UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"ACATLAN"

METODOLOGIA PARA EL ANALISIS Y DISEÑO DE SISTEMAS ORIENTADO A OBJETOS CON UML

T E S I S

QUE PARA OBTENER EL TITULO DE:
LICENCIADO EN MATEMATICAS APLICADAS Y COMPUTACION

P R E S E N T A :

JUAN MANUEL ROLDAN ZAMUDIO

ASESOR: LIC. OSCAR GABRIEL CABALLERO MARTINEZ.

ENERO DEL 2004

Dedicatoria

A MI PADRE JUAN MANUEL

Porque gracias a tu cariño, inteligencia, paciencia y generosidad me diste el ánimo para lograr mis metas. Gracias por haberme apoyado en todo momento. Tu amor es lo que ha hecho posible alcanzar esta meta en mi vida. Te quiero mucho.

A MI MADRE CAROLINA

Por siempre estarás en mi vida, en mi mente, en mi corazón.

A MI ABUELITA ELVIRA

Por todos tus cuidados a lo largo de mi vida, has marcado mi existencia con tu cariño, comprensión y amor. Perdóname por todas las desavenencias que te hice pasar.

A MI TÍA SOCORRO, MI PRIMA LESBIA, MI SOBRINO CARLOS ENRIQUE, MI PRIMO ROMÁN

Porque siempre tuvieron una palabra de aliento o acción hacia mí, que me motivo a seguir adelante.

A MI ESPOSA ASIYADETH

Por permitirme formar parte de tu vida y brindarme tu amor y apoyo en todas mis aspiraciones y proyectos.

A MI BEBE'S

Este trabajo es para ti y por ti bebe's, para el futuro.

A MI PRIMO NACHO, MI PRIMA MECHE, MI SOBRINA NADIA, MI SOBRINO ALAN

Por su apoyo incondicional y desinteresado, por estar conmigo en las buenas y en las malas, espero que se sientan orgullosos de mí.

A MIS AMIGOS EDUARDO Y MAURICIO

De los que he aprendido mucho y seguiré aprendiendo, y que se convirtieron en parte de mi vida académica y personal.

A TODAS AQUELLAS PERSONAS

Por depositar en mí su confianza y alentarme para alcanzar mi sueño: la obtención de un título profesional.

A MI ALMA MATER, LA UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Por la formación profesional recibida en la institución educativa más importante de este país.

"Alles ist relativ"
A. Einstein

Contenido

Introducción

Objetivo

El objetivo primordial de esta Tesis es proporcionar al lector un método razonado de proceder, con la utilización del UML (*Unified Modeling Language*, Lenguaje de Modelado Unificado), para que durante el análisis de desarrollo de sistemas computacionales, se identifiquen el alcance y los requerimientos detallados de los mismos, y durante el diseño se modelen las clases principales, así como su comportamiento, sus asociaciones y su arquitectura.

Justificación

No existen dos proyectos de desarrollo de software que sean iguales. Cada proyecto tiene prioridades, requisitos, y tecnologías muy diferentes. En cada

proyecto se requiere de minimizar el riesgo, asegurar resultados predecibles, y entregar software de alta calidad a tiempo.

Hipótesis

El UML es un lenguaje de modelado visual para sistemas de software. La Metodología para el Análisis y Diseño de Sistemas Orientado a Objetos con UML, es una plataforma[1] de desarrollo de software flexible que usa la notación visual del UML, proporciona las pautas de como utilizarlo efectivamente y emplea la técnica de desarrollo iterativo; variando así, el énfasis de cada flujo de trabajo durante el ciclo de vida. El desarrollo iterativo ayuda a localizar los riesgos temprano y continuamente, a través de progreso demostrable y entregas ejecutables frecuentes.

La metodología proporciona una asistencia invaluable en la captura de requisitos de alto nivel, procesos del negocio, y un modelado de objetos detallado de la aplicación de software. De igual forma, comparte esta información en un formato gráfico con los usuarios finales y el equipo de desarrollo, facilitando la comunicación y comprensión común.

Para corroborar la funcionalidad de la metodología, se implementará el sistema denominado "CLC Web", que instituirá la Tesorería de la Federación, Dependencia de la Secretaría de Hacienda y Crédito Público.

[1] Se le denomina plataforma porque proporciona especificaciones técnicas que describen la metodología pero, además, provee las herramientas para analizar y diseñar productos de software (aplicaciones) basados en dichas especificaciones.

Capitulado

Esta Tesis esta dividida en cuatro capítulos, el primero familiariza al lector con el UML, explicando su significado, breve historia de su concepción y desarrollo; considera el modelado visual y sus objetivos, y aclara las diferencias entre el desarrollo iterativo e incremental.

El segundo capítulo, explica detalladamente cada uno de los conceptos utilizados en la notación UML para su entendimiento y compresión adecuada.

El capítulo número tres, define el método que se seguirá en el proceso de desarrollo, en la etapa de análisis, estudia la habilidad de formular requisitos, identificar actores, modelar casos de uso y la identificación de las clases del sistema.

Y finalmente, el último capítulo, aplica la metodología para modelar las clases principales del sistema, redactar los escenarios, definir el comportamiento de la aplicación, agrupar las clases, y simbolizar la arquitectura lógica y física del software.

CAPÍTULO 1

Marco Teórico

1.1 Antecedentes e Historia

La unificación de los métodos de modelado de objetos ha sido posible porque la experiencia ha permitido hacer la ordenación entre los diferentes conceptos propuestos por los métodos existentes.

El UML (*Unified Modeling Language*, Lenguaje de Modelado Unificado) se define como un "lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software"[2]. Es un lenguaje notacional destinado a los sistemas de modelado que utilizan conceptos orientados a objetos.

El UML es un estándar de la industria para construir modelos orientados a objetos. Nació en 1994 por iniciativa de Grady Booch[3] y James Rumbaugh[4] para

---

[2] "*The UML specification documents*", Booch, G., I. y Rumbaugh, J. 1997. Rational Software Corp.

[3] "*http://www.rational.com/university/rubios.jsp?SMSESSION=NO#booch*", Grady Booch, ha estado con la Rational Software Corporation como Jefe de Científicos desde su fundación en 1980. Ha servido como arquitecto para numerosos sistemas de software complejos alrededor del mundo. Se graduó de BS (Bachelor of Science, Licenciado en Ciencias) en la Academia de la Fuerza Aérea de los Estados Unidos en 1977 y su MSEE (Master of Science degree in Electrical Engineering, Grado de Maestro de Ciencias en Ingeniería Eléctrica) de la Universidad de California en Santa Bárbara en 1979.

combinar sus dos famosos métodos; el de Booch y el OMT (*Object Modeling Technique*, Técnica de Modelado de Objetos). Más tarde se les unió Ivar Jacobson[5], creador del método OOSE (*Object-Oriented Software Engineering*, Ingeniería de Software Orientada a Objetos) y de los casos de uso (Use Cases), una técnica muy eficaz para la determinación de las necesidades. Surgió en respuesta a una petición del OMG (*Object Management Group*, asociación para fijar los estándares de la industria) para definir un lenguaje y una notación estándar del lenguaje de construcción de modelos.

La primera versión de la descripción del método unificado fue presentada en octubre de 1995 (ver figura 1.1) en un documento titulado "Unified Method V0.8" (*Método Unificado*). Comentarios provenientes de la comunidad de usuarios fueron tomados en cuenta en la versión 0.9 aparecida en junio de 1996, pero es especialmente la versión 0.91, disponible desde octubre de 1996, la que permite notar la evolución del método unificado.

La principal modificación consiste en la reorientación del alcance del esfuerzo de unificación, primero hacia la definición de un lenguaje universal para el modelado de objetos, y eventualmente, hacia la estandarización del proceso de desarrollo de objetos. El Método Unificado (*Unified Method*) se transforma en UML.

[4] *"http://www.rational.com/university/rubios.jsp?SMSESSION=NO#rumbaugh"*, Dr. James Rumbaugh, ha sido un líder en el desarrollo del UML como representante de Rational en el OMG (Object Management Group) y ha contribuido en muchos de los conceptos del UML. Jim tiene un BS (Bachelor of Science, Licenciado en Ciencias) en Física de MIT (Massachusetts Institute of Technology, Instituto de Tecnología de Massachusetts), un MS (Master of Science) en Astronomía de Caltech (California Institute of Technology, Instituto de Tecnología de California) y un Ph. D. (Programa de Doctorado) en Ciencias de la Computación de MIT.

[5] *"http://www.rational.com/university/rubios.jsp?SMSESSION=NO#jacobson"*, Dr. Ivar Jacobson, es Vicepresidente de Estrategia de Procesos en Rational, fundo la compañía sueca Objectory AB, la cual se fusionó con Rational en 1995. Jacobson es el principal autor de cinco libros en ingeniería de software orientada a objetos, reutilización de software y el proceso de desarrollo de software. Recibió su Ph. D. (Programa de Doctorado) en el Instituto Real de Estocolmo.

En 1996, se crea un consorcio de colaboradores para trabajar en la definición de la versión 1.0 del UML que agrupa, por ejemplo, a: DEC, HP, i-Logix, Intellicorp, IBM, Icon Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI y Unisys.

De esta colaboración nace la descripción del UML versión 1.0 enviada al OMG el 17 de enero de 1997, para su estandarización durante el año de 1997.

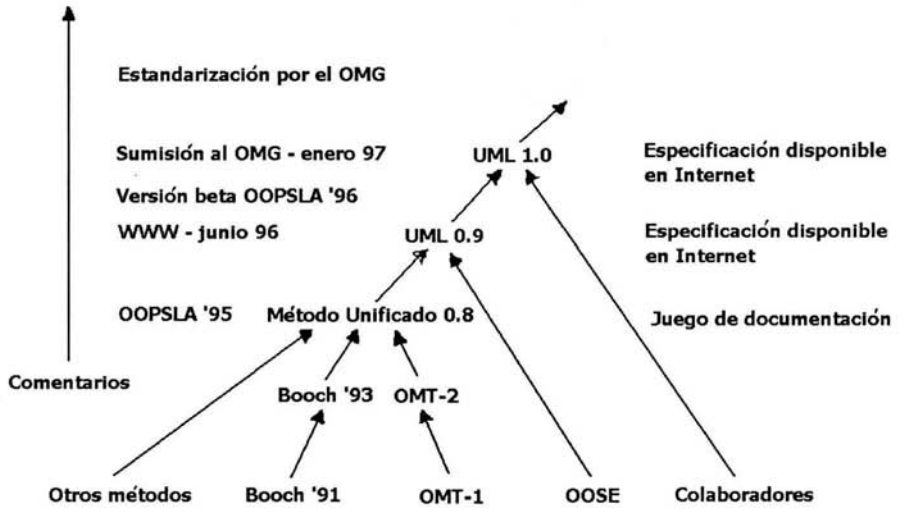


Fig. 1.1 Principales etapas de la definición del UML.

La versión 1.5 del UML es la especificación actual adoptada por el OMG (*Object Management Group*).

El Lenguaje de Modelado Unificado es el sucesor de los métodos de análisis y diseño orientado a objetos (*OOA&D, Object Oriented Analisis and Design*) que aparecieron al final de la década de los 80's y a principios de los 90's.

El UML es un lenguaje de modelado, *no un método,* es decir, **no es un estándar para el proceso de desarrollo**. La mayoría de los métodos consisten, al menos en principio, de un lenguaje de modelado y un proceso. El **lenguaje de modelado** es la notación (principalmente gráfica) que provee un estándar para los artefactos del desarrollo (semántica, modelos, notación sintáctica, y diagramas); y los métodos la usan para expresar sus diseños. El **proceso** es el consejo de qué pasos tomar para realizar una tarea dada.

Las partes de los procesos de muchos métodos son solo bocetos. Cuando la mayoría de la gente dice que está utilizando un método, en realidad usan un lenguaje de modelado, pero raramente siguen un proceso. Visto así, el lenguaje de modelado es de muchas formas la parte más importante de un método. Ciertamente es la llave para establecer la comunicación. Si se desea discutir el diseño de un sistema con alguien, es el lenguaje de modelado el que ambos necesitan entender, no el proceso usado para llegar a ese diseño.

A pesar del valor que un lenguaje de modelado común trae, el desarrollo exitoso de los sistemas complejos de hoy no puede ser realizado únicamente con el uso del UML. El desarrollo exitoso también requiere del empleo de un igualmente robusto proceso de desarrollo.

Booch, Jacobson y Rumbaugh se fijaron cuatro objetivos:

- Representar sistemas completos (más allá de un solo programa) por conceptos de objetos.

- Establecer una relación explícita entre los conceptos y los artefactos ejecutables.
- Tener en cuenta los factores de escala inherentes a los sistemas complejos y críticos.
- Crear un lenguaje de modelado utilizable tanto por los humanos como por las máquinas.

El UML no es una notación propietaria; es accesible para todos, y los fabricantes de herramientas, al igual que las empresas de formación, pueden usarlo libremente.

1.2 Notación y Meta-modelos

La **notación** es el material gráfico que se ve en los modelos; es la sintaxis del lenguaje de modelado. Por ejemplo, en la notación del diagrama de clases se define la representación de conceptos tales como clases, asociaciones, y multiplicidad.

La mayoría de los métodos orientados a objetos tienen un rigor bajo, la notación parece más intuitiva que una definición formal. Estos métodos pueden ser informales, pero mucha gente aún los encuentra útiles; y a fin de cuentas, eso es lo que cuenta.

Sin embargo, la gente está buscando formas de mejorar el rigor de los métodos orientados a objetos sin sacrificar su utilidad. Una forma de hacer esto es definiendo un **meta-modelo** (describe de manera formal los elementos de modelado, la sintaxis y la semántica de la notación que permite manipularlos); un

meta-modelo es un diagrama. La figura 1.2 es una pequeña pieza del meta-modelo del UML que muestra las relaciones entre las clases.

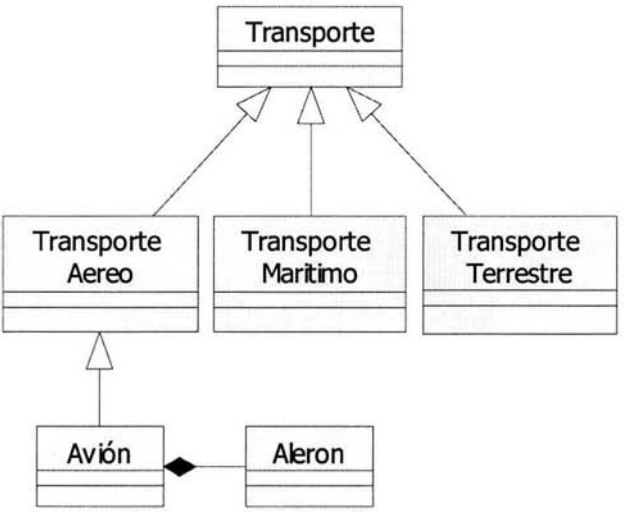


Fig. 1.2 Meta-modelo del UML.

¿Qué tan estricto debe ser el lenguaje de modelado? Eso depende del propósito para el cual se use. Si se tiene una herramienta CASE (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora) que genera código, entonces debe ser tan estricto como para que las interpretaciones de la herramienta del lenguaje de modelado sean correctas y se obtenga un código aceptable. Si se usan los diagramas para propósitos de comunicación, entonces puede ser menos estricto, pero si se extraen partes de la notación oficial, entonces otros desarrolladores no entenderán del todo lo que se está diciendo. Sin embargo, a veces la notación oficial, no puede con el peso de sus necesidades. En ese caso, no se debe tener inconveniente de suavizar el lenguaje. El lenguaje debe ayudar a comunicar, y no lo contrario.

En la construcción del modelo visual de un sistema, diferentes diagramas son necesarios para representar diferentes vistas del sistema. El UML provee una notación rica para la visualización de modelos. Está incluye los siguientes:

- Diagrama de casos de uso; ilustran las interacciones del usuario con el sistema.
- Diagrama de clases; exponen la estructura lógica.
- Diagramas de secuencia y colaboración; muestran el comportamiento y la interacción de las clases.
- Diagrama de paquetes; agrupan las clases y sus dependencias.
- Diagrama de estado; exhiben el comportamiento de clases específicas.
- Diagrama de despliegue; muestran un mapa de la configuración del software y el hardware.

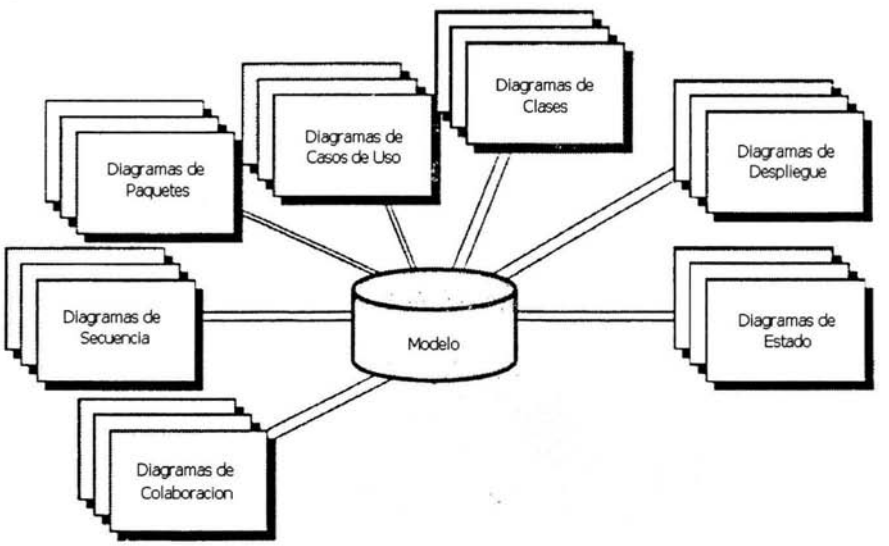


Fig. 1.3 Modelando un sistema desde diferentes perspectivas.

1.3 Modelado Visual

Los **modelos** son abstracciones que retratan lo esencial de un problema complejo o estructura filtrando los detalles no esenciales, haciendo de esta forma el problema más fácil de entender, los modelos simplifican la realidad que describe completamente un sistema desde una perspectiva particular. Se construyen modelos de sistemas complejos porque no se puede comprender tal sistema en su totalidad. La abstracción es una capacidad humana fundamental que nos permite lidiar con la complejidad. Los ingenieros, artistas, y los artesanos han construido modelos por cientos de años para probar los diseños antes de ejecutarlos. El desarrollo de software no podría ser la excepción. Para construir sistemas complejos, el desarrollador debe abstraer diferentes vistas del sistema, construir modelos usando una notación precisa, verificar que los modelos satisfacen los requerimientos del sistema, y gradualmente añadir detalle para transformar los modelos en implementación.

El **modelado visual** es la forma de pensar acerca de los problemas usando modelos organizados con ideas del mundo real. Los modelos son útiles para entender los problemas, comunicarse con todas las personas envueltas en un proyecto (clientes, expertos del dominio, analistas, diseñadores, etc.), modelar el negocio de las empresas, preparar documentación, y diseñar programas y bases de datos. El modelado promueve el mejor entendimiento de las necesidades, diseños más claros, y sistemas más aptos para su manutención.

El modelado es importante porque ayuda al equipo de desarrollo a visualizar, especificar, construir y documentar la estructura y comportamiento de la arquitectura del sistema. Usando un lenguaje de modelado estándar, los diferentes

miembros del equipo de desarrollo pueden sin ambigüedades comunicar sus decisiones a los demás.

La notación juega una parte importante en cualquier modelo, es el pegamento que mantiene al proceso unido.

"La notación juega tres papeles:

- Sirve como el lenguaje para comunicar las decisiones que no son obvias o no pueden ser inferidas del código mismo.
- Provee las semánticas que son lo suficientemente ricas para capturar todas las decisiones de estrategia y tácticas importantes.
- Ofrece una forma concreta suficiente para que los humanos la razonen y las herramientas la manipulen."[6]

Las herramientas de modelado facilitan la administración de estos modelos, permitiendo esconder o mostrar detalles según sea necesario. El modelado visual también ayuda a mantener la consistencia entre los artefactos del sistema, sus requerimientos, diseños e implementaciones. En resumen, el modelado visual ayuda a mejorar la habilidad del equipo para administrar la complejidad del software.

Cuando es acoplado con la práctica del desarrollo de software iterativo, el modelado visual nos ayuda a exponer y valorar los cambios arquitectónicos y comunica aquellos cambios a todo el equipo de desarrollo.

[6] "*Object Solutions*", Booch, Grady. Redwood City, CA: Addison-Wesley, 1995.

Modelar el software visualmente ofrece un número de soluciones a las causas que originan los problemas en el desarrollo de software:

- Los casos de uso y los escenarios especifican el comportamiento sin ambigüedades.
- Los modelos capturan el diseño del software sin ambigüedades.
- El detalle puede ser escondido cuando sea necesario.
- Los diseños sin ambigüedad manifiestan sus inconsistencias prontamente.
- La calidad de la aplicación inicia con un buen diseño.
- Las herramientas de modelado proveen soporte para el modelado con UML.

1.4 Desarrollo Iterativo e Incremental

Los procesos de desarrollo de software clásicos siguen el **ciclo de vida en cascada**, como se ilustra en la figura 1.4. En esta aproximación, el desarrollo procede linealmente desde el análisis de requerimientos a través del diseño, código, unidades de prueba, pruebas del subsistema, y pruebas del sistema.

