

# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA
DIVISIÓN DE INGENIERÍA ELÉCTRICA

## METODOLOGÍA PARA EL DESARROLLO DE APLICACIONES ORIENTADAS A OBJETOS

# T E S I S

QUE PARA OBTENER EL TÍTULO DE:

**INGENIERO EN COMPUTACIÓN**

P R E S E N T A :

RICARDO RUBÉN FRANCO DÍAZ



DIRECTOR DE TESIS:
ING. FRANCISCO LÓPEZ RIVAS

MÉXICO, D.F. 2005

*Este trabajo esta dedicado a Dios por haberme regalado una vida llena de satisfacciones y darme fé para seguir adelante en todo momento, a mis queridos Padre y Madre por ser el impulso y fortaleza para lograr mis sueños a pesar de las desventuras, a mi amada Esposa por ser la inspiración más grande para nunca dejar de luchar y hacerme fuerte en mis debilidades, al Osito por enseñarme que el amor jamás tendrá barreras y a mis hermanos por ser un apoyo incondicional y ser mi conciencia en mis momentos de locura.*

# *AGRADECIMIENTOS*

Agradezco a la Universidad Nacional Autónoma de México por haberme aocogido y brindado una educación de excelencia impactando mi vida en el ámbito profesional y personal. También agradezco a la Facultad de Ingeniería de la UNAM la oportunida de haber integrado sus filas y ayudarme a consolidar como una mejor persona para la sociedad a través de mi profesión.

Al Ing. Francisco López Rivas como el disparador para ayudarme a descubrir el liderazgo con el que contaba, e impulsarlo hacia la responsabilidad social que conlleva, así como a darme el último jalón de orejas para no dejar mi titulación pendiente.

Al Ing. Enrique Larios Canale por fortalecer mi carácter como una persona de bien, responsable con la sociedad y haberme inculcado en todo momento el privilegio de ser universitario.

Al Ing. Gabriel Castillo por ser el promotor de mí espíritu como un desarrollador con una perspectiva integral de cada problema.

Vístanme despacio, que voy de prisa.

*Napoleón Bonaparte*

# METODOLOGÍA PARA EL DESARROLLO DE APLICACIONES ORIENTADAS A OBJETOS

**INTRODUCCIÓN** ..... **13**

**CAPÍTULO I. ORIENTACIÓN A OBJETOS** ..... **15**

*Complejidad* ..... *15*
Naturaleza ..... 15
Atributos ..... 16
Solución a la Complejidad ..... 16
Reglas de Descomposición ..... 16
Regla de Abstracción ..... 17
Regla de Jerarquía ..... 17
Importancia del Diseño ..... 17

*Modelo de Objetos* ..... *18*
Fundamentos ..... 18
Análisis Orientado a Objetos ..... 18
Diseño Orientado a Objetos ..... 18
Programación Orientada a Objetos ..... 18
Elementos ..... 18
Abstracción ..... 19
Encapsulación ..... 20
Modularidad ..... 21
Jerarquía ..... 21
Tipeamiento ..... 22
Concurrencia ..... 23
Persistencia ..... 24

*Objetos* ..... *24*
Naturaleza ..... 24
Estado ..... 25
Funcionamiento ..... 25
Identidad ..... 26
Relaciones entre Objetos ..... 26
Liga ..... 26
Agregación ..... 28

*Clases* ..... *28*
Naturaleza ..... 28
Interface e Implementación ..... 29
Relaciones entre Clases ..... 29
Asociación ..... 30
Herencia ..... 30
Agregación ..... 31

*Clases y Objetos* ..... *31*
Interacción ..... 31
Relación ..... 31
Reglas de Análisis y Diseño ..... 32
Construcción de Calidad ..... 32
Calidad de Abstracción ..... 32
Selección de Métodos ..... 33
Selección de Relaciones ..... 34
Selección de Implementaciones ..... 34

*Clasificación* ..... *34*
Importancia ..... 34

Identificación de Clases y Objetos ....34
Enfoques Modernos y Clásicos....34
Categorización Clásica ....35
Agrupamiento Conceptual ....35
Teoría del Prototipo ....35
Aplicación....35
Análisis Orientado a Objetos....36
Enfoque Clásico ....36
Análisis Funcional ....37
Análisis de Dominio....37
Análisis de Casos....38
Análisis Estructurado....38
Abstracción....39
Mecanismos....40

**CAPÍTULO II. BASE DE DATOS ....43**

*Introducción....43*

*Ventajas y Desventajas ....43*

*Modelos de Datos....44*
Definición ....44
Arquitectura ....45
Tipos de Modelos ....45
Modelos Lógicos Basados en Registros ....46

*Conceptos Básicos del Modelo Relacional....46*
Estructura Básica....47
Terminología....47
Llaves....49
Llaves Primarias y Secundarias....49
Llave Foránea ....50
Propiedades de las Relaciones....50

*Álgebra Relacional....51*
Selección ....52
Proyección ....53
Unión....53
Diferencia....54
Producto Cartesiano....54
Intersección ....54
División ....55
Join ....56

*Formas Normales....57*
Ventajas....57
Dependencia Funcional....57
Dependencia Funcional Total....58
Dependencia Funcional Transitiva....58
Diagrama de Dependencia Funcional....59
Comprensión de Atributos....60
Primera Forma Normal (1FN)....60
Segunda Forma Normal (2FN)....62
Tercera Forma Normal (3FN)....63
Normalizando....65

*Reglas de Codd ....68*

*Diagramas de Entidad/Relación (E/R)....69*

*Diseño de una Base de Datos* ..... *72*

*Problema entre RDBMS y Orientación a Objetos* ..... *74*

**CAPÍTULO III. LENGUAJE DE MODELADO UNIFICADO (UML)** ..... **75**

*Introducción* ..... *75*

*Mecanismos de Extensibilidad* ..... *77*
Notas ..... 77
Estereotipos ..... 78
Etiquetas ..... 78
Restricciones ..... 79

*Paquetes* ..... *79*

*Clases y Objetos* ..... *80*
Sección General ..... 80
Sección de Atributos ..... 80
Sección de Métodos ..... 81
Clases de Interface, Control y Entidad ..... 82
Objeto ..... 85

*Relaciones* ..... *85*
Asociación ..... 85
Generalización ..... 86
Agregación ..... 87
Dependencia ..... 88
Realización ..... 88
Clase de Asociación ..... 89

*Diagramas de Casos de Uso* ..... *89*

*Diagrama de Secuencia* ..... *95*

*Diagrama de Colaboración* ..... *97*

*Diagrama de Actividades* ..... *99*
Actividades ..... 99
Transiciones ..... 100
Bifurcaciones ..... 101
Concurrencias ..... 103
Carriles ..... 103

*Diagrama de Estados* ..... *104*

*Diagrama de Componentes* ..... *107*

*Diagramas de Distribución* ..... *108*

**CAPÍTULO IV. METODOLOGÍA DE DESARROLLO BASADA EN UML** ..... **111**

*Modelo del Negocio* ..... *112*
Casos de Uso del Negocio ..... 113
Diccionario del Negocio ..... 113
Diagramación de Casos de Uso ..... 113
Especificación de Casos de Uso ..... 113
Diagramas de Roles ..... 114
Diagramas de Procesos ..... 114
Diagrama de Escenarios ..... 115

*Modelo del Dominio* ..... *115*

*Modelo de Requerimientos* ..... *115*

Especificación de Requerimiento Global y Detallados ..... 117
Diagramas de Casos de Uso ..... 118
Casos de Uso Funcionales ..... 118
Diagramación de Casos de Uso ..... 118
Clasificación de Casos de Uso ..... 118
Especificación de Relación con Actores ..... 118
Clasificación de Actores por Paquete ..... 119
Casos de Uso No Funcionales ..... 119

*Análisis* ..... *119*
Diagrama de Generalización de Casos de Uso ..... 119
Diagrama de Extensión de Casos de Uso ..... 120
Diagrama de Inclusión de Casos de Uso ..... 120
Especificación de Casos de Uso ..... 120
Entradas y Salidas ..... 121
Especificación de Interfaces ..... 121
Descripción de Caso de Uso ..... 121
Prototipo ..... 122

*Diseño* ..... *122*
Identificación de Clases y Relaciones ..... 122
Clases de Entidad ..... 122
Normalización de Base de Datos ..... 123
Clases de Interface ..... 123
Clases de Control ..... 123
Diagrama de Estado, Secuencia, Colaboración y Actividades ..... 123
Identificación de Métodos y Atributos ..... 123
Diagrama de Componentes ..... 123

*Organización del Proyecto (Plan de Trabajo)* ..... *124*

**CAPÍTULO V. ANÁLISIS Y DISEÑO DEL SISTEMA DE HISTORIA CLÍNICA ÓPTICA ..... 127**

*Modelo del Negocio* ..... *127*
Casos de Uso del Negocio ..... 127
Diccionario del Negocio ..... 127
Diagramación de Casos de Uso ..... 129
Especificación de Casos de Uso ..... 130
Diagramas de Roles ..... 133
Diagramas de Procesos ..... 135
Diagrama de Escenarios ..... 137

*Modelo del Dominio* ..... *138*

*Modelo de Requerimientos* ..... *138*
Especificación de Requerimiento Global y Detallados ..... 138
Diagramas de Casos de Uso ..... 140
Casos de Uso Funcionales ..... 140
Diagramación de Casos de Uso ..... 140
Clasificación de Casos de Uso ..... 141
Especificación de Relación con Actores ..... 141
Clasificación de Actores por Paquete ..... 142
Casos de Uso No Funcionales ..... 142

*Análisis* ..... *144*
Diagrama de Generalización de Casos de Uso ..... 144
Diagrama de Extensión de Casos de Uso ..... 145
Diagrama de Inclusión de Casos de Uso ..... 146
Especificación de Casos de Uso ..... 147

Prototipo ..... 159

*Diseño* ..... *159*

Identificación de Clases y Relaciones ..... 160

Clases de Entidad ..... 160

Normalización de Base de Datos ..... 160

Identificación de Sustantivos ..... 160

Identificación de Verbos ..... 160

Identificación de Atributos ..... 160

Análisis de Atributos ..... 161

Selección de Llaves Primarias Iniciales ..... 161

Generar Atributos Clave ..... 162

Diagrama E/R Inicial ..... 162

Dependencias Funcionales con Selección de Llaves Primarias ..... 163

Graficar Dependencias Funcionales ..... 164

Normalización ..... 165

Diagrama E/R ..... 166

Diagrama de Clases de Entidad ..... 167

Clases de Interface ..... 168

Clases de Control ..... 170

Diagramas de Estados ..... 173

Diagramas de Secuencias ..... 174

Diagramas de Colaboración ..... 178

Identificación de Métodos y Atributos ..... 181

Diagrama de Componentes ..... 181

**CAPÍTULO VI. CONSTRUCCIÓN DEL SISTEMA DE HISTORIA CLÍNICA ÓPTICA ..... 183**

*GrlProcesoDCOM.pas* ..... *183*

*GrlCatalogoDCOM.pas* ..... *189*

*SegUsuario.pas* ..... *194*

*SQLMngBD.pas* ..... *198*

*SQLEntSeg.pas* ..... *203*

**CONCLUSIONES ..... 205**

**BIBLIOGRAFÍA ..... 207**

# INTRODUCCIÓN

Desde el principio de la historia el ser humano ha basado su aprendizaje en mecanismos exitosos para repetir procesos y posteriormente afinarlos hasta obtener mejoras al proceso. A este tipo de aprendizaje basado en aciertos y errores lo denominamos experiencia, la cual puede ser adquirida en forma personal u obtenida con ayuda de la experiencia de otros, y en la medida que la experiencia trasciende se convierte en conocimiento, a tal grado que esta sea la forma principal de educación en las escuelas, la cual toma conocimientos de la experiencia profesional de profesores y escritos para transmitirla a los alumnos.

La principal ventaja de transmitir el conocimiento de esta forma, es el evitar descubrir repetidamente las mismas soluciones, es decir, no descubrir el hilo negro, sobre todo en situaciones donde se esta enfocado a resultados en corto tiempo, adicionalmente se reduce en gran medida el riesgo de la dependencia respecto a la capacidad y habilidad de los individuos para afrontar situaciones, especialmente si son desconocidas, lo que nos permite contar con soluciones ya exploradas y experimentadas previamente.

Dentro del área de la computación en el desarrollo de sistemas informáticos a nivel nacional, se sigue trabajando artesanalmente de forma general e incluso habitual, basado en una cultura mediocre de trabajar a “maquinazos”, es decir, la creencia de la productividad enfocada al tiempo dedicado a escribir código, y no perder el tiempo en documentación inútil, llevando a lo grandes fracasos de problemas mundiales del desarrollo del software, por no haber dedicado el suficiente tiempo a comprender la problemática y violando conceptos del desarrollo de mejores prácticas : dejar todo por escrito y en la medida de lo posible en forma gráfica.

Este escrito ofrece una metodología de desarrollo de aplicaciones de software basado en la orientación a objetos, para lograr ese resultado, es imprescindible mencionar la base teórica del análisis y diseño orientado a objetos, la teoría de los manejadores de bases de datos relacionales, el diseño de base de datos relacionales, todo esto acompañado de un lenguaje de modelado.

Aunque todo lo anterior nos da los elementos suficientes para realizar un sistema informático, se demuestra el potencial a través de la implementación de un sistema de historia clínica de óptica. Dentro de la mucha información que se puede encontrar en internet referente al tema, en muy pocas ocasiones se muestra el desarrollo de una aplicación, y cuando se logra encontrar se utiliza en aplicaciones con características muy particulares de problemas poco frecuentes.

# CAPÍTULO I. ORIENTACIÓN A OBJETOS

## Complejidad

### *Naturaleza*

El software modela segmentos del mundo real para tratar de replicar su funcionamiento en sistemas discretos, por ende cualquier modelo mantiene una complejidad al tratar de imitar la realidad con herramientas electrónicas. La naturaleza del software desprende su complejidad en cuatro elementos: complejidad del dominio del problema, la dificultad del manejo de procesos de desarrollo, la posible flexibilidad del software y los problemas de representación del funcionamiento en sistemas discretos.

Complejidad del dominio del problema: El requerimiento de software para satisfacer un problema particular requiere del conocimiento y experiencia del dominio del negocio, en donde aún así las personas con estas características pocas veces dominan la totalidad del problema. Por eso el desarrollador debe de escuchar y cuestionar para lograr obtener la experiencia del usuario referente al dominio del problema en unos cuantos meses, semanas o días. Se incrementa la complejidad si el problema a resolver mantiene muchas condicionantes para tomar decisiones. Adicionalmente es necesario luchar contra elementos no funcionales como reutilización de código, esquemas de seguridad, performance, costo, interactividad y limitaciones físicas de hardware y software.

Dificultad del manejo de procesos de desarrollo: Es común la evolución del software, en donde muchas veces es necesario reescribir código, por eso es casi indispensable construir pequeños bloques de código para su reutilización y limitación de responsabilidades, pero generalmente en un desarrollo participa un conjunto de programadores los cuales después de escribir su código deben hacerlo interactuar con otros códigos, durante esta integración es donde saltan las deficiencias de las etapas de análisis, diseño y desarrollo, sin tomar en cuenta siquiera la complejidad de coordinar en tiempos y objetivos los equipos de trabajo, los cuales generalmente tienen poco trabajo conjunto y/o problemas derivados de la distancia causantes de poca interacción y comunicación.

Posible flexibilidad a través del software: Durante la solución a un problema particular se debe de pensar en aspectos capaces de ofrecer solución a la evolución de las necesidades sin requerir afectar código o al menos tratar de impactarlo en menor medida con unidades de funcionamiento específico. Todo esto obliga a conocer y apostar a la flexibilidad de un sistema para funcionar en un futuro, claro que a mayor flexibilidad el nivel de complejidad se incrementa exponencialmente.

Problemas de representación del funcionamiento en sistemas discretos: Dentro de aplicaciones grandes existen miles de variables, así como también más de un hilo de control. La colección de estas variables, sus valores actuales, su dirección en memoria, constituyen el presente estado de una aplicación. Al

ejecutar nuestro software en sistemas digitales nosotros tenemos estados discretos, así se conforman un número finito de estados. Si diseñamos un sistema con separación de tareas, el funcionamiento de una parte del sistema deberá repercutir en menor grado para el funcionamiento de otra, pero aún así pueden existir errores de transición entre estados, lo cual nunca ocurriría en sistemas continuos. Otro factor importante de complejidad es la descripción de un funcionamiento a través de instrucciones lógicas y matemáticas tratando de representar segmentos del mundo real.

## *Atributos*

Existen al menos cinco atributos comunes a los sistemas discretos:

1. La complejidad habitualmente se muestra de forma jerárquica, es decir, un sistema complejo se compone de subsistemas relacionados, existiendo dentro de un subsistema más subsistemas y así sucesivamente, hasta llegar al nivel más bajo de componentes elementales menos complejos.

2. La decisión de elementabilidad de un componente es arbitraria, por depender del observador del sistema.

3. Las relaciones dentro de un componente son más fuertes que las relaciones entre componentes.

4. Los sistemas jerárquicos se componen normalmente de pocos tipos de subsistemas simples debido a la semejanza de patrones entre los subsistemas compuestos.

5. Un sistema complejo necesita envolver a sistemas simples.

## *Solución a la Complejidad*

La solución a la complejidad del software puede resolverse a través de ciertas reglas previstas para abatir la complejidad a través de diferentes ángulos. Dentro de las reglas para solucionar la complejidad de un problema con orientación a objetos tenemos la regla de descomposición, regla de abstracción y regla de jerarquía (clasificación).

## Reglas de Descomposición

Un argumento muy fuerte para solucionar problemas es dividir para vencer. Segmentando un problema en subproblemas más simples cada subproblema puede ser atacado sin lidiar contra la totalidad. Existen diferentes formas de descomponer un problema: descomposición algorítmica (estructurada) y descomposición orientada a objetos.

La descomposición algorítmica se ocupa en el diseño estructurado conocida como Top-Down, en donde cada módulo en el sistema es desglosado en un conjunto de módulos más pequeños en coordinación

para dar una solución conjunta, y así sucesivamente se hace por cada módulo de nivel inferior obtenido, así hasta llevar cada módulo a un proceso simple de programación.

La descomposición orientada a objetos divide el problema utilizando abstracciones con responsabilidades obtenidas del dominio del problema. El resultado de la abstracción son objetos capaces de brindar coordinadamente una solución al problema.

Mientras la descomposición algorítmica identifica secuencia de procesos, la descomposición orientada a objetos identifica objetos cooperantes.

## Regla de Abstracción

La dificultad de dominar un objeto en su totalidad nos guía a obtener una idea generalizada del objeto, evitando distraer la atención en detalles no esenciales (solución a sus compromisos), para concentrarnos en los detalles más importantes (sus relaciones). La incapacidad humana de dominar la totalidad de un objeto, escoge ignorar los detalles no esenciales del objeto, para crear una idea globalizada del objeto.

## Regla de Jerarquía

Las jerarquías permiten entender en forma más sencilla al mundo real, existiendo dos jerarquías básicas: la estructura de objetos y la de clases. La estructura de objetos muestra la colaboración de diversos objetos en un patrón de interacción con formato de una totalidad / parte, llamado mecanismo. La estructura de clases muestra la estructura común en base a alguna característica con un funcionamiento sin un propósito causal en un sistema.

# *Importancia del Diseño*

El propósito de un diseño es ofrecer una solución a la estructura del problema, conocida como arquitectura. El diseño es la invención a la solución a un problema, satisfaciendo los requerimientos antes de su implementación. Un buen diseño debe de cumplir con:

- Satisfacer funcionalmente la especificación del problema.
- Satisfacer las restricciones de diseño, tiempo, costo y herramientas disponibles para la construcción.

En la ingeniería de software se cubren diferentes métodos de diseño, identificados por un lenguaje de comunicación, denominado notación, un proceso científico y herramientas para respetar la secuencia del proceso a través de su notación, conocido como CASE.

# Modelo de Objetos

## *Fundamentos*

El término de objetos surge en la década de los setentas en la ciencia computacional, para referenciar notaciones a la solución de problemas aparentemente diferentes, pero mutuamente ligados desde una perspectiva de análisis diferente a la estructurada. Estas notaciones definen objetos representando componentes de un sistema modular descompuesto para definir un conocimiento. Con este modelo se veía a la realidad como objetos con responsabilidades interactuando, en donde cada objeto podía contener objetos más internos para conformar una funcionalidad más compleja.

Dentro de la orientación a objetos durante muchos años se plasmaron diferentes formas de realizar su análisis, diseño y programación, y aún esa variedad sigue presente.

## *Análisis Orientado a Objetos*

El Análisis Orientado a Objetos (AOO) es un método de análisis para examinar los requerimientos desde una perspectiva de clases y objetos, buscados en el vocabulario del dominio del problema, para poder representar la experiencia del usuario en la esquematización del requerimiento.

## *Diseño Orientado a Objetos*

El Diseño Orientado a Objetos (DOO) es un método de diseño para comprender el proceso de descomposición y notación orientada a objetos, obteniendo el modelo lógico (estructuras de clases y objetos) y físico (arquitectura de módulos y procesos), así como los modelos estáticos y dinámicos.

## *Programación Orientada a Objetos*

La Programación Orientada a Objetos (POO) es la implementación del diseño, en donde los programas son colecciones de objetos cooperantes. Cada objeto representa una instancia de alguna clase y las clases pertenecen a una jerarquía de clases relacionadas por la herencia.

## *Elementos*

La orientación a objetos define conceptos para crear soluciones útiles en base a su teoría, definiendo cuatro elementos mayores:

- Abstracción
- Encapsulación

- Modularidad
- Jerarquía

La carencia de alguno de los elementos mayores en la solución de un problema deslinda totalmente de la filosofía orientada a objetos porque justamente estos elementos conforman la estructura primordial de esta filosofía.

Adicionalmente se tienen tres elementos menores del modelo de objetos:

- Tipeamiento
- Concurrencia
- Persistencia

Los elementos menores son muy útiles, pero no son imprescindibles para la solución orientada a objetos.

# Abstracción

La abstracción es el reconocimiento de las similitudes existentes entre objetos, situaciones o procesos del mundo real. Al buscar abstracciones es necesario concentrarse en los detalles importantes o incluso en las similitudes ignorando las diferencias por el momento, para poder describir y especificar algunos detalles del sistema, y pasar por desapercibidos los no relevantes. Un concepto puede ser una abstracción sólo si puede ser descrito, entendido y analizado independientemente de los mecanismos utilizados para cumplir sus propósitos. Las abstracciones siempre estarán influenciadas por el observador, de acuerdo a su experiencia y necesidades.

*Una abstracción denota características esenciales de un objeto, distinguibles de otros tipos de objetos y provee límites conceptuales a la perspectiva del observador*[1].

La abstracción fija su atención en la vista exterior del objeto para separar el funcionamiento esencial del objeto de su implementación, siendo este el principio de barrera de abstracción.

La obtención de las abstracciones del dominio de un problema es la parte medular del diseño orientado a objetos, las cuales pueden agruparse en:

- Abstracción de Entidad: Un objeto representa un modelo útil del dominio de un problema o el dominio de una solución, el cual cuenta con una personalidad. Se asocian generalmente a los sustantivos del dominio del problema.

[1] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 41.

- Abstracción de Acción: Un objeto provee un conjunto generalizado de operaciones, las cuales desempeñan un mismo tipo de función. Se asocian a verbos comunes o similares del dominio del problema.

- Máquina Virtual de Abstracción: Un objeto agrupador de operaciones usadas en un nivel superior o inferior de control. Se asocian a un conjunto de verbos con propósitos comunes.

- Abstracción Coincidental: Un objeto agrupa un conjunto de operaciones o modelos sin relación entre sí. Se asocian a un conjunto de verbos o sustantivos sin relación alguna.

De los tipos anteriores las abstracciones de entidad son directamente paralelas al vocabulario del dominio del problema, por lo que es recomendable tratar de obtenerlas.

Como cliente entendemos a un objeto utilizando los servicios de otro objeto, conocido como servidor. Para obtener la abstracción de un objeto nos podemos basar en los servicios requeridos por sus clientes, representando estos servicios la vista exterior para sus clientes.

Una abstracción debe abarcar el concepto de invariancia, así las operaciones de un objeto se conforman por precondiciones (invariantes asumidas por la operación) y postcondiciones (invariantes satisfechas por la operación), es decir, siempre debe de responder en la misma forma siempre al tener la misma entrada para el mismo estado del objeto.

## Encapsulación

La abstracción de un objeto precede a las decisiones de implementación de sus servicios. La implementación de un objeto será un secreto para la abstracción y debe ser ocultada para sus clientes, quedando únicamente como compromiso responder a su servicio sin importar como lo solucione, logrando que ninguna parte de un sistema complejo dependa de la implementación de algún objeto, orillando a respetar el propósito de la abstracción.

Mientras la abstracción permite pensar el que hacer, la encapsulación se dirige en como se va a realizar, permitiendo cambiar programas en forma confiable con un mínimo de esfuerzo por esta barrera. La abstracción y encapsulación son conceptos complementarios: la abstracción se enfoca en el funcionamiento observable de un objeto, mientras la encapsulación se enfoca en la implementación que da origen a ese funcionamiento.

La clase de la que nace un objeto debe tener dos partes, una interface (servicios para relacionarse con otros objetos, conocida como vista exterior) y una implementación (solución a los servicios, conocida como vista interior).

*La encapsulación es el proceso de compartición de elementos de una abstracción que constituyen su estructura y funcionamiento. La encapsulación sirve para separar la interface de una abstracción y su implementación*[2].

## Modularidad

El segmentar un programa en componentes individuales permite reducir su complejidad, además de agrupar un conjunto de componentes con algún propósito o característica común, permitiendo localizar eficazmente sus componentes.

La modularización se realiza para dividir un sistema en módulos compilables separadamente, pero con ciertas conexiones mínimas de dependencias con otros módulos. La modularidad se encarga de empaquetar abstracciones en unidades discretas, llamadas módulos, es decir, la modularización empaqueta físicamente las clases y objetos del diseño de la estructura lógica.

Al modularizar se aventaja la reducción del costo del software al permitir diseñar y revisar módulos independientes, e incluso reutilizar módulos básicos para cualquier sistema.

Para lograr un buena modularidad, los módulos deben construirse con cohesión (abstracciones relacionadas por grupos lógicos) y acoplamiento débil (para minizmizar la dependencia entre módulos).

*La modularidad es la propiedad de un sistema de descomponerse en un conjunto de módulos cohesivos con acoplamientos débiles*[3].

Existen tres factores capaces de afectar las decisiones de modularización. El primero, los módulos sirven como unidades indivisibles de software que pueden reutilizarse en otras aplicaciones. Segundo, muchos compiladores pueden generar código de objetos en segmentos, uno para cada módulo. Tercero, una buena modularización permite ubicar código rápidamente al funcionar como un mapa.

## Jerarquía

Un conjunto de abstracciones en varias ocasiones pueden formar una jerarquía. Al identificar jerarquías en el diseño, el entendimiento del problema se simplifica.

*Una jerarquía es un rango u orden de abstracciones*[4].

Dentro de las jerarquías se desprenden dos muy importantes:

[2] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 51.
[3] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 57.
[4] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 59.

- Generalización/Especialización (estructura de clases)
- Total/Parte (estructura de objetos)

La jerarquía de generalización/especialización se conforma al agrupar abstracciones dentro del esquema *b es un tipo de a*. Un ejemplo es: una moto es un tipo de vehículo.

La herencia es muy importante en una jerarquía de generalización / especialización por definir una relación entre clases, en donde una clase comparte la estructura o funcionamiento definido en una o más clases (herencia simple y múltiple, respectivamente). La herencia representa una jerarquía de abstracciones en donde una subclase (clase hijo) hereda de una o más superclases (clases padre).

La estructura y funcionamiento común de diferentes clases pueden formar una superclase, así la herencia es una jerarquía de generalización/especialización, en donde la superclase representa abstracciones generalizadas y la subclase representa la especialización de los métodos de la superclase, los cuales son agregados, modificados y ocultados.

La abstracción intenta proveer una barrera para ocultar los métodos y estados, pero la herencia abre esta interface, para ver los métodos y estados sin abstracción.

Existen niveles de abstracción, hablando en términos de jerarquía, una abstracción de alto nivel es una generalización y una abstracción de bajo nivel es una especialización.

La jerarquía de total/parte se conforma al agrupar abstracciones dentro del esquema *d es parte de c*. Un ejemplo es: una alumno es parte de un grupo.

La agregación denota la relación de total/parte de una jerarquía, en donde varios objetos forman parte de un total. El conjunto de un grupo de objetos denota un total, la carencia de alguno de estos objetos no modifica la identidad del total, y la destrucción del total no implica la destrucción de las partes conformantes. Una familia (total) se conforma de un padre, una madre e hijos (partes), la carencia de un hijo no destruye a la familia, pero tampoco la división de la familia destruye a sus integrantes.

La encapsulación ayuda a disminuir la complejidad ocultando detalles, la modularidad ayuda a agrupar la solución a subproblemas, mientras tanto la abstracción y jerarquía permiten clasificar similitudes para simplificar la comprensión.

# Tipeamiento

Un tipo es una caracterización exacta de las propiedades estructurales y funcionales, compartidas por una colección de entidades, clases u objetos.

*Un tipeamiento es forzar a una clase de un objeto, así como objetos de diferentes tipos, a ser intercambiables, o a la mayoría, pudiendo intercambiarse sólo por caminos muy restringidos*[5].

Al tener tipos como unidades de medida es posible dividir distancia entre tiempo para obtener una velocidad, pero no será posible dividir la temperatura entre el peso. El tipeamiento obliga a sólo poder realizar ciertas combinaciones legales de abstracciones.

Un lenguaje puede ser fuertemente tipeado, regularmente tipeado o no ser tipeado, y aún así ser orientado a objetos.

Un tipeo fuerte previene de mezclar abstracciones ilógicas u obtener resultados inesperados. Un tipeo fuerte introduce dependencia semántica, así cualquier pequeño cambio de la base de la clase puede requerir afectar a los clientes que interactuen con ella.

Los beneficios más importantes de un lenguaje fuertemente tipeado son:

- Permite detectar errores en el ciclo de editar, compilar y depurar, a pesar de ser tedioso.
- La declaración de tipos ayuda a documentar un programa.
- La mayoría de los compiladores generan código más eficiente cuando los tipos son declarados.
- Desaparición de comportamientos no previsibles al asumir conversiones no deseadas.
- Sin chequeo de tipos un programa puede tronar misteriosamente en tiempo de ejecución.

## Concurrencia

La concurrencia permite actuar a diferentes objetos al mismo tiempo. Cada programa tiene por lo menos un hilo de control, pero un programa concurrente puede tener varios hilos de control: algunos transitorios y otros durante todo el tiempo de ejecución. Los sistemas ejecutados en máquinas con multiproceso permiten verdaderos hilos de control concurrentes.

Existen dos tipos de concurrencia: la ligera y la pesada. Un proceso ligero comparte el espacio de memoria con otro proceso ligero, mientras la concurrencia de un proceso pesado tiene destinado su propio espacio de memoria.

Un sistema orientado a objetos puede conceptualizar al mundo real como un conjunto de objetos cooperativos concurrentemente, en donde cada objeto activo es un centro de actividad independiente.

[5] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 66.

*La concurrencia es la propiedad de distinguir un objeto activo de uno no activo, en un multiproceso*[6].

## Persistencia

La persistencia se encarga de guardar el estado y la clase de un objeto a través del tiempo y/o espacio.

*La persistencia es la propiedad de un objeto de trascender en tiempo (el objeto continua existiendo aún después del cese de su creador) y/o espacio (el objeto cambia su posición de espacio de memoria en el que se creó)*[7].

# Objetos

## *Naturaleza*

Desde la perspectiva del conocimiento humano un objeto puede ser:

- Alguna cosa tangible o invisible.
- Alguna cosa que puede ser aprendida intelectualmente.
- Alguna cosa hacia la que el pensamiento o acción puede ser dirigido.

Un objeto representa un individuo, un artículo identificable, una unidad o entidad (real o abstracta) con un rol bien definido en el dominio del problema. Otros tipos de objetos pueden ser invenciones de procesos de diseños, para cooperar con otros. Los objetos sirven como mecanismos para proveer un alto nivel de funcionamiento cuando cooperan entre sí.

Un desarrollo orientado a objetos debe pensar todas las cosas como objetos. Aunque esta perspectiva puede ser un poco ingenua, por que no todo en el mundo real es objeto; atributos como el tiempo, la belleza o el color no son objetos, ni tampoco son objetos emociones como el amor o la indignación, pero sí son propiedades potenciales de los objetos. Se debe tener cuidado en no confundir las propiedades de los objetos con objetos.

*Un objeto tiene un estado, un funcionamiento e identidad, la estructura y funcionamiento de objetos similares son definidas en clases comunes (los términos de instancia de clase y objeto refieren lo mismo)*[8].

[6] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 74.
[7] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 77.
[8] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 83.

## Estado

El funcionamiento de un objeto está influenciado por su historia: el orden bajo el que opera un objeto es importante.

*El estado de un objeto son todas las propiedades (normalmente estáticas) de un objeto, más los valores actuales (normalmente dinámicas) de cada una de estas propiedades*[9].

Definimos como propiedad a un atributo, variable simple o referenciada, con una característica inherente o distintiva, rasgo o facción que contribuyen a tener un objeto único. Todas las propiedades contienen algún valor y este valor puede ser una cantidad simple o denotar otros objetos. Todas las propiedades se representan por variables en términos de programación, siendo las variables más complejas las que referencian a objetos. Una cantidad simple, como un número, es independiente del tiempo al no variar su valor, mientras tanto un objeto existe en el tiempo y tiene un estado, además puede ser creado, destruido o compartido.

Los términos propiedad y atributo son indistintos, ambos hacen referencia a las características de un objeto.

## Funcionamiento

El funcionamiento de un objeto representa su exterior visible y actividad medible.

*El funcionamiento de un objeto actúa y reactúa en términos de los cambios de su estado y mensajes enviados o transmitidos*[10].

Una operación es la acción desempeñada por un objeto al invocar a otro objeto sus servicios, con la finalidad de obtener una reacción. El resultado de la operación estará definido por el estado del objeto en el momento de responder al cliente. Las operaciones definidas para un objeto las conocemos como métodos y en términos de programación los conocemos como procedimentos y/o funciones

El estado de un objeto representa un cúmulo de resultados de su funcionamiento.

Un método denota un servicio ofrecido por un objeto a sus clientes. Los tres tipos de métodos mas comunes son:

- Modificador: Método que afecta el estado de un objeto.

- Selector: Método que consulta el estado de un objeto, sin alterar el estado.

[9] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 84.
[10] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 86.

- Iterativa: Método que consulta todas las partes correspondientes al estado de un objeto con algún valor bien definido, en forma repetitiva.

Adicionalmente también existen métodos para crear y destruir objetos:

- Constructor: Método creador de un objeto y/o inicializador de su estado.
- Destructor: Método liberador del estado de un objeto y/o destructor del mismo.

El rol de un objeto es la máscara para definir un contrato entre la abstracción y sus clientes, es decir, el rol de un objeto se define por sus métodos. Las responsabilidades de un objeto son todos los servicios provistos para todos los contratos soportados. Un rol son los servicios ofrecidos por un objeto, mientras la responsabilidad es el cumplimiento de los servicios soportados en un contrato.

Conjuntamente el estado y funcionamiento definen el rol de un objeto a jugar en el mundo, a la vez que satisfacen sus responsabilidades.

Los términos método y operación son indistintos, ambos hacen referencia a su funcionamiento.

## Identidad

Cada objeto es único, incluso a pesar de compartir las mismos propieades de otro objeto, puesto que ocupa un espacio diferente. Esta autenticidad se muestra en la variable de tiempo y espacio.

*La identidad es la propiedad de un objeto de distinguirse de todos los demás objetos*[11].

# *Relaciones entre Objetos*

Al relacionarse dos objetos estos hacen suposiciones acerca del otro, incluyendo los métodos realizados, así como el resultado de su funcionamiento. La jerarquía de objetos se conforma por dos tipos:

- Liga
- Agregación

## Liga

Una liga se deriva de la conexión física (con un mensaje) o conceptual (sin mensaje) de dos objetos, en donde un objeto puede colaborar con otro objeto por medio de un mensaje. El mensaje representa una

[11] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 91.

asociación entre un objeto (cliente) utilizando los servicios de otro objeto (servidor). El mensaje entre dos objetos es generalmente unidireccional y esporádicamente bidireccional.

Un objeto al relacionarse con otro a través de una liga puede desempeñar tres papeles:

- Cliente o Actor: Un objeto puede operar sobre otros objetos, pero nunca podrá ser operado por otros objetos.
- Servidor: Un objeto no opera sobre otros objetos, pero puede ser operado por otros objetos.
- Agente: Un objeto puede operar sobre otros objetos y ser operado por otros objetos, es decir fungir como cliente o servidor para uno o diferentes objetos.

Si consideramos dos objetos a y b con una liga relacionándolos, a puede mandar un mensaje a b sólo si lo puede visualizar. Durante el análisis de un problema se puede ignorar la visibilidad, pero al implementar es necesario considerar la visibilidad a través de las ligas, para determinar el alcance y acceso de los objetos a cada lado de la liga.

Actualmente un objeto puede divisar a otro de cuatro formas:

- El objeto servidor es global al cliente.
- El objeto servidor es algún parámetro del método del cliente.
- El objeto servidor es una parte del objeto del cliente.
- El objeto servidor está localmente declarado como objeto en algún método del cliente.

Dos objetos pueden comunicarse a través de una liga sólo si se encuentran sincronizados y para lograr la sincronización de un objeto activo con uno pasivo puede presentarse en tres formas:

- Secuencial: Un objeto pasivo garantiza sincronía con la existencia de un sólo objeto activo en el momento de buscar la sincronización.
- Protegido: Un objeto pasivo garantiza sincronía con múltiples hilos de control sólo cuando los objetos activos proporcionen exclusión mutua.
- Sincronizado: Un objeto pasivo garantiza sincronía con múltiples hilos de control sólo cuando el objeto pasivo garantice exclusión mutua.

Cuando un objeto pasa un mensaje a otro a través de una liga, los dos objetos deben estar sincronizados de alguna forma. Los mensajes pasados pueden tomar alguna de las siguientes formas.

- Sincronía: Una operación sólo comienza cuando el emisor ha iniciado la acción y el receptor está listo para aceptar el mensaje, el emisor y receptor esperarán indefinidamente hasta que ambas partes estén listas a proceder.

- Resistimiento: Es similar a la sincronía excepto que el emisor abandonará la operación si el receptor no está inmediatamente listo.

- Tiempo de Espera: Es similar a la sincronía excepto que el emisor esperará un tiempo determinado al receptor para que este preparado.

- Asíncrono: El emisor puede iniciar el mensaje sin considerar si el emisor espera el mensaje.

## Agregación

La liga denota una relación entre cliente y servidor, pero una agregación denota una relación jerárquica parte/total para navegar de la totalidad (agregado) a alguna de sus partes (atributos). Si el tiempo de vida de un objeto atributo está condicionado por el tiempo de vida del objeto totalidad, al ser considerado como un parámetro por valor, se conoce a la agregación como *composición*. Pero si el tiempo de vida de un objeto atributo es independiente al tiempo de vida del objeto totalidad, al ser considerado como un parámetro por referencia, se conoce como *agregación*.

La agregación ayuda a encapsular partes como secretos de una totalidad, pero las ligas pueden sustituir a la agregación si se desea romper el acoplamiento entre objetos. La agregación permite tener objetos con funcionamientos complejos, pero visualizados por la encapsulación como objetos simples.

# Clases

## *Naturaleza*

Una clase es un grupo, tipo o conjunto marcado por propiedades comunes, una distinción, una clasificación de cualidades o una clasificación de condiciones.

*Una clase es un conjunto de objetos compartiendo una estructura y un funcionamiento común*[12].

Un objeto es una instancia de una clase, por lo tanto, un objeto no es una clase, pero de una clase nacen los objetos. Los objetos que no comparten estructuras y funcionamientos comunes, no pueden ser agrupados en una clase, excepto por su naturaleza de objetos. Una clase es el molde de un tipo de dato compuesto (con funciones, procedimientos y variables), y un objeto representa la variable alojada en

[12] Object – Oriented Análisis and Design with Applications, Autor Grady Booch, de Benajmin / Cummings. Pag. 103.

memoria del tipo de alguna clase, cuando se asigna un espacio de memoria para el objeto es cuando decimos que fue instanciado.

## Interface e Implementación

Si segmentamos una clase, esta puede tener dos vistas: una externa y otra interna. La externa forma su interface, mientras la interna forma su implementación.

La interface de una clase provee la vista exterior y enfatiza la abstracción, mientras oculta la estructura y secretos del funcionamiento. La interface consiste en declarar inicialmente todos los métodos aplicables a las instancias de esa clase, también pueden incluirse la declaración de otras clases, constantes, variables y excepciones, es decir sus propiedades, necesarias para completar la abstracción. La implementación de una clase provee la vista interna, la cual abarca los secretos del funcionamiento. La implementación de una clase consiste básicamente en implementar todos los métodos definidos en la interface de la clase.

La interface de una clase se divide normalmente en tres partes:

- Pública: Declaraciones accesibles a todos los clientes.
- Protegida: Declaraciones accesibles sólo a su clase, sus subclases y sus amigos.
- Privada: Declaraciones accesibles sólo a su clase y sus amigos.

## *Relaciones entre Clases*

En el dominio particular de un problema, las abstracciones obtenidas se relacionan de diferentes formas, al conjunto de estas relaciones se les conoce como estructura de clases.

Las relaciones entre dos clases tienen dos razones primordiales. La primera, una relación de clases puede indicar el compartir algunas características jerárquicas (generalización / especialización y total / parte). La segunda, una relación de clases puede indicar algún tipo de conexión semántica.

Existen tres tipos básicos de relaciones:

- Generalización/especialización (dos clases se vinculan por la relación es un).
- Total/parte (dos clases se vinculan por la relación parte de).
- Asociación (dos clases se vinculan por dependencia semántica).

Los lenguajes de programación, específicamente los orientados a objetos, proveen soporte directo para algunas combinaciones de las siguientes relaciones:

- Asociación
- Herencia
- Agregación

## Asociación

La asociación denota una dependencia semántica, es decir, su relación es aplicada de forma conceptual y no es obligatoria su dependencia física al codificar. La asociación es bautizada con el nombre del vínculo a través del cual se enlazan las clases e indica una dirección para conocer el sentido de lectura de la asociación para sus clases. En el momento de análisis son sumamente útiles para indicar dependencias existentes, sin entrar en detalles.

Una asociación tiene una cardinalidad entre sus participantes para indicar la cantidad de clases asociada a otra cantidad de clases. Las cardinalidades comunes entre dos clases son:

- Uno a uno
- Uno a muchos
- Muchos a muchos

## Herencia

Una subclase puede heredar la estructura y funcionamiento de su superclase. La herencia es la relación entre clases, en donde una clase comparte la estructura y/o funcionamiento definido en una clase (herencia simple) o más clases (herencia múltiple). La clase con la propiedad de heredar se le denomina *superclase* y a la clase heredadora se le conoce como *subclase.* Una subclase puede aumentar o restringir la estructura y funcionamiento de su superclase, cuando una subclase aumenta su superclase se dice que hereda por extensión. Cuando una superclase es diseñada para agrupar funcionalidad y estructura, pero no se diseña para crear instancias se le conoce como *clase abstracta*.

Una clase tiene dos tipos de clientes principales:

- Instancias
- Subclases

El *polimorfismo* es un concepto, en donde el nombre de un método, puede denotar operaciones de objetos de varias clases, siempre y cuando estas se relacionen por alguna superclase común. Algún objeto estará habilitado para responder según la necesidad del mensaje, a pesar de tener el mismo nombre de método, por ejemplo, es como hablar del método de manejar un automóvil automático y uno estándar, ambos métodos concuerdan en el nombre, pero tienen secuencias de solución diferentes, además ambos derivan de la clase automóvil. El polimorfismo es muy útil cuando tenemos muchas clases con el mismo protocolo. Con el polimorfismo, se eliminan grandes instrucciones de tipo case, porque cada objeto conoce su tipo y por lo tanto sabe reaccionar.

La necesidad de herencia múltiple en los lenguajes de programación orientada a objetos es todavía un gran debate. La herencia múltiple es como tener múltiples padres para un niño. Uno de los problemas levantados por la presencia de herencia múltiple es heredar de un ancestro por más de un camino conceptos similares, esta situación se llama herencia repetida, el problema surge si los padres del niño tiene religiones diferentes, por lo cual elegir una religión será complicado. Existen tres enfoques para tratar la herencia repetida:

1. Las herencias repetidas son ilegales.

2. Si se duplican métodos o propiedades en una subclase al heredar de dos superclases, estas referencias deben calificarse con el nombre la superclase referenciada.

3. Indicar una prioridad en la selección de la superclase dominante para sólo considerar un método o propiedad.

## Agregación

La relación de agregación entre clases tiene un paralelismo directo a la relación de agregación entre objetos, debido a la necesidad de cualquier objeto de instanciarse de alguna clase.

# Clases y Objetos

## *Interacción*

## Relación

En todas las aplicaciones prácticas las clases siempre son estáticas, por lo tanto su existencia, semántica y relaciones son fijas, pero los objetos son dinámicos, al ser creados y destruidos en tiempo de ejecución de un programa.

## Reglas de Análisis y Diseño

Durante el análisis y etapas iniciales del diseño, los desarrolladores tienen dos tareas primordiales:

- Identificar las clases y objetos del vocabulario del dominio del problema.
- Inventar las estructuras por medio de las cuales un conjunto de objetos trabaja para proveer el funcionamiento que satisfaga el dominio del problema.

Las clases y objetos se llaman *abstracción* del problema, y a las estructuras cooperativas se les llaman *mecanismos* de implementación.

En el análisis es importante fijar la atención en la vista exterior de las abstracciones y mecanismos de implementación. Estas vistas representan el marco lógico del sistema, y abarca la estructura de clases y objetos del sistema. En la implementación del diseño la atención se fija en la vista interior de las abstracciones y mecanismos de implementación, envolviendo así el marco físico.

# *Construcción de Calidad*

## Calidad de Abstracción

El proceso de diseño de clases y objetos es incremental e iterativo. Para determinar si una clase u objeto esta bien diseñado puede medirse en base a:

- Acoplamiento
- Cohesión
- Suficiencia
- Entereza
- Primitividad

El *acoplamiento* es la medida de fortaleza en la asociación establecida por una conexión de un módulo a otro, o bien entre clases. Un fuerte acoplamiento evita la independencia entre los módulos o clases del sistema. Un módulo o clase fuertemente acoplado con otro es rígido de entender, cambiar o corregirse, por lo tanto, la complejidad de un diseño de sistema puede ser reducido utilizando un débil acoplamiento entre módulos o clases.

La *cohesión* es la medida del grado de colectividad entre los elementos de un diseño orientado a objetos como un módulo, objetos simples o clases simples. La forma menos deseable de cohesión es la

coincidental, en la cual abstracciones completas no relacionadas son agrupadas en la misma clase o módulo.

El acoplamiento y la cohesión forman criterios para permitir a una clase o módulo ser suficiente, entero y primitivo.

Por *suficiencia* se entiende que la clase o módulo tiene características indispensables de abstracción para permitir un significado e interacción eficiente, pero utilizando lo mínimo indispensable.

Por *entereza* se entiende la interface de la clase o módulo que captura todo lo significativo de la abstracción, sin carencias. Una clase o módulo debe tener una interface general para ser útil a cualquier cliente, lo que nos permitirá tener código reutilizable para diferentes propósitos. También debemos de considerar a mayor generalidad mayor tiempo de diseño y construcción.

Si se proveen demasiadas métodos en niveles finales obtendremos niveles finales complejos, pero es posible dejar muchas métodos en niveles primitivos para lograr una agrupación de las operaciones. Si fuera necesario incrementar la funcionalidad bastará con apoyarse de las clases primitivas.

## Selección de Métodos

Planear la interface de una clase o un módulo es normalmente un trabajado arduo. Es recomendable hacer un primer intento de diseñar una clase, posteriormente al apoyarse en las necesidades de sus clientes creados, se determina si se aumenta, modifica y se refina la interface, pero vale la pena señalar en tener un pensamiento general en todo momento para permitir clases capaces de responder a diferentes clientes.

Si en una clase dada se guardan todos los métodos en forma primitiva, se exhibe un pequeño y bien definido funcionamiento. Un buen diseñador conoce el balance en crear muchos contratos, clases muy grandes en métodos, y pocos contratos,  clases pequeñas con muchas relaciones.

Los métodos pueden situarse tomando en cuenta los siguientes criterios:

- Reusabilidad: Funcionamiento útil en más de un contexto
- Complejidad: Dificultad de implementar el funcionamiento
- Aplicabilidad: Consideraciones de funcionamiento para contextos determinados
- Conocimiento de Implementación: La implementación del funcionamiento depende de detalles internos de la clase

## Selección de Relaciones

La relación entre clases y objetos esta ligada a la selección de métodos. Si se decide que el objeto *a* envíe un mensaje *m* al objeto *b*, entonces directa o indirectamente, *b* puede ser accesible a *a*, para lo cual se debe nombrar la operación *m* en la implementación de *a*, considerando la visibilidad requerida.

## Selección de Implementaciones

Sólo después de establecer la vista exterior de un objeto o clase se puede iniciar el estudio de su vista interior, sólo hasta conocer los contratos de un objeto podemos decidir como satisfacerlos.

La representación de una clase u objeto debe ser, casi siempre, uno de los secretos de la encapsulación de la abstracción. Una ventaja de encapsular es realizar cambios a la representación, siempre y cuando no implique una violación funcional de las suposiciones hechas por el cliente. Esto sucede sin necesidad de afectar a ninguno de sus clientes, porque el protocolo o contrato no se ve afectado.

# Clasificación

## *Importancia*

Para buscar la solución de un problema es necesario descubrir e identificar. El proceso de descubrir debe encontrar abstracciones y mecanismos, desprendidos del vocabulario del dominio del problema, más algunos otros implícitos. Si nos apoyamos en la clasificación de las abstracciones y mecanismos, se identifican jerarquías de generalización, especialización y agregación. La clasificación es una herramienta para modularizar, se pueden situar ciertas clases y objetos en el mismo módulo, dependiendo de las similitudes de sus declaraciones u otro concepto de clasificación. La similitud se puede obtener utilizando la cohesión y el acoplamiento.

La clasificación siempre estará relacionada al enfoque, visión, experiencia y necesidad del clasificador.

## *Identificación de Clases y Objetos*

### Enfoques Modernos y Clásicos

Históricamente, existen tres enfoques generales para clasificar:

- Clasificación Clásica
- Agrupamiento Conceptual
- Teoría de Prototipo

Ninguna clasificación es absoluta, porque algunas estructuras de clases pueden ser mayormente favorecidas de una aplicación a otra. Ningún esquema de clasificación representa la estructura real o el orden natural, algunas clasificaciones pueden ser más significativas respecto a otras, según sea nuestro interés, pero no por representar la realidad mas precisa o adecuada, sino para servir a nuestros fines.

## *Categorización Clásica*

En el enfoque de clasificación clásica, todas las entidades con alguna propiedad o colección de propiedades comunes forman una categoría. La clasificación clásica viene desde Platón, pasando por Aristóteles, por medio de su clasificación de plantas y animales, a través de la pregunta ¿Es animal, mineral o vegetal?

Una cosa se puede nombrar según el conocimiento obtenido de la naturaleza de sus propiedades y efectos. El enfoque clásico usa propiedades relacionadas, como criterio de semejanza entre objetos. Específicamente, se pueden agrupar objetos en conjuntos, dependiendo de la presencia o ausencia de una propiedad particular.

## *Agrupamiento Conceptual*

El agrupamiento conceptual es una variación moderna del enfoque clásico, derivado de intentar explicar el conocimiento. Este enfoque de clases (grupos de entidades) se genera para formular la primera descripción conceptual de las clases, y entonces clasificar las entidades acorde a su descripción.

El agrupamiento conceptual se estrecha mucho con la teoría difusa (multivaluada), en la cual los objetos pertenecen a uno o más grupos en grados variables de aptitudes. El agrupamiento conceptual juzga la clasificación dependiendo de la funcionalidad de sus entidades.

## *Teoría del Prototipo*

La clasificación clásica y agrupamiento conceptual, son suficientes para explicar a la mayoría de las clasificaciones de diseños complejos. La teoría del prototipo es derivada del campo de la psicología cognoscitiva. Si tratamos de clasificar juegos, estos no pueden ser representados apropiadamente por los moldes clásicos, al no compartir propiedades similares todos los juegos. En cambio, la teoría de prototipo unifica la clasificación de juegos como *familia de semejanzas*. Una clase de objetos es representada por un objeto prototipo, y un objeto se considera miembro de esta clase si y sólo si es semejante al prototipo en forma significativa.

## *Aplicación*

Actualmente, estos tres enfoques de clasificación tienen aplicación directa al diseño orientado a objetos. Primeramente, se deben identificar propiedades relevantes al dominio particular del problema. Aquí se

enfocan estructuras y funcionamientos que forman parte del vocabulario del problema (categorización clásica).

Si este enfoque falla en la clasificación de la estructura de una clase, entonces se considera la agrupación de objetos por conceptos. Aquí, se enfocan el funcionamiento de colaboración entre objetos (agrupamiento conceptual).

En caso de falla de los dos enfoques anteriores en capturar el entendimiento del domino del problema, entonces se considera la clasificación por asociación, mediante la cual el agrupamiento de objetos es definido en base a las estrechas semejanzas con algún objeto prototipo (teoría de prototipo).

## *Análisis Orientado a Objetos*

En el análisis, se busca modelar el mundo descubriendo clases y objetos conformantes del dominio del problema, mientras en el diseño, se inventan abstracciones y mecanismos para proveer el funcionamiento del problema.

Existen diferentes enfoques para encontrar las clases y objetos del dominio de un problema:

- Enfoques Clásicos
- Análisis de Funcionamiento
- Análisis de Dominio
- Análisis de Casos
- Análisis Estructurado

## Enfoque Clásico

Algunos metodologistas han propuesto diversas formas de encontrar clases y objetos provenientes de los requerimientos del dominio del problema. Estas aproximaciones se denominan clásicas, por provenir de los principios de la clasificación clásica. En donde la mayor parte de las clases y objetos candidatos se desprenden de los sustantivos obtenidos del dominio del problema. Algunas fuentes de clases y objetos candidatos usualmente son:

- Cosas
- Roles
- Eventos

- Gente
- Lugares
- Organizaciones o Grupos de trabajo
- Conceptos
- Estructuras
- Sistemas
- Dispositivos

En un alto nivel de abstracción, se pueden manejar áreas de asuntos, las cuales son básicamente grupos lógicos de clases relacionados a un alto nivel de funcionamiento del sistema.

## Análisis Funcional

Mientras las aproximaciones clásicas se enfocan a los sustantivos del dominio del problema. También es posible enfocarse a funcionamientos dinámicos de las primeras clases y objetos encontrados. La responsabilidad del análisis es transmitir el propósito de un objeto y su localización en el sistema. Las responsabilidades de un objeto son todos los servicios provistos por los contratos que soporta.

Un grupo de cosas con responsabilidades comunes y jerarquía de clases envuelven superclases (personifican responsabilidades generales), a las que pertenecen las subclases (se especializan en su funcionamiento).

Un enfoque para identificar clases y objetos se deriva del funcionamiento del sistema. Primero, se obtiene el funcionamiento global del sistema, después se asignan funcionamientos a partes del sistema y se comprende a los iniciadores y participantes del funcionamiento de las partes del sistema.

Los iniciadores y participantes con roles importantes se reconocen como objetos, y se les asignan responsabilidades de funcionamiento para estos roles.

## Análisis de Dominio

El análisis de dominio busca identificar clases y objetos comunes a todas las aplicaciones en un dominio dado.

El análisis de dominio es un intento de identificar los objetos, operaciones y relaciones que dominios expertos perciben a ser importantes en nuestro dominio del problema.

