñador, un hilo de ejecución se controla por la lógica del programa y éste es ordenado por el sistema operativo. Cuando el diseñador selecciona hacer un procedimiento que invoque otro, el hilo de control cambia a otro procedimiento y regresa al que lo llamo cuando este acaba. Desde el punto de vista del CPU solo hay un hilo principal que se “teje” a través del software, suplementado por medio de hilos control en respuesta a interrupciones.

Como se puede pensar, hay varios niveles de abstracción cuando se trata con el problema de la concurrencia, y el programador, analista y diseñador deben ver diversas abstracciones según sea la complejidad del sistema.

4.2.5.1 Mecanismos para lograr la concurrencia

Hay muchas maneras de implementar la concurrencia, pero por regla, el sistema debe administrar múltiples hilos de control para lograr esta implementación. Los siguientes puntos muestran los principales mecanismos de implementación.

- **Multiprocesamiento:** Varios CPUs corriendo concurrentemente
- **Multitarea:** El S.O. simula la concurrencia en un CPU individual
- **Soluciones basadas en la aplicación:** En este caso la aplicación es la que toma la responsabilidad para cambiar entre los diferentes hilos de control en los momentos indicados.

4.2.5.2 Comunicación sincrónica y asincrónica

Los hilos o procesos a menudo se deben comunicar entre ellos y esto trae algunos puntos importantes que tratar. Entre los más importantes es la forma en que se comunican y en este sentido hay dos formas: comunicación asincrónica y comunicación sincrónica.

- **Comunicación asincrónica:** La actividad que manda la información no se preocupa si la actividad receptora esta lista o no, simplemente manda la información y sigue con sus tareas. En esta forma de comunicación, la actividad que recibe, debe tener implementos para poder guardar la información o requerimiento y contestarlo cuando esté lista.
- **Comunicación sincrónica:** Este tipo de interacción requiere de sincronización entre el emisor y el receptor. Un problema en la comunicación sincrónica es que cuando una actividad está esperando que la otra esté lista para recibir información, no responde a ningún evento, es decir, esta bloqueada.

4.2.5.3 El modelo de objetos activos

Para diseñar un sistema de software concurrente, se deben combinar los bloques de software (procedimientos y estructuras de datos) con los bloques de construcción de la concurrencia (hilos de control). Para la anterior tarea el objeto es el mas idóneo y natural, ya que en si mismo empaca procedimientos y estructuras de datos (atributos y operaciones), solo faltaría implementar la concurrencia y esto se podría realizar manualmente.

Sin embargo la concurrencia introduce trabajo extra y por lo tanto mayores oportunidades de error. Al ser un problema complicado de resolver, en ocasiones se obscurece la esencia del problema de aplicación. Por las razones anteriores, es necesario minimizar la necesidad de los programadores de tratar con ella explícitamente.

Una forma de resolver el problema de la concurrencia más eficientemente es construir un ambiente orientado a objetos con soporte de paso de mensajes entre objetos concurrentes.

A los objetos con sus propios hilos de control se les llaman “objetos activos”. A fin de soportar la comunicación asincrónica con otros objetos activos, cada objeto activo tiene una cola de mensajes o “buzón”. Cuando un objeto de este tipo se crea, el ambiente le da su propio hilo de control, que el objeto encapsula hasta que muere.

Un objeto activo como cualquiera permanece ocioso hasta que le llega un mensaje. Cualquier mensaje que le llega al objeto cuando está ocupado se almacena en el buzón hasta que el objeto este listo. Esta capacidad se le debe dar solo a los objetos que necesiten características de concurrencia y de esta manera se crea un ambiente mixto de objetos pasivos y activos.

4.3 Análisis de caso de uso –análisis de comportamiento de los casos de uso

Los siguientes dos puntos explican el propósito de la actividad "Análisis de caso de uso":

- Identificar las responsabilidades, atributos y asociaciones de las clases de análisis usando el artefacto "realización de caso de uso".
- Avanzar hacia la implementación de mecanismos arquitectónicos

De forma mas abreviada "la meta del análisis de caso de uso es tomar los requerimientos de los casos de uso del sistema e iterativamente transformarlos en representaciones que soporten los conceptos del negocio, y hagan frente a los requerimientos. En el diseño de casos de uso, estos conceptos de negocio se transforman en clases, objetos, relaciones, componentes, interfaces, etc., que pueden implementarse en un ambiente de ejecución" [7]

El análisis de caso de uso no se ha terminado, en el capítulo anterior solo se inició la búsqueda de clases de análisis, queda por distribuirles el comportamiento a través de los diagramas de secuencia y de colaboración que se encuentran dentro de cada una de las realizaciones de caso de uso.

Ya que se han identificado todas las clases de análisis, se describen las responsabilidades y atributos de las mismas, además de establecer asociaciones y dependencias en un diagrama de clases que será una vista conceptual del modelo de diseño. Después se deben de identificar los mecanismos de análisis que serán usados por las clases

4.3.1 Distribución del comportamiento a las clases de análisis

La primera tarea de la actividad "Análisis de caso de uso" es la distribución del comportamiento a las clases que se han identificado. Las instancias por si solas no sirven de mucho, deben de colaborar para poder realizar la funcionalidad del sistema, esto se hace

por medio de los diagramas de interacción que se detallan en las realizaciones de casos de uso.

Este paso inicia analizando los flujos principales y alternativos de cada caso de uso, después se identifican las clases que son responsables para dar el comportamiento requerido y se realizan los diagramas de interacción necesarios para modelar la interacción del sistema con sus actores por medio del paso de mensajes entre objetos.

Los diagramas deben de iniciar con un actor, ya que estos son los que siempre invocan los casos de uso. El número de diagramas de secuencia y colaboración depende del número de flujos alternos, del nivel al que se requiera detallar, de la dificultad del flujo de trabajo etc. Se recomienda iniciar describiendo los flujos principales y seguir con los flujos alternativos más importantes del caso de uso.

Una forma de determinar el comportamiento de las clases de análisis que se han identificado es aprovechando la información que dan los estereotipos. Las clases de interfaz deben tener el comportamiento que involucre la comunicación del sistema con el actor. Las clases de entidad tienen el comportamiento que involucra los datos que se encapsulan en la abstracción. Las clases de control tienen el comportamiento específico a un caso de uso o parte de un flujo de eventos que tenga que ver con cálculos o con partes criticas de control.

4.3.2 Descripción de responsabilidades en las clases de análisis

Hasta este punto se hizo un enfoque en los flujos de eventos de los casos de uso, ahora el foco cambia hacia las clases de análisis.

Antes de empezar a ponerle atributos y operaciones a las clases se debe de determinar las responsabilidades de cada una de ellas. Una responsabilidad es “un contrato u obligación de la clase” [24], es una oración que describe el comportamiento que debe tener, este comportamiento se realiza por medio de atributos y operaciones que se le asignan a la clase de acuerdo a la responsabilidad que esta tenga.

Una clase se puede ocupar de dar comportamiento a uno o más casos de uso, por lo que la misma puede tener varias responsabilidades. Las responsabilidades se derivan de los mensajes de los diagramas de interacción, mayormente los diagramas de colaboración pues estos muestran patrones de colaboración.

Las responsabilidades se escriben con un prefijo de doble diagonal para aclarar que aun no son operaciones. La figura 4.6 muestra el diagrama de colaboración de un caso de uso

para registrar su nombre y password en un sistema. De este diagrama se desprende la clase LoginForm con sus responsabilidades.

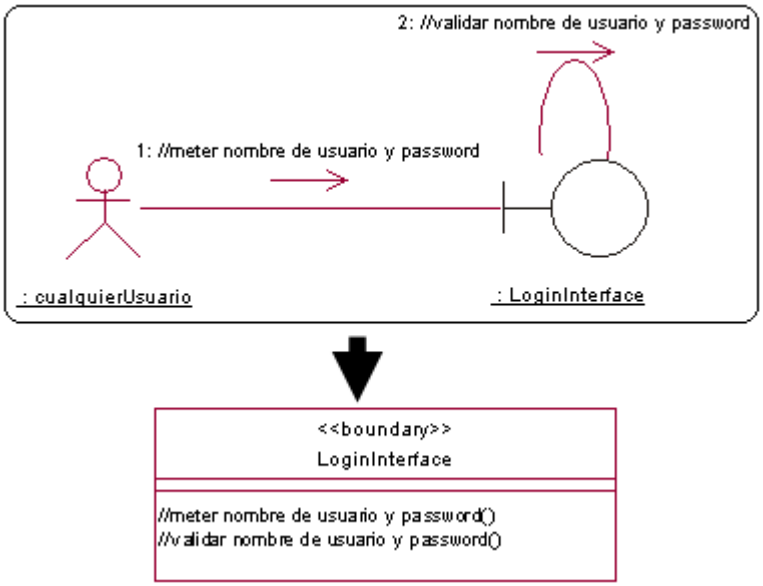

Figura 4.6. Ejemplo de descripción de responsabilidades en una clase.

4.3.3 Definición de atributos

En este paso se le agregan los atributos a las clases, un atributo se define como: "descripciones de características estructurales o estáticas que se usan para modelar la información asociada con una clase" [25]

El nombre de un atributo debe ser algún sustantivo que describa claramente cuál es la información que guarda, cada atributo debe tener tiene su tipo y opcionalmente su valor de inicio como se indica en la figura 4.7. En la etapa de análisis, el tipo de atributo debe ser general o primitivo, es decir, no debe ser específico a ningún lenguaje de programación.

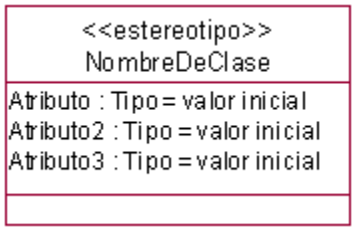

Figura 4.7. Representación de atributos en una clase.

Las fuentes para encontrar los atributos de las clases son las siguientes:

- Experiencia en el dominio del sistema
- Requerimientos
- Glosario

- Modelo de dominio
- Modelo de negocios

4.3.4 Establecimiento de asociaciones entre clases de análisis

El siguiente paso en la definición de las clases de análisis es la búsqueda de asociaciones entre las clases.

Una asociación en este contexto es: “una relación semántica entre dos o más clases que especifica conexiones entre sus instancias” [2]. Las asociaciones son la forma en la que se especifican las relaciones entre clases en UML. Estas se representan por medio de una línea conectando las dos clases asociadas, de esta forma los datos pueden fluir en ambas direcciones.

Un enlace es una instancia de una asociación, así como un objeto es una instancia de una clase. Los mensajes en un diagrama de colaboración pueden viajar por medio de los enlaces.

Los conceptos del párrafo anterior son muy útiles, ya que para encontrar asociaciones entre clases basta con analizar los diagramas de colaboración; los enlaces entre dos clases indican que los objetos de las mismas se necesitan comunicar entre si para realizar el caso de uso. De esta forma se puede encontrar la siguiente guía para encontrar asociaciones: “Representar una asociación en el diagrama de clases por cada enlace del diagrama de colaboración”. A las asociaciones se les puede agregar más información que solo una línea de conexión, los siguientes puntos muestran formas de especificación suplementaria de asociaciones.

4.3.4.1 Definición de roles

Para especificar una asociación se puede definir el rol de cada clase en la asociación, es decir, qué papel juega cada clase que participa en la relación. El nombre del rol debe ser un sustantivo que indique el papel del objeto en el contexto de la asociación con el otro objeto asociado.

La figura 4.8 muestra que existe una asociación entre las clases “Vuelo” y “Avión”. En cada lado de la asociación cada clase tiene su rol, por ejemplo, del lado del avión; la clase juega el rol de un avión asignado a un determinado vuelo.

4.3.4.2 Definición de multiplicidad

La multiplicidad "especifica cuántos objetos de una clase están asociados con un objeto de la otra clase en la asociación" [25]. Al igual que en los roles se puede especificar la multiplicidad en cada lado de la asociación, ésta se representa por medio de uno o varios rangos separados por comas. El símbolo "*" indica un numero ilimitado de objetos. Dos números separados por los símbolos ".." significa un rango continuo entre el primer y segundo valor. De esta forma, "0,2..4,7" indica un rango que estaría en el siguiente conjunto {0,2,3,4,7}.

La figura 4.8 muestra que para cada vuelo puede haber 0 o 1 objetos de la clase avión asignados y que para cada avión puede haber de 0 hasta un número ilimitado de objetos de vuelo asignados. De los puntos anteriores se puede desprender que la multiplicidad especifica información sobre si la asociación es obligatoria u opcional y cuál es el número mínimo y máximo de instancias que pueden ser ligadas a una instancia.

Figura 4.8. Ejemplo de multiplicidad y roles en asociaciones.

4.3.4.3 Definición de asociaciones de agregación

La agregación es una forma especial de asociación, es una relación más "fuerte" que la asociación común, esta sirve para modelar un todo y sus partes, también son conocidas como asociaciones "tiene un". Por ejemplo, una librería tiene libros, un departamento esta compuesto por empleados, etc.

La agregación se representa con una línea entre la clase que representa el todo (agregación) y los agregados (las partes), solo que al final del lado de la clase de agregación se coloca un rombo sin relleno como se indica en la figura 4.9.

La agregación debe ser usada solo en casos donde una clase se compone de otras, es decir, donde las partes no tienen sentido sin estar en el contexto del todo. Por ejemplo en una agencia de autos la relación entre un auto y sus puertas debe ser de agregación porque

el auto no se puede vender sin puertas, pero en el contexto de una refaccionaria la relación es una simple asociación, porque las refacciones se venden por separado.

Figura 4.9. Representación de agregación en UML.

4.3.5 Descripción de dependencias de eventos entre clases de análisis

Los objetos en ocasiones necesitan saber los eventos que pasan en otras instancias, sin que la instancia en la que ocurre el evento sepa cuántos o cuáles objetos necesitan notificación cuando el evento ocurre. Todo lo anterior se puede lograr por medio de una asociación de suscripción. Una asociación de suscripción indica que cuando ocurra un evento en la instancia “editora”, el objeto “subscriptor” será informado.

“Una relación de suscripción tiene una condición definiendo el evento que cause al objeto subscriptor ser notificado (por parte del editor)” [22]. Por lo general el objeto editor tiene el estereotipo de entidad debido a la pasividad que implica la naturaleza del mismo.

La asociación en este caso es unidireccional ya que el objeto editor del evento no se percata que otros objetos se están subscribiendo; por esta razón se pueden adicionar y borrar subscripciones sin problemas.

El ejemplo de la figura 4.10 considera el retiro de efectivo de una cuenta de banco, cuando se realiza una operación como esta, implica una transferencia. El objeto ControladorTransferencias hace los movimientos de dinero en el objeto Cuenta, estereotipado como de entidad, el objeto InterfazAviso esta suscrito al objeto Cuenta, para que en el caso de que exista un evento en que el balance de la transferencia sea menor que cero, le sea notificado para dar el aviso al cliente y no proceda la transacción.

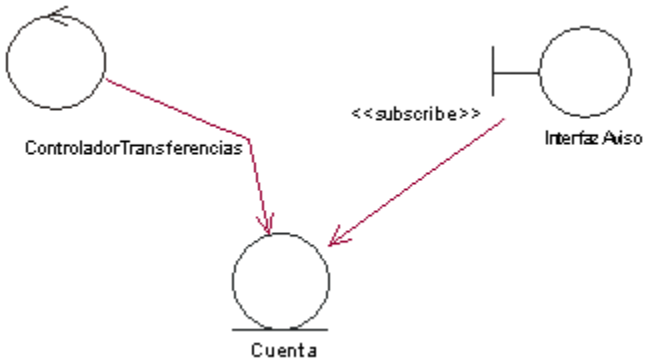

Figura 4.10. Ejemplo de suscripción.

4.3.6 Descripción de los mecanismos de análisis

En este paso se vuelve al tema de los mecanismos de análisis debido a que en este momento se han detallado un poco las clases y estas son más estables. El primer paso es realizar una revisión de la búsqueda y el mapeo, para recordar el trabajo hecho, después se procede con la descripción de los mecanismos de análisis.

El propósito de este paso es proveer información adicional acerca de cómo las clases de análisis van a utilizar los mecanismos de soporte. Se debe de hacer una descripción de las características que se requieren para la clase con respecto al mecanismo de análisis.

Siguiendo con el ejemplo de la figura 4.10, la clase "Cuenta" mapearía con el mecanismo de análisis llamado "persistencia". Lo anterior lleva a describir las siguientes características:

- **Clase de análisis:** Cuenta
- **Mecanismo de Análisis asociado:** Persistencia
- **Características:**
    - Tamaño por objeto: 1-60 Kbytes.
    - Volumen: Arriba de 20,000,000 de cuentas
    - Frecuencia de acceso:
        - *Creación: 5000 por día*
        - *Lectura: 200,000 por hora*
        - *Actualización: 20,000 por día*
        - *Borrado: 2,000 por día*

4.4 Identificación de elementos de diseño

El propósito de la actividad "Identificación de elementos de diseño" es analizar las interacciones de las clases de análisis para identificar los elementos del modelo de diseño

En este punto del desarrollo se acaban las cosas conceptuales (el análisis) y se inicia a particularizar, es decir, a diseñar. La actividad de análisis de caso de uso entrega clases de análisis que representan los elementos conceptuales que pueden realizar el comportamiento de los casos de uso. En el diseño, las clases de análisis evolucionan en diferentes tipos de elementos de diseño, entre los que están:

- **Clases de diseño:** Representan un conjunto de responsabilidades más refinadas.

- **Subsistemas:** Representan un conjunto de responsabilidades más generales que las clases, además estos pueden tener otros elementos como subsistemas o clases.
- **Clases activas:** Representan hilos de control en el sistema.
- **Interfaces:** Representan declaraciones abstractas de responsabilidades proveídas por una clase o un subsistema

También se identifican:

- **Eventos:** Son especificaciones de ocurrencias que requieren alguna respuesta del sistema.
- **Señales:** Representan mecanismos asincrónicos para comunicar ciertos tipos de eventos dentro del sistema.

Separando las partes del sistema y tratando con cada punto representado por estos conceptos, se simplifica el proceso de diseño y se clarifica la solución; los siguientes puntos explican las tareas que se realizan en la actividad “Identificación de elementos de diseño”.

4.4.1 Identificación y especificación de eventos

El propósito de este paso es identificar y especificar las ocurrencias internas y externas que el sistema debe responder.

Los eventos son ocurrencias internas y externas que causan alguna acción dentro del sistema. Estos pueden ocurrir en un tiempo aleatorio e independiente uno de otro, también pueden suceder periódicamente o ser dependientes de algún factor. Las respuestas oportunas a eventos es un punto importante en cuanto a la concurrencia.

Para la identificación de eventos se examinan los casos de uso y sus realizaciones. En este paso debe analizar cómo recibe el sistema los estímulos de los actores.

Ya que se analizó lo anterior, queda por especificar un mecanismo con el cuál el sistema estará “conciente” de estos eventos. Las interrupciones son mecanismos apropiados para eventos asincrónicos y para comunicar eventos no-periódicos que requieren respuesta inmediata.

El checkeo periódico (llamado “polling”) es más apropiado para eventos periódicos o aleatorios que ocurren continuamente y en los que no es necesaria una respuesta al momento. Incluso se puede hacer un buffer de eventos para responder a estos periódicamente.

4.4.2 Identificación y especificación de señales

Una señal es una entidad destinada a la comunicación asíncrona entre objetos. Las señales se usan para comunicar partes internas del sistema entre si y pueden ser usadas para comunicar errores o excepciones, y en general para cualquier situación en donde se requiera comunicación asincrónica. Las señales por lo general se implementan con interrupciones o con técnicas de espera periódica de eventos.

En este paso es importante saber qué mecanismos de señalización se usarán, porque el fallo de estos puede resultar en un comportamiento impredecible.

4.4.3 Identificación de clases

El siguiente paso en la identificación de elementos de diseño es la búsqueda de clases. Cuando una clase de análisis es simple y representa una sola abstracción lógica, puede ser mapeada 1 a 1 a una clase de diseño. Las clases de entidad por lo general permanecen intactas en el diseño.

Cuando se están identificando las clases de diseño, se deben de agrupar en el artefacto “Paquetes de diseño” para los propósitos de administración de la configuración.

4.4.4 Identificación de clases activas

Las clases activas representan hilos de control en el sistema. Para identificar estas clases, se deben considerar los requerimientos de concurrencia del sistema en el contexto de los objetos de análisis que se han identificado. Después se debe hacer la siguiente pregunta: ¿Existe la necesidad del sistema de responder a eventos generados externamente? Y si es así, ¿Qué clases de análisis están activas cuando el evento ocurre? Los eventos externos en el modelo de caso de uso se representan por medio de estímulos que vienen de actores interactuando con un caso de uso. Después se debe de observar a las correspondientes realizaciones de caso de uso para ver qué objetos interactúan cuando el evento ocurre. Entonces se debe iniciar por hacer conjuntos autónomos de objetos que colaboren entre si. Estos agrupamientos representan una vista inicial de las clases activas.

Las instancias de las clases activas representan hilos “lógicos” de ejecución. Estos hilos lógicos de ejecución no se deben de confundir mapeandolos con hilos de ejecución literales en el Sistema Operativo. En vez de eso, representan hilos conceptuales de ejecución en el espacio de solución. La meta es identificarlos en este punto del diseño para ser capaces de

partir la solución en unidades independientes basadas en uniones concurrentes en el sistema. El dividir el trabajo de esta forma hace que se pueda tratar el tema de la concurrencia desde un punto conceptualmente simple, ya que los hilos de ejecución pueden ser tratados de forma separada, excepto que estos hilos compartan clases pasivas.

En general, una clase activa debe de ser considerada cuando hay conflictos de concurrencia. Esta se debe usar para representar algún objeto concurrente externo o una actividad concurrente dentro de la computadora, lo anterior permite monitorear y controlar las actividades de este tipo.

4.4.5 Identificación de subsistemas

Cuando una clase de análisis es compleja, y parece que tiene comportamiento que no es posible que se encapsule por una sola clase puede ser mapeada a un subsistema de diseño. Los subsistemas se usan para encapsular este tipo de colaboraciones de tal forma que para el cliente del subsistema le sea transparente el diseño interno.

