30

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

APLICACION DE UML EN EL DISEÑO DE BASES DE DATOS RELACIONALES

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

A C T U A R I A

P R E S E N T A

ROSALBA ESPINOSA QUINTAS

DIRECTORA DE TESIS:
DRA. AMPARO LÓPEZ GAONA

FACULTAD DE CIENCIAS
UNAM

UNIVERSIDAD NACIONAL
AVENMA DE
MEXICO

M. EN C. ELENA DE OTEYZA DE OTEYZA
Jefa de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunico a usted que hemos revisado el trabajo escrito:

Aplicación de UML en el diseño de bases de datos relacionales

realizado por Rosalba Espinosa Quintas

con número de cuenta 8510411-6 **, quién cubrió los créditos de la carrera de** Actuaría

Dicho trabajo cuenta con nuestro voto aprobatorio.

Atentamente

Director de Tesis Propietario	Dra. Amparo López Gaona	
Propietario	M. en C. Guadalupe Elena Ibargüengoitia González	
Propietario	Mat. Salvador López Mendoza	
Suplente	Mat. Víctor Hugo Dorantes González	
Suplente	Dra. Hanna Oktaba	

Consejo Departamental de Matemáticas

FACULTAD DE CIENCIAS
CONSEJO DEPARTAMENTAL
MATEMÁTICAS

M. en C. José Antonio Flores Díaz

A MICAELA Y TOMÁS
...MIS PADRES, POR SU INFINITO AMOR Y CONTINUO APOYO.

A CARLOS, KARLA Y PAOLA
...MI FAMILIA, POR EL TIEMPO QUE NO LES PUDE DEDICAR.

A AMPARO
...MI DIRECTORA, POR SU TIEMPO, CONFIANZA Y PACIENCIA.

A LUPITA, HANNA, SALVADOR Y VICTOR HUGO
...MIS SINODALES, POR ENRIQUECER ESTE TRABAJO CON SUS OBSERVACIONES.

INTRODUCCIÓN

Debido a la facilidad de uso de las herramientas gráficas para desarrollo de sistemas, muchos desarrolladores después de hacer un análisis de requerimientos, frecuentemente demasiado somero, se van directamente a la construcción de la base de datos, pantallas y reportes, sin detenerse a realizar un verdadero diseño de la base de datos, lo que generalmente provoca una solución incompleta, ineficaz y costosa que no enfrenta todos los requerimientos y posiblemente necesite modificaciones adicionales que implican más esfuerzo e inversión de tiempo y dinero. El desarrollador de sistemas no sólo enfrenta, como parece a simple vista, el resolver una problemática con software, también debe garantizar que la solución propuesta sea efectiva y a un costo aceptable. Los sistemas se construyen para satisfacer ciertas necesidades, las cuales son variables en el tiempo.

Por lo anterior, durante el desarrollo de un sistema un aspecto que no debe dejarse a la ligera es el diseño adecuado y completo de la base de datos que se requiere, pues de esta manera se reduce el riesgo de falta de datos y mal desempeño. Uno de los motivos más argumentados para eludir el trabajo que implica un buen diseño de la base, es el tiempo que consume esta actividad ya que el desarrollador siempre está siendo presionado por el cliente para que presente avances visibles del sistema que se está construyendo

En este trabajo se presenta la utilización de algunos aspectos del lenguaje de modelado: UML (Unified Modeling Language), en el diseño de bases de datos relacionales. Aunque en el esfuerzo de diseño no existen recetas que garanticen el éxito, el objetivo de este trabajo es presentar una metodología, apoyada en conceptos y notación de UML, que permita reducir el tiempo a invertir en la tarea del diseño de la base de datos y genere modelos más completos y correctos que, por la notación utilizada, facilitan la comunicación entre desarrollador y cliente, permitiendo verificaciones conforme se va avanzando. Además, por la naturaleza orientada a objetos de los conceptos de UML, las bases de datos obtenidas pueden resultar más flexibles, reutilizables y de fácil mantenimiento. También se pretende fomentar el uso de UML en el desarrollo de sistemas, pues no obstante que en este trabajo sólo se muestran los aspectos relativos al diseño de la base de datos, el UML tiene elementos para todas las fases del desarrollo de un sistema desde el análisis de requerimientos hasta la implementación.

Para lograr el objetivo planteado este trabajo se divide de la siguiente forma:

En el primer capítulo se presentan los conceptos generales de las bases de datos y en particular del modelo relacional, el cual sigue siendo el más utilizado en el desarrollo de aplicaciones ya que sus fundamentos matemáticos son sencillos pero sólidos.

El segundo capítulo corresponde a la presentación de la mayor parte de la metodología, incluyendo el punto de partida de todo desarrollo de sistemas: el análisis de requerimientos y extendiéndose en los aspectos referentes al modelado de datos, utilizando en ello la notación y conceptos de UML correspondientes.

El tercer capítulo comprende lo referente a la conversión del modelo de datos UML al modelo relacional (la parte final de la metodología), para contar con lo indispensable para la construcción de la base de datos.

Finalmente se ejemplifica la aplicación de la metodología en un problema real de una compañía aseguradora dedicada al ramo de vida que requiere de una base de datos adecuada y completa para soportar un sistema automatizado que le permita administrar su cartera de vida individual, que incluya los datos suficientes para realizar desde la cotización de sus productos hasta la valuación actuarial de las pólizas, contando por supuesto con lo necesario para la emisión, cobranza, endosos, reaseguro y registro de siniestros. En esta parte se presenta la secuencia completa de la metodología, que incluye: recolección de requerimientos, análisis con casos de uso, modelado de datos con diagramas de clase y la conversión del modelo UML al modelo relacional.

CONTENIDO

1. BASES DE DATOS

1.1 EVOLUCIÓN

El advenimiento de los sistemas computacionales como una herramienta práctica de administración contribuyó a que los diseñadores se esforzaran por desarrollar un almacenamiento de datos realista y eficiente así como procedimientos de recuperación de los mismos. Inicialmente los datos eran almacenados simplemente como un registro físico en orden secuencial y dada su naturaleza repetitiva estaban agrupados en el mismo dispositivo constituyendo un archivo. Al llegar el almacenamiento en unidades de disco se aceleró la velocidad de acceso a los datos almacenados; fue en el período de 1950 a 1960 cuando se desarrollaron las principales ideas relativas a la organización de archivos y métodos de acceso (secuencial, directo y secuencial indexado). Aún con numerosos refinamientos por el paso del tiempo, estos métodos permanecen válidos y son los más ampliamente utilizados en los sistemas computarizados.

Sin embargo, estos sistemas de archivos se desarrollaron alrededor de aplicaciones específicas. En muchos casos, estas aplicaciones poseían elementos comunes y asociaciones que no podían ser explotadas por el hecho de que los archivos estaban concebidos para ser utilizados independientemente unos de otros.

A la par de estas restricciones sobre manipulación de archivos, la tecnología produjo dispositivos de almacenamiento que hicieron posible el guardar grandes masas de información a costo menor. El creciente conocimiento sobre las necesidades internas de las organizaciones, proveyó el ímpetu para adquirir más y más información refinada inherente a sus actividades y, finalmente, generó el desarrollo del concepto de "Base de Datos" alrededor de 1962.

En esa década se desarrollaron algunos sistemas tales como MEDLARS, para documentación médica, SABRE para reservaciones de líneas aéreas y otros sistemas de planeación gubernamental.

Las bases de datos han cambiado significativamente su naturaleza en los últimos años. En el pasado, las bases de datos frecuentemente soportaban funciones organizacionales simples y eran utilizadas para proporcionar reportes regulares especificados. Hoy en día las bases de datos juegan un papel central en muchos sistemas de información y soportan diversas funciones más complejas. En consecuencia tanto el software como el diseño de las bases de datos han evolucionado con el tiempo para enfrentar las necesidades de cambio.

1.2 CONCEPTOS GENERALES

Debido a la importancia que ha tomado la información en las organizaciones actuales, la base de datos es un recurso sumamente valioso. Esto condujo al desarrollo de conceptos y técnicas para manejar los datos en forma eficiente.

Un dato se concebirá como un conjunto de caracteres que no tiene un significado propio, sólo es un valor que toma parte en la descripción de un objeto o evento; por su parte la

información es considerada como un dato o conjunto de datos que significan algo para alguien; generalmente estos datos tienen un significado suficiente para provocar acciones siendo utilizados dentro de un cierto contexto.