Los pasos a seguir en el análisis del dominio son:

1. Construir un modelo genérico del dominio del problema basado sobre un dominio experto.

2. Examinar sistemas existentes dentro del dominio y representarlos en un formato común para su entendimiento.

3. Identificar similitudes y diferencias entre el dominio actual contra un dominio experto.

4. Refinar el modelo genérico.

El análisis de dominio puede ser utilizado en aplicaciones similares y algunas partes de cierta aplicación.

## Análisis de Casos

La práctica de análisis clásicos, análisis de funcionamiento y análisis de dominio dependen de una gran experiencia por parte del analista en el problema. En la mayoría de los casos, esto no sucede.

Los casos de uso es un patrón a seguir, ya que son escenarios que inician con algún usuario del sistema comenzando alguna transacción o secuencia interrelacionada de eventos.

El análisis de casos puede aplicarse a los requerimientos de análisis y desarrollo de escenarios fundamentales al sistema. Estos escenarios, colectivamente, describen el funcionamiento de la aplicación.

Posteriormente, el análisis procede a estudiar cada escenario, usando técnicas de storyboard similares a prácticas de programas de televisión. Así, es posible identificar los objetos participantes en cada escenario, las responsabilidades de cada objeto y la colaboración entre diferentes objetos. De esta forma, los escenarios forzan a una separación de las abstracciones. Al continuar el análisis, estos escenarios iniciales son expandidos para considerar condiciones excepcionales. El resultado de estos escenarios secundarios introduce nuevas abstracciones; o agrega, modifica o reasigna las responsabilidades a abstracciones existentes.

## Análisis Estructurado

Una segunda alternativa al análisis orientado a objetos, es usar productos de análisis estructurado como un front-end al diseño orientado a objetos. Esta técnica es útil por el gran número de analistas diestros en el análisis estructurado. Se inicia con los modelos esenciales del sistema, como los descritos por los diagramas de flujo y otros productos de análisis estructurado.

Para proceder a identificar las clases y objetos en el dominio de nuestro problema existen tres diferentes formas. La primera, inicia con el análisis del diccionario de datos; con la lista de elementos de datos esenciales, se piensa acerca de lo que dicen o describen. Si en la lista se encuentra un adjetivo, el sustantivo asociado al adjetivo es un candidato a objeto.

Las siguientes dos técnicas envuelven un análisis individual de los diagramas de flujo. Dado un diagrama de flujo particular, los candidatos a objetos pueden derivar de:

- Entidades externas
- Almacenadores de datos
- Almacenadores de control
- Transformaciones de control

Las clases se pueden derivar de:

- Flujo de datos
- Flujos de control

La última técnica es el análisis de abstracción, enfocado a identificar las identidades centrales, las cuales son similares en naturaleza a la transformación central en el diseño estructurado. En el análisis estructurado las entradas y salidas de datos son examinados y se sigue hacia el interior, siendo examinados hasta alcanzar un alto nivel de abstracción. Los procesos entre entradas y las salidas forman la transformación central. En el análisis de abstracción un diseñador hace algo similar, pero también se examina la transformación central para determinar los procesos y estados que representan el mejor modelo de abstracción. Después de identificar entidades centrales en un diagrama particular de flujo de datos, el análisis de abstracción continúa para identificar todas las identidades.

El uso de análisis estructurado como front-end al diseño orientado a objetos, ofrece fallas cuando el desarrollador es incapaz de resistir el impulso de caer en el abismo del diseño estructurado.

## *Abstracción*

Las abstracciones son clases u objetos que forma parte del vocabulario del dominio del problema. El primer punto para identificar las abstracciones es delimitar el problema, así entonces se destacan las cosas relevantes del sistema y se suprimen las cosas superfluas que se salen del interés del problema. El seleccionar apropiadamente los objetos depende del propósito de la aplicación.

Las abstracciones envuelven el descubrir y el inventar. El descubrir reconoce las abstracciones usadas por los dominios expertos. La invención crea nuevas clases y abstracciones no necesariamente parte del dominio del problema, pero sí son artefactos en el diseño o la implementación.

Para poder identificar clases y objetos se puede recurrir a usar escenarios.

Al identificar una abstracción como candidata, se evalúa con medidas descritas anteriormente. Siempre es bueno identificar los objetos creados por alguna clase, si puede un objeto de una clase ser copiado o destruido, y saber las operaciones hechas sobre un objeto. En caso de no tener un buen fundamento lo anterior es necesario seguir buscando una solución al problema, antes de seguir avanzando.

Al obtener una nueva abstracción, se puede localizar en el contexto existente de la jerarquía de clases y objetos designada. Algunas veces se buscan clases muy generales que dificultan la herencia a una subclase.

Al bautizar a los elementos participantes del dominio del problema se tiene: el nombre del propio objeto, el nombre de su clase y el nombre del módulo en el que se declara la clase. Para evitar problemas relacionados con los nombres se sugiere:

- Los objetos pueden nombrase con sustantivos en singular, como *sensor*.
- Las clases pueden nombrase con sustantivos en plural, como *Sensores*.
- Los métodos modificadores pueden nombrarse con verbos de actividad, como *dibujar*.
- Las métodos de selección pueden nombrarse con preguntas, como *estáAbierto*.

## *Mecanismos*

El diseño de una clase representa el conocimiento del comportamiento de objetos individuales. Un mecanismo es una decisión de diseño acerca de colecciones de objetos cooperantes. Así, los mecanismos representan patrones de funcionamiento.

Los mecanismos son definiciones por medio de las cuales los objetos colaboran para proveer un alto nivel de funcionamiento.

Los mecanismos que un desarrollador escoge, desde un conjunto de alternativas, ofrecen factores como el costo, confiabilidad, manufacturabilidad y seguridad, entre otros.

Durante el proceso de diseño, el desarrollador puede considerar no sólo el diseño individual de clases, sino también las instancias de estas clases. Cada desarrollador elige un patrón particular de

colaboración, el trabajo es distribuido entre muchos objetos para definir los métodos apropiados en las clases apropiadas.

Los mecanismos representan decisiones estratégicas de diseño, la interface de una clase, es más una decisión táctica de diseño.

Un marco de trabajo es una colección de clases para proveer un conjunto de servicios para un dominio particular. Un marco de trabajo exporta un número de clases y mecanismos que los clientes pueden usar o adaptar.

# CAPÍTULO II. BASE DE DATOS

## Introducción

En las últimas décadas, la humanidad ha vivido un cambio revolucionario en cuanto al manejo de la información. Debido a los avances tecnológicos cada vez se puede poseer mayor información, además de permitirse formas de manipulación y explotación más sencillas.

El problema del manejo de mucha información se convirtió en una necesidad vital en los gobiernos y empresas en la década de los 60, así surgió software encargado de manipular la información eficientemente, bautizando al software con esta característica como Sistema Manejador de Bases de Datos (DBMS).

Actualmente es imposible imaginar el funcionamiento de una empresa sin un DBMS, puesto que, a partir del eficiente acceso a toda la información de su interés, podrá tomar decisiones rápidas y correctas, así como agilizar sus procesos, todo esto para ser más productivo y reducir costos y en algunos casos evitar pérdidas.

Tampoco se puede afirmar que un DBMS define el rumbo a tomar por alguna empresa, solamente permite manipular datos particulares o estadísticas de lo que se le solicite al software, por lo cual se vuelve una herramienta imprescindible, pero a final de cuentas, una herramienta es poderosa de acuerdo a la capacidad de las personas para manipularla.

Por Base de Datos se define a un conjunto de información accesada por un software y representa el modelado de información de una empresa. En general se puede entender lo mismo por información y datos, aunque algunos autores le dan significados diferentes a información y datos.

Un DBMS es un software que permite la interacción entre la información almacenada en un disco y los usuarios. Con esta herramienta podemos insertar, modificar y buscar información.

## Ventajas y Desventajas

Las ventajas que ofrece un manejador de datos son[13]:

- Independencia Física: Los datos no dependen de la forma en que sean almacenados en la memoria secundaria.

[13] Bases de Datos: Gestión de Ficheros, El Modelo Relacional, Algoritmos y Lenguajes, Seguridad de los Datos, Autor Georges Gardarin, de Paraninfo. Pag. 57 - 62.

- Independencia Lógica: Cada usuario de la base de datos podrá visualizar los datos según sean sus necesidades, es decir, cada usuario puede ver solamente los datos de su interés.

- Manipulación de los Datos por Usuarios No Expertos: Las personas podrán manipular la base de datos sin la necesidad de tener profundos conocimientos sobre las bases de datos.

- Eficacia de los Accesos a los Datos: El acceso a la memoria secundaria debe de ser transparente al usuario.

- Redundancia Controlada de los Datos: La duplicación de datos en la base de datos es mínima y controlada, permitiendo información correcta y coherente.

- Coherencia en los Datos: No pueden haber datos que se contradigan en una base de datos.

- Compartición de los Datos: Se puede estar compartiendo datos con otros usuarios al mismo tiempo.

- Seguridad de los Datos: La información puede estar protegida otorgando derechos de acceso a cada uno de los usuarios.

- Abstracción de la Información: El usuario podrá ver los datos en forma que él los solicite, independientemente de la forma en que fueron almacenados.

Las desventajas que presenta en general un manejador de bases son:

- Alto Costo: El costo para implantar un sistema de bases de datos dependiendo de su capacidad es mucho más caro que un manejador de archivos, el cual viene integrado al sistema operativo, pero una vez implementado representa un costo menor.

- Bajo Rendimiento: La velocidad para adquirir los datos puede llegar a ser bastante lenta en algunos casos.

# Modelos de Datos

## *Definición*

El modelo de datos permite representar en forma de datos la realidad, a través de técnicas conceptuales para describir los datos con sus relaciones, semántica y limitaciones que existan.

## Arquitectura

Los modelos de datos están basados en una arquitectura de tres niveles: el interno, el conceptual y el externo.

Nivel Interno: En este nivel se encuentran los datos en forma física sobre los medios de almacenamiento secundario. Pero estos no pueden ser directamente interpretados por el usuario.

Nivel Externo: Es el nivel en que se encuentran los usuarios finales, donde estos pueden visualizar la información de la base de datos que deseen consultar.

Nivel Conceptual: Es el enlace entre el nivel interno y externo. Se encarga de interpretar los datos almacenados en las memorias secundarias de una forma, en que los datos ya tengan sentido, es decir una vista lógica de la base de datos.

## Tipos de Modelos

Dentro de los modelos de datos se encuentran tres divisiones principales, según la forma de analizar los datos: Los modelos lógicos basados en objetos, los modelos lógicos basados en registros y los modelos físicos.

Los modelos lógicos basados en objetos describen los datos en los niveles conceptual y externo. Son bastante flexibles y permiten especificar las limitaciones que tienen los datos. En algunos de los modelos de este tipo tenemos el:

- Modelo Entidad-Relación.
- Modelo Binario.
- Modelo Semántico de Datos.
- Modelo Infológico.

Los modelos lógicos basados en registros también describen los datos a nivel conceptual y externo. Permite describir la base de datos en forma muy cercana a la implementación, pero las limitantes de los datos no son tan claros de visualizar. Los modelos más conocidos en esta división son:

- Modelo de Red.
- Modelo Jerárquico.
- Modelo Relacional.

Los modelos físicos describen los datos a nivel interno, pero actualmente es raro ver su utilización. Dentro de los modelos más conocidos en este rango tenemos:

- Modelo Unificador.
- Modelo de la Memoria de Cuadros.

## *Modelos Lógicos Basados en Registros*

Modelo de Red: Los datos están representados por medio de conjuntos de registros, los cuales están unidos o relacionados por ligas o apuntadores. Los registros se unen en forma arbitraria, es decir, se permiten relaciones entre dos registros de una a una, de una a muchas y de muchas a muchas. La ventaja del modelo de red, es la de permitir reflejar el mundo real de forma similar, es sencillo definir las estructuras de datos requeridas, así como sus limitantes, pero estos suelen volverse muy complejos dependiendo de su tamaño.

Modelo Jerárquico: En el modelo jerárquico los datos están representados por medio de conjuntos de registros, los cuales están unidos por ligas o apuntadores. Los registros están unidos como un conjunto de árboles, es decir, se permiten relaciones entre dos registros de una a una y de una a muchas, puesto que las relaciones de muchas a muchas romperían el esquema de árbol. La desventaja principal de este modelo es que el mundo real no es jerárquico, salvo excepciones, pero es muy sencillo entenderlo.

Modelo Relacional: Los datos y las relaciones entre estos se representan por tablas bidimensionales, es decir, se tiene columnas y renglones, donde cada columna tiene un nombre exclusivo. Permite relaciones de uno a uno y de uno a muchos. Su principal ventaja es conservar una base conceptual limpia, además no sólo se basa en la teoría matemática de conjuntos, sino que también ha generado su propia teoría.

# Conceptos Básicos del Modelo Relacional

El padre de las bases de datos relacionales es Edgar F. Codd[14], por haber sido el creador de estas en el año de 1970 en IBM en San José, California. Gran parte de los sistemas hasta 1980 eran diseñados con el modelo de red o el modelo jerárquico.

*Una base de datos relacional es una base de datos percibida por sus usuarios como una colección de tablas[15].*

[14] Diseño y Gestión de Sistemas de Bases de Datos, Autor Angel Lucas, de Paraninfo. Pag. 83, "*Edgar F. Codd es el creador de las bases de datos relacionales*".

[15] An Introduction To Database Systems, Volume I, Autor C. J. Date, de Addison – Wesley Publishing. Pag. 112, "*A relational database is a database that is perceived by its users as a collection of tables (and nothing but tables)*".

## Estructura Básica

Lo más importante en la ingeniería para tener implementaciones eficientes consiste en tener un buen análisis del problema, esto nos lleva a confeccionar un buen análisis de datos. El análisis debe concentrarse en las necesidades del usuario y en el desarrollo de un modelo preciso de los datos del sistema que se almacenarán en la base de datos. Para poder cumplir lo anterior, necesitamos que el modelo cumpla con los siguientes objetivos:

1. El usuario debe entender fácilmente los datos, pero también el desarrollador, el modelo relacional propone el uso de tablas.

2. Transformación fácil a una implementación física.

3. Aportar una herramientas para poder armar la base de datos:

    i) A un evento solamente se le permitirá almacenarse una vez en la base de datos. Así se eliminará el duplicado de almacenamiento. De otra forma, si existen varios eventos repetidos, al modificar el evento puede que se actualice únicamente una vez en la base de datos.

    ii) Facilidad de evolución de la base de datos. Normalmente el usuario necesita establecer nuevos requerimientos, así la base de datos podrá seguir evolucionando de acuerdo a las nuevos requerimientos.

## Terminología

La terminología de las bases de datos relacionales se ha compuesto de dos vocabularios, el cotidiano y el basado en estructuras matemáticas. El cotidiano se refiere a las relaciones como tablas, una tabla se compone de columnas y renglones. La matemática define a las tablas como relaciones, a las columnas como atributos y a los renglones como registros.

Las tablas que pertenezcan a base de datos relacionales deberán cumplir con las siguientes condiciones:

- Todos los registros tienen la misma cantidad de atributos.
- No pueden existir registros repetidos.
- No pueden existir columnas repetidas.
- Cada columna debe tener un nombre exclusivo.

- En la coincidencia de una columna y un registro no existen valores múltiples.
- Una columna tiene un rango de valores, los cuales deberán tomar los registros.

Si una tabla cumple con lo anterior podrá nombrarse tabla relacional o relación.

Por sí mismas, las tablas son formas cotidianas y explicables de representar datos. Gran parte de la gente ha utilizado tablas para manejar información.

*Un atributo es una característica de una relación, y un conjunto de atributos definen las características de una relación*[16].

*Un registro es la definición de los atributos de un evento situándolos en una relación.*

Si definimos una relación llamada Empleado sus atributos pueden ser número de empleado, nombre, RFC, sueldo y departamento. Donde los atributos permiten definir las características de un empleado y los registros serán los valores particulares tomados por los diferentes empleados.

Empleado

| No Empleado | Nombre | RFC | Sueldo | Departamento |
|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $15,000 | 10 |
| 47 | Edith | FADE781002 | $10,000 | 10 |
| 73 | Gerardo | AOMG721118 | $10,000 | 20 |

*El grado de una relación es igual al número de atributos que la componen. Mientras que la cardinalidad es el número de registros que forman la relación*[17].

La relación anterior tiene un grado de 5 y una cardinalidad de 3.

*El conjunto de valores tomados por un atributo se define como dominio*[18].

Dos valores pueden ser del mismo tipo (cadena, carácter, entero) pero pertenecer a diferentes dominios. Los dominios se clasifican en dos tipos:

1. Dominios Generales o Continuos: Contiene todos los valores posibles entre un valor máximo y un valor mínimo. Por ejemplo los números de empleados deben de estar entre uno y 1000, tomando valores enteros positivos.

2. Dominios Restringidos o Discretos: Contiene determinados valores que no presentan un orden. Un ejemplo son los departamentos de una empresa.

[16] Diseño y Gestión de Sistemas de Bases de Datos, Autor Angel Lucas, de Paraninfo. Pag. 31.
[17] Diseño y Gestión de Sistemas de Bases de Datos, Autor Angel Lucas, de Paraninfo. Pag. 63.
[18] Análisis y Diseño de Base de Datos, Autor Igor Hawryszkiewycz, de Limusa. Pag. 39.

*Un valor nulo es un valor asignable a cualquier atributo independientemente del dominio asignado.*

Los valores nulos pueden representar un atributo no aplicable, un atributo desconocido o un atributo no disponible

# *Llaves*

*Una llave es un atributo o asociación de atributos que localizan a uno y solamente un registro en una relación*[19].

Una de las características de las relaciones es la imposibilidad de repetición de dos registros, por lo tanto en una relación siempre existe una llave, en algunos casos la llave se compone por todos sus atributos.

Las llaves deben cumplir con las siguientes propiedades:

1. Unicidad: La llave direcciona solamente un registro en una relación.

2. No Redundancia: Una llave debe estar compuesta por un conjunto mínimo de atributos capaz de direccionar un sólo registro.

3. Validez: El valor del atributo de una llave no puede ser nulo.

*Una superllave es un conjunto no mínimo de atributos con capacidad de direccionar un sólo registro*.

## Llaves Primarias y Secundarias

Existen dos tipos de llaves: las primarias y las secundarias. En una tabla es posible encontrar varias llaves, a todas estas se les llama llaves candidatas, como solamente puede existir una llave en una relación, la llave elegida para identificar un registro en particular se le denomina llave primaria o principal y a las restantes llaves secundarias o alternativas. La elección entre la llave primaria y las secundarias es un proceso lógico en donde pueden intervenir aspectos físicos, siempre es más rápido situar un registro en base a su llave primaria.

Tomando en cuenta lo anterior, aparecen dos tipos de atributos: los atributos primarios y los atributos secundarios. Los atributos primarios son los que conforman a la llave primaria, mientras que los atributos secundarios o no primarios no pertenecen a la llave primaria. Los atributos primarios se indican con un subrayado dentro de una tabla.

La llave primaria se utiliza para poder acceder a una relación y asociarse a otras relaciones. Por eso es importante tomar las siguientes consideraciones:

[19] Análisis y Diseño de Base de Datos, Autor Igor Hawryszkiewycz, de Limusa. Pag. 44.

- No puede tener valores nulos.
- La memoria que ocupe debe ser mínima.
- Codificación sencilla.
- Sus valores no deben cambiar o ser lo más estáticos posible.

## Llave Foránea

Una llave foránea es el atributo o asociación de atributos que en una relación no es llave, pero sus valores corresponden a la llave primaria de otra relación. El dominio de la llave foránea y la llave primaria debe ser compatible.

En el siguiente ejemplo la llave primaria de la relación Empleado es el No. de empleado, la llave secundaria es el RFC, mientras tanto en la relación Departamento la llave primaria es No. de Departamento que es llave foránea en la relación de Empleado.

Empleado

| No. Empleado | Nombre | RFC | Sueldo | Departamento |
|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $15,000 | 10 |
| 47 | Edith | FADE781002 | $10,000 | 10 |
| 73 | Gerardo | AOMG721118 | $10,000 | 20 |

Departamento

| No. Departamento | Descripción |
|---|---|
| 10 | Sistemas |
| 20 | Marketing |

## *Propiedades de las Relaciones*

Las propiedades de las relaciones desde el punto de vista lógico son:

- A una columna de una relación le corresponde un atributo.
- Los valores de los atributos dependen del dominio.
- El orden de las columnas en la relación no es significativo.
- El orden de los renglones en la relación no es significativo.
- No existen renglones duplicados.

- Una relación puede tener varias llaves.

# Álgebra Relacional

El álgebra relacional consiste en una colección de operadores de alto nivel que actuan sobre las relaciones. Estas operaciones toman como operandos a las relaciones y el resultado de la operación es una relación, es decir sus entradas son relaciones y su salida es otra relación.

Las operaciones del álgebra relacional pueden dividirse en básicas y derivadas, las básicas son independientes de las demás y las derivadas son operaciones compuestas de las básicas. Dentro de las básicas existen las que tienen una relación de entrada, llamadas unarias, y las que tienen dos relaciones como entrada, llamadas binarias.

- Operaciones Básicas
    - Operaciones Unarias:
        - Selección
        - Proyección
    - Operaciones Binarias:
    - Unión
    - Diferencia
    - Producto Cartesiano
- Operaciones Derivadas
    - Intersección
    - División
    - Join

Las siguientes relaciones serán utilizadas como operandos para los ejemplos de las operaciones relacionales algebraicas.

Empleado

| No. Empleado | Nombre | RFC | Sueldo | Departamento |
|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $15,000 | 10 |
| 47 | Edith | FADE781002 | $10,000 | 10 |
| 73 | Gerardo | AOMG721118 | $10,000 | 20 |

Empleado_Sistemas

| No. Empleado | Nombre | RFC | Sueldo | Departamento |
|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $15,000 | 10 |
| 47 | Edith | FADE781002 | $10,000 | 10 |

Empleado_Especial

| No. Empleado | Nombre | RFC |
|---|---|---|
| 32 | Marissa | ROCM710110 |

Empleado_Externo

| No. Empleado | Nombre | RFC |
|---|---|---|
| 32 | Manuel | FADM721003 |

Departamento

| No. Departamento | Descripción |
|---|---|
| 10 | Sistemas |
| 20 | Marketing |

## *Selección*

La selección extrae registros específicos desde una relación determinada. El resultado de una selección es una relación formada por un subconjunto de registros de una relación con todos sus atributos. Para poder seleccionar el subconjunto de registros existe una condición. Su sintaxis es: S(R(condición)), donde S es selección y R es una relación. La condición se define por la comparación de un atributo con un valor particular de este, en caso de ser necesario tener condiciones adicionales se utilizan los operadores and, or y not. La comparación utilizan los símbolos =, <, >, <=, >=, <>.

Si hacemos una selección de la relación Empleado con la condición Departamento = 10 tenemos:

S ( Empleado ( Departamento = 10 ) )

| No. Empleado | Nombre | RFC | Sueldo | Departamento |
|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $15,000 | 10 |
| 47 | Edith | FADE781002 | $10,000 | 10 |

## *Proyección*

La proyección extrae atributos específicos desde una determinada relación. El resultado de una proyección es una relación formada por un subconjunto de atributos de una relación con todos sus registros. Los registros duplicados en la relación resultante sólo aparecen una vez. Su sintaxis es: P(R($a_1$, $a_2$, $a_3$,...)), donde P es proyección, R es una relación y $a_i$ es un atributo de la relación.

Si hacemos una proyección de la relación Empleado con los atributos No. Empleado, Nombre y RFC tenemos:

P( Empleado ( No. Empleado, Nombre, RFC ) )

| No. Empleado | Nombre | RFC |
|---|---|---|
| 32 | Marissa | ROCM710110 |
| 47 | Edith | FADE781002 |
| 73 | Gerardo | AOMG721118 |

## *Unión*

La unión construye una relación consistente de todos los registros de las dos relaciones especificadas. Solamente es posible unir dos relaciones si tienen el mismo grado y sus dominios son compatibles. El resultado será una nueva relación con los atributos de una de las relaciones y los registros de ambas. Si en la relación resultante existen registros repetidos solamente aparecerán una vez. Su sintaxis es: $R_1 \cup R_2$, donde $R_1$ y $R_2$ son relaciones.

Si realizamos la unión de Empleado_Especial y Empleado_Externo tenemos:

Empleado_Especial ∪ Empleado_Externo

| No. Empleado | Nombre | RFC |
|---|---|---|
| 32 | Marissa | ROCM710110 |
| 32 | Manuel | FADM721003 |

## Diferencia

La diferencia construye una relación que consiste en todos los registros de la primera relación, quitando los que aparecen en la segunda relación. La diferencia se podrá llevar a cabo entre dos relaciones si tienen el mismo grado y sus dominios son compatibles. Su sintaxis es: $R_1$ - $R_2$, donde $R_1$ y $R_2$ son relaciones. El resultado de la diferencia es una relación con los atributos de $R_1$ y los registros de $R_1$ que no están en $R_2$.

La diferencia entre Empleado y Empleado_Sistemas es:

Empleado – Empleado_Sistemas

| No. Empleado | Nombre | RFC | Sueldo | Departamento |
|---|---|---|---|---|
| 73 | Gerardo | AOMG721118 | $10,000 | 20 |

## Producto Cartesiano

El producto cartesiano construye una relación con todas las posibles combinaciones de registros de dos relaciones. Se lleva a cabo entre dos relaciones con grado m y n. El resultado es una relación con grado m + n con todos los registros que resulten de concatenar las dos relaciones, por tanto la cardinalidad de la nueva relación es el producto de las cardinalidades de las dos relaciones. Su sintaxis es: $R_1$ X $R_2$, donde $R_1$ y $R_2$ son relaciones.

El producto cartesiano de Empleado por Departamento es:

Empleado X Departamento

| No. Empleado | Nombre | RFC | Sueldo | Departamento | No. Departamento | Descripción |
|---|---|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $15,000 | 10 | 10 | Sistemas |
| 47 | Edith | FADE781002 | $10,000 | 10 | 20 | Marketing |
| 73 | Gerardo | AOMG721118 | $10,000 | 20 | 10 | Sistemas |
| 32 | Marissa | ROCM710110 | $15,000 | 10 | 20 | Marketing |
| 47 | Edith | FADE781002 | $10,000 | 10 | 10 | Sistemas |
| 73 | Gerardo | AOMG721118 | $10,000 | 20 | 20 | Marketing |

## Intersección

La intersección construye una relación formada por todos los registros que aparezcan en las dos relaciones especificadas. La intersección se deriva de la operación de diferencia, por lo tanto, las dos relaciones deben tener el mismo grado y dominios compatibles. Su sintaxis es $R_1 \cap R_2$ y es igual a: $R_1 \cap R_2 = R_1 - (R_1 - R_2)$, donde $R_1$ y $R_2$ son relaciones. El resultado es una relación con las columnas de $R_1$ y los registros comunes a $R_1$ y $R_2$.

Si hacemos la intersección de Empleado con Empleado_Sistemas obtenemos:

Empleado ∩ Empleado_Sistemas

| No. Empleado | Nombre | RFC | Sueldo | Departamento |
|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $15,000 | 10 |
| 47 | Edith | FADE781002 | $10,000 | 10 |

## *División*

La división se realiza entre dos relaciones $R_1$ y $R_2$ con diferente grado que cumplen con las siguientes condiciones:

1) $R_1$ debe de tener un grado mayor a $R_2$ y contener los atributos de $R_2$.

2) La cardinalidad de $R_2$ debe ser diferente de cero.

El resultado es una nueva relación con grado m, donde m es igual a los atributos de $R_1$ que no están en $R_2$. Su sintaxis es $R_1 / R_2$ y es igual a:

R1 / R2 = W - Z

W = P ( R1 (atributos no comunes a R2))

Z = P (A (atributos no comunes a R2))

A = (W X R2) - R1

Si hacemos la división de Empleado entre Empleado_Especial generamos el resultado:

W = P ( Empleado ( Sueldo, Depto))

| Sueldo | Departamento |
|---|---|
| $15,000 | 10 |
| $10,000 | 10 |
| $10,000 | 20 |

W x Empleado_Especial

| No. Empleado | Nombre | RFC | Sueldo | Departamento |
|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $15,000 | 10 |
| 32 | Marissa | ROCM710110 | $10,000 | 10 |
| 32 | Marissa | ROCM710110 | $10,000 | 20 |

A = ( W x Empleado_Especial ) - Empleado

| No. Empleado | Nombre | RFC | Sueldo | Departamento |
|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $10,000 | 10 |
| 32 | Marissa | ROCM710110 | $10,000 | 20 |

Z = P ( A ( Sueldo, Depto ) )

| Sueldo | Departamento |
|---|---|
| $10,000 | 10 |
| $10,000 | 20 |

Empleado / Empleado_Especial = W - Z

| Sueldo | Departamento |
|---|---|
| $15,000 | 10 |

## *Join*

El join se obtiene de realizar el producto cartesiano de dos relaciones, aplicando sobre la relación resultante la operación de selección. Su sintaxis es: $R_1$ * $R_2$ (condición) = S(($R_1$X$R_2$)(condición)), donde $R_1$ y $R_2$ son relaciones.

Si realizamos un join entre Empleado y Departamento con la condición de Empleado.Departamento = Departamento.No. Departamento tenemos:

Empleado x Departamento

| No. Empleado | Nombre | RFC | Sueldo | Departamento | No. Departamento | Descripción |
|---|---|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $15,000 | 10 | 10 | Sistemas |
| 47 | Edith | FADE781002 | $10,000 | 10 | 20 | Marketing |
| 73 | Gerardo | AOMG721118 | $10,000 | 20 | 10 | Sistemas |
| 32 | Marissa | ROCM710110 | $15,000 | 10 | 20 | Marketing |
| 47 | Edith | FADE781002 | $10,000 | 10 | 10 | Sistemas |
| 73 | Gerardo | AOMG721118 | $10,000 | 20 | 20 | Marketing |

Empleado * Departamento ( Empleado.Departamento = Departamento.No. Departamento )

| No. Empleado | Nombre | RFC | Sueldo | Departamento | No. Departamento | Descripción |
|---|---|---|---|---|---|---|
| 32 | Marissa | ROCM710110 | $15,000 | 10 | 10 | Sistemas |
| 47 | Edith | FADE781002 | $10,000 | 10 | 10 | Sistemas |
| 73 | Gerardo | AOMG721118 | $10,000 | 20 | 20 | Marketing |

# Formas Normales

Codd inventó las tres primeras formas para normalizar una base de datos[20]. El objetivo de la normalización es obtener independencia de los datos con sus aplicaciones, para esto, trata de obtener el mayor número de relaciones posibles, haciendo relaciones con la cantidad mínima indispensable de atributos.

## *Ventajas*

Algunas de las ventajas obtenidas tras normalizar una base de datos son:

- Claridad: Las relaciones sólo tienen los datos indispensables, identificando claramente su objetivo.
- Precisión: La conexión entre las relaciones brinda información exacta.
- Seguridad: La seguridad es más sencilla de implementar.
- Independencia de Datos: Los datos no dependen de los programas, así se podrá permitir el crecimiento de la base de datos.
- Mínima Redundancia: La duplicidad de información será la mínima necesaria.
- Rendimiento: Sólo se tratará información útil al usuario.

## *Dependencia Funcional*

Si A y B son atributos o conjunto de atributos, entonces B depende funcionalmente de A, y se representa como A$\rightarrow$B ó A DF B sí y sólo sí a cada valor de A corresponde un valor de B.

Si en una relación tuviéramos los atributos de empleado y nombre, existe una dependencia funcional entre los dos atributos, porque a un empleado le corresponde un sólo nombre, es decir:

$$\text{empleado} \rightarrow \text{nombre}$$

Si suponemos que el nombre no puede repetirse, entonces se cumple también con:

$$\text{nombre} \rightarrow \text{empleado}$$

Por lo tanto podemos abreviar este ejemplo con:

$$\text{empleado} \leftrightarrow \text{nombre}$$

[20] An Introduction To Database Systems, Volume I, Autor C. J. Date, de Addison – Wesley Publishing. Pag. 528.

En pocas situaciones tenemos el caso de dependencia funcional biunívoca, como la anterior, generalmente tenemos una dependencia de uno a muchos. Un empleado pertenece a un departamento, pero a un departamento pertenecen varios empleados:

$$\text{empleado} \rightarrow \text{departamento}$$

$$\text{departamento} \overset{+}{\rightarrow} \text{empleado}$$

También tenemos una dependencia funcional de muchos a muchos. Si analizamos el caso de que un empleado trabaja en diferentes proyectos, pero también en un proyecto participan diferentes empleados:

$$\text{empleado} \overset{+}{\leftrightarrow} \text{proyecto}$$

A veces es necesario tener un conjunto de atributos para identificar otros atributos. Así, a través de un empleado y una fecha de aumento podemos saber el salario en una fecha.

$$\text{empleado.fecha_sueldo} \rightarrow \text{sueldo}$$

El operador punto puede ser leído como: y o junto con.

En otras ocasiones necesitamos indicar que con un atributo podemos encontrar diferentes atributos. Así, por medio de un empleado conocemos el nombre o el departamentoo del trabajador.

$$\text{empleado} \rightarrow \text{nombre} \mid \text{departamento}$$

El operador | se lee como: o también.

# Dependencia Funcional Total

Al unir dos atributos con el operador punto tenemos una llave o una superllave, porque nos permite encontrar un atributo específico.

Si los atributos unido por el operador punto forman una superllave entonces la dependencia funcional es parcial, pero si forman una llave entonces la dependencia funcional es total.

# Dependencia Funcional Transitiva

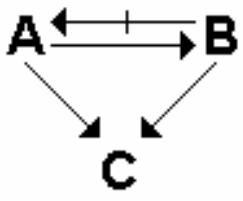
Si existen tres atributos en una relación (A, B, C) que cumplan con las siguientes condiciones:

$$A \rightarrow B \quad \text{y} \quad B \overset{+}{\rightarrow} A$$

Decimos que C tiene una dependencia funcional transitiva con A si se cumple:

$$B \rightarrow C$$

En forma gráfica tenemos:




Por lo tanto, un atributo C es transitivamente dependiente de A si es posible conocerlo por caminos diferentes: directamente y a través de un atributo intermedio B.

## Diagrama de Dependencia Funcional

El diagrama de dependencia funcional es una forma muy clara de visualizar la correspondencia entre los atributos. Los atributos de la llave de la relación se representan dentro de una caja de líneas continuas y el resto de los atributos quedan fuera de la caja. Si existe dependencia de varios atributos primarios estos irán en una caja de líneas discontinuas. Las dependencias de los atributos de las llaves secundarias con los demás atributos no se dibujan.

Si tenemos las dependencias:

$$A.B.C \rightarrow M|N|S$$

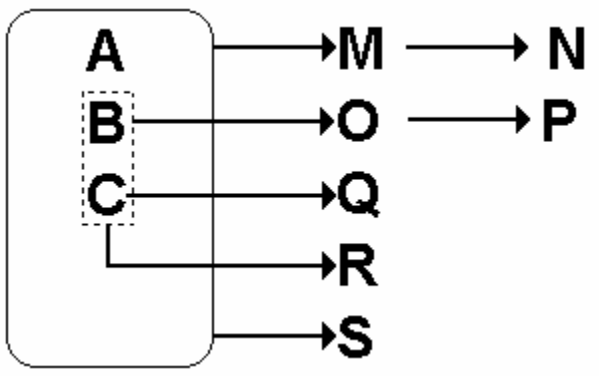
$$M \rightarrow N$$

$$B.C \rightarrow O|P|R$$

$$O \rightarrow P$$

$$C \rightarrow Q$$

El diagrama que las representa es:



## Comprensión de Atributos

Antes de realizar el análisis de dependencia funcional es necesario realizar un análisis de atributos.

Datos

| NumEmp | Nombre | Telefono | CveDepto | Departamento | CveProy | Proyecto | HrsProy |
|---|---|---|---|---|---|---|---|
| 32 | Marissa | 784 42, 536 13 | 10 | Sistemas | M1 | Promociones | 30 |
| 47 | Edith | 559 85, 268 27 | 10 | Sistemas | F1 | Presupuestos | 40 |
| 73 | Gerardo | 557 17 | 20 | Marketing | M1 | Promociones | 40 |
| 32 | Marissa | 784 42, 536 13 | 10 | Sistemas | F1 | Presupuestos | 10 |

Antes de intentar normalizar debemos comprender de manera impecable el significado y dominio de los atributos:

| Campo | Descripción | Tipo de Dato |
|---|---|---|
| NumEmp | Número de Empleado Dentro de la Empresa | Numérico |
| Nombre | Nombre Completo del Empleado | Cadena |
| Telefono | Teléfonos del Empleado | Cadena |
| CveDep | Número asignado a cada departamento | Numérico |
| Depto | Nombre del Departamento | Cadena |
| CveProy | Clave de Proyecto formado por Inicial del Departamento Propietario del Proyecto y un consecutivo | Numérico |
| Proyecto | Nombre del Proyecto | Cadena |
| HrsProy | Horas Semanales Invertidas en el Proyecto por el empleado | Numérico |

El nombre en varios casos es mejor desglosarlo en Nombre, Apellido Paterno y Apellido Materno para realizar búsquedas más eficientes al buscar por cualquiera de los tres campos, pero en este caso por simplificar la cantidad de atributos sólo manejaremos el Nombre.

## *Primera Forma Normal (1FN)*

*Una relación está en 1FN sí y sólo sí los valores de los registros son atómicos para un atributo.*

Es decir, el valor de un atributo para un registro no puede tomar un conjunto de valores, deben ser elementales y únicos.

Para transformar una relación a 1FN seguiremos la siguiente secuencia

1. De la relación identificaremos la llave primaria y los atributos con valores no elementales y únicos.

En la relación Datos su llave primaria es NumEmp y CveProy y los atributos con valores no elementales y únicos tenemos Telefono y CveProy

2. Si algún atributo de la llave primaria no respeta la primera forma normal será necesario encontrar los atributos conformantes y desglosarlo. Si esta labor no puede llevarse acabo debe dudarse si el atributo forma parte de la llave primaria.

El atributo de CveProy se observa que está conformado por un IdDepartamento (M – Marketing, y F – Finanzas) y un consecutivo.

Datos

| NumEmp | IdDepto | CveProy | Nombre | Telefono | CveDepto | Departamento | Proyecto | HrsProy |
|---|---|---|---|---|---|---|---|---|
| 32 | M | 1 | Marissa | 784 42, 536 13 | 10 | Sistemas | Promociones | 30 |
| 47 | F | 1 | Edith | 559 85, 268 27 | 10 | Sistemas | Presupuestos | 40 |
| 73 | M | 1 | Gerardo | 557 17 | 20 | Marketing | Promociones | 40 |
| 32 | F | 1 | Marissa | 784 42, 536 13 | 10 | Sistemas | Presupuestos | 10 |

Como vemos en la relación, CveProyecto ahora representa el consecutivo por departamento para un proyecto. La nueva llave primaria será ahora NumEmp, IdDepto y CveProy.

3. Se realiza una proyección de la relación con los atributos de la llave primaria y los atributos con valores únicos respetando el mismo nombre de la relación (el teléfono puede alojar varios valores).

Datos

| NumEmp | IdDepto | CveProy | Nombre | CveDepto | Departamento | Proyecto | HrsProy |
|---|---|---|---|---|---|---|---|
| 32 | M | 1 | Marissa | 10 | Sistemas | Promociones | 30 |
| 47 | F | 1 | Edith | 10 | Sistemas | Presupuestos | 40 |
| 73 | M | 1 | Gerardo | 20 | Marketing | Promociones | 40 |
| 32 | F | 1 | Marissa | 10 | Sistemas | Presupuestos | 10 |

4. En el ámbito de los atributos con valores múltiples podemos tener diferentes esquemas de solución:

4.1. El atributo es un conjunto de valores del mismo tipo con longitud muy variable, entonces se realiza una proyección sobre la relación original con la llave primaria y los atributos con valores múltiples, descomponiéndolos en valores atómicos y conformándose la llave con todos los atributos de la relación.

Datos_Telefono

| NumEmp | IdDepto | CveProy | Telefono |
|---|---|---|---|
| 32 | M | 1 | 784 42 |
| 32 | M | 1 | 536 13 |
| 47 | F | 1 | 559 85 |
| 47 | F | 1 | 268 27 |
| 73 | M | 1 | 557 17 |
| 32 | F | 1 | 784 42 |
| 32 | F | 1 | 536 13 |

4.2. El atributo se conforma de la unión de otros atributos perfectamente separables por conceptos diferentes, entonces se procede a segmentar el valor del atributo sobre la relación resultante del punto tres.

Datos

| NumEmp | IdDepto | CveProy | Nombre | TelCasa | TelTrabajo | CveDepto | Departamento | ... |
|---|---|---|---|---|---|---|---|---|
| 32 | M | 1 | Marissa | 784 42 | 536 13 | 10 | Sistemas | ... |
| 47 | F | 1 | Edith | 559 85 | 268 27 | 10 | Sistemas | ... |
| 73 | M | 1 | Gerardo | 557 17 | | 20 | Marketing | ... |
| 32 | F | 1 | Marissa | 784 42 | 536 13 | 10 | Sistemas | ... |

4.3. Dependiendo del punto de vista del diseñador podemos ver que en la solución 4.1 el tipo de teléfono se pierde, y en la solución 4.2 fue útil para separar la llave en el punto 2, pero para este caso si se decide aumentar otro tipo de teléfono nuestro diseño requeriría un cambio. Si consideramos la carencia de información de diferentes tipo de teléfono en diferentes registros implicaría una pérdida de espacio reservado. Esto deriva en una tercera solución, incluir al resultado de 4.1 el tipo de teléfono y la nueva llave deberá ser analizada según el planteamiento del problema.

Datos_Telefono

| NumEmp | IdDepto | CveProy | TipoTel | Telefono |
|---|---|---|---|---|
| 32 | M | 1 | CASA | 784 42 |
| 32 | M | 1 | TRABAJO | 536 13 |
| 47 | F | 1 | CASA | 559 85 |
| 47 | F | 1 | TRABAJO | 268 27 |
| 73 | M | 1 | CASA | 557 17 |
| 32 | F | 1 | CASA | 784 42 |
| 32 | F | 1 | TRABAJO | 536 13 |

Las tres soluciones son válidas, lo importante es darle peso a los diferentes factores, si fuera simplicidad el resultado de 4.2 lo ofrece, si fuera crecimiento y espacio las otras dos soluciones lo ofrecen, la diferencia entre ambas es la pérdida del tipo de teléfono de 4.1, pero si no fuera información de interés para el negocio pudiera resultar en la mejor opción. El atributo nombre pudiera se segmentado en Nombre, Apellido Paterno y Apellido Materno, la decisión lo determinarán las necesidades actuales y futuras en base al requerimiento del entorno.

## *Segunda Forma Normal (2FN)*

*Una relación se encuentra en 2FN sí y sólo sí cumple con:*

- *Se encuentra en 1FN.*
- *Todo atributo secundario tiene una dependencia funcional total con la llave primaria.*

La 2FN sólo es estudiada en relaciones con llaves primarias formadas por un conjunto de atributos. Si la llave primaria tiene un sólo atributo entonces ya está en 2FN.

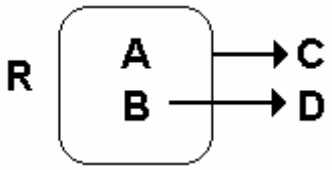
Si una relación tiene como atributos A, B, C, D y la llave primaria es A.B y se poseen las siguientes dependencias funcionales:

$$A.B \rightarrow C$$

$$B \rightarrow D$$

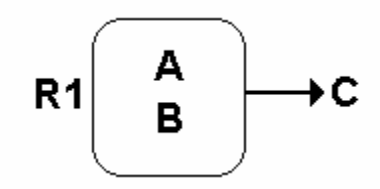
Entonces la relación no se encuentra en 2FN por no tener D una dependencia funcional total con la llave A.B, sino con una parte de la llave (B).

El diagrama de dependencia funcional es:

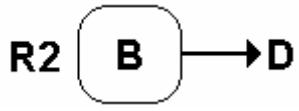



Si existe una flecha que parta del interior de la caja, entonces la relación no está en 2FN. Para pasar una relación a 2FN realizamos:

1. Una proyección de la relación original con la llave primaria y los atributos secundarios con dependencia funcional total.




2. La proyección de la relación original con la parte de la llave primaria y los atributos secundarios con dependencia funcional parcial. Donde la nueva relación tendrá por llave primaria la parte de la llave primaria original.




Las tablas obtenidas por normalización son:

R1

| A | B | C |
|---|---|---|
| | | |

R2

| B | D |
|---|---|
| | |

## *Tercera Forma Normal (3FN)*

*Una relación está en 3FN sí y sólo sí se cumple:*

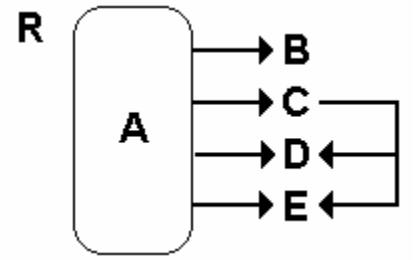
- *Se encuentra en 2FN.*
- *Los atributos secundarios sólo se conocen por la llave primaria o las llaves secundarias.*

En el diagrama de dependencias funcionales sólo se deben mostrar las dependencias funcionales transitivas.

Si en una relación tenemos los atributos A, B, C, D y E, donde A es la llave primaria, B es llave secundaria y se tienen las siguientes dependencias funcionales:

| | |
|---|---|
| A→B | A→C |
| A→D | A→E |
| B→A | B→C |
| B→D | B→E |
| C→D | C→E |

El diagrama de dependencias funcionales es:

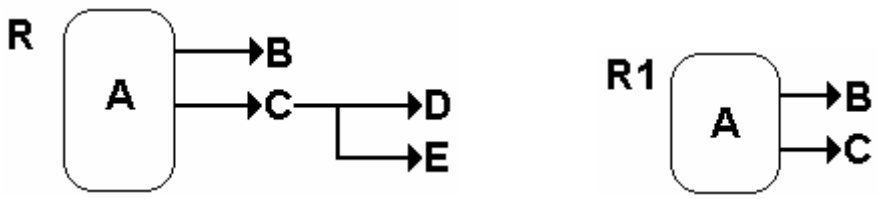



Visto de otra forma:

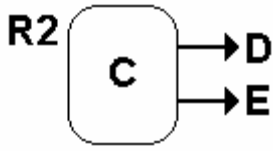
Las flechas que tienen dependencia funcional de las llaves secundarias no se ilustran. La relación no está en 3FN, por que los atributos D y E son transitivamente dependientes de la llave primaria A.

Para transformar una relación a 3FN se realiza:

1. La proyección de la relación con la llave primaria y todos los atributos secundarios no transitivos. El atributo secundario que genera transitividad será una llave foránea en esta nueva relación.




2. La proyección de la relación con los atributos transitivos y el atributo secundario por el cual hay transitividad (nueva llave primaria de la relación).




Las tablas obtenidas por normalización son:

R1

| A | B | C |
|---|---|---|
| | | |

R2

| C | D | E |
|---|---|---|
| | | |

## *Normalizando*

Después de realizar la comprensión de atributos y aplicar la 1FN es necesario identificar las relaciones existentes entre los atributos, principalmente las relaciones de dependencia funcional total.

Es muy importante entender el concepto de dependencia funcional total, en caso de duda podemos apoyarnos en la siguiente pregunta: ¿A un atributo1 cuantos valores de atributo2 le pueden corresponder como máximo?, por ejemplo ¿A un NumEmp cuantos valores de Nombres le pueden corresponder como máximo?, si la respuesta es uno entonces es una dependencia funcional total o transitiva. El atributo1 puede representar la unión de varios atributos. La pregunta siempre debe plantearse en términos generales para todo el universo de posibles valores y como resultado de la respuesta sólo considerar el mayor resultado posible.

También podemos observar que IdDepto representa CveDepto, por lo tanto debido a no tener un uso importante IdDepto, salvo la de identificar el proyecto asignado a un departamento trabajaremos sólo con CveDepto para dejar de utilizar IdDepto.

NumEmp → Nombre | TelCasa | TelTrabajo | CveDepto | Departamento

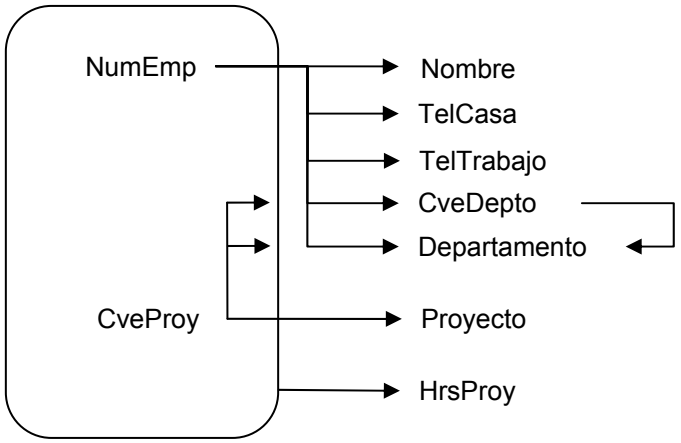
CveDepto → Deptartamento y Departamento → CveDepto

CveProy → Proyecto | CveDepto | Departamento y Proyecto → CveProy | CveDepto | Departamento

NumEmp.CveProy → HrsProy

Las dependencias de Departamento → CveDepto y Proyecto → CveProy, son más útiles en el sentido inverso, es más útil tener una llave primaria corta al momento de accesar las relaciones.

El siguiente diagrama de dependencias muestras las dependencias anteriores:



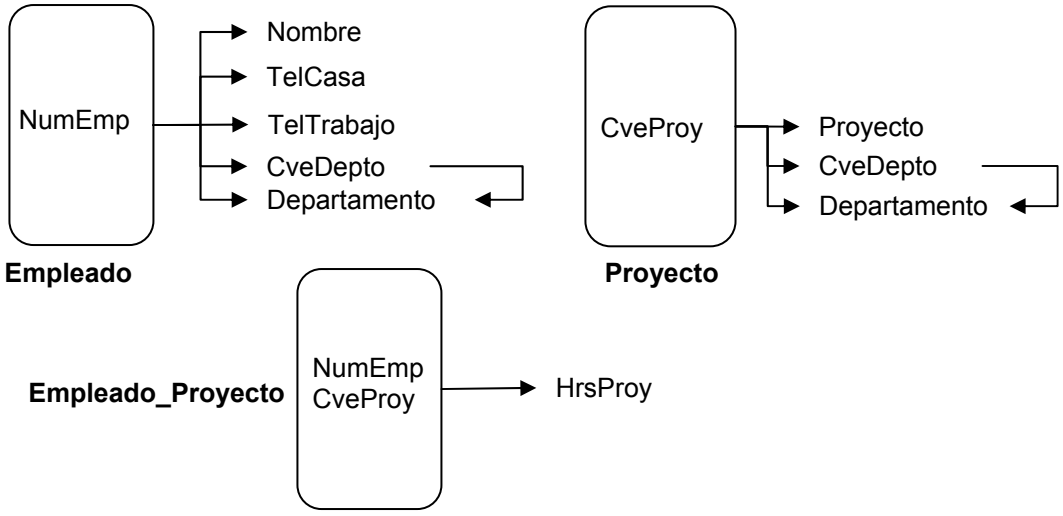

Nuestra relación presenta dos dependencias para normalizar a 2FN.

NumEmp → Nombre | TelCasa | TelTrabajo | CveDepto | Departamento (Dependencia Funcional Tipo B → D)

CveProy → Proyecto | CveDepto | Departamento (Dependencia Funcional Tipo B → D)

NumEmp.CveProy → HrsProy ((Dependencia Funcional Tipo A.B → C)

Normalizando con 2FN tenemos:



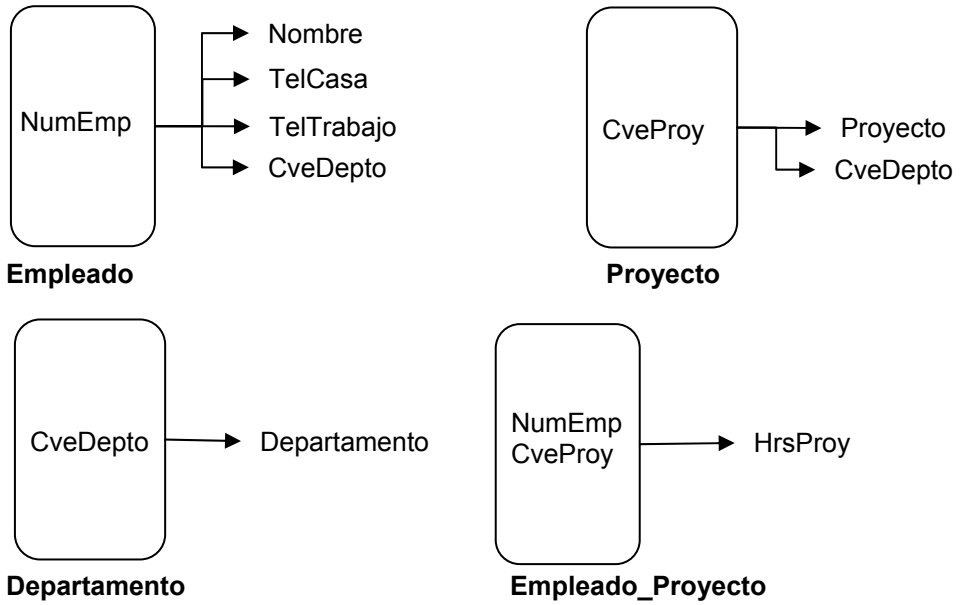

Todavía nuestro diagrama presenta una dependencia para aplicar la 3FN.

NumEmp → Nombre | TelCasa | TelTrabajo | CveDepto | Departamento (Dependencia Funcional Tipo A → D | E)

CveProy → Proyecto | CveDepto | Departamento (Dependencia Funcional Tipo A → D | E)

CveDepto → Deptartamento (Dependencia Funcional Tipo C → D | E)

Normalizando con 3FN tenemos:



Este sería nuestro diseño final después de normalizar la relación Datos.

# Reglas de Codd

Después de la gran popularidad de las bases de datos relacionales surgieron muchas bases de datos que se decían relacionales, pero dejaban mucho que desear. Por esto, Codd publicó un artículo en la revista Computerworld en donde dictaminó las doce reglas a cumplir por una base de datos relacional[21].

Las doce reglas de Codd son[22]:

1. Regla de Información: La regla de la información requiere que toda la información de una base de datos este representada de una y solo una forma, específicamente por valores dentro de columnas posicionados dentro de los renglones de una tabla. Este requerimiento se conoce como Principio Básico del Modelo Relacional.

2. Regla de Acceso Garantizado: Todos los valores atómicos en una base de datos son lógicamente direccionables al especificar el nombre de la relación, el nombre de la columna y un valor de la llave primaria.

3. Tratamiento Sistemático de Valores Nulos: Los valores nulos (diferente de cadena vacía, cadena en blanco, número cero o cualquier otro valor) los soporta un RDBMS para representar falta de información, información no aplicable o información no disponible, haciéndolo independiente del tipo de datos.

4. Catálogo Dinámico en Línea Basado en el Modelo Relacional: La descripción de la base de datos es representada a nivel lógico de forma que usuarios autorizados puedan realizar consultas en un lenguaje natural.

5. Regla de Sublenguaje Comprensivo de Datos: El sistema debe soportar por lo menos un leguaje relacional con la siguientes características:

i) Tener una sintaxis lineal.

ii) Poder ser usado de forma interactiva o por programas de aplicación.

iii) Soportar operaciones de definición de datos (incluyendo definición de vistas), operaciones de manipulación de datos (actualización y recuperación), seguridad y restricciones de integridad, y operaciones de manejo de transacciones.

[21] Diseño y Gestión de Sistemas de Bases de Datos, Autor Angel Lucas, de Paraninfo. Pag. 83-84,
[22] An Introduction To Database Systems, Volume I, Autor C. J. Date, de Addison – Wesley Publishing. Pag. 391 – 393.

6. Regla de Actualización de Vistas: Todas las vistas teóricamente actualizables deben ser actualizables por el sistema.

7. Alto Nivel de Inserción, Actualización y Borrado: La capacidad de manejar una relación base o derivada con un operador simple no sólo aplica a la recuperación de datos sino también a la inserción, actualización y borrado.