Para tomar la decisión de si un conjunto de colaboraciones debe de ser un subsistema se debe de hacer la pregunta: ¿Esta colaboración puede ser o debe ser desarrollada por un equipo de diseño separado?, efectivamente, los subsistemas también sirven para reunir en un solo componente el trabajo de un equipo de trabajo.

Los subsistemas también sirven para representar productos existentes o sistemas externos en el diseño tales como BD, componentes comprados, sistemas heredados, etc.

En seguida se muestran algunos tips para el agrupamiento de clases en subsistemas en los que se busca una mayor independencia para crear una mantenibilidad correcta del software en su conjunto.

- Si las interfaces están relativamente aisladas de las clases de entidad; se deben crear subsistemas que integren horizontalmente a las interfaces y también agrupar las clases de entidad en subsistemas horizontales, después vincular ambos subsistemas.
- Si las clases de interfaz y las clases de entidad están muy acopladas; se deben de agrupar las colaboraciones verticalmente, agrupando las interfaces con sus clases de entidad en varios subsistemas.
- Del punto anterior se puede sacar la idea de agrupar conjuntos de clases muy cohesivas. Por otro lado si una clase no se acopla mucho con el grupo quiere decir

que debe partirse en varias clases las cuales entre si formarán un subsistema. “La finalidad es hacer que los subsistemas sean muy independientes y que las clases estén más acopladas entre si”.

- Se debe de observar la distribución de los subsistemas en los nodos, en muchas arquitecturas no es posible repartir una instancia de un subsistema en varios lados, en este caso se deben hacer muchas instancias del mismo subsistemas y delimitar la funcionalidad en cada nodo según sea la necesidad.

4.4.5.1 Identificación de interfaces de los subsistemas

Para cada subsistema se deben identificar sus interfaces. Ya que se han identificado las colaboraciones, se debe identificar cuál es la responsabilidad que se activa cuando inicia la colaboración. La responsabilidad se refina cuando se determina cuál es la información de entrada del cliente y cuál es la información de salida cuando la colaboración acaba. Estos valores son un prototipo de parámetros de entrada-salida y valores de retorno de las operaciones del subsistema.

Después del paso anterior puede que existan “n” interfaces para el subsistema. El siguiente paso es encontrar las interfaces con operaciones redundantes y después unir estas interfaces de tal modo que no exista duplicación de funciones.

4.4.5.2 Mapeo de las interfaces a los subsistemas

Ya que se definieron las interfaces se debe crear una asociación de realización entre el subsistema y las interfaces que realiza. “Una realización de un subsistema a una interfaz indica que hay uno o más elementos dentro del subsistema que realizan las operaciones de la interfaz.” [22]. Después, cuando se diseñan los subsistemas, se refinarán estas realizaciones ya que el diseñador del subsistema definirá qué elementos del subsistema son los que realizan las operaciones de la interfaz.

4.5 Revisión del diseño

La tercera y última actividad del detalle de flujo de trabajo del análisis de los casos de uso es la “Revisión de diseño”, sus propósitos se describen en los siguientes puntos:

- Verificar que el modelo de diseño cumple con los requerimientos del sistema y que sirve como base para la implementación.

- Asegurarse que el modelo de diseño es consistente con respecto a las guías generales de diseño.
- Asegurarse que las guías de diseño cumplen con sus objetivos

La revisión del diseño puede ser de los pasos descuidados durante el análisis y diseño, sin embargo es uno de los más importantes en el proceso. La revisión se debe realizar en cada iteración de la fase de elaboración y construcción.

El encargado de realizar la actividad es el "Revisor de diseño", en el anexo B se explica el perfil del trabajador.

4.5.1 Revisión del modelo de diseño en su conjunto

Este paso tiene como propósito asegurarse que toda la estructura para el diseño esté bien formada. Si se encuentra en las primeras iteraciones de la fase de elaboración, la revisión se debe enfocar en la estructura general del modelo, con un énfasis especial en el layering. Se deben examinar las dependencias entre paquetes y subsistemas para asegurar un acoplamiento holgado. También se deben examinar los contenidos internos de subsistemas y paquetes para asegurar una alta cohesión entre los elementos. En general todos los elementos se deben revisar para asegurar que tienen responsabilidades apropiadas y que los nombres reflejan el propósito del elemento.

En este paso se insta a ver el sistema en lo general, ya que observando las cosas "desde arriba", ofrece una perspectiva diferente y se pueden ver problemas que de una forma minuciosa no se encontrarían, éste es el caso del layering

4.5.1.1 Revisión del layering

El primer paso es revisar el layering del sistema, usando el siguiente criterio:

- No hay más de 7 (+/- 2) capas
- La racionalidad de la descripción de cada capa se presenta claramente y es aplicada consistentemente.
- Los limites de las capas se han respetado en el diseño
- Las capas se usan para encapsular los límites conceptuales entre diferentes tipos de servicios y proveen abstracciones útiles que hacen más fácil el entendimiento del diseño.

4.5.1.2 Revisión del acoplamiento y cohesión

El segundo paso es la revisión del acoplamiento y la cohesión de las clases. “El acoplamiento entre paquetes tiene aspectos positivos y negativos, es bueno porque mejora el reuso de los componentes, es malo (si se aplica incorrectamente) porque representa dependencias que hacen al sistema difícil de cambiar o de evolucionar.” [24]

A continuación se muestran tres consideraciones sobre el acoplamiento de paquetes:

- Los paquetes no se deben acoplar de forma cruzada, es decir, que el paquete A dependa del B y que el B dependa del A
- Los paquetes de las capas más bajas no deben depender de paquetes de las capas superiores. Estos paquetes deben depender solo de capas del mismo nivel y de la capa inferior siguiente.
- En general, las dependencias no deben saltar capas.

También se debe revisar que exista un acoplamiento “holgado” entre los elementos. En figura 4.11 se muestran dos ejemplos de acoplamiento de paquetes, a la izquierda se muestra un acoplamiento estrecho, es muy difícil reusar el paquete A porque todas las clases de este paquete están relacionadas con el paquete B, si se quisieran separar ambos paquetes, se tendrían que sumar al paquete B las clases A1, A2, A3 y A4.

A la derecha hay un acoplamiento holgado, debido a que solo una clase del paquete A esta relacionada con el paquete B. Por esta razón es mucho más fácil desconectar la relación entre ambos paquetes.

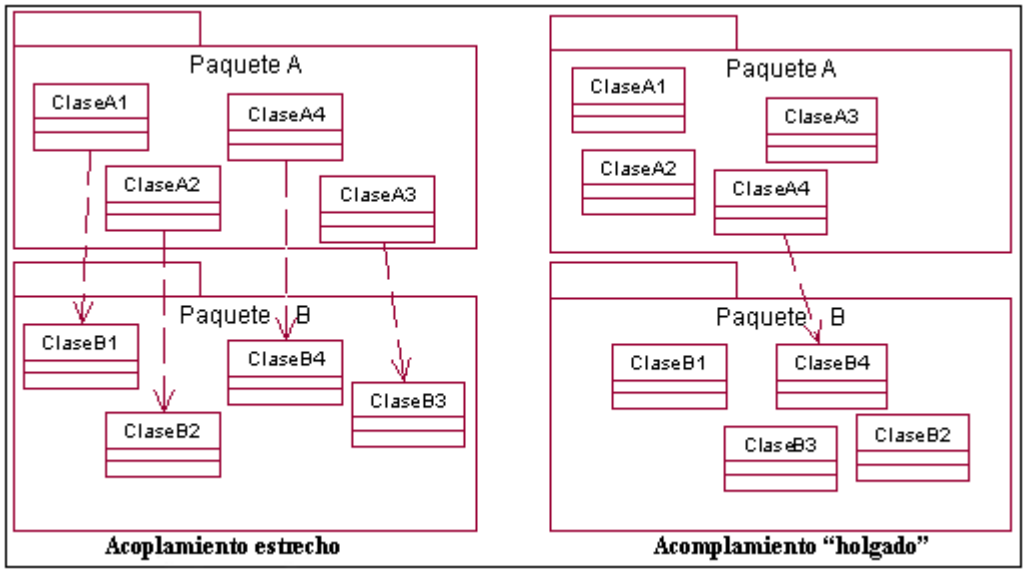

Figura 4.11. Tipos de acoplamientos entre elementos de diseño.

Otro aspecto importante es la cohesión entre los elementos, la figura 4.12 muestra un ejemplo de este concepto. A la izquierda se muestra un paquete con alto nivel de cohesión debido a que todas las clases que encapsula están relacionadas funcionalmente unas con otras. En cambio, la clase de la izquierda tiene una cohesión baja porque no solo atiende las necesidades de los cursos, sino que también es responsable por crear y borrar horarios.

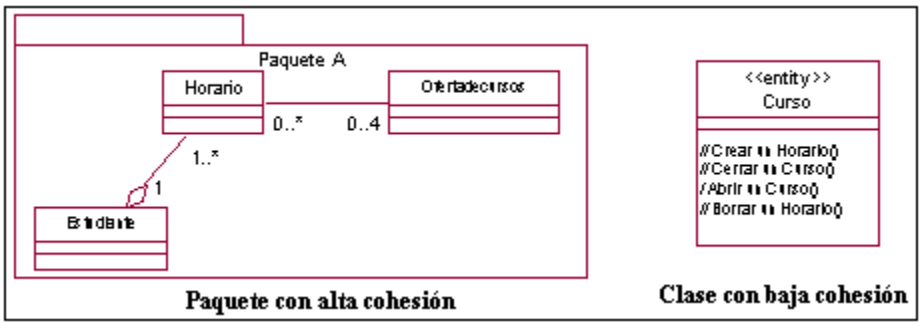

Figura 4.12. Ejemplos de niveles de cohesión.

4.5.2 Revisión de cada realización de caso de uso

El paso siguiente en la actividad es revisar cada una de las realizaciones de caso de uso. Primero de debe estar seguro que no hay comportamiento faltante, revisando que todos los escenarios de la iteración han sido cubiertos por las realizaciones de caso de uso. Todo el comportamiento relevante de los flujos secundarios se debe haber descrito en la realización que le toca.

El siguiente paso es revisar que el comportamiento de la realización de caso de uso está distribuida correctamente entre los elementos del modelo; en este punto también se debe asegurar que las operaciones se usan correctamente, que todos los parámetros están bien definidos y que los valores de retorno son del tipo correcto.

4.5.3 Preparación del registro de revisión y los documentos de defectos

El registro de revisión es una forma que se llena por cada revisión formal, se crea como un documento de control para administrar la ejecución de la revisión de los artefactos del proyecto.

El registro de revisión se emite a todos los participantes al iniciar el proceso de revisión para capturar los resultados de cualquier acción que surja a lo largo de las tareas que se realicen, este artefacto forma un registro auditable en el documento, el equipo debe listar cualquier problema identificado durante la revisión.

Los revisores deben identificar:

- Problemas con los artefactos revisados (defectos).
- Problemas del proyecto cuyos síntomas se han identificado de los artefactos de revisión
- Problemas del producto cuyos síntomas son identificados de los artefactos.

4.6 Artifacts principales

Los siguientes puntos muestran los artifacts importantes del detalle de flujo de trabajo "Análisis del comportamiento de los casos de uso".

4.6.1 Modelo de análisis

El modelo de análisis describe la realización de casos de uso y sirve como una abstracción del modelo de diseño, es decir, contiene los resultados del análisis de caso de uso (clases de análisis). Puede ser un artefacto temporal ya que más tarde evolucionará en un modelo de diseño, todo depende si se sigue actualizando conforme avanza el proyecto. Este modelo se puede ver más tarde como un resumen conceptual del sistema.

La meta del modelo de análisis es hacer un mapeo preliminar del comportamiento requerido en los elementos del modelo sin preocuparse por lo pronto en el ambiente de implementación.

4.6.1.1 Partes importantes del artefacto

Los siguientes puntos muestran las partes que debe tener este artefacto

- **Introducción:** Esta sección del modelo debe incluir el propósito, alcance, definiciones, acrónimos, abreviaciones, referencias y un resumen.
- **Paquetes de análisis:** Representan las capas que se definieron en el layering. "Los paquetes proporcionan un medio para organizar los artefactos del modelo de análisis en piezas manejables" [13]
- **Clases de análisis:** En esta sección se especifican las clases de análisis que corresponden a cada paquete.
- **Realizaciones de caso de uso:** Las realizaciones de caso de uso son las que definen la trazabilidad entre el análisis y los requerimientos.
- **Diagramas:** Muestra las relaciones entre las clases de análisis y otros elementos.

4.6.2 Modelo de diseño

El modelo de diseño es un conjunto de diagramas de de objetos que describe la realización de casos de uso, pero a diferencia del modelo de análisis no es conceptual. Este modelo es mucho más detallado y acorde al lenguaje de programación de la implementación, por esta razón sirve como una abstracción de la implementación y del código fuente. Este artefacto es la entrada principal para las disciplinas de implementación y pruebas.

El modelo de diseño "es un artefacto comprensivo y compuesto porque incluye todas las clases de diseño, subsistemas, paquetes, colaboraciones y las relaciones entre ellos" [22]

4.6.2.1 Partes importantes del artefacto

Los siguientes puntos muestran las partes que debe tener el modelo de diseño.

- **Introducción:** Esta sección del modelo debe incluir el propósito, alcance, definiciones, acrónimos, abreviaciones, referencias y un resumen.
- **Paquetes y subsistemas de diseño:** Los paquetes en el modelo representan una jerarquía o un nivel en el layering. Estos paquetes hacen más manejable el modelo porque agrupan los elementos.
- **Elementos agrupados por paquetes y subsistemas:** Estos elementos son los que hacen en si el modelo, cada elemento se agrupa en un subsistema o paquete según el nivel de abstracción. Entre estos elementos se encuentran por lo general clases y sus relaciones
- **Diagramas:** Son todos los diagramas que hacen relacionar a las clases para realizar el comportamiento deseado a un nivel de diseño. En este apartado se encuentran también las realizaciones de caso de uso.

4.6.2.2 El modelo de diseño en Rational Rose

La figura 4.13 muestra una pantalla de Rational Rose donde se muestra la ventana del modelo de diseño, en ella se encuentran las diferentes vistas, el modelo de diseño se encuentra dentro de la vista lógica. El diagrama principal del modelo de diseño muestra la estructura de capas del sistema (paquetes), dentro de estos paquetes se encuentran los elementos que pertenecen a un determinado nivel de abstracción (capa).

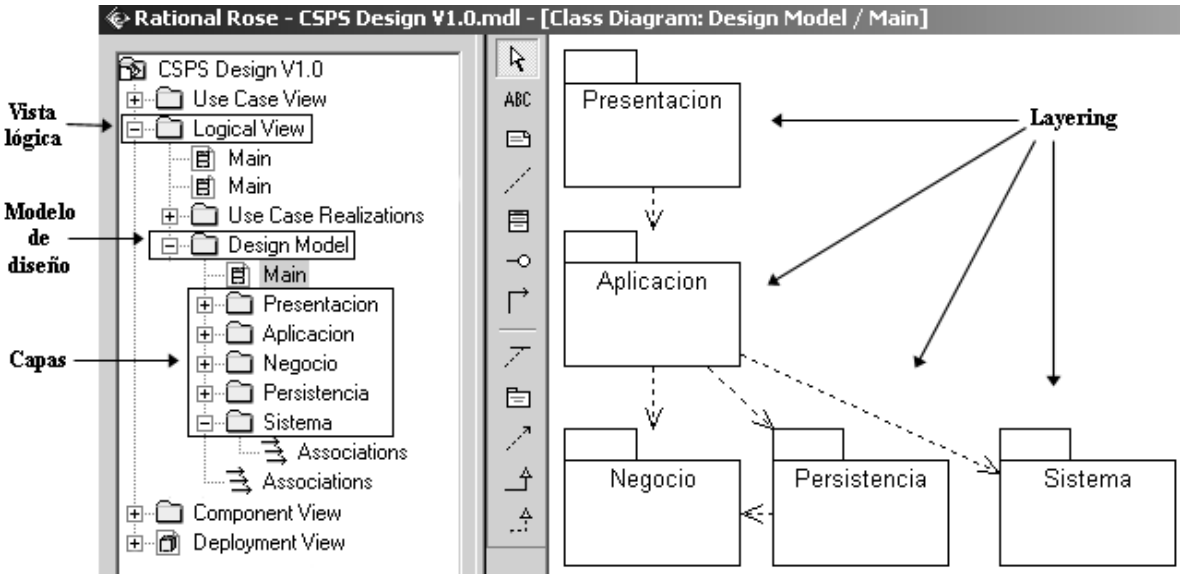

Figura 4.13. Vista del modelo de diseño en Rational Rose.

4.6.3 Registro de revisión

El registro de revisión es una forma o documento que sirve para capturar los resultados de la revisión de un artefacto.

4.6.3.1 Partes importantes del artefacto

A continuación se listan las partes más importantes del registro de revisión.

- **Identificación del proyecto y tipo de revisión:** Un tipo de revisión puede ser: inspección de código, inspección de trazabilidad de requerimientos,…
- **Artefactos revisados y objetivos:** En esta parte del documento se listan los artefactos que son sujetos a revisión y que describen los objetivos de la misma
- **Participantes:** En esta parte se describe el nombre, puesto y rol del revisor
- **Hora, Fecha y Lugar de la revisión**
- **Problemas y recomendaciones de solución**
- **Status:** En esta parte del artefacto se listan todas las acciones que resulten de la revisión, también puede ir el nombre del responsable de la acción y la fecha plazo. Las acciones típicamente son preventivas o correctivas para eliminar o evitar que aparezca un problema.

Capítulo 5

Refinación de la arquitectura

El arquitecto ideal debe ser una persona de letras, un matemático, familiarizado con los estudios históricos, un estudiante diligente de la filosofía, conocedor de la música, no ignorante en medicina, familiarizado con la astronomía y sus cálculos

Vitruvius, circa A deC.

Introducción

El detalle de flujo de trabajo “Refinación de la arquitectura” provee una transición natural entre el análisis y el diseño, ya que resuelve los problemas que se han dejado pendientes tales como los mecanismos, la actualización del documento de arquitectura de software y los aspectos de la concurrencia.

El círculo de la figura 5.1 muestra en qué parte se encuentra este detalle de flujo de trabajo dentro del análisis y diseño, en ella se puede observar que se realiza concurrentemente con el detalle del análisis de comportamiento de los casos de uso, es por esta razón que la actividad “identificación de los elementos de diseño” aparece en ambas, solo que aquí sirve para actualizar el documento de arquitectura de software.

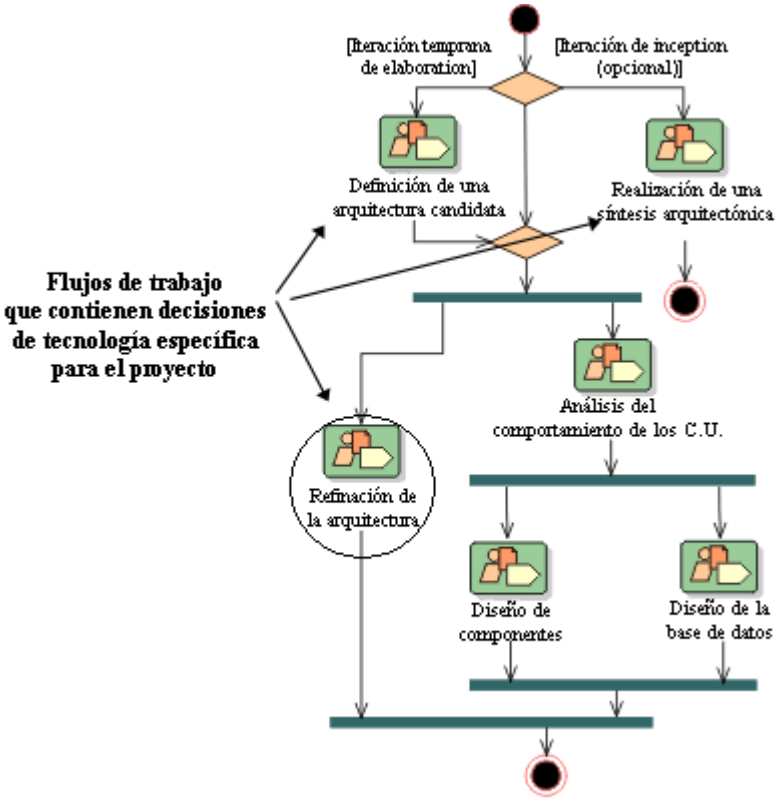

Figura 5.1. Detalle de flujo de trabajo “Refinación de la arquitectura”.

La figura 5.1 también muestra los tres detalles de flujo de trabajo que tienen que ver más con la arquitectura. Estos son los que contienen las actividades más críticas en cuanto a la toma de decisiones de tecnología específica para el proyecto.

En la figura 5.2 se muestran los artefactos de entrada y salida de cada actividad. “El documento de arquitectura de software es el artefacto principal de los detalles del flujo de

trabajo Refinación de la arquitectura, Definición de una arquitectura candidata y Realización de una síntesis arquitectónica" [3]

El encargado de realizar la mayoría de las actividades es el arquitecto de software, en el apéndice B se puede encontrar un resumen de su perfil.