El problema fundamental de esta aproximación es que empuja el riesgo hacia delante en el tiempo resultando costoso deshacer los errores de fases tempranas. Un diseño inicial puede ser defectuoso con respecto a sus requerimientos, y, además, el descubrimiento tardío de estos defectos tiende a resultar en costosas modificaciones o la cancelación del proyecto. Como Tom Gilb adecuadamente dijo, "si no ataca activamente los riesgos en su proyecto, ellos lo atacaran activamente"[7]. El desarrollo en cascada tiende a disfrazar los riesgos reales de un proyecto hasta que es demasiado tarde para hacer algo significativo por ellos.

[7] "*Principles of Software Engineering Management*", Gilb, Tom. Harlow, UK: Addison-Wesley, 1998.

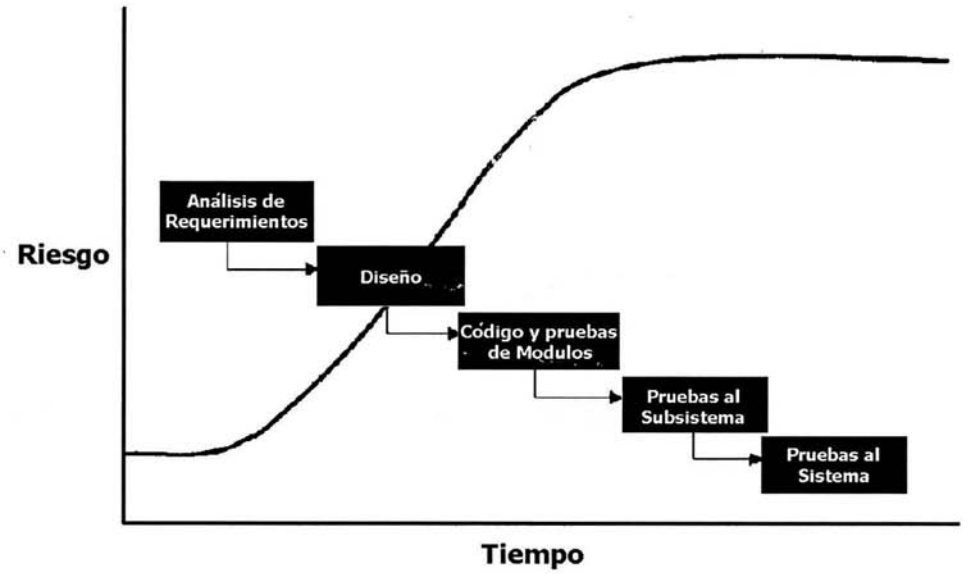


Fig. 1.4 Ciclo de Vida en Cascada.

Como una respuesta al prolongado tiempo de desarrollo requerido para producir un producto final, fue propuesto el modelo de **desarrollo iterativo**, este modelo es una alternativa al desarrollo en cascada. El modelo de desarrollo iterativo reconoció que muchos de los escenarios de desarrollo podían ser hechos en paralelo, en lugar de esperar a que la fase de diseño fuese completada.

Este ciclo de vida refleja con más veracidad las actividades de desarrollo durante un proyecto, y es uno de los más completos usado por las organizaciones de desarrollo. Este ciclo de vida fue adicionalmente refinado por Tom Gilb, en su Entrega Evolucionaría (*Evolutionary Delivery*, llamada *EVO*)[8], que intenta entregar

[8] "*http://c2.com/cgi/wiki?EvolutionaryDelivery*", la propuesta de Tom Gilb al principio de los ochentas para el desarrollo de subsistemas de software utilizables en pasos muy pequeños, con los requerimientos del usuario siendo continuamente reevaluados después de cada entrega.

muchas versiones de un producto parcial, cada una con funcionalidad incrementada. Cada fase es un mini proyecto desarrollado con características propias y cada iteración consiste de uno o más de los siguientes componentes del proceso; captura de requerimientos, análisis, diseño, implementación y pruebas.

Fig. 1.5 El Modelo Espiral de Barry Boehm.

Los desarrolladores asumen que no todos los requerimientos son conocidos al inicio del ciclo de vida; realmente el cambio es anticipado a través de todas las fases. Tom sugiere que "los usuarios no saben en realidad lo que quieren en un principio, así que cada iteración significa averiguar una parte, desarrollar esa parte, averiguar un poco más, y así sucesivamente"[9].

[9] "*Principles of Software Engineering Management*", Gilb, Tom. Harlow, UK: Addison-Wesley, 1998.

En esta aproximación, basada en el trabajo del modelo espiral de Barry Boehm[10] (ver figura 1.5), la identificación de los riesgos de un proyecto es forzada tempranamente en el ciclo de vida, siendo así posible atacar y reaccionar a tiempo de una manera eficiente.

Los términos desarrollo "iterativo" e "incremental" son usualmente usados para describir esta aproximación, especialmente en los círculos orientados a objetos. Hablando estrictamente, el **desarrollo iterativo** se refiere al proceso de evolucionar una pieza de funcionalidad revisitandola un número *n* de veces, mientras que el **desarrollo incremental** describe construir un sistema añadiendo nueva funcionalidad, pieza por pieza. En la práctica, los dos procesos son indistinguibles, y el término *desarrollo iterativo* es comúnmente el más usado.

Los métodos orientados a objetos tienden a ser particularmente apropiados en el desarrollo iterativo.

El proceso de planeación de la iteración es el siguiente:

1. Identificar y priorizar los mayores riesgos del proyecto.
2. Seleccionar un pequeño número de escenarios los cuales direccionan directamente los riesgos de mayor prioridad.
3. Los escenarios seleccionados son usados por:
    a. Los desarrolladores para identificar que será implementado en la iteración.
    b. El equipo de pruebas para desarrollar los planes de pruebas y procedimientos para la iteración.

[10] "*A Spiral Model of Software Development and Enhancement*", Boehm, Barry W. IEEE Computer, Mayo de 1998.

4. Al final de la iteración
    a. Determinar que riesgos han sido reducidos o eliminados.
    b. Determinar si algún riesgo nuevo ha sido descubierto.

Los riesgos son valorados, priorizados y revisados durante el desarrollo de cada iteración. La construcción del sistema de esta forma expone y mitiga los riesgos del sistema temprano en el ciclo de vida.

El sistema crece al incorporar nuevas funciones en cada ciclo de desarrollo. Tras una fase preliminar de planeación y especificación, el desarrollo pasa a la fase de construcción a través de una serie de ciclos de desarrollo (ver figura 1.6). En cada ciclo se aborda un conjunto relativamente pequeño de requerimientos pasando por el análisis, el diseño, la implementación y las pruebas.

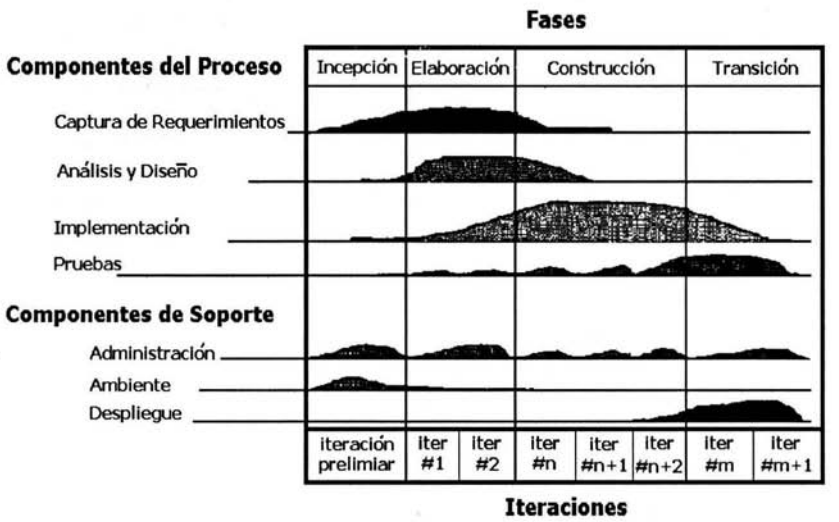


Fig. 1.6 Fases del Proceso de Desarrollo Iterativo.

Esto contrasta con el ciclo clásico de la vida en cascada, en el cual las actividades (análisis y diseño, entre otras) se llevan a cabo una vez con todos los requerimientos del sistema.

Entre las ventajas del desarrollo iterativo figuran las siguientes:

- La complejidad nunca resulta abrumadora.
- Se produce retroalimentación en una etapa temprana, porque la implementación se efectúa rápidamente con una parte pequeña del sistema.

CAPÍTULO 2

Notación UML

2.1 Diagrama de Casos de Uso

El modelo de casos de uso captura todos los requisitos funcionales del sistema. Se define un **caso de uso** de manera precisa como sigue: "Un caso de uso especifica una secuencia de acciones, incluyendo variantes, que el sistema puede llevar a cabo, y que producen un resultado observable de valor para un usuario (actor) concreto"[11].

Se identifican los casos de uso examinando como los usuarios necesitan utilizar el sistema para realizar su trabajo. Cada uno de esos modos de utilizar el sistema es un caso de uso candidato. Estos candidatos se ampliarán, se cambiarán, se dividirán en casos de uso más pequeños, o se integrarán en casos de uso más completos. El modelo de casos de uso está terminado cuando recoge todos los requisitos funcionales correctamente de un modo sencillo y claro que puedan comprender los clientes, usuarios y desarrolladores.

---

[11] *"Visual Modeling with Rational Rose and UML"*, Quatrany, Terry. Addison-Wesley

La secuencia de acciones realizada por un caso de uso durante su operación (es decir, una *instancia* o *realización* del caso de uso) es un camino específico a través del caso de uso. Puede haber muchos caminos, muchos de ellos muy parecidos, estos son variantes de la ejecución de la secuencia de acciones especificada en el caso de uso. Para hacer que un modelo de casos de uso sea más comprensible, se agrupan las descripciones de caminos variantes parecidas en un solo caso de uso. Cuando se identifica y describe un caso de uso, realmente se está diciendo que se identifican y describen los diferentes caminos que es práctico definir como un solo caso de uso.

Además de introducir los casos de uso como elementos primarios en el desarrollo de software, Jacobson también introdujo un diagrama para visualizar los casos de uso.

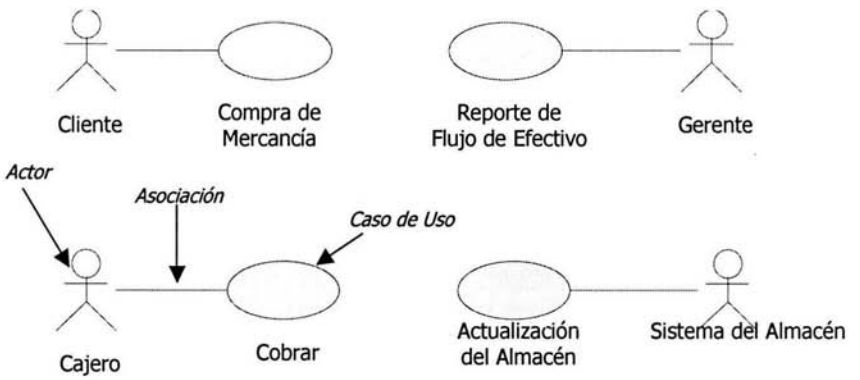


Fig. 2.1 Diagrama de Casos de Uso.

2.1.1 Actores

Un **actor** es un papel que un usuario juega con respecto al sistema. Los actores son la gente, las organizaciones, los sistemas, o los dispositivos que usan o interactúan con el sistema. Hay cuatro actores en la figura 2.1: *Cliente*, *Cajero*, *Gerente*, y *Sistema del Almacén*.

Pueden existir muchos cajeros en la organización, pero para el sistema, todos juegan el mismo papel. Un usuario puede jugar más de un papel. Cuando se trata con los actores, es más importante pensar en los papeles que en la gente o los puestos de trabajo.

Se hace referencia a que los actores no necesitan ser personas, aún cuando los actores sean representados como figuras de palo dentro de un diagrama de casos de uso. Un actor puede ser también un sistema externo que aporta o requiere alguna información del sistema actual. En la figura 2.1 se puede ver la necesidad de actualizar el *Sistema del Almacén*.

Los casos de uso tratan de la funcionalidad externa requerida. Si el *Sistema de Almacén* necesita un archivo, ese es un requerimiento que debe satisfacerse.

2.1.2 Realizaciones

Los actores llevan a cabo los casos de uso. Un simple actor puede *realizar* o *instanciar* muchos casos de uso, recíprocamente, un caso de uso puede tener varios actores realizándolo.

Existe más de una manera de llevar a cabo un caso de uso. Esto dicho en el lenguaje de UML, se dice que un caso de uso puede tener muchas **instancias** o **realizaciones.**

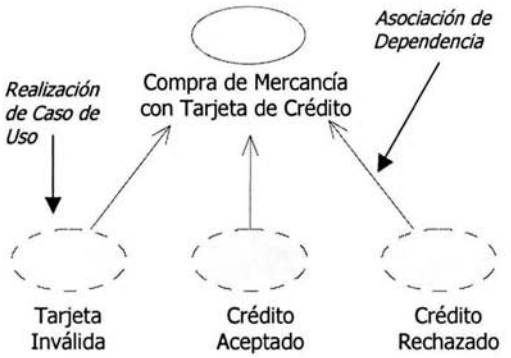


Fig. 2.2 Realizaciones de Caso de Uso.

2.1.3 Agregación y Generalización

Además de las relaciones entre los actores y los casos de uso, existen dos tipos más de relaciones. Estas representan las relaciones de **agregación (usa)** y **generalización (extiende)** entre los casos de uso.

La relación de **agregación** ocurre cuando se tiene una parte del comportamiento que es similar entre más de un caso de uso y no se desea copiar la descripción de dicho comportamiento. Por ejemplo, *Compra de Mercancía* y *Devolución de Mercancía* requieren de seleccionar el artículo a comprar o devolver (*Seleccionar Artículo*).

Se utiliza la relación de **generalización** cuando se tiene un caso de uso que es similar a otro pero éste hace un poco más; es decir, el caso de uso *extiende* o *hereda* de otro caso de uso.

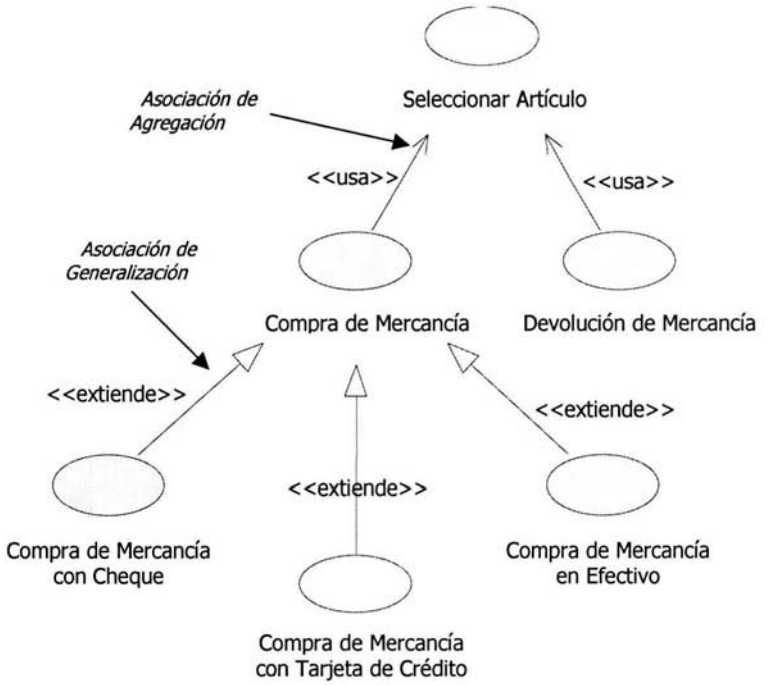


Fig. 2.3 Agregación y Generalización de Casos de Uso.

En este ejemplo, el caso de uso básico es *Compra de Mercancía*. Este es el caso cuando todo es fácil. Existen factores que pueden complicar las cosas, uno de ellos es cuando la compra se realiza con tarjeta de crédito (*Compra de Mercancía con Tarjeta de Crédito*) o cheque (*Compra de Mercancía con Cheque*).

Se notan las similitudes entre las relaciones de agregación y generalización. Ambas requieren de factorizar el comportamiento común de varios casos de uso en

uno más abstracto, que es *usado* o *extendido*, por otros casos de uso. Sin embargo la *intención* es diferente.

Los dos tipos de relaciones implican diferentes cosas en sus conexiones con los actores. Con la relación de agregación, frecuentemente no existe un actor asociado al caso de uso común, sin embargo, en el caso de la generalización, los actores tienen una relación con el caso de uso que es extendido. Se asume que el actor ejecutará el caso de uso base y todas sus extensiones.

Se aplican las siguientes reglas para el uso de estas asociaciones:

- Se emplea la agregación cuando se está repitiendo el comportamiento en dos o más casos de uso separados y se desea evitar esa repetición.
- Se utiliza la generalización cuando se está describiendo una variación en el comportamiento normal.

2.1.4 Escenario

Dentro del UML, el término **escenario** se refiere a un camino singular a través del caso de uso, uno que muestra una combinación particular de condiciones. Un escenario es una instancia de un caso de uso. Los escenarios son construidos basándose en los casos de uso y en el diagrama de clases; el diagrama de casos de uso provee el contexto y el diagrama de clases provee las entidades del negocio.

Por ejemplo, si se quisieran comprar mercancías, se tendría un caso de uso único con varios escenarios asociados: uno en el cual todo está bien, otro en

donde no existen suficientes mercancías; uno en el que el crédito no es aceptado; y así entre otros.

2.2 Diagrama de Clases

2.2.1 Clases

Las **clases** se representan por rectángulos compartimentados. El primer compartimento contiene el nombre de la clase. El nombre de la clase debe permitir comprender lo que es la clase, y no lo que hace. Una clase no es una función, una clase es una descripción abstracta de un conjunto de objetos del ámbito de la aplicación. Los otros dos compartimentos contienen, respectivamente, los atributos y las operaciones de la clase.

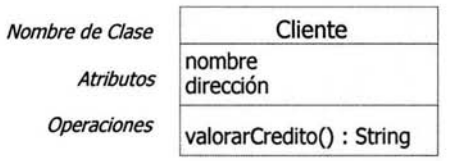


Fig. 2.4 Representación gráfica de las Clases.

2.2.1.1 Atributos

Los **atributos** son una definición de datos contenidos por instancias de una clase, no tienen comportamiento, son de un tipo de dato definido y opcionalmente tienen un valor inicial. Un atributo indica que un objeto es responsable de conocer una cierta pieza de información.

2.2.1.2 Operaciones

Las **operaciones** son los procesos que una clase sabe llevar a cabo. Estos corresponden a los métodos de una clase. Una operación indica que un objeto es responsable de proveer una cierta pieza de funcionalidad al sistema.

2.2.1.3 Visibilidad

La idea más simple es que cualquier clase tiene elementos públicos y privados. Los elementos públicos pueden ser usados por cualquier otra clase; los elementos privados pueden ser usados solo por la clase propietaria.

El UML provee tres abreviaciones para la visibilidad: + (público), - (privado), y # (protegido).

- Un miembro público es visible en cualquier ámbito del programa y puede ser llamado por cualquier objeto dentro del sistema.
- Un miembro privado solo puede ser usado por la clase que lo define.
- Un miembro protegido puede ser usado por (a) la clase que lo define o (b) una subclase de la clase.

2.2.2 Diagrama de Clases

Un **diagrama de clases** describe los tipos de objetos en el sistema y los diversos tipos de relaciones estáticas que existen entre ellos. Existen dos tipos de relaciones estáticas:

- Asociaciones (por ejemplo, un cliente puede rentar un número de videos).
- Subtipos (una enfermera es un tipo de persona).

El diagrama de clases muestra también los atributos y las operaciones de una clase.

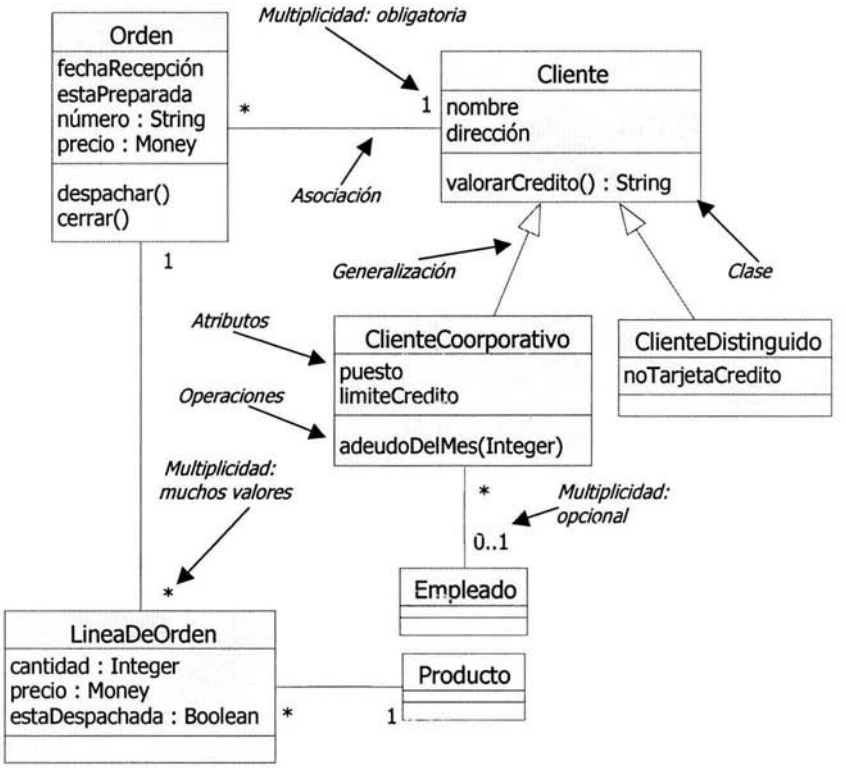


Fig. 2.5 Diagrama de Clases.