8. Independencia Física de Datos: Los datos no dependen de la forma de almacenamiento físico.

9. Independencia Lógica de Datos: Cada usuario podrá visualizar los datos según sean sus necesidades. Las aplicaciones permanecen lógicamente intactas al ofrecer cambios en tablas bases, al ofrecer vistas restringidas de esas tablas a las aplicaciones o usuarios.

10. Independencia de Integridad: Las restricciones de integridad deben ser especificadas separadamente de los programas de aplicación y almacenadas en un catálogo, teniendo acceso a estas a través de un sublenguaje de datos relacional.

11. Independencia de Distribución: Para el usuario debe ser transparente la distribución de la base de datos, incluso si esta se encuentra para una misma base de datos en diferentes lugares físicos.

12. Regla de no Subversión: Si un sistema tiene un lenguaje de bajo nivel (un registro a la vez), el bajo nivel no puede utilizarse para eliminar las reglas de integridad y seguridad expresadas en el lenguaje de nivel superior (múltiples registros a la vez).

# Diagramas de Entidad/Relación (E/R)

Los términos manejados en los diagramas E/R son similares a los conceptos de la teoría relacional, el modelo E/R representa al mundo real por conjuntos de objetos lógicos, llamados entidades, los cuales pueden unirse a través de relaciones. Los diagramas de E/R ayudan a mostrar de forma gráfica las relaciones (tablas) y sus ligas, pero denominando a las relaciones (relaciones de datos) como entidades y a sus ligas como relaciones (relaciones de tablas).

Los diagramas de E/R permiten comprender en un lenguaje cotidiano la relación entre las entidades componentes del modelo. Las estructuras gramaticales nombran los objetos pertenecientes al diagrama de E/R ocupando sujetos, adjetivos y verbos. La representación gráfica de las entidades y relaciones nos generan un panorama claro de la forma de ligarse nuestro modelo.

Una situación o problema, puede representarse mediante estructuras gráficas para analizar visualmente su información. Al tener una situación esta podrá ser plasmada en un modelo gráfico y en algunos casos podrá ser un diagrama de entidad - relación. Una entidad es una persona, lugar, cosa, concepto o suceso

del cual es necesario guardar información, es decir, la entidad es un sujeto. Se recomienda que los nombres de las entidades se escriban en singular.

Un atributo es un adjetivo, es decir, es la palabra encargada de calificar a una entidad o sujeto.

Una relación representa la conexión, liga o asociación entre entidades en forma bidireccional, en donde toda relación se nombra con un verbo.

La siguiente gráfica muestra los nombres a asignar a los objetos del diagrama de E/R, dependiendo de las estructuras gramaticales:

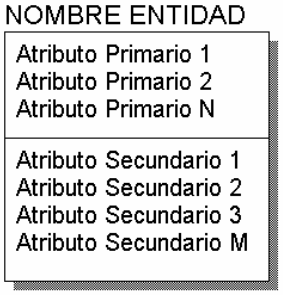
Situación→Modelo

Sujeto→Entidad

Adjetivo→Atributo

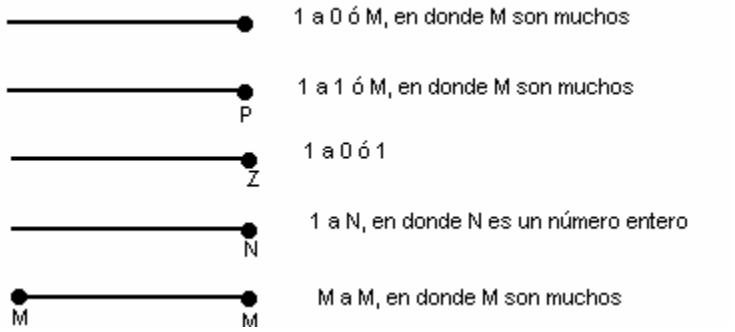
Verbo→Relación

Una entidad está calificada por atributos y su representación gráfica es la siguiente:



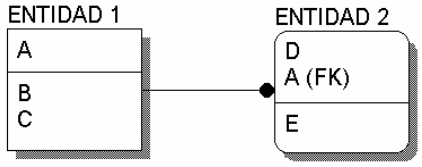

Las llaves foráneas se indican en seguida del nombre del atributo con (FK).

La cardinalidad de una relación indica como se conectan dos entidades entre sí en función de sus registros. Existen cuatro tipos básicos de cardinalidades y se leen de la siguiente forma:

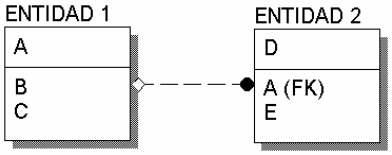


Las relaciones se dividen en identificadoras y no identificadoras. Las relaciones identificadoras indican una llave foránea en una entidad en su zona de atributos primarios, mientras las no identificadoras indican una llave foránea en una entidad en su zona de atributos secundarios.

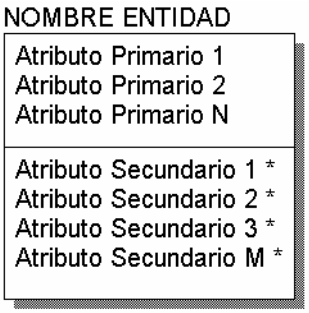
Relación identificadora.




Relación no identificadora.

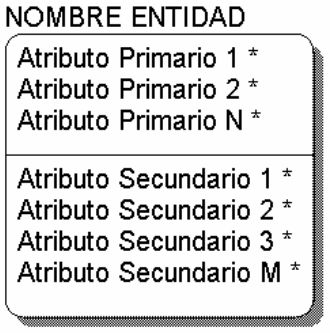



Una entidad independiente no necesita de ninguna otra entidad en el modelo para su identificación, por lo tanto, su llave primaria no requiere para su conformación de ninguna llave foránea, pero sus atributos secundarios puedieran ser llaves foráneas. Una entidad independiente se representa como una caja con esquinas rectas.



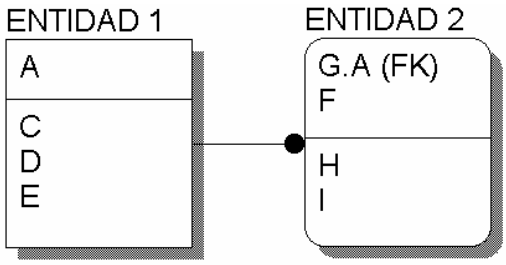

* Los atributos marcados con asterisco pueden ser llaves foráneas.

Una entidad dependiente necesita de alguna entidad del modelo para poder identificarse, es decir, su llave primaria se compone al menos por una llave foránea. Una entidad dependiente se representa como una caja con esquinas redondeadas.

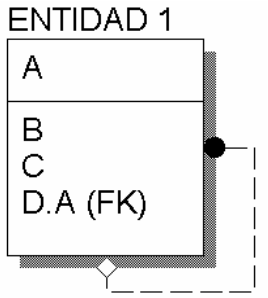



* Los atributos marcados con asterisco pueden ser llaves foráneas.

Un nombre indicativo (rolename) es el nombre nuevo de una llave foránea en una entidad, con la finalidad de representar su función dentro de la entidad. En el siguiente ejemplo G.A representa un rolename.




Una entidad también podría relacionarse a ella misma, es decir, tener una relación recursiva.




# Diseño de una Base de Datos

Después de revisar las herramientas y teoría de las bases de datos relacionales es necesario proponer una metodología de diseño de base de datos, pero esto no implica ser el único camino.

Para poder diseñar una base de datos se recomienda:

1. Tener una descripción clara y completa de cada función específica del sistema

2. Buscar todos los sustantivos de interés de la situación.

3. Encontrar los sustantivos relacionados, generalmente será a través de verbos.

4. Buscar todos los adjetivos de interés de los sustantivos encontrados.

5. Realizar el análisis de atributos.

6. Seleccionar llaves primarias de las entidades, dando mayor peso para la elección a los atributos con características de claves y baja posibilidad de cambio.

7. Para los atributos foráneos, se puede crear un atributo nuevo para representar una clave del atributo, evitando cadenas largas o valores no propicios al momento de realizar búsquedas cuando se relacionan dos entidades.

8. Graficar el diagrama de E/R en donde los sustantivos se transforman en entidades, los adjetivos en atributos y los verbos en sus relaciones. Se obtiene el diagrama E/R sin normalización.

9. Generar para cada entidad localizada todas las dependencias funcionales existentes entre sus atributos.

10. Seleccionar llaves primarias de las entidades, dando mayor peso para la elección a los atributos con características de claves y baja posibilidad de cambio.

11. Graficar las dependencias funcionales.

12. Normalizar las gráficas de dependencia funcional de cada una de las entidades.

13. Transformar las gráficas de dependencias funcionales normalizadas a un diagrama de E/R, indicando las llaves primarias y foráneas.

En algunas ocasiones un atributo contiene una alta incidencia de valores nulos, en este caso podemos desprender este atributo a otra entidad viajando con su llave, y solamente se insertaran los registros con valor no nulo de este atributo. La mayor parte de los manejadores asignan el espacio correspondiente de un registro de manera secuencial y reservan el máximo del registro no importando sus valores para poder seguir manteniendo el registro de manera secuencial después de posibles actualizaciones. Si desprendemos un atributo nulo evitaremos tener espacio desperdiciado y buscar información en regiones más pequeñas del disco duro, pero obviamente también puede afectar en tiempo de búsqueda al tener información en dos entidades en vez de una, lo importante es valorar ambas opciones para encontrar la más eficiente para nuestras necesidades.

También por cuestiones de acceso, revisando el funcionamiento del manejador de base de datos para nuestra implantación, si tenemos entidades excesivamente grandes es necesario evaluar la

segmentación de atributos de la entidad considerando algún criterio de acceso a los campos como: campos de consultas excesivas o pocas consultas, acceso de información en base a perfiles, etc.

Como podemos ver después de finalizar nuestra normalización del modelo, siempre será necesario reevaluarlo en este momento o durante la fase de pruebas para obtener una base de datos más útil, sin tratar de degradar la normalización realizada, y en base a las características del manejador seleccionado la decisión de diseño puede variar drásticamente.

# Problema entre RDBMS y Orientación a Objetos

El Análisis Orientado a Objetos no maneja en forma natural las tablas de las bases de datos relacionales. Existen diversos problemas para utilizar el modelo relacional en el análisis orientado a objetos:

1. El Análisis Orientado a Objetos maneja objetos, mientras el modelo relacional maneja tablas. A este problema se le conoce como impedancia.

2. Pocas partes del sistema deben de conocer la interface del RDBMS, pero la impedancia crea un fuerte acoplamiento entre la aplicación y el RDBMS.

3. Las bases de datos no se representan en términos de herencia.

Las tablas pueden convertirse en objetos, pero antes deben de estar normalizadas. Obedeciendo la siguiente secuencia se puede conseguir:

1. Asignar a cada tabla una nueva clase.

2. Cada atributo de la clase tiene una columna en la tabla.

3. La llave primaria es el único identificador de la instancia.

4. Cada instancia de la clase se representa por un renglón en la tabla.

5. La relación de la llave primaria y foránea se representa con la asociación de agregación.

# CAPÍTULO III. LENGUAJE DE MODELADO UNIFICADO (UML)

## Introducción

Una imagen dice más que mil palabras, una frase muy hecha, pero con una gran verdad, por su naturaleza los diagramas y gráficas dentro del desarrollo de un sistema de cómputo permiten plasmar sus flujos, almacenamiento y funcionalidad.

Durante la evolución de la tecnología orientada a objetos, diversos autores han participado en la aportación de conceptos y, técnicas de análisis y diseño. Dentro de las técnicas de análisis y diseño también se han propuesto diferentes elementos para representar su resultado apoyado en elementos gráficos. En este ámbito existe una premisa primordial, el ofrecer diagramas útiles, tanto para el desarrollador y el usuario, así evitando la duplicidad de diagramas, además de reducir tiempos durante el desarrollo se garantiza la actualización de documentación con mayor eficacia en el proceso de mantenimiento.

Sin duda alguna, de los autores más reconocidos en la materia de objetos, tres de ellos se reunieron para ofrecer una metodología de desarrollo (Los Tres Amigos: Grady Booch, Ivar Jacobson y James Rumbaugh), y un estándar gráfico. Este estándar gráfico es el resultado de unir conceptos de la orientación a objetos, tomando lo mejor de varios autores. Este estándar gráfico se conoce como UML (Unified Modeling Language).

UML muestra a través de un modelo diferentes perspectivas de una situación integrado por diferentes diagramas.

Algunas de las principales ventajas de utilizar UML son:

- Estándar aceptado mundialmente
- Soporta todo el ciclo de vida de un desarrollo
- Soporta la representación de distintos tipos de sistemas, incluyendo a los informáticos
- Es legible para usuarios y desarrolladores
- Lenguaje visual de modelación simple y expresivo
- Puede abarcar con precisión y de manera completa el modelo del sistema
- Es independiente y adaptable a la programación

- Permite visualizar, especificar, construir y documentar al mismo tiempo.
- Muestra el modelo estático y dinámico de un sistema.

Nunca se debe confundir a UML con un proceso o metodología para el desarrollo, UML es un lenguaje de modelado, es decir, nos ayuda a representar el resultado del análisis y diseño, sin ofrecer un procedimiento para lograr un análisis o diseño. Finalmente podemos decir que UML es un lenguaje, y como todo lenguaje tiene reglas para ser expresado de manera correcta.

UML está basado en tres tipos de bloques para modelar:

- Elementos
  - Elementos Estructurales (Clases, Interfaces, Casos de Uso, Actores, Componentes y Nodos)
  - Elementos de Comportamiento (Mensajes y Estados)
  - Elementos de Agrupación (Paquetes)
  - Elementos de Anotación (Notas)
- Relaciones
  - Asociación
  - Generalización
  - Dependencia
  - Realización
- Diagramas
  - Diagramas de Clases
  - Diagramas de Objetos
  - Diagramas de Casos de Uso
  - Diagrama de Secuencia
  - Diagrama de Colaboración

- Diagrama de Estados
- Diagrama de Actividades
- Diagrama de Componentes
- Diagramas de Distribución

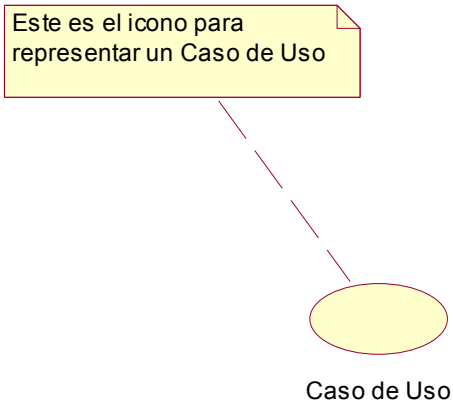
# Mecanismos de Extensibilidad

Los mecanismos de extensibilidad apoyan para ofrecer una mayor potencia a UML, extendiendo y/o personalizando UML según las circunstancias requeridas. Esto sin duda abre las posibilidades de representación de cualquier modelo con circunstancias poco comunes, así como complejas. Existen cuatro tipos:

- Notas
- Estereotipos
- Etiquetas
- Restricciones

## *Notas*

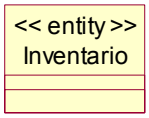
Las notas plasman información adicional de cualquier índole. Se representa con un rectángulo con la esquina superior derecha doblada. En el interior del rectángulo se tiene un área libre para expresar lo necesario. Las notas se pueden ligar a algún elemento a través de una línea discontinua.



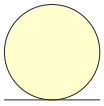
El contenido de la nota no está restringido, puede ser utilizado para realizar observaciones, aclaraciones, advertencias, referencias, etc.

## *Estereotipos*

Los estereotipos ayudan a personalizar los elementos de UML para dar un sentido de mayor precisión. Un estereotipo se representa como una descripción contenida entre dos signos de menor al principio y dos signos de mayor al final y se asocia a cualquier elemento o parte de algún elemento. Algunos estereotipos podrán evolucionar para tener iconos propios.



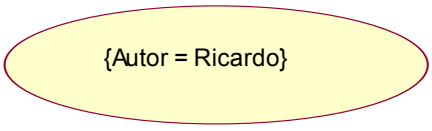

En la figura anterior se muestra una clase del *inventario* físico de una empresa, esta clase almacenará datos por eso se denomina con un estereotipo <<*entity*>> para indicar la naturaleza de la clase de tipo entidad de datos, pero también algunos estereotipos evolucionan a iconos y este es el caso del siguiente, representando exactamente los mismo una clase de entidad de datos.



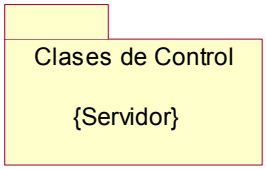
Inventario

## *Etiquetas*

Las etiquetas adicionan nuevas propiedades a los elementos de UML. Se representa por una leyenda contenida entre llaves, la etiqueta lleva un nombre seguido del signo igual más el valor de la etiqueta, o bien, solamente el nombre.



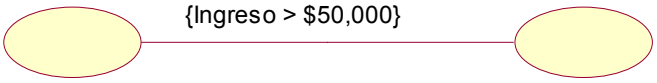

Catálogo de Ciudades



En la figura anterior la etiqueta adiciona la propiedad de *autor* a un caso de uso para especificar el analista de cada caso de uso. Al paquete se le adiciona la propiedad de especificar donde radicará, en este caso las *clases de control* radicarán en un servidor.

## *Restricciones*

Una restricción refiere al condicionamiento aplicado sobre algún elemento de UML. Se representa por una leyenda condicionada entre llaves a manera de texto libre o condiciones lógicas.

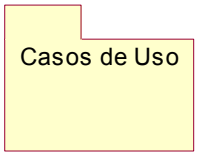



Captura de Sueldo  Captura de Sueldos Especiales

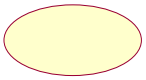
Se muestra una relación entres dos casos de uso, esta relación sólo aplicará si el ingreso es superior a los $50,000.

# Paquetes

Los paquetes son un elemento organizacional aplicable a cualquier elemento de UML, incluso a los mismos paquetes, para referenciar la agrupación de elementos dentro de un paquete. La representación de un paquete es a través del icono de un fólder y su nombre se sitúa en la pestaña del fólder o el cuerpo del fólder.




Para indicar el nombre de un elemento contenido en un paquete, se puede referenciar al nombre del paquete seguido de dos veces dos puntos y posteriormente el nombre del elemento, incluso pueden existir n niveles de anidamiento de paquetes.



Casos de Uso :: Captura de Sueldo

El caso de uso *de captura de sueldo* indica que está contenido dentro del paquete *de casos de uso*.

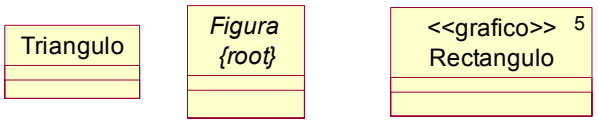
# Clases y Objetos

Las clases gráficamente se representan por un rectángulo dividido hasta por cuatro secciones horizontales. La primera sección contiene información general, la segunda sección indica atributos, la tercera refiere a los métodos y la cuarta menciona la descripción de responsabilidades a manera de leyenda.

## *Sección General*

La expresión gráfica mínima de una clase es la sección general acompañada del nombre de la clase, cualquier sección y elemento diferente a estos siempre serán opcionales para mostrarlos, más aún, dependiendo del ámbito de interés en ese momento.

En la sección general se indica el nombre de la clase. Si el nombre está escrito en letra cursiva representa una clase abstracta. Si la clase no tiene padres se le pone el valor etiquetado de root. Cuando para una clase es necesario limitar la multiplicidad, es decir, limitar el número de instancias existentes para una clase se especifica un número en la parte superior derecha.




La clase *Triángulo* es una clase sin ninguna indicación adicional, la clase *Figura* es una clase abstracta por estar en letras cursivas además contiene el valor etiquetado de *{root}* indicando que no tiene ninguna clase por padre y la clase *Rectángulo* define una multiplicidad para no más de cinco instancias. Adicionalmente arriba del nombre de la clase se puede especificar un estereotipo, para la clase rectángulo se tiene un estereotipo definido como una clase gráfica.

## *Sección de Atributos*

La sección de atributos indica los atributos de una clase, teniendo cero o n atributos, independientemente de si se decide mostrar una parcialidad de atributos o la totalidad. El mínimo requerido para indicar un atributo es la mención del nombre. La sintaxis para especificar un atributo es:

[visibilidad] nombre_atributo [multiplicidad] [:tipo] [= valor_inicial] [{propiedades}]

La visibilidad aplica para atributos y métodos informando si el atributo o método puede ser visto por otras clases:

- Público (+): Cualquier clase con visibilidad a la clase definida es capaz de acceder el atributo o método

Protegido (#): Los atributos o métodos sólo son accesibles para los hijos de la clase definida

Privado (-): Los atributos o métodos sólo los puede acceder la misma clase

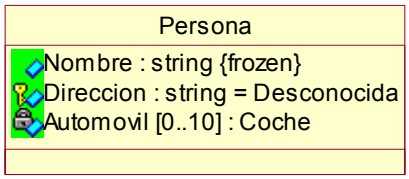
La multiplicidad es la representación análoga con la estructura de programación de arreglos, es decir una matriz de una dimensión para almacenar valores. Sus variantes de especificación son: $[n_1..n_2]$ o [n].

El valor inicial será el valor por default obtenido por el atributo al crearse un objeto de la clase.

El tipo hará mención al tipo de dato o clase representada por el atributo.

Dentro de las propiedades para los atributos tenemos:

- changeable: Sin restricción para modificar el atributo, aunque se considera el default
- frozen: El atributo no puede modificarse tras inicializar el objeto
- addOnly: Aplica para atributos con multiplicidad mayor a uno, sólo se permite adicionar atributos, pero no se pueden eliminar ni modificar.




La representación de la clase *Persona* muestra el atributo público *nombre* de tipo *string* con una propiedad *frozen* para indicar que no puede modificarse el valor del *nombre* después de inicializar una clase. El atributo protegido *dirección* también es de tipo *string* con un valor inicializado de *Desconocida*. El atributo privado *automóvil* inidica una multiplicidad de hasta once automóviles siendo cada uno de estos del tipo de dato *coche.*

## *Sección de Métodos*

La sección de métodos indica los métodos de una clase, teniendo cero o n métodos, independientemente de si se decide mostrar una parcialidad de métodos o la totalidad. El mínimo requerido para indicar un método es la mención del nombre. La sintaxis para especificar un método es:

[visibilidad] nombre_metodo [(lista de parámetros)] [:tipo] [{propiedades}]

La visibilidad es la misma definida para los atributos.

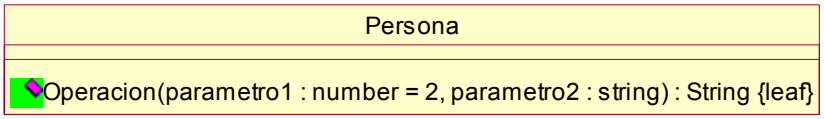
La lista de parámetros, es una lista de parámetros de entrada y/o salida al método, separando el nombre del parámetro por comas y pudiendo acompañar al nombre del parámetro el tipo de dato separado por dos puntos y enseguida un signo igual con valor por default del parámetro.

El tipo indica el tipo de dato o clase retornado por el método.

Dentro de las propiedades para los métodos tenemos:

- leaf: El método no puede ser redefinido por sus hijos
- isQuery: La ejecución del método no afecta el estado del sistema
- sequential: Si existen hilos de control debe ser invocado secuencialmente
- guarded: El método se ejecuta secuencialmente, aún si es invocado por diferentes hilos de control al mismo tiempo
- concurrent: El método puede ser ejecutado concurrentemente por diferentes hilos

de control




La representación de la clase *Persona,* muestra el método público *operacion* con dos parámetros de entrada, el primero de tipo number con un valor por default de dos y el segundo parámetro de tipo string, el método retorno un tipo de dato string y menciona a través de la propiedad *leaf* la limitante de no poder ser modificado por sus descendientes.

# *Clases de Interface, Control y Entidad*

Los sistemas han tendido a segmentarse en sistemas de tres capas:

- Capa de Interface
- Capa de Reglas de Negocio
- Capa de Datos

Aunque estas capas llegan a ser subdivididas siguen respetando la filosofía de su capa origen.

La capa de interface es la responsable de agrupar la funcionalidad correspondiente al manejo de la interface sistema – usuario primordialmente, en esta capa se provee de la funcionalidad para permitir la comunicación entre ambas partes, sin confundirse con la comunicación interna de un mismo sistema, típicamente esta capa es la encargada de manejar las pantallas para los usuarios con sus eventos necesarios, ocupándose de entender, validar y disparar las acciones ejercidas por el usuario, siempre y cuando no intervenga en decisiones y acciones correspondientes a las reglas del negocio. Un ejemplo de una interface es un reporte físico, una ventana gráfica o algún dispositivo de hardware representante de una interacción directa con un usuario, pero también lo puede ser un componente visual agregado a una ventana gráfica, en general, cualquier artefacto con interacción directa con el usuario.

La capa de datos es la responsable de controlar el almacenamiento y extracción de la información almacenadas en medios físicos, típicamente tablas de bases de datos y archivos. En esta capa tiene la especialización de controlar el manejo de datos y todo lo relacionado a ellos como la integridad, seguridad, etc.

La capa de reglas de negocio obedece a respetar, cumplir y accionar la manera de ejercer las reglas de negocio definidas para un sistema, se puede definir como el núcleo del sistema basado en que esta capa ofrecerá el funcionamiento esperado por el usuario. La capa de las reglas de negocio une a la de interface y datos.

Un flujo típico en un sistema de tres capas sería:

1. Capa de Interface
    a. Restringe acciones del usuario para seleccionar eventos válidos
    b. Valida información ingresada por el usuario
    c. Notifica a Capa de Reglas de Negocio solicitud de usuario
2. Capa de Reglas de Negocio
    a. Verifica perfil de seguridad de usuario
    b. Procesa regla de negocio
        i. Solicita extracción de información a Capa de Datos
3. Capa de Datos
    a. Verifica perfil de seguridad de la aplicación

b. Realiza extracción de datos

c. Devuelve extracción de datos a Capa de Reglas de Negocio

4. Capa de Reglas de Negocio

a. Continua con proceso de regla de negocio

b. Devuelve extracción de información formateada a Capa de Interface

5. Capa de Interface

a. Recibe información para presentar al usuario y despliega en pantalla.

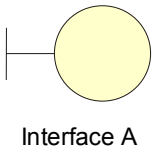
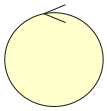
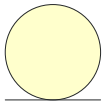
Con el sistema de tres capas se vuelve altamente útil el reutilizar código y el dar mantenimiento por manejar una independencia lógica entre cada una de las capas dependiendo del propósito de la funcionalidad. También ayuda a los diseñadores a dividir el problema de manera rápida en tres y por lo tanto permite un mantenimiento al código más sencillo.

Dentro de las clases existen iconos diferentes al rectángulo para representar una clase derivado de su propósito. Estos iconos ayudan a visualizar y entender gráficamente el tipo de funcionalidad aportada por una clase dentro del sistema y apoyan a la representación de las tres capas:

- Capa de Interface / Clases de Interface
- Capa de Reglas de Negocio / Clases de Control
- Capa de Datos / Clases de Entidad

Estos tipos de clases no son clases diferentes, simplemente son una representación diferente para indicar un propósito en su funcionalidad y mantendrán todas las características de una clase, así como su manera de relacionarse.

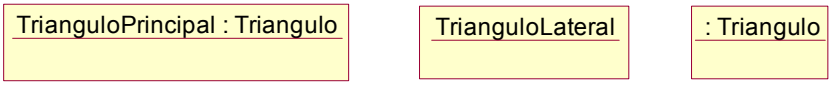
Los estereotipos para estos tipos de clases son:

| Tipo de Clase | Estereotipo Simple | Estereotipo Evolucionado |
|---|---|---|
| Clase de Interface | << interface >> Interface A |  Interface A |
| Clase de Control | << control >> Control B |  Control B |
| Clase de Entidad | << entity >> Entidad C |  Entidad C |

Se puede utilizar cualquier clase con estereotipo, pero resulta más gráfico utilizar las de estereotipo evolucionado.

## *Objeto*

Los objetos finalmente son una instanciación de una clase y por ende toman la misma forma de una clase pero adicionando el nombre del objeto. El nombre del objeto se separa del nombre de la clase por dos puntos, además de subrayarse, las variantes soportada son indicar: sólo el nombre del objeto, sólo el nombre de la clase (objeto anónimo) o indicar ambos nombres.

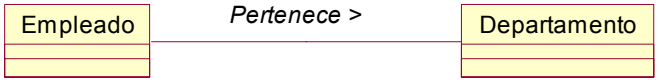



# Relaciones

Las relaciones se ayudan a ligar diferentes elementos estructurales, aunque las relaciones serán ejemplificados primordialmente con clases, puede aplicar también para otros elementos.

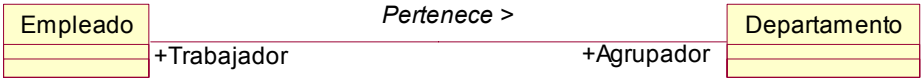
## *Asociación*

La asociación se representa por una línea continua para conectar dos elementos estructurales. La asociación puede tener un nombre para describir su función y acompañarse de una flecha al final del nombre para indicar el sentido de lectura del nombre de la asociación.



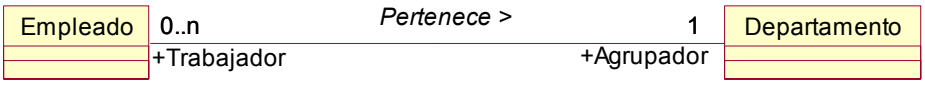

En el ejemplo de arriba se muestra la relación existente entre un empleado y un departamento, el nombre de la relación avisa de que un empleado pertenece a un departamento.

La asociación puede especificar el rol de cada elemento estructural dentro de la relación, situándose el rol en el extremo de la asociación y precediéndose de un signo de +.



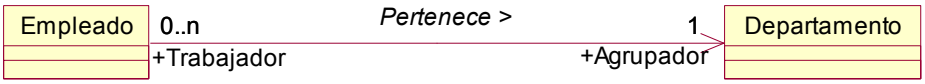

Un empleado juega el papel de ser trabajador y el departamento el rol de agrupar a los empleados.

La multiplicidad de una asociación se representa a través de un rango en el extremo de uno de los elementos estructurales y especifica que para cada elemento estructural del extremo contrario a donde se indica la multiplicidad le corresponde ese múltiplo de elementos estructurales.




La multiplicidad de la relación anterior menciona que para un departamento existen cero a n número de empleados.

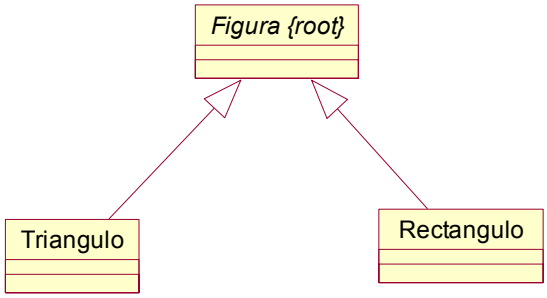
La asociación también puede indicar la navegación a través de una flecha en el extremo de la línea de asociación, insinuando la navegación principal más no limitando a ser el único sentido de la navegación.




Para este caso la navegación principal será buscar la descripción del departamento de un empleado.

## *Generalización*

La relación de generalización o herencia se plasma utilizando una línea dirigida continua con una flecha en forma de triángulo vacía, señalando la flecha al padre de la relación. Aunque se permite dar un nombre a la generalización es extremadamente raro realizarlo.




El diagrama plasma que *Figura* es el padre de *Triangulo* y *Rectangulo* y hereda a sus hijos sus atributos y métodos.

## *Agregación*

La agregación es una variante de la asociación para modelar una relación de total / parte, tiene su representación a través de un rombo vacío del lado más cercano al representante de la totalidad. Una agregación no limita el ciclo de vida de sus componentes a la existencia de la totalidad, es decir, sus componentes pueden existir aún si la totalidad se destruye.

La composición es una variante de la agregación para limitar el ciclo de vida de las partes, en donde la destrucción de la totalidad finaliza con la vida de las partes. La representación de la composición es similar a la de la agregación, exceptuando el rombo el cual no se indica vacío.




Los empleados pertenecen a un departamento el cual es un agregado de los empleados, si el departamento deja de existir los empleados seguirán viviendo. Una empresa se compone de departamentos formando una composición, si la empresa deja de existir los departamentos dejarán de existir también.

## *Dependencia*

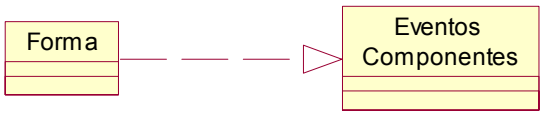
La dependencia indica una relación en donde un cambio de un elemento afecta a otro elemento. La dependencia se muestra con una línea discontinua señalando con una flecha el elemento del cual se depende, incluso puede asignarse un nombre, aunque es poco frecuente hacerlo. Las dependencias son más útiles al apoyarse de los estereotipos, siendo más común su utilización en los casos de uso. Entre paquetes pueden marcarse dependencias.




En este caso el paquete *de reglas de negocio* depende del paquete de *acceso a base de datos*, un cambio en los elementos de *acceso a base de datos* afecta o puede afectar al paquete *reglas de negocio.*

## *Realización*

Es la relación para garantizar el cumplimiento de un contrato por un elemento estructural. Se representa por una línea discontinua con una flecha en forma de triángulo vacía. Se utiliza primordialmente para señalar la relación entre dos clases, una de ellas de tipo interface de métodos (clase únicamente con declaración de métodos).



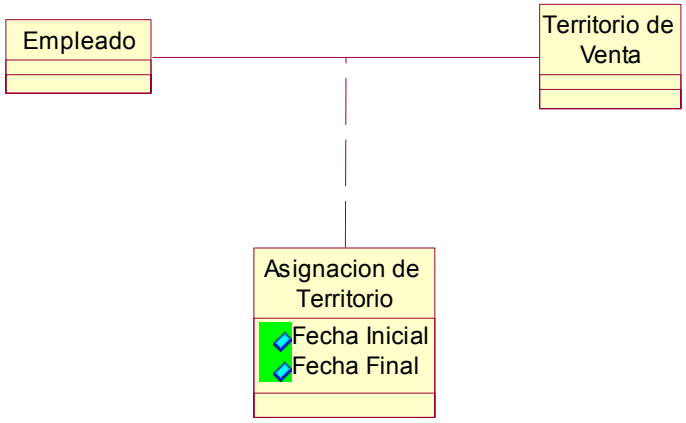

La clase *forma* indica que tiene una interface *eventos componentes* para cumplir su contrato de responder a eventos. La clase *eventos componentes,* puede ser una interface de métodos reutilizable para responder a diferentes contratos de eventos de diversos componentes.

Otra manera de representar la realización es de la siguiente manera:



## *Clase de Asociación*

La clase de asociación es una asociación entre dos clases pero con propiedades propias y se representa asociando las dos clases con una línea continua como en cualquier asociación y uniendo una tercera clase a la asociación con una línea discontinua.




Un *empleado* puede tener asociación con un *territorio de venta*, pero dentro de esta relación resulta útil saber la fecha de inicio de la asignación y la fecha final de asignación a uno de los *territorios de venta* para un *empleado,* derivado de tener una asociación con atributos propios para marcar las fechas de asignación se utiliza una clase de asociación.

# Diagramas de Casos de Uso

Un caso de uso es un conjunto de acciones para satisfacer un comportamiento, cada bloque de acciones es disparado por eventos generados por actores para satisfacer alguno de los propósitos del sistema.

Cada caso de uso debe representar un requisito funcional del sistema, y al momento de especificarse describir el comportamiento deseado y no como solucionar ese comportamiento. Por todo anterior, los casos de uso deben ser especificados de manera clara, ordenada y completa de absolutamente todo lo que debe hacer el sistema.

Los casos de uso ayudan a aterrizar los requerimientos del usuario. Un buen analista en varias ocasiones tendrá que guiar al usuario a través de preguntas que colaboren en complementar la información de la interacción entre casos de uso, actores, así como el detalle del mismo. Es habitual pensar que los buenos analistas son aquellos capaces de decidir sobre el funcionamiento, pero tal vez sean mejores analistas, aquellos con la capacidad de preguntar y orientar al usuario para decidir sobre el funcionamiento.

Dentro de los diagramas de casos de uso intervienen los siguientes elementos:

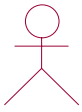
- Elementos Estructurales
    - Casos de Uso
    - Actores
- Relaciones
    - Asociación
    - Generalización
    - Dependencia (Inclusión y Extensión)

Un caso de uso es representado por una elipse, su nombre puede ser localizado en el exterior o interior de la elipse.



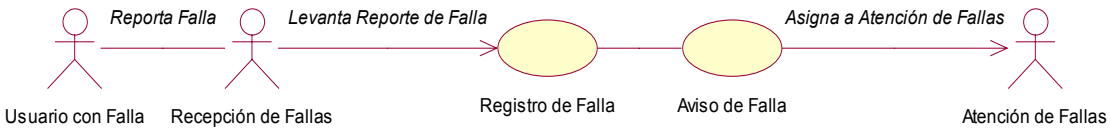
Caso de Uso 1

Un actor es la entidad responsable de utilizar un caso uso. El actor representa un rol dentro del sistema, por ejemplo un conjunto de personas, un sistema informático, una organización, etc. Derivado de que un actor representa un rol, dentro de un rol se pueden agrupar un conjunto de entidades, incluso una entidad puede jugar diferentes roles dentro de un sistema, implicando una entidad ser representada por diferentes actores. El actor no forma parte del sistema, pero se modela para conocer la interacción del sistema con sus usuarios. La representación gráfica de un actor es muñeco con un nombre.



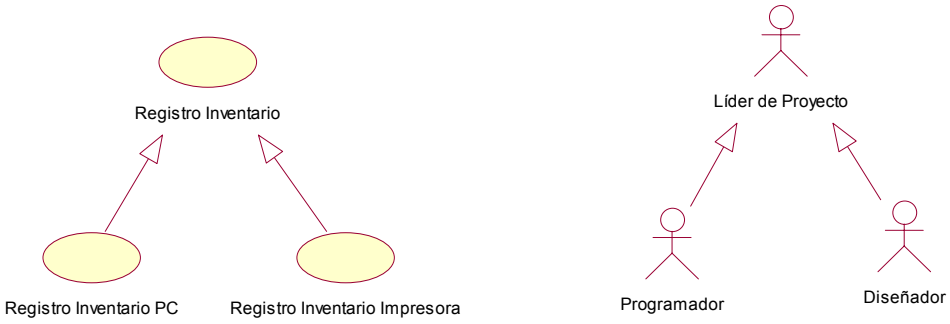
Actor 1

La asociación es la relación entre un actor, un caso de uso, entre casos de uso y entre actores. La asociación entre actores pocas veces se representa por no ser de importancia en la mayor parte de los análisis.



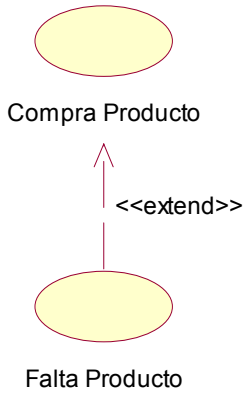

Para mostrar el uso de las asociaciones se muestra un actor *usuario con falla* el cual realiza un reporte telefónico de falla al actor *recepción de fallas*, el cual captura en el sistema el registro de la falla, la falla es enviada al módulo *aviso de fallas* para su asignación vía correo electrónico al actor *atención de fallas*. El actor de *recepción de fallas* se muestra con una asociación direccionada al caso de uso de *registro de falla,* indicando el sentido de la asociación de interés para nosotros, el caso de uso *registro de falla* devolverá un ticket electrónico de seguimiento al actor de *recepción de falla,* visualizando una comunicación inversa a la señalada, pero no se muestra por resultar de menor interés.

La generalización grafica la relación de herencia entre un padre y un hijo, y aplica para casos de uso y actores.



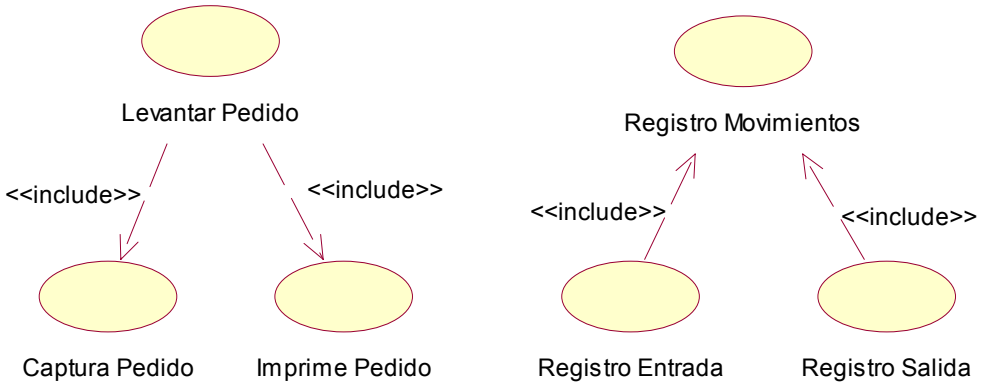

El registro de inventario de equipo de cómputo en una empresa se realiza para PC's e Impresoras, pero para ambas implica el capturar información común y otra parte particular derivado del tipo de hardware. La herencia entre actores nos permite modelar organigramas y perfiles de seguridad, entre otras cosas. Si la herencia modela un organigrama, este debe ser interpretado como las responsabilidades del padre son heredadas a sus hijos. En el ejemplo anterior se entiende que un líder de proyecto hereda actividades asignadas a él a un programador o a un diseñador.

Los casos de uso pueden relacionarse a través de dos tipos de dependencia, la extensión y la inclusión. La extensión nos ayuda a modelar flujos opcionales dentro de un caso de uso, siempre representa un conjunto de acciones, con un nivel de complejidad merecedor de realizarle un análisis independiente, a través de un caso de uso. La extensión se representa como una línea discontinua direccionada hacia el caso de uso integrador del opcional y nombrado con el estereotipo <<extend>>.




El caso de uso de *compra producto* es utilizado para registrar el levantamiento de pedido de un cliente, la mayor parte de las ocasiones todos los productos comprados tienen existencia, pero cuando algún producto hace falta se inicia el caso de uso opcional de *falta producto*, derivado de la opcionalidad se convierte en una relación de extensión.

La inclusión representa la segmentación de un caso de uso para su análisis independiente, derivado de la complejidad o tamaño, o bien cierta funcionalidad común a dos o más casos de uso que merezcan ser considerados como casos de uso. La inclusión se representa como una línea discontinua direccionada al caso de uso que se quiere incluir dentro de la funcionalidad de otro.




Si suponemos que el caso de uso *levantar pedido* tiene una complejidad importante, sobre todo al momento de capturar pedido e imprimir pedido, podríamos dejar parte de la funcionalidad en *levantar pedido* e incluir la funcionalidad de *captura pedido* e *imprime pedido,* pensando en brindar un análisis detallado de estos dos últimos casos de uso.

Para la situación de *registro de entrada* de personal y *registro de salida* de personal, ambos presentan de manera diferente la información, pero ambos requieren registrar el movimiento de una misma tabla de base de datos, esto se puede manejar por el caso de uso de registro de movimientos común a ambos, y ambos lo pueden incluir sin entrar en conflictos. La herencia también hubiera podido aplicar para los casos de uso de registro de movimientos, pero en este caso suponemos diferencias importantes entre ambos, y la similitud aparece exclusivamente en un punto, siendo más conveniente la inclusión.

Dentro de los casos de uso podemos ver que la parte gráfica no informa de los detalles de los casos de uso, es por eso la necesidad de contar con una especificación de casos de uso para detallar su funcionalidad. El formato de especificación de caso de uso propuesto es el siguiente, pero debe ser ajustado para cubrir su propósito: ser legible para el usuario y el programador.

| **PAQUETE** | **CASO DE USO** | OBJETIVO |
|---|---|---|
| | | |

| | |
|---|---|
| Precondiciones | |
| Postcondiciones | |
| Actores | Actor 1: Funciones<br>Actor 2: Funciones |
| Generaliza de | |
| Extiende a | |
| Incluye a | |
| Entradas | |
| Salidas | |
| Dudas | |

| PROCESO/EVENTO | DESCRIPCIÓN |
|---|---|
| P: | |
| E: | |

- ✓ Diseño de Interface (Mostrar el diseño de la interface)
- ✓ Descripción de campos de interface (La captura de campos puede ser adecuada a los propósitos específicos)

| CAMPO | DESCRIPCIÓN | CAMPOS DE CAPTURA | | | | REQUERIDO |
|---|---|---|---|---|---|---|
| | | ALTA | BAJA | CONS. | MOD. | |
| | | | | | | |
| | | | | | | |

*Paquete*: Indica el módulo de pertenencia del caso de uso.

*Caso de Uso*: Nombre del caso de uso.

*Objetivo*: Propósito general y breve del caso de uso sin entrar en precisiones de funcionalidad.

*Precondiciones*: Condiciones previas que deben cumplirse antes de iniciar el caso de uso.

*Postcondiciones*: Condiciones posteriores como resultado de la ejecución del objetivo del caso de uso. Tanto para las precondiciones como para las condiciones sólo se consideran las de relevancia que se deseé controlar, resulta obvio que si no está disponible la base de datos, la red se encuentra caída, el

monitor dejo funcionar, se presenta un maremoto, etc., el caso de uso no podrá ejecutarse, pero también es cierto que son situaciones que no se controlarán a través del sistema. Si por el contrario, el caso de uso emitiera un correo electrónico se tendrá como precondición un estatus de correcta funcionalidad para el envío de correo electrónico. Las precondiciones y postcondiciones ayudan a delimitar el alcance ante contingencias del caso de uso.

*Actores*: Actores con interacción dentro del caso de uso. Es conveniente indicar la función particular del actor dentro del caso de uso.

*Generaliza de*: Nombre del caso de uso del cual generaliza el actualmente descrito.

*Extiende a*: Nombres de los casos de uso a los cuales podría extenderse de manera opcional el actual caso de uso.

*Incluye a*: Nombre de los casos de uso a los cuales se incluyen para poder cumplir con su objetivo el caso de uso actual.

*Entradas*: Indica la información de entrada al caso de uso.

*Salidas*: Indica la información de salida del caso de uso. Para las entradas y salidas es ideal especificar al mayor detalle posible, si no se cuenta con esta información se deberá poner el detalle que se tenga, para posteriormente cuando se encuentre definida esta información completarla.

*Dudas*: Es común el tener dudas al momento de especificar el caso de uso, por eso deben ser apuntadas para aclararlas con el usuario posteriormente sin depender de notas adicionales o la memoria.

*Proceso / Evento*: Los casos de uso son ejecutados por eventos, para resolver a veces los eventos se requiere descomponer en procesos. Para indicar un evento se pone la letra E seguida de dos puntos acompañando luego del nombre del evento, para un proceso se indica la letra P seguida de dos puntos acompañando luego del nombre del proceso.

*Descripción*: Detalla la descripción del evento o proceso. A mayor nivel de detalle y especificación será de mayor utilidad. La descripción debe ser mencionada en términos fácilmente legibles para un usuario sin conocimientos de programación, es habitual realizar erróneamente la descripción en términos de programadores. La descripción nos brinda el detalle del comportamiento de un caso de uso, cualquier punto no mencionado en el caso de uso referente al comportamiento el sistema final no lo incluirá por no haber sido especificado.

*Diseño de Interface*: Si el caso de uso requiere de una interface para el actor se realiza un esquema de la interface.

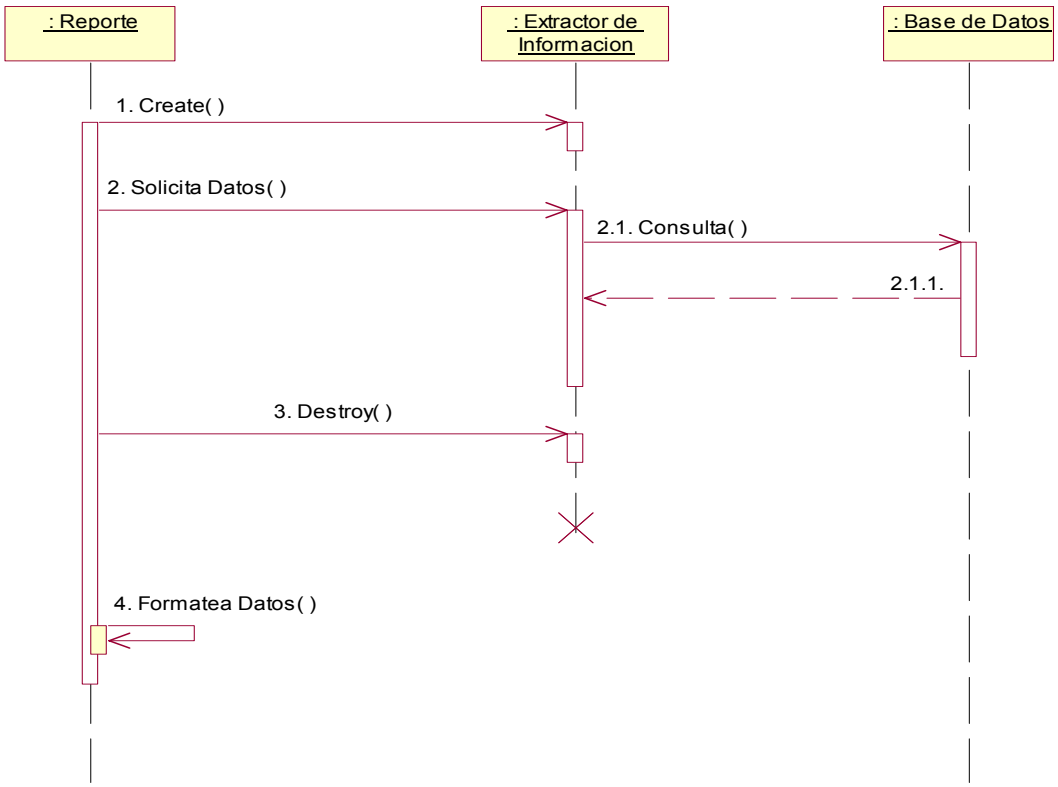
*Descripción de Campos de Interface*: Esta sección puede ser modificada de acuerdo a las necesidades de especificación de la interface, en la mayor parte de los sistemas se cuenta con pantallas de captura, en donde el formato mencionado resulta de utilidad, pero aún así puede ser ajustado.

# Diagrama de Secuencia

Los diagramas de secuencia son útiles para mostrar la relación existente entre objetos a través de mensajes en un tiempo determinado. Muestran la interacción entre objetos priorizando el orden de los mensajes. El diagrama de secuencia se representa situando los objetos en la parte superior horizontal y los mensajes debajo de los objetos en el orden de emisión de arriba hacia abajo. Su principal uso interviene para reflejar el funcionamiento de un caso de uso, pero claro serán usados en otros lugares si se requieren.

Los objetos son ordenados de la manera que generen un diagrama más sencillo de leer. Debajo de cada objeto se dibuja una línea discontinua vertical para representar la línea de vida del objeto. Algunos objetos pueden presentar una línea de vida más allá del diagrama de secuencia, otros tantos iniciarán y concluirán su ciclo de vida dentro del diagrama. Cuando un objeto deja de existir dentro del diagrama se marcará con un X al final de la línea de vida, llegando la línea de vida abajo del último mensaje recibido.

El envío de un mensaje entre objetos se representa uniendo la línea de vida entre ambos con una línea continua dirigida al recepto del mensaje y el nombre del mensaje sobre la línea. En el momento de recibir el mensaje, el objeto receptor también recibe el foco de control. El foco de control se representa con un rectángulo delgado para responder al mensaje recibido, la longitud del rectángulo dependerá de los diferentes mensajes necesarios de enviar a otros objetos o a el mismo para satisfacer el mensaje recibido. La parte superior del rectángulo debe iniciarse al recibir el mensaje, y la parte inferior del rectángulo con su terminación la cual puede retornar si se quisiera, a manera de un mensaje de retorno, representado por una línea discontinua dirigida al emisor original del mensaje. Cuando un mensaje es enviado al mismo objeto, este nuevo foco de control se representa con un rectángulo sobrepuesto y ligeramente a la derecha del foco de control original. En algunas situaciones resultará muy útil numerar los mensajes de acuerdo al nivel de anidamiento.



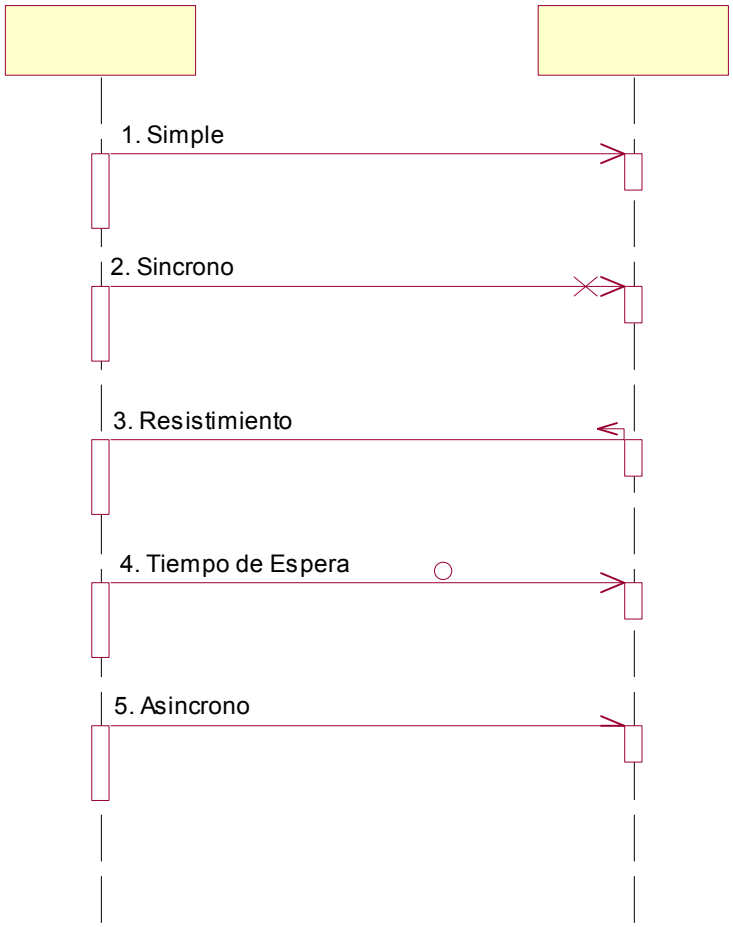

En la parte superior se colocan los objetos, el objeto *Reporte*, *Extractor de Información* y *Base de Datos.* El objeto *Reporte* envía un mensaje de *Create()* al *Extractor de Información* para crear una instancia de este e iniciar su ciclo de vida, al concluir el mensaje regresa el foco de control al *Reporte* para posteriormente enviarle un nuevo mensaje de *Solicita Datos()*, para satisfacer el mensaje el *Extractor de Información* emite un mensaje de *Consulta()* a la *Base de Datos,* siendo un objeto que ya tiene vida, por eso no es necesario crearlo. De manera explícita en este caso se regresa el control indicado por el mensaje *2.1.1*, al recibir nuevamente el control el *Extractor de Información* concluye el foco obtenido y aunque no se ve explícitamente se regresa el control al objeto *Reporte*, para que este ya con los datos le solicite al *Extractor de Información* concluir con su ciclo de vida representado por una X al final del ciclo de vida. Al tener el foco el *Reporte* envía un mensaje así mismo, representado por el mensaje de *Formatea Datos (),* representado por un mensaje recursivo, pero con un rectángulo de vida sobrepuesto sobre el principal.

Los mensajes pueden tomar las siguientes formas:

- Simple: Para mensajes con un solo hilo de control sin especificar sincronía.
- Síncrono: El emisor espera al receptor indefinidamente.

- Resistimiento: El emisor abandona si el receptor no se encuentra preparado al momento de recibir el mensaje.
- Tiempo de Espera: El emisor espera un tiempo especificado antes de abandonar.
- Asíncrono: El emisor envía el mensaje al receptor y continúa sin esperar respuesta el receptor.