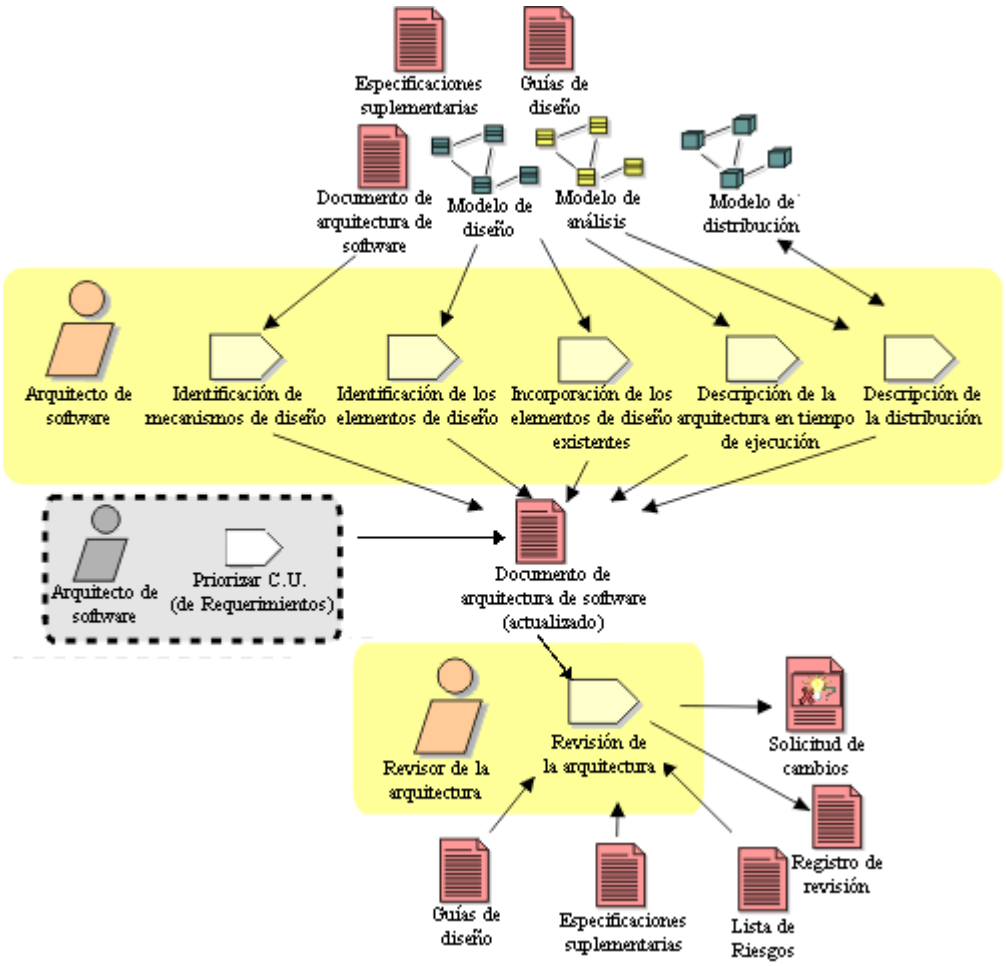

Figura 5.2. Actividades, Artefactos y Roles del detalle de flujo de trabajo "Refinación de la arquitectura".

Las actividades del detalle de flujo de trabajo "Refinación de la arquitectura" se realizan de la manera siguiente: Se inicia con las actividades "Identificación de mecanismos de diseño" e "Identificación de los elementos de diseño". Después se realiza la actividad "Incorporación de los elementos de diseño existentes" para asegurarse que los nuevos elementos no duplican la funcionalidad de los elementos existentes.

Ya que se va entrando en el diseño, los aspectos de concurrencia y distribución se introducen en las actividades "Descripción de la arquitectura en tiempo de ejecución" y "Descripción de la distribución" respectivamente.

5.1 Propósito del detalle de flujo de trabajo "Refinación de la arquitectura"

El propósito del detalle de flujo de trabajo "Refinación de la arquitectura es:

- Proveer una transición natural de las actividades de análisis al diseño identificando elementos y mecanismos de diseño apropiados
- Mantener la consistencia e integridad de la arquitectura asegurándose de que los elementos de diseño identificados se integran de manera natural a los ya existentes y que los componentes existentes se re-usan eficientemente en el esfuerzo de diseño.
- Describir la organización de los sistemas en tiempo de ejecución y de la arquitectura de distribución.

5.2 Conceptos preliminares

Los siguientes puntos muestran los conceptos que son necesarios para comprender mejor el resto del capítulo.

5.2.1 Mecanismos de diseño y mecanismos de implementación

A medida que pasan las iteraciones, se conocen mejor los requerimientos y se particulariza más y más el sistema, en el caso de los mecanismos, también se van detallando, hasta que se pueden implementar concretamente.

Un mecanismo de diseño es un refinamiento de un mecanismo de análisis, en el sentido de que se le agregan requisitos para que deje de ser conceptual y empiece a ser más particular a la implementación del sistema. Ahora que se conoce mejor la arquitectura, se pueden agregar detalles tales como la tecnología de implementación.

El refinamiento del mecanismo de análisis no llega hasta determinar un proveedor del mecanismo en particular o un lenguaje de programación en específico.

De manera similar, los mecanismos de implementación son refinamientos de los mecanismos de diseño, esta vez se agregan requisitos tales como el manejador de base de datos, el lenguaje de programación, es decir, el mecanismo esta listo para implementarse.

5.3 Identificación de mecanismos de diseño

La primera actividad del detalle de flujo de trabajo "Refinación de la arquitectura" es la identificación de mecanismos de diseño, su propósito es refinar los mecanismos de análisis

para llegar a ser mecanismos de diseño basados en las limitantes impuestas por el ambiente de implementación.

Los mecanismos de análisis proveen al sistema un conjunto de servicios a manera conceptual, estos son usados por las clases de análisis y ofrecen una manera conveniente para tratar con el comportamiento complejo que requiere el sistema desde el inicio.

En este momento es tiempo de empezar a refinar la información recopilada durante el análisis. Los siguientes puntos muestran el desarrollo de la actividad.

5.3.1 Clasificación de clientes de los mecanismos de análisis

La primera tarea de este punto es la identificación de los clientes de cada mecanismo de análisis.

Se deben revisar todos los clientes de un mecanismo de análisis determinado observando las características que requieren que soporte el mecanismo. Por ejemplo, algunos objetos de análisis necesitan la persistencia, pero los requerimientos de éste pueden variar: una clase que tenga mil objetos persistentes tiene requerimientos muy diferentes a una clase que deba tener cuatro millones de objetos persistentes. De manera similar hay objetos que necesitan tiempo de respuesta diferente aunque ambos tengan las mismas características.

Después de la tarea anterior se realiza la identificación de los perfiles de cada mecanismo, ya que puede haber una amplia variedad en las características que debe soportar cada mecanismo de acuerdo a los grados de desempeño, seguridad, costo de implementación, etc. Cada mecanismo de análisis es diferente y cada cliente del mecanismo requiere diferentes servicios. Muchos mecanismos deben saber cuántas instancias van a manejar, además del tamaño de los objetos. El movimiento de grandes cantidades de datos en el sistema provoca problemas críticos de desempeño con los que se debe tratar. En este paso se identifican las características que debe soportar cada mecanismo.

Por ultimo se hace una agrupación de clientes según los perfiles de mecanismos. En este paso se forman grupos de clientes que comparten la necesidad de un mecanismo de análisis con perfiles similares; después se identifica un mecanismo de diseño con base en tal necesidad. Estas agrupaciones proveen una aproximación inicial a los mecanismos de diseño.

Por otro lado, diferentes perfiles de características pueden llevar a diferentes mecanismos de diseño, aunque surjan del mismo mecanismo de análisis. Por ejemplo, el

mecanismo de persistencia en el análisis puede dar origen a muchos mecanismos de diseño tales como: la memoria, bases de datos distribuidas o archivos planos. Los mecanismos de diseño son los refinamientos de los mecanismos de análisis, con base en perfiles característicos diferentes.

5.3.2 Creación del inventario de los mecanismos de implementación disponibles

Ya que se ha hecho una primera aproximación hacia los mecanismos de diseño y antes de seguir con el refinamiento, se hace un inventario de los mecanismos de implementación que se tengan a disposición. Los siguientes puntos muestran algunos ejemplos:

- Mecanismos ofrecidos por medio de Middleware
- Mecanismos de los Sistemas Operativos
- Mecanismos que ofrece algún componente
- Mecanismos que ofrecen las librerías de clases
- Paquetes de propósito especial (constructores de GUI, sistemas de información geográfica, DBMS, etc.).

Después de hacer el inventario se determina en dónde se pueden usar los mecanismos de implementación inventariados y dónde se necesita construir o comprar un nuevo mecanismo.

5.3.3 Mapeo de los mecanismos de diseño a mecanismos de implementación

Los mecanismos de diseño proveen una abstracción de los mecanismos de implementación, haciendo la comunicación entre el análisis y la implementación de estos mismos. La abstracción de mecanismos arquitectónicos permite abordar el tema de cómo proveer los servicios necesarios al sistema sin desviarse con detalles particulares.

El primer paso es el mapeo y análisis para la compra de mecanismos faltantes, se toman todas las características de los mecanismos de diseño y en base a esto de determinan cuáles se mapearán a los mecanismos de implementación, de tal manera que la decisión sea razonable, económica y factible.

El siguiente paso es hacer un análisis costo-beneficio sobre la compra de los mecanismos de implementación faltantes en base a criterios como economía de la licencia del producto, madures, soporte y relación con el vendedor del elemento a comprar.

Con base en el análisis de los mecanismos se realiza una toma de decisiones, ya que puede que algunos mecanismos de implementación no cumplan con todas las características idóneas de su contraparte de diseño, en este caso se puede optar por desarrollarlo en la misma organización. También se puede encontrar que algunos mecanismos de implementación no se usan tanto como para comprarlos.

Si se tiene duda en algún mecanismo se pueden tomar medidas provisionales para observar los riesgos que tiene cada mecanismo, estas pueden ir desde desarrollar un prototipado del mecanismo, hacer una prueba del mismo en un ambiente adecuado o hasta comprar una licencia provisional del producto.

5.3.4 Documentación de los mecanismos arquitectónicos

El rol del arquitecto de software en esta actividad está en validar y decidir sobre los mecanismos, ya sea construyendo o integrándolos para después verificar su efectividad en el ambiente de trabajo.

Los mecanismos y los detalles con respecto a su uso se documentan el en artefacto guías de diseño. El mapeo de los mecanismos de análisis a mecanismos de diseño a mecanismos de implementación se explican en el documento de arquitectura de software junto con las validaciones, decisiones y raciocinios.

5.4 Identificación de los elementos de diseño (Actualización del Documento de Arquitectura de Software)

El propósito de la actividad “Identificación de los elementos de diseño” es analizar las interacciones de las clases de análisis para identificar los elementos del modelo de diseño

Esta actividad es muy importante porque es donde se empieza a identificar los elementos que después se refinarán adicionándole atributos, operaciones y relaciones. Los elementos de diseño se pueden empezar a identificar en iteraciones anteriores, es por esta causa que la actividad se ha explicado antes.

Solo queda por actualizar el documento de arquitectura de software y al mismo tiempo hacer una revisión de todo el artefacto para asegurar su consistencia e integridad antes de pasar a las actividades del diseño del sistema.

5.5 Incorporación de los elementos de diseño existentes

Los propósitos de la actividad “Incorporación de los elementos de diseño existentes” son:

- Analizar las interacciones de las clases de análisis para encontrar interfaces, elementos de diseño y subsistemas de diseño.
- Refinar la arquitectura, incorporando elementos de reuso donde sea posible.
- Identificar soluciones comunes a problemas de diseño comunes
- Incorporar los elementos de diseño importantes arquitectónicamente

En esta actividad se integran los elementos nuevos con elementos que ya se han identificado o elementos que se pretende reusar, además se trata de mantener la consistencia e integridad del documento de arquitectura de software haciendo una actualización del mismo.

El proceso es el siguiente: el primer paso es identificar los elementos de reuso, después se hace un proceso de ingeniería inversa a estos elementos para incorporarlos al modelo de diseño y por ultimo se actualiza el documento de arquitectura de software.

5.5.1 Identificación de oportunidades de reuso

Esta actividad inicia comparando cada interfaz identificada en las actividades anteriores con las interfaces proveídas por subsistemas o componentes existentes. Probablemente las interfaces no sean del todo iguales, pero se pueden buscar aquellas que sean muy similares. La búsqueda de similaridades se hace primero en el comportamiento y los valores de retorno, después en los parámetros.

En el siguiente paso se pretende que con cambios menores a las nuevas interfaces se pueda hacer una estrategia de reuso. Los cambios incluyen reestructurar o adicionar parámetros a la interfaz candidata, después hacer un split, es decir, dividir la interfaz en varias y administrar el comportamiento de modo que una o más de las nuevas interfaces tenga el comportamiento de la interfaz (o interfaces) de reuso.

Después de la simplificación y factorización de interfaces, si hay dos interfaces que empatan exactamente, se deja la interfaz existente y se elimina la interfaz candidata para eliminar la redundancia de comportamiento.

Por ultimo se analizan los componentes existentes y el conjunto de subsistemas candidatos. Se deben unir ambos de manera que los componentes existentes se usen lo más

posible para satisfacer el comportamiento requerido del sistema. Cuando un subsistema candidato se realiza por medio de un componente existente se crea una relación de trazabilidad entre el subsistema de diseño y el componente en el modelo de implementación.

Cuando se mapean los subsistemas a componentes, se deben considerar los mecanismos de diseño asociados con el subsistema ya que los requerimientos de performance o seguridad pueden deshabilitar al componente existente para ser reusado, aun cuando empata perfectamente en los requerimientos operacionales.

5.5.2 Aplicación de ingeniería inversa a componentes y bases de datos

En este paso se integran los elementos con mayores probabilidades de reuso provenientes de proyectos anteriores o fuentes externas. Con base en el código existente, bases de datos heredadas, etc., se puede hacer una búsqueda de elementos que ayuden en la iteración actual para que no se haga trabajo doble.

En las organizaciones que están especializadas en sistemas similares, a menudo hay componentes comunes que proveen la mayoría de los mecanismos arquitectónicos que necesitan sus sistemas. También hay compañías que hacen sus propias librerías de clases o código para fomentar la reusabilidad y eficiencia en el desarrollo de sistemas. Estos componentes se deben examinar para determinar la conveniencia y compatibilidad con la arquitectura del sistema.

La ingeniería inversa o cracking es un termino general que se refiere a la acción de tomar “algo terminado” y tratar de ir “hacia atrás” en el proceso de manufactura, con el objetivo de conocer este proceso y/o cambiar el producto. En este caso ese algo es el componente o B.D y el objetivo es hacer una regresión en el proceso de manufactura y mostrarlo en forma de subsistemas y sus relaciones para poder integrarlo en el modelo de diseño.

A los componentes existentes se les debe aplicar ingeniería inversa e incorporarlos en el modelo de diseño en forma de subsistema, con sus interfaces correspondientes.

Para reusar las definiciones de clases que están implícitamente en una Base de Datos primero se determina qué información de la aplicación reside en la base de datos, después se aplica ingeniería inversa a las estructuras que contienen esta información y por ultimo se hace un mapeo de las clases de la aplicación y las estructuras usadas de la B.D.

5.5.3 Actualización de la organización del modelo de diseño

Después de aplicar ingeniería inversa a los elementos, se actualiza el modelo de diseño y se rebalancea su estructura si es necesario. Esto implica reempacar algunos elementos para reducir el acoplamiento y mejorar la cohesión entre paquetes. Una cohesión idónea permite que cada subsistema pueda ser desarrollado individualmente o por equipos de trabajo, según la complejidad. Es sabido que una independencia total es imposible, sin embargo, lo que si se puede lograr es un acoplamiento holgado que haga más fácil el desarrollo de sistemas grandes o complejos.

Si a algún paquete se le agregan muchos elementos, a tal grado que sea difícil de administrar, se puede partir en varios paquetes. Debido a lo anterior, el sistema se puede tornar más complejo y difícil de administrar, se pueden necesitar más capas para que el sistema se vuelva de nuevo administrable y comprensible.

En este paso se definen los elementos de las capas mas bajas del sistema es decir, el Middleware y la capa de sistema. Si se está desarrollando software que corra en intranet o en internet, los elementos de Java deberían ir en el Middleware, ya que son elementos independientes a la plataforma, si se utilizan servicios inherentes al sistema operativo, los elementos se colocan en la capa del sistema.

5.5.4 Actualización de la vista lógica del documento de arquitectura de software

El siguiente punto es actualizar la vista lógica del documento de arquitectura de software ya que puede haber clases, paquetes o subsistemas que sean importantes desde la perspectiva arquitectónica. Esta acción no solo permite mantener consistente e integro el documento, sino ser un medio de comunicación fidedigno del cual se puedan guiar los demás equipos de desarrollo o los elementos individuales.

5.6 Descripción de la arquitectura en tiempo de ejecución

El propósito de la actividad “Descripción de la arquitectura en tiempo de ejecución” es analizar los requerimientos de concurrencia para identificar los procesos necesarios en el sistema y definir los elementos que los soporten.

Los objetos activos (instancias de clases activas) se usan para representar hilos concurrentes de ejecución en el sistema. Teóricamente, cada objeto activo tiene su propio hilo de control. El mapeo de objetos activos a los hilos o procesos actuales en el sistema

varía según los requerimientos del sistema. En esta actividad se muestran los pasos para la descripción y diseño de los aspectos relacionados a la concurrencia y procesos.

5.6.1 Análisis de los requerimientos de la concurrencia

En este paso se define el alcance y los requerimientos de las tareas que se ejecutan en paralelo, ya que esta información ayuda en el desarrollo de la arquitectura.

Durante la identificación de elementos de diseño, las clases activas ayudaron a representar eventos que pudieran ocasionar conflictos de concurrencia, las clases activas por su naturaleza independiente en forma de hilo de ejecución permiten simplificar este tipo de problemas.

En éste paso se toman en cuenta otros aspectos inherentes a la concurrencia, aquellos que provienen de los requerimientos no funcionales, estos se manejan por:

- El grado de distribución del sistema.
- La complejidad de los principales algoritmos.
- El grado de ejecución paralela que soporta el ambiente.
- Requerimientos de tolerancia a fallas.

Como indican los puntos anteriores, hay varios problemas derivados de la concurrencia y además de identificarlos, es útil ordenarlos por prioridades para su mejor tratamiento.

5.6.2 Identificación de procesos e hilos

El siguiente paso es identificar los procesos e hilos que soportará el sistema. Para esto, el método mas simple es agrupar todos los objetos activos y asignarle un proceso o hilo común el cual tendrá un calendarizador de objetos activos que los controle. Este método tiene ventajas y desventajas, del lado de las desventajas entra la situación en donde un objeto activo llama a otro proceso o hilo diferente al que pertenece, como resultado, se bloquearán todos los objetos del proceso o hilo que es invocado.

El ejemplo anterior lleva a la conclusión que los objetos activos se deben agrupar en hilos o procesos con base en sus necesidades de correr concurrentemente con invocaciones asincrónicas, ya que estas son las que provocan los bloqueos. En casos extremos se puede optar por darle a cada objeto su propio hilo.

“Los hilos tienen la ventaja que adicionan menos carga de trabajo, por otro lado comparten el mismo direccionamiento en memoria y son inherentemente mas riesgosos que

los procesos. Por otra parte, ya que los procesos representan unidades independientes de recuperación, en la mayoría de los Sistemas Operativos puede ser útil asignar los objetos activos a procesos basados en la necesidad de recuperarse de manera independiente" [22]

5.6.3 Identificación de los mecanismos de comunicación entre procesos

En este paso se identifican las formas y mecanismos para la comunicación de los procesos del sistema. Los mecanismos de comunicación entre procesos (IPC) permiten enviar mensajes de objetos que se encuentran en procesos separados, algunos tipos de mecanismos son:

- **Memoria compartida:** Este mecanismo sirve para permitir la sincronización
- **Semáforos:** Se usa para bloquear accesos simultáneos a componentes compartidos tales como la memoria.
- **Paso de mensajes:** El paso de mensajes puede ser punto a punto o multipunto.
- **RPC:** Llamadas a procedimientos remotos en el caso de sistemas distribuidos.

5.6.4 Asignación de recursos a los mecanismos de comunicación entre procesos

Los mecanismos de comunicación entre procesos por lo general son escasos y el costo de su incremento es alto, se debe hacer una planeación y asignación correcta de estos recursos porque cuando son rebasados por las peticiones, el desempeño del sistema se ve severamente afectado.

En el caso de que los recursos sean escasos se puede seguir la estrategia de reducir los procesos para minimizar la necesidad de utilización de los mismos o aumentar los activos insuficientes, aunque esto ultimo es muy costoso.

Dependiendo de la decisión que se tome, el problema en el sistema se ira incrementando paulatinamente en vez de hacer que el sistema se caiga de manera repentina. Si el sistema requiere una configuración especial del ambiente en tiempo real para incrementar la disponibilidad del recurso, la instalación del sistema lo deberá hacer automáticamente o instruir al administrador del sistema antes de que esté en producción.

5.6.5 Distribución de los elementos de modelo entre los procesos

Las instancias de una clase o subsistema se deben ejecutar al menos en un proceso, o en varios procesos. “El proceso provee un ambiente para la ejecución de la clase o subsistema” [22]

Por medio de las estrategias siguientes se determina el grado correcto de concurrencia y el conjunto correcto de procesos.

- Primero se agrupan clases y subsistemas del modelo de diseño en conjuntos que tienen las siguientes características (a) Los elementos cooperan mutuamente (b) los elementos además se necesitan ejecutar en el mismo hilo de control.
- Después se ponen en hilos separados los subsistemas y clases que no interactúan.
- El agrupamiento sigue hasta que se reduzca al mínimo y hasta que permita la distribución y uso de los recursos físicos.

5.7 Descripción de la distribución

El propósito de la actividad “Descripción de la distribución” es detallar la repartición de la funcionalidad a través de los nodos.

La distribución de procesos en dos o mas nodos requiere primero examinar la comunicación entre los procesos del sistema. La idea de que la distribución reduce la carga de trabajo de las maquinas es relativa pues todo se puede perder por el trabajo adicional derivado de la comunicación entre procesos.

“La distribución es un área donde la suma puede ser, y usualmente es, menos que la suma de las partes. Lograr beneficios reales de la distribución requiere trabajo y planeación” [22]

Por todo lo anterior es necesaria una actividad donde se analicé la distribución de la funcionalidad del sistema.

5.7.1 Definición de la configuración de red

Este paso tiene como objetivo entender la configuración y topología de la red, además de las capacidades y características de los procesadores y dispositivos que determinarán la naturaleza y grado de distribución posible en el sistema. La información que se requiere para esta tarea es la siguiente:

- La topología física de la red.