Debido a que un mismo dato puede tener diferentes significados para diversos usuarios, lo que se almacena es sólo el dato en bruto y es por ello que se da la acepción de Base de Datos y no la de Base de Información.

Base de Datos: Es una colección de datos estructurados, mutuamente relacionados, almacenados en un medio computacional accesible, con la mínima redundancia e inconsistencia con el objetivo de satisfacer las necesidades de varios usuarios simultáneamente.

Por mutuamente relacionados se entiende que los datos representan conocimiento acerca de un ente específico, como una empresa, escuela o institución gubernamental. Los datos pueden estar relacionados también porque corresponden a cierta área de un problema, por ejemplo la nómina de una compañía.

Cuando se utiliza un sistema de archivos, cada aplicación tiene sus propios archivos privados, lo que a menudo origina una redundancia enorme (es decir, tener almacenados los mismos datos en dos o más lugares), así como un desperdicio del espacio de almacenamiento. Esos archivos separados pueden integrarse para eliminar o disminuir la redundancia, pues no se pretende que toda la redundancia sea eliminada debido a que, ocasionalmente, hay fuertes razones comerciales o técnicas para mantener múltiples copias de los mismos datos. Sin embargo en una base de datos debe tenerse un control correcto sobre la redundancia.

Entre más grande sea la masa de información registrada en una base de datos es mayor el riesgo de error. La Integridad de los Datos tiene que ver con el grado de exactitud, oportunidad y relevancia de los datos dentro de la Base. Esto puede ser logrado realizando ciertas verificaciones al adicionar una nueva ocurrencia de un registro o cuando se modifica algún dato.

Los datos deben estar organizados de manera que sea posible procesarlos para obtener información. Esta organización debe representar el significado de fondo o semántica en forma correcta y eficiente, pues el contenido, total o parcial, de la base de datos será utilizado por muchos y diversos programas.

1.3 SISTEMAS ADMINISTRADORES DE BASES DE DATOS

El software que permite al usuario comunicarse con una base de datos es provisto por el **SISTEMA ADMINISTRADOR DE LA BASE DE DATOS –SABD–** (Data Base Managment System). En esencia, este incluye las herramientas para organizar los datos en los dispositivos periféricos de almacenamiento, los procedimientos para recuperarlos y poder llevar a cabo procesamientos posteriores de estos datos.

Por lo general, las bases de datos requieren de una gran cantidad de espacio para almacenarse. Comúnmente una base de datos de una organización se mide en términos de Gigabytes. Ya que la memoria principal de la computadora no puede almacenar tal

cantidad de datos, éstos se almacenan en discos u otro tipo de dispositivos secundarios, pero como el movimiento de datos al y del disco es lento, comparado con la velocidad de la unidad central de procesamiento de las computadoras, es necesario que los datos se organicen de tal forma que se reduzca la necesidad de transferirlos entre el disco y la memoria principal.

Uno de los objetivos principales de un SABD es crear un ambiente en el que pueda almacenarse y recuperarse información en la Base de Datos en forma conveniente y eficiente. Estos sistemas permiten evitar o controlar las principales desventajas de los sistemas de archivos convencionales entre las cuales se pueden enunciar:

- Redundancia e inconsistencia de los datos.
- Dificultad para acceder a los datos.
- Aislamiento de los datos.
- Problemas de concurrencia.
- Problemas de seguridad.
- Problemas de integridad.

Los actuales SABD's vienen en diversas clasificaciones y con diferentes capacidades de procesamiento, pero en general tratan de lograr tres objetivos:

1. Consolidación de Datos. Se refiere a la combinación o unificación de archivos de datos dispersos, dentro de una estructura centralizada, y al almacenamiento de datos en un formato no redundante. Un formato redundante es una estructura que almacena los mismos datos en dos o más localidades.

2. Compartimiento de Datos. Habilidad del sistema para permitir el acceso concurrente de múltiples usuarios a partes individuales de la Base de Datos. Frecuentemente dos usuarios desearán acceder un mismo registro en la Base al mismo tiempo, por ello es obligatorio el que cada acceso sea procesado en un orden completamente secuencial para asegurar que las entradas sean actualizadas correctamente.

3. Protección de Datos. Se refiere a la habilidad de un SABD para mantener la integridad de sus datos al ocurrir cierto tipo de hechos adversos al procesamiento (por ejemplo: caídas del Sistema, fallas de programas, etc.).

Las bases de datos se modifican con el tiempo, al insertar o eliminar información de ellas. El conjunto de información almacenado en la base de datos en cierto momento se denomina un **ejemplar** de la base de datos. El diseño general de la base de datos es llamado **esquema** de la base de datos. Los esquemas se alteran muy raras veces o nunca.

Haciendo una analogía con los conceptos de lenguajes de programación, la idea de un esquema de base de datos corresponde al concepto de definición de tipo en lenguajes de programación. Una variable de un determinado tipo tiene un valor específico en un momento dado lo que corresponde al concepto de un ejemplar en una base de datos.

Existen varios esquemas en la base de datos, y éstos se dividen de acuerdo con los Niveles de Abstracción que se definirán a continuación.

NIVELES DE ABSTRACCIÓN

Uno de los objetivos principales de un Sistema de Base de Datos es proporcionar a los usuarios una visión abstracta de la información, es decir, el sistema debe ocultar ciertos detalles relativos a la forma en como se almacenan y mantienen los datos. Sin embargo, la recuperación de la información debe hacerse en forma eficiente para que el Sistema sea considerado útil.

Para lograr tal eficiencia se recurre al diseño de estructuras de datos complejas para representar la información en la Base de Datos. Pero como, generalmente, los Sistemas de Bases de Datos son utilizados por personal que no cuenta con conocimientos lo suficientemente amplios en computación, la complejidad debe estar escondida para los usuarios, para ello se definen varios niveles de abstracción en los que puede observarse la Base de Datos, entre ellos se encuentran:

- **Nivel Físico.** En este nivel se describe cómo se almacenan realmente los datos en los dispositivos físicos. Se detallan las estructuras de datos complejas del nivel más bajo. Se pueden tener varios grados de detalle: a nivel bit, o a nivel registro o archivo tal como se interpretan en lenguajes de programación. En este nivel se tiene el Esquema Físico.

- **Nivel Conceptual.** Aquí se describen cuáles son los datos reales que están almacenados en la Base y las relaciones existentes entre ellos. Este nivel contiene toda la base de datos en términos de unas cuantas estructuras relativamente sencillas. Aunque es posible que la implantación de las estructuras simples del nivel conceptual requiera de estructuras complejas en el nivel físico, no es obligatorio que el usuario del nivel conceptual se percate de ello. En este nivel se define el Esquema Conceptual.

- **Nivel Externo**. Es el nivel de abstracción más alto, en el cual se describe solamente una parte de la Base de Datos. Aunque en el nivel conceptual se utilizan estructuras simples, todavía queda una forma de complejidad que resulta del gran tamaño de la Base. Muchos usuarios de ésta no tendrán que ocuparse de toda la información; más bien sólo necesitarán una parte de la Base. Para simplificar la interacción entre estos usuarios y el Sistema, se define el nivel de abstracción Externo.

Este nivel corresponde a la vista de todo o una parte del esquema conceptual como podría ser observada por un grupo de usuarios a los que les interesa una aplicación particular y es necesario para describir, con ayuda del Esquema Externo, frecuentemente referido como una vista, cómo se percibirán los datos por un programa de aplicación. El Esquema Externo o Vista puede tomarse como un SubEsquema del Esquema Conceptual aunque, en algunos casos, se puede considerar que el Esquema Externo provee más información que el Esquema Conceptual. El sistema puede proporcionar muchas vistas diferentes de la misma Base de Datos.

Una vista es sólo la imagen o descripción de información y no los datos físicos. Las vistas tienen una capacidad limitada para la actualización; se utilizan principalmente para simplificar el proceso de consulta y como parte del sistema de seguridad para restringir el acceso a subconjuntos particulares de datos.

INDEPENDENCIA DE LOS DATOS
Se han definido los tres niveles de abstracción en los que se puede ver una base de datos. La capacidad de modificar una definición de esquema en un nivel sin afectar la definición del esquema en el nivel inmediato superior se denomina Independencia de los Datos. Existen dos niveles de tal independencia:

- **Independencia Física.** Referida a la capacidad de modificar el esquema físico sin obligar a que se vuelvan a escribir programas de aplicación. En algunas ocasiones son necesarias las modificaciones a nivel físico para mejorar el rendimiento.