2.2.2.1 Asociaciones

Las **asociaciones** representan las relaciones entre las instancias de clases. El diagrama de la figura 2.5 indica que una *Orden* debe de venir de un solo *Cliente* y que el cliente puede hacer varias *Ordenes* eventualmente.

Una asociación tiene **multiplicidad**, la cual es una indicación de cuantos objetos pueden participar en la relación dada. El * (asterisco) entre *Cliente* y *Orden* indica que el *Cliente* puede tener muchas *Ordenes* asociadas con él; el 1 indica que una *Orden* viene de un solo *Cliente*.

En general, la multiplicidad indica límites inferiores y superiores para los objetos participantes. El * (asterisco) en realidad representa el rango *0..infinito*; un *Cliente* no requiere de tener una *Orden* colocada, y no existe un limite superior al número de *Ordenes* que un *Cliente* puede colocar. El 1 delimita para *1..1*; una *Orden* debe ser colocada por exactamente un *Cliente*.

Para una multiplicidad más general, se puede definir un simple número (por ejemplo *11* para los jugadores de un equipo de fútbol soccer), un rango (tal como *2..4* para jugadores de un juego de canasta), o combinaciones discretas de los números y rangos (tales como *2,4* para las puertas de un automóvil).

La tabla 2.1 representa los valores de multiplicidad convencionales:

1	Uno y sólo uno
0..1	Cero o uno
M..N	De M a N (enteros naturales)
*	De cero a muchos
0..*	De cero a muchos
1..*	De uno a muchos

Tabla 2.1 Multiplicidad en las Asociaciones.

Ahora se podría implicar que existen punteros en ambas direcciones entre las clases relacionadas. El diagrama diría que la clase *Orden* tiene un atributo que

es una colección de punteros a *LineaDeOrden* y también un puntero a *Cliente*. En Java, se inferiría algo como lo siguiente:

```
class Orden {
        private Cliente _cliente;
        private Vector _lineaDeOrden;
}

class Cliente {
        private Vector _ordenes;
}
```

La figura 2.6 es básicamente la misma que la figura 2.5, excepto que se han añadido un par de flechas en las líneas de asociación. Estas flechas indican **navegabilidad**.

Esto indica que una *Orden* tiene la responsabilidad de decir a cual *Cliente* pertenece, pero un *Cliente* no tiene la habilidad correspondiente de decir que *Ordenes* tiene. En lugar de responsabilidades simétricas, ahora se tienen responsabilidades en un solo lado de la línea. Esto podría expresarse de la siguiente manera; una *Orden* apunta hacia un *Cliente*, pero un *Cliente* no apunta hacia una *Orden*.

Si existe la navegabilidad en un solo sentido, se llama a la asociación **uni-direccional**. Una asociación **bi-direccional** contiene navegabilidad en ambos sentidos. El UML dice que las asociaciones sin flechas pretenden significar ya sea que la navegabilidad es desconocida o que la asociación es bi-direccional.

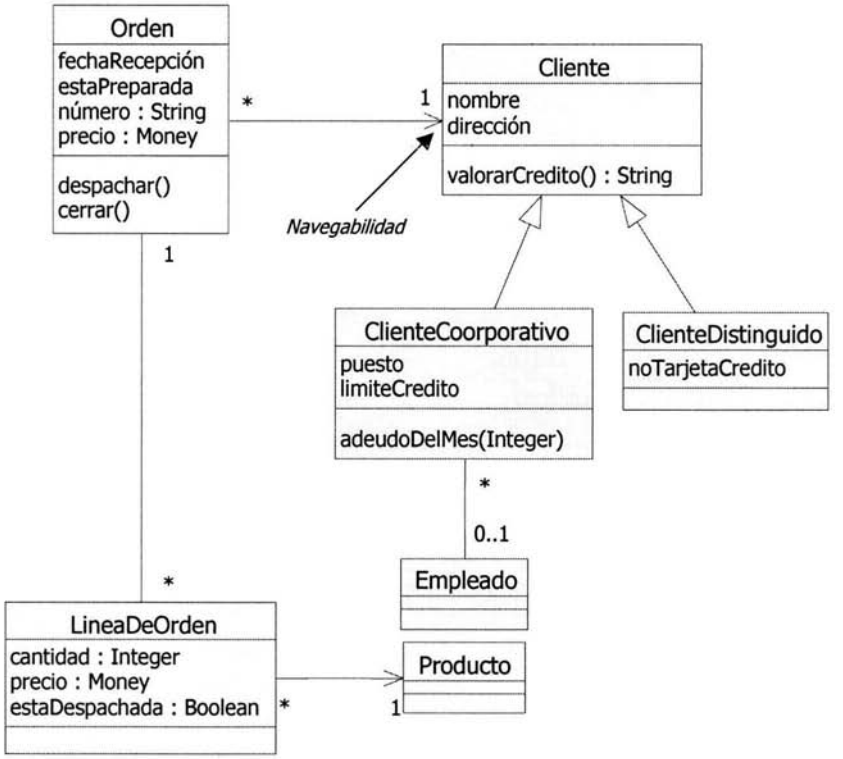


Fig. 2.6 Diagrama de Clases con navegabilidad.

2.2.2.2 Generalización

La **generalización** está asociada con la herencia en los lenguajes de programación. La subclase hereda todos los métodos y los atributos de la superclase y puede sobre escribir los métodos heredados.

El UML emplea el término generalización para designar la relación de clasificación entre un elemento más general y un elemento más específico. En realidad, el término generalización designa un punto de vista centrado sobre un árbol de clasificación. Por ejemplo, un animal es un concepto más general que un

gato, un perro o un ratón. Inversamente, un gato es un concepto más especializado de un animal. La relación de generalización significa *es un o es una especie de*; un gato *es un* animal.

La relación de generalización se representa por medio de una flecha que apunta de la clase más especializada hacia la clase más general. La punta de la flecha se realiza por un pequeño triangulo vacío, lo que permite distinguirla de una flecha abierta, símbolo de la propiedad de navegación de las asociaciones.

Los atributos, las operaciones, las relaciones y las restricciones definidas en las superclases se heredan íntegramente en las subclases.

2.2.2.3 Agregación y Composición

La **agregación** es el parte-de la relación. Esto es como decir que un coche tiene un motor y llantas como sus partes.

El UML ofrece una variedad más fuerte de agregación, llamada composición. Con la **composición**, la parte del objeto puede pertenecer a *un solo* todo; por ello, las partes usualmente viven y mueren con el todo. Cualquier borrado del todo es considerado una cascada de las partes.

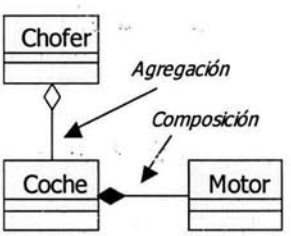


Fig. 2.7 Agregación y Composición.

En la figura 2.7 cuando el *Coche* es creado o instanciado también se crea un *Motor* que le pertenece solo al *Coche*, si el *Coche* es destruido el *Motor* por consecuencia deja de existir, la relación del *Coche* con el *Chofer* es de agregación ya que el *Coche* es operado por un *Chofer* a la vez, pero si el *Coche* es destruido, el *Chofer* sigue existiendo.

2.3 Diagramas de Interacción

Los **diagramas de interacción** son modelos que describen como los grupos de objetos colaboran en algún comportamiento. Típicamente, un diagrama de interacción captura el comportamiento de un solo caso de uso. El diagrama muestra objetos de ejemplo y los mensajes que son pasados entre estos objetos dentro del caso de uso.

Existen dos tipos de diagramas de interacción: diagrama de secuencia y diagrama de colaboración.

2.3.1 Diagrama de Secuencia

El **diagrama de secuencia** esencialmente contiene la misma información que el escenario, pero es expresado de una manera más formal. Normalmente el esquema formal del diagrama de secuencia promueve ir a un mayor detalle que el del escenario; consecuencia de esto, se encuentra que el escenario estaba incorrecto o incompleto. Este es uno de los principales motivos del diagrama de secuencia; descubrir ambigüedades u omisiones en los escenarios.

El diagrama de secuencia contiene objetos, no clases; esta es una diferencia del diagrama de clases, el cual contiene clases que significan la existencia de objetos. Si más de un objeto de una clase en particular debe de aparecer en el diagrama de secuencia, múltiples encabezados y líneas verticales son requeridas para mostrar cada objeto claramente. Por ejemplo, un escenario que describe una transferencia de fondos de una cuenta a otra debería mostrar ambas cuentas como objetos individuales en el diagrama de secuencia.

Los objetos en un diagrama de secuencia provienen de los sustantivos de los escenarios, y también de cualquier diagrama de clase creado hasta el momento. Algunos sustantivos de los escenarios pueden ser idénticos a las clases del diagrama de clases; otros pueden ser sinónimos y necesitan ser reconocidos como tales; y otras pueden ser nuevas clases que deberían ser añadidas al diagrama de clases.

La mayoría de los escenarios son iniciados por un actor primario. Algunos escenarios tienen actores secundarios que están involucrados, pero que no inician los procedimientos. Algunos escenarios no tienen actores del todo, pero son iniciados por el sistema mismo, usualmente cuando algún nivel ha sido rebasado, tal como el punto de ser el cliente un millón.

La razón primaria del diagrama de secuencia es el identificar las operaciones en las clases. Es bastante difícil hacer esto directamente del diagrama de clases estático, ya que el diagrama de clases no contiene ningún contexto para el uso de los objetos. Los casos de uso, los escenarios y los diagramas de secuencia atacan el problema preguntando lo siguiente:

- ¿Qué quiere el usuario que haga el sistema?
- ¿Qué objetos están involucrados?

- ¿Qué comportamiento debe estar presente dentro del sistema para proveer lo que el usuario necesita?
- ¿Cómo puede el comportamiento ser asignado para identificar a los objetos?

Los **mensajes** son las comunicaciones entre los objetos. Desde el punto de vista del emisor, un evento le pide al receptor hacer algo, o le dice al receptor que algo ha ocurrido. Desde el punto de vista del receptor, es una orden que debe ser obedecida, o una notificación de que algo ha sucedido.

Un diagrama de secuencia muestra interacciones en secuencias de tiempo. Pinta los objetos y clases implicadas en el escenario y la secuencia de los mensajes intercambiados entre los objetos necesitados para llevar a cabo la funcionalidad del escenario.

En el UML, un objeto en un diagrama de secuencia es dibujado como un rectángulo conteniendo el nombre del objeto subrayado. Un objeto puede ser nombrado en una de tres formas: el nombre del objeto, el nombre del objeto y su clase, o solo el nombre de la clase (objeto anónimo). Las tres denominaciones de un objeto se muestran en la figura 2.8.

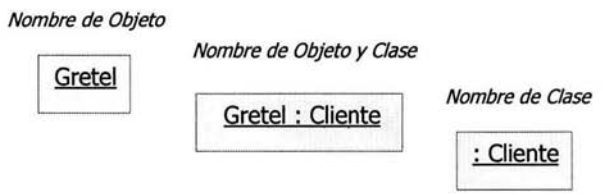


Fig. 2.8 Nombrando objetos en un Diagrama de Secuencia.

El diagrama de secuencia muestra interacciones entre objetos según un punto de vista temporal. Un diagrama de secuencia representa una interacción entre objetos insistiendo en la cronología de los envíos de mensajes. Un objeto se materializa por un rectángulo y una barra vertical llamada línea de vida de los objetos.

Los objetos se comunican intercambiando mensajes. Cada mensaje es representado por una flecha entre las líneas de vida de dos objetos. El orden de envío de los mensajes viene dado por la posición sobre el eje vertical. Las medias flechas indican un **mensaje asíncrono**; este tipo de mensajes no bloquean al emisor, y puede seguir con su propio proceso. Un mensaje asíncrono puede hacer una de tres cosas.

1. Crear un nuevo hilo de ejecución.
2. Crear un nuevo objeto.
3. Comunicarse con un hilo de ejecución que ya esté corriendo.

Cada mensaje es etiquetado al menos con el nombre del mensaje. Un objeto puede enviarse también un mensaje así mismo (**auto-delegación**). Esta situación se representa por una flecha que traza un bucle sobre la línea de vida del objeto. Esta construcción no corresponde siempre a un verdadero mensaje; puede indicar un punto de entrada en una actividad de más bajo nivel, que se ejerce dentro del objeto. El diagrama incluye un **regreso**, el cual indica el regreso de un mensaje, no un mensaje nuevo.

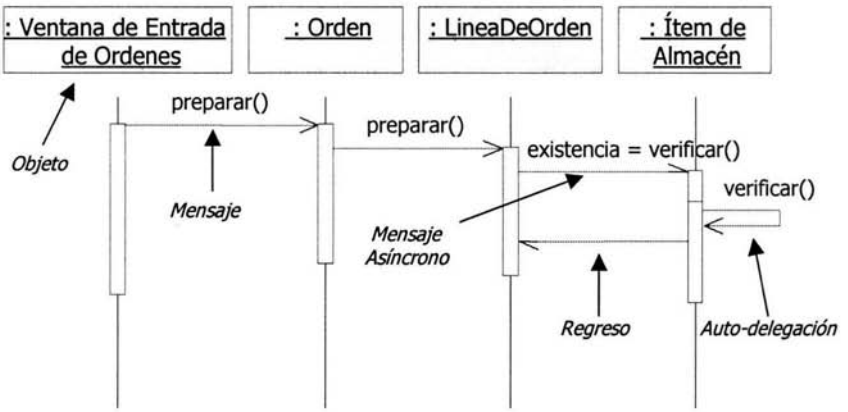


Fig. 2.9 Diagrama de Secuencia.

En el modelado de objetos, el diagrama de secuencia se utiliza de dos maneras diferentes, según la fase del ciclo de vida y el nivel de detalle deseado.

La primera utilización corresponde a la documentación de los casos de uso; se concentra sobre la descripción de la interacción, a menudo en términos próximos al usuario y sin entrar en los detalles de la sincronización. La indicación que acompaña a las flechas corresponde entonces a eventos que ocurren en el ámbito de la aplicación. En esta etapa del modelado, las flechas no corresponden aún a envíos de mensajes en el sentido de lenguajes de programación.

La segunda utilización corresponde a un uso más informático y permite la representación precisa de las interacciones entre los objetos. El concepto de mensaje unifica todas las formas de comunicación entre objetos, en particular la llamada de procedimiento.

2.3.2 Diagrama de Colaboración

En el **diagrama de colaboración**, los objetos de ejemplo son mostrados como íconos. Al igual que en el diagrama de secuencia, las flechas indican que los mensajes son enviados en un caso de uso dado. El tiempo no se representa de manera implícita, como en un diagrama de secuencia, de modo que los diferentes mensajes se numeran para indicar el orden de los envíos. Numerar los mensajes hace más difícil ver la secuencia que colocar las líneas hacia abajo en la página. Un **mensaje** se representa por una flecha colocada cerca de un enlace y dirigida hacia el objeto destinatario del mensaje. Un **enlace** sirve de soporte de transmisión para el mensaje. Un mensaje desencadena una acción en el objeto destinatario.

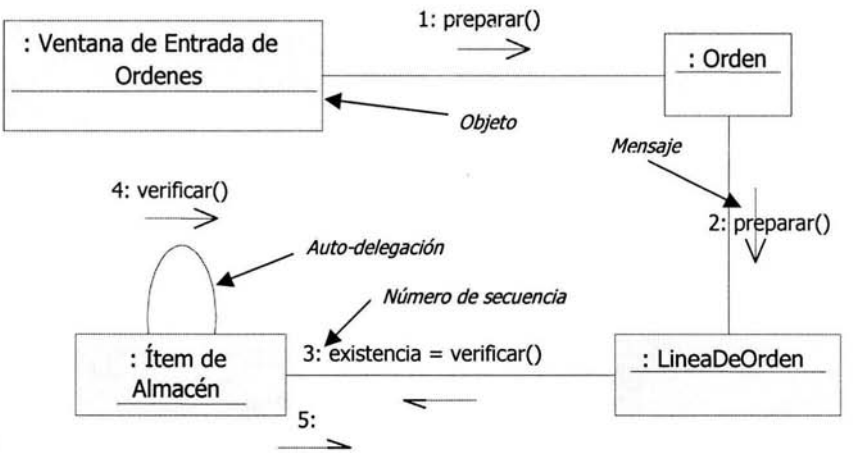


Fig. 2.10 Notación UML para Objetos y Mensajes en un Diagrama de Colaboración.

Los nombres del esquema son de la forma *nombreObjeto: NombreClase*, el nombre del objeto puede ser omitido, no así el de la clase.

La diferencia básica entre el diagrama de secuencia y el de colaboración es que el primero hace énfasis en el orden en que los sucesos ocurren; y el segundo indica como los objetos están estáticamente conectados.

2.4 Diagrama de Paquetes

Los grandes sistemas requieren la definición de muchas clases, esto conduce a diagramas de clases muy complejos que ocupan muchas páginas de notación. Visto como un todo el sistema, representaría una gran superficie de área, lo que haría difícil su comprensión. Es por esta razón que el sistema se parte en pequeños subsistemas, es decir, en relaciones muy cercanas de porciones del diagrama de clases que pueden ser encapsuladas y abstraídas en grupos separados de clases llamados **paquetes**.

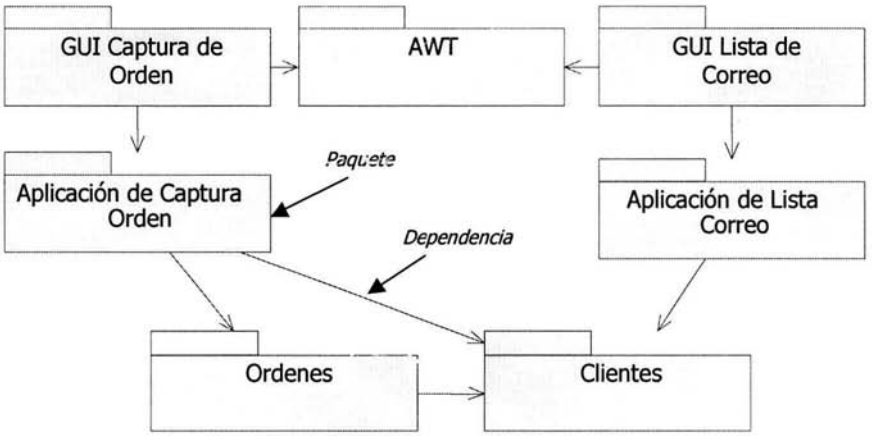


Fig. 2.11 Diagrama de Paquetes.

El término diagrama de paquetes es usado para un diagrama que muestra paquetes de clases y las dependencias entre ellos. Los paquetes y las dependencias son elementos en un diagrama de clases, luego entonces, un diagrama de paquetes es solo una forma de diagrama de clases.

Cada paquete provee una función principal, y puede ser identificada del modelo de análisis buscando clases fuertemente acopladas o clases funcionalmente relacionadas. Las clases tienden a "agruparse", particularmente alrededor de jerarquías de agregación o herencia; estas "agrupaciones" pueden ser vagamente asociadas con otras "agrupaciones".

Los paquetes son usados para particionar grandes modelos en partes manejables. Cada paquete contiene clases relacionadas, asociaciones y generalizaciones, y cada clase pertenece a un solo paquete. Los paquetes no tienen un contenido semántico, solo son usados como una forma de organización del modelo.

Los grandes modelos pueden ser particionados de una forma descendente al principio del, o incluso precediendo, al análisis, o puede ser particionada de una forma ascendente agrupando clases relacionadas durante el análisis o diseño. Paquetes adicionales orientados al diseño probablemente aparecerán más tarde, conteniendo por ejemplo, interfases gráficas de usuario o clases de base de datos.

Un paquete forma un nombre para la clase; el nombre de la clase debe ser único dentro de un paquete. Las clases pueden tener el mismo nombre solo si vienen de diferentes paquetes; en este caso son calificadas con el nombre del paquete *nombrePaquete::nombreClase*. Un paquete puede contener otros paquetes, permitiendo a los grandes sistemas ser particionados de una forma

anidada. La denominación de la clase sería de la forma *superPaquete::subPaquete::clase*. El sistema en si puede verse como un paquete de alto nivel.

2.5 Diagrama de Estados

2.5.1 Estado

Un **estado** es una propiedad durante la vida de un objeto cuando satisface alguna condición, realiza alguna acción, o espera por algún evento. El estado de un objeto puede ser caracterizado por el valor de uno o más de los atributos de la clase.

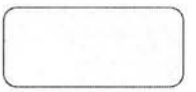


Fig. 2.12 Notación UML para un Estado.

2.5.2 Transición de Estado

Una **transición de estado** representa el cambio de un estado original a un estado sucesor (el cual puede ser el mismo al estado original) y es representada por una flecha que apunta del estado original al estado sucesor. Una acción puede acompañar una transición de estado.