- Los nodos, capacidades de los mismos, hardware, software, número de procesadores y disco duro.
- Ancho de banda.
- El papel de internet (si aplica).

La figura 5.3 muestra un ejemplo con los nodos y conexiones de la vista de distribución del sistema de nomina que se explica en el anexo A.

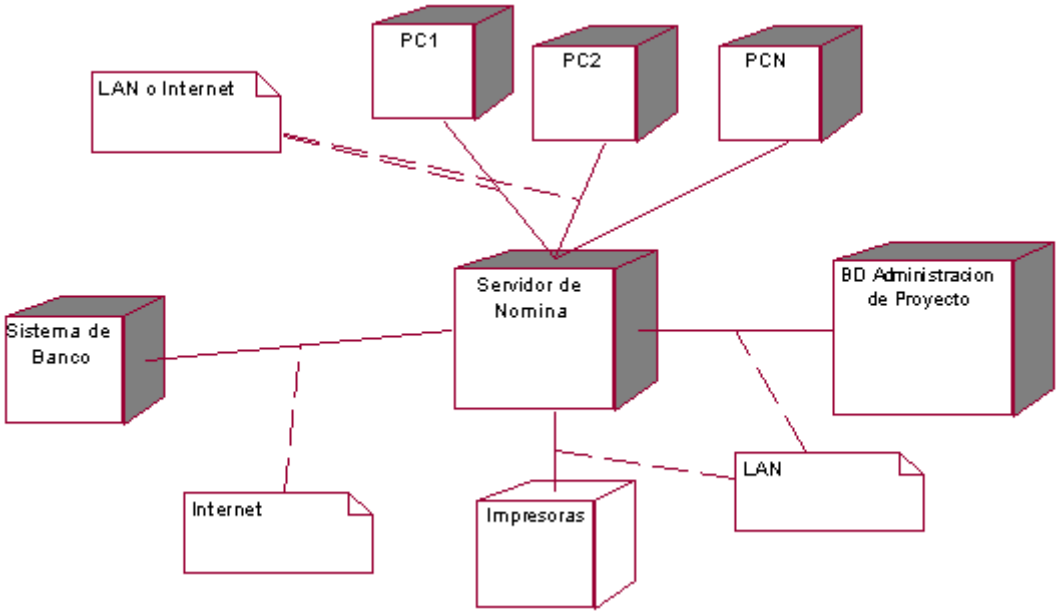

Figura 5.3. Vista de distribución del sistema de banco.

Para el ejemplo anterior los nodos de la red tienen las siguientes características:

- **PCs:** Procesador: Pentium 4 a 2.26Ghz; Disco duro de 40 G; Software: Internet Explorer; Procesadores:1.
- **Servidor:** Procesador: Intel Xeon 2.80 Ghz; Disco duro: 64 Tb; Cache: 2MB.
- **Impresora:** Tipo: Láser; Marca: Samsung; Modelo: ML1710.

Las conexiones tienen las siguientes características:

- **Tipo de Red:** LAN;
- **Velocidad y configuración:** Fast Ethernet a 100 Mbps.
- **Cable:** UTP Categoría 5.

5.7.2 Asignación de procesos a los nodos

En este punto se distribuye la carga de trabajo en los nodos según las necesidades y tomando en cuenta el trafico de la red. Los procesos que tienen mucha interacción se deben

colocar en un mismo nodo y los procesos que interactúan poco o nada se colocan en nodos diferentes como sea posible.

La asignación toma en cuenta los siguientes aspectos:

- Memoria y poder de procesamiento de cada nodo.
- Ancho de banda.
- Disponibilidad del Hardware.
- Requerimientos de redundancia y tolerancia a fallos.
- Requerimientos de tiempo de respuesta.

Los siguientes puntos muestran la asignación de procesos a los nodos de la figura 5.3 que corresponden al sistema de nomina del anexo A.

Los siguientes procesos corren en las PCs

- AplicaciónEmpleado

Los siguientes procesos corren en el servidor de nomina

- ControladordeProcesosdeNomina
- ControladordeProcesosdeTarjetas
- AccesoaBDdeAdministraciondeProyecto
- AccesoaSistemadeBanco
- AccesoaImpresora

5.8 Revisión de la arquitectura

Los siguientes puntos muestran el propósito de la actividad “Revisión de la arquitectura”:

- Descubrir errores que se perciben para cumplir con el tiempo y presupuesto estipulados
- Detectar defectos arquitectónicos de diseño
- Detectar incoherencias entre los requerimientos y la arquitectura, es decir, requerimientos no realistas
- Evaluar las cualidades arquitectónicas como confiabilidad, escalabilidad, seguridad, desempeño.

Los aspectos que tienen que ver con la arquitectura, por lo general son los que carecen de mediciones objetivas en el desarrollo de un sistema. Cuando se quiere medir alguna

cosa, es necesario tener referencias, así que una solución es tomar un sistema o versión anterior para comparar en ambos y así tener un dato que sea útil para analizar.

La revisión de la arquitectura se hace hasta este momento (al final de la fase de elaboración) porque ahora se tiene una arquitectura concreta, el cual no es el caso de los detalles de flujo de trabajo de "Definición de una arquitectura candidata" y mucho menos cuando se inicia con el "Análisis arquitectónico". Los resultados de la revisión se documentan en el artefacto "Registro de revisión".

5.8.1 Evaluación de la arquitectura

Para hacer una revisión de este tipo se pueden tomar varios enfoques, los cuales se explican en los siguientes puntos.

5.8.1.1 Revisión guiada por la representación de la arquitectura

Con este enfoque primero se obtiene o se construye una representación de la arquitectura y después se cuestionan los porqués de tal representación.

Al referirse a representación de la arquitectura hay varias opciones, tanto informales como formales, una forma es plasmarla en el documento de arquitectura de software ya que la vista lógica ofrece un panorama de las clases principales en el sistema, además la vista de procesos muestra los hilos de ejecución mas críticos del sistema, de esta forma cada vista da una forma distinta de la arquitectura del sistema

5.8.1.2 Revisión guiada por la información

En este enfoque primero se obtiene información sobre la arquitectura del sistema, las fuentes son variadas, se pueden hacer pruebas en diversos aspectos del software y usar los datos de ellas para razonar acerca de la arquitectura, este enfoque aplica principalmente para los aspectos que son mas fáciles de medir, tales como el rendimiento y el grado de tolerancia a fallos.

5.8.1.3 Revisión guiada por escenarios

En este enfoque primero se plantean una serie de escenarios y después se analizan los aspectos que se quieren medir de la arquitectura.

Los siguientes puntos muestran algunos puntos a evaluar con sus correspondientes escenarios:

- **Portabilidad:** El sistema corre en la plataforma A y en la plataforma B.
- **Extensibilidad:** Al sistema se le adiciona las siguientes funciones.
- **Usabilidad:** El sistema se instala tal cual en el site con el usuario final.

Este método permite encontrar varios errores porque es una forma de evaluación concreta y se pueden distinguir fácilmente aspectos difusos como requerimientos informales o no escritos.

5.8.2 Determinación de fallos

La determinación de los fallos potenciales se basa principalmente en la experiencia y el conocimiento, comúnmente los puntos débiles se vuelven patrones no escritos que se llevan de sistema en sistema y de organización en organización.

Los puntos deben catalogarse de forma objetiva, con base en los datos recabados anteriormente y priorizarlos para eliminarlos en el orden del más critico al menos.

Cuando se identifica el problema concreto, se determinan las causas y se plantean las formas de eliminarlo, afectando lo menos posible el estado actual.

Después de lo anterior se asignan las responsabilidades de los fallos con el fin de investigar y elaborar formas de mitigar o eliminar el riesgo.

Capítulo 6

Diseño de componentes

Introducción

Una de las bondades más difundidas de la tecnología orientada a objetos es la oportunidad de reuso, de echo, una de las llamadas “seis mejores practicas de la ingeniería de software” que implementa el RUP es el uso de arquitecturas basadas en componentes, sin embargo, esta promesa no viene por arte de magia, al contrario, se deben invertir muchos recursos al principio con beneficios a mediano y largo plazo ya que “se debe primero generalizar antes de particularizar”.

Es en este detalle de flujo de trabajo donde se diseñan los componentes que serán útiles en el proyecto actual y en proyectos a futuro, la figura 6.1 muestra en qué tiempo se realiza el diseño de componentes dentro del análisis y diseño.

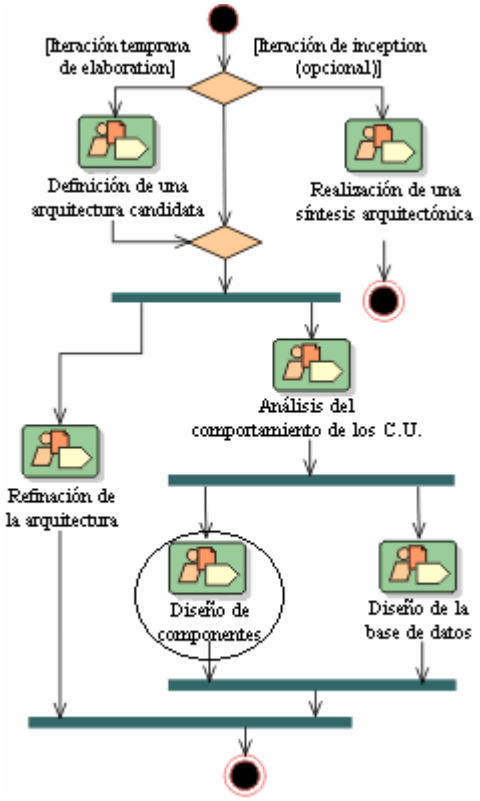

Figura 6.1. Detalle de flujo de trabajo “Diseño de componentes”.

Antes de iniciar con las actividades y artefactos, se define el termino componente, el cual es: “una pieza de software reusable,…un paquete de implementación de software que (a)puede ser de desarrollado independientemente, (b) tiene interfaces explicitas y bien especificadas para los servicios que provee y que necesita y (c) puede ser compuesto de otros paquetes o componentes” [5], entonces, se podría deducir que un componente seria una parte del rompecabezas llamado “sistema”, este tiene partes que hacen que el

componente se especialice, también tiene partes que hacen que “embone” con otros componentes para hacer uno mas grande y con mas servicios.

Ya que se discutieron algunos conceptos, se procede a explicar de manera general las actividades de este detalle de flujo de trabajo, que junto con los trabajadores y los artefactos se muestran en la figura 6.2.

En el diseño de componentes se reparten los elementos de diseño que se deben complementar, ya sea a individuos o equipos de trabajo, estos se hacen responsables por que los elementos estén en condiciones de su implementación ya que ahora se conoce el lenguaje de implementación y los algoritmos de las clases.

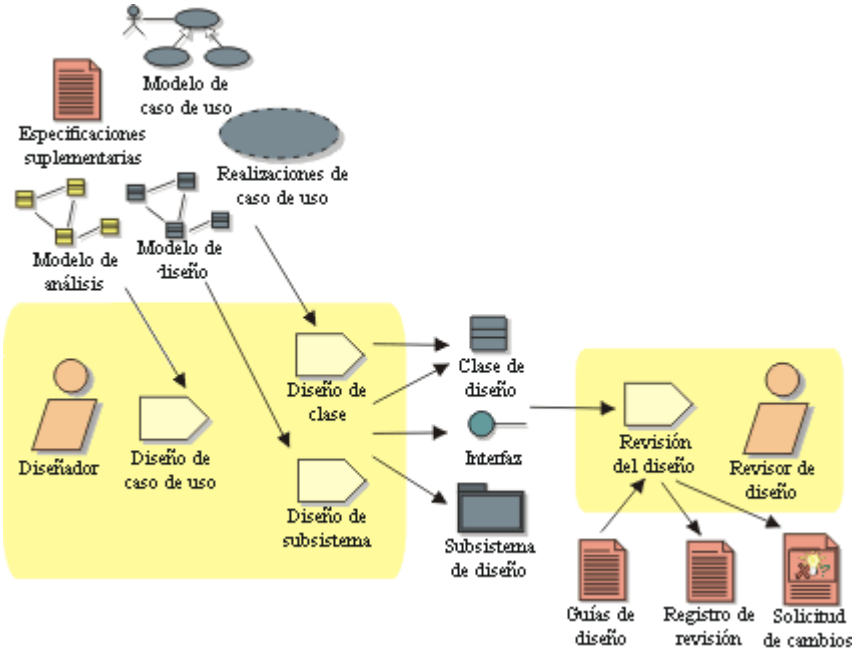

Figura 6.2. Actividades, Artefactos y roles del detalle de flujo de trabajo “Diseño de componentes”.

Como se puede indica en la figura 6.2, las actividades del diseño de componentes se organizan alrededor del tipo de elemento de diseño, el cual se enfoca en tres niveles: el nivel mas alto es el diseño de caso de uso porque es la actividad que une todos los elementos y los junta para analizar cómo colaboran para realizar los casos de uso, el segundo nivel corresponde a los componentes y subsistemas, que son abstracciones de varios servicios que debe prestar el sistema, y por ultimo las clases que se unen para realizar la funcionalidad del sistema.

6.1 Propósito del detalle de flujo de trabajo "Diseño de componentes"

Los siguientes puntos muestran los propósitos del diseño de los componentes.

- Refinar las definiciones de los elementos de diseño detallándolos para que posteriormente se implementen.
- Refinar y actualizar las realizaciones de caso de uso con base en los nuevos elementos de diseño.

6.2 Diseño de caso de uso

La primera actividad del detalle de flujo de trabajo de diseño de componentes es el diseño de caso de uso, su propósito es "transformar cada abstracción de negocio (clases de análisis) en una o mas clases de diseño que sean representaciones implementábles, tomando en cuenta las propiedades del ambiente de ejecución" [8]

El comportamiento de un sistema se describe por lo general con diagramas de interacción. La descripción en este caso se hace con diagramas de secuencia pero ya no con clases de análisis, sino con elementos de diseño.

El resultado de esta actividad produce cambios en casi todo el modelo de diseño. Las clases de análisis evolucionan en clases de diseño y las operaciones y atributos se especializan, las interacciones ya no son entre clases de análisis, sino entre elementos de diseño, y así se va completando el modelo para posteriormente ser la entrada principal de la disciplina de implementación. Los siguientes puntos muestran los pasos que se siguen en esta actividad.

6.2.1 Descripción de interacciones entre objetos de diseño

Por cada realización de caso de uso se crean diagramas de secuencia para ilustrar las interacciones de los objetos de diseño. El número de diagramas depende de la cantidad de flujos alternos y de la complejidad de los mismos.

El primer paso es identificar las clases de diseño que van a interactuar en la realización de caso de uso, también se deben estudiar los requisitos especiales y determinar qué clases llenan estos requisitos.

Posteriormente de realizan varios diagramas de interacción y preferentemente un diagrama de clases que represente las clases participantes en cada caso de uso y sus relaciones. En esta actividad se hace más énfasis en los diagramas de secuencia, debido a

que estos contienen las instancias y mensajes, además porque se puede tratar mejor la complejidad y la concepción del tiempo en que sucede algún evento determinado. En esta tarea es útil tomar como ayuda la realización de caso de uso del análisis.

Como en las realizaciones de caso de uso del análisis, se comienza por describir el flujo principal y después se siguen con los flujos alternativos por el grado de complejidad y por su importancia en el sistema.

6.2.2 Simplificación de los diagramas de secuencia con subsistemas

Los diagramas de secuencia a menudo pueden ser demasiado largos y complejos, lo anterior se puede simplificar por medio de subsistemas, para mejorar la comprensión de la solución, que es uno de los propósitos del diseño.

Este paso no solo ayuda a la simplificación, sino a encontrar patrones de interacción y además fomenta el reuso. Cuando se hace la simplificación, "subsecciones largas del diagrama de secuencia se reemplazan con un mensaje al subsistema que encapsula el comportamiento" [22]. La figura 6.3 muestra gráficamente cómo los sistemas pueden reducir la complejidad de los diagramas de secuencia.

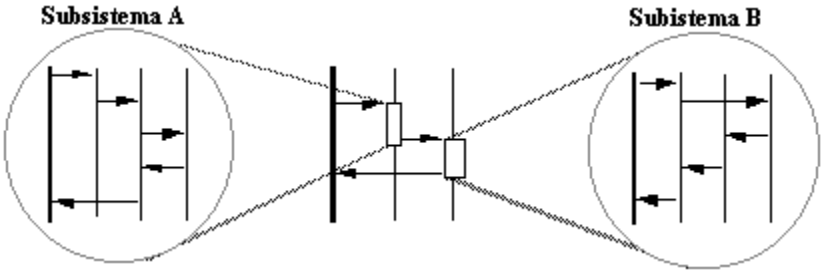

Figura 6.3. Los subsistemas simplifican los diagramas de secuencia.

Algunas partes de un diagrama de secuencia pueden encapsularse por un subsistema cuando ocurren los siguientes casos.

- Una parte del diagrama de secuencia se repite en varias realizaciones de caso de uso, es decir, mensajes iguales o similares se mandan a los mismos objetos (o similares) y estas interacciones producen un mismo resultado.
- Una subsecuencia ocurre en una sola realización de caso de uso, pero se espera que se repita el comportamiento en un futuro.
- El comportamiento es complejo y el encapsulamiento es fácil porque tiene un resultado bien definido. Puede ser que determinada parte de éste diagrama se deba desarrollar por un equipo o individuo.

6.2.3 Unificación de clases y subsistemas

Ya que se han realizado todos y cada uno de los casos de uso, es necesario unificar las clases y subsistemas para asegurarse de la homogeneidad y consistencia en el modelo, debido a que el trabajo se realiza por varios equipos.

Los puntos a considerar son:

- Los nombres de los elementos de modelo deben describir su función.
- Se deben evitar nombres similares o sinónimos.
- Se deben unificar los elementos de modelo que definen comportamiento similar o que representan la misma abstracción.
- Se deben unir aquellas clases de entidad que representan el mismo concepto o que tienen los mismos atributos, aun si definen un comportamiento diferente, porque se caería en redundancia de información.
- Se debe usar la herencia para abstraer elementos de modelo que denotan jerarquía o atributos iguales, ya que ello tiende a hacer el modelo más robusto.
- Cuando se está actualizando un elemento, se debe hacer lo mismo con los flujos de eventos correspondientes de las realizaciones de caso de uso, es decir cuando se cambia el nombre de una clase o se unifican varias clases, también se deben actualizar las realizaciones de caso de uso que tengan que ver con estos elementos de modelo.

6.3 Diseño de subsistemas

El propósito de la actividad “Diseño de subsistemas” es:

- Definir el comportamiento especificado por las interfaces de los subsistemas en términos de colaboraciones de clases.
- Documentar la estructura interna de cada subsistema
- Definir las realizaciones entre las interfaces de subsistemas y las clases que contienen los subsistemas
- Determinar dependencias entre subsistemas.

Los subsistemas son los elementos que permiten administrar la complejidad y repartir el trabajo a los equipos de desarrollo, los siguientes puntos muestran los pasos para su diseño.

6.3.1 Distribución del comportamiento en el subsistema

Este paso se inicia especificando el comportamiento interno de cada subsistema, además se identifican las nuevas clases que se necesitan para satisfacer todos los requerimientos, aun los no funcionales.

Una vez que la interfaz de un subsistema define algún comportamiento, esto se convierte en un contrato para que las partes internas realicen este conjunto de operaciones que ofrece tal interfaz. Las operaciones de una interfaz deben ser realizadas por los elementos internos que contiene el subsistema, que por lo general son las mismas operaciones de las clases internas.

Las colaboraciones de los elementos de modelo se deben documentar usando diagramas de secuencia. Cada operación de una interfaz realizada por un subsistema debe tener uno o más diagramas de secuencia. Este diagrama se asocia al subsistema y se usa para diseñar el comportamiento interno.

Los diagramas de secuencia clarifican los puntos desconocidos sobre cómo le va hacer el subsistema para realizar el comportamiento al que esta comprometido, ya que muestra las clases internas en colaboración y cuáles son las que mandan mensajes fuera del subsistema.

Por cada operación de la interfaz, se busca alguna clase o en el caso de una operación compleja un subsistema interno que la realice. Si no existen clases o subsistemas entonces se crean.

Se debe ser cuidadoso de no tener dos clases iguales en subsistemas diferentes, lo anterior implica que las fronteras de los subsistemas no se definieron del todo bien.

6.3.2 Documentación de los elementos del subsistema

Para hacer este paso se crean uno o mas diagramas que muestren los elementos contenidos en el subsistema y las asociaciones entre ellos. El que exista más de un diagrama de clases para algún subsistema es señal de que éste es muy complejo y requiere una división para mejorar la lectura del diagrama.

La descripción de las clases se trata con mas detalle en la actividad “Diseño de clase”. En la siguiente figura se muestra el ejemplo del diagrama de clases para un subsistema de orden de productos de un cliente.

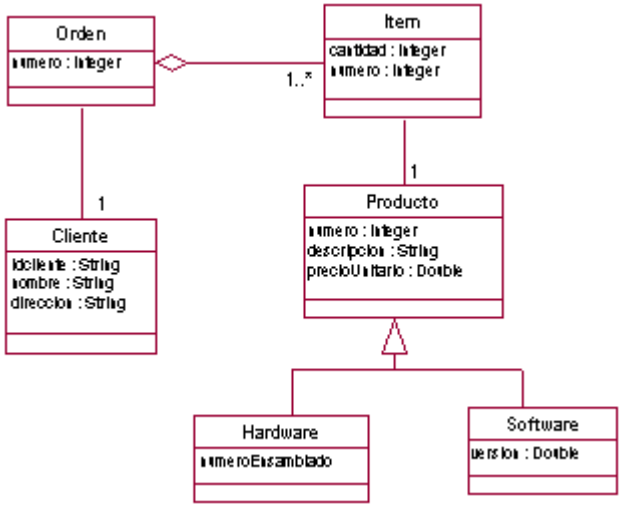

Figura 6.4. Diagrama de clases para subsistema de ordenes de cliente.

6.3.3 Descripción de dependencias entre subsistemas

Cuando un subsistema utiliza algún tipo de comportamiento de otro, se crea una relación de dependencia entre las interfaces de ambos. Para mejorar el reuso, se hace una asociación de dependencia a la interfaz y no sobre el subsistema en si.