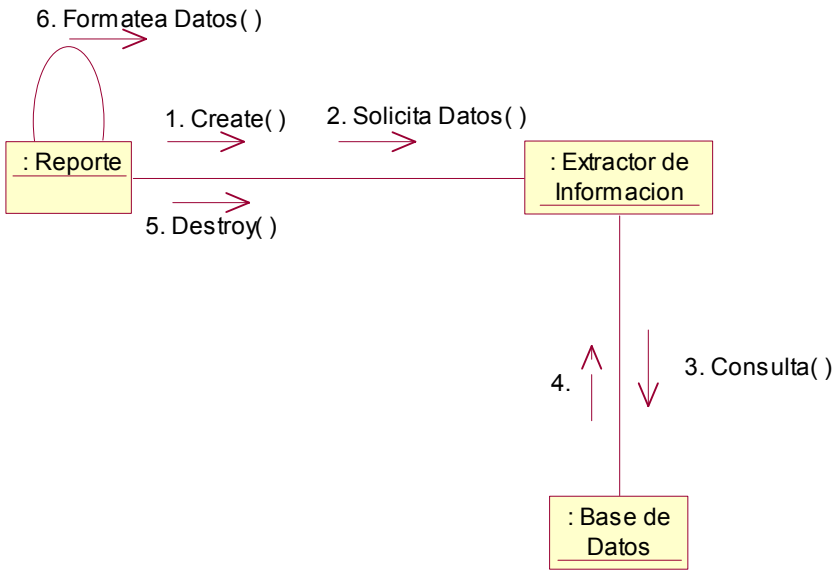
El siguiente diagrama muestra la representación de los estereotipos de cada uno de los mensajes enunciados.




# Diagrama de Colaboración

Los diagramas de colaboración muestran la misma información que un diagrama de secuencia, pero destacan la organización de objetos. Ambos tipos de diagrama son útiles dependiendo de la necesidad. El diagrama de colaboración se representa situando los objetos sin un orden específico dentro del

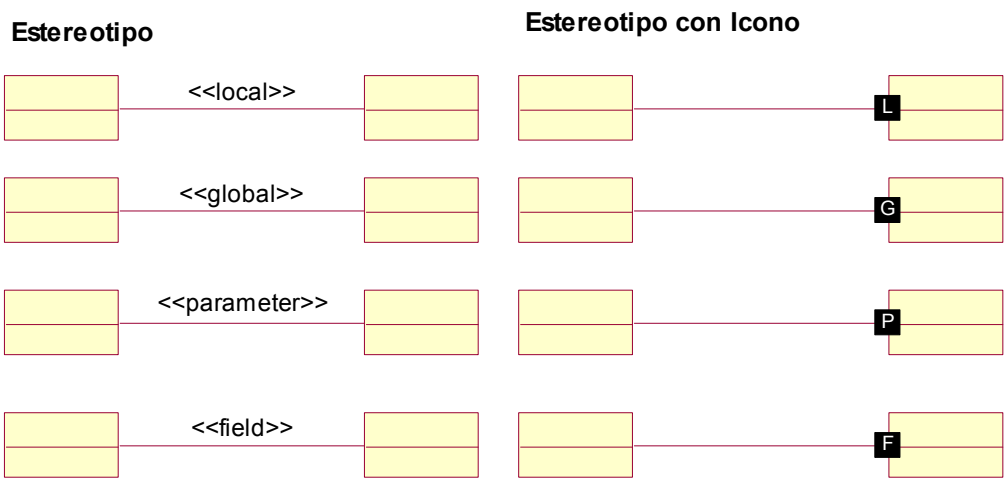
espacio del diagrama, los objetos son unidos por enlaces a través de una línea continua y los mensajes acompañan a los enlaces señalizados de la misma manera a los diagramas de secuencia. Aquí es muy importante numerar los mensajes.




Adicionalmente el enlace puede indicar la visibilidad con un estereotipo, los más comunes son:

- <<local>> El receptor es local al emisor
- <<global>> El receptor es global al emisor
- <<parameter>> El receptor es un parámetro de alguna operación del emisor
- <<field>> El receptor es un atributo del cliente

En el siguiente diagrama se muestran los estereotipos y su correspondiente estereotipo con icono del lado izquierdo:




En los estereotipos con iconos incluso se puede representar fácilmente la visibilidad tanto del emisor al receptor y viceversa.

# Diagrama de Actividades

Los diagramas de actividades tienen el interés en plasmar el flujo existente entre actividades. Cada actividad es una agrupación de subactividades con un propósito definido por el nombre de la actividad que las contiene, incluso las actividades pueden aterrizarse hasta convertirse en instrucciones análogas a sentencias de programación, como siempre dependerá de las necesidades del problema el nivel de detalle de una actividad.

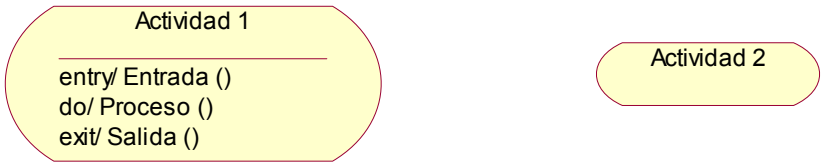
Los diagramas de actividades se componen de:

- Actividades
- Transiciones
- Bifurcaciones
- Concurrencia
- Carriles

## *Actividades*

Las actividades se representan por rectángulos con sus lados laterales redondeados, el nombre de la actividad se posiciona en el interior de la figura. Si una actividad se requiere indicar mayor detalle para visualmente dar mayor información o incluso no elaborar un diagrama adicional para desglosar una

actividad se pueden mencionar sus actividades de entrada, proceso y salida colocando una línea continua bajo el nombre y posteriormente las leyendas de *entry, do, exit*, en los renglones requeridos, incluso puede repetirse el uso de éstas.

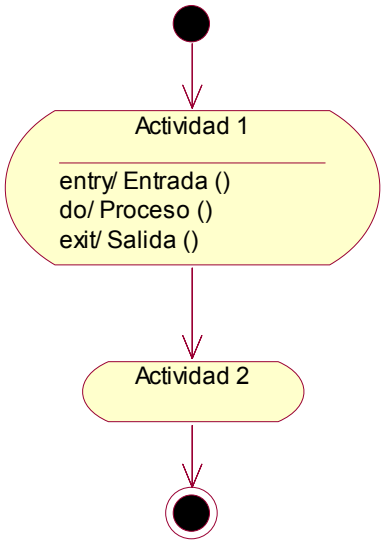



## *Transiciones*

La transición es la culminación de una actividad para el inicio de otra actividad, para indicar la transición entre actividades se utilizan líneas dirigidas uniendo a las actividades. En los diagramas de actividades se representa el inicio y el fin con los iconos:



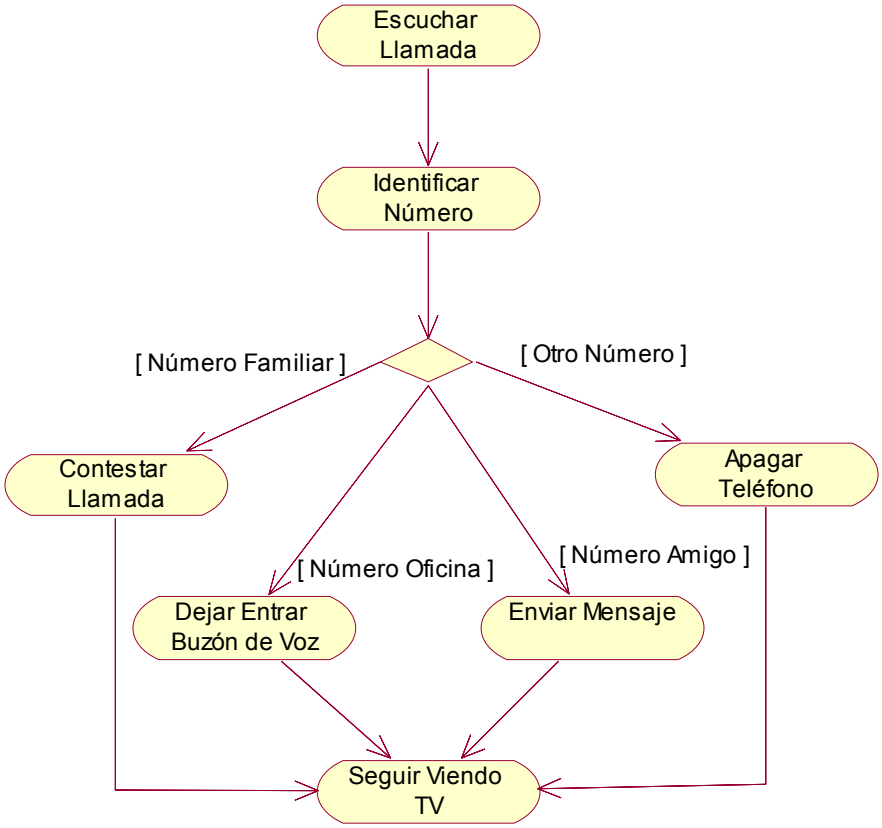

Un diagrama de actividades con la unión de los elementos revisados podría quedar de la siguiente manera:



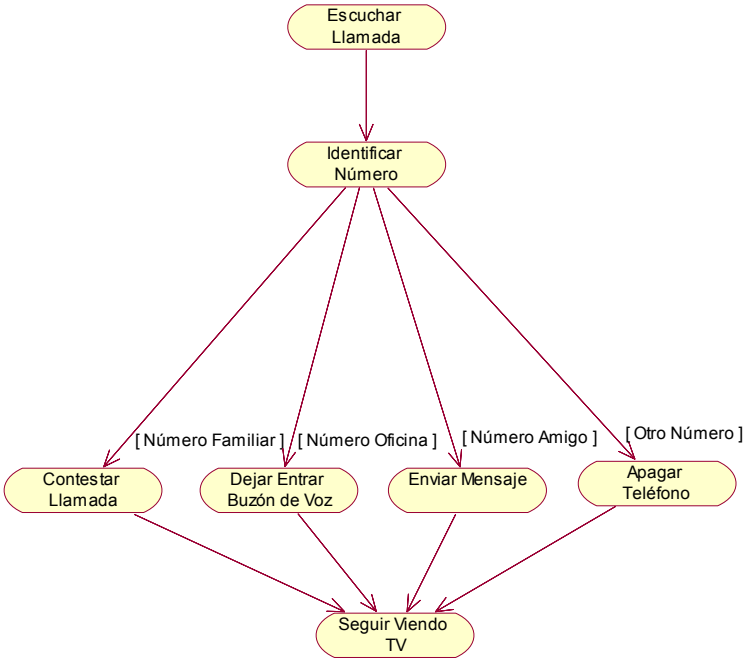
Puede existir más de un inicio y un fin para un mismo diagrama, es útil contar con dos o más finales algunas ocasiones, pero tener dos inicios resultará muy confuso.

## *Bifurcaciones*

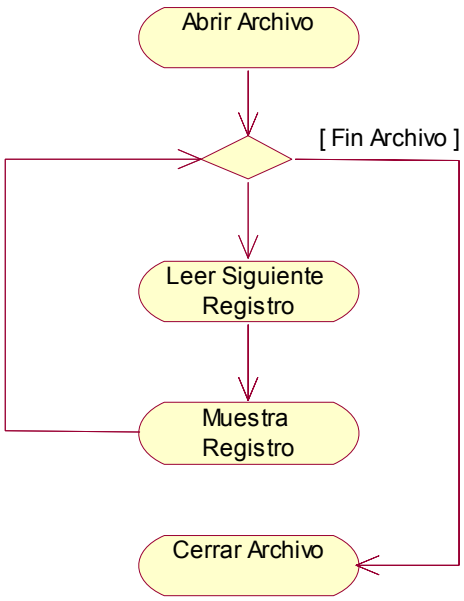
La bifurcación es la toma de decisión para continuar el flujo a través de un camino, teniendo varias opciones. La bifurcación se puede representar a través de un rombo recibiendo una transición y ofreciendo transiciones de salida, cada transición de salida se deberá acompañar con la condición requerida para seleccionar esa transición de salida continuando con el flujo, sin considerar las otras transiciones de salida. La condición se deberá poner entre corchetes. Cada punta de rombo puede tener varias transiciones de salida.




Otra manera de representar lo anterior es sin utilizar el rombo y emitir transiciones directamente de la actividad predecesora a la toma de decisión, pero cada transición deberá estar aún así acompañada de su condición.



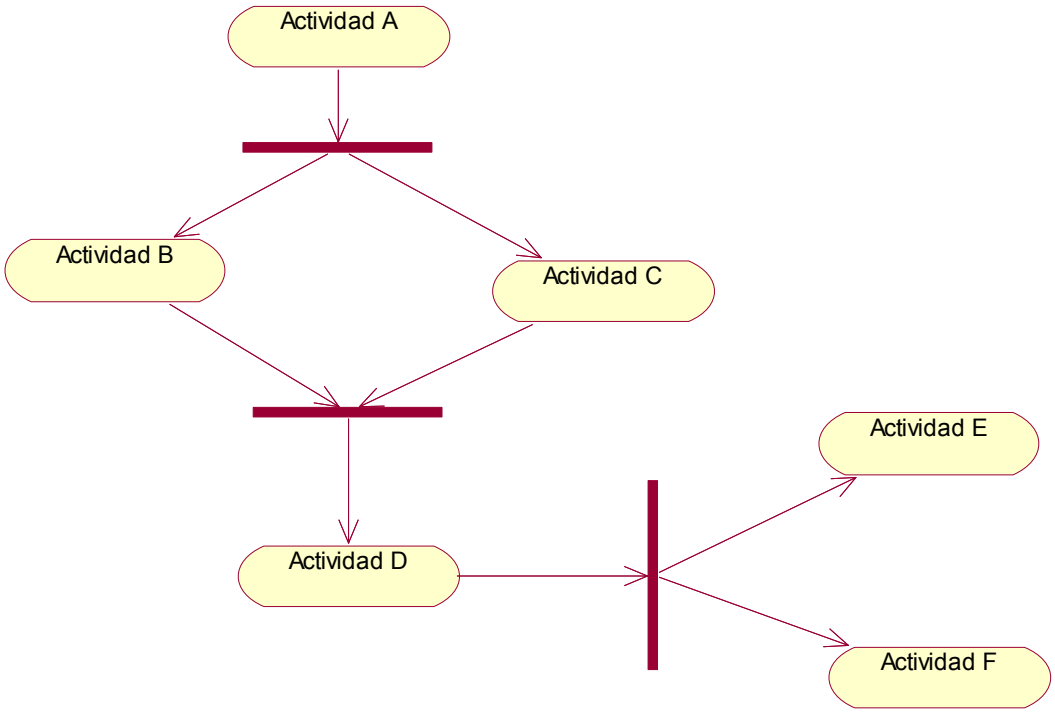

La bifurcación nos puede ayudar a representar bucles o ciclos, a veces si en una bifurcación una transición de salida no indica condición, representa la salida en caso de no cumplir con ninguna de las condiciones, por eso sólo debe dejarse a lo más una transición de salida sin condición.



## Concurrencias

Una actividad puede recibir el flujo de varias transiciones, pero sólo una transición de entrada llevará el flujo, las otras transiciones no estarán activas, como sucede en el diagrama de lectura de un archivo en donde el rombo de bifurcación tiene dos entradas, pero sólo una de ellas estará activa.

Para representar la multiplicidad de diferentes hilos de control se utilizan las barras de sincronización, tanto para indicar la división en diferentes hilos de control, como la unión de diferentes hilos de control. Si requerimos representar concurrencia este deberá ser el camino único para representarla. La barra de sincronización es una línea continua gruesa horizontal o vertical.

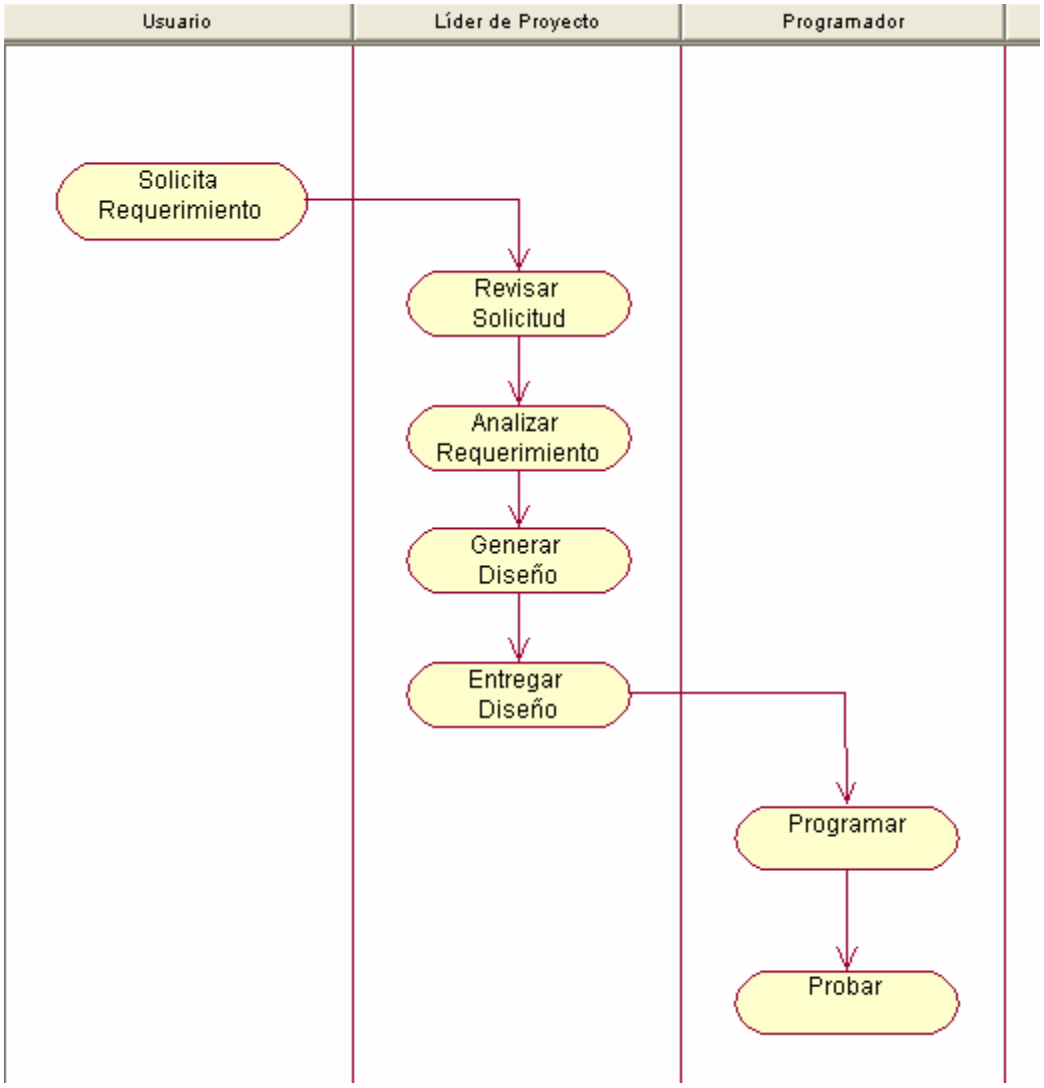



Después de concluir la *actividad a* la transición lleva generar dos hilos de control, es decir procesar en paralelo la *actividad b* y la *actividad c*, al concluir ambas actividades se unen nuevamente en una barra de sincronización horizontal para iniciar la *actividad d,* la transición de esta actividad se vuelven a generar dos hilos de control mostrados a través de una barra de sincronización vertical.

## Carriles

Algo muy útil dentro de los diagramas de actividad son los carriles, estos nos ayudan a mostrar la responsabilidad de cada carril. Cada carril tiene su nombre en la parte superior del diagrama y es

delimitado por una línea vertical continua, ninguna actividad podrá situarse en medio de dos carriles, mientras las transiciones son las únicas con la posibilidad de cruzar el carril. El nombre de cada carril se otorga para mencionar una responsabilidad de alto nivel, siendo su uso más común el nombrar carriles con los nombres de los actores.

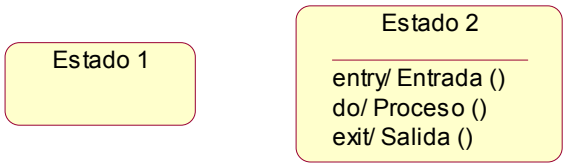



# Diagrama de Estados

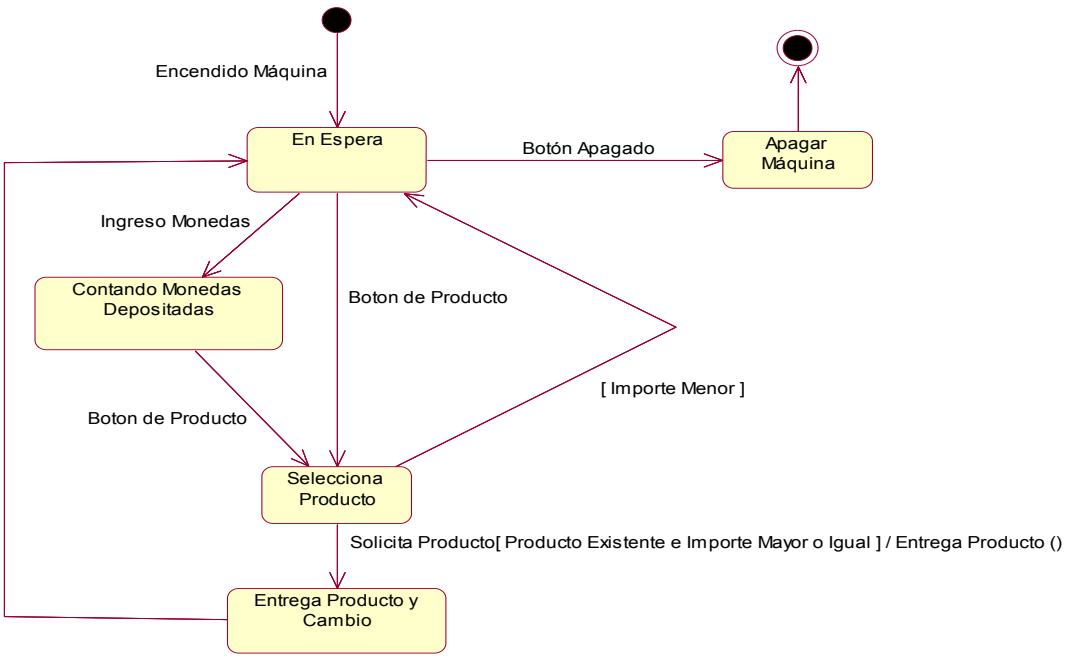
Los diagramas de estado muestran los diferentes estados por los cuales pasa un objeto a través de las transiciones.

Del diagrama de actividades se pueden recuperar los iconos de inicio, fin, bifurcación y concurrencia, además de las transiciones jugando estas últimas un papel diferente.

Un estado se representa como un rectángulo con las esquinas redondeadas, dentro del rectángulo se pone el nombre del estado, si fuera necesario indicar actividades de entrada, proceso y salida se coloca una línea continua bajo el nombre y posteriormente las leyendas de *entry, do, exit*, de la misma manera similar a los diagramas de actividades.



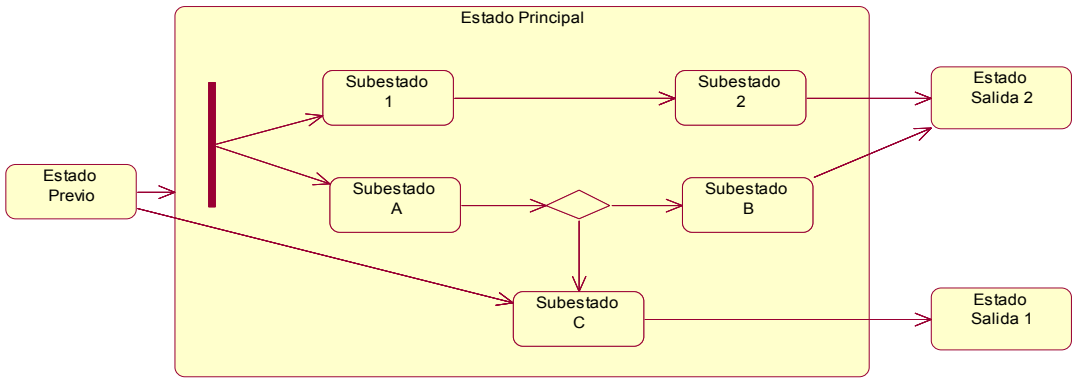

Las transiciones unen a los estados con una línea dirigida la cual puede tener un nombre, mientras en los diagramas de actividades el nombre de la transición no es importante, en los diagramas de estado resulta generalmente de gran interés. El nombre de la transición de estado se compone del nombre del evento causante de un cambio de estado y una condición que se cumpla para disparar el evento, adicionalmente si se necesita se puede separar con una línea diagonal para indicar el nombre de la acción (actividad) requerida para transformar el objeto a otro estado.



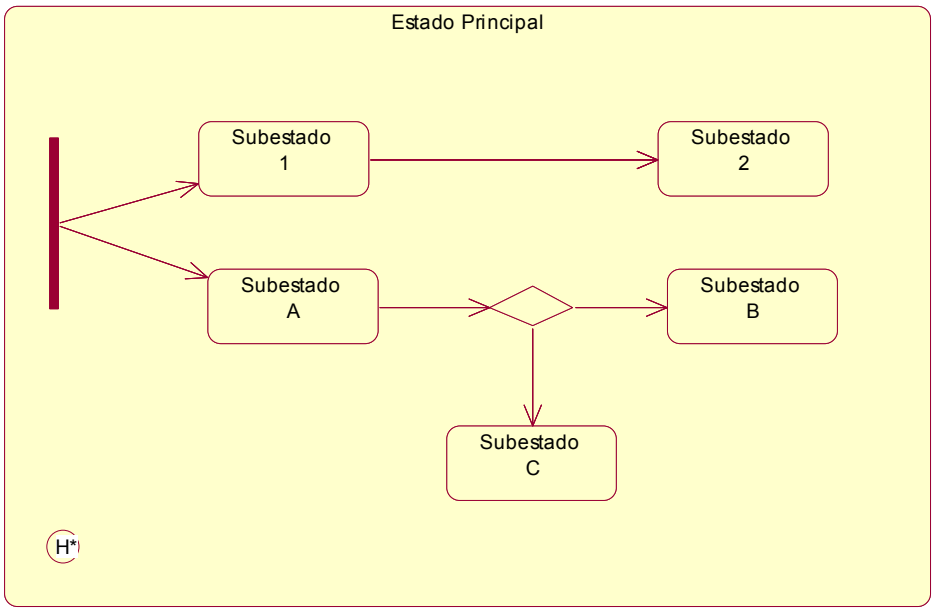

Como podemos ver en el diagrama de estados se muestra el funcionamiento (simplificado) de los estados por los cuales pasa una máquina de refrescos. Inicia el ciclo de vida de la máquina cuando se enciende y entra a un estado de espera, a partir de ahí se puede tomar dos caminos disparados por dos eventos, el ingreso de monedas o pulsar el botón de un producto, dependiendo de la selección del evento se entrará a un estado diferente. Si las monedas ingresadas son inferiores en valor al precio del producto se regresa

al estado de espera, ya sea para ingresar más monedas o seleccionar un producto de menor precio. Si las monedas recibidas satisfacen el precio y existe el producto entonces pasa al estado de entregar producto. Algunas transiciones entre estados como la de *entrega producto y cambio* hacia *en espera* llega a no ser necesario indicar el nombre, o bien como en el caso de la transición entre *selecciona producto* y *en espera,* la relación sólo es interesante conocer su condición.

Dentro del rectángulo de estado, pueden indicarse subestados utilizando diagramas de estado anidados incluso un estado puede presentar concurrencia utilizando las barras de sincronización, así como bifurcaciones.




Dentro de un estado pueden presentarse eventos que obliguen a salir del estado actual y cambiar a otro estado, pero puede necesitarse recordar el subestado de ejecución al regresar al estado. Este histórico de estado se representa indicando una letra H con un círculo alrededor dentro del estado, si la H lleva adicionalmente un * tendrá la capacidad de recordar cualquier nivel de anidamiento de sus subestados, en caso contrario sólo el primer nivel de anidamiento.




# Diagrama de Componentes

Los componentes se utilizan para modelar la distribución de elementos lógicos (clases, código fuente, etc.) dentro de elementos físicos asociados a un nodo, siendo el nodo la representación de cualquier tipo de recurso computacional como memoria, tiempo de procesador, espacio en disco, etc.

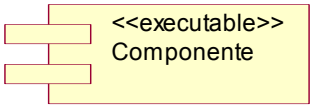
La importancia de los componentes es la de mostrar el mapa donde radicarán las clases, código fuente, etc. Los componentes los podemos clasificar en:

- Componentes de Despliegue: Representan a sistemas ejecutables, como EXE's, DLL's, Java Beans, Bases de Datos, etc.
- Componentes Producto del Trabajo: Dentro de este segmento tenemos el código fuente, documentación, script's, etc.
- Componentes de Ejecución: Son el resultado de sistemas en ejecución, como puede ser la instanciación de una clase, creación de un archivo, etc.

Los componentes surgen principalmente por la necesidad de generar código reutilizable y cada vez realizar el desarrollo más rápidamente. La ventaja de formar componentes es lograr sustituir a cualquier de estos sin afectar la funcionalidad de un sistema, para esto será necesario contar con una interface bien definida, de esta manera puede cambiar la manera de solucionar la interface pero la respuesta siempre será la misma.

Un componente se representa por un rectángulo mayor con dos rectángulos menores situados en su costado izquierdo. Además del nombre puede ir acompañado de un estereotipo, algunos de los más usuales son:

- executable: Componentes ejecutables en algún nodo
- library: Componentes de tipo biblioteca dinámica o estática
- table: Componente para representar una base de datos o tabla
- file: Componente asociado a un archivo fuente, archivos de
- ayuda, archivos de salida, etc.
- document: Componentes documentales del sistema

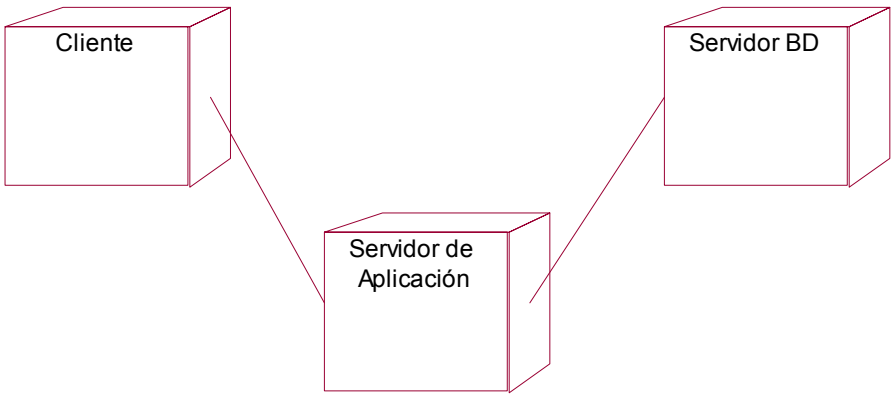



La principal relación útil entre componentes es la relación de dependencia




# Diagramas de Distribución

Un nodo es un elemento físico situado en un tiempo dado de ejecución y consume recursos computacionales. Los nodos alojan a los componentes para ser ejecutados y representan el hardware, sobre el cual es implementada una solución a un sistema. Un nodo se representa a través de un cubo, relacionándose con otros cubos a través de asociaciones, formando de esta manera un diagrama de distribución.


Cliente
Servidor BD
Servidor de
Aplicación

# CAPÍTULO IV. METODOLOGÍA DE DESARROLLO BASADA EN UML

Dentro del planteamiento de esta metodología de trabajo, basada en RUP (Rational Unified Process), creada también por los padres de UML, la secuencia es recomendada, pero incluso se puede omitir, modificar o anexar nuevos elementos, realmente esta metodología debe ser utilizada como una base sólida para la creación de software y una guía confiable, más no dictatorial. Sería sumamente complejo ofrecer y seguir una metodología capaz de indicar todas las decisiones requeridas durante el desarrollo. Adicionalmente, esta metodología ofrece el equilibrio razonable de análisis, diseño y documentación para aplicarla en proyectos de empresas grandes, en donde se buscan resultados en tiempos cortos, soluciones eficientes e inversiones justificadas contra un análisis de costo beneficio.

Cuando exista alguna duda en algún proceder si la decisión tomada complica la situación generalmente es un síntoma inequívoco de una mala decisión.

Durante un desarrollo, en todo momento es necesario decidir diferentes aspectos, por eso nunca se deben de perder de vista los siguientes puntos:

- La sencillez de la solución debe predominar ante todos los factores, y sólo en casos específicos con gran importancia deberá ser desplazada por algún otro factor
- El detalle de las cosas debe llegar hasta donde el camino a seguir posteriormente sea obvio para cualquier persona con capacidad de revisar la documentación o en algunos momentos el detalle no aporte una trascendencia vital
- Se puede auxiliar de cualquier elemento para cubrir el propósito anterior, de preferencia existente dentro de la metodología
- Todos los modelos deberán mostrar los detalles más importantes y útiles, para no excedernos en su complejidad
- Los modelos son interpretaciones, por eso depende de la vista del modelador para dar un significado a sus componentes, aunque entre menos interpretaciones será mejor
- Ante duda para actuar, buscar una solución ordenada pero sumamente intuitiva, todas las metodologías nacen con ese principio
- El usuario es el experto, no el desarrollador, en caso de duda consultarlo

- El usuario no siempre transmite el conocimiento de manera acertiva o no tiene en todo momento solución a todos los problemas, es necesario cuestionarlo, guiarlo y darle tiempo para satisfacer respuestas

- En caso de contradicción entre usuarios es necesario buscar una negociación entre ellos

- Nombrar las cosas por su propósito y función, a pesar de que resulten nombres largos, es la mejor documentación

Una cuestión muy importante es no entender una metodología como una secuencia de actividades estáticas, en cualquier momento podemos regresarnos unos pasos atrás, aunque el repetir pasos anteriores debemos tratar de evitarlo.

# Modelo del Negocio

Cuando un usuario tiene la idea de eficientar un proceso y piensa en una solución informática de software realiza una petición para obtener una asesoría, al tener contacto con ingeniero de software entiende ese requerimiento como el requerimiento global de un nuevo software, habitualmente el requerimiento representa un proceso del negocio completo. El requerimiento global es el objetivo primordial de eficientar un proceso, los requerimientos para satisfacer el requerimiento global son los requerimientos detallados. Para comprender el entorno de su requerimiento es necesario realizar un análisis referencial del ámbito del negocio a través de un modelo del negocio.

El modelo del negocio nos ayuda a visualizar los procesos primordiales del negocio para comprender el entorno del negocio y ofrecer soluciones con visión integral, además de integrarnos al dominio del vocabulario del negocio. De manera simple, podemos entender a un proceso del negocio como un conjunto de procesos agrupados, para otorgar un propósito completo dentro del negocio, dichos procesos se pueden desprender de los objetivos del negocio u operación de sus áreas principalmente. Si fuera necesario cada proceso del negocio puede descomponerse en subprocesos del negocio.

El detalle de análisis de los procesos del negocio debe ser, al menos, el mínimo indispensable para dominar el vocabulario del negocio, la conformación del negocio y sus principales procesos, aunque es importante conocer más a detalle los procesos con mayor interacción con el requerimiento global. Los procesos de negocio pueden ser de toda una institución, un área de trabajo o donde nuestro entorno de requerimiento lo solicite.

El modelo del negocio en contadas ocasiones se realiza, pero en la medida de lo posible debe realizarse para inducirse más rápido al dominio del negocio por medio del vocabulario propio del negocio.

## *Casos de Uso del Negocio*

Para encontrar los casos de uso de los procesos del negocio es necesario realizar sesiones de trabajo con usuario para iniciarnos en el conocimiento del negocio, sin ahondar de momento en los requerimientos. Para esto debemos recavar información acerca del organigrama organizacional, planes estratégicos, recomendaciones emitidas, sistemas actuales, etc. Las reuniones de trabajo deberán estar canalizadas a obtener el modelo del negocio, o el segmento del modelo de negocio de nuestro interés. Cada caso de uso de negocio representa el entorno macro en donde se relaciona nuestro requerimiento, siempre y cuando el requerimiento represente un grado de dificultad importante, como para analizar su entorno de negocio, incluso algunas ocasiones pueden existir requerimientos globales situados en diferentes casos de uso del negocio. Generalmente cada caso de uso de proceso de negocio se convierte en un sistema, sin ser imperante lo anterior.

## Diccionario del Negocio

El diccionario del negocio debe iniciarse a la brevedad posible para indicar los términos y palabras propias del negocio, con un significado enfocado a personas con nulo conocimiento sobre el negocio, este diccionario deberá actualizarse durante todo la duración del proyecto.

## Diagramación de Casos de Uso

Del resultado de las reuniones de análisis de los procesos del negocio debemos listar los casos de uso del negocio para proceder a graficarlos, indicando los actores con alguna interacción con el caso de uso, así como generalizaciones, extensiones e inclusiones.

Un actor representa un rol dentro de un proceso, por eso una persona puede jugar diferentes roles como actor o un actor agrupar a diferentes personas. Un actor no se limita únicamente a personas, también puede representar áreas, organizaciones, software, hardware, etc., es decir, cualquier entidad con interacción hacia nuestro caso de uso.

En este punto las generalizaciones y extensiones serán poco frecuentes, la gran mayoría de las ocasiones las relaciones de inclusión resultarán más útiles, pero dependerá del tamaño y cantidad de los procesos del negocio, además de la profundidad necesaria para empaparse del negocio en términos generales.

## Especificación de Casos de Uso

La especificación de casos de uso del proceso deberá contener información textual descriptiva para comprender los diagramas de caso de uso indicados. Entre la información básica contenida deberá estar el nombre del proceso del negocio, objetivo, importancia y riesgos, prioridad, frecuencia de ejecución,

tiempo de ejecución, costo de ejecución y una descripción. Al indicar la descripción no se deberá tratar de llegar a los detalles del proceso, sólo describir aspectos generales e importantes para ser un primer acercamiento formal al dominio del problema.

| PROCESO DE NEGOCIO | |
|---|---|
| ***OBJETIVO*** | |
| ***IMPORTANCIA*** | |
| ***RIESGOS*** | |
| ***PRIORIDAD*** | |
| ***RESTRICCIONES*** | |
| ***FRECUENCIA DE EJECUCION*** | |
| ***TIEMPO DE EJECUCION*** | |
| ***COSTO DE EJECUCION*** | |
| ***USUARIO / ACTOR*** | |
| ***DESCRIPCION*** | |

El especificar todos los casos de uso de negocio puede resultar una labor demasiada ardua, esto nos orilla a especificar los casos de uso de negocio con menor relación a nuestro requerimiento con menores detalles, y a mayor grado de relación con el requerimiento mayor detalle. Generalmente el caso de uso de negocio relacionado al requerimiento en forma directa le especificaremos con una buena descripción y todos los puntos indicados.

## *Diagramas de Roles*

Los diagramas de roles son diagramas de clases de los actores participantes dentro de los casos de uso de negocio con sus relaciones de asociación, herencia, agregación, composición.

Por ejemplo la herencia nos podría a ayudar a modelar el organigrama de la empresa, la agregación y composición a la integración de departamentos y la asociación para indicar las principales relaciones laborales. Se puede tener un solo diagrama de roles para todos los casos de uso de negocio o un diagrama de roles por cada caso de uso de negocio. Adicionalmente se debe incluir un diccionario de roles.

## *Diagramas de Procesos*

Los diagramas de procesos son diagramas de actividades desprendidos de cada caso de uso de negocio especificado, muestran las actividades realizadas por cada uno de los roles involucrados dentro de un caso de uso de negocio, es decir, cumplen el propósito de mostrar el flujo de trabajo completo del un proceso de negocio. Las actividades deberán ser coherentes con la especificación del caso de uso de negocio. Adicionalmente debemos incluir un diccionario de actividades para cada caso de uso de negocio.

## *Diagrama de Escenarios*

Los diagramas de escenarios son diagramas de colaboración o secuencia desprendidos de cada caso de uso de negocio especificados, muestran la comunicación existente entre los diferentes roles del caso de uso de negocio, es decir, cumplen el propósito de visualizar los mensajes enviados entre los diferentes actores para satisfacer el caso de uso de negocio. Adicionalmente debemos incluir un diccionario para indicar el propósito de una relación de emisor, receptor y mensaje.

Tanto los diagramas de procesos como los de escenarios muestran perspectivas diferentes de un mismo caso de uso de negocio para tener un panorama más completo, son el resultado del entendimiento del caso de uso de negocio.

# Modelo del Dominio

El modelo del dominio es representado a través de un diagrama preeliminar de objetos, para obtener el diagrama podemos partir del modelo del negocio, en especial del caso de uso en donde se ubica el requerimiento global. En caso de no existir un modelo del negocio, se pueden iniciar pláticas, habitualmente se aprovechan las sesiones iniciales del modelo de requerimientos, para ubicar el vocabulario del dominio del problema en el ámbito del requerimiento. El diagrama preeliminar de objetos es un diagrama en donde representamos las relaciones existentes entre los principales elementos del vocabulario, sin ser su propósito obtener el diagrama de objetos preciso del problema, debe ser una aproximación inicial. Este diagrama debe ser la base del vocabulario para el modelo de requerimiento. El modelo del dominio es importante para iniciar la unificación del vocabulario del requerimiento entre los participantes del desarrollo, por eso es importante realizar este modelo.

# Modelo de Requerimientos

Una vez comprendido el modelo del negocio y modelo del dominio podemos situar perfectamente la relevancia y ubicación del requerimiento global dentro del negocio, en la mayor parte corresponderá a un caso de uso de negocio y en menor ocasión a una parte de este. La entrada deseable del modelo de requerimientos es el modelo del negocio y modelo del dominio.

Este modelo brindará la información suficiente para detallar las necesidades de los usuario, después de concluir este modelo todos los involucrados en el desarrollo podrán entender a la perfección la necesidad de los usuario, sin pensar en la manera de resolverlo y sin profundizar en los detalles de funcionamiento.

Las deficiencias del modelo de requerimientos pueden ser causadas primordialmente por:

- Transmisión de una visión incompleta o errónea por parte del usuario

- Inexperiencia en el proceso por parte del usuario
- Ocultar información o deformarla intencionadamente por parte del usuario
- Cambios próximos al proceso sin definiciones detalladas
- Carencia de habilidad del desarrollador para interpretar y encausar los requerimientos
- Temor del desarrollador a cuestionar al usuario ante las dudas
- Suponer cosas aparentemente obvias
- Automatizar procesos deficientes o erróneos

Adicionalmente es necesario guiar al usuario para ofrecerle soluciones adicionales como:

- Seguridad
- Creación de perfiles para corregir información errónea
- Históricos de información
- Automatización con interfaces
- Cifras control
- Respaldos
- Reorganizaciones de base de datos
- Procesos alternos en caso de contingencias

También será necesario informarle de las limitantes tecnológicas existentes o en su defecto los costos para minimizarlas:

- Disponibilidad del sistema
- Anchos de banda
- Limitantes de lenguajes de programación

Obviamente entre más se acrecente el número de requerimientos implicará un esfuerzo más grande para satisfacerlos. Una típica actitud de un desarrollador inexperto ante un problema complejo es decir que no se puede, pero todo se puede, la diferencia pueden ser los esfuerzos o costos requeridos para lograrlos.

Aunque el usuario sea el experto debemos cuestionarle el porque de las cosas, el como funcionan, si ya se han cubierto todas las variantes, etc. Muchos usuarios olvidan que un desarrollador es un experto informático pero un ignorante en el proceso de negocio, y como tal se tiene la obligación de acentuarlo, ningún sistema hará cosas no especificadas e indicadas, por más obvias que resulten. Debemos tener cuidado en no ofrecer mejoras importantes al proceso hasta no comprenderlo de manera integral.

## *Especificación de Requerimiento Global y Detallados*

Antes de iniciar cualquier actividad propia del modelo de requerimientos debemos llevar a cabo reuniones para entender los requerimientos a satisfacer. Si contamos con el modelo del negocio parte de esta labor se encontrará muy avanzada, pero a partir de este momento es cuando ponemos foco total al requerimiento global y los requerimientos detallados.

Posterior a las reuniones se plasma de manera escrita, listada y breve cada uno de los requerimientos detallados. De ser posible los requerimientos se agrupan por criterios de funcionalidad. Para cumplir esta labor nos podemos apoyar de la siguiente plantilla:

| **REQUERIMIENTO** | |
|---|---|
| ***OBJETIVO*** | |
| ***IMPORTANCIA*** | |
| ***RIESGOS*** | |
| ***PRIORIDAD*** | |
| ***RESTRICCIONES*** | |
| ***FRECUENCIA DE EJECUCION*** | |
| ***TIEMPO DE EJECUCION*** | |
| ***COSTO DE EJECUCION*** | |
| ***USUARIO / ACTOR*** | |
| ***DESCRIPCION*** | |

Si el requerimiento global corresponde a un caso de uso de negocio, el listado de requerimientos detallados se desprenderá del diagrama de procesos del modelo del negocio, un requerimiento detallado por cada actividad, y si el requerimiento global es parcial, el listado de caso de uso de negocio saldrá del conjunto de actividades correspondientes al requerimiento global. No todas las actividades del diagrama de procesos pueden ser solicitadas como requerimientos.

A la especificación del requerimiento le debemos anexar los requerimientos no funcionales como seguridad y desempeño entre otras.

También es conveniente revisar, si no se tiene el modelo de negocio, un listado de actores para determinar si un actor representa a un usuario real y en caso contrario eliminar dicho actor, evitando tener actores engañosos. Otra manera de encontrar actores es resumir las responsabilidades de roles similares en un actor.

Para encontrar requerimientos en ausencia del modelo del negocio, podemos revisar las responsabilidades de los roles de los actores con relación directa al requerimiento global, obviamente esta labor es muy similar a la realización del diagrama de procesos. También podemos concentrarnos en los requerimientos primordiales del negocio e ir avanzando hacia los menos importantes y redondearlos con requerimientos para soportarlos, como pudiera ser la creación de catálogos.

La descripción de requerimientos es una aproximación de la futura especificación del caso de uso, por eso no se trata de profundizar en los detalles finos, solamente se extrae la información proporcionada por el usuario como sus inquietudes, ya en la especificación del caso de uso se realiza una revisión para definir todos los detalles y solucionar posibles impactos no deseados con otros casos de uso.

# *Diagramas de Casos de Uso*

Una vez concluida la especificación de requerimiento podemos iniciar la diagramación de casos de uso.

## Casos de Uso Funcionales

Los casos de uso se desprenderán de listado de requerimientos funcionales detallados. La diagramación, clasificación, relación con actores pueden realizarse en paralelo.

### *Diagramación de Casos de Uso*

Se realiza la diagramación de los caso de uso funcionales sin considerar relaciones con actores. En este momento deben indicar en la plantilla del caso de uso el nombre del caso de uso, objetivo, precondiciones y postcondiciones.

### *Clasificación de Casos de Uso*

Clasificamos los casos de uso en paquetes acordes a criterios de clasificación como pudieran ser módulos de funcionalidad, distribución geográfica, áreas, etc. Una vez concluida la clasificación se captura en la plantilla de caso de uso el paquete asignado a cada caso de uso.

### *Especificación de Relación con Actores*

Es necesario retomar los diagramas de caso de uso para indicar las relaciones entre los actores y los diferentes casos de uso basado en la especificación del requerimiento. En este momento se actualiza en la plantilla de caso de uso los actores y su función dentro del caso de uso.

### *Clasificación de Actores por Paquete*

De ser necesario los actores pueden clasificarse en paquetes.

## Casos de Uso No Funcionales

Los casos de uso no funcionales son aquellos que no ofrecen una solución directa al objetivo del negocio, pero ayudarán a ofrecer una solución completa para controlar el sistema, habitualmente como casos de uso no funcionales encontraremos la seguridad, respaldos de información, transferencias de información, etc.

Son tratados de la misma manera que los casos de uso funcionales y se integran a la misma documentación. Algunas situaciones pueden derivar en incluir la necesidad no funcional dentro de un caso de uso funcional, un ejemplo es el tiempo de respuesta máximo para un proceso o el registro de una auditoría de transacciones.

# Análisis

El análisis es la etapa en donde el usuario participa para detallar el funcionamiento de los casos de uso, abarcando todas las variantes de cada caso de uso. Cualquier punto no incluido dentro del análisis el sistema no lo resolverá. En algunas ocasiones es necesario refinar de manera gradual el análisis de cada caso de uso. Es muy común refinar el modelo repitiendo pasos ya concluidos, eso no implica un mal análisis o diseño, simplemente es un proceso gradual para encontrar el mejor modelo, preferentemente siempre deben ser ampliaciones y no correcciones a lo realizado.

De manera común una generalización representa flujos de estados similares con propósitos diferentes, la inclusión frecuentemente se usa para extraer funcionamientos comunes a varios casos de uso y una extensión se usa para desprender en otros casos de uso funcionamientos complejos o grandes. La generalización y la inclusión tienen diferencias sustanciales, el primero encuentra un gran número de coincidencias entre algunos casos de uso mientras la inclusión encuentra coincidencias mínimas que no requieren una generalización.

## *Diagrama de Generalización de Casos de Uso*

Las generalizaciones son el agrupamiento de secuencias de flujos o procesos factorizables común a varios procesos de propósito similar. De los todos los caso de uso debemos encontrar aquellos con

semejanzas de comportamiento (generalmente se presentarán en la transición entre los estados de sus interfaces), si enfocamos nuestra búsqueda en términos de herencia será mucho más sencillo encontrar las posibles generalizaciones. De los casos de uso definidos con características de generalización se desprenderá un caso de uso a un nivel superior para especificar la generalización. Está búsqueda se realiza en forma iterativa, teniendo cuidado en todo momento, de no generar demasiados niveles de generalización para complicar de manera innecesaria el análisis. Si un caso de uso de generalización pudiera observarse como poco reutilizable o con funcionamiento escaso, debemos dudar si es útil la generalización en términos prácticos. La generalización se verá como un comportamiento común, en donde cada caso de uso generalizado aportará los detalles específicos para satisfacer un comportamiento común.

## *Diagrama de Extensión de Casos de Uso*

La extensión es un flujo opcional o excepcional con un grado de complicación suficiente como para realizar un análisis desglosado de la misma. De todos los casos de uso será necesario determinar la navegación entre los diferentes casos de uso, esta navegación puede representarse a través de la extensión, para esto probablemente tendremos que crear casos de uso nuevos para ayudar a la navegación entre los diferentes casos de uso de una manera agrupada.

## *Diagrama de Inclusión de Casos de Uso*

La inclusión es una modularización o factorización de una secuencia de uno o más procesos, generalmente con propósitos diferentes. En caso de contar con un caso de uso complejo de analizar podemos modularizarlo para dividirlo en problemas más simples de entender. También se puede presentar una funcionalidad común en diferentes casos de uso, entonces podemos crear una inclusión y llevar esa funcionalidad a la inclusión para realizar su análisis una sola vez.

## *Especificación de Casos de Uso*

Durante la especificación del caso de uso es necesario reunirse con el usuario para abarcar en su totalidad cada uno de los casos de uso desprendidos a un nivel de detalle total, esta revisión puede hacerse por paquetes de caso de uso, complejidad, importancia en el negocio, etc. Para desglosar el caso de uso es conveniente entender las entradas al caso de uso y la salida del mismo, entender la relación entre dicha información para proponer un diseño de interface y posteriormente detallar la funcionalidad del caso de uso.

Para los casos de uso de extensión en algunos casos se puede omitir su especificación por la simpleza del mismo.

La manera de iniciar con las especificaciones de caso de uso puede realizarse en cualquier orden, pero siempre será conveniente definir una estrategia como puede ser seguir el orden de los requerimientos detallados sugerido por el diagrama de procesos en orden inverso y dando mayor peso a los flujos con mayor grado de importancia, logrando intuir gradualmente la responsabilidad de los casos de uso previos, o acompañar la técnica del orden inverso, pero partiendo de los procesos de negocio con mayor complejidad o importancia. Cuando se requiera una de un caso de uso padre para completar la funcionalidad de un caso de uso hijo, se dará peso a la especificación del caso de uso padre.

## Entradas y Salidas

Buscamos toda la información de entrada pensando en el objetivo del caso de uso, ya sea directa o indirecta, como por ejemplo información digitada por el usuario, algún archivo, etc. La información de entrada ayuda a comprender cual debe ser el resultado generado por el caso de uso.

A veces la especificación de la entrada y salida pueden ser un conjunto de datos o un resumen muy general del resultado del caso de uso como pudiera ser: Cálculo del pago por persona actualizado, pero siempre entre más detalle se especifique resultará de mayor utilidad.

## Especificación de Interfaces

Tomando como base las entradas y salidas, y la relación entre estas nos permiten ofrecer un diseño de interface en base a la relación de la información y el propósito de caso de uso.

Al diseñar una interface gráfica debemos tener en cuenta la manera ideal de capturar la información, orden de recepción de la información, agrupación física de información, evitar el manejo de claves difíciles de recordar para el usuario, menor número de acciones posibles para el usuario.

## Descripción de Caso de Uso

La descripción del caso de uso debe tener algunos principios básicos:

- Orientada a la funcionalidad y no a la implementación
- El vocabulario empleado debe ser el del proceso
- Debe soportar crecimiento, mantenimientos y reutilización
- Bajo acoplamiento interno y externo
- Alta modularidad interna y externa

Debemos recordar que la descripción del caso de uso se desprende de la necesidad del usuario y se debe contemplar todas las posibles variaciones, incluso si algunas variaciones complican demasiado el caso de uso pudiera requerirse actualizar el análisis, para incluir un nuevo caso de uso.

## *Prototipo*

Una vez concluido prácticamente el análisis podemos realizar las interfaces en las herramientas seleccionadas para el desarrollo y mostrar el prototipo al usuario para reafirmar si el camino tomado es el esperado por él.

# Diseño

En el diseño es cuando tomamos el análisis para tratar de satisfacer las necesidades a través de estructuras de programación. Primero nos enfocamos a encontrar las clases salientes de la descripción de los casos de uso, es decir ver el futuro sistema como un esquema estático, es como plasmar parte del vocabulario del dominio del sistema como clases, y posteriormente analizamos y diseñamos una solución para hacer funcionar esas clases en un esquema dinámico.

## *Identificación de Clases y Relaciones*

Al identificar y proponer alguna clase debemos definir si queremos que sea usada por cualquier aplicación, por la aplicación en desarrollo o se trata de una clase de un funcionamiento muy particular, esto nos ayudará a definir el alcance de satisfacción de la interface de la clase, para saber que tan general o particular deben ser sus contratos soportados.

Dentro de las clases las podemos categorizar en tres tipos en base a su propósito:

- Clases de Entidad
- Clases de Interface
- Clases de Control

A cada conjunto de clases las podemos clasificar en paquetes. Una vez obtenidas las clases será necesario indicar sus relaciones entre las de su mismo tipo, y posteriormente sus relaciones entre tipos diferentes. Concluyendo esta actividad contaremos con el modelo estático del sistema

## Clases de Entidad

Para encontrar las clases de entidad realizamos una revisión con la descripción del caso de uso y utilizamos los conceptos de diseño de base de datos.

### *Normalización de Base de Datos*

Cuando ya encontramos las clases de entidad les aplicamos una normalización de base de datos para encontrar el diseño óptimo y posteriormente regresamos a graficar las clases de entidad con todos sus atributos.

## Clases de Interface

Las clases de interface se desprenden directamente de la especificación de interfaces tomada de la especificación de caso de uso.

## Clases de Control

Las clases de control son paralelas al flujo de solución de un caso de uso.

## Diagrama de Estado, Secuencia, Colaboración y Actividades

Estos diagramas nos deben ayudar a desmenuzar a mayor detalle la interacción de los objetos, dependiendo de las partes que así lo requieran, generando un modelo dinámico del sistema.