- **Independencia Lógica.** Es la capacidad de modificar el esquema conceptual sin obligar a que se vuelvan a escribir los programas de aplicación. Las modificaciones en el nivel conceptual son requeridas siempre que se altera la estructura lógica de la base de datos.

La independencia lógica de los datos es más difícil de lograr que la independencia física, ya que los programas de aplicaciones dependen estrechamente de la estructura lógica de los datos a los que tienen acceso. La ventaja de la independencia lógica es que permite que los programas de aplicación continúen funcionando a pesar de las modificaciones realizadas al sistema.

Por lo anterior, el concepto de independencia de los datos es similar en muchos aspectos al concepto de Tipo Abstracto de Datos en los lenguajes de programación modernos, debido a que ambos ocultan los detalles de la implantación, lo que permite a los usuarios concentrarse en la estructura general más que en los detalles del nivel inferior.

LENGUAJES DE DEFINICIÓN Y MANEJO DE DATOS.
El SABD debe proveer de una herramienta que permita al usuario describir el conjunto de datos a ser almacenados en la base. Se ha visto que hay varios niveles de descripción dependiendo de si nos interesa una vista de usuario o de cómo estos datos están organizados para ser almacenados en los dispositivos físicos; el primer caso es llamado Descripción Lógica, mientras que el segundo corresponde a la Descripción Física.

Un esquema de Base de Datos se especifica por medio de una serie de definiciones que se expresan en un lenguaje específico llamado **Lenguaje de Definición de Datos** (Data Definition Language -DDL-). El resultado de la compilación de las proposiciones en DDL es un conjunto de tablas que se almacena en un archivo especial llamado diccionario de datos.
Un diccionario de datos es un archivo que contiene metadatos, esto es, datos acerca de los datos. Este archivo se consulta antes de leer o modificar los datos reales en el sistema de base de datos.

La estructura de almacenamiento y los métodos de acceso empleados por el sistema de base de datos se especifican por medio de un conjunto de definiciones de un tipo especial de DDL llamado Lenguaje de Almacenamiento y Definición de los Datos. El resultado de la compilación de estas definiciones es una serie de instrucciones que especifican los detalles de implantación de los esquemas de la base de datos que normalmente no pueden ver los usuarios.

Los niveles de abstracción que se trataron anteriormente no sólo se aplican a la definición o estructuración de los datos, sino también a su manejo, ésta manipulación consiste en:

- La recuperación de información almacenada en la base de datos.
- La inserción de nueva información en la base de datos.
- La eliminación de información de la base de datos.

En el nivel físico, deben definirse algoritmos que permitan tener acceso a los datos en forma eficiente. En los niveles más altos de abstracción lo importante es la facilidad de uso. La meta es lograr una interacción eficiente entre el sistema y sus usuarios.

Un **Lenguaje de Manejo de Datos** (Data Management Language -DML-) permite a los usuarios manipular o acceder los datos que están organizados por medio del modelo apropiado. Básicamente, existen dos tipos de DML:

- **De procedimiento**, los cuales requieren que el usuario especifique cuáles datos desea y cómo deben ser obtenidos.
- **Sin procedimientos**, que sólo necesitan que el usuario especifique cuáles datos quiere sin especificar como obtenerlos.

Los DML sin procedimientos son, generalmente, más fáciles de aprender y utilizar que los de procedimientos. Pero, debido a que el usuario no tiene que especificar la forma de obtención de datos, estos lenguajes pueden generar un código menos eficiente que el producido por los lenguajes de procedimientos. Este problema puede ser resuelto empleando diversas técnicas de optimización.

Una **consulta** es una proposición que solicita la recuperación de información. La parte de un DML que realiza esta función es llamado **Lenguaje de Consultas**. No obstante que técnicamente es incorrecto, se suelen ocupar indistintamente los términos lenguaje de consultas y lenguaje de manejo de datos.

ESTRUCTURA GENERAL DE UN SABD

Un Sistema de Base de Datos se divide en módulos que realizan cada una de las funciones del sistema general. Algunas de las tareas del sistema de Base de Datos pueden ser llevadas a cabo por el Sistema Operativo. Sin embargo, el sistema operativo, por lo regular, proporciona únicamente los servicios más fundamentales y para el diseño de la Base de Datos se debe tomar en cuenta esta limitación e incluir una consideración de la interfaz entre el sistema de Base de Datos y el Sistema Operativo.

Algunos de los componentes funcionales de un Sistema de Base de Datos son los siguientes:

- **El manejador de archivos**, que es el encargado de asignar espacio en el disco y de la utilización de las estructuras de datos que se emplearán para representar la información almacenada en los dispositivos auxiliares.

- **El manejador de Base de Datos,** que constituye la interfaz entre los datos de bajo nivel guardados en la Base de datos, los programas de aplicación y las consultas que se hacen al sistema. El manejador de base de datos es responsable de las siguientes actividades:

- Interacción con el manejador de archivos. Los datos sin procesar se almacenan en los dispositivos secundarios mediante el sistema de archivos proporcionado normalmente por un sistema operativo convencional. El manejador de base de datos traduce las diferentes proposiciones en DML a comandos de sistema de archivos de bajo nivel, esto es, se encarga realmente del almacenamiento, recuperación y actualización de los datos.
- Implantación de la integridad. Los valores de los datos almacenados en la base deben satisfacer ciertos tipos de limitantes de consistencia. Si se especifican estas limitantes, el manejador debe poder verificar si las actualizaciones a la base de datos violan cualquiera de esas limitantes y en este caso, realizar la acción apropiada.
- Mantenimiento de la Seguridad. Ya que no es necesario ni conveniente el que cualquier usuario tenga acceso a toda la base de datos, el manejador debe hacer que se cumplan los lineamentos de seguridad.
- Respaldo y recuperación. Debido a que un sistema de cómputo está sujeto a fallas como errores de software, interrupción del suministro de energía, etc., y en cualquiera de esos casos cabe la posibilidad de perder información, el manejador tiene como responsabilidad el detectar esas fallas y la restauración de la base de datos al estado que tenía antes de presentarse la falla.
- Control de concurrencia. Cuando varios usuarios actualizan la base de datos en forma concurrente, es posible que no se conserve la consistencia de los datos, por ello es necesario que el manejador controle la interacción entre los usuarios concurrentes.

- **El Procesador de Consultas,** el cual se encarga de traducir las proposiciones en lenguaje de consulta a instrucciones de bajo nivel que pueda comprender el manejador de la Base de Datos. Además, debe tratar de convertir la solicitud del usuario a una forma equivalente pero más eficiente.

- **El Precompilador de DML,** que convierte las proposiciones en DML en llamadas normales a procedimientos en el lenguaje huésped. Debe interactuar con el procesador de consultas para generar el código apropiado.

- **El Compilador de DDL,** que convierte las proposiciones en DDL, en un conjunto de tablas que contienen metadatos.

Además requiere de varias estructuras de datos como parte de la implantación de la parte física, incluyendo:

- **Archivos de datos,** que contendrán la Base de Datos.

- **Diccionario de datos,** que almacenará la información relativa a la estructura de la base de datos. Su uso es constante por lo que se debe desarrollar un diseño apropiado y una implantación eficiente.

- **Indices,** para permitir el acceso apropiado a elementos dato que contienen valores determinados.

ADMINISTRADOR DE LA BASE DE DATOS

Uno de los motivos principales para contar con sistemas de manejo de base de datos es tener un control centralizado tanto de los datos como de los programas que tienen acceso a ellos. La persona que tiene este control se denomina Administrador de la Base de Datos (DataBase Administrator -DBA-).

Entre otras, las siguientes son funciones que debe realizar el DBA:

- Definición de Esquema.
- Definición de la estructura de almacenamiento y del método de acceso.
- Modificación del esquema y de la organización física.
- Concesión de autorización para el acceso a los datos.
- Especificación de las limitantes de integridad. Estas limitantes se conservan en una estructura especial del sistema que el manejador de base de datos consulta cada vez que se lleva a cabo una actualización en el sistema.