Existen dos formas de hacer una transición de estado, **automática** y **no automática**. Una transición de estado automática ocurre cuando la actividad del

estado original es completada, no existe un nombre de evento asociado con la transición. Una transición de estado no automática es causada por un evento.

Una transición puede tener una acción y/o una condición de guarda asociada con ella, y puede también disparar un evento. La sintaxis para la etiqueta de una transición tiene tres partes, las cuales son todas opcionales: *Evento [Guarda] / Acción*. Un **evento** es un mensaje que es enviado a otro objeto en el sistema. Una **guarda** es una condición lógica que valida o no el desencadenamiento de una transición en la ocurrencia de un evento. Una **acción** es el comportamiento que acontece cuando el estado de transición ocurre. Cuando el evento tiene lugar, las guardas, que deben ser mutuamente excluyentes, se evalúan, validan y desencadenan una transición. Las acciones y las guardas son ambas comportamientos del objeto y se convierten en operaciones.

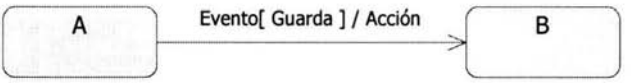


Fig. 2.13 Sintaxis de la Etiqueta de una Transición.

Las acciones que acompañan a las transiciones de estado hacia el estado pueden ser colocadas como **acciones de entrada** dentro del estado. Así también, las acciones que acompañan a las transiciones de estado fuera del estado pueden ser colocadas como **acciones de salida** dentro del estado. El comportamiento que ocurre dentro del estado es llamado **actividad**. Una actividad comienza cuando se entra al estado y cualquiera se complete o sea interrumpida por una transición de estado saliente. El comportamiento puede ser una simple acción o puede ser un envío de evento a otro objeto. La notación UML para la información de estado se aprecia en la figura 2.14.

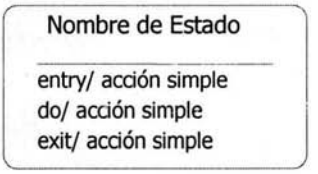


Fig. 2.14 Detalles del Estado.

2.5.3 Estados especiales

Existen dos estados especiales que son añadidos al diagrama de transición de estados. El primero es un **estado inicial**. Cada diagrama debe tener uno y solo un estado inicial ya que el objeto debe estar en un estado consistente cuando es creado. La notación UML para un estado inicial es un pequeño círculo sólido, como se muestra en la figura 2.15. El segundo estado especial es un **estado final**. Un objeto puede tener o no un estado final, o puede tener múltiples estados finales. La notación UML para un estado final es un ojo de toro.

Fig. 2.15 Notación UML de los estados Inicial y Final.

2.5.4 Diagrama de Transición de Estados

Los casos de uso y los escenarios proveen una forma de describir el comportamiento del sistema; esto es, la interacción entre los objetos en el sistema. Un **diagrama de transición de estados** o **diagrama de estados** muestra los estados de un objeto en particular, los eventos o mensajes que causan la transición de un estado al otro, y las acciones que resultan del cambio de estado.

El diagrama de estado es útil para describir el comportamiento de un objeto a través de varios casos de uso. Por el contrario, no lo es para describir el comportamiento de un número de objetos que colaboran entre sí. Un diagrama de transición de estados no debe ser creado para cada clase en el sistema. Son creados únicamente para clases con comportamiento dinámico significante. Las interfases de usuario y los objetos de control tienen la clase de comportamiento que es útil representar con un diagrama de estado.

La figura 2.16 muestra un diagrama de estado de UML para la clase *Orden* en el sistema de procesamiento. El diagrama indica los diferentes estados de una *Orden*.

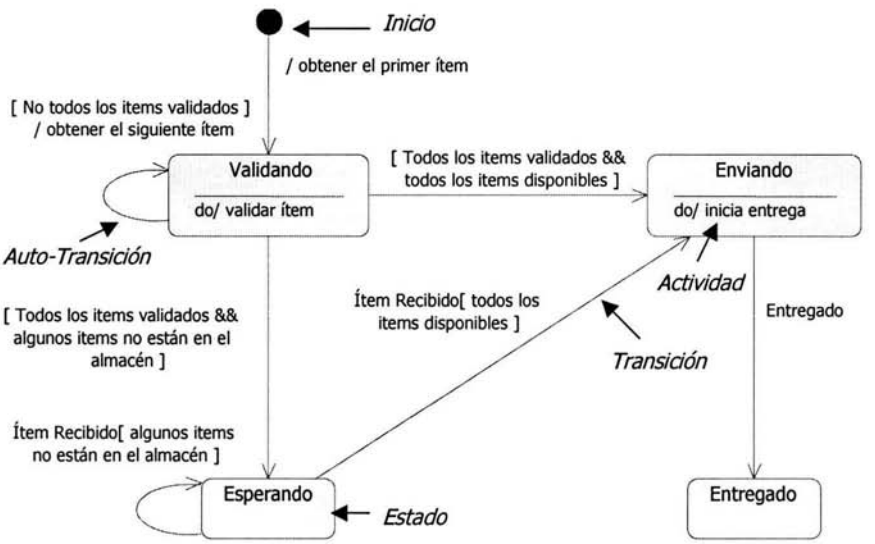


Fig. 2.16 Diagrama de Estado.

Se comienza en el punto de inicio y se muestra una transición inicial en el estado *Validando*. La transición es etiquetada con la acción */obtener el primer*

ítem. Una vez ejecutada la acción, se entra al estado *Validando*. Este estado tiene una actividad asociada con él, la actividad es denominada *validar ítem*.

Tres transiciones salen del estado *Validando*. Las tres contienen únicamente guardas en su etiqueta.

1. Si no se han validado todos los ítems, se obtiene el siguiente ítem y se regresa al estado *Validando* a validarlo.
2. Si se validaron todos los ítems y todos están disponibles, se hace la transición al estado *Enviando*.
3. Si validamos todos los ítems pero no todos están disponibles, se hace la transición al estado *Esperando*.

Se observará primero el estado *Esperando*. No existen actividades para este estado, luego entonces, este estado se encuentra esperando por un evento. Ambas transiciones que salen del estado *Esperando* son etiquetadas con el evento *Ítem Recibido*. Esto significa que la *Orden* espera hasta que detecta este evento. En este punto, evalúa las guardas en las transiciones y hace la transición apropiada (ya sea a *Enviando* o regresa a *Esperando*).

Dentro del estado *Enviando*, se tiene una actividad que inicia una entrega. Existe también una transición sencilla, una transición sin guardas disparada por el evento *Entregado*. Esto indica que la transición ocurrirá siempre que el evento ocurra.

Lo último por manejar es una transición denominada *Cancelado*. Se debe poder cancelar una orden en cualquier punto antes de que sea entregada. Se puede hacer esto dibujado transiciones separadas para cada uno de los estados. Una alternativa útil es crear un **súper estado** de los tres estados y entonces

dibujar una sola transición. Los subestados heredan cualquier transición del súper estado.

Las figuras 2.17 y 2.18 muestran ambas formas de reflejar el mismo comportamiento del sistema. Aún con solo tres transiciones duplicadas, la figura 2.17 se ve bastante más desordenada. La figura 2.18 hace todo mucho más claro, y si se requieren cambios posteriores, será difícil olvidar el evento *Cancelado*.

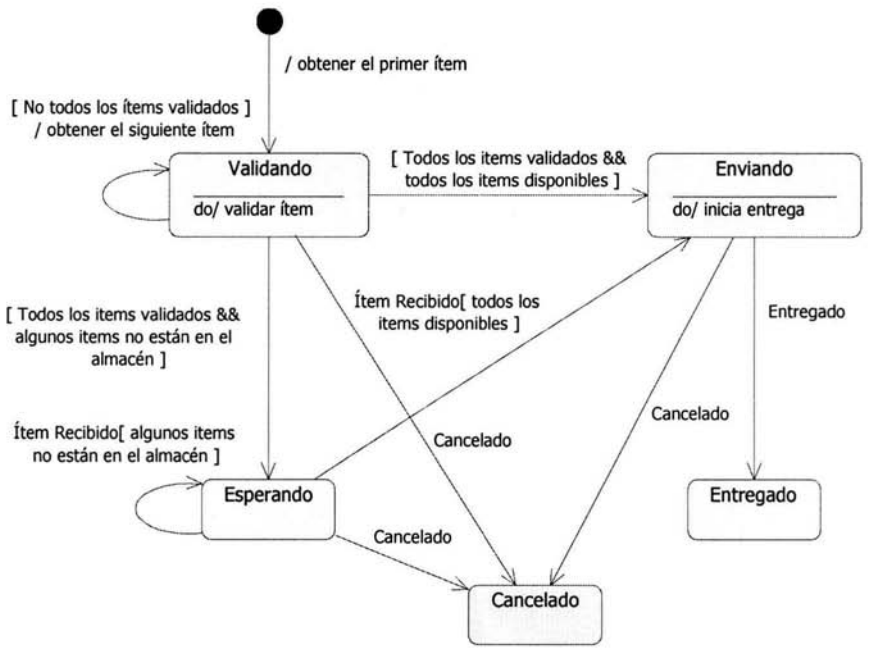


Fig. 2.17 Diagrama de Estado sin Súper Estados.

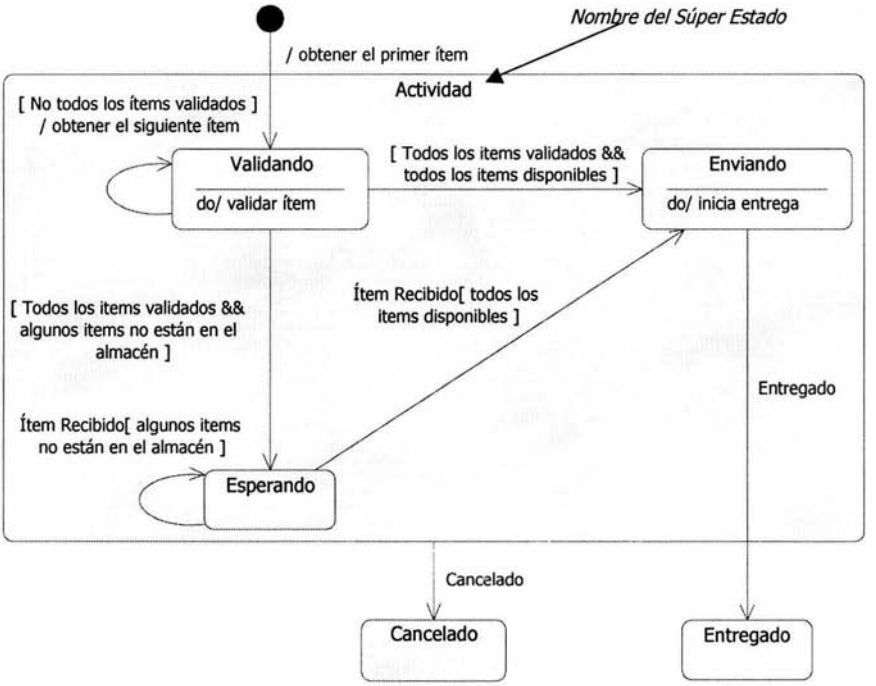


Fig. 2.18 Diagrama de Estado con Súper Estados.

2.6 Diagrama de Despliegue

EL **diagrama de despliegue** es una representación gráfica del hardware, procesadores y procesos dentro del sistema de software. El diagrama consiste de dos iconos principales; nodos y conexiones, cada icono puede hacer uso del estereotipo.

Un **nodo** contiene un recurso material y es representado por un cubo que evoca la presencia física del equipo en el sistema. La naturaleza del equipo puede precisarse por medio de un estereotipo. Cada nodo debe ser nombrado de acuerdo al tipo de hardware que representa.

Los diferentes nodos que aparecen en el diagrama se conectan entre sí por líneas que simbolizan un soporte de comunicación a priori bidireccional, denominado **conexión**. La naturaleza de este soporte puede precisarse por medio de un estereotipo. Donde más de un elemento de un tipo puede involucrarse en una conexión, la **multiplicidad** puede ser mostrada en la conexión.

El diagrama de despliegue es en realidad un diagrama de clases.

- Los nodos son las clases, las conexiones son las asociaciones.
- La herencia, la agregación, y demás pueden ser usados.

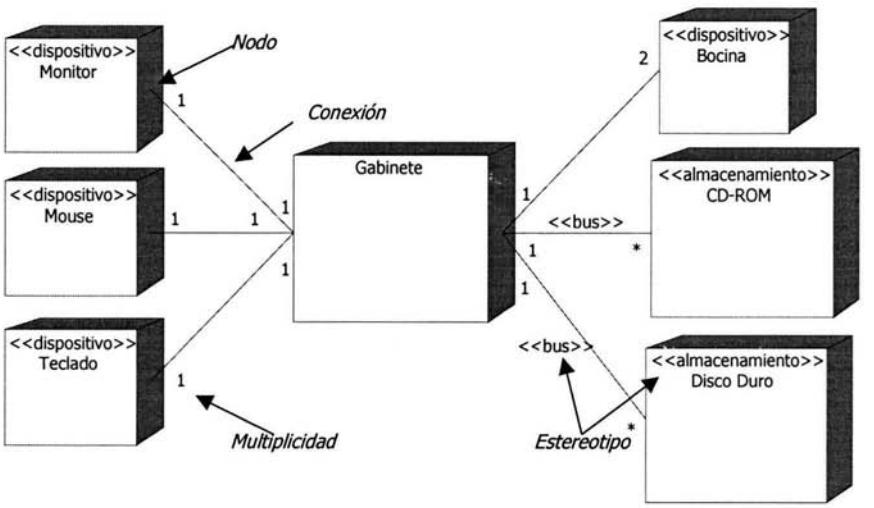


Fig. 2.19 Diagrama de Despliegue.

CAPÍTULO 3

Metodología de Análisis

Este trabajo ha descrito como se ha desarrollado el UML desde sus inicios hasta la última versión adoptada por el OMG. Se ha dicho también que el UML es un lenguaje de modelado, *no un método*. Se ha mencionado lo que es el modelado visual y como es útil para entender los problemas y comunicarse con todas las personas envueltas en un proyecto. Se mencionaron las técnicas de desarrollo en cascada y el desarrollo iterativo e incremental. Finalmente se han visto los diferentes tipos de diagramas de los cuales consta el UML, las características y funcionalidades de cada uno de ellos y la forma de generarlos.

Ahora que se tiene todo este conocimiento conjuntado, ha llegado el momento de ordenarlo y agruparlo, es decir, de presentar formalmente el método que se seguirá en el proceso de desarrollo; para que durante el análisis del sistema se identifiquen el alcance y los requerimientos detallados del mismo, y durante el diseño se modelen las clases principales del sistema, así como su comportamiento, sus asociaciones y su arquitectura.

El procedimiento que ejecutará la Metodología para el Análisis y Diseño de Sistemas Orientado a Objetos con UML es el siguiente:

1. El modelo de casos de uso capturará todos los requisitos funcionales del sistema y definirá el alcance.
2. Este modelo poblará el diagrama de clases y generará muchos escenarios.
3. Los escenarios serán soportados por el diagrama de clases y a su vez, serán la semilla que definirá el comportamiento y la interacción de las clases.
4. Los diagramas de secuencia expresarán de una manera más formal como los grupos de objetos colaborarán en algún comportamiento.
5. La agrupación de las clases y sus dependencias será encapsulada y abstraída en el diagrama de paquetes, creando de esta forma la arquitectura lógica del software.
6. Finalmente el diagrama de despliegue será la vista de la arquitectura física del software.

La figura 3.1 resume las relaciones entre los casos de uso, el diagrama de clases, los escenarios, el diagrama de secuencia, el diagrama de paquetes y el diagrama de despliegue.

Estos pasos deberán ser ejecutados utilizando la técnica del modelo de desarrollo iterativo reflejando con más veracidad las actividades de desarrollo durante un proyecto.

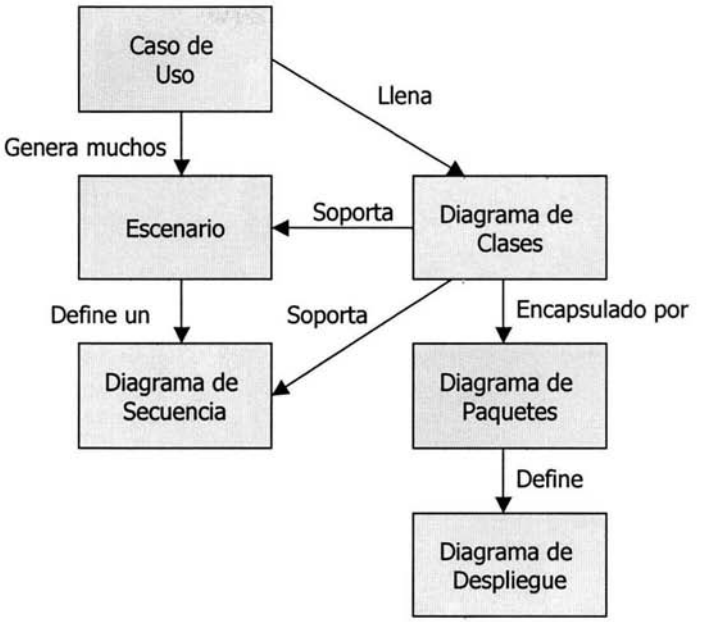


Fig. 3.1 Metodología para el Análisis y Diseño de Sistemas Orientado a Objetos con UML.

Para corroborar la funcionalidad de la metodología, se implementará el sistema denominado "CLC Web", que instituirá la Tesorería de la Federación, Dependencia de la Secretaría de Hacienda y Crédito Público, cuyas características son idóneas para su aplicación, este sistema requiere ser funcional en Internet y será desarrollado en el lenguaje de programación Java. Este lenguaje de programación es orientado a objetos y se acopla de una manera nativa a los browsers (navegadores) necesarios para la interacción con el Web.

3.1 Requerimientos

La Tesorería de la Federación (TESOFE) tiene considerado incluir un nuevo mecanismo de registro de Cuentas por Liquidar Certificadas (CLC) que durante el 2003 se enlazará con el Sistema Integral de Pagos (SIP). El sistema será denominado "CLC Web".

Este registro se realizará desde la ubicación física de las Unidades Administrativas en las Dependencias del Gobierno Federal, afectando los techos presupuéstales en el SIP, la TESOFE, continuará con el proceso de programación de pagos hasta efectuar el depósito en la cuenta de los beneficiarios como actualmente se realiza.

Por otro lado, se considera aprovechar este registro de la CLC para incluir la captura de la Estructura Programática (EP) completa, incluyendo las CLC's que se pagan en efectivo y las de tipo compensada, las cuales se almacenarán en una base de datos independiente para efectos de explotación posterior, sin embargo, para efectos del registro y actualización del SIP, se tomará solo la información necesaria para continuar con el control y seguimiento a nivel Línea Global (LG) y Sublínea Global (SL).

Con la finalidad de brindarle mayores facilidades a las Dependencias se desarrollará una aplicación en Web con una mascarilla de captura que realiza en línea la validación de los datos correspondientes, así mismo la actualización de los techos presupuéstales del SIP.

El objetivo general es desarrollar una aplicación que permita a las Dependencias, la captura en línea de las CLC's a través de una pagina de Internet, instrumentando los medios de seguridad necesarios para su envío y recepción.

La tabla 3.1 muestra los objetivos particulares:

No.	OBJETIVO
1	Recibir y registrar las CLC's a través del portal de Internet de la TESOFE.
2	Contar con un mecanismo de seguridad para que solamente los usuarios autorizados tengan acceso a esta aplicación.
3	Agregar las páginas necesarias para registrar las CLC's.
4	Establecer un mecanismo en línea de retroalimentación a las Dependencias, para comunicar la aceptación o rechazo de las CLC's o bien de las anomalías que se presenten durante el proceso de registro.

Tabla 3.1 Objetivos Particulares.

La autenticación del usuario permitirá su acceso a la aplicación, mediante el uso de una clave de identificación y la contraseña personalizada.

El elemento de control de operación deberá ser el que permita la verificación del rol y la cobertura operativa del usuario, para la operación del documento en el sistema.

El rol se refiere a las actividades que el usuario puede realizar, siendo estos:

- *Capturista*: Su función es llevar a cabo el ingreso de la información correspondiente al documento CLC en efectivo o compensada, esto lo hará a través de las mascarillas de captura diseñadas para cada uno de estos

elementos. En caso de existir un rechazo, ya sea por sistema o por el usuario autorizador, tendrá la facultad de corregir la información de la CLC.

- *Autorizador*: Es el rol a través del cual un usuario puede llevar a cabo la aceptación o rechazo de una CLC y, con ello, la disposición de recursos y la instrucción de programación del pago.

La cobertura operativa se refiere al Ramo y específicamente la Unidad Responsable (UR) que un usuario puede operar y/o afectar. Como regla general los usuarios no podrán operar y/o afectar ramos distintos a los de su adscripción.

Un usuario que cuente con los dos roles, no podrá operar el sistema con ambos de manera simultánea.

Se deberá contar con un módulo que permita la verificación de la información capturada en las mascarillas de los documentos, a fin de validar que dicha información sea congruente. Esta validación entre otras cosas deberá verificar que la información esté contenida en catálogos, la afectación presupuestal esté dentro del techo, se calcule la LG y SL.