# *Identificación de Métodos y Atributos*

Al comprender el significado de un método y atributo muchos de estos están indicados en las especificaciones de casos de uso. También podemos ubicar los métodos y atributos de la interface de una clase a través de los diagramas dinámicos y otros de los diagramas estáticos. De los diagramas estáticos es importante sólo considerar los métodos y atributos de interés dentro de la solución de nuestro problema, tomando en cuenta si la clase es de propósito muy general o particular, de esta manera estaremos evitando el diseño de métodos y atributos sin interés alguno en nuestra solución. Los métodos y atributos de implementación se derivarán a partir de las necesidades internas de la clase que ayuden a satisfacer su interface y no sean de interés para sus clientes.

# *Diagrama de Componentes*

El segmentar una aplicación obedece a causas de requerimientos no funcionales como: ligereza de clientes, servidores de aplicaciones, rapidez de ejecución y algunos otros factores. Para vislumbrar esta separación podemos utilizar los diagramas de componentes. Cuando se presenta la segmentación de una aplicación tiene ventajas pero también se sacrificarán algunas otras. Si tuviéramos clientes con poca capacidad de procesamiento podríamos optar por usar una aplicación cliente muy ligera y una aplicación servidor con las reglas del negocio. También pudiéramos tener diferentes aplicaciones con uso de transacciones comunes lo que podría implicar crear un servidor de aplicaciones de ciertas transacciones y de esa manera sólo desarrollar una vez las reglas del negocio. Ahora imaginemos que decidimos

mantener todo en el cliente para disminuir la comunicación con el servidor que es lenta además de tener un servidor poco robusto, pero tuviéramos clases útiles a diferentes aplicaciones llevándonos a crear componentes dentro del cliente. Si se presenta una aplicación muy grande podríamos recurrir a segmentarla en componentes para exclusivamente trabajar con aquellas que puedan ser utilizadas por algún perfil de usuario. Como podemos ver pueden existir diferentes razones para decidir si creamos componentes, si contamos con diagramas que nos muestren su relación será más sencillo comprenderlos.

# Organización del Proyecto (Plan de Trabajo)

Cuando se inicia todo proyecto es importante dimensionar el tiempo y recursos requeridos para su realización. Sin embargo, al realizar un proyecto, existen algunas actividades pocas veces programadas, pero indispensables dentro del proyecto. A pesar de no ser el objetivo de este tema, si mencionaremos la secuencia de actividades recomendada a seguir, siendo una guía útil para modificar, eliminar o adicionar actividades:

| ID | Nivel | Actividad | Predecesora |
|---|---|---|---|
| 1 | 1 | **PROYECTO** | |
| 2 | 1.1 | **FASE INICIO DE PROYECTO** | |
| 3 | 1.1.1 | Asignación de Espacio y Accesorios de Trabajo | |
| 4 | 1.1.2 | Solicitud de Inicio de Proyecto | **3** |
| 5 | 1.1.3 | Agenda de Reuniones para Fase | **4** |
| 6 | 1.1.4 | Definición de Estándares de Seguimiento de Proyecto y Documentación | **5** |
| 7 | 1.1.5 | Definición de Estándar UML | **6** |
| 8 | 1.1.6 | Obtención de Documentación Existente | **7** |
| 9 | 1.1.7 | Modelo del Negocio | **8** |
| 10 | 1.1.7.1 | Casos de Uso de los Procesos del Negocio | |
| 11 | 1.1.7.1.1 | Diagramas de Caso de Uso | |
| 12 | 1.1.7.1.2 | Especificación de Casos de Uso | **11** |
| 13 | 1.1.7.2 | Diagrama de Roles | **10** |
| 14 | 1.1.7.3 | Diagramas de Procesos | **13** |
| 15 | 1.1.7.4 | Diagrama de Escenarios | **14** |
| 16 | 1.1.8 | Modelo de Requerimientos | **9** |
| 17 | 1.1.8.1 | Diagrama Preeliminar de Objetos | |
| 18 | 1.1.8.2 | Diagramas de Casos de Uso | **17** |
| 19 | 1.1.8.2.1 | Casos de Uso Requeridos | |
| 20 | 1.1.8.2.1.1 | Diagramación de Casos de Uso | |
| 21 | 1.1.8.2.1.2 | Clasificación de Casos de Uso | **20** |
| 22 | 1.1.8.2.1.3 | Especificación de Relación con Actores | **21** |
| 23 | 1.1.8.2.1.4 | Clasificación de Actores por Paquete | **22** |
| 24 | 1.1.8.2.2 | Casos de Uso No Requeridos | **19** |
| 25 | 1.1.8.2.2.1 | Diagramación de Casos de Uso | |
| 26 | 1.1.8.2.2.2 | Clasificación de Casos de Uso | **25** |
| 27 | 1.1.8.2.2.3 | Especificación de Relación con Actores | **26** |
| 28 | 1.1.8.2.2.4 | Clasificación de Actores por Paquete | **27** |
| 29 | 1.1.9 | Identificación de Responsables de Areas a Intervenir en el Proyecto | **16** |
| 30 | 1.1.10 | Elaboración de Plan de Trabajo | **29** |
| 31 | 1.1.11 | Autorización de Modelo del Negocio, Modelo de Requerimientos y Plan de Trabajo | **30** |
| 32 | 1.2 | **FASE ANALISIS** | **2** |
| 33 | 1.2.1 | Agenda de Reuniones para Fase | |
| 34 | 1.2.2 | Inducción a Nuevos Integrantes | **33** |
| 35 | 1.2.3 | Diagramas de Generalización de Casos de Uso | **34** |
| 36 | 1.2.4 | Diagramas de Extensión de Casos de Uso | **35** |
| 37 | 1.2.5 | Diagramas de Inclusión de Casos de Uso | **36** |

| | | | |
|---|---|---|---|
| 38 | 1.2.6 | Especificación de Casos de Uso | **37** |
| 39 | 1.2.6.1 | Entradas y Salidas | |
| 40 | 1.2.6.2 | Definición de Estándar de Interfaces | **39** |
| 41 | 1.2.6.3 | Especificación de Interfaces | **40** |
| 42 | 1.2.6.4 | Descripción de Casos de Uso | **41** |
| 43 | 1.2.6.5 | Autorización de Análisis | **42** |
| 44 | 1.3 | **FASE DISEÑO** | **32** |
| 45 | 1.3.1 | Agenda de Reuniones para Fase | |
| 46 | 1.3.2 | Inducción a Nuevos Integrantes | **45** |
| 47 | 1.3.3 | Identificación de Clases y Relaciones | **46** |
| 48 | 1.3.3.1 | Clases de Entidad | |
| 49 | 1.3.3.1.1 | Normalización de Base de Datos | |
| 50 | 1.3.3.2 | Clases de Interfaz | **48** |
| 51 | 1.3.3.3 | Clases de Control | **50** |
| 52 | 1.3.3.4 | Autorización de Clases | **51** |
| 53 | 1.3.3.5 | Diagramas de Estado | **52** |
| 54 | 1.3.3.6 | Diagramas de Secuencia | **53** |
| 55 | 1.3.3.7 | Diagramas de Colaboración | **54** |
| 56 | 1.3.3.8 | Diagramas de Actividades | **55** |
| 57 | 1.3.4 | Identificación de Métodos y Atributos | **47** |
| 58 | 1.3.5 | Diagramas de Componentes | **57** |
| 59 | 1.3.6 | Autorización de Diseño | **58** |
| 60 | 1.4 | **FASE CONSTRUCCION** | **44** |
| 61 | 1.4.1 | Agenda de Reuniones para Fase | |
| 62 | 1.4.2 | Definición de Estándar de Programación y Base de Datos | **61** |
| 63 | 1.4.3 | Inducción a Nuevos Integrantes | **62** |
| 64 | 1.4.4 | Preparación de Ambiente de Desarrollo | **63** |
| 65 | 1.4.5 | Construcción de la Base de Datos | **64** |
| 66 | 1.4.6 | Construcción | **65** |
| 67 | 1.5 | **FASE PRUEBAS** | **60** |
| 68 | 1.5.1 | Pruebas Unitarias Técnicas | |
| 69 | 1.5.1.1 | Elaboración de Check List de Prueba | |
| 70 | 1.5.1.2 | Elaboración de Casos de Prueba | **69** |
| 71 | 1.5.1.3 | Preparar Base de Datos con Casos de Prueba | **70** |
| 72 | 1.5.1.4 | Preparar Claves de Acceso y Perfiles de Usuario | **71** |
| 73 | 1.5.1.5 | Pruebas | **72** |
| 74 | 1.5.1.6 | Verificar Resultados | **73** |
| 75 | 1.5.1.7 | Ajustes en programación | **74** |
| 76 | 1.5.1.8 | VoBo de Pruebas Técnicas | **75** |
| 77 | 1.5.2 | Pruebas de Integrales, Volumen y Concurrencia Técnicas | **68** |
| 78 | 1.5.2.1 | Elaboración de Check List de Prueba | |
| 79 | 1.5.2.2 | Elaboración de Casos de Prueba | **78** |
| 80 | 1.5.2.3 | Preparar Base de Datos con Casos de Prueba | **79** |
| 81 | 1.5.2.4 | Preparar Claves de Acceso y Perfiles de Usuario | **80** |
| 82 | 1.5.2.5 | Pruebas | **81** |
| 83 | 1.5.2.6 | Verificar Resultados | **82** |
| 84 | 1.5.2.7 | Ajustes en programación | **83** |
| 85 | 1.5.2.8 | VoBo de Pruebas Técnicas | **84** |
| 86 | 1.5.3 | Agenda de Reuniones para Fase | **77** |
| 87 | 1.5.4 | Preparación | **86** |
| 88 | 1.5.4.1 | Elaboración de Plan de Pruebas | |
| 89 | 1.5.4.2 | Capacitación de Usuarios para Pruebas | **88** |
| 90 | 1.5.4.3 | Preparar Infraestructura de Pruebas | **89** |
| 91 | 1.5.4.4 | Realizar Pruebas de Conectividad | **90** |
| 92 | 1.5.4.5 | Preparar matrices de pruebas con resultados esperados | **91** |
| 93 | 1.5.4.6 | Preparar base de datos de prueba con la información requerida en las matrices | **92** |
| 94 | 1.5.4.7 | Preparar claves de acceso y perfiles de usuario para pruebas | **93** |
| 95 | 1.5.5 | Pruebas Unitarias Usuario | **87** |
| 96 | 1.5.5.1 | Elaboración de Check List de Prueba | |
| 97 | 1.5.5.2 | Elaboración de Casos de Prueba | **96** |
| 98 | 1.5.5.3 | Preparar Base de Datos con Casos de Prueba | **97** |
| 99 | 1.5.5.4 | Preparar Claves de Acceso y Perfiles de Usuario | **98** |
| 100 | 1.5.5.5 | Pruebas | **99** |
| 101 | 1.5.5.6 | Verificar Resultados | **100** |
| 102 | 1.5.5.7 | Ajustes en programación | **101** |

| | | | |
|---|---|---|---|
| 103 | 1.5.5.8 | Documentar resultados | **102** |
| 104 | 1.5.5.9 | VoBo de Pruebas Unitarias | **103** |
| 105 | 1.5.6 | Pruebas de Integrales, Volumen y Concurrencia Usuario | **95** |
| 106 | 1.5.6.1 | Elaboración de Check List de Prueba | |
| 107 | 1.5.6.2 | Elaboración de Casos de Prueba | **106** |
| 108 | 1.5.6.3 | Preparar Base de Datos con Casos de Prueba | **107** |
| 109 | 1.5.6.4 | Preparar Claves de Acceso y Perfiles de Usuario | **108** |
| 110 | 1.5.6.5 | Pruebas | **109** |
| 111 | 1.5.6.6 | Verificar Resultados | **110** |
| 112 | 1.5.6.7 | Ajustes en programación | **111** |
| 113 | 1.5.6.8 | Documentar resultados | **112** |
| 114 | 1.5.6.9 | VoBo de Pruebas Integrales | **113** |
| 115 | 1.6 | **FASE CAPACITACION** | **67** |
| 116 | 1.6.1 | Elaboración de Manual de Usuario y Técnico | |
| 117 | 1.6.2 | Elaborar y Autorizar Plan de Capacitación | **116** |
| 118 | 1.6.3 | Preparar Aulas y Ejercicios de Capacitación | **117** |
| 119 | 1.6.4 | Capacitar | **118** |
| 120 | 1.7 | **FASE DE LBERACIÓN** | **115** |
| 121 | 1.7.1 | Agenda de Reuniones para Fase | |
| 122 | 1.7.2 | Elaborar Plan de Liberación | **121** |
| 123 | 1.7.3 | Solicitud de Ambiente de Producción | **122** |
| 124 | 1.7.4 | Preparar Infraestructura | **123** |
| 125 | 1.7.5 | Instalar Software Operativo y Aplicativo | **124** |
| 126 | 1.7.6 | Construcción de la Base de Datos | **125** |
| 127 | 1.7.7 | Ejecutar Pruebas de Aceptación | **126** |
| 128 | 1.7.8 | Monitorear Funcionamiento | **127** |
| 129 | 1.7.9 | Documentar las Mejoras y Posibles Cambios al Sistema | **128** |
| 130 | 1.7.10 | Entrega de Producto al Area de Sistemas | **129** |
| **131** | **1.7.11** | **Autorización de Liberación** | **130** |

# CAPÍTULO V. ANÁLISIS Y DISEÑO DEL SISTEMA DE HISTORIA CLÍNICA ÓPTICA

## Modelo del Negocio

Después de platicar con el usuario su deseo es automatizar cada uno de los ciclos de negocio de una óptica, pero siendo su principal prioridad la automatización de la historia clínica.

Una óptica, como cualquier otro negocio, requiere mantener un control de sueldos y comisiones de sus empleados, controlar las compras apoyado en sus niveles de inventario y registrar las ventas realizadas a cada uno de los pacientes.

Obviamente, en la sesión inicial se platican mucho más detalles pero en todo momento debemos direccionar al usuario a concentrarse en enunciar los procesos del negocio y una breve descripción de los mismos.

### *Casos de Uso del Negocio*

Como resultado de nuestra reunión hemos detectados los siguientes procesos de negocio para convertirse en los casos de uso del negocio:

- Inventarios
- Administración de Personal
- Ventas
- Historia Clínica

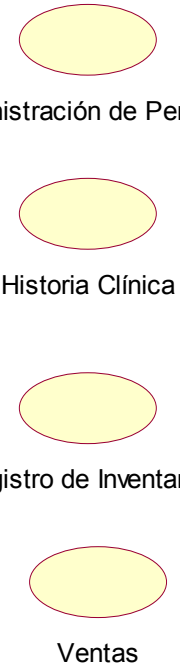
## Diccionario del Negocio

| Término | Descripción |
|---|---|
| Adición | Graduación especial par solucionar las presbicia |
| Agudeza Visual | Capacidad para ver distintamente los detalles de un objeto a 6 metros de distancia |
| Ajustar Armazón | Realizar cambios ligeros al armazón para lograr una mejor adaptación al rostro del paciente |
| Altura de Oblea | Distancia de la pupila a la zona inferior del lente para ubicar un bifocal |
| Anteojos | Complemento óptico que consta de un par de lentes oftálmicas colocadas sobre una montura, bien (sin aros que sujeten las lentes) descansando en la nariz y sujetas con varillas que se prolongan hasta las orejas |
| Antireflejante | Tratamiento químico realizado a los lentes para disminuir el grado de brillo recibido por el ojo. |
| Armazón | Estructura para sostener lentes (cristales o micas) y adaptarse a la cara humana |

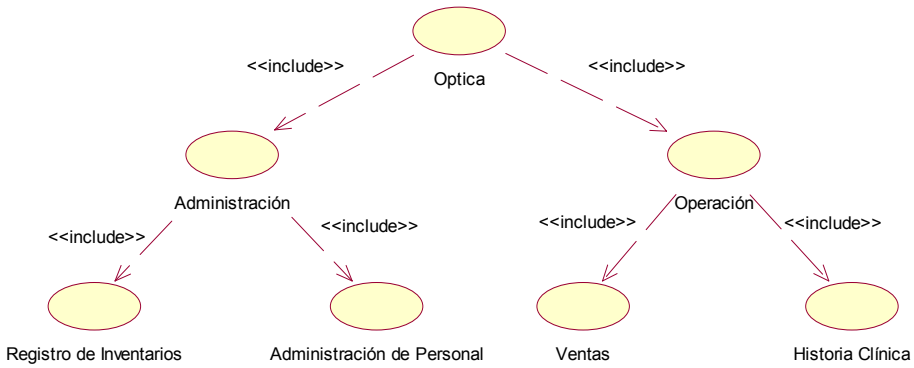
| | |
|---|---|
| Astigmatismo | Paciente con la córnea con una superficie más curva en una dirección comparada contra la dirección perpendicular |
| Capacidad Visual | Máxima capacidad para ver distintamente los detalles de un objeto a 6 metros de distancia sin problemas de refracción |
| Cilindro | Curvatura de una lente para solucionar el astigmatismo |
| Cliente | |
| Comisión | Bonificación en efectivo paga a un empleado de acuerdo a los objetivos de ventas |
| Contactología | Ciencia encargada de estudiar la refracción a través de lentes de contacto |
| Córnea | Es la ventana transparente del ojo por donde ingresa la luz refractando los rayos luminosos al interior del mismo |
| Cristalino | Lente elástica ajustable para lograr un enfoque correcto |
| Diagnóstico | Resultado de la salud visual del paciente |
| Eje | Línea que une los centros ópticos de las superficies refractivas del ojo |
| Esfera | Curvatura de una lente para solucionar la miopía o hipermetropía |
| Estuche | Accesorio para guardar los anteojos |
| Graduación | Es la combinación de una esfera, cilindro, eje y adición para solucionar el problema refractivo de un paciente |
| Hipermetropía | Paciente con el ojo más corto de lo habitual formando la imagen correcta por detrás de la retina |
| Historia Clínica | Registro histórico médico de un paciente en lo referente a su salud visual en este caso específico |
| Humor Acuoso | Líquido transparente proteínico situado en el globo ocular para lograr mantener su presión |
| Laboratorio | Lugar en donde se tallan lentes, se reparan armazones y se les de tratamientos químicos a los lentes |
| Lente | Cristales o micas transparentes a la luz con capacidad de ocasionar una convergencia o divergencia de la luz para solucionar problemas astigmatismo, hipermetropía y miopía |
| Lentes de Contacto | Lentes especiales montados sobre el globo ocular, son colocados y retirados por el mismo paciente |
| Líquido Limpiador | Líquido especial para limpiar los lentes |
| Miopía | Paciente con el ojo más largo de lo habitual formando la imagen correcta por delante de la retina |
| Oftalmología | Rama de la ciencia medica referente al tratamiento medico y quirúrgico del ojo y de sus accesorios |
| Oftalmoscopio | Instrumento para observar el interior y el fondo del ojo |
| Optica | Negocio especializado en solucionar problemas de refracción de la visión humana |
| Paciente | Persona con algún problema ocular |
| Padecimiento | Sufrimiento físico asociado a la salud de un paciente |
| Patología | Parte de la medicina que estudia las enfermedades |
| Plaqueta | Soporte de la armazón que cae sobre la nariz |
| Presbicia | Vista cansada provocada por la disminución de la característica elástica del cristalino. |
| Refracción | Visión ocular con un enfoque correcto de la imagen sobre la retina |
| Retina | Capa interna del ojo receptora de luz, constituida por una fina membrana transparente nerviosa |
| Retinoscopio | Instrumento para determinar objetivamente el estado refractivo del ojo |
| Solución de Lente de Contacto | Líquido limpiador de lentes de contacto para desinfectarlos de su uso diario. |
| Tensión Ocular | La presión en le globo ocular debida a la constante formación y drenaje del humor acuoso |

# Diagramación de Casos de Uso

Graficamos en primera instancia los casos de uso de los procesos del negocio:




Al tener listados nuestros casos de uso sabemos que todos ellos solucionan al negocio de una óptica, entonces podríamos realizar una inclusión de todos los casos de uso de negocio a un caso de uso de negocio de óptica. Si segmentamos el comportamiento de una óptica, tenemos una parte administrativa y otra parte relacionada a la operación del negocio, estas divisiones a pesar de no haber sido nunca mencionada por el usuario otorgarán un mayor nivel de clasificación e incluso apoyar para soportar futuros crecimientos. Como resultado de estas divisiones podemos representar en su conjunto la relación entre los casos de uso de negocio a través de la inclusión.




Al hablar de la inclusión en este diagrama entendemos la necesidad de funcionamiento de una óptica a través de una parte administrativa y operativa. La parte administrativa controlará el registro de inventarios y la administración del personal de la óptica. La parte operativa mantendrá el registro de la historia clínica de los pacientes y las ventas realizadas a los mismos. Debemos de tener cuidador en no confundir un proceso de negocio con requerimientos detallados particulares. Con el diagrama de inclusión podemos entender perfectamente el segmento de negocio en donde apoyaremos a nuestro usuario, además de lograr una visión integral del negocio.

# Especificación de Casos de Uso

| PROCESO DE NEGOCIO | OPTICA |
|---|---|
| ***OBJETIVO*** | Cuidar la salud visual de los pacientes ofreciendo solución a problemas de refracción y orientación a problemas patológicos.<br>Representar un negocio rentable a través de eficientar gastos sin sacrificar el servicio brindado a un paciente |
| ***IMPORTANCIA*** | Fuente de ingresos de empleados y dueño |
| ***RIESGOS*** | Disminución del nivel de servicio y por ende baja de número de pacientes |
| ***PRIORIDAD*** | Unica |
| ***RESTRICCIONES*** | Ninguna |
| ***FRECUENCIA DE EJECUCION*** | Lunes a Sábado |
| ***TIEMPO DE EJECUCION*** | 10 horas por día |
| ***COSTO DE EJECUCION*** | No calculado |
| ***USUARIO / ACTOR*** | Administrador, optometristas, pacientes |
| ***DESCRIPCION*** | Es el centro de atención a pacientes con problemas de refracción en la vista, solucionando el padecimiento a través de lentes oftálmicos.<br>Se realiza la venta de lentes montados sobre armazones, así como los accesorios inherentes al producto |

| PROCESO DE NEGOCIO | ADMINISTRACION |
|---|---|
| ***OBJETIVO*** | Control de los procesos administrativos |
| ***IMPORTANCIA*** | Ofrecer una base sólida para despreocupar a los procesos operativos |
| ***RIESGOS*** | Una mala administración acarreará problemas distrayendo el foco de la parte operativa |
| ***PRIORIDAD*** | Media |
| ***RESTRICCIONES*** | Ninguna |

| | |
|---|---|
| ***FRECUENCIA DE EJECUCION*** | Diario |
| ***TIEMPO DE EJECUCION*** | Depende del proceso |
| ***COSTO DE EJECUCION*** | No calculado |
| ***USUARIO / ACTOR*** | Administrador |
| ***DESCRIPCION*** | Administra el negocio a través del registro de inventarios de armazones, lentes y accesorios ópticos, así como el control y administración del personal de la óptica |

| **PROCESO DE NEGOCIO** | **REGISTRO DE INVENTARIOS** |
|---|---|
| ***OBJETIVO*** | Controlar los inventarios de armazones, lentes y accesorios ópticos |
| ***IMPORTANCIA*** | Tener alternativas de selección de productos para el paciente y disponibilidad de lentes para tener mejores tiempos de respuesta al enviar trabajos a los laboratorios |
| ***RIESGOS*** | Limitar las opciones de selección de producto al paciente y retardar el tiempo de entrega de los lentes a los pacientes |
| ***PRIORIDAD*** | Media |
| ***RESTRICCIONES*** | Ninguna |
| ***FRECUENCIA DE EJECUCION*** | Cuando se requiera |
| ***TIEMPO DE EJECUCION*** | Depende del proceso |
| ***COSTO DE EJECUCION*** | No calculado |
| ***USUARIO / ACTOR*** | Administrador |
| ***DESCRIPCION*** | Apoyar a mantener niveles de inventario óptimos para ofrecer una variedad amplia de armazones para diferentes tipos de paciente según sexo y edad. También controla la variedad requerida de accesorios ópticos como tornillos, estuches, plaquetas, líquidos limpiadores, soluciones para lentes de contacto, etc.<br>Controla el inventario de diferentes tipos de lentes de las graduaciones más comunes, apoyado en procesos estadísticos de las graduaciones más requeridas |

| **PROCESO DE NEGOCIO** | **ADMINISTRACION DE PERSONAL** |
|---|---|
| ***OBJETIVO*** | Controla todos los procesos administrativos referentes al personal de la óptica tales como: sueldos y comisiones, aumentos de sueldo, capacitación, rentabilidad, control de altas y bajas de personal, etc. |
| ***IMPORTANCIA*** | Conocer el estatus histórico de los empleados para toma de decisiones, así como el pago de comisiones y sueldos |
| ***RIESGOS*** | Mala administración del personal de la óptica creando distracción en los empleados |
| ***PRIORIDAD*** | Media |
| ***RESTRICCIONES*** | Ninguna |
| ***FRECUENCIA DE EJECUCION*** | Cuando se requiera |
| ***TIEMPO DE EJECUCION*** | Depende del proceso |
| ***COSTO DE EJECUCION*** | No calculado |
| ***USUARIO / ACTOR*** | Administrador |
| ***DESCRIPCION*** | Realiza el cálculo de pago de sueldos quincenales en conjunto con las comisiones obtenidas por los optometristas en base a sus metas quincenales.<br>Controla prestaciones, asistencias, vacaciones, días sin goce de sueldo, etc. de cada uno de los empleados.<br>Apoya el seguimiento de un plan de carrera profesional de los optometristas.<br>Da seguimiento a la rentabilidad de cada uno de los optometristas considerando sus ventas, sueldos, gastos de materiales y recursos, gastos de capacitación, etc.<br>Realiza el proceso de alta y baja de empleados dependiendo del tipo de esquema de pago, dentro de la baja se considera el cálculo liquidación. |

| PROCESO DE NEGOCIO | OPERACIÓN |
|---|---|
| ***OBJETIVO*** | Dar un buen nivel de servicio a los pacientes de la óptica a través de procesos efectivos de ventas y registros de histórica clínica |
| ***IMPORTANCIA*** | Es donde se generan los ingresos de la óptica |
| ***RIESGOS*** | Una mala operación afectará los ingresos |
| ***PRIORIDAD*** | Media |
| ***RESTRICCIONES*** | Ninguna |
| ***FRECUENCIA DE EJECUCION*** | Lunes a Sábado |
| ***TIEMPO DE EJECUCION*** | 10 horas por día |
| ***COSTO DE EJECUCION*** | No calculado |
| ***USUARIO / ACTOR*** | Optometrista, Paciente |
| ***DESCRIPCION*** | Controla las ventas realizadas a los pacientes y el registro de las historias clínicas de los mismos |

| PROCESO DE NEGOCIO | VENTAS |
|---|---|
| ***OBJETIVO*** | Ofrecer las diferentes alternativas de elección adecuadas para un paciente, como tipos de armazones, tipos de lentes, accesorios, etc. |
| ***IMPORTANCIA*** | Guiar al paciente a realizar la mejor elección de sus compras |
| ***RIESGOS*** | Malas ventas o pacientes molestos por productos no idóneos a sus necesidades |
| ***PRIORIDAD*** | Media |
| ***RESTRICCIONES*** | Ninguna |
| ***FRECUENCIA DE EJECUCION*** | Por cada paciente atendido en horario de servicio |
| ***TIEMPO DE EJECUCION*** | Depende del paciente |
| ***COSTO DE EJECUCION*** | No calculado |
| ***USUARIO / ACTOR*** | Optometrista, Paciente |
| ***DESCRIPCION*** | Guiar al paciente para realizar una compra de acuerdo a sus necesidades de los productos requeridos para solucionar sus problemas de refracción. Dependiendo del problema refractivo se deberá recomendar el uso de cierto tipo de lentes (micas o cristales) acompañados de tratamientos especiales a los lentes (antireflejante, micas teñidas, etc), armazón recomendada de acuerdo a su tipo y tamaño de cara, así como armazones idóneas para el tipo de lente seleccionado, entre otras tantas cosas.<br>Seguimiento de trabajos, desde soldaduras hasta lentes con armazones para indicar a paciente fecha de entrega compromiso, envío a laboratorio, etc. |

| PROCESO DE NEGOCIO | HISTORIA CLINICA |
|---|---|
| ***OBJETIVO*** | Controlar la historia clínica de los pacientes de la óptica para conocer sus antecedentes clínicos en sus subsecuentes visitas |
| ***IMPORTANCIA*** | Dar un diagnóstico más preciso al paciente por tener de manera rápida sus antecedentes clínicos |
| ***RIESGOS*** | Tener diagnósticos menos eficientes |
| ***PRIORIDAD*** | Alta |
| ***RESTRICCIONES*** | Sólo se considera para pacientes con problemas visuales, no aplica para pacientes con descompostura de armazón y venta de accesorios. |
| ***FRECUENCIA DE EJECUCION*** | Por cada paciente con problemas visuales |
| ***TIEMPO DE EJECUCION*** | Depende del paciente |
| ***COSTO DE EJECUCION*** | No calculado |
| ***USUARIO / ACTOR*** | Optometrista, paciente |
| ***DESCRIPCION*** | Registra la información general de un paciente, sus padecimientos actuales, historia médica general, y examen de la vista. |

En algunas ocasiones no se cuenta con toda la información requerida en el formato de proceso de negocio, implicando la omisión de la misma o contestar el formato con las causas de la falta de

información. Un ejemplo claro de lo anterior fue el llenado del formato para el proceso de negocio de Administración, en el renglón de tiempo de ejecución la respuesta fue "*Depende del proceso*", aunque la respuesta pareciera obligar a indicar un tiempo, es necesario tener la habilidad de no forzar a contestar preguntas sin tener detalles, evitando entrar en respuestas complejas para calcular el tiempo o incluso respuestas con la posibilidad de confusión.

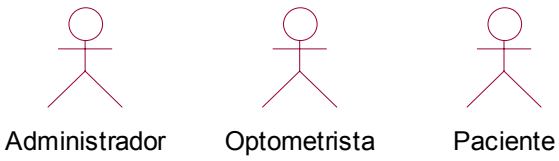
En la descripción de los procesos del negocio se debe indicar la descripción del proceso del negocio y nunca la lista de requerimientos, puesto que esta es más limitada. También se podrán considerar nuevas actividades a incorporarse en un futuro ya sea por la automatización informática o por cambios próximos.

Como nuestro caso de uso de negocio es el de Historia Clínica el nivel de detalle requerido será mucho mayor al momento de especificarlo, comparado con los otros casos de uso de negocio.

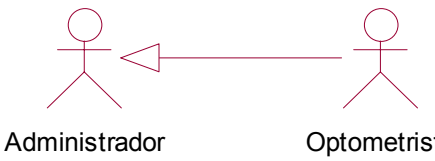
## *Diagramas de Roles*

El diagrama de roles lo haremos de todos los procesos del negocio por su relativa sencillez, pero si fuera complejo podríamos realizarlo de los roles con mayor grado de participación dentro de nuestro caso de uso de negocio a solucionar.

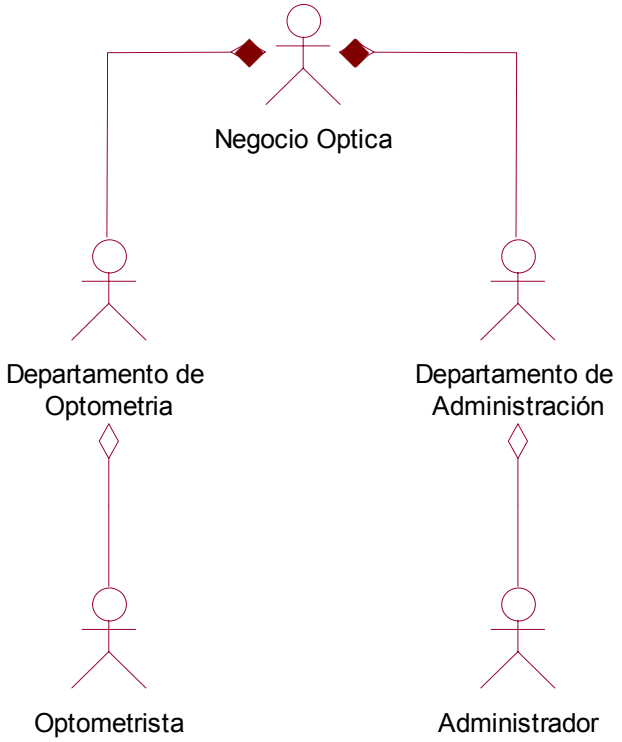
La relación jerárquica entre los empleados, en donde el administrador es el responsable del negocio, los optometristas responsables de atender a los pacientes y el paciente es el cliente de la óptica.



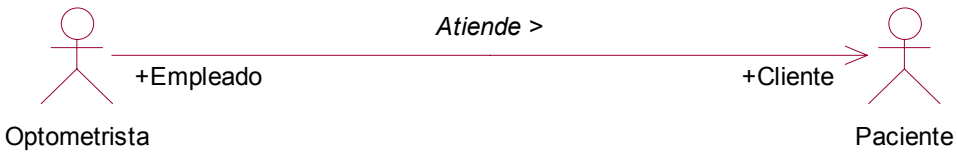

Para plasmar la relación de organigrama, en donde el administrador es el jefe de un optometrista tenemos:



Ahora buscaremos las relaciones de composición y agregación si las existiera, y aunque no hemos hablado de departamentos, la estructura organizacional la podríamos consultar en este momento con el usuario, para conocer la integración de los mismos quedando de la siguiente manera:




También nos hacen falta buscar asociaciones, la de un optometrista con un administrador queda expresada de manera implícita en la generalización, quedando pendiente sólo la del optometrista con el paciente:




El diccionario de roles queda definido de la siguiente manera:

*Administrador*: Responsable de controlar los procesos administrativos de la óptica, así como estar al pendiente de los niveles de servicio otorgados al paciente.

*Optometristas*: Profesional responsable de solucionar los problemas refractivos del paciente a través de realizar las ventas de lentes y accesorios para corregir el problema

*Paciente*: Cliente de la óptica, comprador de accesorios para sus lentes y revisión de sus problemas refractivos.

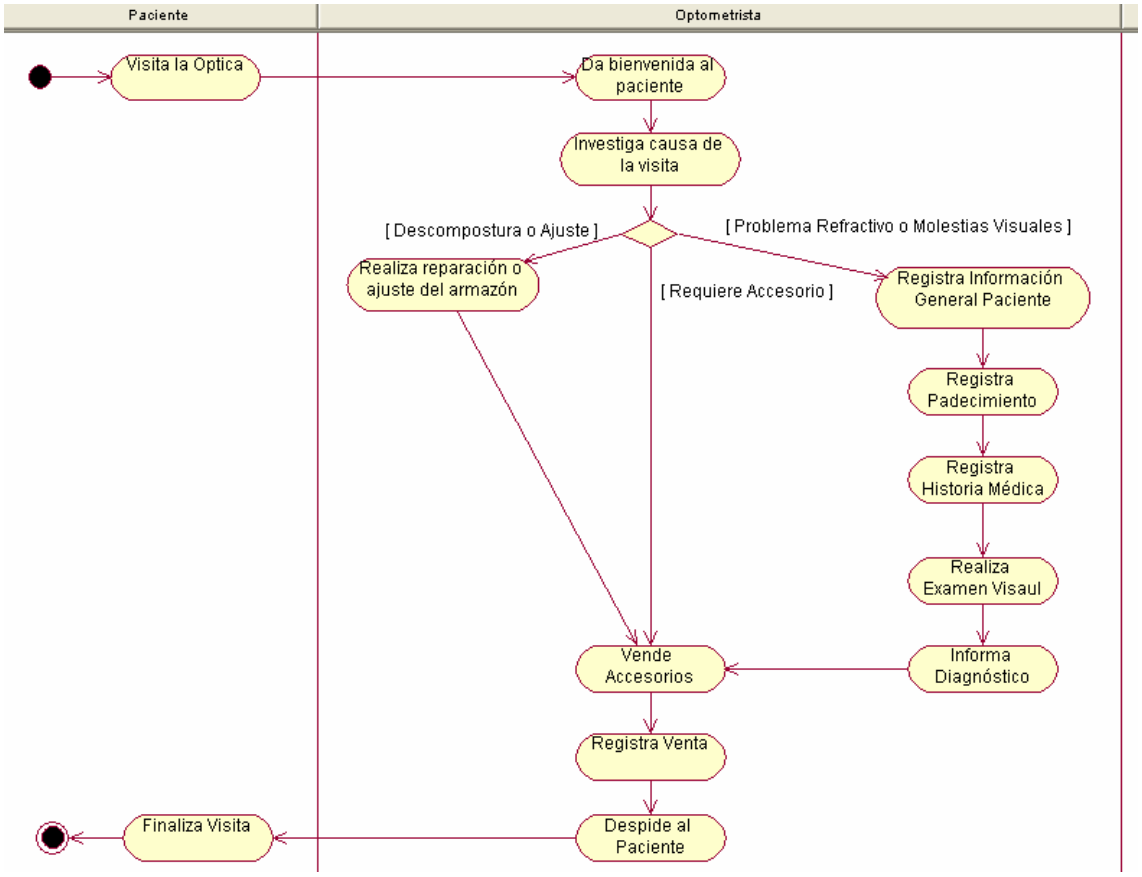
*Negocio Optica*: Es el negocio especializado en dar solución a problemas refractivos de los pacientes.

*Departamento de Optometría*: Es el departamento responsable de coordinar a los optometristas de una óptica

*Departamento de Administración*: Es el departamento responsable de la coordinación administrativa de la óptica.

## *Diagramas de Procesos*

Para esta situación en particular sólo analizaremos el flujo de los procesos del caso de uso de negocio de interés, enfocandonos a partir de este momento en el conocimiento más detallado del caso de uso de negocio al que daremos solución.



Aunque se trató de concentrarse exclusivamente en el registro de la historia clínica, resultó interesante conocer el proceso de negocio de venta como parte del ciclo de visita de un paciente, pero sin ahondar en tantos detalles del proceso de venta. El proceso de registro de historia clínica se inicia con el registro de información general del paciente y concluye con el informe del diagnóstico verbal del resultado de su revisión.

El diccionario de actividades queda integrado de la siguiente manera:

*Visita Optica*: El paciente entra a la óptica para ser atendido.

*Da Bienvenida al Paciente*: El optometrista libre de atender a algún paciente da la bienvenida al paciente recién llegado. Si no existen optometristas disponibles cualquier optometrista de la bienvenida y le pide aguardar.

*Investiga Causa de la Visita*: El optometrista averigua el origen de la visita del paciente.

*Realiza Reparación o Ajuste del Armazón*: Corrige la descompostura del armazón si es viable en ese momento, en caso contrario se retiene la armazón para enviarla al laboratorio.

*Registra Información General del Paciente*: Recaba información genérica del paciente para su registro.

*Registra Padecimiento*: Se registra la información del padecimiento actual del paciente, así como actividades habituales del paciente. Se consultan padecimientos anteriores del paciente si se tiene registro.

*Registra Historia Médica*: Se actualiza la información de antecedentes clínicos del paciente con relación médica directa a su salud visual.

*Realiza Examen Visual*: Realiza las diferentes pruebas visuales, para conocer la refracción visual del paciente y registra el resultado del examen visual.

*Informa Diagnóstico*: Comunica al paciente de manera verbal el diagnóstico visual, como resultado de la información de la historia médica, padecimientos y examen visual.

*Vende Accesorios*: Se realiza la venta de accesorio, lentes, armazones, líquido limpiador de cristales, líquido para lentes de contacto, etc.

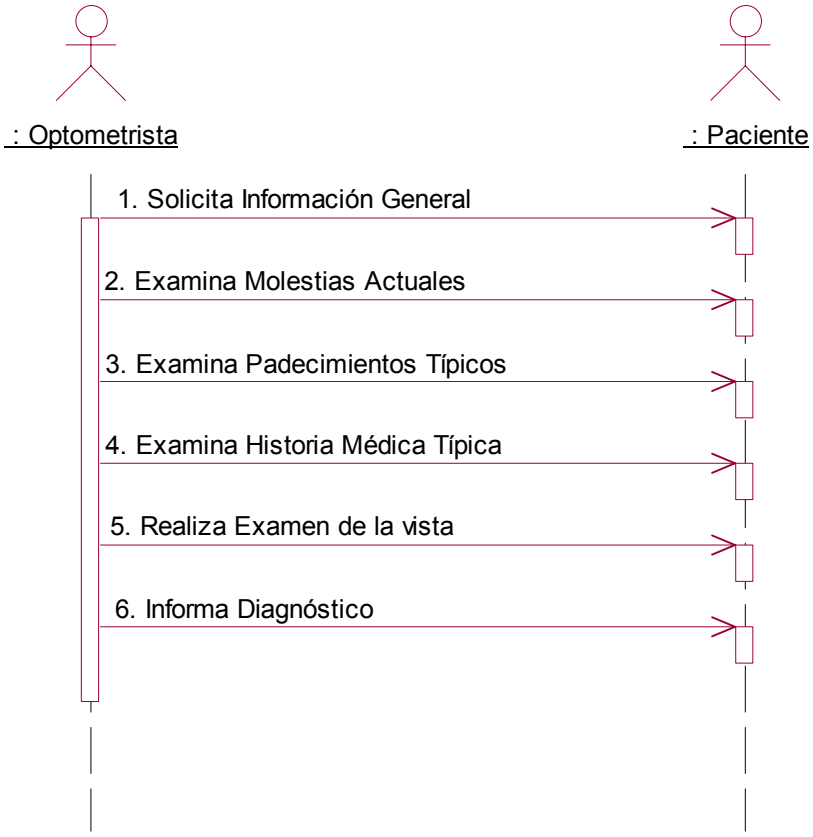
*Registra Venta*: Registra la venta realizada al paciente por número de accesorios, tipo de accesorio y precio.

*Despide al Paciente*: El optometrista da las gracias por la visita del paciente y pone la óptica a su disposición para próximas visitas.

*Finaliza Visita*: El paciente concluye su visita a la óptica.

## *Diagrama de Escenarios*

El diagrama de escenarios producto de nuestro caso de uso de negocios muestra una interacción de baja complejidad, siendo tal vez poco interesante realizar un diagrama, pero a pesar de eso se documentará para mostrar la manera de solucionar un diagrama de escenarios.




*Solicita Información General*: El optometrista solicita el nombre completo del paciente y su dirección.

*Examina Molestias Actuales*: El optometrista cuestiona y revisa al paciente con referencia al padecimiento actual.

*Examina Padecimientos Típicos*: El optometrista cuestiona y revisa al paciente sobre padecimientos típicos.

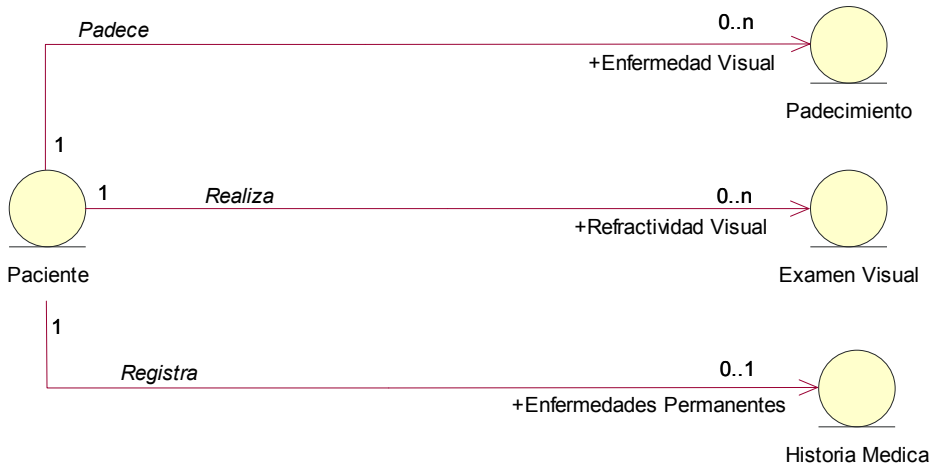
*Examina Historia Médica Típica*: El optometrista cuestiona al paciente sobre padecimientos indirectos con posibilidad de afección a la refracción.

*Realiza Examen de la Vista*: El optometrista examina al paciente a través de un examen de la vista.

*Informa Diagnóstico*: El optometrista informa al paciente del diagnóstico después de tener información del padecimiento actual, historia médica y el examen de la vista.

# Modelo del Dominio

Dentro del dominio del negocio podemos identificar algunas clases para ayudarnos a ir aterrizando gradualmente la concepción del modelo estático del sistema.




Un paciente padece cero o n padecimientos, un paciente realiza cero o n exámenes visuales y un paciente registra cero o una historia médica. Aunque sea un modelo simple nos ayuda a entender las principales relaciones del vocabulario del dominio del problema. En general el diagrama preeliminar de objetos estará enfocado a clases de entidad, y nunca debemos tratar de llegar a ofrecer el modelo de objetos del sistema, esta será la primera aproximación formal al mismo.

# Modelo de Requerimientos

## *Especificación de Requerimiento Global y Detallados*

La especificación del Requerimiento Global es la misma referente al proceso del negocio, pero si no se obtiene el modelo de negocio al iniciar esta fase, será necesario capturar el requerimiento global en este punto.

Los requerimientos detallados en general corresponden al diagrama de procesos efectuado en el modelo del dominio del negocio, es por eso que derivaremos de este los nombres de los requerimientos.

| REQUERIMIENTO | Registro de Información General del Paciente |
|---|---|
| ***OBJETIVO*** | Mantener el registro de pacientes de la óptica |
| ***IMPORTANCIA*** | Identificar de manera muy rápida a pacientes atendidos |
| ***RIESGOS*** | Contar un registro lento de pacientes y capturar información incorrecta |
| ***PRIORIDAD*** | Alta |
| ***RESTRICCIONES*** | Sólo registrar a pacientes a efectuarse una revisión óptica |
| ***FRECUENCIA DE EJECUCION*** | Por cada paciente |
| ***TIEMPO DE EJECUCION*** | Menor a un minuto |
| ***COSTO DE EJECUCION*** | |
| ***USUARIO / ACTOR*** | Optometrista |
| ***DESCRIPCION*** | 1. Registrar nombre completo del paciente y dirección |

| REQUERIMIENTO | Registro de Padecimiento |
|---|---|
| ***OBJETIVO*** | Tener un histórico de los padecimientos presentados por un paciente |
| ***IMPORTANCIA*** | Brindar una evaluación óptica más precisa en base a padecimientos presentados |
| ***RIESGOS*** | Mal registro de la información |
| ***PRIORIDAD*** | Alta |
| ***RESTRICCIONES*** | Sólo registrar a pacientes a efectuarse una revisión óptica |
| ***FRECUENCIA DE EJECUCION*** | Por cada paciente |
| ***TIEMPO DE EJECUCION*** | De 5 a 10 minutos |
| ***COSTO DE EJECUCION*** | |
| ***USUARIO / ACTOR*** | Optometrista |
| ***DESCRIPCION*** | 1. Registrar la información del padecimiento actual, padecimientos comunes relacionados a problemas de visión y entorno ambiental del paciente |

| REQUERIMIENTO | Registro de Historia Médica |
|---|---|
| ***OBJETIVO*** | Contar con la historia médica del paciente para ofrecer una mejor solución a sus problemas refractivos |
| ***IMPORTANCIA*** | Conducir la evaluación óptica en base a su historia médica |
| ***RIESGOS*** | Mal registro de la información generando ofrecer un solución visual incorrecta |
| ***PRIORIDAD*** | Alta |
| ***RESTRICCIONES*** | Sólo registrar a pacientes a efectuarse una revisión óptica |
| ***FRECUENCIA DE EJECUCION*** | Por cada paciente |
| ***TIEMPO DE EJECUCION*** | De 5 a 10 minutos |
| ***COSTO DE EJECUCION*** | |
| ***USUARIO / ACTOR*** | Optometrista |
| ***DESCRIPCION*** | 1. Registrar la información de la historia médica del paciente con relación directa a problemas de oculares<br><br>Nota: La historia médica es una por paciente y se actualiza por cada visita. |

| REQUERIMIENTO | Registro de Examen Visual |
|---|---|
| ***OBJETIVO*** | Tener el registro de su examen visual para futuras visitas |

| | |
|---|---|
| ***IMPORTANCIA*** | Registro preciso de su graduación |
| ***RIESGOS*** | Si un paciente requiere hacer nuevamente sus lentes de contacto o lentes con poco tiempo después de realizarse el examen se tomaría como base la graduación almacenada |
| ***PRIORIDAD*** | Alta |
| ***RESTRICCIONES*** | Sólo registrar a pacientes a efectuarse una revisión óptica. |
| ***FRECUENCIA DE EJECUCION*** | Por cada paciente |
| ***TIEMPO DE EJECUCION*** | De 10 a 25 minutos |
| ***COSTO DE EJECUCION*** | |
| ***USUARIO / ACTOR*** | Optometrista |
| ***DESCRIPCION*** | 1. Registrar la agudeza visual y la capacidad visual<br>2. Registrar la graduación de sus lentes actuales (exteriores o de contacto) además de la agudeza visual con los mismos<br>3. Registrar información de contactología en caso de requerirse lentes de contacto<br>4. Registrar el resultado de la oftalmoscopia en caso de encontrarse un padecimiento<br>5. Registrar el resultado de la retinoscopía<br>6. Registrar el resultado de la refracción<br>7. Registrar la medición de la tensión ocular |

Algo importante de resaltar es que donde se indica al actor, el paciente ya no se está considerando, esto por ser más preciso en el requerimiento, en donde se ofrece una herramienta de registro al optometrista y el paciente mantiene una relación directa con el optometrista.

El tiempo de ejecución de cada requerimiento involucra el tiempo dedicado a interactuar con el paciente, así como el tiempo de registro de información, esta información nos sirve en diferentes sentidos, por ejemplo, se entiende que la captura de la información no será continua por dedicar un tiempo a examinar al paciente y posteriormente registrar, por eso el diseño de interface deberá ser de fácil acceso para identificar el siguiente punto a registrar. En algunas otras ocasiones las interfaces deberán dar prioridad a la elección de reducción de tiempo de captura, por ejemplo en un call center.

# *Diagramas de Casos de Uso*

Aunque la diagramación de casos de uso se mostrará en etapas para llegar a un diagrama final, es recomendable integrar en la documentación final el diagrama final de los casos de uso, ya que el diagrama final es el de interés para la documentación y no la secuencia lógica de cómo se obtuvo.

## Casos de Uso Funcionales

En base a los requerimientos cada requerimiento se convierte en la representación de un caso de uso.

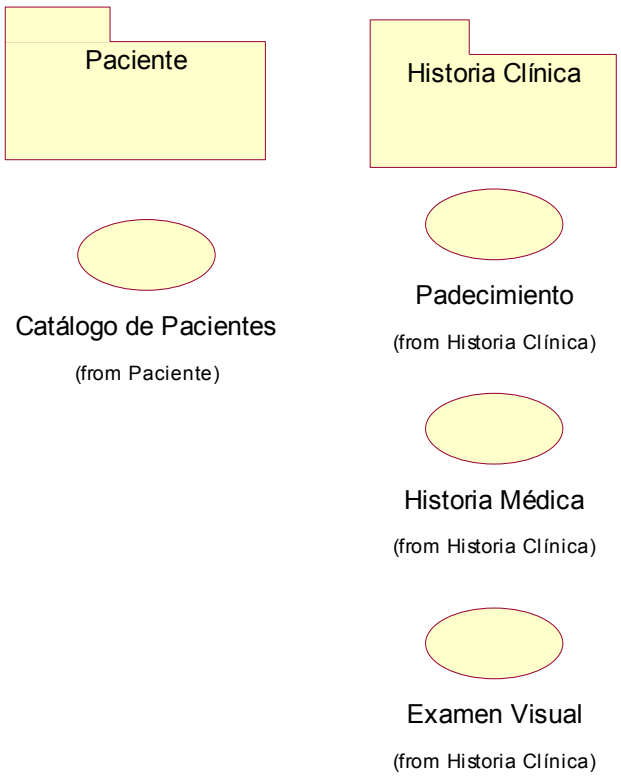
### *Diagramación de Casos de Uso*



En este punto se genera la plantilla de caso de uso, capturando el nombre del caso de uso, objetivo, precondiciones y postcondiciones. Más adelante se encontrará la plantilla de cada caso de uso completa.

## *Clasificación de Casos de Uso*

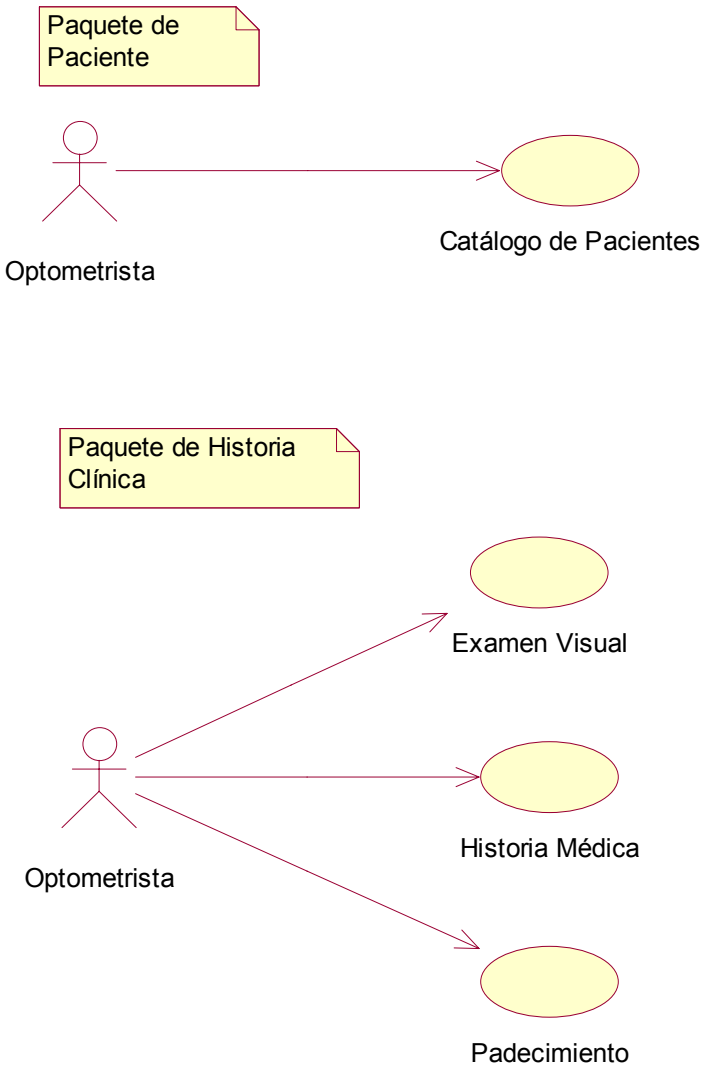
Los casos de uso los podemos clasificar en base a los futuros módulos del sistema en dos: historia clínica y paciente, aunque el paquete de paciente actualmente sólo clasificará un caso de uso al pensar en la futura expansión del sistema, el paquete pudiera incorporar más casos de uso y agregarse a este paquete. Otra opción pudo ser no clasificar los casos de uso por considerarse pocos.




En la plantilla de caso de uso debemos indicar por cada caso de uso el paquete al que pertenece.

## *Especificación de Relación con Actores*

Buscamos las relaciones entre actores y casos de uso.