1.4 CICLO DE VIDA DE LAS BASES DE DATOS

El ciclo de vida de una base de datos es, en realidad, muchos pequeños ciclos. Particularmente cuando se utilizan técnicas orientadas a objetos, el diseño de la base de datos es un proceso iterativo e incremental. Se puede ir del análisis al diseño y de éste a la construcción y regresar en cualquier orden que tenga sentido. Sin embargo es posible distinguir las siguientes fases en el ciclo de vida de una base de datos:

- Análisis de requerimientos
- Diseño, construcción y optimización
- Revisión y prueba
- Certificación
- Mantenimiento y mejoramiento

ANÁLISIS DE REQUERIMIENTOS

Las bases de datos tienen un ciclo de vida que inicia con las necesidades del usuario. Cuando se empieza a formular una base de datos, es indispensable poner toda la atención en las necesidades de los diferentes usuarios de la base de datos: usuarios finales, usuarios de sistema actuales y futuros. Los usuarios finales necesitan cosas que reflejen su mundo, los usuarios de sistema necesitan estructuras que les permitan hacer su trabajo efectiva y productivamente, los usuarios futuros necesitan documentación completa, clara y estándar que les proporcione información acerca de cómo se conforma y trabaja la base de datos para poder utilizarla.

Los requerimientos de información se mezclan en forma casi indistinguible dentro de los requerimientos de un sistema más grande del cual es parte la base de datos. En un sistema centrado en la base de datos, el análisis de requerimientos es crítico, por lo que se debe utilizar una gran cantidad de tiempo en entender los requerimientos.

En esta fase de la vida de la base de datos, se debe preguntar el QUÉ se espera de la base de datos.

DISEÑO, CONSTRUCCIÓN Y OPTIMIZACIÓN

Una vez que se tienen suficientemente claras las necesidades de los usuarios, es necesario modelar formalmente el problema. El modelado de datos es el primer paso en el diseño de la base de datos y tiene varios propósitos:

- Ayuda a organizar las ideas acerca de los datos, clarificando su significado y su aplicación práctica
- Ayuda a comunicar las necesidades y cómo se intentará enfrentarlas
- Provee una plataforma desde la cual se puede proceder al diseño y construcción con cierta seguridad de éxito

Todo Sistema de Base de datos requiere de una fase de diseño preliminar durante la cual los datos y los programas de aplicación son especificados. Esta fase descriptiva puede ser llevada a cabo eficientemente sólo si el diseñador tiene un modelo de datos disponible que le ayude a interpretar la aplicación

Un **Modelo de Datos** es un grupo de herramientas formales utilizadas para entender e interpretar el mundo real. Un modelo de datos permite definir las relaciones que pueden existir entre los diferentes datos o elementos, así como proporcionar una forma de describir la semántica y las limitantes de los datos. Los modelos de datos se pueden clasificar en tres amplios grupos:

1. **Modelos Lógicos Basados en Objetos.**
   Se utilizan para describir los datos en los niveles conceptual y externo. Permiten una estructuración bastante flexible y hacen posible especificar claramente las limitantes de los datos. Algunos de los más conocidos son:
   - El modelo Entidad-Relación.
   - El modelo Binario.
   - El modelo Semántico de Datos.
   - El modelo Infológico.
   - El modelo Objeto

2. **Modelos Lógicos basados en registros.**
   Sirven para describir los datos en los niveles conceptual y externo, pero a diferencia de los modelos de datos basados en objetos, estos modelos se utilizan para especificar tanto la estructura lógica general de la base de datos como una descripción en un nivel más alto de implantación. Sin embargo, no permiten especificar en forma clara las limitantes de los datos. Los modelos que han tenido más aceptación son los siguientes:
   - Modelo Jerárquico.
   - Modelo de Red.
   - Modelo Relacional.
   - Modelo Objeto-Relacional

3. **Modelos Físicos de Datos.**
   Estos modelos sirven para describir los datos en el nivel más bajo. A diferencia de los modelos lógicos de datos son muy pocos los modelos físicos utilizados. Estos modelos capturan aspectos sobre la implantación de los SABD's. Algunos de los más conocidos son:
   - El modelo unificador.
   - La memoria de cuadros.

El modelado de datos es la abstracción inicial que esconde la complejidad del sistema. El modelo de datos reduce la complejidad a un nivel que el diseñador puede comprender y manipular. Dado que las bases de datos y las estructuras de los mismos crecen en número y complejidad, el modelado de datos toma cada vez más importancia.

La parte más difícil del diseño llega cuando se pasa del modelo de datos a un esquema. En algún punto el diseño pasa de lógico a físico siendo el inicio de la construcción de la base de datos. El diseño físico también es un proceso iterativo, no una secuencia rígida de pasos. Mientras se desarrolla el esquema físico, es posible darse cuenta que ciertos aspectos del diseño lógico pudieran afectar el diseño físico de manera negativa y por lo tanto requiere una revisión.

Dado que el diseño de la base de datos es un proceso iterativo e incremental no se puede dar el lujo de permitir fallas en el modelo. Si el modelo de datos está fuera de sincronía con el esquema, será cada vez más difícil el regresar a la parte inicial del diseño

Al estar pasando del diseño lógico al diseño físico, la atención cambia del modelado del mundo real al mejoramiento del desempeño de la base de datos. La mayoría de los aspectos del diseño físico tienen impacto directo en el rendimiento de la base de datos. En particular es importante tomar en cuenta cómo accesarán los datos los usuarios finales. El diseño físico es crítico para la afinación de la base de datos; sobretodo si se trata de sistemas de data warehouse que soportan decisiones de misión crítica, sistemas de proceso de transacciones en línea de respuesta instantánea o sistemas de combinaciones avanzadas de software/hardware como multiproceso simétrico (SMP) o procesamiento paralelo masivo (MPP).

El diseño no está completo si no considera los riesgos de la base de datos y los métodos de manejo de riesgos para reducirlos o evitarlos. Riesgo es el potencial de ocurrencia de algo que resultará en consecuencias negativas. En el área de bases de datos se incluyen cosas como: desastres, fallas de hardware, fallas y defectos de software, corrupción accidental de datos y ataques deliberados a los datos o al servidor. Para enfrentar los riesgos, es necesario determinar la tolerancia de riesgo a fin de que el riesgo se maneje dentro de ésta.

REVISIÓN Y PRUEBA DE LA BASE DE DATOS

La calidad de la base de datos viene de tres fuentes: requerimientos, diseño y construcción. La calidad de los requerimientos y el diseño utiliza técnicas de revisión, mientras que la construcción utiliza pruebas. La prueba a la base de datos se hace en tres formas: prueba de contenido, prueba de estructura y prueba de conducta.

El contenido es lo que frecuentemente se llama "calidad de datos". Se debe desarrollar un modelo que describa qué suposiciones y reglas hay para los datos. Parte de este modelo viene del modelo de datos. Los buenos planes de prueba de contenido cubren el rango completo del contenido, no sólo la vista del modelo de datos.

Tanto la prueba de estructura como la de conducta requieren una "cama de datos de prueba". La mayoría de los desarrolladores creen que datos "reales" es todo lo que se necesita. Desafortunadamente, los datos reales sólo cubren una pequeña parte de las posibilidades. Es necesario desarrollar colecciones de datos, sistemáticas y consistentes, que cubran todas las posibilidades que se requiere probar.

El desarrollo de pruebas procede en paralelo con el diseño de la base de datos y la construcción. Se debe utilizar el mismo proceso iterativo e incremental del diseño en las pruebas y probar las pruebas. Las pruebas generan un claro entendimiento de los riesgos de uso de la base de datos.

CERTIFICACIÓN

Es muy raro encontrar bases de datos certificadas, y por lo tanto es común encontrar usuarios queriendo reutilizar una base de datos o su diseño sin poder hacerlo, ya sea porque no es posible imaginarse como trabaja la base de datos o porque los vendedores de software no permiten el acceso por temer una corrupción.

Esto es un caso especial de un problema más general: la falta de reutilización del software. La realidad es que la reutilización es difícil y pocos proyectos la hacen bien. La clave para la reutilización viene en dos partes: diseño para reutilización y certificación de la reutilización.

Una de las ventajas establecidas de la tecnología orientada a objetos es el incremento de la productividad a través de la reutilización. La certificación de la reutilización tiene tres aspectos: riesgo, función y responsabilidad. Los esfuerzos de revisión y prueba proporcionan datos que sirven para valorar el riesgo de reutilizar la base de datos y su diseño. La parte funcional de la certificación consiste de documentación clara de los esquemas lógico y físico, así como el establecimiento preciso de las metas intentadas por la base de datos. Sin entender cómo funciona, nadie será capaz de reutilizar la base. Finalmente, una clara definición de quién tiene y es responsable del mantenimiento de la base, permite a otros reutilizarla casi sin problemas en el futuro.