La razón para que las dependencias se hagan a las interfaces es por la mantenibilidad, ya que de esta manera se puede cambiar un subsistema por otro siempre y cuando ambos tengan la misma interfaz. Otra razón es dar libertad al diseñador de desarrollar las partes internas, siempre y cuando se cumpla con el comportamiento externo que ofrece la interfaz.

La figura 6.5 muestra las dependencias entre subsistemas y paquetes (sin interfaces) del sistema de nomina del anexo A.

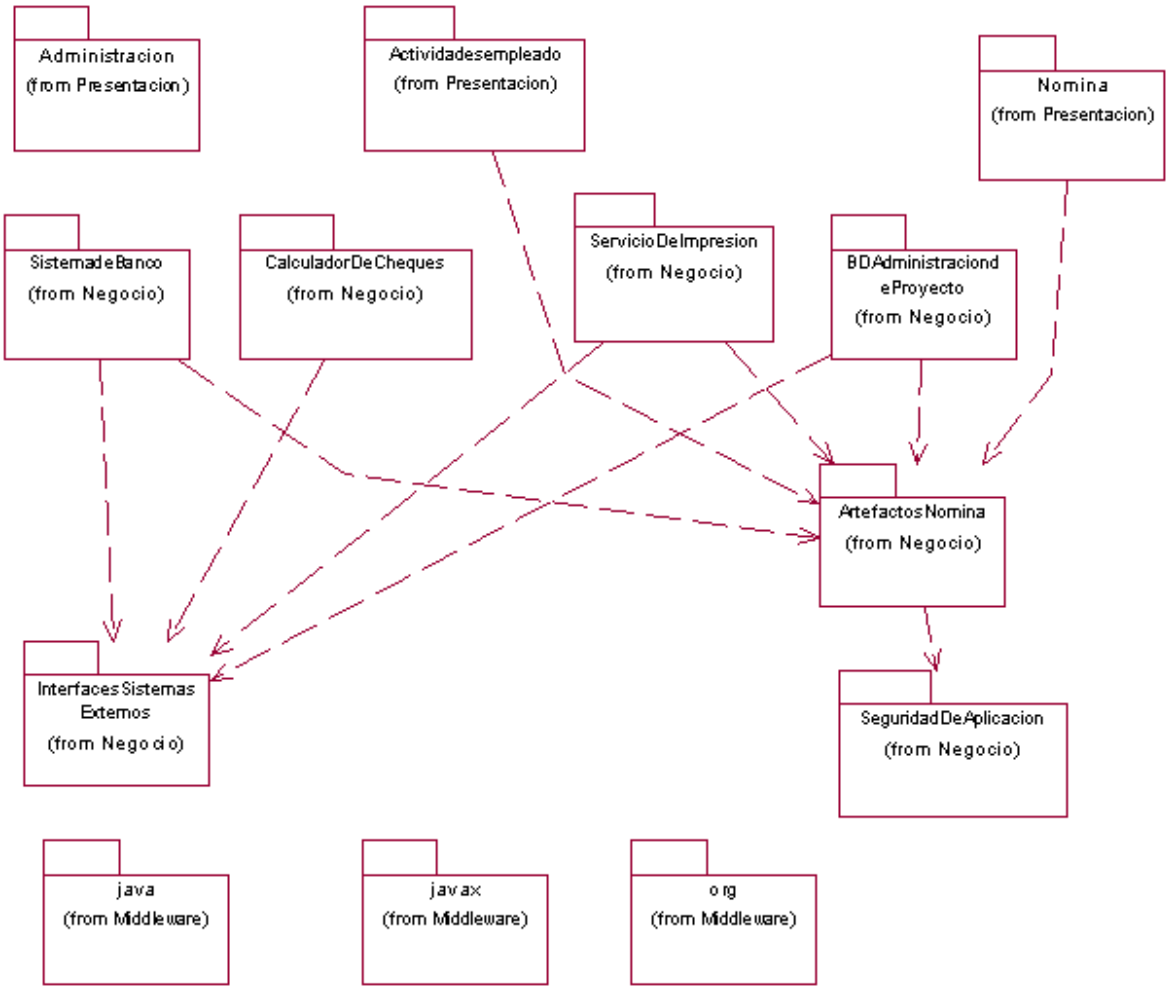

Figura 6.5. Dependencias entre subsistemas y paquetes.

"En subsistemas donde hay probabilidad de substitución es mejor que las dependencias se hagan a la interfaz misma que es realizada por el subsistema. Esto permite usar cualquier elemento de modelo siempre y cuando realice la misma interfaz." [22]

La figura 6.6 muestra un ejemplo en donde SistemadeBanco, CalculadorDeCheques, ServicioDeImpresion y BDadministraciondeProyecto tienen una dependencia a la interfaz del subsistema ArtefactosNomina llamada IArtefactosNomina.

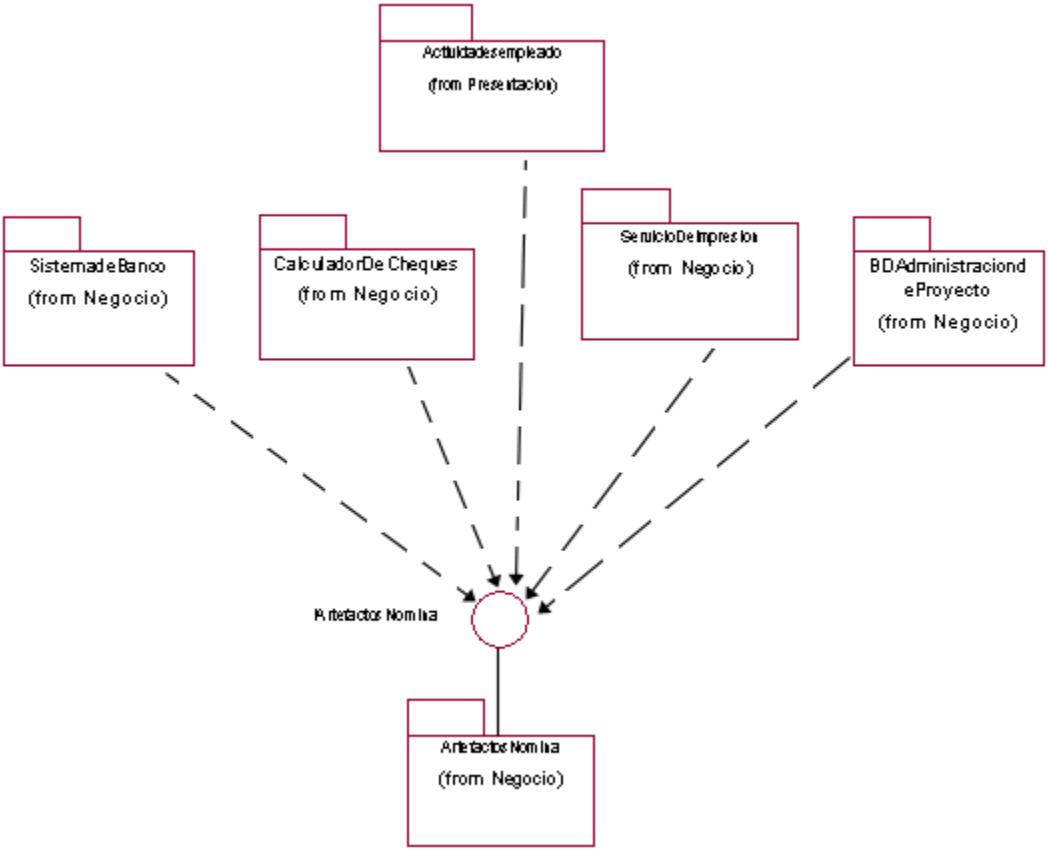

Figura 6.6. Ejemplo de dependencias a una interfaz de subsistema.

6.4 Diseño de clase

El propósito de la actividad “Diseño de clase” es:

- Asegurarse que la clase provee el comportamiento que requiere la realización de caso de uso.
- Asegurarse que se hay suficiente información para implementar la clase.
- Tratar requerimientos no funcionales que se relacionan con la clase.
- Incorporar los mecanismos de diseño que usa la clase.

Hasta este punto se han identificado la mayoría de las clases durante la actividad “Identificación de los elementos de diseño”. También se identificaron clases adicionales durante la actividad “Diseño de subsistema”. Aquí se detallan todas las clases de diseño encontradas anteriormente ya que hasta ahora se ha estabilizado la estructura del modelo de diseño.

Esta actividad es muy importante ya que las clases son la materia prima del diseño, los otros elementos tales como subsistemas, paquetes y colaboraciones solo describen cómo se

agrupan y operan las clases. Los siguientes puntos muestran los pasos a seguir para esta actividad.

6.4.1 Identificación de clases persistentes

A las clases que necesitan la capacidad de mantener su estado en un medio de almacenamiento permanente se les llama “clases persistentes”. Las necesidades de estas pueden tener varios orígenes tal como: necesidad de datos por un largo tiempo, elaboración de reportes, respaldos en caso de fallas en el sistema o para intercambio de información. Lo anterior no significa que todas las instancias de la clase deban ser persistentes, solo pueden ser algunas de ellas, pero a estas se les debe dar la capacidad de almacenamiento permanente.

La tarea de identificar clases persistentes sirve para notificar al diseñador de base de datos que una determinada clase requiere especial atención en cuanto a las características de almacenamiento físico, entonces se le da la responsabilidad al diseñador de base de datos de encontrar los mecanismos persistentes para que los objetos suplan sus necesidades. Esta responsabilidad por lo general deriva en la tarea de mapear las clases persistentes a la Base de Datos y así poder almacenar determinadas instancias de la clase.

6.4.2 Definición de visibilidad

“La visibilidad de una clase se puede asignar de acuerdo al paquete de diseño en el que reside la misma. Se debe tener en cuenta que los elementos dentro de un subsistema de diseño son implícitamente privados y por lo tanto no necesita ser declarada su visibilidad” [6]

El significado de que una clase sea “publica” es que puede ser referenciada fuera del paquete que la contiene, una clase “privada” puede ser referenciada solo por clases dentro del mismo paquete.

6.4.3 Definición de operaciones

Para esta tarea lo primero que se hace es identificar las operaciones de cada clase, esto se realiza estudiando las responsabilidades de la correspondiente clase de análisis, después se crea una operación por cada responsabilidad. Otra fuente para identificar operaciones son las realizaciones de caso de uso en las que participa la clase, ya que las operaciones son el medio de soporte de los mensajes de los diagramas de secuencia.

El párrafo anterior muestra las fuentes principales para la identificación de operaciones que realizan la funcionalidad. El siguiente paso es identificar las operaciones necesarias para creación, eliminación, copia, prueba o comparación de objetos y la identificación de clases que soporten los requerimientos no-funcionales.

Para nombrar una operación se debe tener en cuenta las reglas y tipos del lenguaje de implementación.

El siguiente paso es identificar la visibilidad de cada operación ya sea publica (la operación es visible fuera de la clase), protegida (la operación es visible en la clase o en sus subclases) o privada (la operación solo es visible en la misma clase). Es posible que el lenguaje tenga otras opciones, se debe de acoplar a la mejor elección de visibilidad.

6.4.4 Definición de métodos

"Un método especifica la implementación de una operación" [22]. En muchos casos los métodos se implementan directamente en el lenguaje de programación, estos son los casos en los que el comportamiento requerido por la operación se define completamente con el nombre de la operación, descripción y parámetros. Sin embargo, hay veces en las que se requiere de un algoritmo específico, o se requiere mas información de la que se presenta en la descripción de la operación, en estos casos se requiere una descripción separada para cubrir estos detalles. Los siguientes puntos muestran que aspectos debe describir un método.

- Modo de implementación del método en el lenguaje de programación.
- Modo de implementación de atributos y operaciones.
- Modo de implementación de las relaciones.

6.4.5 Definición de atributos

En este paso se detallan los atributos que se han definido anteriormente y se definen aquellos que sean necesarios para realizar las operaciones. Para cada atributo se debe definir lo siguiente:

- **Nombre de atributo:** El nombre debe reflejar las reglas del proyecto y del lenguaje de programación.
- **Tipo:** El tipo debe ser un tipo elemental que soporte el lenguaje de implementación.

- **Visibilidad:** La visibilidad puede ser pública privada o protegida.
    - **Publica:** El atributo es visible dentro y fuera del paquete que contiene la clase.
    - **Privada:** El atributo solo es visible dentro de la clase y a clases “friend” asociadas
    - **Protegida:** El atributo es visible dentro de la clase y en las subclases.

6.4.6 Definición de dependencias

Una dependencia es un tipo de asociación entre dos clases en donde una clase “cliente” necesita hacer uso de otra clase “origen”, cuando se habla de hacer uso, quiere decir que la clase cliente necesita algún tipo de información o alguna característica de la clase origen. Las dependencias son transitorias, es decir, existen por un tiempo limitado por lo que no se considera una relación estructural entre clases. Una clase A es dependiente de la clase B por alguna de las siguientes razones:

- Un objeto de la clase B se pasa como parámetro a una operación de la clase A.
- Un objeto de la clase B se pasa como un valor de retorno de una operación de la clase A.
- Una instancia de la clase B se usa en la implementación de alguna de las operaciones de la clase A.

6.4.7 Definición de asociaciones

“Las asociaciones proveen los mecanismos para la comunicación entre objetos” [22], a diferencia de las dependencias, una asociación es una relación entre dos clases que **SI** implica una relación estructural.

Las asociaciones se forman desde de los diagramas de secuencia y de colaboración del modelo de análisis. A una asociación se les puede dar varios atributos, es decir, información adicional a la semántica de la misma, los siguientes puntos muestran las formas en que se refina una asociación.

- **Definición de roles:** Esta información especifica el papel que juega cada una de las dos clases en el contexto de la relación.
- **Definición de multiplicidad:** Se refiere a cuántas instancias de una clase pueden estar asociadas con una instancia de otra clase.

- **Navegabilidad:** Indica si la asociación es unidireccional o bidireccional.

La figura 6.7 muestra la ventana para especificar las características anteriores de una asociación en Rational Rose. Dependiendo de las características de la asociación, ésta puede ser una agregación, una asociación unidireccional o bidireccional

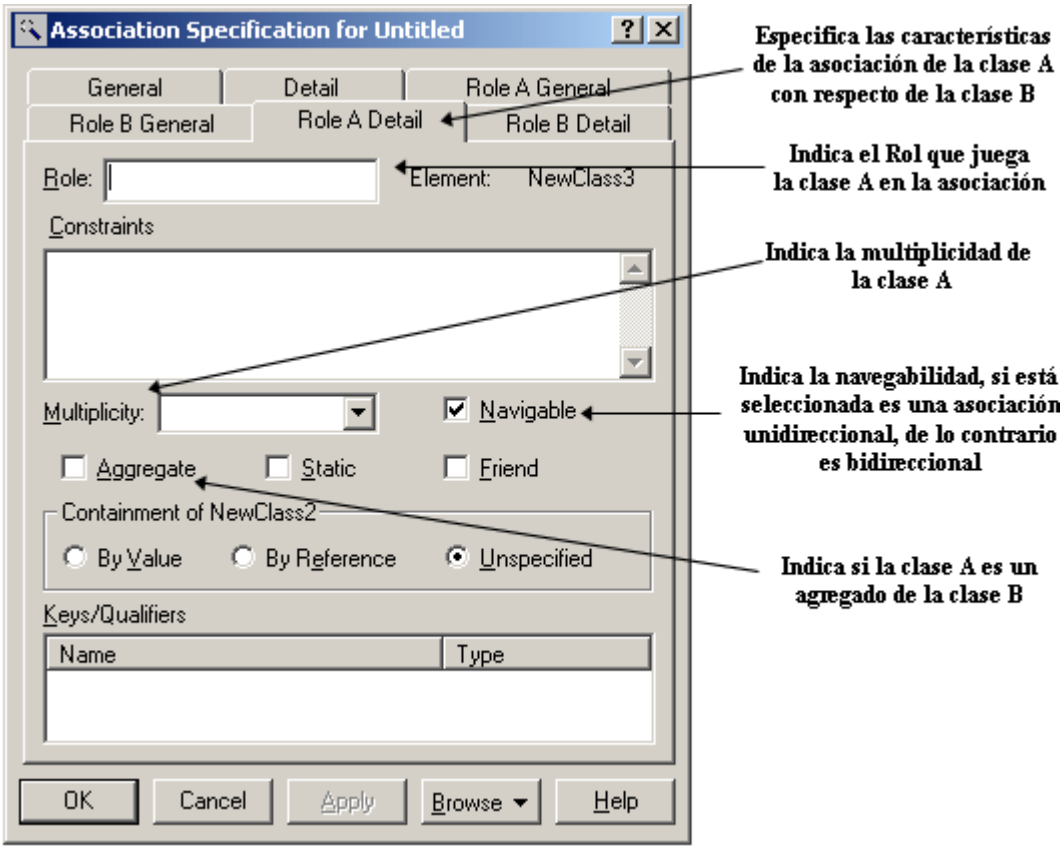

Figura 6.7. Ventana de especificación de una asociación en Rational Rose.

Hay varios tipos de asociaciones, en los siguientes párrafos se explican las características de cada una, sin embargo la característica principal es la relación de ambas clases, las divisiones son para particularizar y clarificar mejor el significado de la relación.

Una **asociación bi-direccional** indica que el flujo de comunicación o de mensajes va en ambas direcciones, es decir, los mensajes pueden ir de la clase “A” a la clase “B” y viceversa, este tipo de asociación se representa por medio de una línea sólida uniendo las clases asociadas.

En el caso de una **asociación unidireccional** los mensajes solo pueden ir en una dirección. Las asociaciones unidireccionales se representan en UML con una flecha entre las clases de la asociación para indicar el sentido de la misma.

La figura 6.8 muestra un ejemplo de asociación unidireccional entre las clases "ReporteCuentasSobregiradas" y "CuentasdeBanco".

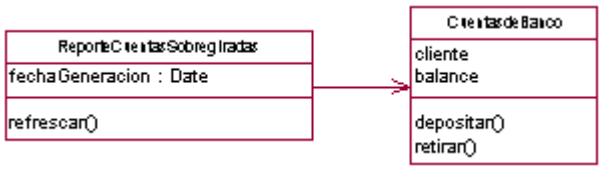

Figura 6.8. Asociación unidireccional.

Una **agregación** es un tipo especial de asociación usada para modelar un todo en sus partes. Existen dos tipos de agregación: La composición y la agregación básica.

La **agregación básica** indica que una clase es subordinada de otra clase. En una relación de agregación una instancia de la clase "hija" puede sobrevivir aun si la instancia de clase padre se destruye. Para representar una relación de agregación básica, se debe trazar una línea de la clase padre a la clase subordinada y al final de la clase padre se coloca un rombo sin color de relleno. La figura 6.9 muestra la representación en UML de una agregación básica entre la clase compañía y la clase departamento.

Figura 6.9. Agregación básica.

La **composición** es otra forma de agregación en donde el ciclo de vida de la instancia de una clase "hija" depende del ciclo de vida de la instancia de la clase "padre", es decir, si se destruye el objeto de la clase padre, también se destruye el objeto de la clase hija, no puede vivir el uno sin el otro. En UML una composición entre clases se representa como una línea de la clase padre a la clase hija y un rombo con color de relleno negro al final de la línea de la clase padre. La figura 6.10 muestra la representación en UML de una composición entre las clases Estudiante y Horario.

Figura 6.10. Composición.

6.4.8 Definición de generalizaciones

La herencia es una de las principales características de la filosofía orientada a objetos, es muy poderosa porque es un mecanismo que permite el reuso de comportamiento (operaciones) y de estructura (atributos)

La generalización es una relación entre un elemento general y uno o más elementos específicos, esta utiliza la herencia como mecanismo para que una clase “específica” pueda heredar (tomar) todos y cada una de las operaciones y atributos de la clase “general”. Cuando se está hablando de generalización, a la clase general se le llama “superclase” y a la clase específica se le llama “subclase”

En UML, una generalización se representa como una línea que va de la superclase a la subclase, con una flecha cerrada (o triangulo) apuntando a la superclase. La figura 6.11 muestra una generalización entre las clases CuentadeBanco, CuentadeAhorros y CuentadeCheques.

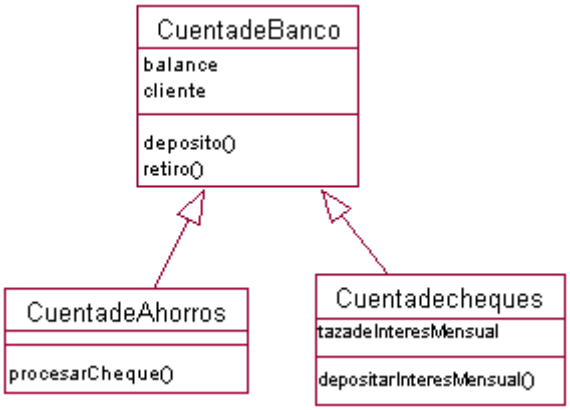

Figura 6.11. Ejemplo de Generalización.

6.4.8.1 Procedimiento para la generalización

“Las clases se pueden organizar en una jerarquía para reflejar comportamiento (operaciones) y estructuras (atributos) comunes. Se puede definir una “superclase” común, de la cual las subclases puedan heredar comportamiento y estructura” [6]

Lo primero que se hace es encontrar las clases que tengan comportamiento y/o estructura comunes, después se crea una clase, la cual será la “superclase” y contendrá los elementos generales, el nombre de esta debe denotar la jerarquía o los elementos generales que contiene. Después se colocan los atributos y operaciones que son generales en la superclase y se eliminan estos mismos atributos y operaciones de las subclases, por ultimo se define una relación de generalización entre cada una de las subclases y la superclase.

6.5 Artifacts principales

En esta sección se explica el artifact “Guías de diseño” el cual es uno de los mas importantes durante la etapa de diseño, a continuación se muestra la estructura que debe contener.

6.5.1 Guías de diseño

Las guías de diseño son un producto de la definición de la arquitectura. Mucha gente dentro del desarrollo puede necesitar de este documento porque describe las pautas a seguir durante el diseño y la implementación.