La tabla 3.2 indica los componentes del encabezado de la CLC:

ENCABEZADO DE LA CLC	
COMPONENTE	CARACTERÍSTICAS
FOLIO DE CLC	Automático por el sistema, lo asigna al concluir su captura, no modificable. Número consecutivo que asigna el sistema para identificación del documento, único por Ramo y UR.
FOLIO DEPENDENCIA	Obligatorio, capturable, único por Ramo y UR. Número que le asigna la Dependencia a la CLC. Tipo Numérico. Tamaño 7.

ENCABEZADO DE LA CLC (cont.)	
COMPONENTE	CARACTERÍSTICAS
FECHA DE CAPTURA	Automática por el sistema, debe ser la fecha del servidor. Día, mes y año en la que se opera la CLC. Tipo Fecha.
FECHA DE APLICACIÓN	Automático igual a la Fecha de Captura, en caso de que la captura se realice en un día inhábil, el sistema deberá modificar automáticamente la fecha de aplicación al siguiente día hábil, no modificable. Día, mes y año en que se registrará la aplicación. Tipo Fecha.
IMPORTE TOTAL DIVISA	Total del importe de la CLC, en la divisa en la cual se realiza el pago. Automático por el sistema, no modificable, suma del Importe Divisa del detalle de la CLC. Tamaño 20 enteros, 2 decimales.
IMPORTE TOTAL MXN	Automático por el sistema, no modificable, suma del Importe MXN del detalle de la CLC. Tamaño 20 enteros, 2 decimales.
RAMO EJECUTOR	Ramo al que pertenece el usuario que emite la CLC. Automático por el sistema, no modificable, lo toma del usuario que se firma al entrar al sistema.
UNIDAD EJECUTORA	UR a la que pertenece el usuario que emite la CLC. Automático por el sistema, no modificable, lo toma del usuario que se firma al entrar al sistema.
RAMO AFECTADO	Ramo al que se afecta el presupuesto. Automático por el sistema, no modificable, lo toma del usuario que se firma al entrar al sistema.
UNIDAD AFECTADA	UR a la que se afecta el presupuesto. Automático por el sistema, no modificable, lo toma del usuario que se firma al entrar al sistema.

ENCABEZADO DE LA CLC (cont.)	
COMPONENTE	CARACTERÍSTICAS
RAMO RECEPTOR	Ramo de la unidad responsable que recibe la operación. Automático por el sistema con valor 6, no modificable.
UNIDAD RECEPTORA	UR que recibe la operación. Automático por el sistema con valor 600, no modificable.
DIVISA	Divisa sobre la cual se realiza la CLC. El sistema debe poner por omisión MXN (Pesos Mexicanos), presentar ayuda del catálogo de Divisas.
TIPO DE CAMBIO	Capturable, obligatorio en caso de que la Divisa sea diferente a MXN, dato por omisión 1, modificable solo en caso de que la Divisa sea diferente a MXN y solo debe aceptar valores mayores a 0. Longitud 3 enteros, 6 decimales.
CÓDIGO DE ENTIDAD	Clave que identifica al Beneficiario. Al proporcionar la clave de la entidad, se validará que exista en dicho catálogo y desplegará de manera informativa la Cuenta Bancaria para tener la certeza en donde se realizará el abono. En caso de no existir el Código de la Entidad, deberá enviar una leyenda que indique "Código de Entidad invalido" y no permitirá continuar con el envío de la CLC, se considera como dato obligatorio.
CUENTA BANCARIA	Cuenta bancaria del Beneficiario. Este campo se llenará automáticamente conforme al catalogo de Entidades de la TESOFE y estará ligado al campo de Código de Entidad.
RFC	Registro Federal de Causantes del Beneficiario. Este campo se llenara automáticamente conforme al catalogo de Entidades de la TESOFE y estará ligado al campo de Código de Entidad.

ENCABEZADO DE LA CLC (cont.)	
COMPONENTE	CARACTERÍSTICAS
NOMBRE DEL BENEFICIARIO	Nombre del Beneficiario. Obligatorio. Alfanumérico, 80 posiciones.
DESCRIPCIÓN	Leyenda, referencias u observaciones de la forma de pago. CLC en Efectivo: 1. Depósito en cuenta del Beneficiario. 2. Este documento liquidará los importes de la Carta de Crédito Comercial Irrevocable No. _____. 3. Pago a través del Sistema de Compensación de Adeudos. 4. Cheque a favor del Beneficiario. No. De Relación _____. 5. Otros. CLC Compensada: 1. No se paga en efectivo, regulariza el Acuerdo de Ministración de fondos No. _____, con folio SIAFF _____. 2. No se paga en efectivo, regulariza el Fondo Rotatorio o Revolvente No. _____. 3. No se paga en efectivo, será para cubrir las contribuciones por concepto de _____. 4. No se paga en efectivo, regulariza Diferencias Cambiarias a Cargo, originadas de la CLC No. _____. 5. No se pagará en efectivo, se expedirá Certificado Especial. 6. Este documento no se cubrirá en efectivo, únicamente servirá para respaldar pagos en el exterior vinculados al Crédito Externo No. _____. 7. Financiamiento por medio de Línea de Crédito Bilateral.

ENCABEZADO DE LA CLC (cont.)	
COMPONENTE	CARACTERÍSTICAS
AUTORIZACIÓN	Obligatorio para el rol de Autorizador, valores S o N.
OBSERVACIÓN POR RECHAZO	Observaciones opcionales al rechazar el documento.

Tabla 3.2 Encabezado de la CLC.

La tabla 3.3 indica los componentes del detalle de la CLC:

DETALLE DE LA CLC	
COMPONENTE	CARACTERÍSTICAS
NO.	Número de renglón o secuencia del detalle. Automático por el sistema, no modificable, consecutivo inicia en 1. Numérico, 4 dígitos.
EVENTO	Clave de las cuentas contables. Automático por el sistema, no modificable. Alfanumérico, 8 posiciones.
ESTRUCTURA PROGRAMÁTICA	Componentes de la Estructura Programática Presupuestaria a liquidar. Captura obligatoria, debe presentar ayuda para su captura, validar contra catálogo.
CLAVE DE MES	Es el mes sobre el cual se debe realizar la validación presupuestal.
IMPORTE DIVISA	Importe de la divisa que corresponda. Captura obligatoria. Longitud 20 enteros, 2 decimales.
IMPORTE MXN	Importe equivalente en MXN. Automático, calculado al multiplicar el Importe Divisa por el Tipo de Cambio, no modificable.
FECHA PAGO	Fecha propuesta por la Dependencia para realizar el pago. Capturable, obligatorio.

Tabla 3.3 Detalle de la CLC.

Los beneficios que se obtendrán están resumidos en la tabla 3.4.

No.	BENEFICIO
1	Brindar a las Dependencias a través de Internet el registro de sus CLC's.
2	Reducir las visitas de los gestores de las Dependencias a la TESOFE, así como eliminar la recepción de las CLC's en papel.
3	Disminuir el margen de error en el registro de las CLC's, validando los procesos en línea.
4	Eliminación de la captura en la TESOFE.
5	Control del Presupuesto a nivel EP.

Tabla 3.4 Beneficios del Sistema.

3.2 Identificación de Actores

Ahora que se conocen las necesidades del sistema "CLC Web", se identificarán los actores, para lograrlo se responderán las siguientes preguntas:

1. ¿Quién está interesado en algún requerimiento?
   La TESOFE y las Unidades Administrativas en las Dependencias del Gobierno Federal.
2. ¿Quién proveerá, usará y/o removerá esta información en el sistema?
   El Capturista tiene como función principal el ingresar la información correspondiente a la CLC, así como de corregirla en caso de rechazo. El Autorizador será el encargado de aceptar o rechazar la CLC. Finalmente el SIP será actualizado y tomará la información para continuar con el control de seguimiento.

3. ¿Quién usará esta funcionalidad?
   Será utilizada por las Unidades Administrativas en las Dependencias del Gobierno Federal, a través de los usuarios Capturista y Autorizador; y en la TESOFE a través del SIP.
4. ¿Qué otros sistemas necesitaran interactuar con este?
   El SIP será actualizado y tomará la información para continuar con el control de seguimiento.

Las respuestas a las preguntas anteriores destacan como posibles actores a:

1. La TESOFE.
2. Las Unidades Administrativas en las Dependencias del Gobierno Federal.
3. El usuario Capturista.
4. El usuario Autorizador.
5. El SIP.

Se concluye que la TESOFE interactuara con el sistema a través del SIP, y que las Unidades Administrativas en las Dependencias del Gobierno Federal lo harán con los usuarios Capturista y Autorizador. La tabla 3.5 describe a cada uno de los actores.

ACTORES	
NOMBRE	DESCRIPCIÓN
Capturista	Usuario de las Unidades Administrativas en las Dependencias del Gobierno Federal. Ingresa y/o corrige la información correspondiente a la CLC.
Autorizador	Usuario de las Unidades Administrativas en las Dependencias del Gobierno Federal. Acepta y/o rechaza la CLC.

ACTORES (cont.)	
NOMBRE	DESCRIPCIÓN
SIP	Sistema de la TESOFE que es actualizado y toma la información necesaria para continuar con el control de seguimiento.

Tabla 3.5 Descripción de los Actores del sistema "CLC Web".

3.3 Modelo de Casos de Uso

Se identificarán los casos de uso examinando como los usuarios necesitan utilizar el sistema para realizar su trabajo. Cada uno de esos modos de utilizar el sistema será un caso de uso candidato. Para esto, se responderán, de forma similar que con los actores, las siguientes preguntas:

1. ¿Cuáles son las tareas primarias que el actor quiere que el sistema realice?
   Las tareas primarias que el usuario Capturista necesita son las de la captura y corrección de la CLC. El usuario Autorizador requiere de autorizar o rechazar la CLC que el Capturista ha capturado o corregido. Finalmente el SIP solicita ser actualizado para continuar con su proceso.
2. ¿El actor creará, almacenará, cambiará, removerá o leerá datos en el sistema?
   El usuario Capturista creará la CLC, cambiará la información de ser necesario y la almacenará en el sistema. El Autorizador leerá los datos para aceptarlos o rechazarlos. El SIP leerá la información y continuará con el control de seguimiento.
3. ¿El actor necesita informar al sistema acerca de cambios externos?
   No. Los cambios externos son ajenos a los requerimientos del sistema.

4. ¿Necesita el actor ser informado acerca de ciertas ocurrencias en el sistema?

   El SIP podrá ver la actualización de la información desde sus propias consultas.

Ya se tiene identificado cada uno de los casos de uso y los actores que intervienen en ellos. El modelo de casos de uso resultante se muestra en la figura 3.2.

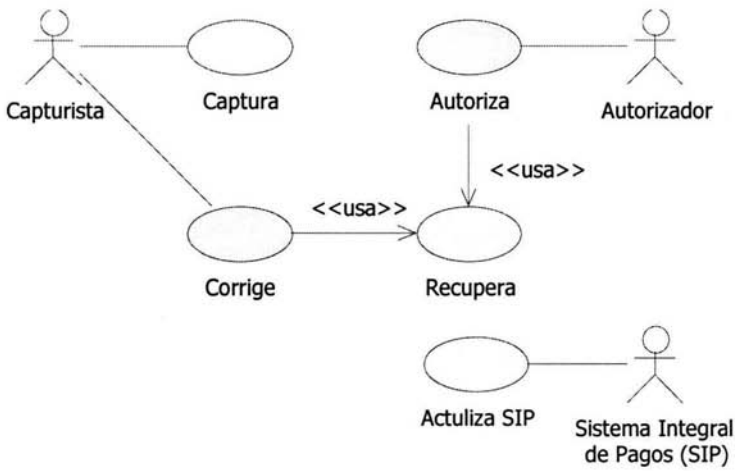


Fig. 3.2 Modelo de Casos de Uso del sistema "CLC Web".

Como ya se ha mencionado un caso de uso puede tener muchas realizaciones; estas realizaciones son los diferentes flujos que sigue un caso de uso. Se describirá cada una de ellas para tener un panorama más específico de su funcionalidad.

Caso de Uso: **Captura**.

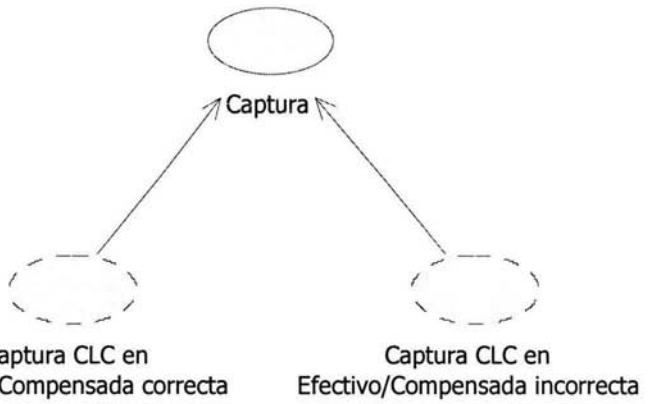


Fig. 3.3 Realizaciones del caso de uso Captura.

Descripción.

- El caso de uso es iniciado por el Capturista. Provee la capacidad al Capturista de ingresar una CLC nueva en efectivo o compensada.

Precondiciones.

- El Capturista deberá estar registrado en el sistema.

Flujo Principal; Captura de CLC en Efectivo/Compensada correcta.

1. El Capturista ingresa su usuario y contraseña.
2. El Capturista selecciona el tipo de CLC (Efectivo/Compensada) y presiona Capturar CLC.
3. El Capturista introduce los datos de la CLC y presiona Enviar CLC (E1).
4. El sistema imprime el Acuse de CLC indicando que la operación se acepto.

Flujo de Excepciones; Captura de CLC en Efectivo/Compensada incorrecta.

- E1: Los datos capturados en la CLC son incorrectos, el sistema muestra la lista de errores y construye una mascarilla con los datos suministrados previamente para su corrección. El Capturista corrige los datos y continúa en el paso 3.

Caso de Uso: **Corrige**.

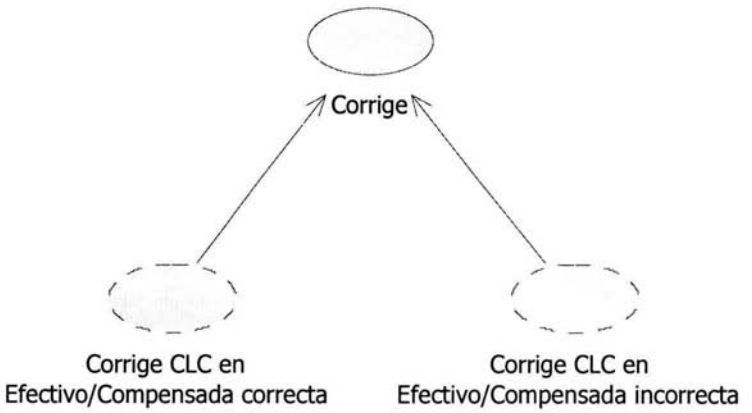


Fig. 3.4 Realizaciones del caso de uso Corrige.

Descripción.

- El caso de uso es iniciado por el Capturista. Suministra la capacidad al Capturista de corregir una CLC en efectivo o compensada, que ha sido rechazada por el Autorizador o por el sistema.

Precondiciones.

- El Capturista deberá estar registrado en el sistema.

Flujo Principal; Corrige CLC en Efectivo/Compensada correcta.

1. El Capturista ingresa su usuario y contraseña.
2. El Capturista introduce el Folio de CLC que desea corregir y presiona Recuperar CLC (E1, E2).
3. El Capturista corrige los datos de la CLC y presiona Enviar CLC (E3).
4. El sistema imprime el Acuse de CLC indicando que la operación se acepto.

Flujo de Excepciones; Corrige CLC en Efectivo/Compensada incorrecta.

- E1: El Folio de la CLC no ha sido digitado. El sistema envía un mensaje de error advirtiendo la causa. El Capturista digita el Folio de la CLC y continua en el paso 2.
- E2: El Folio de la CLC es incorrecto o inexistente. El sistema muestra la pantalla de error, el Capturista regresa al Menú Principal y continúa en el paso 2.
- E3: Los datos capturados en la CLC son incorrectos, el sistema muestra la lista de errores y construye una mascarilla con los datos suministrados previamente para su corrección. El Capturista corrige los datos y continúa en el paso 3.

Caso de Uso: **Autoriza**.

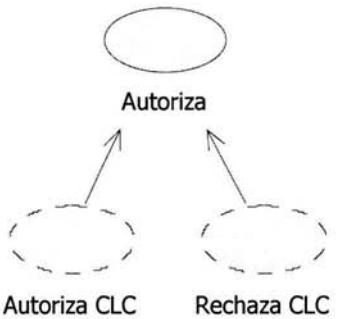


Fig. 3.5 Realizaciones del caso de uso Autoriza.

Descripción.

- El caso de uso es iniciado por el Autorizador. Proporciona la capacidad al Autorizador de aceptar una CLC, que ha sido previamente capturada.

Precondiciones.

- El Autorizador deberá estar registrado en el sistema.

Flujo Principal; Autoriza CLC.

1. El Autorizador ingresa su usuario y contraseña.
2. El Autorizador introduce el Folio de CLC que desea autorizar y presiona Recuperar CLC (E1, E2).
3. El Autorizador acepta los datos de la CLC y presiona Enviar CLC (E3).
4. El sistema imprime el Acuse de CLC indicando que la operación se acepto.

Flujo de Excepciones; Rechaza CLC.

- E1: El Folio de la CLC no ha sido digitado. El sistema envía un mensaje de error advirtiendo la causa. El Autorizador digita el Folio de la CLC y continua en el paso 2.
- E2: El Folio de la CLC es incorrecto o inexistente. El sistema muestra la pantalla de error, el Autorizador regresa al Menú Principal y continúa en el paso 2.
- E3: Los datos en la CLC son incorrectos, el Autorizador rechaza la CLC, el sistema imprime el Acuse de Rechazo y el caso de uso termina.

Caso de Uso: **Recupera**.

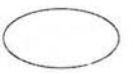


Fig. 3.6 Caso de uso Recupera.

Descripción.

- El caso de uso es utilizado por los casos de uso Corrige y Autoriza. Obtiene la información de la CLC de la base de datos.

Precondiciones.

- Requiere el Folio de CLC para obtener la información.

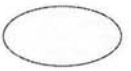
Flujo Principal; Recupera.

1. Obtiene de la base de datos la información de la CLC.

Flujo de Excepciones.

- Ninguno.

Caso de Uso: **Actualiza SIP**.



Actuliza SIP

Fig. 3.7 Caso de uso Actualiza SIP.

Descripción.

- El caso de uso es utilizado por el caso de uso Autoriza. Actualiza la información de la base de datos del SIP.

Precondiciones.

- El Autorizador acepta los datos de la CLC.

Flujo Principal; Actualiza SIP.

1. Actualiza la base de datos del SIP.

Flujo de Excepciones.

- Ninguno.

3.4 Identificación de Clases

Los casos de uso son un punto de inicio ideal para identificar las clases del sistema. Los puntos a seguir para ello son:

- Identificar los sustantivos en los casos de uso.
- Escoger el mejor nombre para los sinónimos.
- Mantener los términos dentro del límite del sistema.
- Los actores pueden ser también clases si se necesita modelar su comportamiento, o retener alguna información acerca de ellos.
- Señalar sustantivos sin importancia como atributos.
- Idear nuevas clases relevantes.

Las clases resultantes se arreglan en grupos lógicos los cuales en la mayoría de los casos son un proceso casi automático cuando se identifican las clases. Algunas de estas clases se volverán atributos de otras clases en lugar de clases de primer orden.

CAPÍTULO 4

Metodología de Diseño

4.1 Diagrama de Clases

Las clases del diagrama se han identificado buscando en los casos de uso y escogiendo los conceptos del dominio. Algunas de estas clases han sido identificadas solo pensando en el detalle de la operación del sistema.

Algunos puntos clave de este diagrama son:

- Es un modelo de diseño por lo tanto contiene clases de diseño que manejan la interfase grafica de usuario, clases contenedoras, accesos a la base de datos y clases de control.
- Este modelo refleja el entendimiento del problema.
- Se encontró que las clases *CLCDrivenCapturista* y *CLCDrivenAutorizador* tienen comportamientos similares y por lo tanto sus propiedades comunes han sido generalizadas en la clase *CLCDriven* de la cual, las otras dos clases heredan.

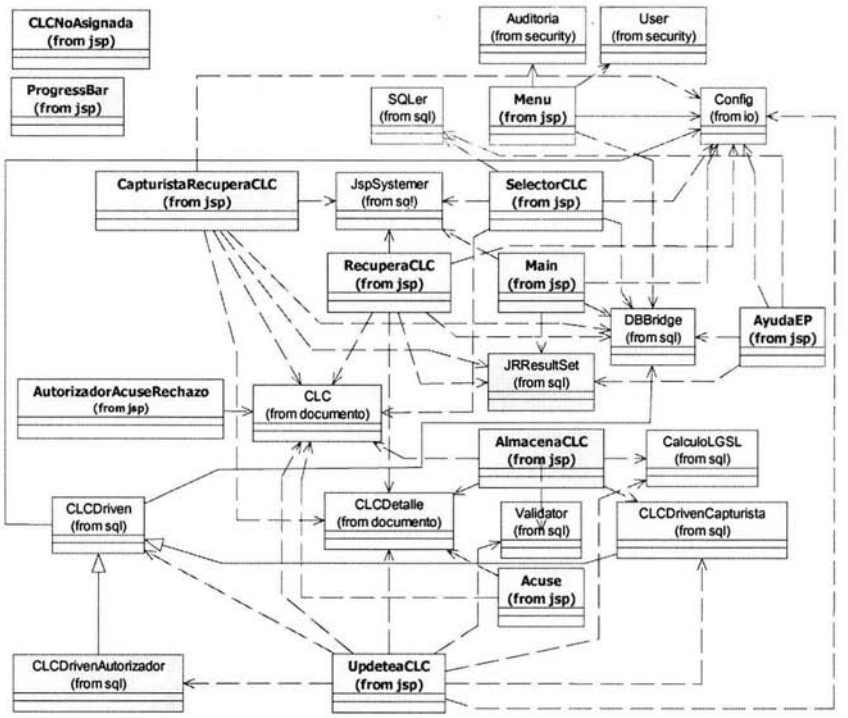


Fig. 4.1 Diagrama de Clases del sistema "CLC Web".