MANTENIMIENTO Y MEJORAMIENTO

El mantenimiento y mejoramiento forman la etapa final del ciclo de vida de una base de datos. Frecuentemente es necesario iniciar el diseño con una base de datos ya establecida o con un diseño heredado de una versión anterior del sistema. La inercia de sistemas existentes vuelve locos a los diseñadores ya que es común que el mantenimiento lo realice personal que no participó directamente en el desarrollo del sistema original, además de que generalmente la construcción de un sistema tiene como objetivo final la primera versión del sistema y no las posibles nuevas funciones que adicionarán versiones futuras.

Es importante no olvidar que la naturaleza incremental del diseño de base de datos no termina con la liberación de la primera base de datos "viva", sino hasta que la base de datos deja de existir. Una base de datos va a través de muchos cambios.

1.5 EL MODELO RELACIONAL

POR QUÉ ESTE MODELO

Este es el modelo más aceptado por su sólida fundación matemática pues toma su origen de la noción de **relación**, la cual tiene una correspondencia directa con un concepto simple e intuitivo como el de tabla. Las bases de datos relacionales han sido y continúan siendo la solución más ampliamente adoptada en todo el mundo para el almacenamiento y recuperación de datos pues su simplicidad y rigor matemático son su mayor atracción.

Además, aún con los avances en la tecnología de bases de datos, será difícil remover el uso del modelo relacional ya que cubre y resuelve perfectamente un amplio espectro de los problemas que enfrentan los desarrolladores de sistemas de información.

ESTRUCTURA RELACIONAL DE LOS DATOS

A continuación se tratará con la terminología relacional formal, que incluye conceptos como: relación, tupla, atributo, llave primaria y dominio.
Intuitivamente, una **relación** corresponde a lo que generalmente es denominado **tabla**. Una **tupla** corresponde a un renglón de tal tabla y un **atributo** a una columna.

El término **llave** es uno de los más sobrecargados en el campo de las bases de datos. En el modelo relacional sólo encontramos llaves primarias, candidatas, alternativas y externas. Sin embargo, en otras áreas de la tecnología de bases de datos, se pueden encontrar llaves índice, de búsqueda, secundarias, de orden, padres, hijas y muchos otros tipos de llave. A pesar de esto, de entre la multiplicidad de llaves existentes, para quien se reserva el término llave es justamente la llave primaria, pues es la más importante de todas.

La **llave primaria** es un identificador único para la tabla, es decir, una columna o conjunto de columnas con la propiedad de que dos renglones o más de la tabla no puedan contener el mismo valor en esa columna o combinación de columnas.

Dominios

La unidad de dato más pequeña en el modelo relacional es un dato valor individual. Tales valores se consideran atómicos, es decir son indivisibles. Un **dominio** es un conjunto de tales valores, todos del mismo tipo, del cual uno o más atributos (columnas) toman sus valores actuales. Además, debe hacerse notar que no todos los valores incluidos en un dominio dado aparecerán en alguno de los atributos a los que les corresponda ese dominio.

Los dominios de valores atómicos son generalmente referidos como **dominios simples**, para distinguirlos de los **dominios compuestos**. De la misma forma los atributos pueden clasificarse en simples o compuestos (dependiendo de qué tipo sea el dominio en que estén definidos). Los dominios tienen un significado operacional: si dos atributos toman sus valores del mismo dominio, entonces las comparaciones que involucren estos dos atributos probablemente tengan sentido. Los dominios son principalmente conceptuales, es decir pueden o no pueden ser explícitamente almacenados en la base de datos como conjuntos de valores actuales, y en muchos casos no son almacenados. Sin embargo, podrían ser especificados como parte de la definición de la base de datos (en un sistema que soporte el concepto totalmente), y por tanto, cada definición de atributo podría incluir una referencia a su domino correspondiente, de tal forma que el sistema pueda percatarse de qué atributos son comparables entre sí y cuales no. Un dominio compuesto está definido como el producto cartesiano de alguna colección de dominios simples. Por supuesto, el sistema podría soportar atributos compuestos sin el soporte explícito de los correspondientes dominios.

Una observación importante es que la noción relacional de dominio está estrechamente relacionada al concepto de tipo de dato en los lenguajes de programación. Los dominios podrían ser imaginados como tipos de dato semántico y no considerados de naturaleza básicamente sintáctica como en el caso de la noción primitiva de dominio en SQL dado

que este provee de ciertos tipos de dato primitivos (Smallint, Float, Char, etc.), pues no conllevan una interpretación semántica.

El concepto de dominio es considerablemente más complejo de lo que a simple vista parece (tal vez esta sea la razón por la cual muchos de los sistemas actuales no lo soportan).

Un soporte completo del concepto de dominio podría, por tanto, requerir de lo siguiente:

- La habilidad de especificar completamente el conjunto de dominios **D**, para una base de datos dada. **D** podría ser un conjunto cerrado, en el sentido de que alguna operación a nivel elemento, soportada por el sistema (tal como **-Z**, o **A+B** o **X>Y**) podría tener un valor resultado que pertenezca a algún dominio del conjunto **D**.
- La capacidad de especificar exactamente, para cada dominio D_i que pertenezca a D, cuales son los operadores unarios aplicables a los elementos d_i de ese dominio D_i.
- La habilidad de especificar precisamente, para cada par de dominios D_i y D_j (no necesariamente distintos) que pertenecen a **D**, cuales son los operadores binarios que pueden ser aplicados a pares de elementos d_i y d_j, donde d_i E D_i y D_j E d_j.

Relaciones

Una **relación** sobre los dominios D_1, D_2, ..., D_n (no necesariamente distintos todos) consiste de un encabezado y un cuerpo:

- El encabezado consta de un conjunto fijo de atributos A_1, A_2, ..., A_n, de tal forma que cada atributo A_i corresponde a exactamente uno de los dominios D_i (i = 1, 2, ..., n).
- El cuerpo consiste de un conjunto de tuplas variables con el tiempo, donde cada tupla consta de un conjunto de pares atributo-valor (A_i:v_i) (i = 1, 2, ..., n), un tal par para cada atributo A_i del encabezado. Para algún par atributo-valor dado (A_i:v_i), v_i es un valor del dominio único D_i que está asociado con el atributo A_i.

El número **n** (que indica el total de atributos en la relación o, equivalentemente, el total de dominios básicos), es llamado el **grado** de la relación. Una relación de grado uno es llamada unaria y consecuentemente, una relación de grado **n** es llamada n-aria. Asimismo, al número de tuplas en una relación se le denomina cardinalidad. La cardinalidad de una relación cambia con el tiempo, mientras que el grado no.

Propiedades de las relaciones

Las relaciones poseen importantes propiedades, consecuencias directas de la definición de relación, que son las siguientes:

1. No tienen tuplas duplicadas: Esta propiedad resulta de que el cuerpo de la relación es un conjunto matemático (un conjunto de tuplas), y por definición los conjuntos matemáticos no incluyen elementos duplicados. Un corolario importante de este punto es que la Llave Primaria siempre existe, pues dado que las tuplas son únicas se tiene que en la mínima combinación de todos los atributos de la relación tiene la propiedad de unicidad, así que ésta mínima combinación puede servir de Llave Primaria. En la práctica usualmente no es necesario el involucrar a todos los atributos, pues generalmente una combinación más pequeña es suficiente ya que la Llave Primaria no incluye atributos que sean superfluos para el propósito de identificación única. Sin embargo, la Llave Primaria puede ser compuesta, es decir, involucrar varios atributos simples.

2. Las tuplas no tienen un orden. Esta propiedad también se deriva del hecho de que el cuerpo de la relación es un conjunto matemáticamente hablando. Los conjuntos matemáticos no están ordenados. Las tuplas podrían colocarse en cualquier secuencia y continuar siendo la misma relación. El hecho de que las relaciones son típicamente representadas en una forma tabular tiene como consecuencia varias ideas que no son verdaderas, como el que las tuplas tienen un orden de arriba hacia abajo. Pero esto puede entenderse dado que las tablas son una representación concreta de una construcción abstracta: la relación, y ciertos aspectos de esta representación concreta no corresponden a algo de la construcción abstracta básica. No obstante esta debilidad, es un punto importante, del modelo relacional, el que sus estructuras de datos tengan una representación concreta que es familiar y fácil de entender.