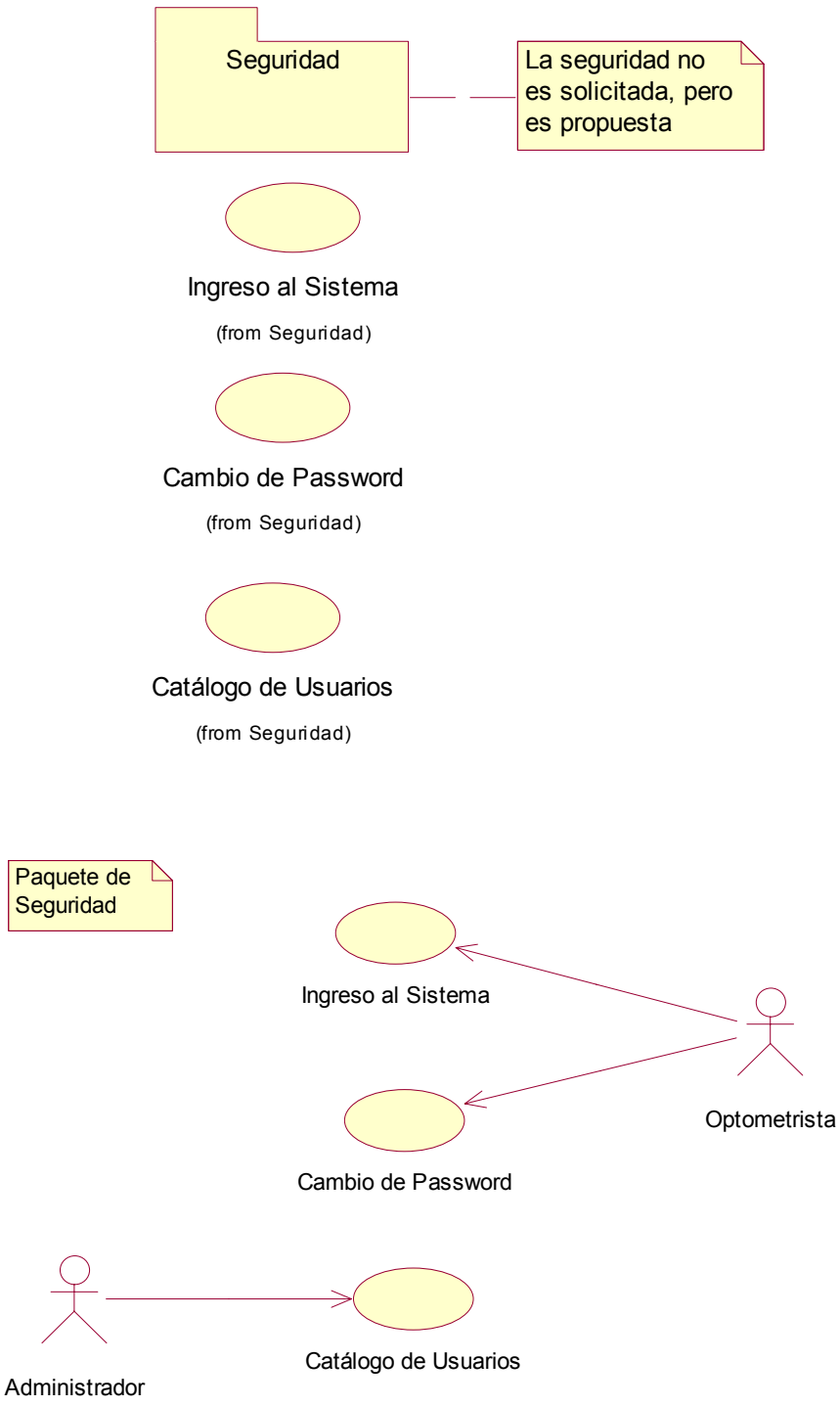

Se debe indicar en la plantilla de caso de uso la relación de cada caso de uso con actores.

### *Clasificación de Actores por Paquete*

Al tener un solo actor no es necesario clasificarlo en ningún paquete.

## Casos de Uso No Funcionales

La seguridad no forma parte de los requerimientos funcionales del sistema, pero ayudará a mantener un control de acceso sobre la aplicación. En base a lo anterior se adiciona un nuevo paquete denominado seguridad con los siguientes casos de uso.




Los casos de uso no funcionales deben seguir la misma lógica de tratamiento a la de los casos de uso funcionales.

# Análisis

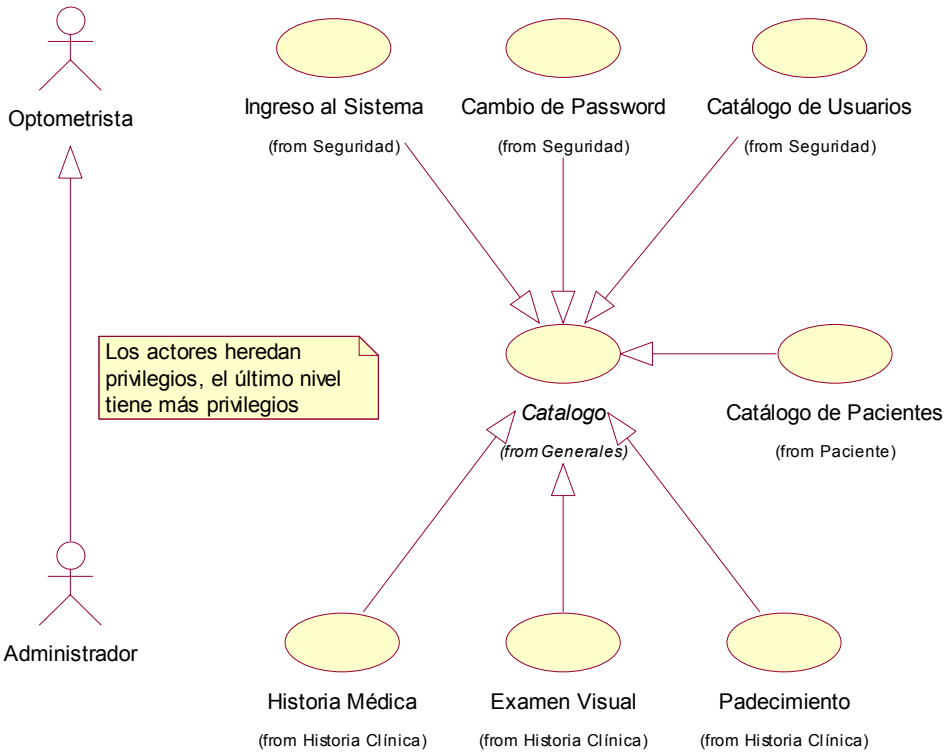
Al concluir el modelo de requerimientos ya podemos entrar a realizar el análisis de los requerimientos para extraer las necesidades detalladas. Al ingresar a la etapa de análisis ya podemos ir perfilando las posibles decisiones de diseño, pero sin querer diseñar durante el análisis.

## *Diagrama de Generalización de Casos de Uso*

Como parte de la generalización es momento de buscar los comportamientos comunes entre casos de uso por un lado y actores por otro. Todos los casos de uso hasta este punto manejan un interface, también todas las interfaces tendrán una funcionalidad similar por la forma de comportarse, si pensamos en una interface de características de catálogo para dar altas, bajas, cambios y consultas todas las interfaces se lograrían satisfacer, excepto tal vez los casos de uso de ingreso al sistema y cambio de password (más adelante se anexará a esta excepción la historia médica, pero de momento pudiera no parecer obvio). Las excepciones de comportamiento de interface de catálogo pueden comportarse como un catálogo, limitando su funcionalidad a contar con un tipo de operación, de esta manera podemos ver las excepciones como una variante de la funcionalidad de catálogo.

Para manejar la seguridad dependiendo del actor a ingresar al sistema tendremos dos perfiles: administrador y optometrista, el optometrista tendrá ingreso limitado a las funciones del sistema, pero el administrador tendrá las funciones del optometrista más las funciones restantes.

Como resultado de lo anterior tenemos:

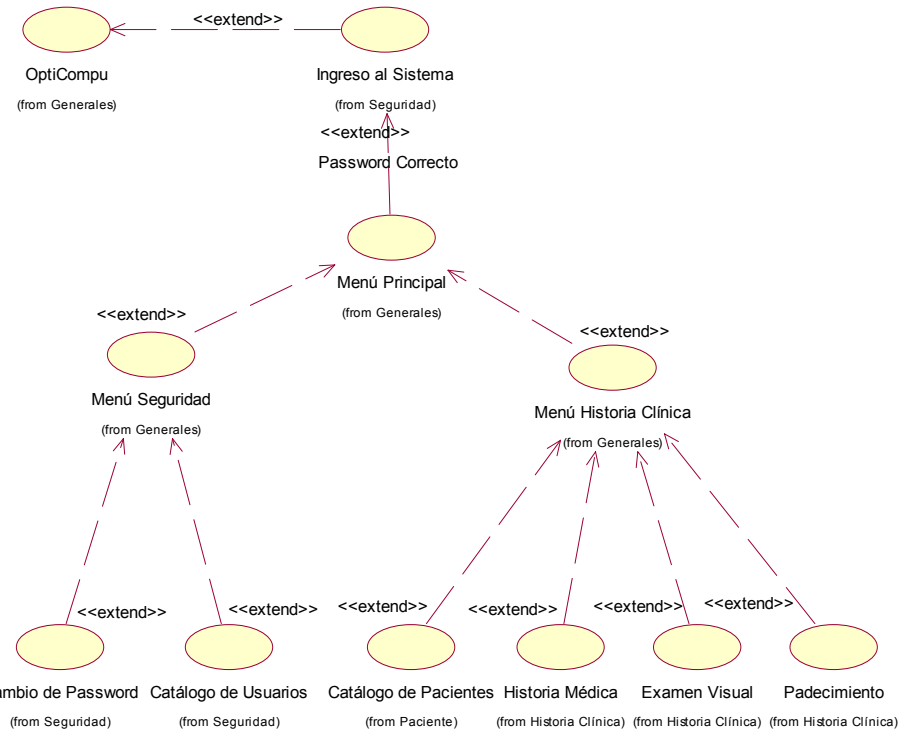



Al adicionar un nuevo caso de uso para satisfacer los comportamientos de las interfaces de los casos de uso también se adiciona un paquete llamado *Generales*, siguiendo la lógica de paquetes empleada hasta este momento. Después de agregar el caso de uso de catálogo debemos generar su plantilla de caso de uso y actualizarla al mismo punto adelantado por todos los casos de uso en ese momento.

Indicar en las plantilla de casos de uso el segmento de generalizaciones.

## *Diagrama de Extensión de Casos de Uso*

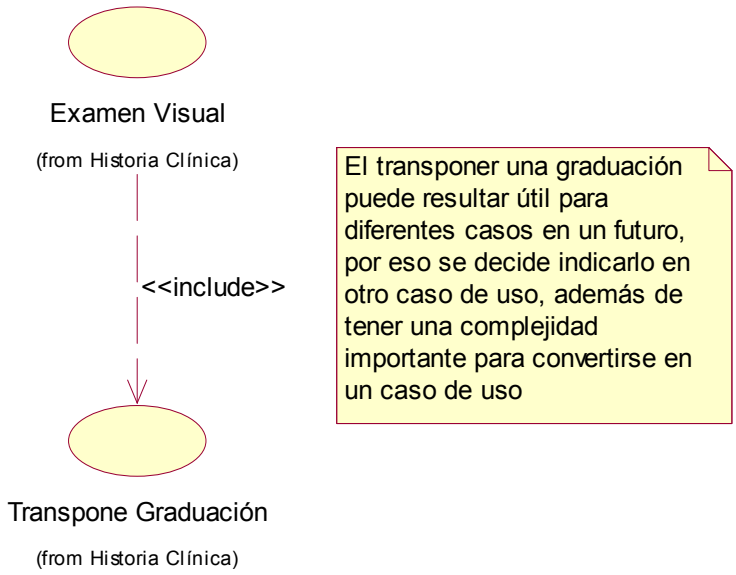
Las extensiones son muy útiles para representar los menús de un sistema, para esto generaremos nuevos casos de uso para representarlos. Puesto que los menús tienen una funcionalidad y complejidad extremadamente baja no generaremos plantilla de caso de uso para los mismos.




Indicar en las plantillas de casos de uso el segmento de extensiones.

## *Diagrama de Inclusión de Casos de Uso*

Al buscar inclusiones pensamos en factorizar algunos funcionamientos, en este caso no se presentan. Otro punto para buscar inclusiones, son comportamientos dentro de un caso de uso con la suficiente complejidad para ser analizados independientes o casos de uso para satisfacer a más de un caso de uso, aunque en este momento sólo logren satisfacer a uno.




La transposición de graduación es la conversión de una graduación con un cilindro positivo en una graduación con cilindro negativo, esta acción se realiza por ser el estándar en la industria óptica. El caso de uso de transponer graduación será enviado al paquete de historia clínica, por tratarse de un caso de uso para calcular una graduación.

Indicar en las plantillas de casos de uso el segmento de inclusiones.

## *Especificación de Casos de Uso*

Se recomienda seguir la secuencia de indicar las entradas y salidas, diseño de interface y por último la descripción. Si al momento de especificar se detecta algún cambio a los diagramas, es muy válido realizarlo, más aún porque en esta parte es donde se detalla la necesidad del usuario.

| **PAQUETE** | **CASO DE USO** | OBJETIVO |
|---|---|---|
| GENERALES | CATALOGO | Controlar el funcionamiento de cualquier interface con características de catálogo |

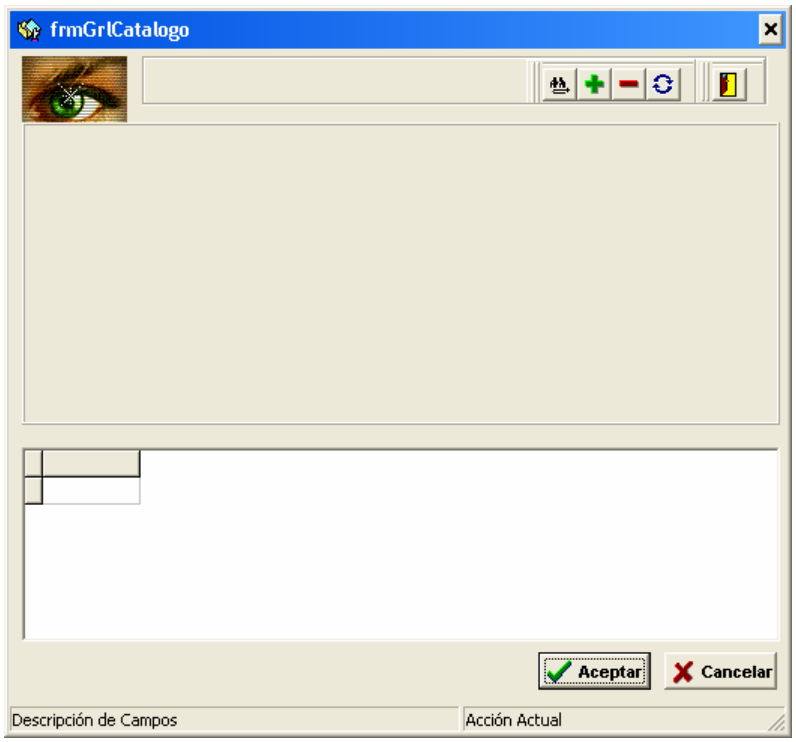
| | |
|---|---|
| Precondiciones | |
| Postcondiciones | Alta, baja, cambio y/o consulta, o proceso |
| Actores | |
| Generaliza de | |
| Extiende a | |
| Incluye a | |
| Entradas | Catálogo: Campos del catálogo<br>Proceso: Campos para iniciar el proceso |
| Salidas | Catálogo: Campos del catálogo<br>Proceso: Campos resultado del proceso |
| Dudas | |

| PROCESO/EVENTO | DESCRIPCIÓN |
|---|---|
| E: Cargar Forma | 1. P (Inicializa forma)<br>2. P (Prepara Lectura de la Opción) |

| | |
|---|---|
| | 3. Si es de tipo proceso<br>1.1. Sólo muestra opción de salir en la barra de opciones<br>1.2. No muestra el grid<br>1.3. Habilita captura de campos |
| E: Click Barra de Opciones | 1. Click Salir: Sale de la forma<br>2. Click Alta: P (Habilita Opción (Alta))<br>3. Click Baja: P (Habilita Opción (Baja))<br>4. Click Actualización: P (Habilita Opción (Actualización))<br>5. Click Consulta: P (Habilita Opción (Consulta)) |
| E: Click Aceptar | 1. Si P(Valida Datos)<br>1.1. Si es de tipo proceso ejecuta P (Proceso)<br>1.2. En caso contrario<br>1.2.1. Ejecuta el tipo de operación seleccionado previamente<br>1.3. P (Realiza Consulta)<br>1.1.1. Si es alta realiza la consulta del nuevo registro<br>1.1.2. Si es baja o actualización repite la última consulta<br>1.1.3. Si es consulta aplica los criterios de búsqueda indicados en pantalla<br>2. Si los datos no son válidos notifica con un mensaje<br>En caso de error regresa al estatus previo al dar click en el botón de aceptar. |
| E: Click Cancelar | 1. Posiciona el detalle del primer renglón del grid en los campos correspondientes<br>2. P (Prepara Lectura Opción) |
| E: Click Grid | 1. Posiciona el detalle del renglón seleccionado del grid en los campos correspondientes |
| E: Entrar a un campo de captura | 1. Enviar descripción larga del campo con el foco actual a la barra de estado |
| E: Salir de campo de captura | 1. Si el campo es de tipo texto elimina blancos iniciales y finales |
| P: Inicializa Forma | 1. P (Carga Catálogos Iniciales)<br>2. Formatea grid<br>3. P (Inicializa Campos) |
| P: Prepara Lectura de la Opción | 1. Limpia las barras de estado de operación y descripción de campo<br>2. Habilita barra de opciones<br>3. Habilita grid<br>4. Si es de tipo catálogo<br>1.1. Habilita botones de acción<br>1.2. Deshabilita captura de campos<br>2. Si es de tipo proceso<br>2.1. Habilita botones de acción |
| P: Habilita Opción (Operación) | 1. Si operación es consulta o alta P (Inicializa Campos)<br>2. Dependiendo del tipo de operación inicializa los campos de captura<br>3. Deshabilita barra de opciones<br>4. Deshabilita grid<br>5. Habilita botones de acción<br>6. Indica en la barra de estado la operación seleccionada |
| P: Inicializa Campos | 1. Limpia Campos de Texto<br>2. Selecciona el primer elemento de cada combo<br>3. Quita la selección de los check box<br>4. Marca la primera opción de los radio buttons |
| P: Realiza Consulta | 1. Ejecuta consulta en la base de datos con los criterios solicitados<br>1. Devuelve el resultado de la consulta al grid<br>2. Posiciona el detalle del primer renglón del grid en los campos correspondientes<br>2. P (Prepara Lectura Opción) |
| P: Carga Catálogos Iniciales | Depende del caso de uso |
| P: Valida Datos | 1. Valida datos requeridos para alta y cambio |

Nota: La barra de opciones no habilita el botón de actualizar ni baja si el grid no tiene registros.

- ✓ Diseño de Interface




| PAQUETE | CASO DE USO | OBJETIVO |
|---|---|---|
| HISTORIA CLINICA | EXAMEN VISUAL | Tener el registro de su examen visual para futuras visitas |

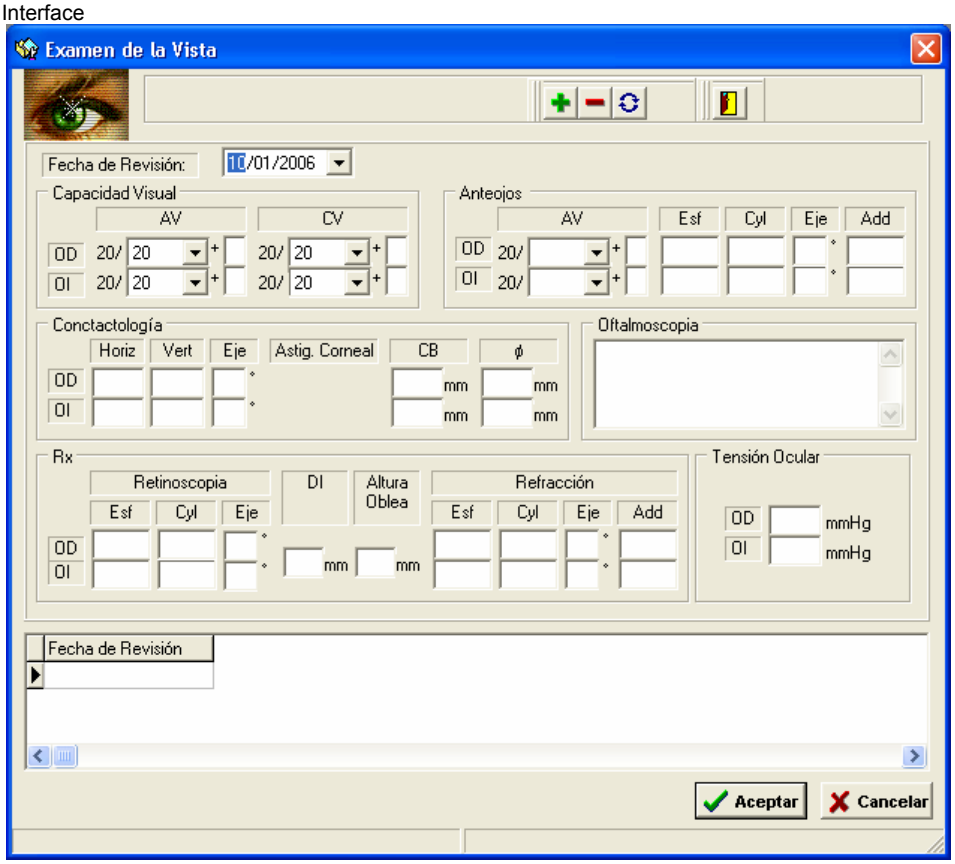
| | |
|---|---|
| Precondiciones | Existencia del paciente |
| Postcondiciones | Alta, baja, cambio y/o consulta del examen visual |
| Actores | Optometrista: Manipular el registro de los diferentes exámenes visuales realizados al paciente |
| Generaliza de | Generales.Catalogo |
| Extiende a | |
| Incluye a | Historia Clinica.Transpone Graduación |
| Entradas | Fecha de Revisión<br>**Capacidad Visual:** Agudeza Visual OD y OI, Agudeza Visual Siguiente Línea OD y OI, Capacidad Visual OD y OI, Capacidad Visual Siguiente Línea OD y OI.<br>**Anteojos:** Agudeza Visual OD y OI, Agudeza Visual Siguiente Línea OD y OI, Esfera OD y OI, Cilindro OD y OI, Eje OD y OI, Adición OD y OI.<br>**Contactología:** Diámetro Horizontal del Iris OD y OI, Diámetro Vertical del Iris OD y OI, Eje OD y OI, Curva Base del Iris OD y OI, Diámetro Queratometrico OD y OI..<br>Descripción Oftalmoscopica.<br>**Rx:** Retinoscopia Esfera OD y OI, Retinoscopia Cilindro OD y OI, Retinoscopia Eje OD y OI, Distancia Interpupilar, Altura de Oblea, Refracción Esfera OD y OI, Refracción Cilindro OD y OI, Refracción Eje OD y OI, Refracción Adición OD y OI.<br>Tensión Ocular OD y OI. |
| Salidas | Igual a Entradas, se muestra en pantalla el astigmatismo corneal |
| Dudas | |

| PROCESO/EVENTO | DESCRIPCIÓN |
|---|---|
| E: Cargar Forma | 1. Realiza las acciones indicadas en su padre<br>2. P (Realiza Consulta) con el criterio del paciente seleccionado en el menú principal |
| P: Carga Catálogos Iniciales | 1. Cargar los catálogos de Agudeza Visual y Capacidad Visual con los siguientes valores: 3200, 1600, 800, 600, 400, 200, 100, 70, 50, 40, 30, 25, 20, 15, 13, 10<br>2. Para los catálogos de Agudeza Visual de Anteojos agregar un elemento el blanco. |
| P: Inicializa Campos | 1. Realiza las acciones indicadas en su padre<br>2. Fecha de revisión se inicializa con fecha del sistema<br>3. Posiciona combos de agudeza visual y capacidad visual en el elemento 20, para |

| | |
|---|---|
| | la agudeza visual de anteojos posiciona combo en elemento en blanco<br>4. Limpia etiquetas de astigmatismo corneal |
| P: Realiza Consulta | 1. Realiza las acciones indicadas en su padre<br>2. Calcula astigmatismo corneal para OD y OI como sigue: Abs (Horiz – Vert) *(-1) concatenar una 'x' concatenar el eje de queratometria concatenar '°'<br>Ordena la consulta por las fechas de revisión más recientes |
| P: Valida Datos | 1. Realiza las acciones indicadas en su padre<br>2. Valida la graduación de retinoscopia, refracción y anteojos:<br>2.1. Si tiene cilindro es obligatorio el eje y viceversa<br>2.2. El eje debe estar entre cero y 180 grados<br>2.3. Las dioptrias de la esfera y cilindro deben estar en múltiplos de 0.25 si se capturaron<br>2.4. Las dioptrias de adición deben estar en múltiplos de 0.25 si se capturaron<br>3. Si se capturó el eje de concactologia debe estar entre 0 y 180 grados |

La consulta no se encuentra habilitada como opción para este caso de uso.

✓ Diseño de Interface




✓ Descripción de campos de interface

| CAMPO | DESCRIPCIÓN | CAMPOS DE CAPTURA | | | | REQUERIDO |
|---|---|---|---|---|---|---|
| | | ALTA | BAJA | CONS. | MOD. | |
| Fecha de Revisión | Fecha de realización del examen visual | X | | | X | X |
| | | | | | | |
| **Capacidad Visual** | | | | | | |
| AV OD | Agudeza visual ojo derecho | X | | | X | X |
| AV OD+ | Agudeza visual siguiente línea ojo derecho | X | | | X | |
| CV OD | Capacidad visual ojo derecho | X | | | X | X |
| CV OD+ | Capacidad visual siguiente línea ojo derecho | X | | | X | |
| AV OI | Agudeza visual ojo izquierdo | X | | | X | X |
| AV OI+ | Agudeza visual siguiente línea ojo izquierdo | X | | | X | |
| CV OI | Capacidad visual ojo izquierdo | X | | | X | X |
| CV OI+ | Capacidad visual siguiente línea ojo izquierdo | X | | | X | |
| | | | | | | |
| **Anteojos** | | | | | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| AV OD | Agudeza visual ojo derecho | X | | | X | Si captura AV OD+ |
| AV OD+ | Agudeza visual siguiente línea ojo derecho | X | | | X | |
| Esf OD | Esfera ojo derecho | X | | | X | |
| Cyl OD | Cilindro ojo derecho | X | | | X | |
| Eje OD | Eje ojo derecho | X | | | X | |
| Add OD | Adición ojo derecho | X | | | X | |
| AV OD | Agudeza visual ojo derecho | X | | | X | Si captura AV OD+ |
| AV OI+ | Agudeza visual siguiente línea ojo izquierdo | X | | | X | |
| Esf OI | Esfera ojo izquierdo | X | | | X | |
| Cyl OI | Cilindro ojo izquierdo | X | | | X | |
| Eje OI | Eje ojo izquierdo | X | | | X | |
| Add OI | Adición ojo izquierdo | X | | | X | |
| | | | | | | |
| **Contactología** | | | | | | |
| Horiz OD | Diámetro horizontal iris derecho | X | | | X | Si se captura algún dato del ojo en contact. |
| Vert OD | Diámetro vertical iris derecho | X | | | X | Si se captura algún dato del ojo en contact. |
| Eje OD | Eje ojo derecho | X | | | X | Si se captura algún dato del ojo en contact. |
| Astig. Corneal OD | Astigmatismo corneal ojo derecho | X | | | X | |
| Curva Base OD | Curva base del ojo derecho | X | | | X | |
| Diámetro OD | Diámetro para lente de contacto ojo derecho | X | | | X | |
| Horiz OI | Diámetro horizontal iris izquierdo | X | | | X | Si se captura algún dato del ojo en contact. |
| Vert OI | Diámetro vertical iris izquierdo | X | | | X | Si se captura algún dato del ojo en contact. |
| Eje OI | Eje ojo izquierdo | X | | | X | Si se captura algún dato del ojo en contact. |
| Astig. Corneal OI | Astigmatismo corneal ojo izquierdo | X | | | X | |
| Curva Base OI | Curva base del ojo izquierdo | X | | | X | |
| Diámetro OI | Diámetro para lente de contacto ojo izquierdo | X | | | X | |
| | | | | | | |
| Oftalmoscopia | Descripción de las condiciones del fondo de ojo a través de la revisión con el oftalmoscopio | X | | | X | |
| | | | | | | |
| **RX** | | | | | | |
| Retinoscopia Esf OD | Retinoscopia esfera ojo derecho | X | | | X | |
| Retinoscopia Cyl OD | Retinoscopia cilindro ojo derecho | X | | | X | |
| Retinoscopia Eje OD | Retinoscopia eje ojo derecho | X | | | X | |
| Retinoscopia Esf OI | Retinoscopia esfera ojo izquierdo | X | | | X | |
| Retinoscopia Cyl OI | Retinoscopia cilindro ojo izquierdo | X | | | X | |
| Retinoscopia Eje OI | Retinoscopia eje ojo izquierdo | X | | | X | |
| DI | Distancia interpupilar | X | | | X | X |
| Altura Oblea | Altura de Oblea | X | | | X | |
| Refracción Esf OD | Refracción esfera ojo derecho | X | | | X | |
| Refracción Cyl OD | Refracción cilindro ojo derecho | X | | | X | |
| Refracción Eje OD | Refracción eje ojo derecho | X | | | X | |
| Refracción Add OD | Refracción adición ojo derecho | X | | | X | |
| Refracción Esf OI | Refracción esfera ojo izquierdo | X | | | X | |
| Refracción Cyl OI | Refracción cilindro ojo izquierdo | X | | | X | |
| Refracción Eje OI | Refracción eje ojo izquierdo | X | | | X | |
| Refracción Add OI | Refracción adición ojo izquierdo | X | | | X | |
| | | | | | | |
| Tensión Ocular OD | Tensión ocular ojo derecho | X | | | X | |
| Tensión Ocular OI | Tensión ocular ojo izquierdo | | | | | |

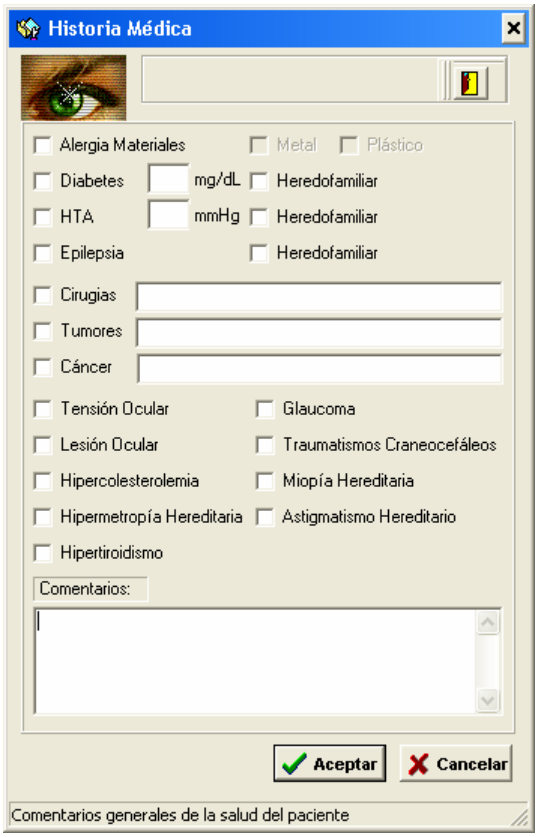
| PAQUETE | CASO DE USO | OBJETIVO |
|---|---|---|
| HISTORIA CLINICA | HISTORIA MEDICA | Contar con la historia médica del paciente para ofrecer una mejor solución a sus problemas refractivos |

| | |
|---|---|
| Precondiciones | Existencia del paciente |
| Postcondiciones | Cambio y consulta de la historia médica |
| Actores | Optometrista: Manipular la historia médica del paciente |
| Generaliza de | Generales.Catalogo |
| Extiende a | |
| Incluye a | |
| Entradas | Alergia a Metal, Alergia a Plástico, Diabetes, Nivel de Diabetes, Diabetes Heredofamiliar, Hipertensión Arterial, Nivel de Hipertensión Arterial, Hipertensión Arterial Heredofamiliar, Epilepsia, Epilepsia Heredofamiliar, Cirugías, Descripción de Cirugías, Tumores, Descripción de Tumores, Cáncer, Descripción de Cáncer, Tensión Ocular, Glaucoma, Lesión Ocular, Traumatismos Craneocefaleos, Hipercolesterolemia, Miopía Hereditaria, Hipermetropía Hereditaria, Astigmatismo Hereditario, Hipertiroidismo, Comentarios Generales |
| Salidas | Igual a Entradas |
| Dudas | |

| PROCESO/EVENTO | DESCRIPCIÓN |
|---|---|
| E: Cargar Forma | 1. Realiza las acciones indicadas en su padre<br>2. P (Realiza Consulta) con el criterio del paciente seleccionado en el menú principal |
| P: Valida Datos | 1. Realiza las acciones indicadas en su padre<br>2. Si se indica alergia se debe indicar alergia a metal y/o alergia a plástico |
| P: Proceso | 1. Actualiza información |

El caso de uso es de tipo proceso.

✓ Diseño de Interface




✓ Descripción de campos de interface

| CAMPO | DESCRIPCIÓN | CAMPOS DE CAPTURA. | REQUERIDO |
|---|---|---|---|
| Alergia Materiales | Alergia a materiales para implantar una armazón | X | Si se captura metal o plástico |

| | | | |
|---|---|---|---|
| Metal | Alergia a metal | X | |
| Plástico | Alergia a plástico | X | |
| Diabetes | Paciente con diabetes | X | Si se captura nivel diabetes |
| Diabetes mg/dL | Nivel de diabetes habitual del paciente | X | Si se captura diabetes |
| Diabetes Heredofamiliar | | X | |
| HTA | Paciente con hipertensión arterial | X | Si se captura nivel HTA |
| HTA mmHg | Nivel de hipertensión arterial habitual del paciente | X | Si se captura HTA |
| HTA Heredofamiliar | Pacientes con familiares con hipertensión arterial | X | |
| Epilepsia | Paciente con epilepsia | X | |
| Epilepsia Heredofamiliar | Paciente con familiares epilépticos | X | |
| Cirugías | Paciente con cirugías | X | Si se captura cirugías (descripción) |
| Cirugías (descripción) | Descripción de cirugías | X | Si se captura cirugías |
| Tumores | Paciente con tumores | X | Si se captura tumores (descripción) |
| Tumores (descripción) | Descripción de tumores | X | Si se captura tumores |
| Cáncer | Paciente con cáncer | X | Si se captura cáncer (descripción) |
| Cáncer (descripción) | Descripción de cáncer | X | Si se captura cáncer |
| Tensión Ocular | Paciente con tensión ocular | X | |
| Glaucoma | Paciente con glaucoma | X | |
| Lesión Ocular | Paciente con lesión ocular | X | |
| Traumatismos Craneocefálicos | Paciente con traumatismos craneocefálicos | X | |
| Hipercolesterolemia | Paciente con hipercolesterolemia | X | |
| Miopía Hereditaria | Paciente con miopía hereditaria | X | |
| Hipermetropía Hereditaria | Paciente con hipermetropía hereditaria | X | |
| Astigmatismo Hereditario | Paciente con astigmatismo hereditario | X | |
| Hipertiroidismo | Paciente con hipertiroidismo | X | |

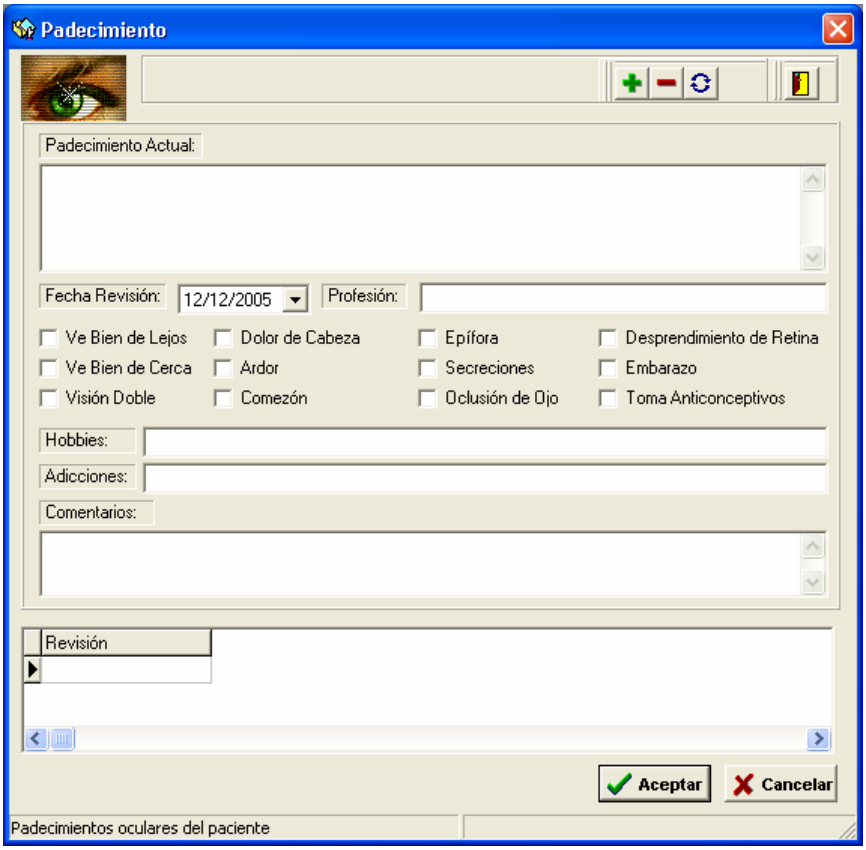
| **PAQUETE** | **CASO DE USO** | OBJETIVO |
|---|---|---|
| HISTORIA CLINICA | PADECIMIENTO | Tener un histórico de los padecimientos presentados por un paciente |

| | |
|---|---|
| Precondiciones | Existencia del paciente |
| Postcondiciones | Alta, baja, cambio y/o consulta de padecimientos |
| Actores | Optometrista: Manipular el registro de los diferentes padecimientos del paciente |
| Generaliza de | Generales.Catalogo |
| Extiende a | |
| Incluye a | |
| Entradas | Padecimiento Actual, Fecha de Revisión, Profesión, Ve Bien de Lejos, Ve Bien de Cerca, Visión Doble, Dolor de Cabeza, Ardor, Comezón, Epífora, Secreciones, Oclusión de Ojo, Desprendimiento de Retina, Embarazo, Toma Anticonceptivos, Hobbies, Adicciones, Comentarios |
| Salidas | Igual a Entradas |
| Dudas | |

| PROCESO/EVENTO | DESCRIPCIÓN |
|---|---|
| E: Cargar Forma | 1. Realiza las acciones indicadas en su padre<br>2. P (Realiza Consulta) con el criterio del paciente seleccionado en el menú principal |
| P: Inicializa Campos | 1. Realiza las acciones indicadas en su padre<br>2. Fecha de revisión se inicializa con fecha del sistema |
| P: Realiza Consulta | 1. Realiza las acciones indicadas en su padre<br>Ordena la consulta por las fechas de revisión más recientes |

La consulta no se encuentra habilitada como opción para este caso de uso.

✓ Diseño de Interface




✓ Descripción de campos de interface

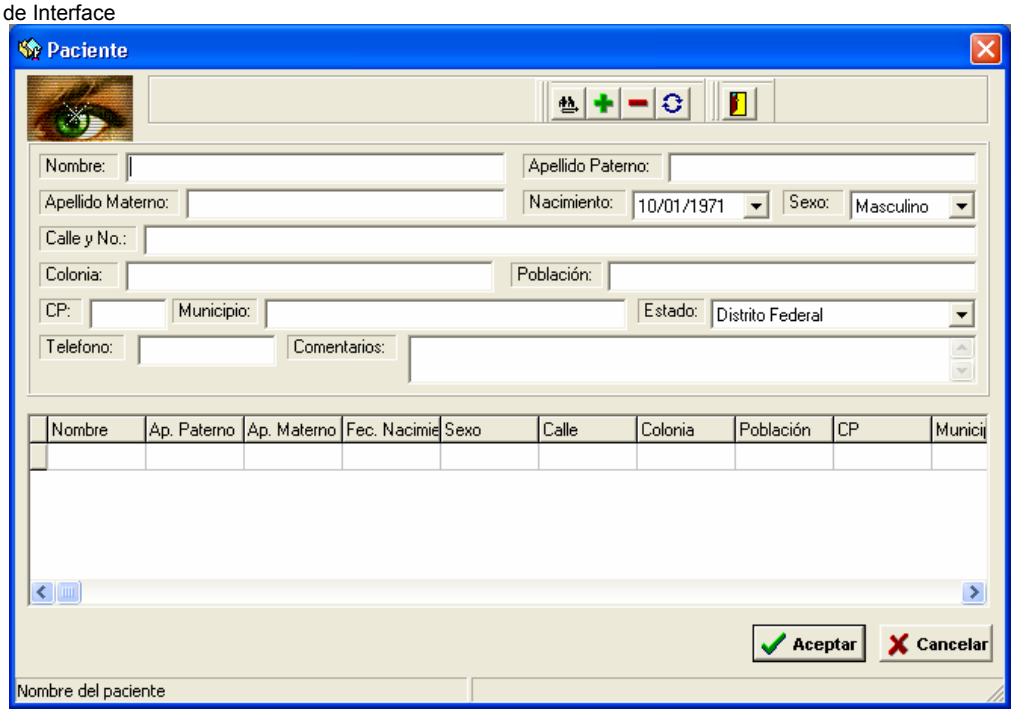
| CAMPO | DESCRIPCIÓN | CAMPOS DE CAPTURA | | | | REQUERIDO |
|---|---|---|---|---|---|---|
| | | ALTA | BAJA | CONS. | MOD. | |
| Padecimiento Actual | Padecimiento actual del paciente | | | | X | X |
| Fecha de Revisión | Fecha de realización del examen visual | | | | X | X |
| Profesión | Profesión del paciente | | | | X | |
| Ve Bien de Lejos | Paciente ve bien de lejos | | | | x | |
| Ve Bien de Cerca | Paciente ve bien de cerca | | | | x | |
| Visión Doble | Paciente con visión doble | | | | x | |
| Dolor de Cabeza | Paciente con dolor de cabeza | | | | x | |
| Ardor | Paciente con ardor | | | | x | |
| Comezón | Paciente con comezón | | | | x | |
| Epífora | Paciente con epífora | | | | x | |
| Secreciones | Paciente con secreciones | | | | x | |
| Oclusión de Ojo | Paciente con oclusión de ojo | | | | x | |
| Desprendimiento de Retina | Paciente con desprendimiento de retina | | | | x | |
| Embarazo | Paciente con embarazo | | | | x | |
| Toma Anticonceptivos | Paciente toma anticonceptivos | | | | x | |
| Hobbies | Hobbies del paciente | | | | x | |
| Adicciones | Adicciones del paciente | | | | x | |
| Comentarios | Comentarios del paciente | | | | x | |

| **PAQUETE** | **CASO DE USO** | OBJETIVO |
|---|---|---|
| PACIENTE | CATALOGO DE PACIENTE | Mantener el registro de pacientes de la óptica |

| | |
|---|---|
| Precondiciones | |
| Postcondiciones | Alta, baja, cambio y/o consulta de pacientes |
| Actores | Optometrista: Manipular el catálogo de pacientes |
| Generaliza de | Generales.Catalogo |
| Extiende a | |
| Incluye a | |
| Entradas | Nombre, Apellido Paterno, Apellido Materno, Fecha de Nacimiento, Sexo, Calle y Número, Colonia, Población, Código Postal, Municipio, Estado, Teléfono, Comentarios |
| Salidas | Igual a Entradas |
| Dudas | |

| PROCESO/EVENTO | DESCRIPCIÓN |
|---|---|
| E: Carga Forma | 1. Realiza las acciones indicadas en su padre<br>2. Habilita el botón de paciente del menú principal |
| E: Click Salir | 1. Deshabilita el botón de paciente del menú principal<br>2. Realiza las acciones indicadas en su padre |
| P: Carga Catálogos Iniciales | 3. Cargar los catálogos de Sexo y Estados |
| P: Inicializa Campos | 1. Realiza las acciones indicadas en su padre<br>2. Fecha de nacimiento se inicializa con fecha del sistema |
| P: Realiza Consulta | 1. Realiza las acciones indicadas en su padre<br>Ordena la consulta por las apellido paterno, apellido materno, nombre y fecha de nacimiento. La consulta se realiza utilizando el inicio de las cadenas especificadas con un comodín de búsqueda al final de las mismas. |
| P: Valida Datos | 1. Realiza las acciones indicadas en su padre<br>2. La fecha de nacimiento debe ser menor a la del sistema |

✓ Diseño de Interface




✓ Descripción de campos de interface

| CAMPO | DESCRIPCIÓN | CAMPOS DE CAPTURA | | | | REQUERIDO |
|---|---|---|---|---|---|---|
| | | ALTA | BAJA | CONS. | MOD. | |
| Nombre | Nombre del paciente | X | | X | X | X |
| Apellido Paterno | Apellido paterno del paciente | X | | X | X | X |
| Apellido Materno | Apellido materno del paciente | X | | X | X | X |
| Nacimiento | Fecha de nacimiento del paciente | X | | | X | X |
| Sexo | Sexo del paciente | X | | | X | X |
| Calle y No. | Calle y número | X | | | X | X |
| Colonia | Colonia | X | | | X | X |
| Población | Población | X | | | X | X |
| CP | Código Postal | X | | | X | X |

| | | | | | | |
|---|---|---|---|---|---|---|
| Municipio | Municipio | X | | | X | X |
| Estado | Estado | X | | | X | X |
| Teléfono | Teléfono | X | | | X | |
| Comentarios | Comentarios generales del paciente | X | | | X | |

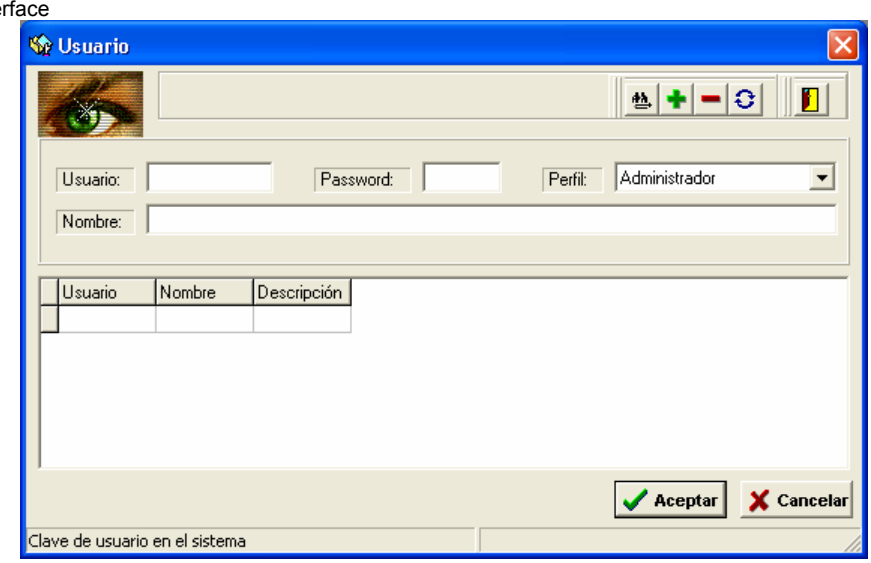
| **PAQUETE** | **CASO DE USO** | OBJETIVO |
|---|---|---|
| SEGURIDAD | CATALOGO DE USUARIOS | Dar de alta a los usuarios con acceso al sistema restringiéndolos con un perfil. |

| | |
|---|---|
| Precondiciones | Tener privilegio de administrador |
| Postcondiciones | Alta, baja, cambio y/o consulta de usuarios |
| Actores | Administrador: Manipular catálogo de usuarios |
| Generaliza de | Generales.Catalogo |
| Extiende a | |
| Incluye a | |
| Entradas | Usuario, Password, Perfil, Nombre |
| Salidas | Igual a Entradas |
| Dudas | |

| PROCESO/EVENTO | DESCRIPCIÓN |
|---|---|
| E: Cargar Forma | 1. Realiza las acciones indicadas en su padre |
| E: Click Grid | 1. Realiza las acciones indicadas en su padre<br>2. El password se transfiere en blanco |
| P: Carga Catálogos Iniciales | 1. Cargar el catálogo de perfiles |
| P: Realiza Consulta | 1. Realiza las acciones indicadas en su padre<br>Ordena la consulta por el nombre |
| P: Cambio | 1. Si el password es capturado entonces actualizar todos los datos<br>2. En caso contrario actualiza todos los datos excepto el password |

Al momento de realizar el cambio,

✓ Diseño de Interface




✓ Descripción de campos de interface

| CAMPO | DESCRIPCIÓN | CAMPOS DE CAPTURA | | | | REQUERIDO |
|---|---|---|---|---|---|---|
| | | ALTA | BAJA | CONS. | MOD. | |
| Usuario | Usuario del sistema | X | | X | X | X |
| Password | Contraseña de acceso al sistema | X | | | X | X |
| Perfil | Perfil de acceso para el usuario | X | | X | X | X |
| Nombre | Nombre real del usuario | X | | X | X | X |

| PAQUETE | CASO DE USO | OBJETIVO |
|---|---|---|
| SEGURIDAD | CAMBIO DE PASSWORD | Permitir al usuario actualizar su password en cualquier momento requerido |

| | |
|---|---|
| Precondiciones | Existencia del usuario |
| Postcondiciones | cambio de password |
| Actores | Optometrista: Manipular el registro de los diferentes exámenes visuales realizados al cliente |
| Generaliza de | Generales.Catalogo |
| Extiende a | |
| Incluye a | Historia Clinica.Transpone Graduación |
| Entradas | Usuario, Password, Confirmación Password |
| Salidas | Igual a Entradas |
| Dudas | |

| PROCESO/EVENTO | DESCRIPCIÓN |
|---|---|
| P: Inicializa Campos | 1. Realiza las acciones indicadas en su padre<br>2. Asigna al usuario el usuario firmado al sistema |
| P: Valida Datos | 1. El password y la confirmación deben ser iguales |
| P: Proceso | 1. Actualiza password |

El caso de uso es de tipo proceso..

✓ Diseño de Interface




✓ Descripción de campos de interface

| CAMPO | DESCRIPCIÓN | CAMPOS DE CAPTURA. | REQUERIDO |
|---|---|---|---|
| Usuario | Usuario firmado al sistema | | X |
| Password | Nuevo password del usuario | X | X |
| Confirmación | Confirmación del nuevo password del usuario | X | X |

| PAQUETE | CASO DE USO | OBJETIVO |
|---|---|---|
| SEGURIDAD | INGRESO AL SISTEMA | Brindar seguridad de acceso al sistema |

| | |
|---|---|
| Precondiciones | |
| Postcondiciones | Ingreso al Menú Principal |
| Actores | Optometrista: Ingresar al sistema |
| Generaliza de | Generales.Catalogo |
| Extiende a | |
| Incluye a | |
| Entradas | Usuario, Password |
| Salidas | Usuario, Password |
| Dudas | |

| PROCESO/EVENTO | DESCRIPCIÓN |
|---|---|
| P: Proceso | 1. Realiza la consulta del usuario y password<br>2. Si encuentra la coincidencia accesa al menú principal<br>3. En caso contrario notifica mensaje de password y contraseña inválidos |

La consulta no se encuentra habilitada como opción para este caso de uso.

✓ Diseño de Interface




✓ Descripción de campos de interface

| CAMPO | DESCRIPCIÓN | CAMPOS DE CAPTURA. | REQUERIDO |
|---|---|---|---|
| Usuario | Usuario firmado al sistema | | X |
| Password | Nuevo password del usuario | X | X |
| Confirmación | Confirmación del nuevo password del usuario | X | X |

| **PAQUETE** | **CASO DE USO** | OBJETIVO |
|---|---|---|
| GENERALES | MENU PRINCIPAL | Ayudar a navegar dentro de la aplicación a las diferentes opciones |

| | |
|---|---|
| Precondiciones | Haber ingresados un usuario y password válido |
| Postcondiciones | Muestra opciones en base al perfil |
| Actores | Optometrista: Navegar por el sistema |
| Generaliza de | |
| Extiende a | Menu Seguridad, Menu Historia Clinica |
| Incluye a | |
| Entradas | Usuario |
| Salidas | Perfil habilitado |
| Dudas | |

| PROCESO/EVENTO | DESCRIPCIÓN |
|---|---|
| E: Carga Forma | 1. Limpiar etiqueta de paciente<br>2. Deshabilitar botón de paciente<br>3. Deshabilitar todas las opciones del menú<br>4. Si el perfil del usuario es optometrista habilitar las opciones de cambio de password y paciente<br>5. Si el perfil del usuario es administrador habilitar todas las opciones del optometrista y la del catálogo de usuario |
| E: Click Botón Paciente | 1. Muestra en la etiqueta de paciente el paciente seleccionado en el caso de uso de Catálogo de Pacientes.<br>2. Habilita todas las opciones del menú de historia clínica |

La consulta no se encuentra habilitada como opción para este caso de uso.

✓ Diseño de Interface




✓ Descripción de campos de interface

| CAMPO | DESCRIPCIÓN | CAMPOS DE CAPTURA. | REQUERIDO |
|---|---|---|---|
| Paciente | Nombre del paciente seleccionado | | |

## *Prototipo*

En base a las interfaces se arma un ejecutable para la navegación y se complementa emitiendo mensajes representativos, para indicar acciones teóricamente ejecutadas. Aunque el prototipo se puede llevar a más detalle, lo importante de este será identificar si cubre las necesidades del usuario enfocado al enlace de las interfaces.

# Diseño

Al concluir el análisis ya contamos con una descripción detallada de las expectativas del usuario, por ende podemos iniciar la propuesta de un diseño para satisfacer el conjunto global de sus necesidades. El realizar diseños parciales sin tener todo el análisis conlleva la dificultad de iniciar segmentos de diseños, para después buscar integrarlos como un solo diseño generando el riesgo de no acoplarse fácilmente.

# *Identificación de Clases y Relaciones*

## Clases de Entidad

La función de las clases de entidad es almacenar la información por eso para identificar las clases de entidad será necesario revisar las especificaciones de casos de uso, considerando principalmente entradas, salidas e interface.

En algunos casos también pueden encontrarse en la descripción del caso de uso información importante, como atributos útiles para ser recordados posteriormente como estatus, cálculos, entre otros, pero sólo sí tienen una trascendencia para ser almacenados en una variable del sistema. Este tipo de atributos apoyan para describir a una clase.

## *Normalización de Base de Datos*

Para normalizar la base de datos seguiremos la metodología propuesta para el diseño de una base de datos.

### Identificación de Sustantivos

Aunque en este caso no tenemos propiamente sustantivos descritos en la especificación de caso de uso, los podemos intuir en base a la descripción o bien verificar la especificación de requerimientos detallados.

Sustantivos: Usuario, Perfil, Paciente, Padecimiento, Historia Médica, Examen Visual.

### Identificación de Verbos

Los verbos pueden ser obtenidos de la misma fuente de los sustantivos. Los verbos relacionados con sustantivos son:

- Un usuario *tiene* un perfil
- Un paciente *padece* padecimientos
- Un paciente *registra* una historia médica
- Un paciente *realiza* un examen visual

### Identificación de Atributos

Los atributos se emiten de la especificación de casos de uso y se relacionan a un sustantivo.

*Usuario:* Clave, Password, Perfil, Nombre

*Perfil*: Descripción

*Paciente:* Nombre, Apellido Paterno, Apellido Materno, Fecha de Nacimiento, Sexo, Domicilio, Colonia, Población, Código Postal, Municipio, Estado, Teléfono, Comentarios

*Padecimiento:* Paciente, Padecimiento Actual, Fecha de Revisión, Profesión, Ve Bien de Lejos, Ve Bien de Cerca, Visión Doble, Dolor de Cabeza, Ardor, Comezón, Epífora, Secreciones, Oclusión de Ojo, Desprendimiento de Retina, Embarazo, Toma Anticonceptivos, Hobbies, Adicciones, Comentarios

*Historia Médica:* Paciente, Alergia Materiales, Metal, Plástico, Diabetes, Nivel Diabetes, Diabetes Heredofamiliar, HTA, Nivel HTA, HTA Heredofamiliar, Epilepsia, Epilepsia Heredofamiliar, Cirugías, Descripción Cirugías, Tumores, Descripción Tumores, Cáncer, Descripción Cáncer, Tensión Ocular, Glaucoma, Lesión Ocular, Traumatismos Craneocefálicos, Hipercolesterolemia, Miopía Hereditaria, Hipermetropía Hereditaria, Astigmatismo Hereditario, Hipertiroidismo, Comentario

*Examen Visual*: Paciente, Fecha de Revisión, AV OD, AV OD+, CV OD, CV OD+, AV OI, AV OI+, CV OI, CV OI+, Anteojos AV OD, Anteojos AV OD+, Anteojos Esf OD, Anteojos Cyl OD, Anteojos Eje OD, Anteojos Add OD, Anteojos AV OD, Anteojos AV OI+, Anteojos Esf OI, Anteojos Cyl OI, Anteojos Eje OI, Anteojos Add OI, Horiz OD, Vert OD, Eje OD, Curva Base OD, Diámetro OD, Horiz OI, Vert OI, Eje OI, Astig. Corneal OI, Curva Base OI, Diámetro OI, Oftalmoscopia, Retinoscopia Esf OD, Retinoscopia Cyl OD, Retinoscopia Eje OD, Retinoscopia Esf OI, Retinoscopia Cyl OI, Retinoscopia Eje OI, DI, Altura Oblea, Refracción Esf OD, Refracción Cyl OD, Refracción Eje OD, Refracción Add OD, Refracción Esf OI, Refracción Cyl OI, Refracción Eje OI, Refracción Add OI, Tensión Ocular OD, Tensión Ocular OI

*Estado*: Descripción

Para el examen visual el astigmatismo corneal no se considera como atributo derivado de su posibilidad de ser calculado, si el cálculo fuera muy costoso en tiempo pudiera considerarse para almacenarse pero no es el caso.