Cuando se crea este documento, una entrada importante es el ambiente de implementación en el cual entra la plataforma (hardware y sistema operativo), las herramientas de desarrollo (lenguaje de programación), el DBMS (Sistema Administrador de Base de Datos) y la librería de componentes existentes.

Los siguientes puntos muestran las partes importantes de las guías de diseño.

6.5.1.1 Introducción

Esta parte contiene un resumen de todo el documento, incluye el propósito, limites, definiciones, acrónimos, abreviaturas y referencias.

6.5.1.2 Guías generales de diseño e implementación

Esta sección describe los principios y estrategias que se usarán para el diseño e implementación del sistema, los siguientes puntos muestran los aspectos principales de las guías generales.

- **Documentación de operaciones:** Es importante formular un estándar para describir operaciones. Esta descripción consiste de un nombre, argumentos, una breve descripción y una especificación de implementación.
- **Documentación de mensajes:** En este punto es recomendable no documentar todos los parámetros actuales de los mensajes ya que resulta redundante con el código y es difícil de mantenerse.
- **Detección, manejo y reporte de fallos:** En este punto se formula una estrategia para la administración de fallos la cual depende en gran manera del lenguaje de programación seleccionado.

(continuación de las guías generales de diseño e implementación)

- **Administración de memoria:** En este punto se forman estrategias para asegurarse de que la memoria estará siempre disponible, esto significa remover objetos que no están referenciados de tal manera que el espacio se pueda ocupar.
- **Distribución del software:** En caso de un sistema distribuido, antes del diseño se debe preparar esta parte del documento para especificar estrategias generales para la distribución de objetos entre los nodos.
- **Administración de transacciones:** En esta sección se discuten las estrategias para realizar la administración de transacciones.
- **Características especiales del lenguaje:** En esta sección se discuten las políticas y restricciones que tiene el lenguaje de implementación.
- **Guías de algoritmos:** En esta sección se discuten los algoritmos particulares que usará el sistema y el razonamiento en la selección de los algoritmos más importantes.
- **Interfaz con el hardware:** En esta sección se discuten las interfaces con el hardware incluyendo el uso de interrupciones, memoria, etc.

6.5.1.3 Guías de la base de datos

Las guías de base de datos es una sección del documento donde se discuten las reglas y recomendaciones para el diseño de la B.D. Los siguientes puntos muestran los aspectos principales.

- Estrategias de mapeo de clases persistentes a estructuras de datos incluyendo la resolución de conflictos como asociaciones muchos a muchos en el modelo de diseño y la herencia.
- Estrategias de mapeo de atributos de la clase de diseño a tipos de datos primitivos de la base de datos.
- El uso de la vista de distribución para describir la distribución física de los datos en los nodos.

Capítulo 7

Diseño de la base de datos

Los futuros usuarios de grandes bancos de datos ya no tendrán que conocer cómo se organizan los datos en la maquina (la representación interna).

E. F. CODD, 1970 - A Relational Model of Data for Large Shared Data Banks.

Introducción

Después o a la par del diseño de los componentes, ya que se han identificado las clases persistentes se inicia con el diseño de la base de datos, una parte medular principalmente en las aplicaciones administrativas, el RUP la pone como una actividad opcional en el caso de que el sistema no contenga datos persistentes, pero, actualmente hay muy pocos sistemas con esta cualidad.

La persistencia debe ser tratada como una parte integral del esfuerzo de diseño y la colaboración estrecha entre los diseñadores de la base de datos y los diseñadores de los componentes normales es esencial. Es por esto que en la figura 7.1 se muestran ambas actividades realizándose concurrentemente.

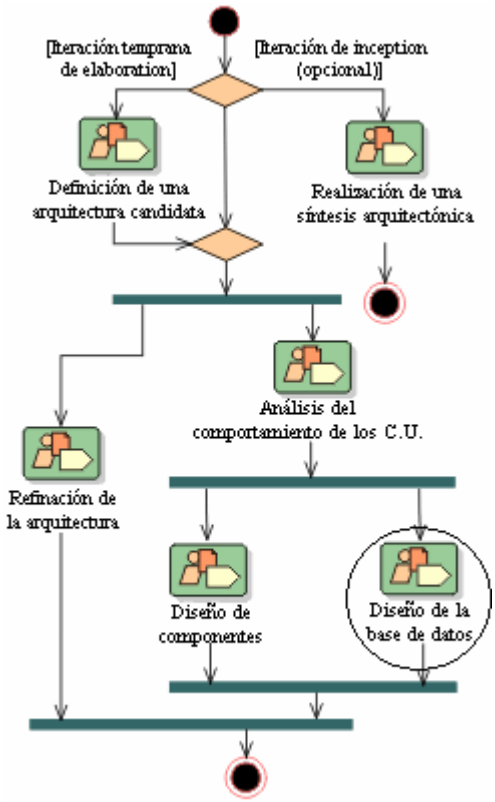

Figura 7.1. Detalle de flujo de trabajo “Diseño de la base de datos”.

La figura 7.2 muestra el detalle de flujo de trabajo del diseño de base de datos, en ella se indican dos actividades fundamentales que son el “Diseño de clase”, “Diseño de la base de datos”. La actividad “diseño de clase” del capítulo anterior tiene mucho que ver con este detalle de flujo de trabajo porque cuando se diseñan las clases, se identifican aquellas que son persistentes y es donde radica la responsabilidad de comunicación del diseñador de componentes con el diseñador de la base de datos.

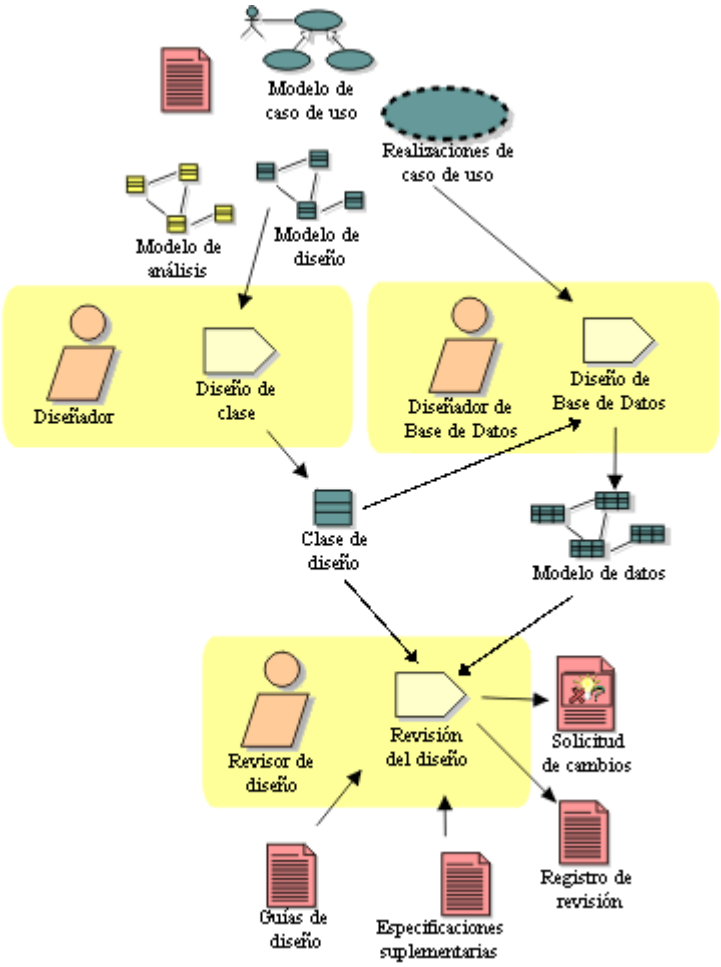

Figura 7.2. Resumen del detalle de flujo de trabajo “Diseño de la base de datos”.

Ya que se han identificado las clases persistentes, se toman las características de cada una de ellas (Artefacto Clase de diseño) y además se toman las relaciones entre las clases (Artefacto Modelo de diseño), después se hace un mapeo para realizar el modelo de datos y por ultimo se optimiza la base de datos para tener un rendimiento óptimo.

7.1 Propósito del detalle de flujo de trabajo “Diseño de la B.D.”

El propósito de este detalle de flujo de trabajo es diseñar estructuras de base de datos apropiadas para almacenar las clases persistentes, además de definir los mecanismos y estrategias para almacenar y recuperar los datos tomando en cuenta un criterio de rendimiento idóneo.

7.2 Conceptos preliminares

A continuación se describen algunos conceptos preliminares que son necesarios para comprender el diseño de la base de datos.

7.2.1 El modelo relacional y el modelo de objetos

Esta sección trata las características de las bases de datos relacionales y cómo es que se puede unir esta tecnología con el modelo de objetos. Se puede preguntar el porqué no se trata en éste proceso a las bases de datos orientadas a objetos, la respuesta es que la mayoría de los datos que están en una BD se encuentran en forma relacional, además de que hay más soporte para esta tecnología por ser mas conocida y madura, es por estas razones que se ha utilizado la tecnología de bases de datos relacionales.

7.2.1.1 El modelo relacional

La manera de lograr la persistencia de la información en un sistema por lo general es una Base de datos, su definición es la siguiente: "**Una B.D. es una colección cohesiva de datos, la cohesividad o nivel de asociación de sus elementos refleja una representación de una organización o unidad en el mundo real**".

La representación de una base de datos y sus relaciones generalmente se dividen en dos grandes formas: el modelo relacional y el modelo orientado a objetos, el RUP en su versión 2002 toma como forma principal para el proceso de diseño al modelo relacional el cual se explica a continuación.

El modelo relacional se compone de entidades y relaciones. **Una entidad es la abstracción de un conjunto de cosas (objetos, personas,…)** que comparten las mismas características y participan en las mismas relaciones, el concepto se parece mucho al de una clase a excepción del comportamiento. En el lenguaje de bases de datos relacionales a las características de una entidad se le llaman columnas.

"Una entidad tiene registros o filas. Cada registro representa un conjunto único de información que muy frecuentemente representa los datos persistentes de un objeto" [22]. Las entidades se implementan como tablas por lo que aun en el modelo relacional así se les llama.

Por otro lado **una relación es una asociación entre dos entidades**. Las relaciones también pueden tener multiplicidad o cardinalidad.

La figura 7.3 muestra un modelo relacional (sin laves) entre las entidades alumno y calificación, en ésta se puede observar por la cardinalidad, que un alumno puede tener muchas calificaciones.

Figura 7.3. Modelo relacional.

Sin embargo el modelo aun no esta completo, falta agregar "llaves" que son las que implementan las relaciones entre las tablas. Existen dos tipos de llaves: primarias y foráneas. Una **llave primaria** es aquella columna que **sirve para identificar de manera única los registros o filas de la tabla.** Si se quiere implementar la relación de dos tablas, entonces se tiene que poner la llave primaria de la tabla que participa con "1" instancia, como una columna mas de la tabla que participa con "n" instancias en la relación, a ésta columna se le llama llave foránea, en la figura 7.4 se muestra el modelo conceptual con llaves foráneas y primarias; las llaves foráneas son las que tienen el prefijo (FK) y una llave en la parte izquierda de la columna, las demás son llaves primarias.

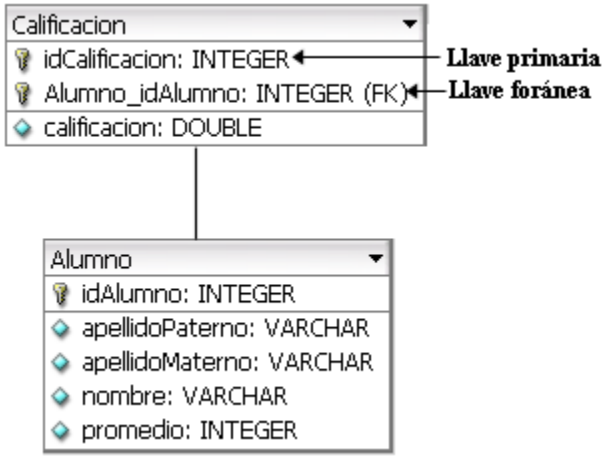

Figura 7.4: Llaves primarias y foráneas

7.2.1.2 El modelo de objetos

El modelo de objetos contiene muchos elementos, uno de los más importantes es la clase, una clase define la estructura y el comportamiento de un conjunto de objetos. La estructura se representa como atributos (valores de datos) y asociaciones (relaciones entre clases) y el comportamiento por medio de operaciones o métodos.

Los conceptos más importantes del modelo de objetos son la abstracción, el encapsulamiento y la herencia. "La abstracción y el encapsulamiento son los componentes más importantes del paradigma orientado a objetos. Usando ambos se puede ofrecer al

mundo exterior la vista de un objeto que presenta solo las características y comportamiento esenciales, independientes de su implementación" [10]

La abstracción permite modelar un objeto extrayendo las características más importantes (en el contexto del modelado) y eliminando las características menos importantes. El encapsulamiento, a veces llamado ocultamiento de información permite a un objeto comunicarse con otro si saber como está trabajando internamente, tal como una caja negra. Por ultimo la herencia permite formar una jerarquía de objetos en donde el criterio de esta jerarquía es el nivel de generalidad o particularidad del mismo. La herencia permite que una clase específica "tome" comportamiento y estructura de una clase mas general, lo que permite entre otras cosas la reusabilidad.

7.2.1.3 Unión del modelo relacional y el modelo de objetos

Los dos puntos anteriores muestran que ambas formas de ver el mundo tienen diferencias y similitudes. Esto se debe a que UML es un mejoramiento del modelo relacional, de ahí vienen las semejanzas y las diferencias.

No es que una perspectiva sea mejor que la otra, ya que cada una funciona bien en el ambiente que fue concebido; el modelo orientado a objetos tiende a trabajar bien en sistemas con comportamiento complejo en que los datos son secundarios. El modelo RDBMS es bueno para aplicaciones de reportes y sistemas administrativos.

Las similitudes son fáciles de unir, sin embargo se tiene que hacer un trabajo importante para resolver las diferencias, parte de este trabajo se realiza haciendo un mapeo del modelo de objetos al modelo relacional que sirve para convertir un conjunto de clases y asociaciones en un modelo relacional con entidades, relaciones y llaves.

7.2.2 Mapeo del modelo de objetos a un modelo de datos relacional[1]

En los últimos años el modelo relacional ha sido el más popular para el diseño conceptual de bases de datos y UML es ampliamente usado en el mundo del análisis y diseño orientado a objetos, pero a pesar del dominio de las técnicas orientadas a objetos durante el diseño de software y la fase de su desarrollo, las bases de datos orientadas a objetos todavía no son de uso generalizado ya que los diseñadores frecuentemente vuelven a las bases de datos relacionales para soportan una gran parte de la persistencia que requiere

[1] La información de este punto fue sacada del libro "Unified Modeling Language: Systems Analysis, Design and Development Issues", del capítulo modelado de datos y UML, se encuentra en la bibliografía como [26]

el software. Lo anterior suma una tarea en la lista de cosas por hacer: Mapear las clases y sus asociaciones a entidades y relaciones.

Trasladar un diagrama de clases al modelo relacional puede ser una tarea difícil, ya que en el diagrama de clases se usan muchos símbolos y notaciones (por ejemplo la generalización) que no mapean directamente en el modelo relacional.

El diseño de una base de datos inicia con un diagrama conceptual, en el caso del RUP el diagrama conceptual es el modelo de objetos. Después hay que trasladar el modelo conceptual a un modelo lógico que es una abstracción de las tablas y relaciones de la base de datos. El traslado en este caso se realiza haciendo un mapeo de los elementos del modelo de objetos.

Los siguientes puntos revisan las diferentes construcciones de UML/diagrama de clases y discute un proceso que puede usarse para trasladar estas construcciones en la vista lógica.

7.2.3 Mapeo de clases

Una clase es un descriptor para un conjunto de objetos con estructura, comportamiento y relaciones similares. Representa un concepto dentro del sistema que se está modelando.

En el mapeo se puede crear una tabla por cada clase persistente. Sin embargo las cosas se pueden complicar cuando se considera la generalización y asociaciones con otras clases, por eso se deben considerar tales aspectos.

7.2.4 Mapeo de atributos

Un atributo muestra las propiedades que tiene la clase. Los atributos pueden mapearse a cero, una o más columnas, los siguientes puntos muestran la forma de mapear atributos de acuerdo a sus propiedades.

- **Visibilidad:** A un nivel de BD todas las columnas de una tabla son publicas, por lo tanto cuando se mapéan los atributos, esta propiedad se ignora.
- **Multiplicidad:** La multiplicidad indica el “limite inferior…limite superior”. Por ejemplo, una multiplicidad 0...1 denota un atributo no-obligatorio y de un solo valor, mientras que la multiplicidad 1...* denota un atributo como obligatorio y de múltiples valores. Una propiedad, ya sea obligatoria o no obligatoria puede ser trasladada como una propiedad nulo/no nulo respectivamente. Por otro lado no es

fácil trasladar un atributo de múltiples valores porque es imposible almacenar múltiples valores en una columna.

- Si el **numero de valores de un atributo multi-valor** es constante, se puede crear una columna por cada atributo. Si el número de valores varía, la mejor manera es crear una tabla separada para almacenar los valores y ligarlos a la tabla original usando una llave foránea.
- **Propiedad-"String":** La sintaxis de UML usa la propiedad "String" para denotar un tipo de atributo tipo cadena de caractéres. La tarea de trasladar este tipo al nivel de base de datos depende del conjunto de datos soportado por un DBMS particular. Puede haber algunas diferencias. Por ejemplo, un atributo de tipo "String" en C++ puede ocupar n- número de caracteres, mientras que un tipo correspondiente a Oracle llamado VARCHAR2 restringe el tamaño de columna a solo 255 caracteres. El problema se puede aliviar identificando estas diferencias antes e implementar limitantes a nivel de código.

7.2.5 Mapeo de asociaciones

Una asociación de manera general es una relación entre dos o más clases. Las relaciones binarias se muestran como líneas conectando a los dos clasificadores.

Un numero de multiplicidad especifica el rango de cardinalidades permitidas que un conjunto puede asumir. La multiplicidad se puede dar de la forma "LimiteInterior...LimiteSuperior", donde "LimiteInterior..LimiteSuperior" son literales enteras especificando un rango que va del Limite Inferior al Limite Superior. El símbolo "*" se usa para denotar un rango entero no negativo al infinito, dependiendo del rango de la asociación se emplea una técnica diferente de mapeo, los siguientes puntos muestran los diferentes rangos de asociaciones y su estrategia de mapeo.

7.2.5.1 Mapeo de asociaciones uno a uno

Para esta opción se pueden crear tablas separadas para cada clase involucrada en la asociación o las clases pueden combinarse para formar una sola tabla. La decisión comúnmente depende del dominio de la aplicación y el grado de acoplamiento que existe entre las dos de tablas.

Al nivel de base de datos, puede ser ineficiente almacenar la información en dos tablas separadas, y por lo tanto la información puede combinarse y ser almacenada en una tabla.

7.2.5.2 Mapeo de asociaciones uno a muchos

Para esta opción se crean tablas separadas para cada clase involucrada en la asociación. Este tipo de asociación se realiza almacenando la llave primaria de la clase que participa con una instancia en la relación como una llave foránea en la otra clase (la clase que participa con “n” instancias en la relación).

Para ilustrar este punto, se puede poner el ejemplo de la relación entre un enfermo que va al medico, este le da una consulta en la que le diagnostica tanto la enfermedad como medicinas para aliviar el padecimiento. En este ejemplo existe una relación uno a muchos entre la clase Paciente y Consulta, ya que un paciente puede tener muchas consultas, la figura 7.5 muestra la relación entre ambas clases.

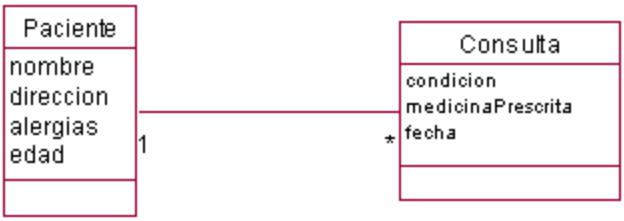

Figura 7.5. Ejemplo de asociación uno a muchos.

Con los conceptos antes explicados, la relación entre ambas clases se implementa almacenando la llave primaria de la tabla “Paciente” en “Consulta” como una llave foránea como se indica en la figura 7.6.

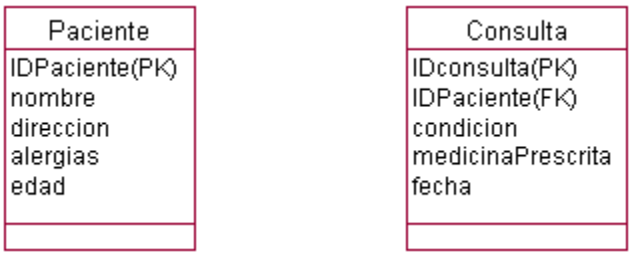

Figura 7.6. Mapeo de una asociación uno a muchos.

7.2.5.3 Mapeo de asociaciones muchos a muchos

Para este tipo de asociaciones se crean tablas separadas para cada clase involucrada en la asociación., además se debe crear otra tabla para almacenar la asociación misma.

La tabla agregada gestionará la asociación almacenando las llaves primarias de las tablas asociadas como llaves foráneas.

Siguiendo con el ejemplo del doctor, al paciente le pueden recetar muchas medicinas. Por otro lado hay formulas para tratar diversas enfermedades las cuales tienen varias medicinas y las mismas medicinas pueden tener varias formulas o combinaciones de medicinas.

La figura 7.7 muestra una relación de muchos a muchos entre las clases formulario y medicina.

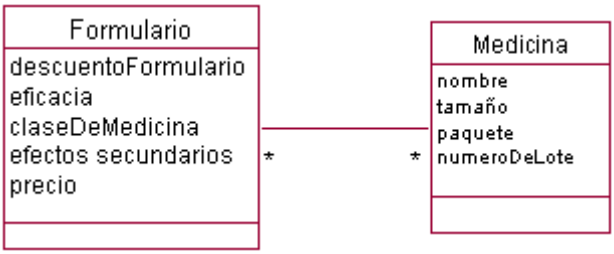

Figura 7.7. Ejemplo de una asociación muchos a muchos.