3. Los atributos no tienen un orden. Esta propiedad se deriva de que el encabezado de una relación está también definido como un conjunto (un conjunto de atributos). Estrictamente hablando, los atributos siempre son referenciados por su nombre y nunca por su posición. La cuestión del orden de los atributos es otro punto donde la representación concreta de la relación como una tabla sugiere cosas que no son realmente ciertas: Las columnas de una tabla tienen, obviamente, un orden de izquierda a derecha, pero los atributos de una relación no.

4. Todos los valores de atributo son atómicos. Esta propiedad puede enunciarse más exactamente así: "Todo valor de atributo simple es atómico", esto es consecuencia del hecho de que el dominio básico es simple, por tanto contiene valores atómicos únicamente. En el caso dado de que el dominio sea compuesto, sus valores no son otra cosa más que una simple concatenación de atributos simples. Se puede enunciar informalmente esta propiedad como sigue: En cada posición dentro de la tabla, existe precisamente un valor, nunca un conjunto de valores. Una relación que satisface esta condición se dice que está **normalizada**. Lo anterior implica que toda relación es normalizada dentro del modelo relacional. De hecho, el término "relación" siempre significa "relación normalizada" en el contexto del modelo relacional. Sin embargo, una relación matemática no necesariamente está normalizada, pues se puede dar el caso en que una relación tome sus valores de un dominio-relación, esto es, que sus elementos son relaciones también. El modelo relacional no permite la existencia de dominios con valores-relación. Además, una relación con este tipo de dominios puede ser reemplazada por una relación que semánticamente hablando sea equivalente y esté normalizada. El proceso de convertir una relación no normalizada, en una que sí lo esté, es llamado **Normalización.** El motivo por el cual se hace énfasis en que las relaciones deben ser normalizadas es el siguiente: Una relación normalizada es una estructura más simple, matemáticamente hablando, que una que no lo está, y por lo tanto, los operadores correspondientes son simples también y se requiere de un menor número de aplicaciones de estos operadores para realizar alguna tarea. Lo anterior no es sólo aplicable a las operaciones de manipulación de datos, sino a todas las operaciones dentro del sistema: Seguridad de los datos, Integridad de los datos, etc.

REGLAS DE INTEGRIDAD RELACIONAL

La parte de integridad en el modelo relacional consiste de dos reglas generales: integridad de identidad e integridad referencial. Estas reglas son generales en el sentido de que se aplican a cada base de datos que esté basada en el modelo relacional. Por supuesto,

alguna base de datos dada, tendrá reglas específicas para sí misma aparte de estas dos reglas generales. Las dos reglas generales tienen que ver con las llaves primarias y externas respectivamente.

Llaves Primarias

Las llaves primarias son un caso especial de una construcción más general llamada llaves candidatas. Una llave candidata es básicamente un identificador único. Por definición cada relación tiene al menos una llave candidata, en la práctica, muchas relaciones tienen exactamente una, pero es posible que algunas tengan dos o más. Para una relación dada, se selecciona una de las llaves candidatas para ser la llave primaria y las restantes (si existen) son llamadas llaves alternativas.

Más formalmente, si **R** es una relación con Atributos A_1, A_2, ..., A_n. El conjunto de atributos $K = \{ A_i, A_j, ..., A_k \}$ de **R** es llamado Llave Candidata de **R** si y sólo si satisface las siguientes propiedades independientes del tiempo:

- Es único: En un tiempo dado, dos distintas tuplas de **R** no tienen el mismo valor para A_i, el mismo valor para A_j,...,ni el mismo valor para A_k.
- Es mínimo: Ningún A_i, A_j, ..., A_k puede ser eliminado de **K** sin destruir la propiedad de unicidad.

Para una relación dada, la elección de la llave primaria se hace arbitrariamente del conjunto de llaves candidatas. La importancia de la llave primaria reside en que es la que provee un mecanismo de direccionamiento a nivel tupla dentro del modelo relacional. Esto es, la única forma de que el sistema garantice el reconocimiento de una tupla es vía la combinación (**R**, **k**), donde **R** es el nombre de la relación y **k** es el valor de la llave primaria para la tupla de interés.

Frecuentemente se afirma que el modelo relacional requiere de "direccionamiento asociativo", y dado que el direccionamiento relacional, en particular el direccionamiento por la llave primaria, es claramente asociativo ya que está basado en valores y no en posiciones, la afirmación es verdadera en el nivel lógico. Pero esto no implica que el modelo relacional requiera de hardware asociado, ya que el modelo se refiere al nivel lógico del sistema no al físico, y por tanto los dispositivos de entrada/salida y las estructuras de almacenamiento convencionales soportan adecuadamente un sistema relacional.

Llaves Externas.

En general, una llave externa es un atributo (o combinación de atributos) en una relación R_2 cuyos valores son requeridos para hacer una correspondencia con la llave primaria de alguna relación R_1 (R_1 y R_2 no necesariamente distintas). Las llave externa y la correspondiente llave primaria podrían ser definidas en el mismo dominio básico. No es requisito el que la llave externa sea un componente de la llave primaria sin embargo puede darse el caso.

La correspondencia de la llave externa a la llave primaria representa referencias de una relación a otra, es decir, esta correspondencia representa ciertas relaciones entre tuplas. Sin embargo no todas las relaciones entre tuplas pueden ser representadas por esta correspondencia.

Las dos reglas de Integridad
Ahora se pueden especificar formalmente las dos reglas de integridad del modelo relacional:

1. **Integridad de Identidad.** Ningún atributo, que forme parte de la llave primaria, puede aceptar valores nulos. La justificación para la regla de integridad de identidad engloba las siguientes afirmaciones:
   - Las relaciones básicas corresponden a entidades en el mundo real.
   - Por definición, las entidades en el mundo real son distinguibles, esto es, tienen una identificación única de algún tipo.
   - Las llaves primarias realizan la función de identificación única en el modelo relacional.

   Un valor nulo en la llave primaria sería una contradicción, pues significaría que alguna entidad no tendría un identificador y por lo tanto no existiría.

2. **Integridad Referencial.** Si la relación R_2 incluye una llave externa E_K que corresponde a una llave primaria P_K de alguna relación R_1, entonces cada valor de E_K en R_2 debe cumplir cualquiera de las siguientes condiciones:
   - Ser igual al valor de P_K en alguna tupla de R_1.
   - Ser totalmente nulo, es decir, cada valor de atributo de E_K debe tener el valor nulo. Desde el punto de vista de la integridad, un valor "nulo" simplemente significa un valor que es sobreentendido por convención y no está representado por un valor real dentro del dominio aplicable.

   La regla de integridad referencial tiene un intento básico: si alguna tupla t_2 referencia a alguna tupla t_1, entonces la tupla t_1 debe existir. Por lo tanto, es claro que dado un valor no nulo para la llave externa debe existir el correspondiente valor de la llave primaria en alguna parte de la relación referenciada.

Implicaciones de las Reglas de Integridad.
Las dos reglas de integridad están formuladas en términos de estados de la base de datos. Si algún estado de la base de datos no satisface las dos reglas implica que la definición de la base es incorrecta. Sin embargo, las reglas no indican por si mismas cómo evitar los estados incorrectos.

Una posibilidad es que el sistema pueda rechazar simplemente alguna operación, que de ejecutarse, provocaría un estado ilegal. No obstante, en muchos casos una alternativa preferible es el aceptar la operación pero realizar ciertas operaciones de compensación, si es necesario, para garantizar que el resultado sea un estado legal.

Hasta aquí se han presentado los conceptos fundamentales, en cuanto a bases de datos y el modelo relacional, que es necesario tener presentes al realizar la conversión del modelo de datos al esquema relacional. A continuación se muestra la utilización de UML en la elaboración del modelo de datos.

2. MODELADO DE DATOS CON UML

Actualmente, la mayoría del modelado de Bases de Datos utiliza alguna variante del modelo Entidad-Relación (ER). Tales modelos se enfocan en las cosas y sus relaciones, sin embargo el modelo ER no es apropiado para el modelado de la arquitectura de un sistema completo, ya que su notación no hace explícitos: el flujo de datos, los procesos que conforman el sistema, el contexto exterior del sistema, las interfaces con los usuarios, etc., elementos indispensables para la definición de la arquitectura del sistema. Dado que la base de datos es parte de un sistema mayor, es necesario realizar su modelado dentro de una estructura unificada que diseñe sistemas, no sólo partes.