4.2 Escenarios

Los escenarios se construirán basándose en los casos de uso y en el diagrama de clases. Estos proveerán una secuencia específica de las interacciones entre los actores y las entidades del negocio dentro del sistema. También proporcionarán un nivel extra de detalle, identificando las operaciones requeridas por los objetos para que puedan comunicarse los unos con los otros durante el escenario.

Caso de Uso: **Captura**.

- *Captura de CLC en Efectivo correcta.*

El Capturista ingresa su usuario y contraseña, el sistema verifica que sea un usuario valido y basándose en su rol construye la pantalla correcta. Esta pantalla le muestra los tipos de CLC que puede capturar y la opción de recuperar una CLC para su corrección.

El Capturista selecciona el tipo de CLC en Efectivo y presiona Capturar CLC. El sistema confirma si se trata de una captura nueva o de una recuperación de CLC y construye de forma dinámica la mascarilla de captura.

El Capturista llena los campos con la información correspondiente a la CLC en Efectivo y al continuar el proceso el sistema valida contra catálogo los datos y techos financieros. El sistema almacena la información validada en la base de datos e imprime el Acuse de CLC indicando que la operación se acepto informando el Folio de CLC que le asignó al documento.

- *Captura de CLC en Efectivo incorrecta.*

El Capturista ingresa su usuario y contraseña, el sistema verifica que sea un usuario valido y basándose en su rol construye la pantalla correcta. Esta pantalla le muestra los tipos de CLC que puede capturar y la opción de recuperar una CLC para su corrección.

El Capturista selecciona el tipo de CLC en Efectivo y presiona Capturar CLC. El sistema confirma si se trata de una captura nueva o de una recuperación de CLC y construye de forma dinámica la mascarilla de captura.

El Capturista llena los campos con la información correspondiente a la CLC en Efectivo y al continuar el proceso el sistema valida contra catálogo los datos y techos financieros. Los datos capturados en la CLC son incorrectos, el sistema muestra la lista de errores y construye una mascarilla con los datos suministrados previamente para su corrección.

Caso de Uso: **Corrige**.

- *Corrige CLC en Efectivo correcta.*

El Capturista ingresa su usuario y contraseña, el sistema verifica que sea un usuario valido y basándose en su rol construye la pantalla correcta. Esta pantalla le muestra los tipos de CLC que puede capturar y la opción de recuperar una CLC para su corrección.

El Capturista introduce el Folio de CLC que desea corregir y presiona Recuperar CLC. El sistema recupera la información de la CLC y construye de forma dinámica la mascarilla de corrección con los datos correspondientes.

El Capturista corrige los campos de la CLC recuperada y al continuar el proceso el sistema valida contra catálogo los datos y techos financieros. El sistema almacena la información validada en la base de datos e imprime el Acuse de CLC indicando que la operación se acepto.

- *Corrige CLC en Efectivo incorrecta.*

El Capturista ingresa su usuario y contraseña, el sistema verifica que sea un usuario valido y basándose en su rol construye la pantalla correcta. Esta pantalla le muestra los tipos de CLC que puede capturar y la opción de recuperar una CLC para su corrección.

El Capturista introduce el Folio de CLC que desea corregir y presiona Recuperar CLC. El sistema recupera la información de la CLC y construye de forma dinámica la mascarilla de corrección con los datos correspondientes.

El Capturista corrige los campos de la CLC recuperada y al continuar el proceso el sistema valida contra catálogo los datos y techos financieros. Los datos capturados en la CLC son incorrectos, el sistema muestra la lista de errores y construye una mascarilla con los datos suministrados previamente para su corrección.

Caso de Uso: **Autoriza**.

- *Autoriza CLC.*

El Autorizador ingresa su usuario y contraseña, el sistema verifica que sea un usuario valido y basándose en su rol construye la pantalla correcta. Esta pantalla le muestra la opción de recuperar una CLC para su autorización.

El Autorizador introduce el Folio de CLC que desea autorizar y presiona Recuperar CLC. El sistema recupera la información de la CLC y construye de forma dinámica la mascarilla de autorización/rechazo con los datos correspondientes.

El Autorizador acepta la CLC recuperada y al continuar el proceso el sistema valida contra catálogo los datos, los techos financieros y valida si la CLC ha sido autorizada o rechazada. El sistema almacena la información validada en la base de datos e imprime el Acuse de CLC indicando que la operación se acepto.

- *Rechaza CLC.*

El Autorizador ingresa su usuario y contraseña, el sistema verifica que sea un usuario valido y basándose en su rol construye la pantalla correcta. Esta pantalla le muestra la opción de recuperar una CLC para su rechazo.

El Autorizador introduce el Folio de CLC que desea rechazar y presiona Recuperar CLC. El sistema recupera la información de la CLC y construye de forma dinámica la mascarilla de autorización/rechazo con los datos correspondientes.

El Autorizador rechaza la CLC recuperada y al continuar el proceso el sistema valida contra catálogo los datos, los techos financieros y valida si la CLC ha sido autorizada o rechazada. El sistema imprime el Acuse de Rechazo.

Caso de Uso: **Recupera**.

- *Recupera.*

Con el Folio de la CLC selecciona los datos de encabezado y detalle de la CLC y construye la forma correcta basándose en el rol del usuario.

Caso de Uso: **Actualiza SIP**.

- *Actualiza SIP.*

El Autorizador acepta la información de la CLC. El sistema actualiza la base de datos del SIP y los techos financieros son afectados.

4.3 Diagramas de Secuencia

Los diagramas de secuencia muestran las interacciones lógicas entre los objetos relevantes de los diferentes escenarios del sistema.

Existen algunos puntos de interés que denotar con estos diagramas:

- Son diagramas lógicos únicamente. No toman en cuenta decisiones de diseño físico tales como interfases de usuario graficas, accesos a la base de datos, o como el sistema se dividirá en un posible sistema distribuido.
- Los diagramas muestran objetos y no clases. La mayoría de los objetos son tratados como anónimos.

Caso de Uso: **Captura**.

- *Captura de CLC en Efectivo correcta.*

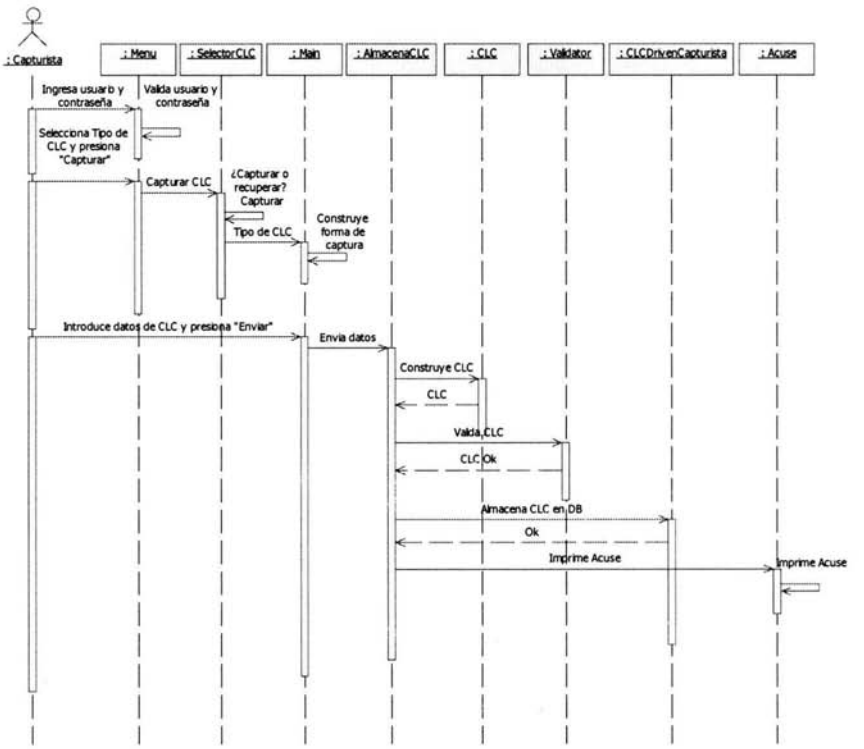


Fig. 4.2 Diagrama de Secuencia del escenario
Captura de CLC en Efectivo correcta.

- *Captura de CLC en Efectivo incorrecta.*

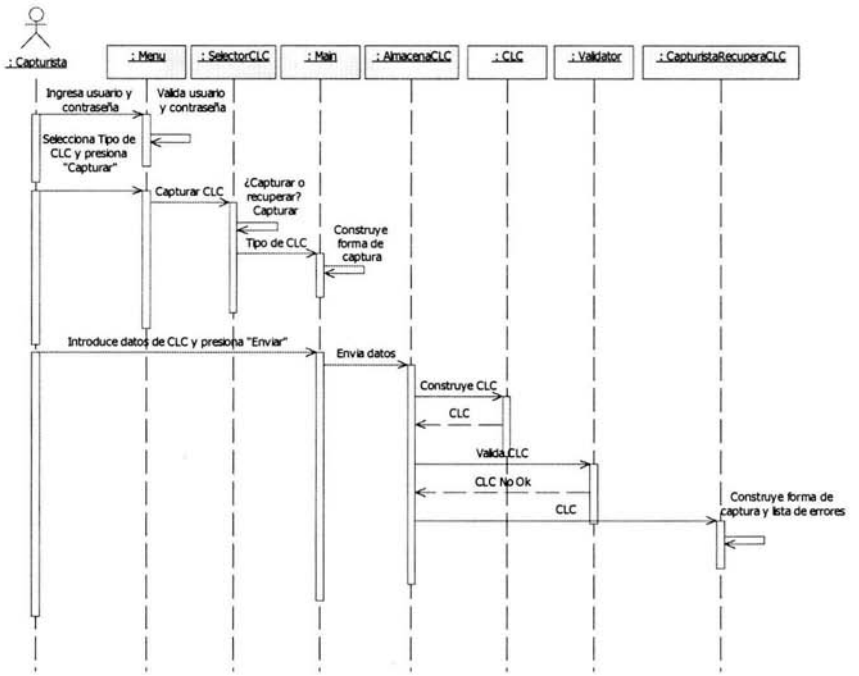


Fig. 4.3 Diagrama de Secuencia del escenario
Captura de CLC en Efectivo incorrecta.

Caso de Uso: **Corrige**.

- *Corrige CLC en Efectivo correcta.*

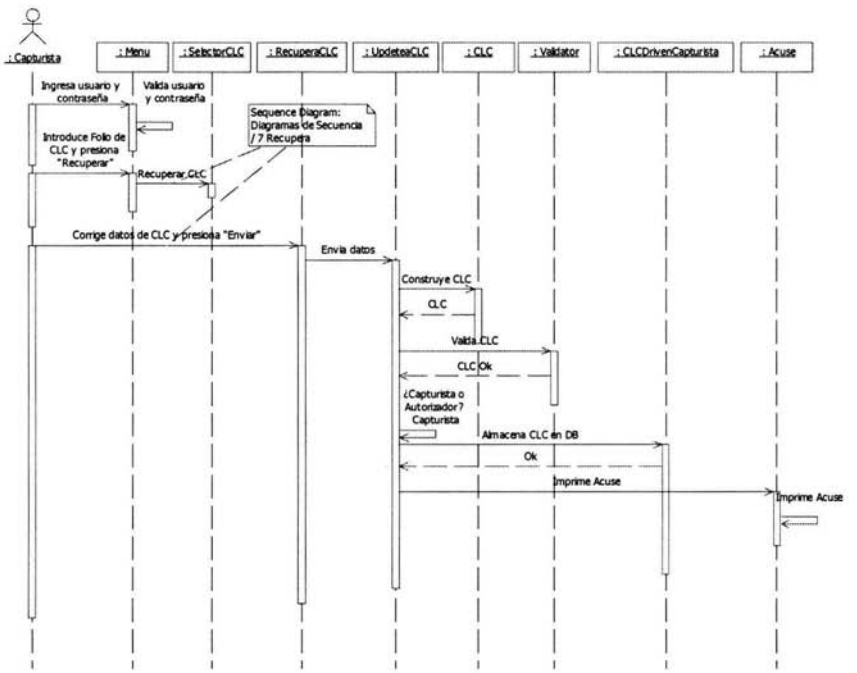


Fig. 4.4 Diagrama de Secuencia del escenario
Corrige CLC en Efectivo correcta.

- *Corrige CLC en Efectivo incorrecta.*

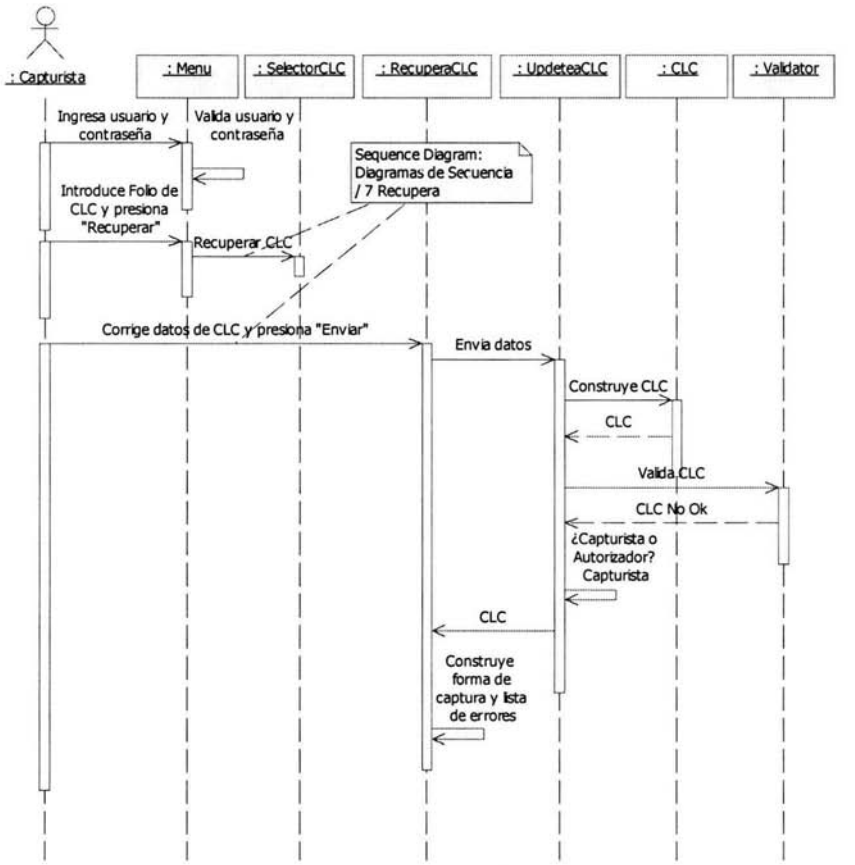


Fig. 4.5 Diagrama de Secuencia del escenario
Corrige CLC en Efectivo incorrecta.

Caso de Uso: **Autoriza.**

- *Autoriza CLC.*

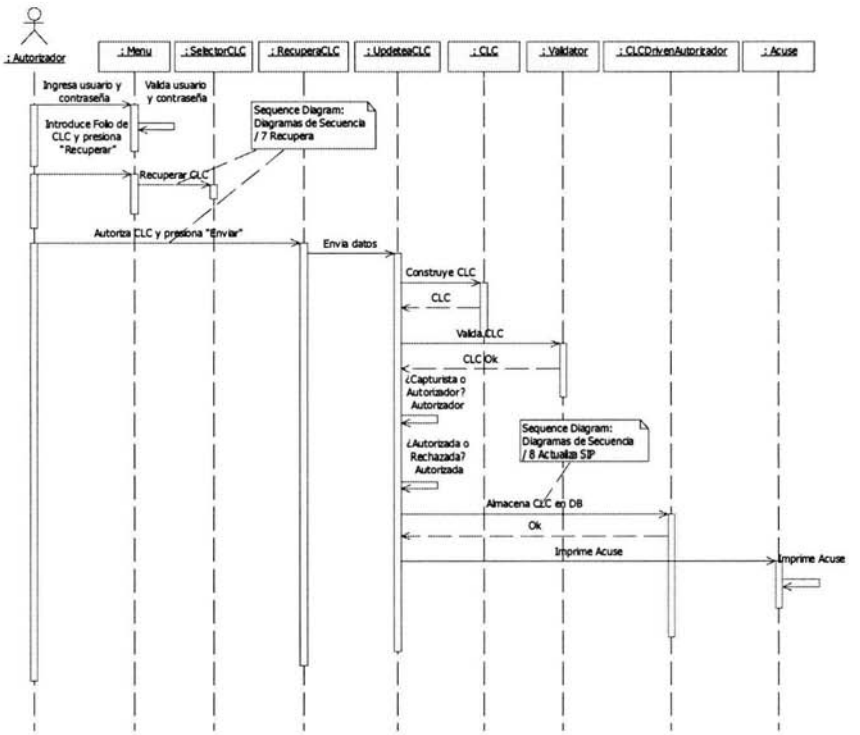


Fig. 4.6 Diagrama de Secuencia del escenario Autoriza CLC.

- *Rechaza CLC.*

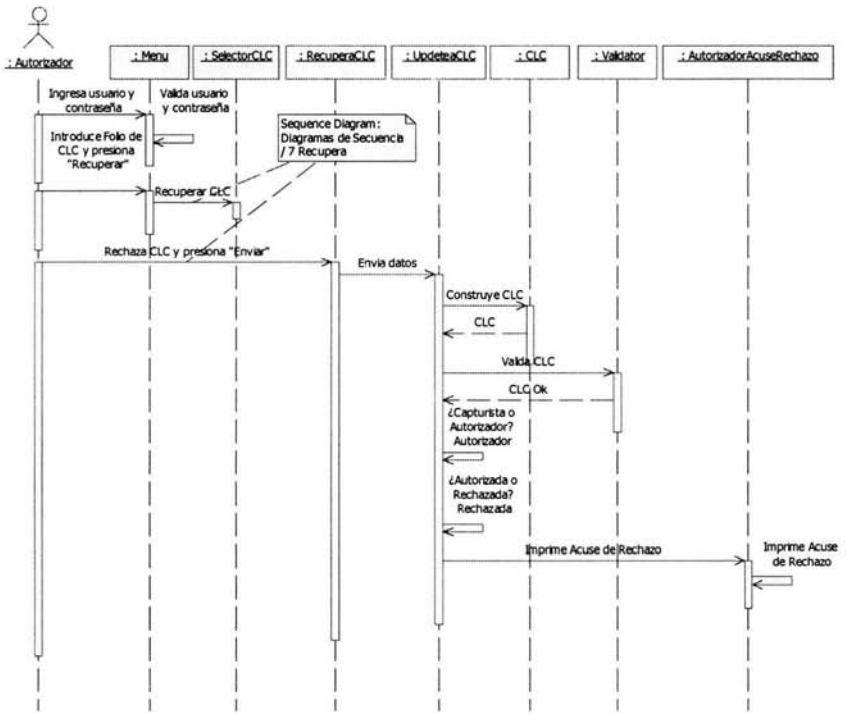


Fig. 4.7 Diagrama de Secuencia del escenario Rechaza CLC.

Caso de Uso: **Recupera**.

- *Recupera.*

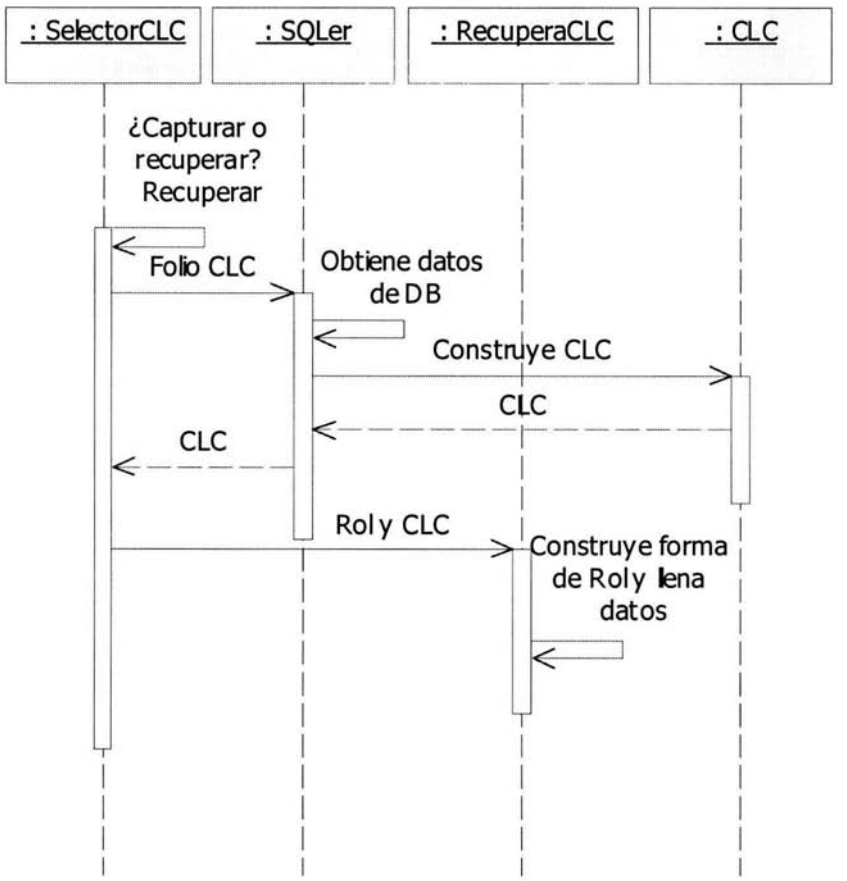


Fig. 4.8 Diagrama de Secuencia del escenario Recupera.