Con los conceptos antes explicados, la relación muchos a muchos se implementa creando una tabla llamada SoporteFormularioMedicina que soporte la relación muchos a muchos, esta tabla además de contener su propia llave primaria contiene las llaves primarias de ambas asociaciones como llaves foráneas. La figura 7.8 muestra el mapeo del ejemplo anterior.

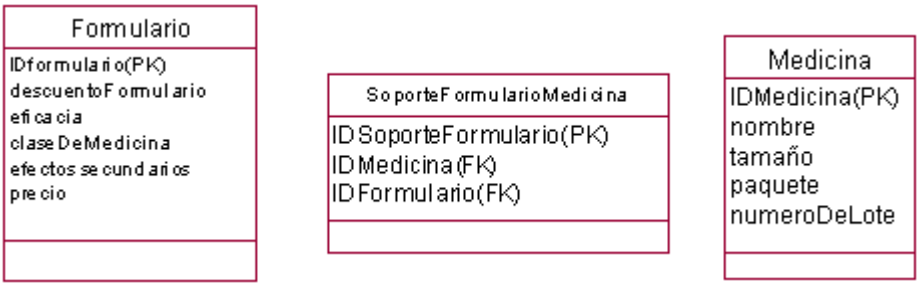

Figura 7.8. Mapeo de una asociación muchos a muchos.

7.2.6 Mapeo de generalizaciones

La generalización se refiere a una relación entre un elemento general (el padre) y un elemento más específico (el hijo), en donde el hijo hereda las propiedades del padre. La figura 7.9 sirve como ejemplo para ilustrar la generalización de las medicinas, ya que entre todas las medicinas se hay una clasificación que es: de patente y genéricas.

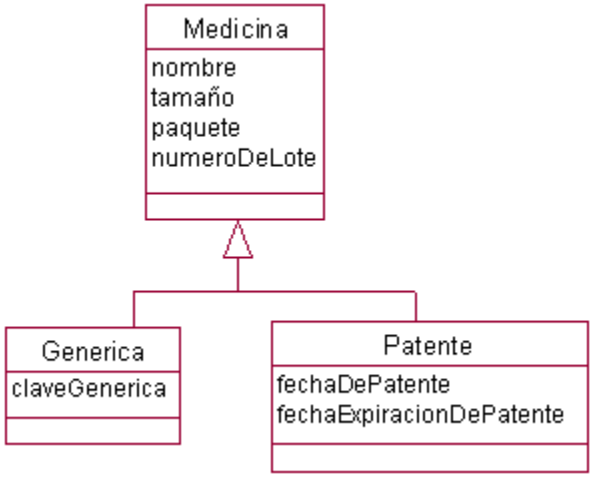

Figura 7.9. Ejemplo de generalización.

La implementación de la generalización es una de las tareas más difíciles cuando se convierte de un diagrama de clases a una vista lógica. Para realizar el mapeo de una generalización hay tres alternativas, las cuales se explican en los siguientes puntos, la decisión depende de las ventajas, desventajas de cada técnica y del propio sistema

7.2.2.4.1 Solución uno; destinar una tabla separada por cada clase

La primera solución para el mapeo de una generalización es crear una tabla por cada clase, incluyendo la clase abstracta.

La tabla que representa la clase general debe contener las columnas generales y las tablas que representan las clases específicas deben tener las columnas específicas. En este caso todas las tablas deben tener la misma llave primaria como columna. La figura 7.10 muestra la solución uno al mapeo de la generalización de medicinas.

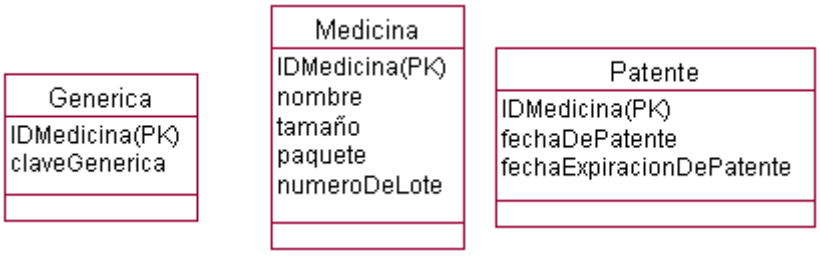

Figura 7.10. Solución uno al mapeo de una generalización.

7.2.2.4.2 Solución dos; crear una tabla por cada clase hija

La segunda solución para el mapeo de una generalización es crear una tabla por cada clase hija, después se almacenan los atributos de la clase padre en cada clase hija.

Una ventaja aparente de esta técnica es mejor rendimiento ya que no habrá un acceso extra para la súper clase. Una desventaja es que la solución no es lo suficientemente flexible para cambios futuros con respecto a la clase padre.

7.2.2.4.3 Solución tres; crear una tabla por toda la generalización

"Con este método se mapea toda la jerarquía de clases en una tabla En esta tabla se deben almacenar los atributos tanto de las clases especificas como de la clase general" [1]. La figura 7.11 muestra la tabla que reúne todos los campos de las clases Genérica, Medicina y Patente.

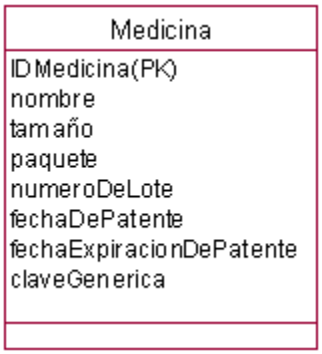

Figura 7.11. Solución tres al mapeo de una generalización.

7.2.7 Mapeo de agregaciones

El proceso de mapeo de una agregación inicia creando una tabla por cada clase de diseño y su mapeo de atributos, posteriormente se le agregan llaves primarias a cada clase. Después de esto se coloca la llave primaria de la tabla que participa con una instancia en la agregación como una columna en la tabla que participa con "n" instancias en la agregación, es decir, como llave foránea.

Para el caso de una composición, se puede implementar integridad referencial con el modo de eliminación en cascada para que cada vez que se elimine un registro del agregado (del todo) se eliminen de manera automática los registros asociados (partes del todo).

7.2.8 Selección de llaves primarias

En una base de datos relacional se necesita un identificador por cada tabla para que cada registro sea único y para almacenar las relaciones. Se debe asignar un identificador por cada objeto (o cada registro en términos de BD relacionales) a fin de identificarlo de manera única. Hay varias opciones para seleccionar llaves primarias de entre las columnas de la tabla que se explican en los siguientes párrafos.

La primera opción es seleccionar un atributo que tenga un significado funcional o de negocio y que además que sea único. Una ventaja de este método es que los usuarios estarán familiarizados con la llave. La desventaja es que si el dominio del negocio cambia entonces la llave que tiene un significado funcional o de negocios también cambiará.

La otra opción es crear una columna de tipo entera o una cadena de caracteres definida que almacene la llave primaria. Una de las ventajas de esta solución es que la llave se puede generar automáticamente por el sistema, de echo, algunos manejadores de bases de datos tienen como propiedades de los atributos enteros el “auto incremento”, lo que quiere decir que cada vez que se agrega un registro, la columna de la llave primaria se auto incrementa en uno automáticamente.

7.3 Diseño de la base de datos

El propósito de la actividad “Diseño de la base de datos” es:

- Asegurarse que los datos persistentes se almacenan de manera consistente y eficiente.
- Definir el comportamiento que debe implementar la base de datos.

Para esta actividad se asume que los lectores comprenden los conceptos básicos de bases de datos y en especial las BD relacionales.

7.3.1 Mapeo de clases persistentes al modelo de datos

En este paso se crea y define el modelo de datos para el soporte de almacenamiento y recuperación de datos persistentes.

Una vez que se estabiliza la estructura de las clases de diseño persistentes, se deben de mapear al modelo de datos. Sin embargo, actualmente hay herramientas automatizadas para realizar tal tarea, si se dispone de ellas, es en este paso donde se utilizan. Las herramientas que realizan el mapeo agilizan mucho la tarea debido a lo iterativo del proceso.

7.3.2 Optimización del modelo de datos

En este paso se optimiza el modelo de datos para mejorar lo más posible el rendimiento de la B.D.. En ocasiones el mapeo de clases a entidades da como resultado una tabla por clase. Para el caso anterior, hay veces en donde se necesitan recuperar datos de dos o más tablas y se hace lo que se llama un “join”, esta operación tiene un costo computacional muy alto. Para eliminar estos inconvenientes primero se deben seleccionar aquellas tablas que

son propensas a joins y aplicar una de-normalización. La de-normalización hace un split inverso de tablas, es decir, une dos tablas en una. Para aplicar una de-normalización primero se debe analizar el costo-beneficio, ya que, por un lado las operaciones de recuperación de información son menos costosas y por otro las operaciones de actualización tienen un mayor costo computacional. En cuanto a la de-normalización no es aconsejable realizarlo en más de dos tablas.

7.3.3 Optimización del acceso a los datos

En este punto se definen índices para algunas tablas en función de la determinación de las consultas más frecuentes a la base de datos. Las consultas "sirven para analizar y seleccionar información que está contenida en una base de datos para después crear un reporte con los resultados, la ventaja de las consultas es que para ejecutarlas no se necesitan conocimientos de lenguajes de programación" [12]

"La utilización de índices permite al sistema gestor de base de datos localizar y proporcionar de una forma rápida y sencilla una combinación de registros de entre los millones e incluso billones que puede contener una tabla" [27]

Una vez que se ha diseñado y estabilizado la estructura de las tablas, se determinan los tipos de consultas que se van a realizar a la base de datos.

Para ilustrar el término de indexamiendo o "agregar índices" considérese la siguiente consulta para la tabla 7.1:

SELECT * FROM Telefono WHERE APaterno = "Álvarez"

El sistema gestor de base de datos tendría que buscar registro por registro y encontraría el apellido "Álvarez" en el tercero, ahora supóngase una base de datos de 2 millones de registros, seguro que la consulta seria muy costosa y podría afectar el rendimiento.

# de registro	APaterno	AMaterno	Nombre	Telefono
1	Fernández	Hernández	Alberto	57-58-59-69
2	García	Pérez	Fabiola	54-98-85-19
3	Álvarez	López	Fernanda	12-58-84-95
4	Hernández	Hernández	Laura	12-59-89-65

Tabla 7.1 Registros de la tabla Telefono

Pero qué pasaría si los registros estuvieran ordenados por la columna "APaterno", seguro que la consulta sería mucho más rápida. Los índices hacen algo parecido a un ordenamiento por lo que hacen que muchas veces las consultas sean mas rápidas.

Para agregar un índice en MySQL a la columna “APaterno” se escribiría una sentencia como la siguiente:

ALTER TABLE Telefono ADD INDEX (APaterno)

Los siguientes puntos muestran estrategias de indexamiento para optimizar el acceso a los datos.

- La llave primaria de cada tabla debe estar siempre indexada. A menos que esta tenga menos de 100 registros.
- Además de los índices del punto anterior, también se deben agregar aquellas columnas que estén constantemente en las consultas.
- Los índices por lo general deben ir en columnas que son identificadores, es decir, no deben ser índices valores numéricos o información textual.
- Los índices por lo general deben ser enteros en vez de números de punto flotante, rara vez cadenas de caracteres.

Los siguientes puntos muestran precauciones cuando se usan índices.

- No se debe indexar para mejorar la velocidad de consultas que no sean frecuentes.
- El uso de índices debe ser moderado ya que estos ocupan espacio en el disco.
- En aplicaciones donde las operaciones de actualización o inserción son de más alta prioridad que las consultas debe haber pocos índices porque en este caso el costo excede su beneficio, ejemplos de esto son los sistemas orientados a la web en donde las operaciones de inserción son mas frecuentes que las consultas.

7.3.4 Definición de las reglas de integridad referencial y de datos

Este paso tiene como propósito definir las reglas que aseguren la integridad de la base de datos. “Una regla o “constraint” es una restricción o limitación que se aplica en la B.D. para asegurar la consistencia los datos cuando se agregan, actualizan o eliminan registros” [11]

Las reglas de integridad de datos aseguran que la información en la BD está en un determinado rango valido para así preservar su integridad. Esto no significa que la aplicación no tenga que definir sus propias reglas de validación, sino que las reglas de integridad de la base de datos se definen como una segunda capa de seguridad. Para esta tarea primero se definen los rangos en los que deben caer las columnas de la base de datos,

después se implementan las constraints dependiendo del sistema manejador de base de datos que se tenga.

Además de estas reglas de integridad de datos, también se pueden implementar reglas de integridad referencial, estas aseguran que por cada valor de una llave foránea, debe haber un valor de llave primaria en la tabla referenciada.

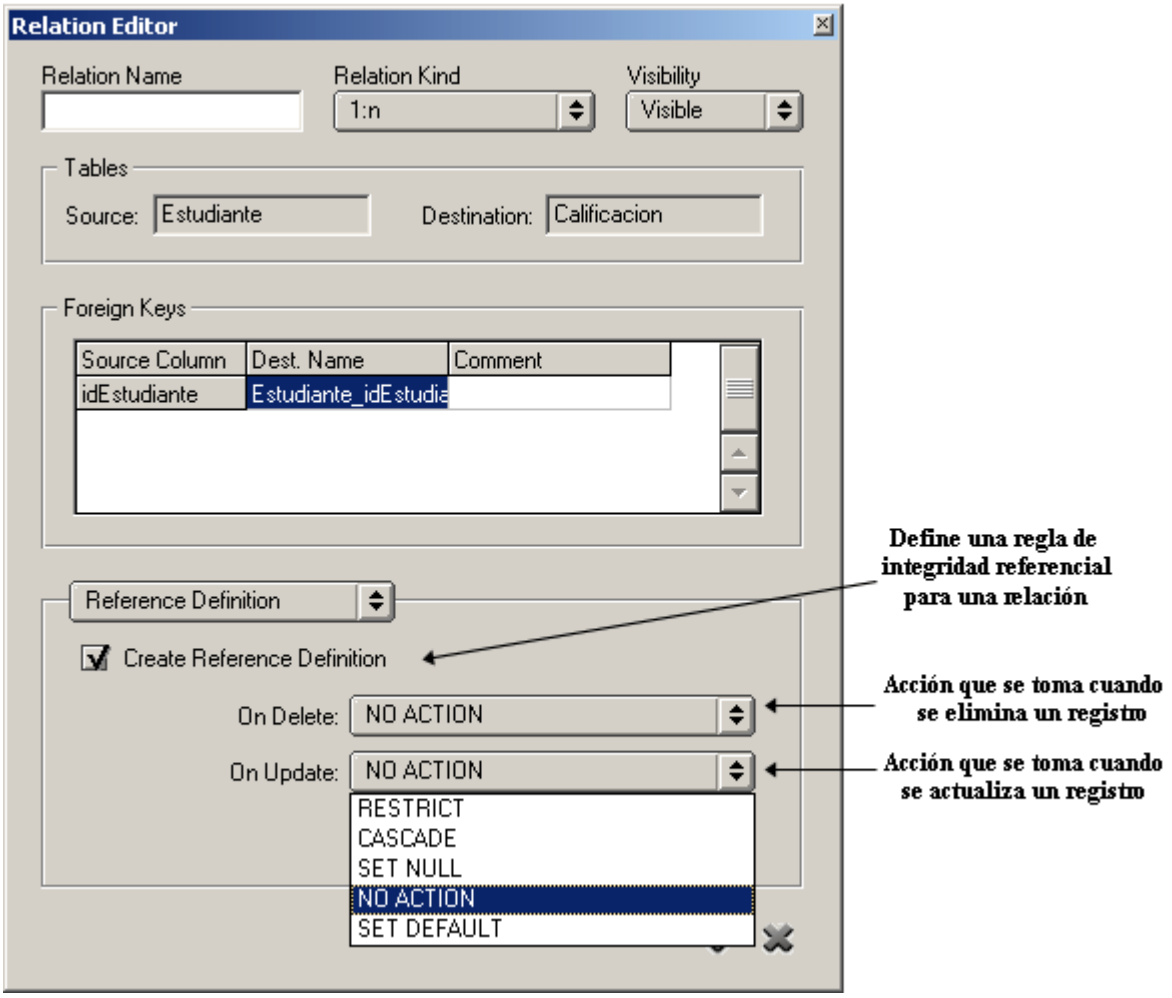

Figura 7.12. Implementación de integridad referencial en DBDesigner.

La figura 7.12 muestra el ejemplo de una ventana para definir integridad referencial entre dos tablas en una herramienta de diseño de bases de datos para MySQL llamada DBDesigner, en ella se indican diversas características de la relación, sin embargo, en este caso las que importan son las que se encuentran en la parte inferior, la primera leyenda apunta en la parte donde se indica que se quiere definir una regla de integridad referencial. La segunda flecha apunta a la acción que tomará el gestor cuando se borra un registro de las tablas relacionadas (On Delete), la tercera flecha apunta hacia la acción que se ejecutará

cuando se actualice un registro de la tabla que participa con la llave primaria en la relación (On Update).

7.3.5 Distribución del comportamiento de la clase a la base de datos

En este paso se determina el comportamiento de aquellas clases que se debe distribuir e implementar en la base de datos y no en la aplicación.

Para la implementación del comportamiento en una base de datos se utilizan procedimientos almacenados. Un procedimiento almacenado no es más que código ejecutable que corre dentro del espacio del proceso del sistema gestor de base de datos, estos procedimientos tienen la habilidad de realizar operaciones en la base de datos. Los beneficios de los procedimientos almacenados es un mejor rendimiento del sistema, siempre y cuando se utilicen de manera inteligente.

Existen dos tipos de procedimientos almacenados: Los “procedimientos actuales” y los “triggers”. Los procedimientos actuales se ejecutan explícitamente por la aplicación, generalmente tienen parámetros y valores de retorno. Los triggers se invocan implícitamente cuando algún evento específico ocurre en la base de datos (agregar, eliminar, borrar o actualizar un registro), los triggers no tienen parámetros ni valores de retorno.

En este paso se deben analizar las clases de diseño persistentes para ver si hay operaciones que se deben implementar usando procedimientos almacenados o triggers; los candidatos para este caso son los siguientes:

- Operaciones que realizan borrado, actualización o agregación de datos persistentes (datos de la BD).
- Operaciones que se pueden traducir en consultas a datos persistentes.
- Operaciones que necesitan acceso a la BD para validar datos.

En todos los casos anteriores las operaciones se pueden realizar sin la mediación de la aplicación, es decir, el comportamiento lo puede realizar el sistema gestos de base de datos por si mismo, es por esto que los triggers hacen que mejore el rendimiento del sistema.

7.4 Artifacts principales

En esta sección se explica el modelo de datos, este artefacto es el más importante en cuanto al diseño de la B.D.

7.4.1 Modelo de datos

El modelo de datos es un artefacto que describe la representación lógica y física de los datos persistentes que administra el sistema. Este modelo se usa para definir el mapeo de las clases de diseño persistentes a un modelo que pueda implementarse como una base de datos.

7.4.1.1 Partes importantes del artefacto

Los siguientes puntos muestran las partes mínimas que debe tener el modelo de datos.

- **Introducción:** En esta parte del documento se elabora un resumen de todo el documento, incluye el propósito, limites, definiciones, acrónimos, abreviaturas y referencias.
- **Tablas:** En esta sección se documenta cada tabla y cualquier aspecto importante que tenga que ver con el diseño.
- **Relaciones:** En esta sección se documentan las relaciones entre las tablas y su implementación por medio de las llaves primarias y foráneas.
- **Columnas:** Las columnas representan el mapeo de las clases de diseño a las propiedades de cada tabla. Para cada columna se debe documentar el tipo de dato que contiene, si es o no llave primaria o foránea, si se tiene un valor por defecto para la columna o si los valores deben entrar en un rango definido.
- **Índices:** En esta sección se documenta el proceso y la racionalidad que se siguió para la definición de índices en las tablas a fin de acelerar el acceso a los datos.
- **Triggers:** En esta sección del modelo de datos se documentan todos los triggers que se definan en la base de datos, por cada trigger se documenta el mapeo del comportamiento de la clase de diseño a la implementación del trigger en la base de datos, también se documentan las razones por las que se utilizó este método en vez de implementarse en el comportamiento de la clase por medio de la aplicación.

Conclusiones

Los procesos para el desarrollo de software han evolucionado, y seguirán evolucionando, eso no es nuevo, cada día el software se desarrollará más rápido y con mayor calidad debido a la experiencia en el área y también al refinamiento del proceso. En esta evolución es claro que el RUP siempre estará como un referente importante en el camino hacia el "proceso perfecto".

Uno de los aspectos críticos en el éxito del RUP esta precisamente en su nombre: la **unificación** de conceptos de Orientación a Objetos, procesos de ingeniería de software, herramientas, experiencia y representación de conceptos (UML).

UML es de vital importancia en el RUP y más en la disciplina de análisis y diseño como vehículo de representación y comunicación a lo largo de la concatenación de actividades, trabajadores y artefactos. Y es que es en esta disciplina se crean y actualizan el mayor número de artefactos que tienen que ver con UML, de echo se podría decir que la disciplina de análisis y diseño marca el inicio en forma de la creación de los modelos de objetos ya que el modelo de casos de uso, que se crea en los flujos de trabajo anteriores, no contiene objetos, el propósito principal del modelo de caso de uso es la representación de procesos en general, ya sea de negocios o en los requerimientos del sistema.

Ya entrando en el proceso, se concluye que la serie de pasos que se ejecutan en el análisis y diseño es muy eficiente y sigue la línea general del RUP de mitigar los riesgos al inicio del proyecto, ya que en el primer detalle de flujo de trabajo se realiza un estudio de factibilidad con base a un resumen de la arquitectura del sistema que solucione los requerimientos mas críticos e importantes. Después, cuando se sabe que el proyecto es viable se realiza un primer esfuerzo para lograr una arquitectura estable en el detalle de flujo de trabajo "Definición de una arquitectura candidata".