El enfoque orientado a objetos plantea el criterio de la reutilización de componentes; considerando que un componente es una parte que es correcta y que sólo se necesita incorporar al sistema sin hacer modificaciones internas. De esta forma un sistema será el conjunto de componentes que sólo necesitan conectarse. Cuando se requiera modificarlo bastará con cambiar los componentes involucrados sin alterar la estructura del sistema. Así se minimizan los costos de mantenimiento e inclusive los de desarrollo mejorando la relación costo-beneficio del sistema.

Este trabajo aborda UML (Unified Modeling Language) [2.1] porque además de ser la conjunción de muchos años de trabajo de tres expertos modeladores de objetos (Grady Boock, Ivar Jacobson, James Rumbaugh) que han reunido sus técnicas dentro de una notación simple y completa, es el lenguaje de modelado que el Object Managment Group ha aprobado como estándar. También porque representa un desarrollo evolucionario, pues no rompe con los avances anteriores, más bien construye sobre lo ya probado. El UML proporciona elementos para el modelado de todos los aspectos de un sistema en general y de un sistema de software en particular, desde los requerimientos hasta la implementación, pero aquí se presentan sólo los aspectos necesarios para obtener el modelo de datos que después se traducirá al modelo relacional.

2.1 ANÁLISIS DE REQUERIMIENTOS

Como con toda herramienta de modelado de sistemas, el modelado con UML inicia con la recolección de los requerimientos del usuario, pues este es el punto inicial para el diseño del sistema en general y de la base de datos en particular. Esta es una etapa fundamental para el éxito de todo proyecto, pues el reunir correcta y claramente los requerimientos permite realizar un mejor análisis de los mismos. Por experiencia he visto que los requerimientos proporcionados por los usuarios, sufren de falta de precisión, están llenos de ambigüedades y contradicciones y a veces son demasiado breves o incompletos.

El objetivo principal no es el levantamiento de requerimientos en sí, sino hacer que el equipo de desarrollo se familiarice rápidamente con el problema. La calidad del sistema está en función directa del conocimiento que el equipo de desarrollo tenga acerca del problema, pues sería casual el crear soluciones efectivas de problemas que no se conocen. El proceso de análisis de requerimientos en UML pretende disminuir el tiempo que se invierte en esta adquisición de conocimiento y obtener un modelo sobre el cual basarse para realizar el detalle necesario.

Inicialmente se debe realizar una exploración con el o los usuarios para clarificar los requerimientos, después elaborar una definición del problema y preguntarse si el problema tiene solución, en caso afirmativo, se hará un refinamiento del problema utilizando como guía ya sea el sistema existente, o personas (usuarios) que lo hayan enfrentado anteriormente, pues con ello se podrán identificar los problemas fundamentales que el sistema trata de manejar y las limitaciones que tiene. Para esto se pueden utilizar preguntas libres de contexto del tipo de quién, cómo, cuando, dónde, en referencia al problema.

Al iniciar el desarrollo de un sistema, lo primero que se pregunta es qué es lo que el sistema va a hacer. El usuario empieza a describir lo que espera del sistema, y la labor del analista es limitar al usuario para que indique los procesos que a su juicio son los más importantes y que caracterizan al sistema, tarea nada fácil ya que el analista apenas empieza a conocer el problema, sin embargo se puede acotar al usuario pidiéndole que trate de indicar a lo más siete u ocho procesos. De esta manera, se elimina de entrada gran cantidad de procesos que en el inicio sólo causan confusión y restan atención a los aspectos fundamentales. Es muy importante resolver la ambigüedad en esta parte del modelado, pues el establecimiento claro y válido de los requerimientos es la plataforma para el diseño. La ambigüedad se establece cuando es posible interpretar en forma diferente los requerimientos determinados para un sistema o base de datos. Para resolver la ambigüedad es necesario: hacer preguntas al cliente, formular situaciones, e investigar más profundamente el significado de las palabras ambiguas.

El tener una lista de lo que la gente quiere no significa haber entendido los requerimientos. Hay que pensar acerca de lo obtenido y también de lo no obtenido así como sus efectos laterales. Aunque no existen recetas mágicas para el éxito las siguientes recomendaciones son útiles:

1. Establecer las metas claramente
2. Entender dónde hay compromisos y conflictos entre metas
3. Establecer prioridades y cambiarlas cuando sea necesario
4. Modelar el sistema incluyendo efectos laterales y cambios a largo plazo
5. Buscar cuanto dato se requiera y entenderlo.
6. No ser excesivamente abstracto.
7. No reducir todo a una simple causa o hecho.
8. Atender las sugerencias de otros diseñadores.
9. Estudiar la historia propia y aplicar las lecciones aprendidas en el nuevo sistema.
10. Pensar en términos de sistemas, no de simples requerimientos.

Es necesario priorizar los requerimientos para enfrentar la complejidad de un sistema. Además, entender el alcance de los requerimientos permite determinar el tipo de arquitectura que el sistema requiere. En el nivel más fundamental se pueden observar cuatro tipos de requerimientos:

1) **Objetivos Operacionales.** Expresan el propósito del sistema y el método en el que se intenta llegar a ese propósito. Metas, funciones, conductas y operaciones son sinónimos de objetivos operacionales. Son el tipo más importante de requerimientos porque expresan el significado fundamental del sistema, el por qué de lo que se está haciendo. Algunos objetivos están relacionados con el usuario y otros están escondidos para permitir realizar otro objetivo o para hacer el sistema más flexible y fácil de usar.

2) **Propiedades de objeto.** Probablemente es la categoría más importante para el diseñador de la base de datos. Mientras se desarrollan los requerimientos de datos del sistema, se empiezan a ver los objetos que existen y sus propiedades. Estas pueden ser columnas de tablas, atributos de objetos o miembros de una clase de datos. No todas las propiedades se relacionan directamente a elementos de la base de datos.

3) **Reglas.** Son requerimientos condicionales sobre las propiedades de objeto. La mayoría de las restricciones de integridad referencial están en esta categoría de requerimientos. La regla de tiempo de respuesta también está dentro de esta categoría, así como las reglas de calidad para establecer tolerancia o fronteras. Es importante tener cuidado con las reglas a fin de no hacer un sistema demasiado rígido.

4) **Preferencias.** Son condiciones sobre propiedades de objeto que expresan un estado preferido. Los valores por omisión representan una preferencia. Es necesario entender la diferencia entre regla y preferencia; las primeras son cuestiones de integridad, las segundas de deseo y gusto.

Es necesario relacionar los requerimientos entre sí, tratándolos como un sistema. Las conexiones que se pueden establecer entre requerimientos usualmente indican cómo se puede organizar sus subsistemas, por ejemplo, si un conjunto particular de requerimientos se relacionan fuertemente con otros pero están relativamente desconectados a otros, ahí hay un candidato a subsistema. En cambio, si intuitivamente se siente que varios requerimientos deben ir juntos pero no se encuentra la manera de relacionarlos seguramente se ha omitido algo.

Algunos requerimientos son vitales para el proyecto, mientras que otros son irrelevantes. Desafortunadamente, se tiene que pasar por todo para ganar experiencia en la capacidad de distinguir entre irrelevancias y cosas importantes. Cada uno de los diferentes tipos de requerimientos tiene un conjunto diferente de prioridades.
Los objetivos operacionales se pueden priorizar por su contribución a la misión básica del sistema como:

- Crítico: el éxito del sistema requiere de la realización adecuada del objetivo.
- Marginal: el objetivo se puede pasar por alto con problemas menores.
- Incorrecto: el objetivo no está relacionado con la misión del sistema.

Las propiedades objeto se priorizan en categorías similares:

- Indispensable: es una propiedad sin la cual el sistema no logrará su misión.
- Deseada: es una propiedad que es conveniente tener.
- Ignorada: es una propiedad que puede ser o no útil y que no vale la pena invertir tiempo en considerarla.

Las preferencias se deben priorizar por su naturaleza. Las reglas aplican a situaciones que se deben tener. Si se ha expresado una regla, se tiene que incluir. En cambio, las preferencias deben dividirse en posibles y óptimas, lo cual es una de las cosas más difíciles en el análisis de requerimientos. Las prioridades son relativas pues dependen de la naturaleza de la gente involucrada en el proyecto.

Entender los requerimientos, relacionarlos y clasificarlos es muy importante. Es un proceso iterativo que requiere de un profundo análisis, el cual puede ser auxiliado a través de los casos de uso (elemento del UML para modelado de requerimientos), a fin de determinar cuáles son prioritarios y cuáles dependen de otros.