En este caso las entidades de datos van muy ligadas a cada una de las interfaces, pero no siempre es de esta manera, debemos identificar relaciones de datos y tratar de agruparlas por su concepto lógico independientemente de la interface de origen, es decir considerarlas como sustantivos.

## Análisis de Atributos

En este caso el análisis de atributos queda cubierto en las especificaciones de casos de uso en la sección de descripción de campos.

## Selección de Llaves Primarias Iniciales

Se seleccionarán las llaves primaras:

*Usuario:* Clave

*Perfil*: Descripción

*Paciente:* Nombre, Apellido Paterno, Apellido Materno

*Padecimiento:* Paciente, Fecha de Revisión

*Historia Médica:* Paciente

*Examen Visual*: Paciente, Fecha de Revisión

## Generar Atributos Clave

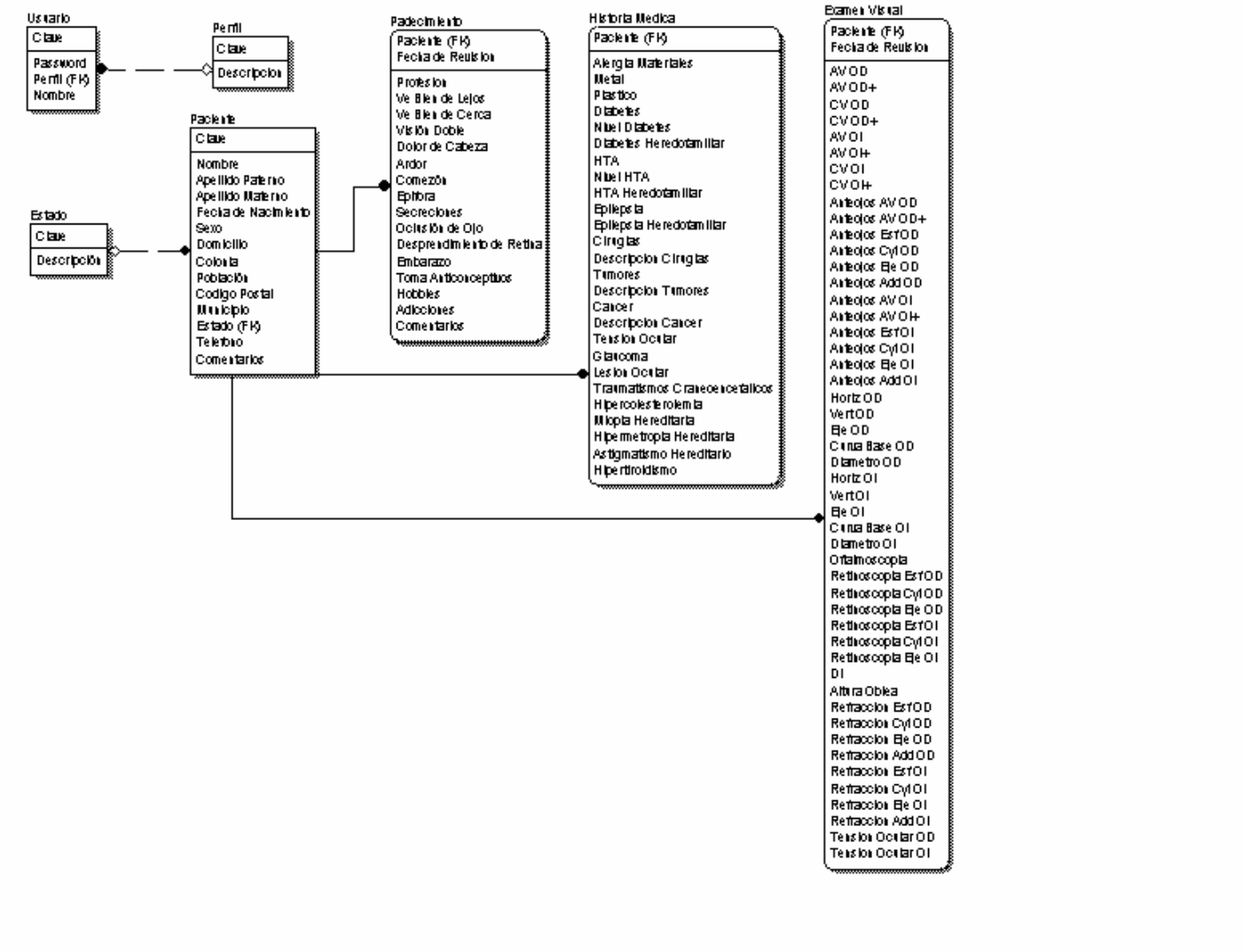
Se observa que un paciente se convierte en llave foránea de diferentes entidades, y estrictamente un paciente lo podemos identificar por su nombre, el viajar el nombre completo como llave foránea a otra entidad provocará enlaces muy largos afectando en tiempos de respuesta, además de ser una llave primaria con posibilidad de modificarse en caso de error al digitar el nombre, esto nos lleva a crear una clave para el paciente, siendo un acceso más corto para enlazarse con otras entidades además de evitar tener llaves primarias modificables.

El atributo del estado de la república estrictamente sólo puede tomar valores fijos, por eso también decidimos convertirlo en catálogo, siendo más eficiente también asignarle una clave y apoyarse para enlazar dos entidades a través de una clave y no una descripción.

Para el sexo se pudo haber generado otro catálogo, pero se decidió no realizarlo para no empezar a tener catálogos por cualquier consideración, bastará con una clave del sexo para no almacenar su descripción larga (F- Femenino y M – Masculino).

Para el perfil de un usuario también se creará una clave para su manipulación.

## Diagrama E/R Inicial




## Dependencias Funcionales con Selección de Llaves Primarias

Al buscar las relaciones de dependencias funcionales realizamos una selección de las llaves primarias con características de valores cortos y baja posibilidad de modificación.

*Usuario:*

Clave Usuario → Password | Perfil | Nombre

*Perfil*:

Clave Perfil → Descripción

*Paciente:*

Clave Paciente → Nombre | Apellido Paterno | Apellido Materno | Fecha de Nacimiento | Sexo | Domicilio | Colonia | Población | Código Postal | Municipio | Estado | Teléfono | Comentarios

*Padecimiento:*

Paciente. Fecha de Revisión → Padecimiento Actual | Ve Bien de Lejos | Ve Bien de Cerca | Visión Doble | Dolor de Cabeza | Ardor | Comezón | Epífora | Secreciones | Oclusión de Ojo | Desprendimiento de Retina | Embarazo | Toma Anticonceptivos | Comentarios

Paciente → Profesión | Hobbies | Adicciones

*Historia Médica:*

Paciente → Alergia Materiales | Metal | Plástico | Diabetes | Nivel Diabetes | Diabetes Heredofamiliar | HTA | Nivel HTA | HTA Heredofamiliar | Epilepsia | Epilepsia Heredofamiliar | Cirugías | Descripción Cirugías | Tumores | Descripción Tumores | Cáncer | Descripción Cáncer | Tensión Ocular | Glaucoma | Lesión Ocular | Traumatismos Craneocefálicos | Hipercolesterolemia | Miopía Hereditaria | Hipermetropía Hereditaria | Astigmatismo Hereditario | Hipertiroidismo | Comentario

*Examen Visual*:

Paciente . Fecha de Revisión → AV OD | AV OD+ | CV OD | CV OD+ | AV OI | AV OI+ | CV OI | CV OI+ | Anteojos AV OD | Anteojos AV OD+ | Anteojos Esf OD | Anteojos Cyl OD | Anteojos Eje OD | Anteojos Add OD | Anteojos AV OD | Anteojos AV OI+ | Anteojos Esf OI | Anteojos Cyl OI | Anteojos Eje OI | Anteojos Add OI | Horiz OD | Vert OD | Eje OD | Curva Base OD | Diámetro OD | Horiz OI | Vert OI | Eje OI | Astig. Corneal OI | Curva Base OI | Diámetro OI | Oftalmoscopia | Retinoscopia Esf OD | Retinoscopia Cyl OD | Retinoscopia Eje OD | Retinoscopia Esf OI | Retinoscopia Cyl OI | Retinoscopia Eje OI | DI | Altura Oblea | Refracción Esf OD | Refracción Cyl OD | Refracción Eje OD | Refracción Add OD | Refracción Esf OI | Refracción Cyl OI | Refracción Eje OI | Refracción Add OI | Tensión Ocular OD | Tensión Ocular OI

*Estado*:

Clave Estado → Descripción

Podemos observar para la entidad de historia médica y paciente la misma llave primaria, esto puede convertirse en la misma entidad, pero derivado de no asegurar tener para un paciente siempre una historia médica, además de manejar información conceptualmente diferente no se unen en la misma entidad, esto se convierte en una decisión de diseño.

Dentro de la entidad de padecimiento detectamos la misma situación anterior, pero en este caso observamos la profesión, los hobbies y las adicciones más como una información propia de la entidad de paciente tomando la decisión de migrar estos atributos a paciente.

*Paciente:*

Clave Paciente → Nombre | Apellido Paterno | Apellido Materno | Fecha de Nacimiento | Sexo | Domicilio | Colonia | Población | Código Postal | Municipio | Estado | Teléfono | Comentarios | Profesión | Hobbies | Adicciones

## Graficar Dependencias Funcionales

En este punto debemos graficar las dependencias funcionales con cierto nivel de complejidad, al tener dependencias funcionales simples este paso se omite.

## Normalización

Al revisar cada una de las entidades aplicaremos las formas normales ya sea las tres por cada entidad de manera gradual o bien por cada forma normal aplicada a todas las entidades.

Al aplicar la 1FN tenemos:

- Usuario: Cumple
- Perfil: Cumple
- Paciente: Cumple
- Estado: Cumple

Padecimiento: Esta entidad tiene diferentes padecimientos con la característica común de poder se agrupados bajo un mismo concepto (un catálogo de padecimientos), el realizar esta normalización dará la ventaja de poder incrementar algún padecimiento a la aplicación, sin necesidad de realizar cambios estructurales a la base de datos. El desglose de la primera forma normal para la entidad padecimiento es:

*Catálogo de Padecimiento:* Clave Padecimiento → Descripción

*Padecimiento:* Paciente . Fecha de Revisión → Padecimiento Actual | Comentarios

*Padecimiento Booleano*: Paciente . Fecha de Revisión . Padecimiento

Historia Médica: Se ve afectada también por un grupo de información clasificables en dos grupos (booleano de historia médica y descripciones de historias médicas). Las clasificables como descripciones también pueden estar marcadas como booleanas para luego indicar su descripción, pero se considera que si tiene descripción se presenta la enfermedad, evitando la posible falla de inconsistencia de tener las descripciones médicas vacías sin indicar un valor booleano. La alergia de materiales es una consecuencia de tener alergia por un metal y/o por un plástico, no podemos tener indicada la alergia a materiales si no se tiene alergia a metal o plástico, es decir la alergia a materiales es un atributo calculable a partir de alergia a metal o alergia a plástico y por eso se deja de considerar.

*Catálogo de Historia Médica Booleana:* Clave Historia Médica Booleana → Descripción

*Catálogo de Historia Médica Descripción:* Clave Historia Médica Descripción → Descripción

*Historia Médica Booleana:* Paciente → Historia Medica Booleana

*Historia Médica Descripcion:* Paciente → Historia Medica Descripción | Descripción (un ejemplo sería como historia medica descripción: tumor y la descripción: en la oreja)

El comentario asociado a historia médica queda totalmente sólo, así que podemos crear una entidad exclusiva con este dato o bién por ser un solo dato incorporarlo a la entidad de paciente, en este caso tomamos la segunda opción.

Examen Visual: Sufre una agrupación también de sus atributos de la siguiente manera:

*Examen*: Paciente . Fecha de Revisión → Oftalmoscopia | DI | Altura Oblea

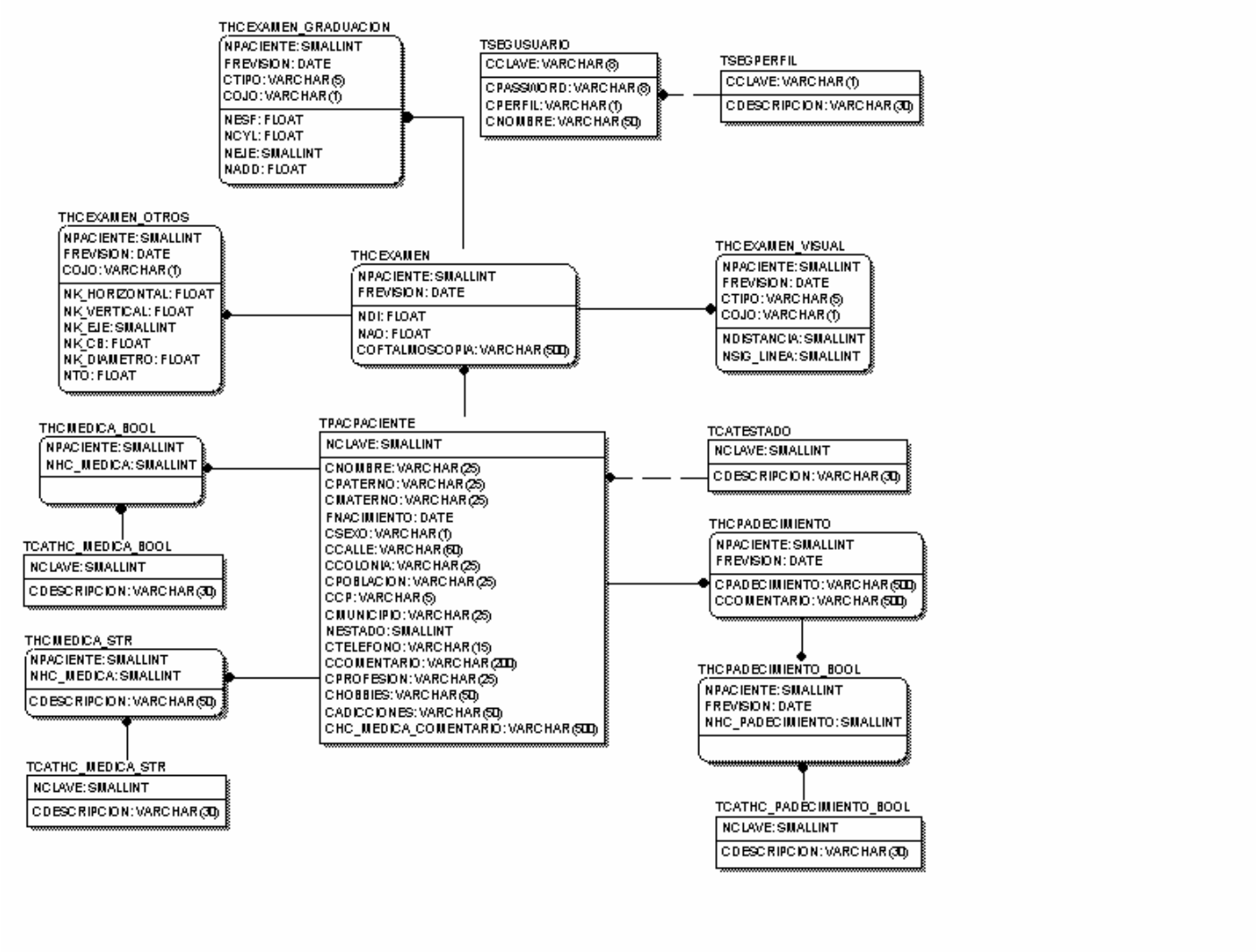
*Examen Visual:* Paciente . Fecha de Revisión . Tipo (AV; CV, Anteojos AV) . Ojo (D, I)→ Distancia (anteriormente AV OD, etc.) | Siguiente Línea (anteriormente AV OD+, etc)

*Examen Graduación:* Paciente . Fecha de Revisión . Tipo (Anteojos, Refracción, Retinoscopia) . Ojo (D, I) → Esf | Cyl | Eje | Add

*Examen Otros:* Paciente . Fecha de Revisión . Ojo (D, I) → Queratometria Horiz | Queratometria Vert | Eje Queratometria | Curva Base Queratometria | Diámetro Queratometria | TO

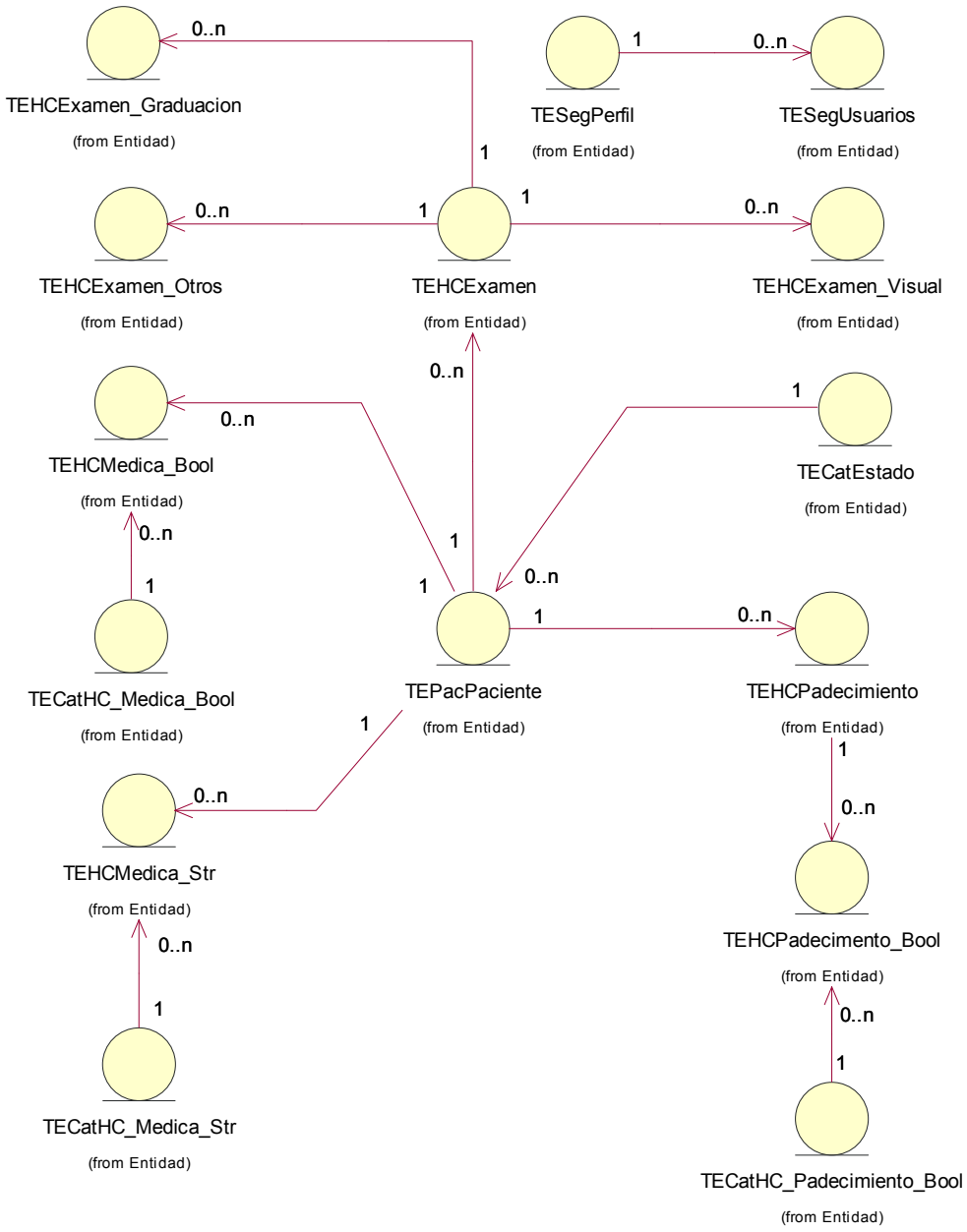
Al revisar todas las entidades cumplen con 2FN y 3FN.

## Diagrama E/R




## *Diagrama de Clases de Entidad*

El mapeo de las entidades resultantes de la normalización de base de datos es uno a uno a entidades de clases.



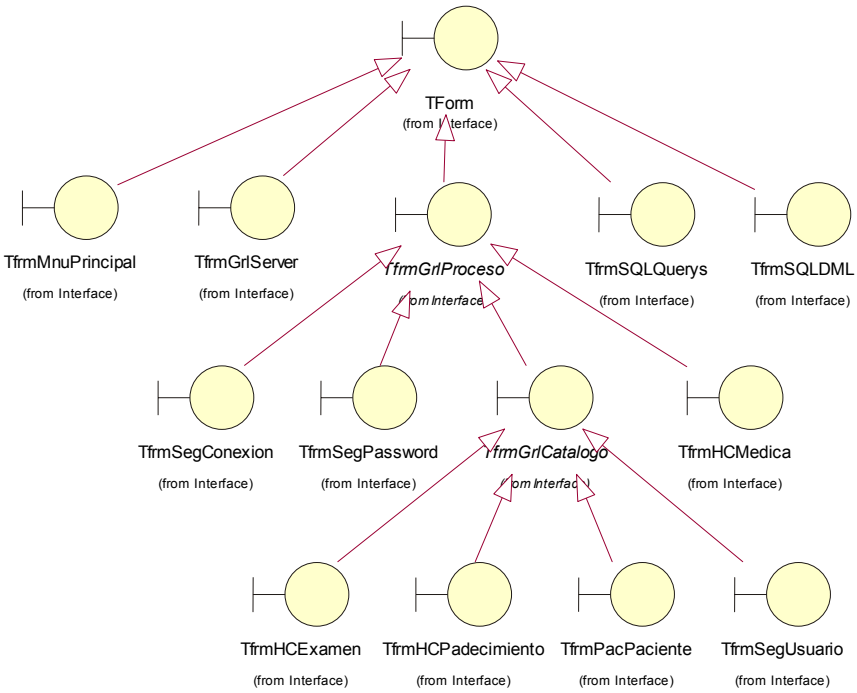

El tipo de relación utilizado comunmente es la relación.

## Clases de Interface

De la descripción de casos de uso conocemos las interfaces principales del sistema, y en base a su funcionalidad buscamos los diferentes tipos de relaciones existentes entre ellas. El tipo de relación más común son la herencia, la extensión y composición entre clases de interface. Aunque la asociación se

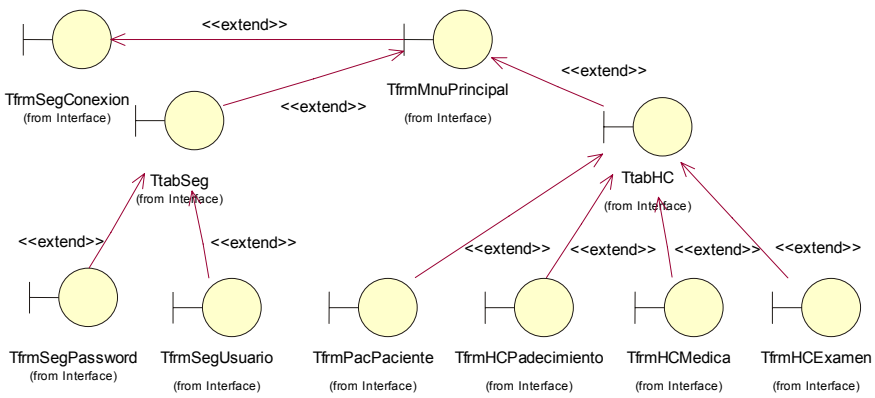
presente se omite para etapas posteriores, en este punto lo más importante es buscar la relación entre las mismas clases de interface.

Se decide, en base a la descripción de los casos de uso, generar un tipo de interface proceso, el cual una de sus variantes es la interface de catálogo. Como parte de una interface de proceso, se presenta una pantalla y al realizar determinadas acciones se presiona un botón de aceptar para iniciar el proceso.




La aplicación se propone realizarla con un servidor de reglas de negocio, es donde podemos detectar la necesidad de contar con una interface del servidor de reglas de negocio, y también se almacenarán los querys y los DML's de la aplicación en interfaces para apoyar en la consulta de estas instrucciones de SQL.

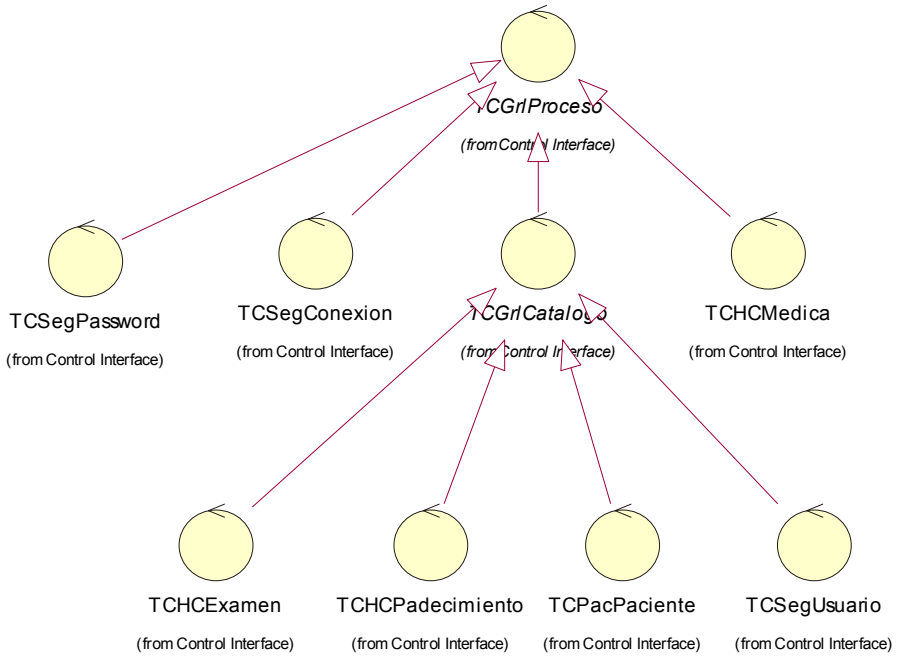
Ahora utilizaremos el diagrama de extensión de casos de uso para generar los elementos de diseño de la propuesta de navegación de los casos de uso y utilizando también una relación de extensión.



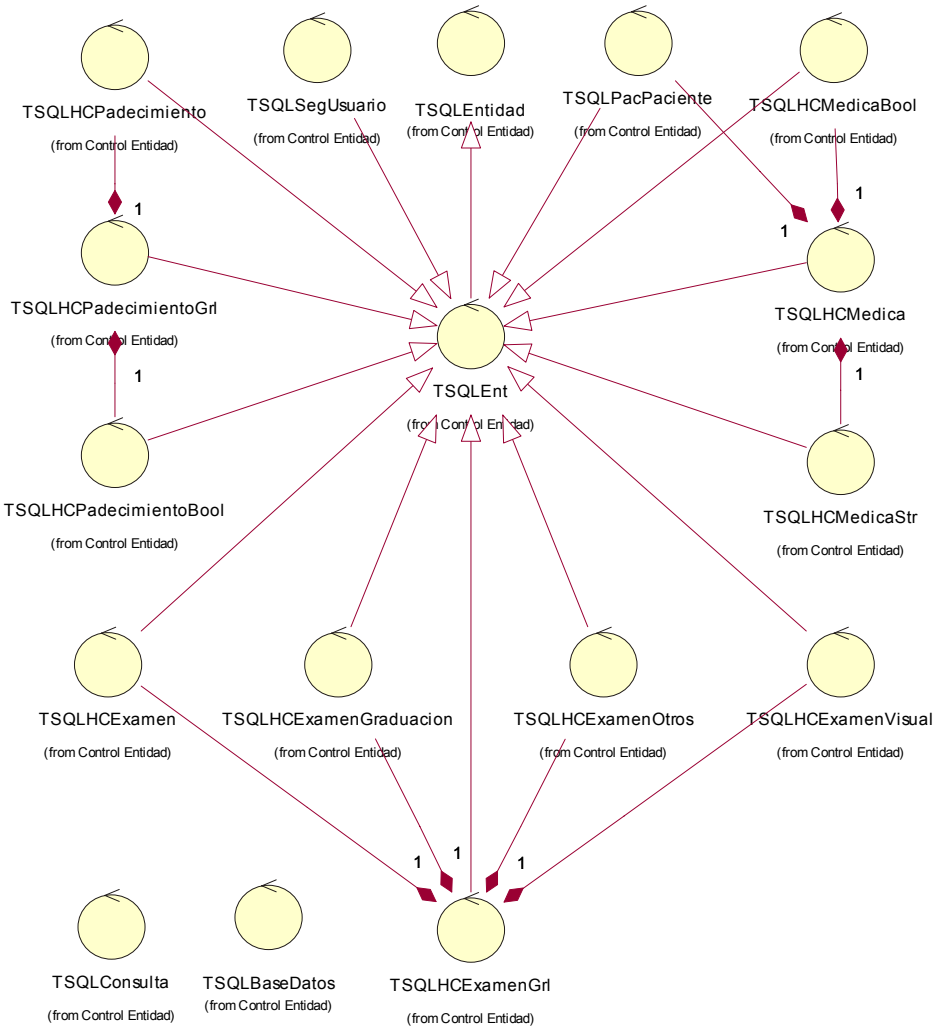

## Clases de Control

Al dividir la aplicación en dos, las clases de control quedan divididas automáticamente en dos clasificaciones: clases de control de interface y clases de control de datos.

Cada caso de uso con interface será resuelto por una clase de control de tipo interface, en donde se controlará la interface en aspectos como validación, estados de pantalla y peticiones de procesos de interface. Adicionalmente, la parte de los casos de uso referente a las reglas de negocio será resuelto por clases de control de entidad de datos, con la habilidad de controlar las clases de entidad de datos para satisfacer las reglas del negocio.



Estas son las clases de control necesarias asociadas a una interface y residirán del lado del cliente.




Los tipos más comunes de relaciones entre clases de control son la herencia, la composición. Aunque la asociación exista se procura no plasmarla en este momento.

Se considera una clase de control de tipo entidad con la habilidad de representar la manipulación de una clase de entidad a través de altas, bajas, cambios.

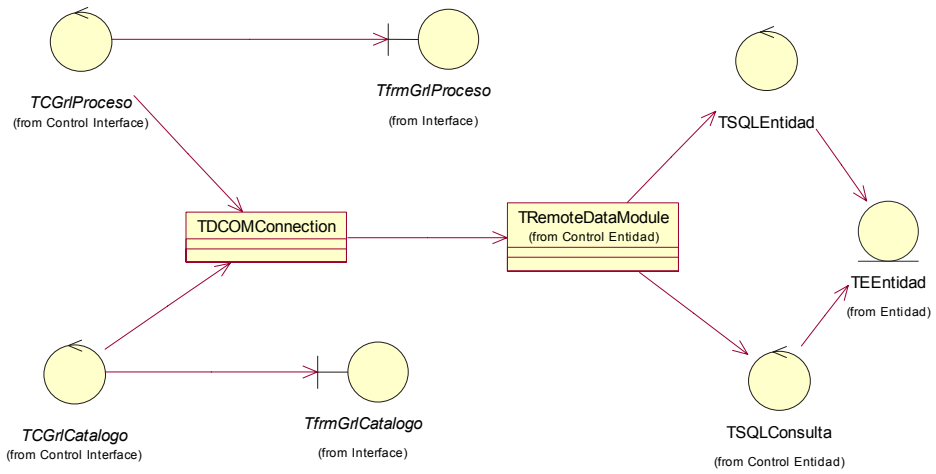
Cada clase de entidad de datos tendrá una clase de control asociada directamente, en algunos casos podemos formar agregaciones o composiciones, para unificar dos clases de control englobando reglas de negocio más complejas y heredaran de la clase de control de tipo entidad.

Se considera una clase de control para conectarse a la base de datos y otra clase de control, para realizar consultas de información a las entidades de datos, el agrupador de estas consultas se realiza a través de una consulta SQL que bien puede realizar un join de entidades de datos, representando una agregación, pero no siendo necesario plasmarla como una clase.

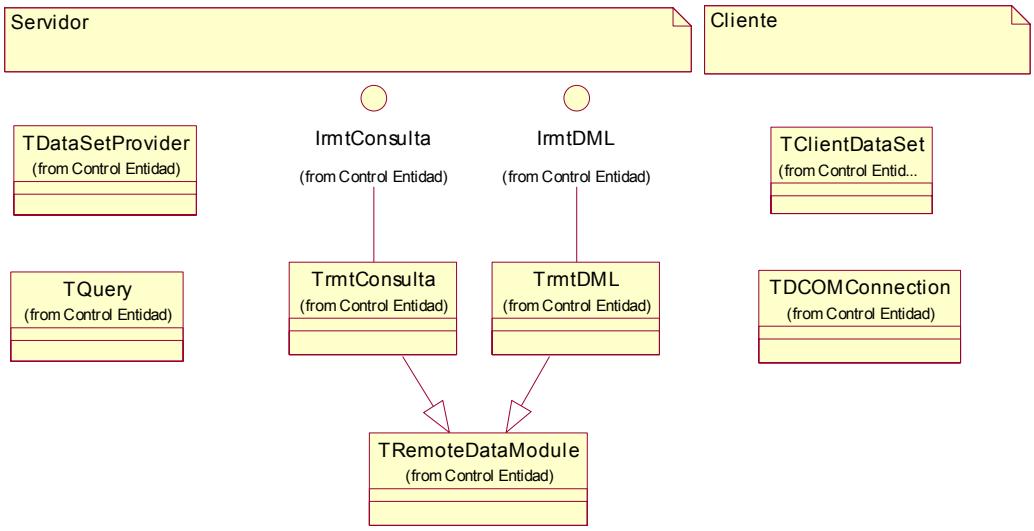
La última etapa es buscar las principales asociaciones entre las clases propuestas, en algunos casos se puede detectar la necesidad de nuevas clases para ayudar a soportar el modelo.

Al tener las relaciones entre clases del mismo tipo, podemos realizar asociaciones entre estos elementos para representar la comunicación entre estos. Para lograr la conexión entre la aplicación del cliente y la aplicación del servidor de reglas de negocio se utilizan clases propias del lenguaje seleccionado (Delphi):

- TDCOMConnection para comunicarse del cliente al servidor
- TDataSetProvider para recibir información a nivel de cursores en el cliente desde el servidor
- TDataSetProvider para transferir información a nivel de cursores del servidor al cliente
- TQuery para realizar la instrucciones SQL a base de datos
- TRemoteDataModule responde a las peticiones del cliente del lado del servidor
- TrmtConsulta: responde a las peticiones de consulta del cliente por parte del servidor.
- TrmtDML: responde a las peticiones de afectación de datos del cliente por parte del servidor.

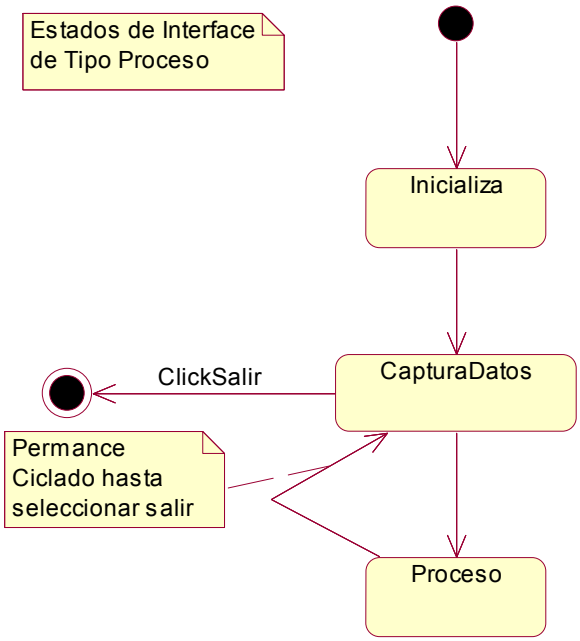


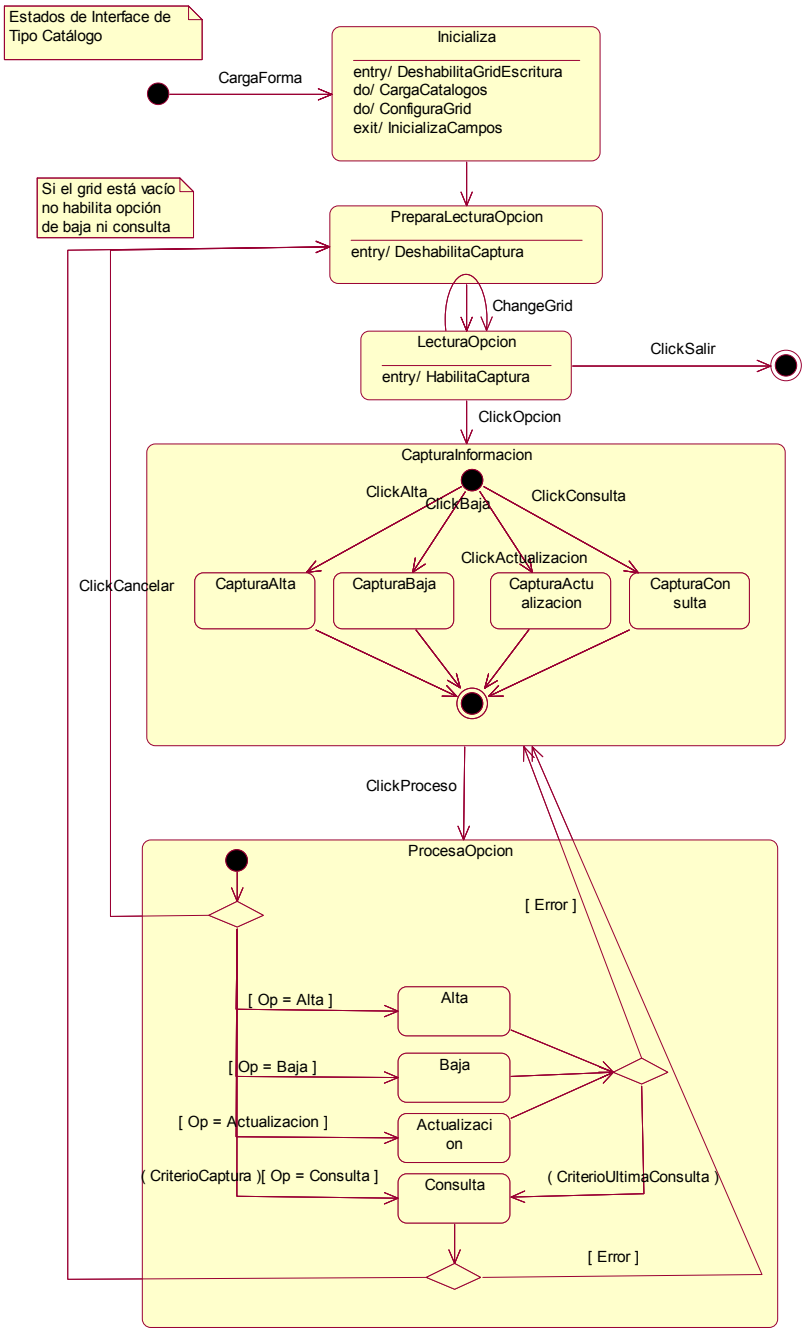
Al realizar estas relaciones e incorporar nuevas clases se plasman relaciones de herencia, agregación, extensión, etc. si hicieran falta.




## Diagramas de Estados

Los diagramas de estados nos pueden ayudar a mostrar los diferentes estados de una clase o conjunto de clases. Una de las partes en donde existe mayor complejidad para un programador es controlar los diferentes estados de las interfaces. Al tener dos tipos de interface (proceso y catálogo), analizaremos los estados de estos tipos de interface.



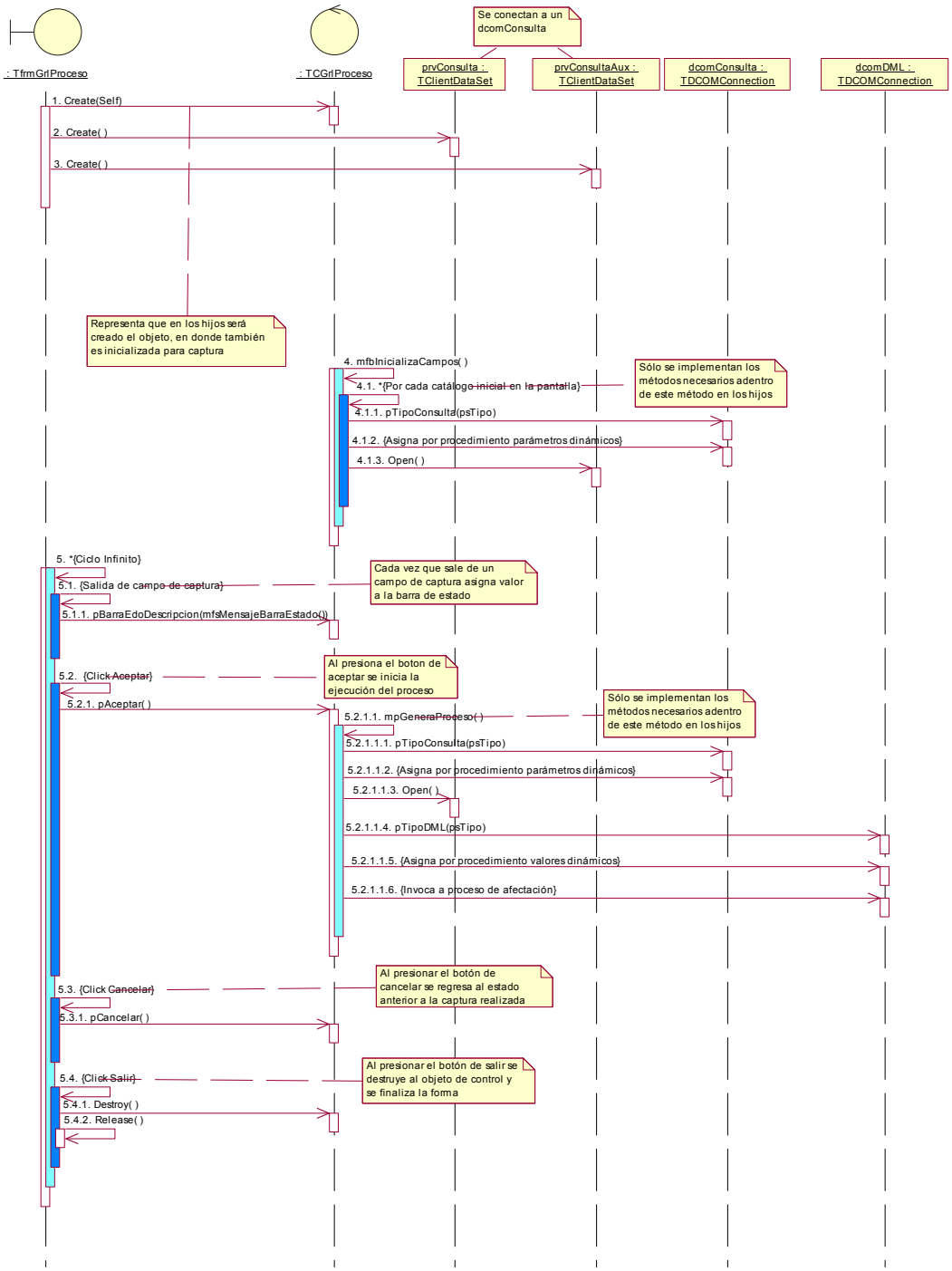



## Diagramas de Secuencias

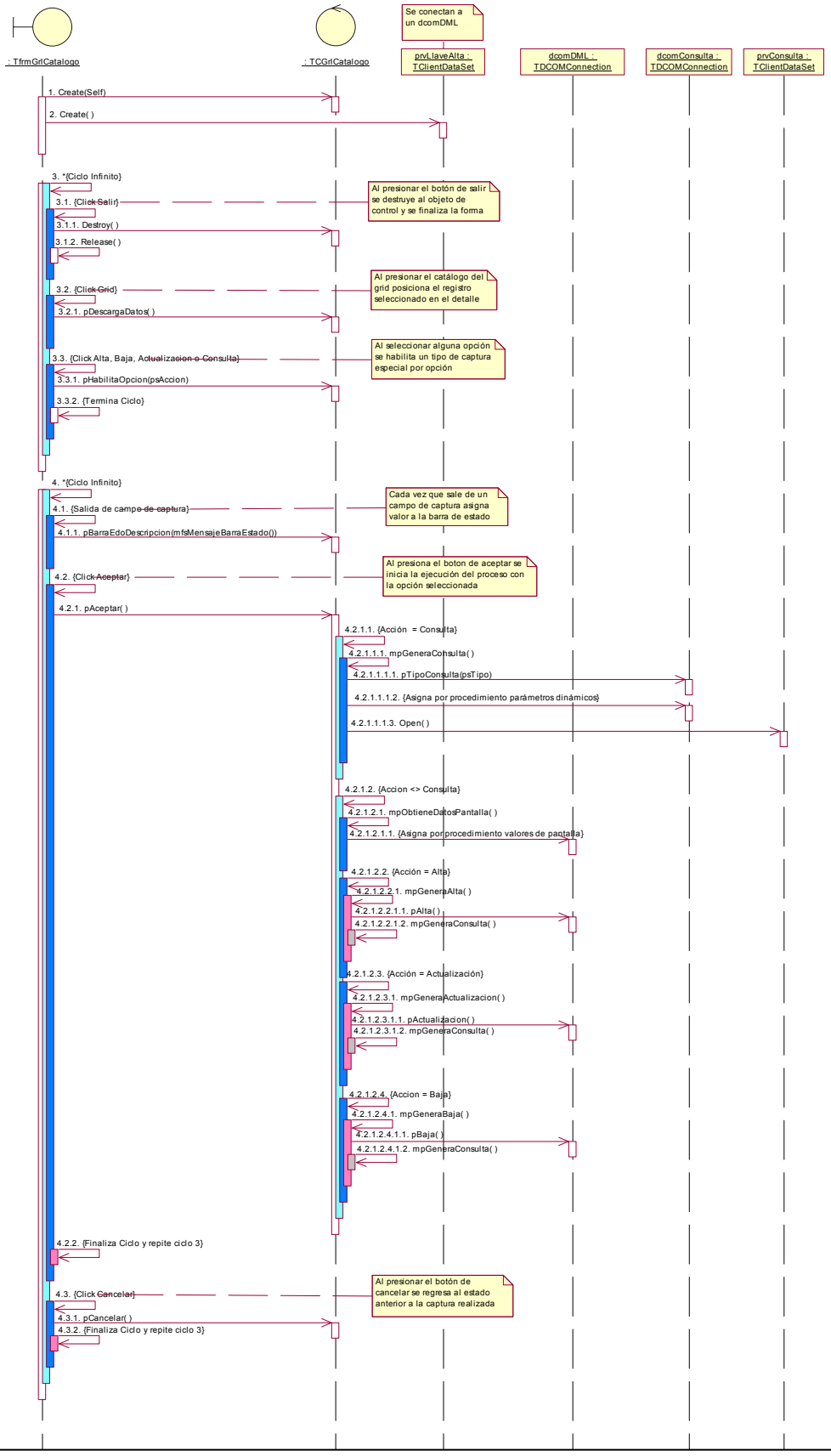
Dentro de la aplicación el menú de navegación tiene la misión de navegar en la aplicación y nos puede ayudar en abrir las conexiones desde el cliente al servidor, evitando conectarnos al ingresar a cada caso de uso al servidor.

La parte más crítica de nuestro diseño es buscar la relación de diferentes objetos para soportar la solución de los casos de uso. Al reducir el funcionamiento de la aplicación a dos tipos principales, catálogo y proceso, bastará con realizar el diagrama de secuencia de estos para soportar el funcionamiento de la mayor parte de casos de uso y sin excedernos en repetir secuencias similares para cada uno de los casos de uso.

Secuencia de un caso de uso de tipo proceso:



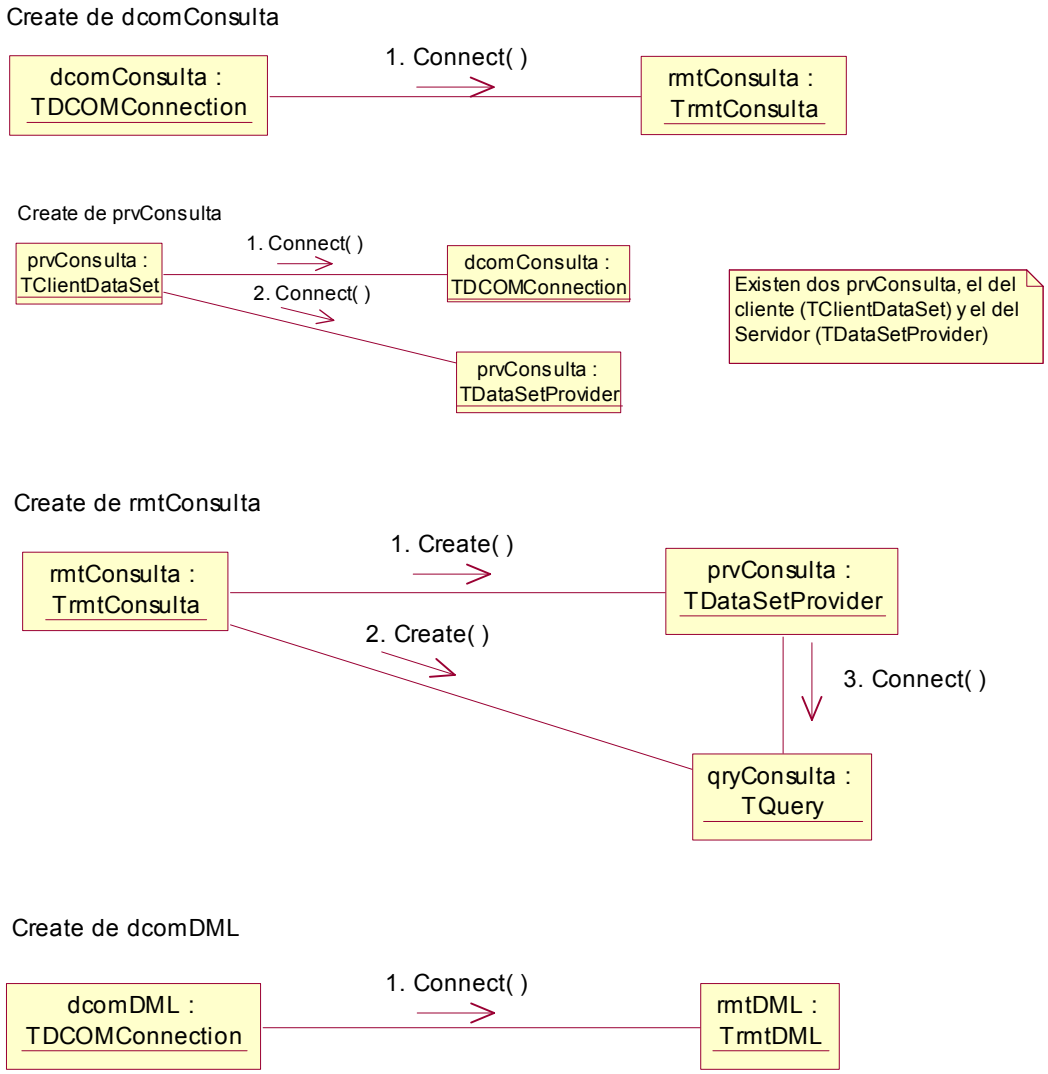

Secuencia de un caso de uso de tipo catálogo:


: TfrmGrlCatalogo
: TCGrlCatalogo
Se conectan a un dcomDML
prvLlaveAlta : TClientDataSet
dcomDML : TDCOMConnection
dcomConsulta : TDCOMConnection
prvConsulta : TClientDataSet
1. Create(Self)
2. Create( )
3. *{Ciclo Infinito}
3.1. {Click Salir}
Al presionar el botón de salir se destruye al objeto de control y se finaliza la forma
3.1.1. Destroy( )
3.1.2. Release( )
3.2. {Click Grid}
Al presionar el catálogo del grid posiciona el registro seleccionado en el detalle
3.2.1. pDescargaDatos( )
3.3. {Click Alta, Baja, Actualizacion o Consulta}
Al seleccionar alguna opción se habilita un tipo de captura especial por opción
3.3.1. pHabilitaOpcion(psAccion)
3.3.2. {Termina Ciclo}
4. *{Ciclo Infinito}
4.1. {Salida de campo de captura}
Cada vez que sale de un campo de captura asigna valor a la barra de estado
4.1.1. pBarraEdoDescripcion(mfsMensajeBarraEstado())
4.2. {Click Aceptar}
Al presiona el boton de aceptar se inicia la ejecución del proceso con la opción seleccionada
4.2.1. pAceptar( )
4.2.1.1. {Acción = Consulta}
4.2.1.1.1. mpGeneraConsulta( )
4.2.1.1.1.1. pTipoConsulta(psTipo)
4.2.1.1.1.2. {Asigna por procedimiento parámetros dinámicos}
4.2.1.1.1.3. Open( )
4.2.1.2. {Accion <> Consulta}
4.2.1.2.1. mpObtieneDatosPantalla( )
4.2.1.2.1.1. {Asigna por procedimiento valores de pantalla}
4.2.1.2.2. {Acción = Alta}
4.2.1.2.2.1. mpGeneraAlta( )
4.2.1.2.2.1.1. pAlta( )
4.2.1.2.2.1.2. mpGeneraConsulta( )
4.2.1.2.3. {Acción = Actualización}
4.2.1.2.3.1. mpGeneraActualizacion( )
4.2.1.2.3.1.1. pActualizacion( )
4.2.1.2.3.1.2. mpGeneraConsulta( )
4.2.1.2.4. {Accion = Baja}
4.2.1.2.4.1. mpGeneraBaja( )
4.2.1.2.4.1.1. pBaja( )
4.2.1.2.4.1.2. mpGeneraConsulta( )
4.2.2. {Finaliza Ciclo y repite ciclo 3}
4.3. {Click Cancelar}
Al presionar el botón de cancelar se regresa al estado anterior a la captura realizada
4.3.1. pCancelar( )
4.3.2. {Finaliza Ciclo y repite ciclo 3}

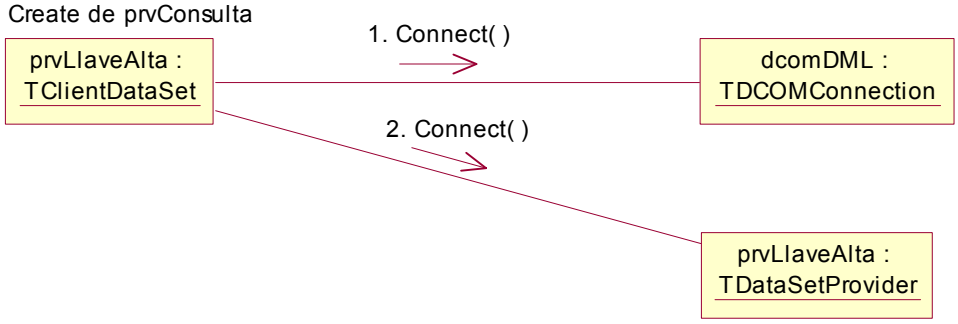
En este caso se detallaron las secuencias internas de cada objeto con el fin de mostrar la incorporación de diferentes elementos de los diagramas de secuencia, además de dejar muy claro el funcionamiento total del grupo de secuencias. El detalle recomendado es plasmar sólo la comunicación entre objetos, y si fuera necesario algunos vínculos hacia el mismo objeto.