Caso de Uso: **Actualiza SIP**.

- *Actualiza SIP.*

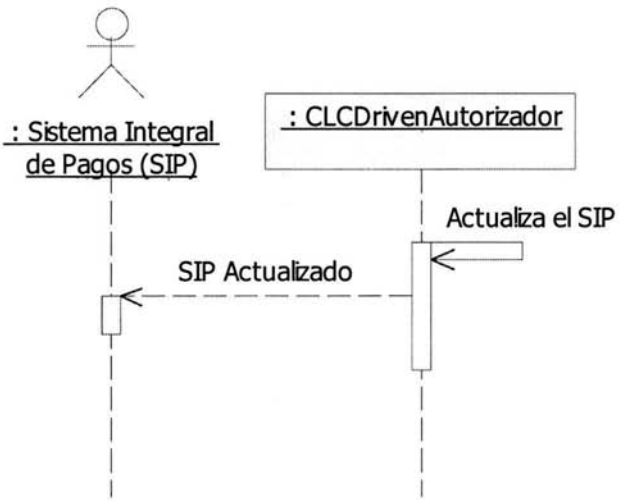


Fig. 4.9 Diagrama de Secuencia del escenario Actualiza SIP.

Los diagramas de secuencia proveen una mejor correlación entre la vista dinámica del sistema, y la vista estática del diagrama de clases.

4.4 Diagrama de Paquetes

Con el diagrama de paquetes del sistema se podrá mostrar la gran imagen pero con menos detalle, las dependencias entre los paquetes/clases, y la complejidad será reducida. El diagrama de paquetes del sistema se muestra en la figura 4.10.

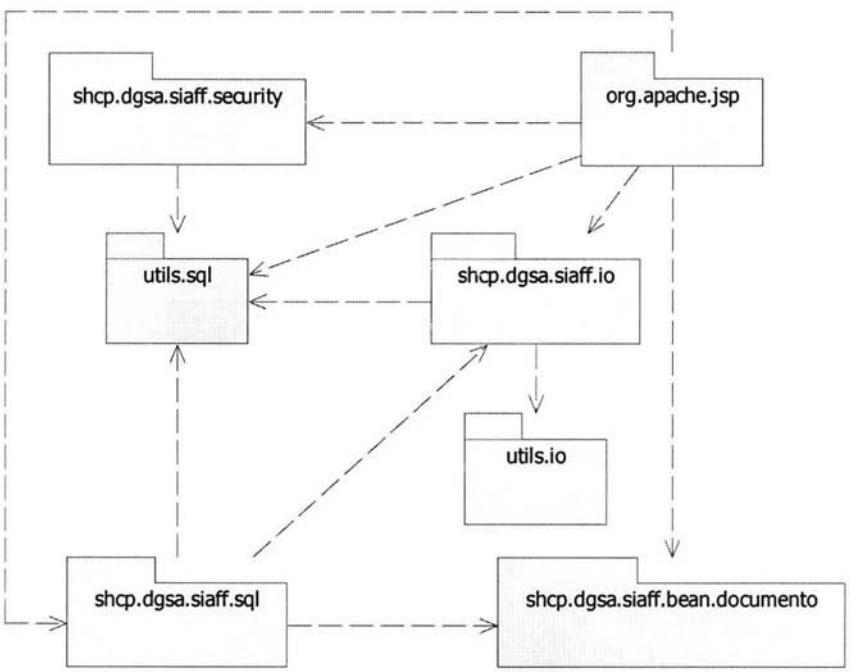


Fig. 4.10 Diagrama de Paquetes del sistema "CLC Web".

4.5 Diagrama de Despliegue

El diagrama de despliegue revelará la distribución del sistema. El sistema se dividirá en tres capas:

1. *Navegador*: Esta será la capa del cliente, desde aquí los usuarios a través del browser se conectaran e interactuaran con la aplicación.
2. *Web Server*: La lógica del negocio se encuentra en esta capa, el procesamiento de validación será realizado aquí.
3. *Base de Datos*: Los datos de la CLC serán almacenados en esta capa.

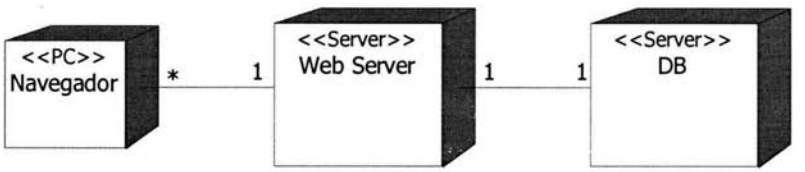


Fig. 4.11 Diagrama de Despliegue del sistema "CLC Web".

Conclusiones

La Licenciatura en Matemáticas Aplicadas y Computación proveyó un conocimiento general y sólido, para realizar el análisis, diseño, desarrollo e implantación de la metodología propuesta en el presente trabajo; a través de sus asignaturas empleando creativa y racionalmente las matemáticas y técnicas computacionales.

Por la versatilidad de la carrera, se puede interactuar fácilmente con profesionistas de otras disciplinas y responder a las necesidades de crear y enfrentarse a posibles cambios tecnológicos en la rama del software, así como, la capacidad de analizar, definir políticas y toma de decisiones sobre el comportamiento de sistemas administrativos, económicos, modelos financieros y de procesos industriales.

La tecnología de objetos se esta volviendo ampliamente aceptada en el progreso de la industria. Con el desarrollo de esta Tesis se tuvo la oportunidad de

practicar el uso del UML para crear modelos de objetos, explicar los principios y la terminología del desarrollo de sistemas orientado a objetos, aplicar las técnicas de orientación a objetos para definir sistemas usando la notación del UML y aplicar un proceso de desarrollo para guiar un proyecto a través del análisis y diseño con éxito.

La Tesorería de la Federación usaba la técnica de desarrollo en cascada y gastaba demasiado tiempo capturando, refinando y documentando los requisitos. Se requería de un desarrollo guiado e iterativo que enfocara las áreas de riesgo críticas temprano en los proyectos.

El problema principal era que el equipo de desarrolladores construía sistemas de software muy complejos y no tenía un proceso escalable para atacar los puntos decisivos.

La Tesorería de la Federación sabía que se necesitaba una manera de asegurar el éxito de los proyectos. Adoptando un proceso iterativo permitiría al equipo resolver el problema básico de como atacar los puntos partiéndolos en pedazos manejables que podrían ser manipulables uno a la vez. La Metodología para el Análisis y Diseño de Sistemas Orientado a Objetos con UML aportó la solución.

La aplicación de la metodología tuvo un efecto inmediato en las operaciones del equipo. Porque proporcionó a sus miembros un vocabulario común y una terminología bien definida, el problema de comunicación entre los individuos y grupos fue rápidamente reducido.

Los analistas del equipo lograron comunicar el alcance y los límites del proyecto más fácilmente con los casos de uso y también consiguieron poner más atención en lo que el sistema necesitaba hacer y menos en como se lograría finalmente.

Usando la metodología, fue posible crear un modelo de negocio que capturó abstracciones importantes y patrones del mundo real, usándose como base para el diseño del sistema.

El modelo abstracto suministró una arquitectura de software robusta con el uso del UML. También introdujo beneficios adicionales incluyendo comunicaciones del equipo mejoradas, ya que los miembros del equipo claramente entendían la estructura de la arquitectura y podían más fácilmente comunicar el diseño a otros.

Otro beneficio importante fue la habilidad de identificar y definir recursos reusables, que significativamente redujeron la duplicación de esfuerzos y proporcionaron al equipo una arquitectura más dinámica.

La Tesorería de la Federación ha iniciado la migración de una técnica de desarrollo en cascada hacia un desarrollo iterativo. Las comunicaciones del equipo mejoraron, se mostró software funcional a los directores, las áreas de riesgo se mitigaron temprano, y la calidad del software mejoró en cada iteración.

La metodología proporcionó a la Tesorería de la Federación una solución a una necesidad inmediata. Pero también proporcionó un camino para adoptar un proceso de desarrollo de software integral.

La Metodología para el Análisis y Diseño de Sistemas Orientado a Objetos con UML brindó a la Tesorería de la Federación con el sistema "CLC Web" los siguientes beneficios importantes:

- Facilitó la comunicación y comprensión común entre los usuarios finales y el equipo de desarrollo, capturando y compartiendo requisitos de alto nivel, procesos del negocio, y modelos de objetos detallados de la aplicación en un formato gráfico.
- Mejoró la comunicación del equipo.
- Reforzó la habilidad de formular requisitos y desarrolló un diseño y arquitectura para satisfacer esos requisitos.
- Dio prioridad a la funcionalidad del sistema, unió requisitos a las iteraciones, y mejoró la calidad del producto a través de los casos de uso y del conocimiento de las necesidades del usuario final.
- Redujo los riesgos del proyecto adoptando un proceso de desarrollo iterativo.
- Con el proceso de desarrollo iterativo, habilitó a los desarrolladores a mantener el modelo del sistema sincronizado con el código fuente.
- Minimizó esfuerzos duplicados y volvió dinámica la arquitectura del sistema identificando y definiendo recursos reusables.
- Exitosamente entregó una aplicación escalable y distribuida, basada en una arquitectura de 3-capas.

Apéndice

Sistema de Cuentas por Liquidar Certificadas en TESOFE "CLC Web"

A.1 Presentación del Sistema "CLC Web"

Fig. A.1 Pantalla de inicio.

La página principal permite a todos los usuarios iniciar en el sistema después de haber ingresado su usuario y contraseña. En caso de que el usuario o contraseña no sean los correctos, el sistema le notificará que no puede ingresar y que debe de validar los datos digitados.

El sistema valida el tipo usuario que esta ingresando, para construir dinámicamente los módulos correspondientes al rol de dicho usuario.

A.2 Rol "Capturista", Captura de CLC en Efectivo

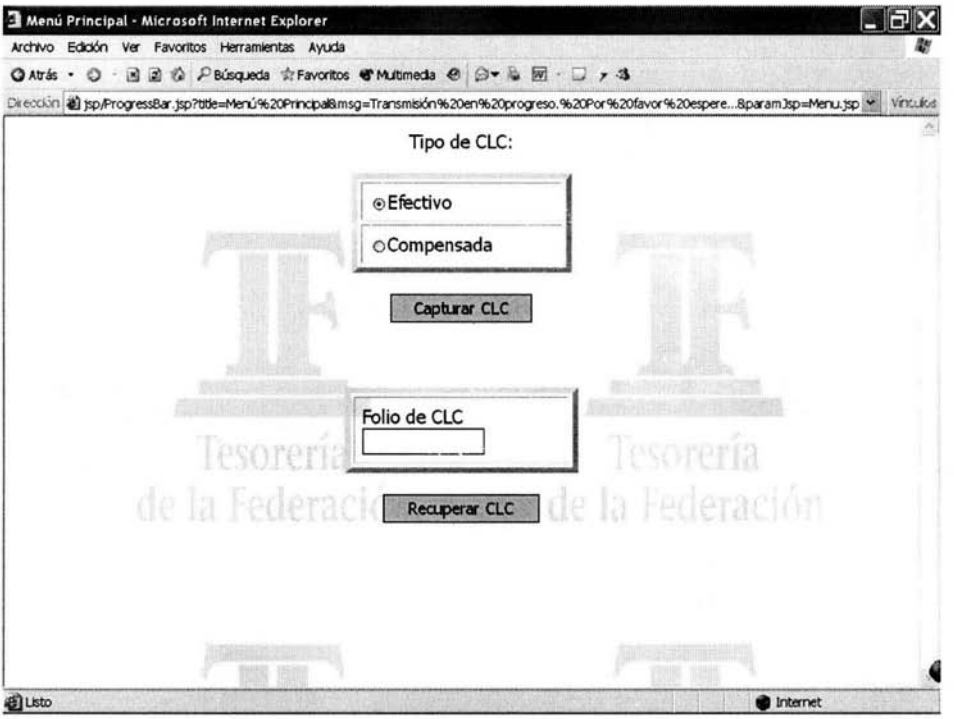


Fig. A.2 Pantalla principal del sistema para el Capturista.

Una vez ingresado al sistema con el rol correspondiente (en este caso con el rol de Capturista), el sistema muestra la página principal.

Se aprecian todas las opciones del menú principal del Capturista del sistema; *Captura de CLC* en *Efectivo* o *Compensada* y *Recuperar CLC* para su corrección.

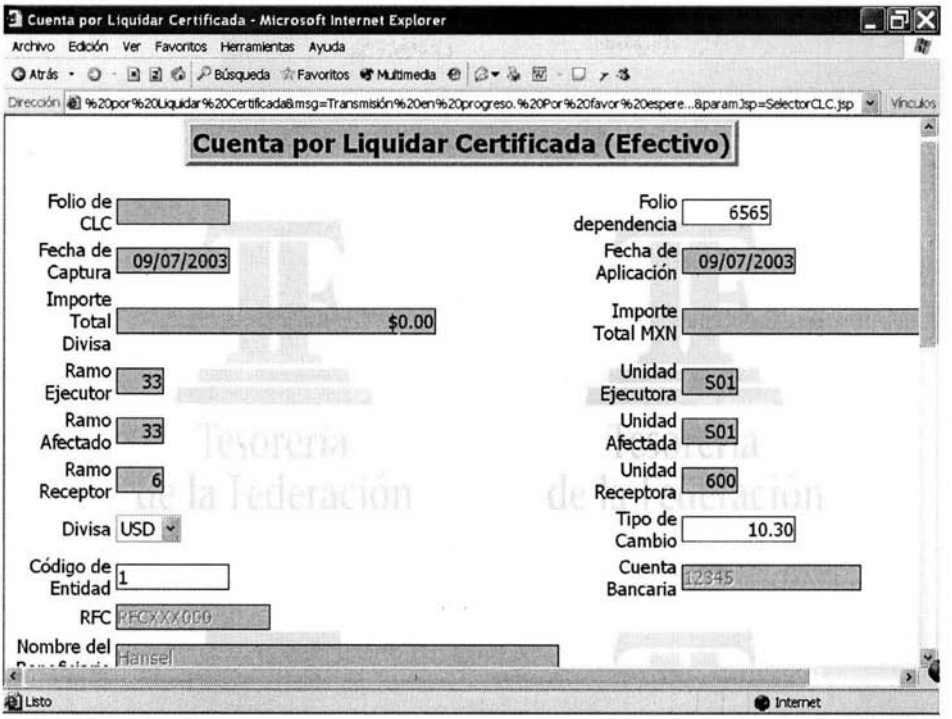


Fig. A.3 Pantalla de captura de CLC en Efectivo (Encabezado).

Los campos que se capturan del encabezado de la CLC en Efectivo son mostrados de color de fondo blanco y son todos obligatorios, los campos de color gris son llenados automáticamente por el sistema y el usuario no los puede modificar.

Uno de los campos más importantes dentro de los datos de encabezado, es el campo *Código de Entidad*, ya que dicho campo determinará el beneficiario y la cuenta bancaria donde se hará el pago.

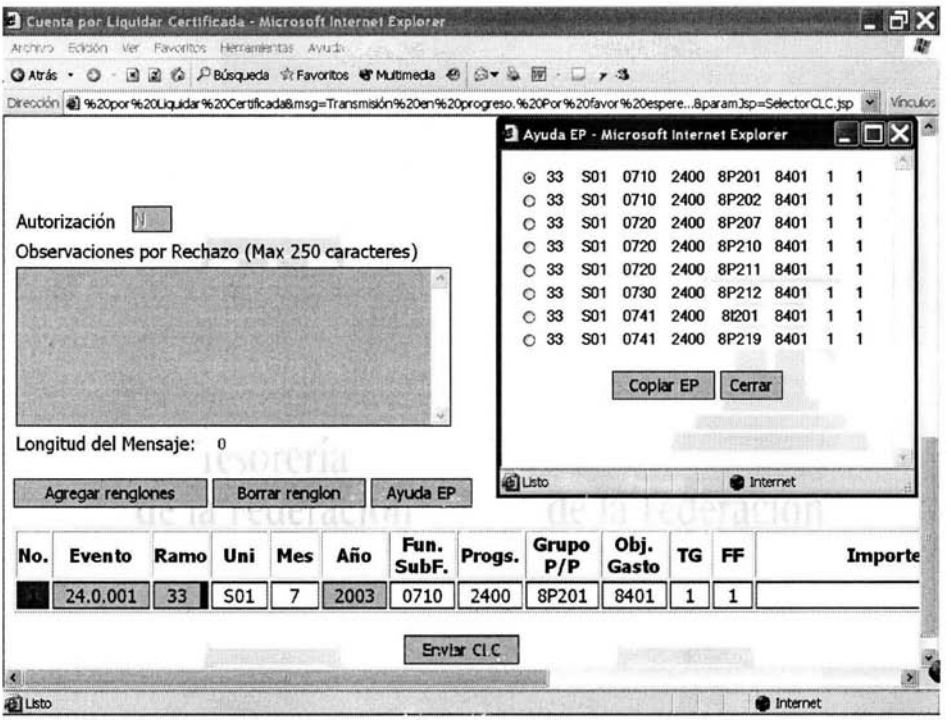


Fig. A.4 Pantalla de captura de CLC en Efectivo (Detalle).

En el detalle de la CLC se captura la Estructura Programática la cual corresponde al concepto por el que se hará el pago.

Al seleccionar un renglón y presionar *Ayuda EP*, aparece una ventana que asiste la captura, evitándole al usuario tener que digitar cada uno de los conceptos de dicha clave.

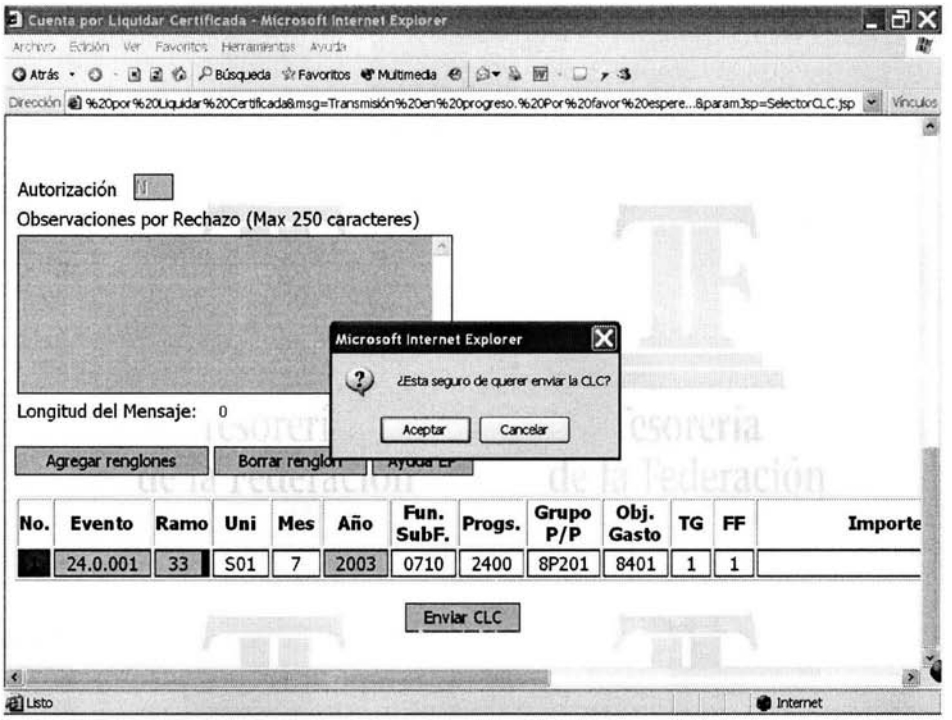


Fig. A.5 Pantalla de captura de CLC en Efectivo (Envío de CLC).

Una vez terminada la captura de los datos del encabezado y del detalle de la CLC, se presiona *Enviar CLC* y el sistema confirma si el usuario esta seguro de querer continuar con el proceso o prefiere detenerse para revisar la información, y de ser necesario cambiarla. Para continuar el envío, el usuario acepta la confirmación.