En el siguiente paso, el diseñador retoma los esfuerzos que se hicieron para la captura de requisitos y convierte las necesidades del usuario en un modelo de análisis el cual es independiente de la tecnología de implementación, esta transformación se lleva a cabo en el detalle de flujo de trabajo "**Análisis del comportamiento de los Casos de Uso**", que permite que los requerimientos puedan ser mejor entendidos por el equipo que conforma el

proyecto, ya que a los desarrolladores les es mas fácil tratar con objetos y estructuras que con actores y casos de uso.

Antes de iniciar el diseño, el detalle de flujo de trabajo “Refinación de la arquitectura” resuelve todos los puntos que pudieran quedar pendientes en cuanto a la arquitectura del sistema tales como los mecanismos, los requerimientos de concurrencia o la distribución, los cuales se plasman en el documento de arquitectura de software.

En el quinto detalle de flujo de trabajo llamado “Diseño de componentes” se hace una unión de lo conceptual y lo específico, es decir, el modelo de análisis y la arquitectura del sistema se mezclan para entrar al espacio de la solución y satisfacer los requerimientos del sistema por medio de clases, subsistemas, paquetes, asociaciones y otros elementos.

Por ultimo, se desarrolla la base de datos haciendo un mapeo de clases persistentes a tablas y llaves que resuelvan el problema del almacenamiento y recuperación de objetos que requieren tener una larga vida dentro del sistema, después se refina la estructura de la B.D de modo que la inserción, modificación, eliminación y consulta de los registros se realice de manera eficiente, de esta forma el diseño esta listo para implementarse en la siguiente disciplina.

Por supuesto que es posible que el lector no sepa o tenga una comprensión difusa de los conceptos que se tratan en la monografía, por eso, al inicio de cada capítulo se explican los conceptos preliminares más importantes, de esta forma se puede entender mejor el desarrollo de cada actividad.

Otro punto importante en cuanto a la monografía es que al final de cada capítulo se muestran los aspectos importantes y la estructura que deben tener los artefactos más significativos que se crean o actualizan en tal capítulo.

De esta forma se han cumplido los objetivos que se plantearon para la monografía, ya que se han explicado los conceptos subyacentes del RUP y particularmente del Análisis y Diseño para después documentar y explicar los artefactos, actividades y roles mas importantes de esta disciplina

Anexo A

Enunciado de problema y caso de uso del sistema de nomina

Este anexo tiene como propósito ejemplificar un diagrama de caso de uso para el sistema de nomina de una empresa, este diagrama se basa en el enunciado del siguiente punto que describe los requerimientos que debe de tener el sistema.

Todo lo anterior es necesario debido a que antes de la disciplina de análisis y diseño, en la etapa de requerimientos se debe realizar un modelo de caso de uso. Este modelo es el que contiene los requerimientos funcionales del sistema y por tanto una entrada principal la disciplina de análisis y diseño. Este anexo es útil para realizar algunos ejemplos, principalmente del capítulo tres y posteriores.

A.1 Enunciado del problema*

La gerencia general ha designado al jefe del departamento de informática para actualizar el Sistema de Nomina de la compañía, el nuevo sistema debe correr tanto en plataformas basadas en ventanas como en la plataforma Web para introducir en la tarjeta de registro personal del empleado ordenes de compra, cambio de información del empleado (tal como método de pago) y crear varios reportes Por razones de seguridad y auditoria, los empleados solo pueden acceder y editar sus propias tarjetas de registro.

El sistema debe de retener información de todos los empleados de la compañía (aprox. 5,000 empleados) El sistema debe pagar a cada empleado el monto correcto a tiempo por el método que el mismo haya especificado. Por razones de costo, el sistema debe heredar la B.D. actual llamada “BD de administración de proyectos” que tiene información histórica de los proyectos de la compañía. Esta B.D. esta implementada en DB2 y corre en un mainframe IBM. El sistema solo accesa pero no actualiza la información de la B.D. heredada.

Algunos empleados trabajan por hora, cada empleado tiene su cantidad de pago por hora, por otro lado las horas extras se pagan a un 150% de la cantidad por hora normal. A los empleados por hora se les paga cada viernes.

* El enunciado fue sacado del libro de ejercicios del curso de análisis orientado a objetos con UML llamado “Payroll Requirements Document” [21]

Otro tipo de empleados trabajan por un salario fijo, estos empleados se les paga cada fin de mes. A estos empleados se les llama asalariados.

Los empleados asalariados también reciben comisión con base en sus ventas. Ellos mandan la información referente al monto de las ventas en una determinada fecha. El porcentaje de comisión depende de cada empleado y los porcentajes validos son: 10%, 15%, 25% o 35%.

Uno de los requisitos del sistema más críticos es el reporte de empleados. Los empleados deben poder consultar el sistema para saber las horas trabajadas, total de pagos recibidos a una fecha, fecha de vacaciones, etc.

Los empleados pueden escogen su método de pago. Las formas disponibles son: Mandar sus cheques de pago por correo postal a su dirección, también se puede hacer el pago por medio de una transacción a la cuenta de algún banco o la tercera es recibir el cheque en la compañía el día de pago.

El administrador de la nomina es el encargado de mantener la información de los empleados, es decir, puede dar de alta , eliminar y modificar la información de cada empleado tal como nombre, dirección, clasificación salarial(Por hora, asalariado y comisionado), así como correr los reportes administrativos.

El administrador de nomina corre un proceso de cálculo de salarios cada viernes y cada fin de mes automáticamente. El sistema debe pagar a los empleados el día correcto y conforme al último día en que se le realizó pago.

A.2 Diagrama de caso de uso

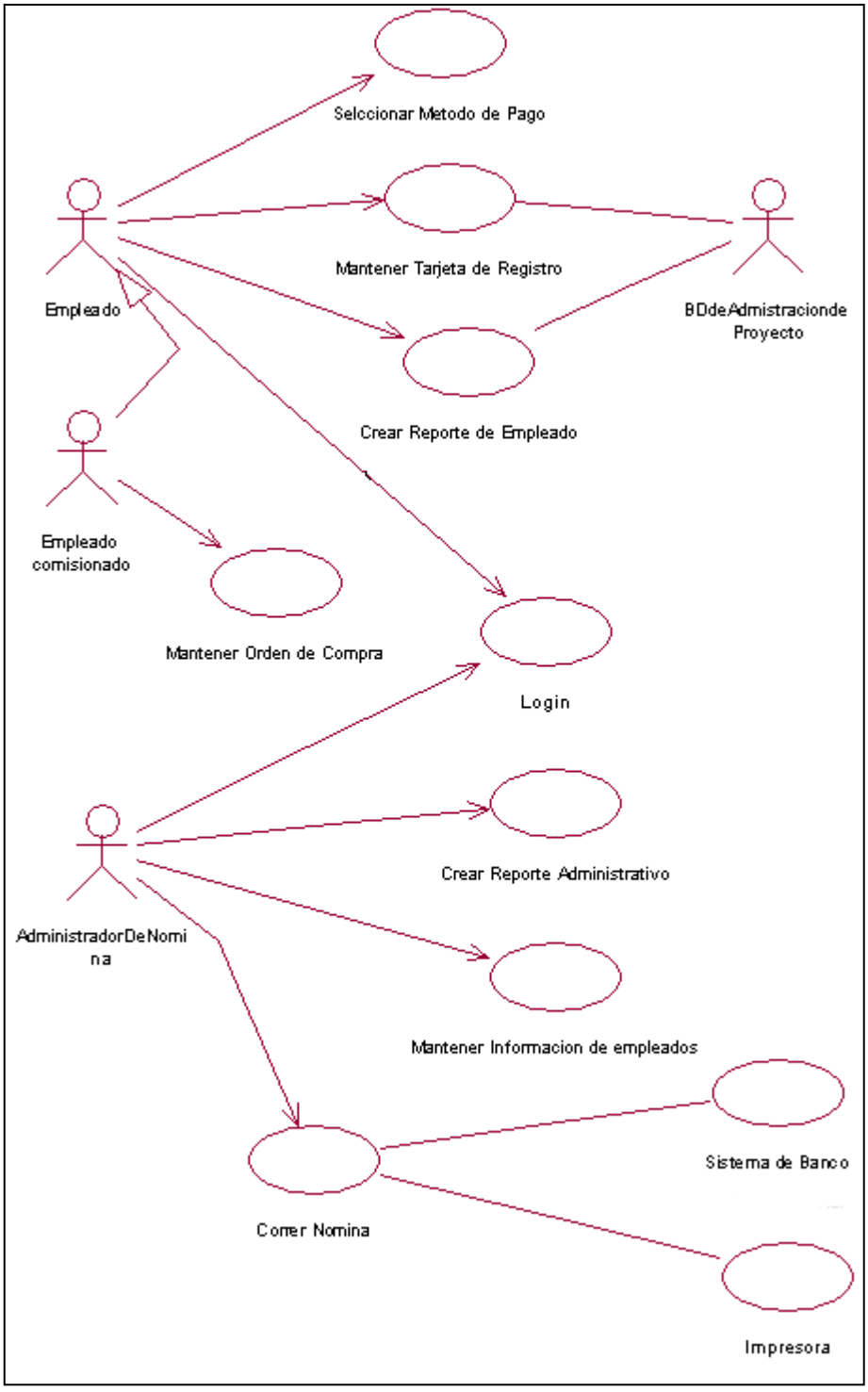

Figura A1.1. Diagrama de caso de uso del sistema de nomina.

Anexo B

Trabajadores del análisis y diseño

Los siguientes puntos muestran los perfiles de los trabajadores o roles más significativos de la disciplina de análisis y diseño del RUP.

Arquitecto de software

El arquitecto de software lidera y coordina las actividades y artefactos de la mayoría del proyecto, además establece toda la estructura de cada vista arquitectónica lo cual incluye: la descomposición de las vistas, la agrupación de los elementos y las interfaces entre los grupos de mayor peso.

La figura A2.1 muestra las actividades y artefactos en los que el arquitecto de software está relacionado.

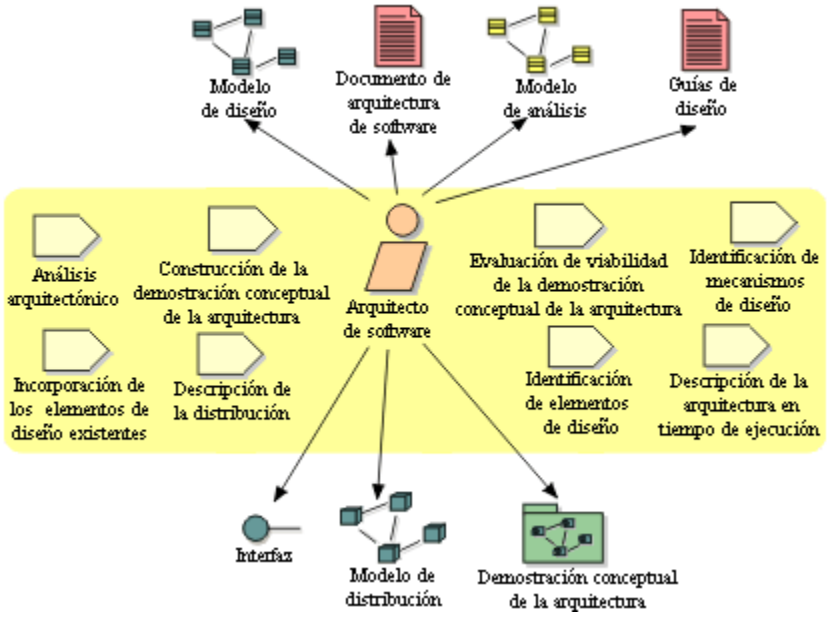

Figura A2.1. Actividades y artefactos del arquitecto de software.

Los siguientes puntos muestran diversos aspectos en el perfil de un arquitecto de software.

- **Experiencia:** Debe ser amplia en cuanto al dominio del problema (requerimientos) y de la ingeniería de software.

- **Liderazgo:** Implica dirigir el esfuerzo técnico de los diferentes equipos de desarrollo hacia las metas propuestas. En cuanto a las decisiones técnicas, el arquitecto de software es el que tiene más autoridad y más responsabilidad.
- **Comunicación:** El arquitecto de software debe tener una amplia y efectiva comunicación con los equipos de desarrollo para persuadir, motivar, además de servir mentor de varios elementos nuevos.

Diseñador

El rol del diseñador define las responsabilidades operaciones, atributos y relaciones de una o varias clases, además de determinar cómo se ajustarán estas en el ambiente de implementación. También puede tener la responsabilidad de uno o más paquetes o subsistemas de diseño.

El perfil del diseñador incluye un sólido conocimiento de:

- Técnicas de modelado de casos de uso
- Técnicas de diseño de sistemas, incluyendo aquellas que tengan que ver con el análisis y diseño orientado a objetos y UML
- Tecnologías y plataformas de implementación del sistema que se esté desarrollando.

Además del conocimiento anterior, el diseñador debe conocer la arquitectura del sistema y conocer aspectos sólidos de pruebas.

La figura A2.2 muestra las actividades y artefactos en los que el diseñador está relacionado.

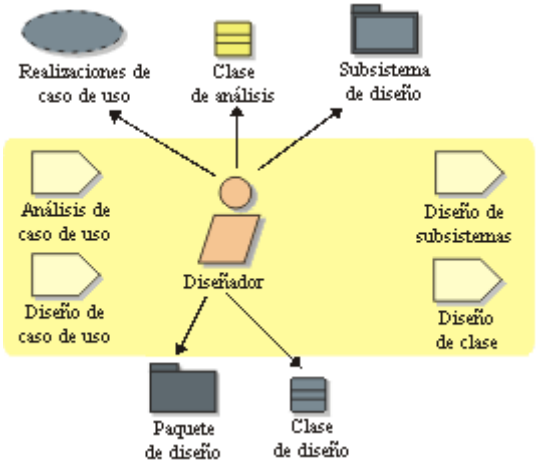

Figura A2.2. Actividades y artefactos del diseñador.

Diseñador de base de datos

El diseñador de base de datos define las tablas, índices, vistas, triggers, parámetros de almacenamiento y cualquier elemento específico de la base de datos que permita almacenar, recuperar y eliminar los objetos persistentes. Esta información se documenta y mantiene en el modelo de datos.

El diseñador de base de datos debe tener experiencia y conocimiento sólidos en los siguientes aspectos:

- Técnicas de análisis y diseño de bases de bases de datos y de orientación a objetos.
- Arquitectura del sistema
- Administración de base de datos.
- Conocimiento del ambiente y lenguaje de implementación.

La figura A2.3 muestra las actividades y artefactos en los que el diseñador de base de datos está relacionado.

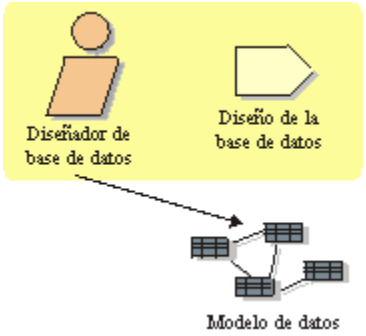

Figura A2.3. Actividades y artefactos del diseñador de base de datos.

Revisor de diseño

El revisor de diseño planea y conduce las revisiones formales del artefacto Modelo de diseño. La figura A2.4 muestra la actividad y el artefacto en que el revisor de diseño esta relacionado.

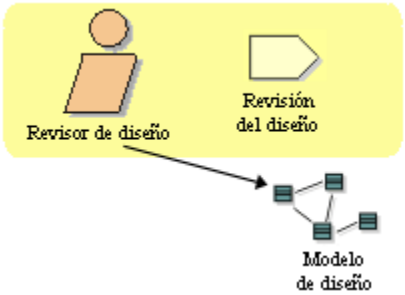

Figura A2.4. Actividades y artefactos del Revisor de diseño.

El revisor de diseño tiene un perfil muy parecido que el arquitecto de software, las diferencias son que este puede tener menos capacidades técnicas, al mismo nivel que el diseñador, además de que los aspectos de liderazgo, madurez y orientación a resultados son menos críticos. Es muy importante la visión a largo plazo que tenga el revisor porque así puede detectar riesgos potenciales en etapas tempranas y analizar sus costos, de esta manera se pueden mitigar más fácilmente.

Glosario

Baseline: Entregable aprobado y revisado de artefactos que constituye una base de acuerdo para evoluciones o desarrollo posteriores y que pueden ser cambiados solo a través de un procedimiento formal, como administración de cambios y control de la configuración.

Build: Artefacto que representa una versión operacional del sistema, parte del sistema que demuestra un conjunto de las capacidades que tendrá el producto final. El build va incrementando sus capacidades conforme van pasando las fases.

Clase de entidad: Es una clase que se usa para modelar información que ha sido almacenada por el sistema y el comportamiento asociado. Una clase genérica, que se usa en muchos casos de uso, frecuentemente contiene características de persistencia. Una clase de entidad, define un conjunto de objetos de entidad que participan en varios casos de uso.

Detalle de Flujo de Trabajo: Es una agrupación de actividades que son desempeñadas en estrecha colaboración para lograr algún resultado. Las actividades se desempeñan típicamente en paralelo o iterativamente, la salida de una actividad puede ser la entrada de otra. El detalle de flujo de trabajo es usado para agrupar actividades y así proveer una abstracción de alto nivel y mejorar la comprensión de los flujos de trabajo.

Disciplina: Es una colección de actividades relacionadas que se relacionan alrededor de un "área de interés" mayor dentro del proyecto total. El agrupamiento de actividades en disciplinas se hace como una ayuda para entender el proyecto desde una perspectiva de "cascada" tradicional.

Como otros flujos de trabajo, un flujo de trabajo de disciplina es una secuencia semi-ordenada de actividades que son realizadas para lograr un resultado particular. Esta naturaleza semi-ordenada de los flujos de trabajo de disciplina enfatiza que no se puede presentar los aspectos reales de calendarización de un proyecto real, ya que no puede representar la opcionalidad de las actividades o la naturaleza iterativa de los proyectos

reales. Sin embargo, las disciplinas aun tienen el valor de ayudar a entender el proceso, partiéndolo en diferentes “áreas de interés”.

Flujo de Trabajo: Un flujo de trabajo es una secuencia de actividades que producen un resultado de valor observable. En términos de UML, un flujo de trabajo puede ser expresado como un diagrama de secuencia, un diagrama de colaboración o un diagrama de actividades. En el RUP se presenta en forma de un diagrama de actividades. Para cada disciplina, se presenta un diagrama de actividades, el cual se presenta en términos de detalle de flujo de trabajo.

Framework: Una micro-arquitectura que provee una plantilla extensible para aplicaciones en un dominio especifico

Rol: Es la definición de comportamiento y responsabilidades de un individuo o un conjunto de individuos que trabajan en un equipo, en el contexto de una organización de desarrollo de software. En UML un rol es un determinado comportamiento específico para una entidad participando en un contexto particular.

Stakeholder: Un individuo que es afectado por el sistema

Patrón: Es una plantilla de solución a un problema que se ha probado como útil en un contexto dado

Patrones de análisis: Grupos de conceptos que representan una construcción común que puede ser relevante a uno o varios dominios. Es un resumen o plantilla conceptual, para instanciarse en un modelo de análisis. Este patrón de análisis necesitará más refinamiento en el diseño.

Referencias bibliográficas

- [1] Ambler, Scott, Mapping objects to relational databases, Rational software, 2000.
- [2] Bell, Donald, “UML basics Part III: The class diagram”, Rational Software, 2003.
- [3] Crain, Anthony, The simple artifacts of Analysis and Design, IBM, 2004.
- [4] U.S. Departament of Defense, “Software Development and Documentation, U.S. Departament of Defense MIL-STD-498”, 1994.
- [5] D'Souza, Desmond; Cameron, Alan, “Objects, Components, and Frameworks With UML: The Catalysis Approach”, Ed Addison Wesley, 1998.
- [6] Eeles, Peter; Houston, Kelli; Kozaczynski, Wojtek, Building J2EE applications with the Rational Unified Process, Ed Addison Wesley, 2003.
- [7] Evans, Gary, “Getting from use cases to code : Part I: Use Case Analysis”, Rational software, 2004.
- [8] Evans, Gary, “Getting from use cases to code : Part II: Use Case Design”, Rational software, 2004.
- [9] Ferm, Fredrik, “The what, why, and how of a subsystem”, Rational software, 2003.
- [10] Heinckiens, Peter, “Building Scalable Database Applications: Object-Oriented Design, Architectures, and Implementations”, Ed Addison Wesley, 2004.
- [11] IBM, DB2 Universal Database for iSeries Database Programming Version 5, 1998.

- [12] IBM, Query for iSeries Workshop, 2003.
- [13] Jacobson, Ivar; Booch, Grady; Rumbaugh, James, El Proceso Unificado de Desarrollo de Software, Ed Addison Wesley, 2000.
- [14] Jacobson, Sten, "The Rational Objectory Process – A UML-based Software Engineering Process", Rational software, 1995.
- [15] Kruchten, Philippe, "From Waterfall to Iterative Lifecycle - A tough transition for project managers", Rational software, 2000.
- [16] Miller, Granville, "Java Modeling: A UML workbook, Part 1", Rational software, 2001.
- [17] OMG, "Unified Modeling Language Specification Version 1.3", 1999.
- [18] Probasco, Leslee, The Ten Essentials of RUP The Essence of an Effective Development Process, Rational software, 2000.
- [19] Quatrani, Terry, Introduction to the Unified Modeling Language, Rational software, 2003.
- [20] Rational software, "Fundamentals Versión 5.5", 2002.
- [21] Rational software, Object Oriented Analysis with UML Payroll Requirements Document, 2002.
- [22] Rational software, Rational Unified Process , 2002.
- [23] Rational software, "Object-Oriented Analysis with UML Version 2002 Volume 1", 2002.
- [24] Rational software, "Object-Oriented Analysis with UML Version 2002 Volume 2", 2002.

- [25] Si Alhir, Sinan, UML in a nutshell, O´Reilly, 1998

- [26] Siau, Keng; Halpin, Terry, “Unified Modeling Language: Systems Analysis, Design and Development Issues”, Ed. IGP Idea Group Publishing, 2001.

- [27] Zawodny, Jeremy; Balling, Derek, “MySQL Avanzado Optimización, copias de seguridad, replicación y equilibrado de carga”, Ed. O´Reilly, 2004.