## Diagramas de Colaboración

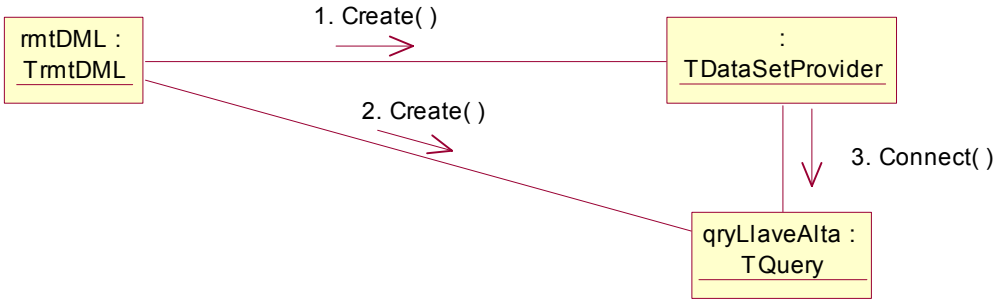
Los diagramas de colaboración nos ayudan para mostrar secuencias más pequeñas para este ejemplo.



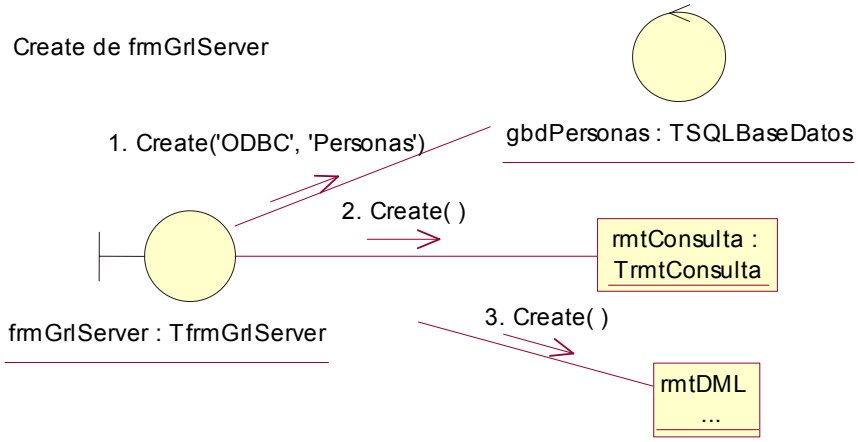
Create de prvConsulta



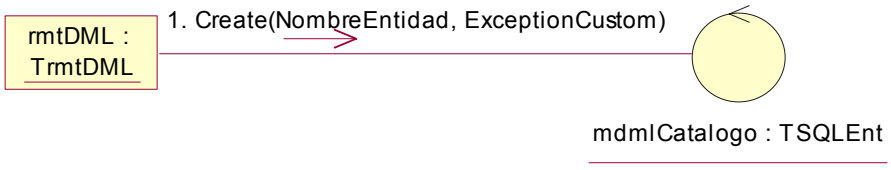

Create de rmtDML

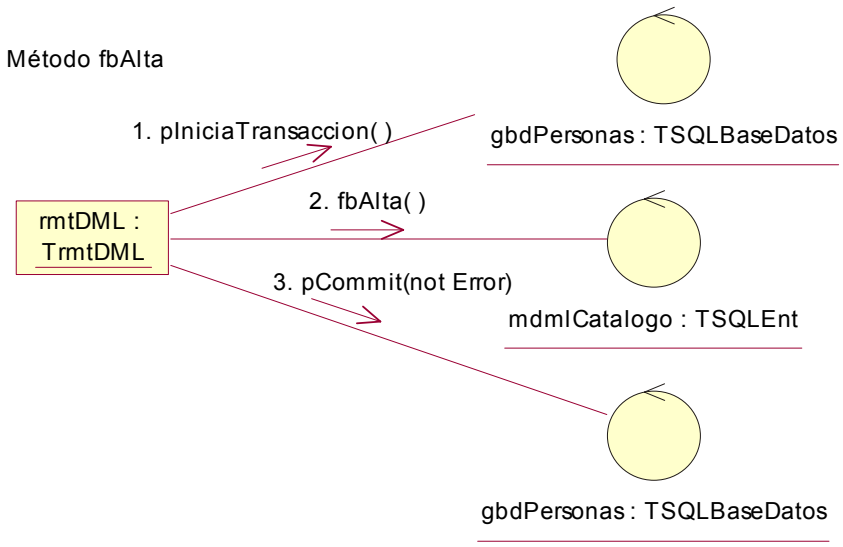


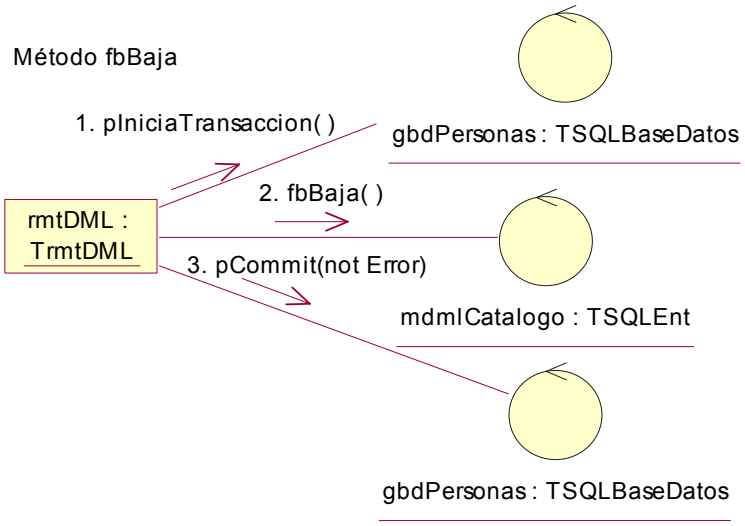

Create de frmGrlServer

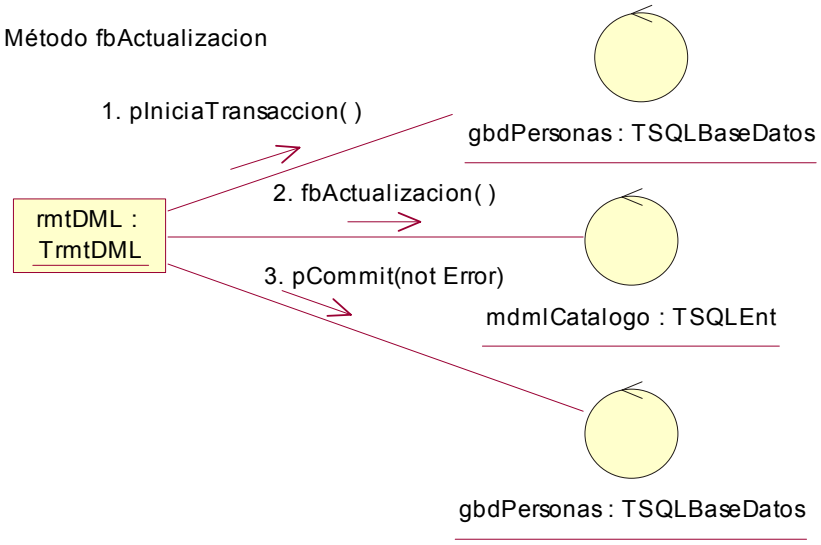



Método pTipoDML




Método fbAlta
1. pIniciaTransaccion( )
gbdPersonas : TSQLBaseDatos
2. fbAlta( )
rmtDML :
TrmtDML
3. pCommit(not Error)
mdmlCatalogo : TSQLEnt
gbdPersonas : TSQLBaseDatos


Método fbBaja
1. pIniciaTransaccion( )
gbdPersonas : TSQLBaseDatos
2. fbBaja( )
rmtDML :
TrmtDML
3. pCommit(not Error)
mdmlCatalogo : TSQLEnt
gbdPersonas : TSQLBaseDatos



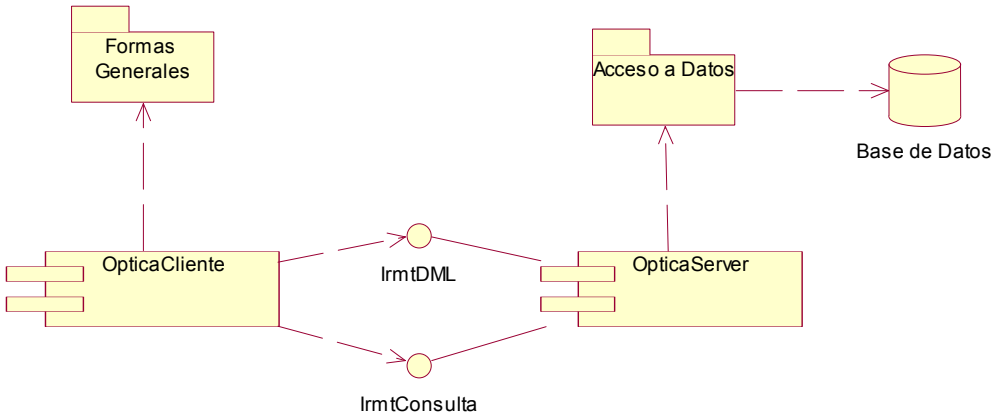

En este caso los diagramas de actividades no fueron necesarios, por eso no se utilizan.

## *Identificación de Métodos y Atributos*

Para identificar los métodos y atributos se pueden desprender de los diagramas utilizados, es decir sólo los públicos. Esta labor también llega a ser común desarrollarse durante la fase de construcción.

## *Diagrama de Componentes*

Lo diagramas de componentes nos ayuda a representar primordialmente la separación de los dos ejecutables el servidor y el cliente.



El componente de OpticaServer tiene dos interfaces para satisfacer a OpticaCliente, OpticaCliente utilizará las Formas Generales. Las Formas Generales contendrán lo relacionado a formas de proceso y catálogo, con el fin de poder ser reutilizado en otros proyectos, algo similar ocurre en el Acceso a Base de Datos.

# CAPÍTULO VI. CONSTRUCCIÓN DEL SISTEMA DE HISTORIA CLÍNICA ÓPTICA

Como parte del resultado de la codificación se mostrarán las clases principales, las interfaces se omitirán por ser las mismas de las especificaciones de casos de uso

## GrlProcesoDCOM.pas

```
unit GrlProcesoDCOM;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, ExtCtrls, ComCtrls, ToolWin, ImgList, ActnList, Db,
  DBClient, MConnect, jpeg;

type
  TCGrlProceso = class; //declaración de forward para permitir usarla en TfrmProceso
  TProcedure = procedure of object;

  TfrmGrlProceso = class(TForm)
    cmdAceptar: TBitBtn;
    cmdCancelar: TBitBtn;
    stbEstatus: TStatusBar;
    imgLogo: TImage;
    barMenu: TControlBar;
    tlbSalir: TToolBar;
    imgProceso: TImageList;
    btnSalir: TToolButton;
    pnlGeneral: TPanel;
    prvConsulta: TClientDataSet;
    prvConsultaAux: TClientDataSet;
    prvLlaveAlta: TClientDataSet;
    procedure btnSalirClick(Sender: TObject);
    procedure cmdAceptarClick(Sender: TObject);
    procedure cmdCancelarClick(Sender: TObject);
    procedure ctlEnter (Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure txtLimpiaBlancosExit (Sender: TObject);
    procedure keyPressNumeroInt (Sender: TObject; var Key: Char);//evita que sea sobreescrito
    procedure keyPressNumeroDec (Sender: TObject; var Key: Char);
    procedure keyPressNumeroIntSig (Sender: TObject; var Key: Char);//evita que sea sobreescrito
    procedure keyPressNumeroDecSig (Sender: TObject; var Key: Char);
    procedure keyUpTel (Sender: TObject; var Key: Word; Shift: TShiftState);
  protected
    mCGrlProceso: TCGrlProceso;
    function mfsMensajeBarraEstado: string; virtual;
  public
    pRegresaCtrAntesDestroy: procedure of object;  {procedimiento que se ejecuta antes de destruirse el objeto, el cual debe ser un
procedimiento de otro objeto}
  end;

  TCGrlProceso = class (TObject)
  private
  {visible sólo solo en la unidad, incluso otras clases}
```

```
 protected
 {visible sólo para la clase y sus hijos, incluso fuera de la unidad}
  mfrmGrlProceso: TfrmGrlProceso;
  function   mfbAbreConsultaDCOM (pprvConsulta: TClientDataSet): boolean; virtual;
  function   mfbEjecutaDMLDCOM (pprcProcedure: TProcedure): boolean; virtual;
  function   mfbInicializaCampos: boolean; virtual; {Carga inicial de campos}
  function   mfbPosicionaCombo (pcboClave, pcboDescripcion: TComboBox; psClave: string): boolean;
  function   mfbValidaDatos: boolean; virtual; abstract;
  function   mfbValidaNumero (psNumero: string): boolean;
  procedure  mpCargaItems (pprvConsulta: TClientDataSet;
                parrsColumnas: array of string;
                parrpntItems: array of pointer); virtual;
  procedure  mpGeneraProceso; virtual; abstract;
  procedure  mpHabilitaBarraOpciones (pbHabilita: boolean); virtual;
  procedure  mpHabilitaBotonesAccion (pbHabilita: boolean); virtual;
  procedure  mpHabilitaCampos (pbHabilita: boolean); virtual; abstract;
  procedure  mpInicializaForma; virtual; {Estado inicial de una forma}
  procedure  mpLimpiaCampos;virtual;abstract; {Asigna valores limpios}
  procedure  mpPreparaLecturaOpcion; virtual;
  procedure  mpProcesoDMLDCOM; virtual;abstract;
  procedure  mpSalirError; virtual;
 public
 {visible para todas las clases}
  constructor Create (pfrmForma: TForm); virtual;
  procedure  pAceptar; virtual;
  procedure  pCancelar; virtual;
  procedure  pBarraEdoDescripcion (psMensaje: string); virtual;
 end;

implementation
uses
 MnuPrincipal;
{rfd no debe referenciar al mnuprincipal}
{$R *.DFM}

{*********************************************************************}
{TCGrlProceso}
constructor TCGrlProceso.Create (pfrmForma: TForm);
begin
 inherited Create;
 mfrmGrlProceso := pfrmForma as TfrmGrlProceso;
 mfrmGrlProceso.pRegresaCtrAntesDestroy:=nil;
 //Habilita el tamaño solicitado para evitar cambios por modo MDI
 with mfrmGrlProceso do
  begin
  //Calcula el tamaño de la pantalla en base a la posición del cmdCancelar
  //Para XP Constante de Height es 57 y Width es 28
  //Para Otros SO Constante de Height es 50 y Width es 13
    Constraints.MaxHeight:= cmdCancelar.Top + cmdCancelar.Height + 57;
    Constraints.MaxWidth:= cmdCancelar.Left + cmdCancelar.Width + 28;
    Height:= Constraints.MaxHeight;
    Width:= Constraints.MaxWidth;
  end;
 mpInicializaForma;
end;

function TCGrlProceso.mfbAbreConsultaDCOM (pprvConsulta: TClientDataSet): boolean;
begin
 Result:= False;
 try
```

```
    pprvConsulta.Close;
    pprvConsulta.Open;
    Result:= True;
  except
   on E: Exception do
    MessageDlg(E.message,mtWarning, [mbOK], 0);
   end;
end;

function TCGrlProceso.mfbEjecutaDMLDCOM (pprcProcedure: TProcedure): boolean;
begin
 Result:= False;
 try
    pprcProcedure;
    Result:= True;
 except
   on E: Exception do
    MessageDlg(E.message,mtWarning, [mbOK], 0);
   end;
end;

function TCGrlProceso.mfbInicializaCampos:boolean;
begin
 mpLimpiaCampos;
 Result:=True;
end;

function TCGrlProceso.mfbPosicionaCombo (pcboClave, pcboDescripcion: TComboBox; psClave: string): boolean;
var
 liNumElemento: integer;
begin
 Result:= False;
 for liNumElemento:= 0 to pcboClave.Items.Count - 1 do
   begin
    if pcboClave.Items[liNumElemento] = psClave then
      begin
       pcboClave.ItemIndex:= liNumElemento;
       if pcboDescripcion <> Nil then
         pcboDescripcion.ItemIndex:= liNumElemento;
       Result:=True;
       break;
      end;
   end;
end;

function TCGrlProceso.mfbValidaNumero (psNumero: string): boolean;
var
 lrNumero: Extended;
begin
 try
   begin
    lrNumero:= StrToFloat (psNumero);
    Result:= True;
   end;
 except
   on E: Exception do
      Result:=False;
 end;

end;
```

```
procedure TCGrlProceso.mpCargaItems (pprvConsulta: TClientDataSet;
                          parrsColumnas: array of string;
                          parrpntItems: array of pointer);
var
 liElemento: word;
 lpntstrItems:^TStrings;
begin
 {Limpia cada una de los items}
 for liElemento:= Low(parrpntItems) to High(parrpntItems) do
  begin
   lpntstrItems:=parrpntItems[liElemento];
   lpntstrItems^.Clear;
  end;
 pprvConsulta.First;
 while (pprvConsulta.EOF = false) do
  begin
   {Pasa por cada apuntador de items y le coloca del registro su columna correspondiente}
   for liElemento:= Low(parrpntItems) to High(parrpntItems) do
    begin
     lpntstrItems:=parrpntItems[liElemento];
     lpntstrItems^.Add(pprvConsulta.FieldByName(parrsColumnas[liElemento]).AsString);
   end;
   pprvConsulta.Next;
  end;
end;

procedure TCGrlProceso.mpHabilitaBarraOpciones (pbHabilita: boolean);
begin
 mfrmGrlProceso.btnSalir.Enabled := pbHabilita;
end;

procedure TCGrlProceso.mpHabilitaBotonesAccion (pbHabilita: boolean);
begin
 mfrmGrlProceso.cmdAceptar.Enabled := pbHabilita;
 mfrmGrlProceso.cmdCancelar.Enabled := pbHabilita;
end;

procedure TCGrlProceso.mpInicializaForma;
begin
 {Si no pudo inicializar la forma}
 If mfbInicializaCampos Then
  mpPreparaLecturaOpcion
 Else
  mpSalirError;
end;

procedure TCGrlProceso.mpPreparaLecturaOpcion;
begin
 mpHabilitaBarraOpciones (True);
 mpHabilitaBotonesAccion (True);
 pBarraEdoDescripcion ('');
end;


procedure TCGrlProceso.mpSalirError;
begin
 mpHabilitaBarraOpciones (False);
 mpHabilitaBotonesAccion (False);
 mpHabilitaCampos (False);
 mpLimpiaCampos;
```

```
  mfrmGrlProceso.btnSalir.Enabled:= True;
end;

procedure TCGrlProceso.pAceptar;
begin
  if mfbValidaDatos then
    begin
      mpGeneraProceso;
    end;
end;

procedure TCGrlProceso.pCancelar;
begin
  mpInicializaForma;
end;

procedure TCGrlProceso.pBarraEdoDescripcion (psMensaje: string);
begin
  mfrmGrlProceso.stbEstatus.Panels[0].Text:= psMensaje;
end;

{*********************************************************************}
{TfrmProceso}
procedure TfrmGrlProceso.btnSalirClick(Sender: TObject);
begin
  Close;
end;

procedure TfrmGrlProceso.cmdAceptarClick(Sender: TObject);
begin
  mCGrlProceso.pAceptar;
end;

procedure TfrmGrlProceso.cmdCancelarClick(Sender: TObject);
begin
  mCGrlProceso.pCancelar;
end;

procedure TfrmGrlProceso.keyPressNumeroInt (Sender: TObject; var Key: Char);
begin
  if not (Key in ['0'..'9', chr(VK_BACK)]) then
    begin
      Key:= chr(1);
    end;
end;

procedure TfrmGrlProceso.keyPressNumeroDec (Sender: TObject; var Key: Char);
begin
  if not (Key in ['0'..'9', '.', chr(VK_BACK)]) then
    begin
      Key:= chr(1);
    end;
end;

procedure TfrmGrlProceso.keyPressNumeroIntSig (Sender: TObject; var Key: Char);
begin
  if not (Key in ['0'..'9', '+', '-', chr(VK_BACK)]) then
    begin
      Key:= chr(1);
    end;
```

```
end;

procedure TfrmGrlProceso.keyPressNumeroDecSig (Sender: TObject; var Key: Char);
begin
  if not (Key in ['0'..'9', '.', '+', '-', chr(VK_BACK)]) then
    begin
      Key:= chr(1);
    end;
end;

procedure TfrmGrlProceso.keyUpTel(Sender: TObject; var Key: Word;
  Shift: TShiftState);
var
  liLongitud:integer;
  lsTexto: string;
  liPosicion: integer;
  liExisteGuion: integer;
  ltxtTelefono: TEdit;
begin
  ltxtTelefono:= Sender As TEdit;
  lsTexto:= ltxtTelefono.Text;
  //Almacena la posición del cursor
  liPosicion:= ltxtTelefono.SelStart;
  //Depura todos los guiones del telefono
  liExisteGuion:= Pos('-',lsTexto);
  while liExisteGuion <> 0 do
    begin
      Delete (lsTexto, liExisteGuion, 1);
      liExisteGuion:= Pos('-',lsTexto);
      liPosicion:= liPosicion - 1;
    end;

  liLongitud:= Length( lsTexto);
  //Pone un guión en la cuarta posición si existen más de cuatro números
  if liLongitud > 4 then
    begin
      if lsTexto[5] <> '-' then
        begin
          Insert('-', lsTexto, 5);
          liPosicion:= liPosicion + 1;
        end;
    end;
  //Pone un guión en la octava posición si existen más de ocho números
  if liLongitud > 8 then
    begin
      if lsTexto[10] <> '-' then
        begin
          Insert('-', lsTexto, 10);
          liPosicion:= liPosicion + 1;
        end;
    end;
    //Cuando se elimina una posicion en donde se quita un guion
    //incrementa la posicion inicial eliminada al inicio
    if (key in [VK_BACK, VK_DELETE])
      and ((liLongitud = 8) or (liLongitud = 4)) then
      liPosicion:= liPosicion + 1;
    ltxtTelefono.Text:= lsTexto;
    ltxtTelefono.SelStart:= liPosicion;
end;
```

```
procedure TfrmGrlProceso.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 mCGrlProceso.Destroy;
 {Si se le asigno valor a pRegresaCtrAntesDestroy se ejecuta}
 if @pRegresaCtrAntesDestroy <> nil then
  begin
     pRegresaCtrAntesDestroy;
  end;
 Release;
end;

procedure TfrmGrlProceso.txtLimpiaBlancosExit (Sender: TObject);
begin
 with (Sender as TEdit) do
  begin
    Text:= Trim(Text);
  end;
end;

procedure TfrmGrlProceso.ctlEnter(Sender: TObject);
begin
 mCGrlProceso.pBarraEdoDescripcion (mfsMensajeBarraEstado);
end;

function TfrmGrlProceso.mfsMensajeBarraEstado: string;
begin
 Result:= '';
end;

end.
```

# GrlCatalogoDCOM.pas

```
unit GrlCatalogoDCOM;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 GrlProcesoDCOM, ImgList, ExtCtrls, ComCtrls, ToolWin, StdCtrls, Buttons,
 Grids, DBGrids, Db, DBClient, MConnect, jpeg;

type
 TCGrlCatalogo = class;

 TfrmGrlCatalogo = class(TfrmGrlProceso)
  tlbCatalogo: TToolBar;
  btnConsulta: TToolButton;
  btnAlta: TToolButton;
  btnBaja: TToolButton;
  btnActualizar: TToolButton;
  dtsCatalogo: TDataSource;
  grdCatalogo: TDBGrid;
  procedure FormCreate(Sender: TObject);
  procedure btnConsultaClick(Sender: TObject);
  procedure btnAltaClick(Sender: TObject);
  procedure btnBajaClick(Sender: TObject);
  procedure btnActualizarClick(Sender: TObject);
```

```
  procedure dtsCatalogoDataChange(Sender: TObject; Field: TField);
 protected
  mCGrlCatalogo: TCGrlCatalogo;
 public
  { Public declarations }
 end;

 TCGrlCatalogo = class (TCGrlProceso)
 protected
  mfrmGrlCatalogo: TfrmGrlCatalogo;
  msAccion: string;
  function   mfbEjecutaDMLDCOM (pprcProcedure: TProcedure): boolean; override;
  function   mfbValidaDatosCambio: boolean; virtual; abstract;
  function   mfbValidaDatosConsulta: boolean; virtual; abstract;
  procedure  mpActualizacionDCOM; virtual;
  procedure  mpAltaDCOM; virtual;
  procedure  mpBajaDCOM; virtual;
  procedure  mpBarraEdoAccion (psAccion: string); virtual;
  procedure  mpConstruyeConsulta; virtual; abstract;{construye query y parametros}
  procedure  mpGeneraActualizacion; virtual;
  procedure  mpGeneraAlta; virtual;
  procedure  mpGeneraBaja; virtual;
  procedure  mpGeneraConsulta; virtual;
  procedure  mpHabilitaBarraOpciones (pbHabilita: boolean); override;
  procedure  mpHabilitaCamposAlta; virtual;
  procedure  mpHabilitaCamposActualizacion; virtual; abstract;
  procedure  mpHabilitaCamposConsulta; virtual; abstract;
  procedure  mpPreparaLecturaOpcion; override;
  procedure  mpObtieneDatosPantalla; virtual; abstract;
  procedure  mpTransfiereGridCampos; virtual; abstract;
 public
  constructor Create (pfrmForma: TForm); override;
  procedure  pAceptar; override;
  procedure  pCancelar; override;
  procedure  pDescargaDatos; virtual;
  procedure  pHabilitaOpcion (psAccion: string); virtual;
 end;

implementation
uses MnuPrincipal;
{rfdno debe referenciar al menu principal}
{$R *.DFM}

{*********************************************************************}
{TCGrlCatalogo}
constructor TCGrlCatalogo.Create (pfrmForma: TForm);
begin
 mfrmGrlCatalogo:= pfrmForma as TfrmGrlCatalogo;
 mfrmGrlCatalogo.dtsCatalogo.DataSet:=  mfrmGrlCatalogo.prvConsulta;
 inherited;
end;

function TCGrlCatalogo.mfbEjecutaDMLDCOM (pprcProcedure: TProcedure): boolean;
begin
 mpObtieneDatosPantalla;
 Result:= inherited mfbEjecutaDMLDCOM (pprcProcedure);
end;

procedure TCGrlCatalogo.mpActualizacionDCOM;
begin
```

```
 frmMnuPrincipal.dcomDML.AppServer.pActualizacion;
end;

procedure TCGrlCatalogo.mpAltaDCOM;
begin
 frmMnuPrincipal.dcomDML.AppServer.pAlta;
end;

procedure TCGrlCatalogo.mpBajaDCOM;
begin
 frmMnuPrincipal.dcomDML.AppServer.pBaja;
end;

procedure TCGrlCatalogo.mpBarraEdoAccion (psAccion: string);
begin
 mfrmGrlProceso.stbEstatus.Panels[1].Text:= psAccion;
end;

procedure TCGrlCatalogo.mpGeneraActualizacion;
begin
 if mfbValidaDatosCambio then
    if mfbEjecutaDMLDCOM (mpActualizacionDCOM) then mpGeneraConsulta;
end;

procedure TCGrlCatalogo.mpGeneraAlta;
begin
 if mfbValidaDatosCambio then
    if mfbEjecutaDMLDCOM (mpAltaDCOM) then mpGeneraConsulta;
end;

procedure TCGrlCatalogo.mpGeneraBaja;
begin
 if mfbEjecutaDMLDCOM (mpBajaDCOM) then mpGeneraConsulta;
end;

procedure TCGrlCatalogo.mpGeneraConsulta;
begin
 if mfbValidaDatosConsulta then
   begin
    //Asigna tipo de consulta y parámetros
    mpConstruyeConsulta;
    if mfbAbreConsultaDCOM (mfrmGrlCatalogo.prvConsulta) then
     begin
       pDescargaDatos;
       mpPreparaLecturaOpcion;
     end;
   end;
end;

procedure TCGrlCatalogo.mpHabilitaBarraOpciones (pbHabilita: boolean);
begin
 inherited;
 mfrmGrlCatalogo.btnConsulta.Enabled:= pbHabilita;
 mfrmGrlCatalogo.btnAlta.Enabled:= pbHabilita;
 mfrmGrlCatalogo.btnBaja.Enabled:= pbHabilita;
 mfrmGrlCatalogo.btnActualizar.Enabled:= pbHabilita;
 //Las condiciones no se puden poner juntas
 //Si no se ha realizado consulta inicial
 if mfrmGrlCatalogo.dtsCatalogo.DataSet = nil Then
   begin
```

```
    mfrmGrlCatalogo.btnBaja.Enabled:= False;
    mfrmGrlCatalogo.btnActualizar.Enabled:= False;
   end
 //Si no tiene registros
 else if mfrmGrlCatalogo.prvConsulta.IsEmpty then
   begin
    mfrmGrlCatalogo.btnBaja.Enabled:= False;
    mfrmGrlCatalogo.btnActualizar.Enabled:= False;
   end
end;

procedure TCGrlCatalogo.mpHabilitaCamposAlta;
begin
 mpHabilitaCampos (True);
end;

procedure TCGrlCatalogo.mpPreparaLecturaOpcion;
begin
 inherited;
 mpHabilitaBotonesAccion (False);
 mpHabilitaCampos(False);
 mfrmGrlCatalogo.grdCatalogo.Enabled:= True;
end;

procedure TCGrlCatalogo.pAceptar;
begin
 if     msAccion = ksActConsulta     then  mpGeneraConsulta
 else
   begin
    if msAccion     = ksActAlta         then mpGeneraAlta
    else if msAccion = ksActBaja         then mpGeneraBaja
    else if msAccion = ksActActualizacion then mpGeneraActualizacion
   end;
end;

procedure TCGrlCatalogo.pCancelar;
begin
 pDescargaDatos;
 mpPreparaLecturaOpcion;
end;

procedure TCGrlCatalogo.pDescargaDatos;
begin
 if mfrmGrlCatalogo.prvConsulta.IsEmpty then
   begin
    mpLimpiaCampos;
   end
 else
   begin
    mpTransfiereGridCampos;
   end;
end;

procedure TCGrlCatalogo.pHabilitaOpcion (psAccion: string);
begin
 msAccion:= psAccion;
 mpBarraEdoAccion (msAccion);

 //Consulta
 if msAccion = ksActConsulta then
```

```
   begin
     mpLimpiaCampos;
     mpHabilitaCamposConsulta;
   end
  //Alta
  else if msAccion = ksActAlta then
   begin
     mpLimpiaCampos;
     mpHabilitaCamposAlta;
   end
  //Baja
  else if msAccion = ksActBaja then
   begin
     mpHabilitaCampos(False);
   end
  //Actualizacion
  else if msAccion = ksActActualizacion then
   begin
     mpHabilitaCamposActualizacion;
   end;
  mpHabilitaBarraOpciones (False);
  mpHabilitaBotonesAccion (True);
  mfrmGrlCatalogo.grdCatalogo.Enabled:= False;
end;
{*********************************************************************}
{TfrmGrlCatalogo}
procedure TfrmGrlCatalogo.FormCreate(Sender: TObject);
begin
  inherited;
  mCGrlProceso:= mCGrlCatalogo;
end;

procedure TfrmGrlCatalogo.btnConsultaClick(Sender: TObject);
begin
  inherited;
  mCGrlCatalogo.pHabilitaOpcion (ksActConsulta);
end;

procedure TfrmGrlCatalogo.btnAltaClick(Sender: TObject);
begin
  inherited;
  mCGrlCatalogo.pHabilitaOpcion (ksActAlta);
end;

procedure TfrmGrlCatalogo.btnBajaClick(Sender: TObject);
begin
  inherited;
  mCGrlCatalogo.pHabilitaOpcion (ksActBaja);
end;

procedure TfrmGrlCatalogo.btnActualizarClick(Sender: TObject);
begin
  inherited;
  mCGrlCatalogo.pHabilitaOpcion (ksActActualizacion);
end;

procedure TfrmGrlCatalogo.dtsCatalogoDataChange(Sender: TObject;
  Field: TField);
begin
  inherited;
```

```
 if mCGrlCatalogo <> nil then
   begin
    mCGrlCatalogo.pDescargaDatos;
   end;
end;

end.
```

# SegUsuario.pas

```
unit SegUsuario;

interface

uses
 Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 GrlCatalogoDCOM, Db, DBClient, ImgList, Grids, DBGrids, ExtCtrls,
 ComCtrls, ToolWin, StdCtrls, Buttons, jpeg;

type
 TCSegUsuario = class;

 TfrmSegUsuario = class(TfrmGrlCatalogo)
   lblUsuario: TStaticText;
   lblPassword: TStaticText;
   txtUsuario: TEdit;
   txtPassword: TEdit;
   lblNombre: TStaticText;
   txtNombre: TEdit;
   lblPerfil: TStaticText;
   cboPerfil: TComboBox;
   cboCvePerfil: TComboBox;
   procedure FormCreate(Sender: TObject);
 protected
   mCSegUsuario: TCSegUsuario;
   function mfsMensajeBarraEstado: string; override;
 end;

 TCSegUsuario = class(TCGrlCatalogo)
 protected
   mfrmSegUsuario: TfrmSegUsuario;
   function    mfbInicializaCampos: boolean; override;
   function    mfbValidaDatosCambio: boolean; override;
   function    mfbValidaDatosConsulta: boolean; override;
   procedure   mpConstruyeConsulta; override;
   procedure   mpHabilitaCampos (pbHabilita: boolean); override;
   procedure   mpHabilitaCamposActualizacion; override;
   procedure   mpHabilitaCamposConsulta; override;
   procedure   mpLimpiaCampos; override;
   procedure   mpObtieneDatosPantalla; override;
   procedure   mpTransfiereGridCampos; override;
 public
   constructor Create (pfrmForma: TForm); override;
 end;


implementation

uses
 MnuPrincipal, GrlConstantesDCOM;
```

```
{$R *.DFM}
{*********************************************************************}
{TCSegUsuario}
constructor TCSegUsuario.Create (pfrmForma: TForm);
begin
  mfrmSegUsuario:= pfrmForma as TfrmSegUsuario;
  inherited;
  frmMnuPrincipal.dcomDML.AppServer.pTipoDML(ksDMLSegUsuario);
end;

function TCSegUsuario.mfbInicializaCampos:boolean;
begin
  frmMnuPrincipal.dcomConsulta.AppServer.pTipoConsulta(ksSQLGrlPerfil);
  //Si logra abrir la consulta
  Result:= mfbAbreConsultaDCOM (mfrmSegUsuario.prvConsultaAux);

  if Result then
    mpCargaItems (mfrmSegUsuario.prvConsultaAux, ['cClave','cDescripcion'],
              [@mfrmSegUsuario.cboCvePerfil.Items,
              @mfrmSegUsuario.cboPerfil.Items]);
  inherited mfbInicializaCampos;
end;

function TCSegUsuario.mfbValidaDatosCambio: boolean;
begin
  Result:= False;
  if mfrmSegUsuario.txtUsuario.Text = '' Then
    begin
      mfrmSegUsuario.txtUsuario.SetFocus;
      MessageDlg('Debe digitar el usuario',mtInformation, [mbOK], 0);
    end
  else if mfrmSegUsuario.txtNombre.Text = '' Then
    begin
      mfrmSegUsuario.txtNombre.SetFocus;
      MessageDlg('Debe digitar el nombre del usuario',mtInformation, [mbOK], 0);
    end
  else
    Result:= True;

 if (mfrmSegUsuario.txtPassword.Text = '') and (msAccion = ksActAlta) Then
   begin
     mfrmSegUsuario.txtPassword.SetFocus;
     MessageDlg('Debe digitar el password del usuario',mtInformation, [mbOK], 0);
     Result:= False;
   end

end;

function TCSegUsuario.mfbValidaDatosConsulta: boolean;
begin
  Result:= False;
  if     (mfrmSegUsuario.txtUsuario.Text = '')
     and (mfrmSegUsuario.txtNombre.Text = '')
     and (mfrmSegUsuario.cboPerfil.ItemIndex = -1) Then
    begin
      mfrmSegUsuario.txtUsuario.SetFocus;
      MessageDlg('Debe capturar algún campo',mtInformation, [mbOK], 0);
    end
  else
```

```
   Result:= True;
end;

procedure TCSegUsuario.mpConstruyeConsulta;
begin
  with mfrmSegUsuario do
    begin
      //Asigna llave a la alta
      if msAccion = ksActAlta then
        begin
          frmMnuPrincipal.dcomConsulta.AppServer.pTipoConsulta(ksSQLSegUsuario);
          prvLlaveAlta.FetchParams;
          prvConsulta.Params.Assign (prvLlaveAlta.Params);
        end
      else if msAccion = ksActConsulta then
        begin
          mpObtieneDatosPantalla;
        end;
    end;
end;

procedure TCSegUsuario.mpHabilitaCampos (pbHabilita: boolean);
begin
  with mfrmSegUsuario do
    begin
      txtUsuario.Enabled:= pbHabilita;
      txtPassword.Enabled:= pbHabilita;
      cboPerfil.Enabled:= pbHabilita;
      txtNombre.Enabled:= pbHabilita;
    end;
end;

procedure TCSegUsuario.mpHabilitaCamposActualizacion;
begin
  with mfrmSegUsuario do
    begin
      mpHabilitaCampos (True);
      txtUsuario.Enabled:= False;
    end;
end;

procedure TCSegUsuario.mpHabilitaCamposConsulta;
begin
  mpHabilitaCampos (True);
  with mfrmSegUsuario do
    begin
      cboPerfil.ItemIndex:= -1;
    end;
end;

procedure TCSegUsuario.mpLimpiaCampos;
begin
  with mfrmSegUsuario do
    begin
      txtUsuario.Text:= '';
      txtPassword.Text:= '';
      cboPerfil.ItemIndex:= 0;
      txtNombre.Text:= '';
    end;
end;
```

```
procedure TCSegUsuario.mpObtieneDatosPantalla;
var
 lsUsuario, lsPassword, lsPerfil, lsNombre: WideString;
begin
 with mfrmSegUsuario do
  begin
   lsUsuario:=txtUsuario.Text;
   lsPassword:= txtPassword.Text;
   lsNombre:= txtNombre.Text;
   if cboPerfil.ItemIndex = -1 then
    lsPerfil:= ''
   else
    lsPerfil:= cboCvePerfil.Items[cboPerfil.ItemIndex];
  end;
 if msAccion = ksActConsulta then
  frmMnuPrincipal.dcomConsulta.AppServer.pSetSegUsuarioParam
                          (lsUsuario, lsPerfil, lsNombre)
 else
  frmMnuPrincipal.dcomDML.AppServer.pSetSegUsuario
                  (lsUsuario, lsPassword, lsPerfil, lsNombre);
end;

procedure TCSegUsuario.mpTransfiereGridCampos;
begin
 with mfrmSegUsuario do
  begin
   with prvConsulta do
    begin
     txtUsuario.Text:= FieldByName('cClave').AsString;
     txtPassword.Text:= '';
     mfbPosicionaCombo (cboCvePerfil, cboPerfil, FieldByName('cPerfil').AsString);
     txtNombre.Text:= FieldByName('cNombre').AsString;;
    end;
  end;
end;

{*********************************************************************}
{TfrmSegUsuario}
procedure TfrmSegUsuario.FormCreate(Sender: TObject);
begin
 mcSegUsuario:= TCSegUsuario.Create(Self);
 mCGrlCatalogo:= mCSegUsuario;
 inherited;
end;

function TfrmSegUsuario.mfsMensajeBarraEstado: string;
begin
 Result:= inherited mfsMensajeBarraEstado;
 if ActiveControl.Name = 'txtUsuario' then
  Result:= 'Clave de usuario en el sistema'
 else if ActiveControl.Name = 'txtPassword' then
  Result:= 'Contraseña de acceso a la aplicación'
 else if ActiveControl.Name = 'cboPerfil' then
  Result:= 'Nivel de seguridad otorgada al usuario'
 else if ActiveControl.Name = 'txtNombre' then
  Result:= 'Nombre completo del usuario'
end;

end.
```

# SQLMngBD.pas

```
unit SQLMngBD;

interface
uses
 BDE, Classes, Controls, DB, DBTables, Dialogs, SysUtils, StdCtrls;

type
 TMsgDML = (tMsgDMLMensaje, tMsgDMLExceptionCustom, tMsgDMLExceptionNative);

 TSQLBaseDatos = class (TObject)
 private
  hbConexion: boolean; {Indica si está conectado}
  hdbBD: TDataBase; {Base de datos}
  {Nombre de la base de datos}
  hsNombreBaseDatos: string;
 public
  constructor Create(psAliasName, psNombreLogico: string); virtual;
  destructor  Destroy; override;
  property    abConectado: boolean read hbConexion; {checar si es necesario una propiedas que se tiene en el objeto}
  property    asNombre: string read hsNombreBaseDatos;
  procedure   pConectar (pbPromptLogin: boolean; psUsuario, psPassword, psParametros: string); virtual; {Conexión a la base de
datos}
  procedure   pCommit (pbCommit: boolean); virtual; {Commit o Rollback}
  procedure   pIniciaTransaccion; virtual; {Inicia una transacción}
 end;

 TSQLConsulta = class (TObject)
 private
 protected
 public
  lqrySQL: TQuery;
  constructor Create (psBD: string); overload; virtual;
  constructor Create (psBD: string; pstrSQL: TStrings); overload; virtual;
  destructor  Destroy; override;
  function    fbAbre (pstrListaInstruccion: TStrings): boolean; virtual;
  procedure   pCargaItems (parrsColumnas: array of string; parrpntItems: array of pointer); virtual;
  procedure   pCierra; virtual;
 end;

 TSQLEntidad = class (TObject)
 private
 protected
  mmsgError: TMsgDML;
  mqryDML: TQuery;
  function    mfbEjecutaSQL (pstrListaInstruccion: TStrings): boolean; virtual;
  function    mfsErrores (perrError: EDBEngineError): string; virtual;
  procedure   mpGeneraError (perrError: EDBEngineError); virtual;
 public
  lprmLlave: TParams;
  constructor Create (psBD: string; pmsgError: TMsgDML); virtual;
  destructor  Destroy; override;
  function    fbAlta: boolean; virtual; abstract;
  function    fbBaja: boolean; virtual; abstract;
  function    fbActualizacion: boolean; virtual; abstract;
  function    fbModificacion: boolean; virtual;
```

```
 end;


implementation
{*************************************************************************}
{Objeto TSQLBaseDatos}
constructor TSQLBaseDatos.Create(psAliasName, psNombreLogico: string);
{Es necesario que el alias se declare como servidor para activar el bloqueo dinámico de registros}
begin
 inherited Create;
 ShortDateFormat:= 'dd/mm/yyyy';
 ShortTimeFormat:= 'hh:mm';
 hsNombreBaseDatos:= psNombreLogico;
 hdbBD:= TDataBase.Create(hdbBD);
 hbConexion:= False;
 with hdbBD do
  begin
   Connected:= False;
   DataBaseName:= hsNombreBaseDatos;
   AliasName:= psAliasName;
   KeepConnection:= True;
   TransIsolation:= tiReadCommitted;
  end;
end;

destructor TSQLBaseDatos.Destroy;
begin
 inherited;
 hdbBD.Connected:= False;
 hdbBD.Destroy;
end;


procedure TSQLBaseDatos.pConectar(pbPromptLogin: boolean; psUsuario, psPassword, psParametros: string);
begin
 with hdbBD do
  begin
   LoginPrompt:= pbPromptLogin;
   Params.Clear;
   Params.Add ('USER NAME=' + psUsuario);
   Params.Add ('PASSWORD=' + psPassword);
   Params.Add (psParametros);
   try
    Open;
   except
    on E: EDBEngineError do
     begin
      Connected:= False;
      MessageDlg (E.message,mtWarning, [mbOK], 0);
     end;
    on E: EDatabaseError do
     begin
      MessageDlg (E.message,mtWarning, [mbOK], 0);
      Connected:= False;
     end;
   end;
  end;
 hbConexion:= hdbBD.Connected;
end;
```

```
procedure TSQLBaseDatos.pCommit (pbCommit: boolean);
begin
 if pbCommit then
  hdbBD.Commit
 else
  hdbBD.Rollback;
end;

procedure TSQLBaseDatos.pIniciaTransaccion;
begin
 hdbBD.StartTransaction;
end;

{*********************************************************************}
{Objeto TSQLConsulta}
constructor TSQLConsulta.Create (psBD: string);
begin
 inherited Create;
 lqrySQL:= TQuery.Create(lqrySQL);
 lqrySQL.DatabaseName:= psBD;
end;

constructor TSQLConsulta.Create (psBD: string; pstrSQL: TStrings);
begin
 Create (psBD);
 lqrySQL.SQL:= pstrSQL;
end;

destructor TSQLConsulta.Destroy;
begin
 inherited;
 lqrySQL.Destroy;
end;

function TSQLConsulta.fbAbre (pstrListaInstruccion: TStrings): boolean;
var
 liParametro: integer;
begin
 Result:= False;
 try
  if pstrListaInstruccion <> Nil then
   begin
    pstrListaInstruccion.Assign (lqrySQL.SQL);
    for liParametro:= 0 to lqrySQL.ParamCount - 1 do
     begin
      pstrListaInstruccion.Add (lqrySQL.Params.Items[liParametro].Name
                + ': ' + lqrySQL.Params.Items[liParametro].AsString);
     end;
   end;
  lqrySQL.Close;
  lqrySQL.Prepare;
  lqrySQL.Open;
  Result:= True;
 except
  on E: EDBEngineError do
   MessageDlg(E.message,mtWarning, [mbOK], 0);
  on E: EDatabaseError do
   MessageDlg(E.message,mtWarning, [mbOK], 0);
  end;
end;
```

```
procedure TSQLConsulta.pCierra;
begin
  lqrySQL.Close;
end;

procedure TSQLConsulta.pCargaItems (parrsColumnas: array of string; parrpntItems: array of pointer);
var
  liElemento: word;
  lpntstrItems:^TStrings;
begin
  {Limpia cada una de los items}
  for liElemento:= Low(parrpntItems) to High(parrpntItems) do
    begin
      lpntstrItems:= parrpntItems[liElemento];
      lpntstrItems^.Clear;
    end;
  lqrySQL.First;
  while (lqrySQL.Eof = false) do
    begin
      {Pasa por cada apuntador de items y le coloca del registro su columna correspondiente}
      for liElemento:= Low(parrpntItems) to High(parrpntItems) do
        begin
          lpntstrItems:=parrpntItems[liElemento];
          lpntstrItems^.Add(lqrySQL.FieldByName(parrsColumnas[liElemento]).AsString);
      end;
      lqrySQL.Next;
    end;
end;

{*********************************************************************}
{Objeto TSQLEntidad}
constructor TSQLEntidad.Create (psBD: string; pmsgError: TMsgDML);
begin
  inherited Create;
  mmsgError:= pmsgError;
  mqryDML:= TQuery.Create (mqryDML);
  mqryDML.DatabaseName:= psBD;
  lprmLlave:= TParams.Create;
end;

destructor TSQLEntidad.Destroy;
begin
  inherited;
  lprmLlave.Destroy;
  mqryDML.Destroy;
end;

function TSQLEntidad.mfbEjecutaSQL (pstrListaInstruccion: TStrings): boolean;
var
  liParametro: integer;
begin
  Result:= False;
  try
    if pstrListaInstruccion <> Nil then
      begin
        pstrListaInstruccion.Assign (mqryDML.SQL);
        for liParametro:= 0 to mqryDML.ParamCount - 1 do
          begin
            pstrListaInstruccion.Add (mqryDML.Params.Items[liParametro].Name
```

```
                    + ': ' + mqryDML.Params.Items[liParametro].AsString);
          end;
      end;
    mqryDML.Prepare;
    mqryDML.ExecSQL;
    Result:= True;
  except
    on E: EDBEngineError do
      mpGeneraError(E);
  end;
end;

function TSQLEntidad.mfsErrores (perrError: EDBEngineError): string;
begin
  Result:= '';
end;

procedure TSQLEntidad.mpGeneraError (perrError: EDBEngineError);
var
  lsErrorApp, lsError: string;
begin
  //Personaliza el mensaje
  lsErrorApp:= mfsErrores (perrError);
  if lsErrorApp = '' then
    lsError:= perrError.Message
  else
    lsError:= lsErrorApp + ' (' + perrError.Message + ') ';

  //Decide que tipo de error generar
  case mmsgError of
    tMsgDMLMensaje:
      MessageDlg (lsError, mtWarning, [mbOK], 0);
    tMsgDMLExceptionCustom:
      Raise EDBEngineError.CreateFmt (lsError,[]);
    tMsgDMLExceptionNative:
      Raise EDBEngineError.Create(perrError.Errors[0].ErrorCode);
  end;
end;

function TSQLEntidad.fbModificacion: boolean;
//Implementa la función de modificar un registro, alta si no existe
//y actualización si existe
var
  lmsgError: TMsgDML;
begin
  //Respalda la excepción enviada en el contructor y asigna la exepción nativa
  lmsgError:= mmsgError;
  mmsgError:= tMsgDMLExceptionNative;
  Result:= False;
  try
    Result:= fbAlta;
    mmsgError:= lmsgError;
  except
    on E: EDBEngineError do
      begin
        //Regresa la excepción solicitada en el constructor
        mmsgError:= lmsgError;
        //Si al dar el alta ya existe el registro sólo lo acutaliza
        if E.Errors[0].ErrorCode = DBIERR_KEYVIOL then
          Result:= fbActualizacion
```

```
      else
        //Si existe un error diferente al alta genera el error
        mpGeneraError(E);
    end;
  end;
end;
end.
```

# SQLEntSeg.pas

```
unit SQLEntSeg;

interface

uses
  SQLMngBD, DBTables, SQLEntidad;

type

  TSQLSegUsuario = class (TSQLEnt)
  protected
  protected
    function    mfsErrores (pError: EDBEngineError): string; override;
  public
    constructor Create (psBD: string; pmsgError: TMsgDML); override;
    function    fbActualizacion: boolean; override;
    function    fbAlta: boolean; override;
    function    fbBaja: boolean; override;
  end;


implementation
uses
  SQLDML, BDE, SysUtils, SQLQuerys, Dialogs, GrlServer;

{*********************************************************************}
{Objeto TSQLSegUsuario}
constructor TSQLSegUsuario.Create (psBD: string; pmsgError: TMsgDML);
begin
  inherited;
  lrecEntidad.Entidad:= teSegUsuario;
end;

function TSQLSegUsuario.mfsErrores (pError: EDBEngineError): string;
begin
  Case pError.Errors[0].ErrorCode of
    DBIERR_KEYVIOL: Result:= 'Usuario duplicado';
//    DBIERR_FORIEGNKEYERR: Result:= '';
//    DBIERR_DETAILRECORDSEXIST: Result:= 'X';
  else Result:= pError.Message;
  end;
end;

function TSQLSegUsuario.fbActualizacion: boolean;
begin
  {Si no tiene perfil sólo actualiza el password (invoca un cambio de password)}
  if lrecEntidad.UsusPerfil = '' then
    begin
```

```
    mqryDML.SQL.Assign (frmSQLDML.txmTUsuarioPwdUpd.Lines);
    mqryDML.ParamByName ('cClave').AsString:= lrecEntidad.UsuxsClave;
    mqryDML.ParamByName ('cPassword').AsString:= lrecEntidad.UsusPassword;
   end
  else
   begin
    {Si se envía el password  y perfil actualiza al usuario}
    if lrecEntidad.UsusPassword <> '' then
     begin
      mqryDML.SQL.Assign (frmSQLDML.txmTUsuarioUpd.Lines);
      mqryDML.ParamByName ('cClave').AsString:= lrecEntidad.UsuxsClave;
      mqryDML.ParamByName ('cPassword').AsString:= lrecEntidad.UsusPassword;
      mqryDML.ParamByName ('cPerfil').AsString:= lrecEntidad.UsusPerfil;
      mqryDML.ParamByName ('cNombre').AsString:= lrecEntidad.UsusNombre;
     end
    else
    {Si no se envía el password  y si el perfil es que no cambio el password}
     begin
      mqryDML.SQL.Assign (frmSQLDML.txmTUsuarioUpdSinPwd.Lines);
      mqryDML.ParamByName ('cClave').AsString:= lrecEntidad.UsuxsClave;
      mqryDML.ParamByName ('cPerfil').AsString:= lrecEntidad.UsusPerfil;
      mqryDML.ParamByName ('cNombre').AsString:= lrecEntidad.UsusNombre;
     end;
   end;
  Result:= mfbEjecutaSQL (frmGrlServer.Memo2.Lines);
end;

function TSQLSegUsuario.fbAlta: boolean;
begin
  mqryDML.SQL.Assign (frmSQLDML.txmTUsuarioIns.Lines);
  mqryDML.ParamByName ('cClave').AsString:= lrecEntidad.UsuxsClave;
  mqryDML.ParamByName ('cPassword').AsString:= lrecEntidad.UsusPassword;
  mqryDML.ParamByName ('cPerfil').AsString:= lrecEntidad.UsusPerfil;
  mqryDML.ParamByName ('cNombre').AsString:= lrecEntidad.UsusNombre;
  lprmLlave.Clear;
  Result:= mfbEjecutaSQL (frmGrlServer.Memo2.Lines);
  if Result then
    lprmLlave.AddParam (mqryDML.ParamByName ('cClave'));
end;

function TSQLSegUsuario.fbBaja: boolean;
begin
  mqryDML.SQL.Assign (frmSQLDML.txmTUsuarioDel.Lines);
  mqryDML.ParamByName ('cClave').AsString:= lrecEntidad.UsuxsClave;
  Result:=  mfbEjecutaSQL (frmGrlServer.Memo2.Lines);
end;

end.
```

# CONCLUSIONES

El desarrollo de aplicaciones de software logra sufrir cambios muy drásticos dependiendo del esquema de trabajo utilizado para lograr el objetivo. Aunque a veces el trabajo planificado y organizado parece ser una serie de pasos burocráticos innecesarios para llegar a los objetivos, al final del camino, siempre se demostrará que el caminar lento pero firme hacia un objetivo claro, es mejor que un paso rápido e inseguro a un objetivo incierto.

El ciclo de desarrollo de software llega a ser mucho más sencillo adoptando esquemas ya comprobados en donde la documentación se convierte en el resultado del pensamiento para resolver el problema de manera gradual, en vez de ser una fase al final de los proyectos en donde la documentación se convierte en una etapa fastidiosa y de poca utilidad por sólo intentar cubrir la entrega del sistema y siendo intracendente al momento de realizar mantenimientos a las aplicaciones.

# BIBLIOGRAFÍA


Booch, Grady, Second Edition 1994. Object – Oriented Analysis and Design with Applications. The Benjamin / Cummings Publishing.

Booch, Grady; Rumbaugh, James y Jacobson, Ivar, 1999. El Lenguaje Unificado de Modelado. Addison – Wesley Iberoamericana.

Booch, Grady; Rumbaugh, James y Jacobson, Ivar, 1999. El Proceso Unificado de Desarrollo de Software. Addison – Wesley Iberoamericana.

Date, C. J., Fifth Edition, 1990. An Introduction To Database Systems, Volume I. Addison – Wesley Publishing.

Gardarin, Georges, 1990. Bases de Datos: Gestión de Ficheros, El Modelo Relacional, Algoritmos y Lenguajes, Seguridad de los Datos. Paraninfo.

Jacobson, Ivar, 1991. Object – Oriented Software Engineering a Use Case Driven Approach. Addison – Wesley Publishing.

Hawryszkiewycz, Igor Titus, 1994. Análisis y Diseño de Base de Datos. Limusa.

Lucas Gómez, Angel, 1993. Diseño y Gestión de Sistemas de Bases de Datos. Paraninfo.

Schmuller, Joseph. Aprendiendo UML en 24 Horas. Prentice – Hall.