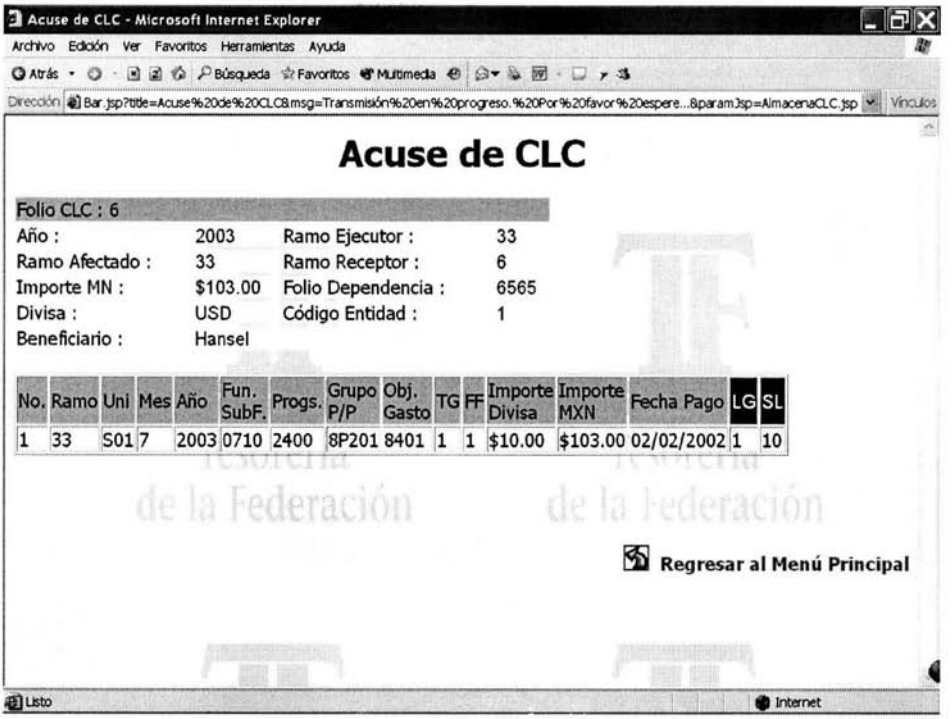


Fig. A.6 Pantalla de Acuse de CLC.

La información se valida contra catalogo y los techos financieros son afectados virtualmente. Si la información es correcta se guarda en la base de datos y el sistema retroalimenta al usuario informándole la aceptación de la CLC.

A.3 Rol "Autorizador", Autoriza CLC

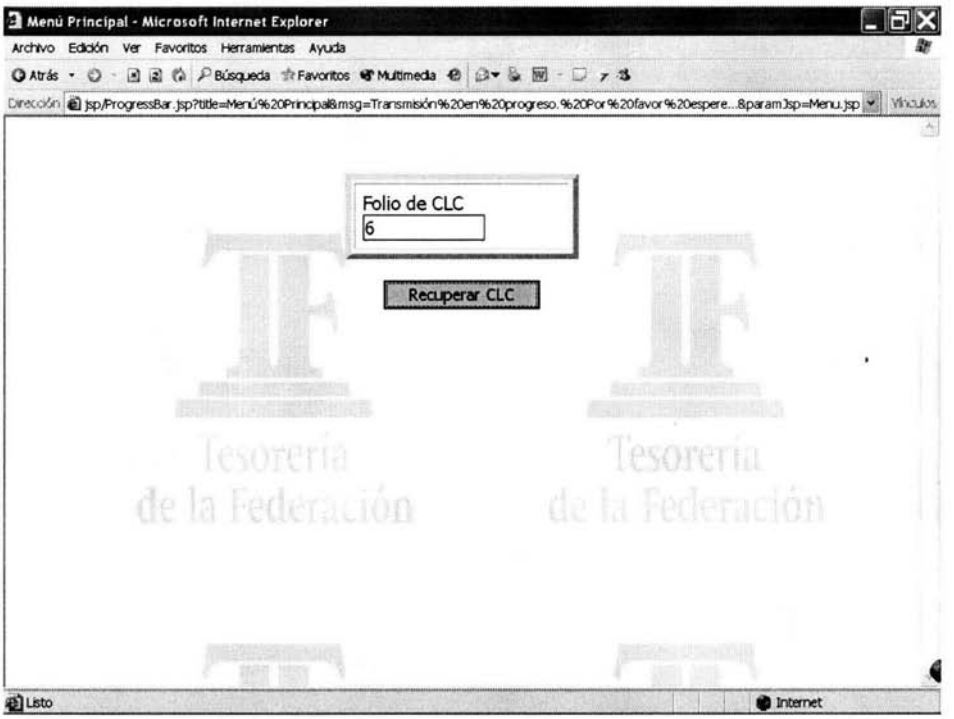


Fig. A.7 Pantalla principal del sistema para el Autorizador.

Una vez ingresado al sistema con el rol correspondiente (en este caso con el rol de Autorizador), el sistema muestra la página principal.

Se aprecia que la única opción del menú principal del Autorizador es *Recuperar CLC* para su autorización.

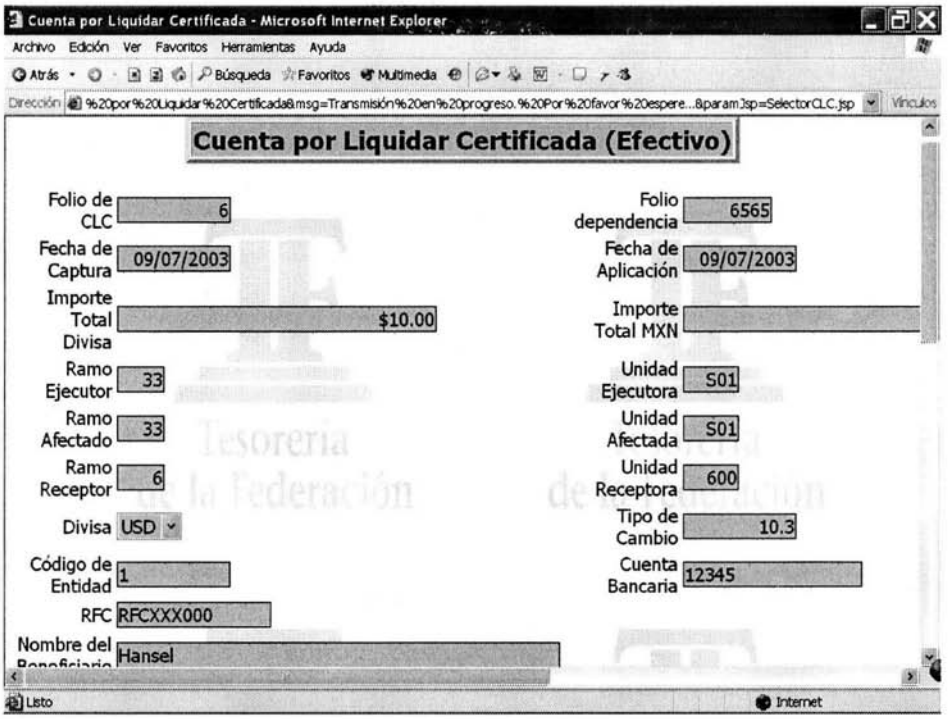


Fig. A.8 Pantalla de autorización de CLC (Encabezado).

El encabezado y el detalle de la CLC (figuras A.8 y A.9) se distinguen de color de fondo gris, esto es, están protegidos por el sistema y no es posible su modificación.

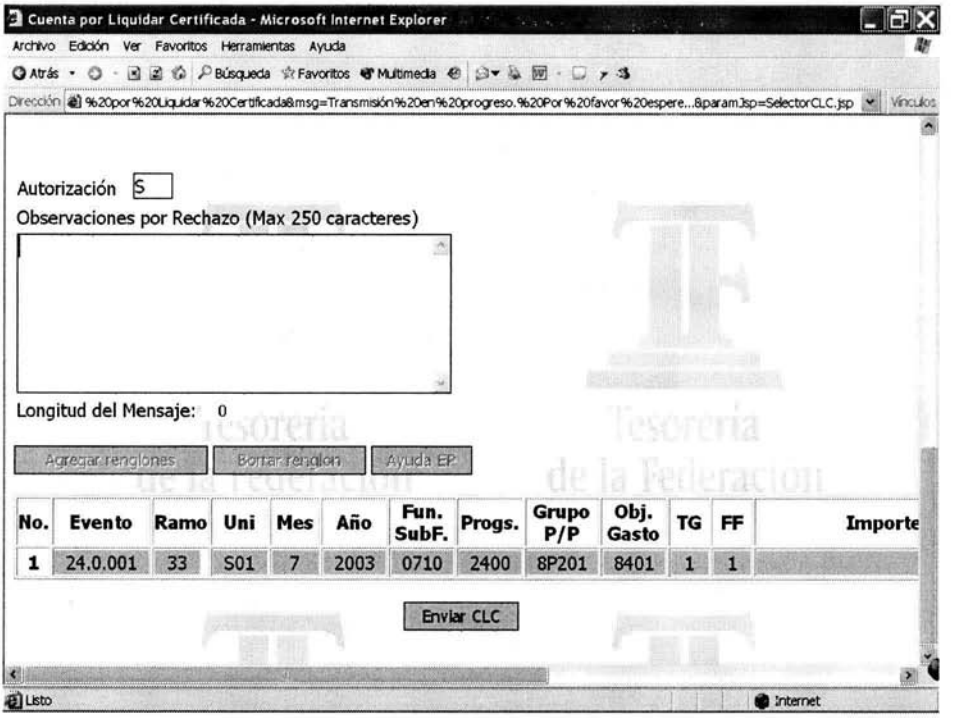


Fig. A.9 Pantalla de autorización de CLC (Detalle).

Los campos habilitados en la pantalla para el Autorizador son *Autorización* y *Observaciones por Rechazo*. Para autorizar la CLC, el usuario digita el valor *S* y se presiona *Enviar CLC*.

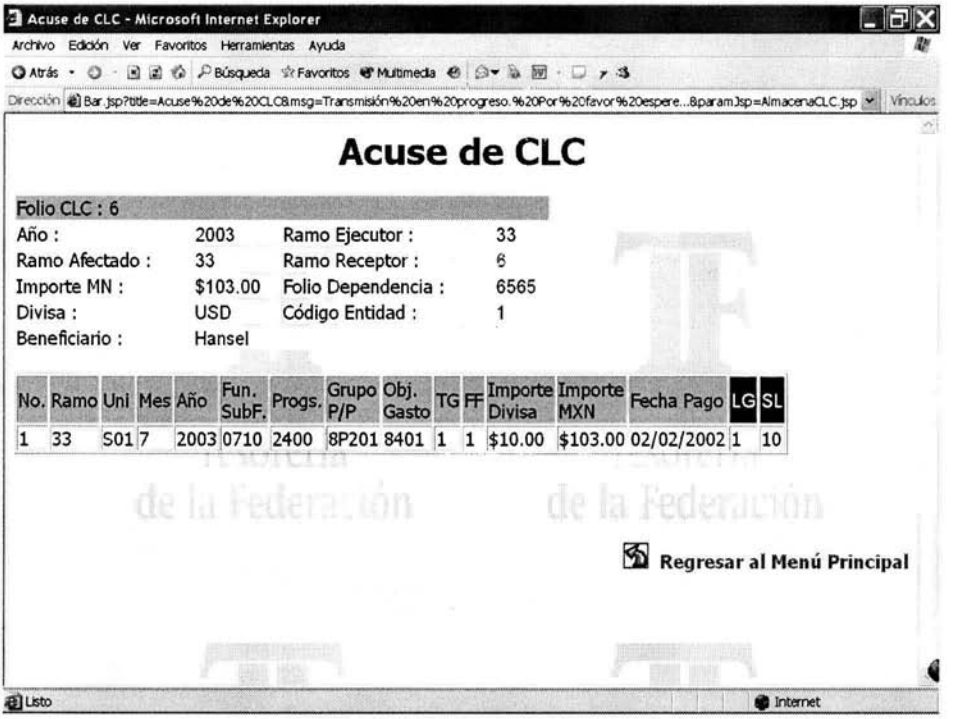


Fig. A.10 Pantalla de Acuse de CLC.

La información se valida contra catalogo y los techos financieros son afectados. Si la información es correcta se guarda en la base de datos y el sistema retroalimenta al usuario informándole la aceptación de la CLC.

Bibliografía

1. "**UML Distilled**"; Martin Fowler, with Kendall Scott, Ed. Addison-Wesley.

2. "**Visual Modeling with Rational Rose and UML**"; Terry Quatrani, Ed. Addison-Wesley.

3. "**El proceso de desarrollo de software**"; Ivar Jacobson, Grady Booch, James Rumbaugh, Ed. Addison-Wesley.

4. "**Modelado de Objetos con UML**"; Pierre-Alain Muller, Ed. Eyrolles.

5. "**UML y Patrones, Introducción al análisis y diseño orientado a objetos**"; Larman, Ed. PHH.

6. "**The Rational Unified Process**"; Philippe Kruchten, Ed. Addison-Wesley.

7. "**Requirements Management with Use Cases**"; Student Manual Version 5.0., Rational University.

8. "**Object-Oriented Analysis and Design with C++**"; Student Manual Version 3.6., Rational University.

9. "**The UML specification documents**"; Booch, G., I. y Rumbaugh, J., 1997. Rational Software Corp.

10. **"Object Solutions"**; Booch, Grady. Redwood City, CA: Addison-Wesley, 1995

11. **"Principles of Software Engineering Management"**; Gilb, Tom. Harlow, UK: Addison-Wesley, 1998.

12. **"A Spiral Model of Software Development and Enhancement"**; Boehm, Barry W., IEEE Computer Mayo de 1998.

13. **"Diccionario Técnico Actualizado"**; Mejía Mesa, Aurelio.

14. **"Diccionario Porrúa de la Lengua Española"**; Ed. Porrúa.

15. **"Diccionario Enciclopédico Ilustrado"**; Ed. Provenza.

16. **"Pequeño Larousse en color"**; Ed. Noguer.

17. **"Diccionario de Sinónimos de la Lengua Castellana"**; Editores Mexicanos Unidos.

REFERENCIAS EN INTERNET.

18. http://www.rational.com/university/rubios.jsp?SMSESSION=NO#booch

19. http://www.rational.com/university/rubios.jsp?SMSESSION=NO#rumbaugh

20. http://www.rational.com/university/rubios.jsp?SMSESSION=NO#jacobson

21. http://c2.com/cgi/wiki?EvolutionaryDelivery

22. http://www.omg.org/uml/

23. http://www.rational.com/uml/resources/documentation/index.jsp?borschtid=1400407079243644031 3&SMSESSION=NO

24. http://boards2.melodysoft.com/app?ID=tematico&msg=660

Glosario

Análisis

Distinción y separación de las partes de un todo hasta llegar a conocer sus principios o elementos. Método que va de lo compuesto a lo sencillo.

Artefacto

Obra mecánica hecha según arte. Artificio, aparato.

Base de datos

Cualquier conjunto de datos organizados para su almacenamiento en la memoria de una computadora, diseñado para facilitar su mantenimiento y acceso de una forma estándar. Los datos suelen aparecer en forma de texto, números o gráficos. Desde su aparición en la década de 1950, se han hecho imprescindibles para las sociedades industriales. Hay cuatro modelos principales de bases de datos: El modelo jerárquico, el modelo en red, el modelo relacional (el más extendido hoy en día; los datos se almacenan en tablas a los que se accede

mediante consultas escritas en SQL) y el modelo de bases de datos deductivas. Otra línea de investigación en este campo son las bases de datos orientadas a objetos, o de objetos persistentes.

Browser

Véase Navegador.

Bucle

Un grupo de declaraciones en un programa ejecutadas repetidamente, ya sea por un número fijo de veces o hasta que una condición específica es verdadera o falsa.

CASE

Computer Aided Software Engineering, véase Ingeniería de Software Asistida por Computadora.

Clase

Una clase es un anteproyecto o prototipo que define las variables y los métodos comunes a todos los objetos de un cierto tipo.

Corroborar

Confirmar, aprobar, ratificar, afirmar, aseverar, probar. Dar nueva fuerza a una opinión, argumento o razón.

Diagrama

Dibujo geométrico que sirve para demostrar una proposición, resolver un problema o figurar de una manera gráfica la ley de variación de un fenómeno. Croquis, esquema, plano, trazado, cuadro sinóptico, mapa.

Diseño

Descripción o bosquejo de alguna cosa. La concepción y planeamiento de las características de un producto, incluyendo la disposición y el desempeño de sus elementos.

Empírico-a

Adj. Perteneciente o relativo al empirismo. Partidario del empirismo. Que exige el concurso de la experiencia.

Empirismo

M. Sistema o procedimiento fundado en mera práctica o rutina. Doctrina que considera la experiencia como la única o más importante fuente del conocimiento.

GUI

Graphic User Interface, véase Interfase Gráfica de Usuario.

Guía

Persona que encamina, conduce y enseña a otra el camino, o el procedimiento para hacer o lograr alguna cosa. Tratado para encaminar o dirigir en cosas materiales o del espíritu. Pieza que obliga a otra a seguir un determinado movimiento. El o lo que sirve de cabeza o principio. Maestro, preceptor.

Herencia

Una clase hereda el estado y conducta de su súper clase. La herencia proporciona un mecanismo poderoso y natural para organizar y estructurar programas de software.

Hipótesis

Lo que tratamos de probar. Es una guía de proposiciones precisa hacia el problema a investigar, para comprobarla o no. Toda explicación tentativa entre variables es una hipótesis, pues no se afirma un hecho hasta que se conozca empíricamente si puede ser verdadera o falsa. La o las hipótesis proponen tentativamente respuestas a los problemas de investigación y a los objetivos que están relacionados con el marco teórico o referencial. Suposición, posible o no, para sacar una consecuencia.

Ingeniería de Software Asistida por Computadora

Software utilizado para el desarrollo automático de sistemas; es decir, código de computadora. Las funciones de las herramientas incluyen análisis, diseño y programación. Las herramientas automatizan los métodos para diseñar, documentar y producir código de computadora estructurado en el lenguaje de programación deseado.

Interfase Gráfica de Usuario

Tipo de visualización que permite al usuario elegir comandos, iniciar programas y ver listas de archivos y otras opciones utilizando las representaciones visuales (íconos) y las listas de elementos del menú. Las selecciones pueden activarse bien a través del teclado o con el ratón.

Internet

Interconexión de redes informáticas que permite a las computadoras conectadas comunicarse directamente. El término suele referirse a una interconexión en particular, de carácter planetario y abierto al público, que conecta redes informáticas de organismos oficiales, educativos y empresariales. También

existen sistemas de redes más pequeños llamados Intranet, generalmente para el uso de una única organización.

Java

El lenguaje de programación Java es un lenguaje de alto nivel que puede ser caracterizado por ser simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutral, portable, de alto desempeño, multihilo, y dinámico.

Mensaje

Es el medio por el cual los objetos de software se comunican e interactúan los unos con los otros.

Método

Modo de decir o hacer con orden una cosa. Modo de obrar o proceder. Libro didáctico de tipo elemental dedicado a la enseñanza de disciplinas que requieren ejercitación. Orden, regla, norma, procedimiento, sistema. Medio o procedimiento para indagar hechos o verdades científicas, o el que se emplea para su exposición.

Metodología

Ciencia del método, parte especial de la lógica.

Navegador

Visualizador y buscador, programa de aplicación utilizado para ubicar y visualizar páginas del WWW en Internet.

Objeto

Un objeto es un paquete de software de variables y métodos relacionados. Los objetos de software son a menudo usados para modelar objetos del mundo real.

OOPSLA, Conference

Conference on Object-Oriented Programming, Systems, Languages, and Applications, véase Programación Orientada a Objetos, Sistemas, Lenguajes y Aplicaciones, Conferencia.

Proceder

Modo, forma y orden de portarse y gobernar uno sus acciones. Comportamiento, conducta.

Procedimiento

Acción de proceder. Método de ejecutar algunas cosas. Sistema.

Proceso

Progreso. Transcurso del tiempo. Conjunto de las fases sucesivas de un fenómeno. Prolongación, parte u órgano que sobresale del resto del organismo.

Programación Orientada a Objetos, Sistemas, Lenguajes y Aplicaciones, Conferencia

La conferencia anual de Programación Orientada a Objetos, Sistemas, Lenguajes y Aplicaciones, es un foro de primera clase en el campo de la tecnología de objetos

Semántico, ca

Ciencia que trata de los cambios de significación de las palabras. Estudio de la significación de las palabras.

Sintáctico, ca

Perteneciente o relativo a la sintaxis.

Sintaxis

Parte de la gramática, que enseña a coordinar y a unir las palabras para formar las oraciones y expresar conceptos. Divídese en regular y figurada. La primera pide que este enlace se haga del modo más lógico y sencillo. La segunda autoriza el uso de las figuras de construcción para dar a la expresión del pensamiento más vigor o elegancia.

Sistema

Conjunto de cosas que ordenadamente relacionadas entre sí contribuyen a determinado objeto. Cualquier conjunto organizado de recursos y procedimientos unidos y regulados por la interacción o interdependencia acompañados de un grupo de funciones específicas.

Técnica

Conjunto de procedimientos y recursos de que se sirve una ciencia o arte. Pericia o habilidad para usarlos. Aplicación de los conocimientos científicos en dirección utilitaria.

Tesis

Conclusión, proposición que se mantiene con razonamientos. Disertación escrita que presenta el aspirante a un título académico, ante un tribunal competente.

Web

Véase WWW.

WWW

World Wide Web (también conocida como Web o WWW) es una colección de archivos, denominados sitios de Web o páginas de Web, que incluyen información en forma de textos, gráficos, sonidos y vídeos, además de vínculos con otros archivos. Los archivos son identificados por un URL (*Uniform Resource Locator*, Localizador Universal de Recursos) que especifica el protocolo de transferencia, la dirección de Internet de la máquina y el nombre del archivo.