2.2 MODELADO DE REQUERIMIENTOS CON CASOS DE USO

Un caso de uso en UML describe, de manera resumida, una situación en términos que el lector pueda entender fácilmente. Elaborados apropiadamente los casos de uso de un sistema describen en el nivel más alto lo que el sistema debería hacer, por lo que más que útiles son esenciales. Los casos de uso provienen del trabajo original de Jacobson. El UML utiliza el término **escenario** para un ejemplar de caso de uso, un camino particular a través del caso de uso.

Un **actor** en UML caracteriza el rol (activo o pasivo) jugado por un objeto exterior que puede ser un usuario, otro sistema de software o una pieza de hardware. Un objeto físico puede jugar varios roles y por lo tanto ser modelado por varios actores. Los actores proporcionan el contexto para el sistema. Los casos de uso representan las transacciones que los actores realizan a través del sistema. Si impacta en los requerimientos, se puede definir como actores a elementos que el sistema controla o usa; por ejemplo, la base de datos puede ser especificada como un actor. Los actores base de datos proporcionan una forma para especificar el empleo de los datos en los casos de uso.

A partir de la descripción del problema generada por la recolección de requerimientos, es posible obtener una definición cuidadosa de las fronteras del sistema y, antes de elaborar los casos de uso, es útil empezar con el modelo de actores. Puede que ciertos actores se relacionen entre sí mediante relaciones de generalización. El modelado de actores de manera jerárquica permite simplificar las descripciones de los casos de uso. En este contexto la herencia significa que un subactor hereda el rol jugado por su actor padre.

Para ejemplificar un modelo de actores se presenta el ejemplo tomado [2.2] en el cual una agencia de detectives privados desea automatizar su sistema manual de registro de casos a fin de que sus detectives en todo el mundo puedan obtener información (en forma textual y gráfica) acerca de hechos relevantes para la práctica de su trabajo. Los hechos relevantes son: casos anteriores, biografías de delincuentes, anuncios de personas extraviadas, etc. La información proviene de diversas fuentes, y se debe tener en cuenta las leyes referentes a propiedad intelectual. La información ha de ser confiable y estar resguardada de gente que pueda hacer mal uso de ella.

De esta descripción se puede identificar varios actores: .

- Detective privado: Es el principal usuario del sistema, además hay diversos tipos de detective dependiendo de la actividad que realizan.
- Informante: la información puede provenir de diversas fuentes: policía, agencias de medios de información, etc.
- Ingeniero de calidad de datos: La calidad de los datos es vital, información errónea impide el trabajo del detective.
- Administrador de la base de datos: mantiene la base de datos.
- Administrador del seguridad: El acceso seguro al sistema es necesario para mantener el secreto y confidencialidad durante la investigación.

- Representante legal: para resolver los problemas con las leyes de invasión a la privacidad y derechos de propiedad intelectual pues muchas de las fuentes de información para el sistema son públicas.
- Persona de ventas y mercadeo: El sistema tiene un gran valor para la agencia, pues si tiene buenos y mejores resultados en sus investigaciones se incrementa la demanda de los servicios de sus detectives.

El modelo jerárquico de actores sería el siguiente:

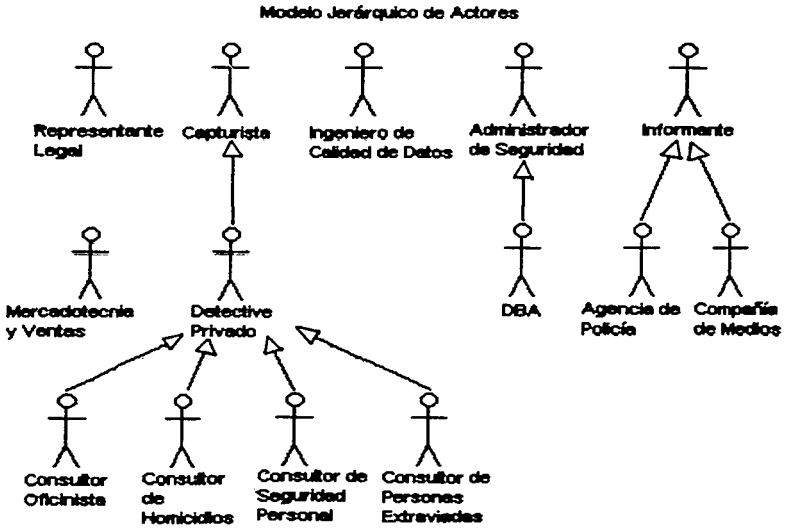


Fig. 2.1 Modelo Jerárquico de Actores del Sistema de Información para Detectives

Teniendo el modelo de actores se puede desarrollar la parte principal del modelado de casos de uso. Aunque es un proceso iterativo, con frecuencia la definición inicial de actores ayuda a establecer los casos más claramente. En el análisis del sistema, los casos de uso deben detallarse creando una serie anidada de ellos, que se combinan para formar el sistema total. El descomponer los casos de uso plantea el punto de manejo de transacciones. Desde la perspectiva de transacciones realmente hay dos tipos de casos de uso: atómicos (tienen asociaciones con actores) y subatómicos (extienden o son usados por otros casos de uso). Algunos SABD orientados a objetos soportan un modelo de transacciones anidado. La mayoría de los SABD relacionales proporcionan el concepto de punto de verificación (checkpoint): un punto en el conjunto secuencial de acciones de una transacción, en el cual ésta puede deshacerse. El uso de transacciones anidadas o puntos de verificación son tópicos de administración avanzada de base de datos.

Al definir las transacciones también se puede considerar algunas de las implicaciones para su procesamiento. La regla en el diseño de transacciones es hacerlas tan cortas como

sea posible. En los inicios del proyecto es importante investigar las opciones tecnológicas para evitar sorpresas al final, particularmente si se tienen transacciones con demasiados datos (cien mil registros o más en una consulta significan demasiados). Aunque estos puntos tecnológicos parecen de implementación y no de requerimientos, la fuente de muchos problemas es el no tomar en cuenta el procesamiento básico de transacciones en los casos de uso.

Hay dos tipos de relación entre casos de uso UML:

1. **Relación de uso.** Expresa factores comunes entre casos de uso. La relación significa que la secuencia de conducta descrita en el caso de uso es incluida en la secuencia de otro caso. Si un caso de uso tiene varias relaciones de uso, su secuencia será el resultado de la intercalación de las secuencias empleadas junto con nuevas piezas de conducta. La manera en cómo se combinan estas partes para formar la nueva secuencia se define en el caso utilizador. Las relaciones de uso utilizan incondicionalmente un caso que dos o más casos de uso comparten, permitiendo un estilo modular en abanico. Así se pueden crear casos de uso compartidos en lugar de transacciones largas compartidas, reduciendo la complejidad y redundancia en todo el sistema. La relación de uso permite consolidar una parte del caso de uso en otro separado que se puede compartir con múltiples transacciones. Esto permite identificar conducta compartida correspondiente a código compartido en el diseño. La relación de uso más común es el compartimiento de una parte simple de conducta tal como el despliegue de una gráfica o el cálculo de un valor algorítmico.

2. **Relación de extensión.** Un caso de uso puede ser extendido con alguna conducta adicional definida en otro caso de uso. Las relaciones de extensión incluyen tanto condición para la extensión como una referencia al punto de extensión en el caso relacionado, es decir, una posición en el caso de uso donde se pueden hacer las adiciones. Una vez que se alcanza un punto de extensión, la condición es evaluada. Si la condición se cumple, la secuencia seguida por el caso de uso es extendida para incluir la secuencia del caso de uso extensión. Diferentes partes del caso de uso extensión pueden ser insertadas en diferentes puntos de extensión de la secuencia original. Si sólo hay una condición, entonces la conducta extendida completa es insertada en la secuencia original. Las relaciones de extensión permiten simplificar un caso de uso excluyendo los elementos condicionales (excepciones) que distraen del punto principal de la transacción. El ejemplo más común es el manejo de errores. Se pueden extender casos de uso diferentes con el mismo caso de uso extensión (bibliotecas).

La utilización de relaciones de uso y de extensión en el modelado de casos de uso tienen dos efectos principales:

- Simplifican el modelo eliminando distracciones condicionales y conducta compartida.
- Representan la misma conducta sólo una vez en lugar de muchas veces.

DIAGRAMA DE CASOS DE USO

Es un diagrama de contexto que muestra el nivel más alto de relación entre los actores y los casos de uso. También se pueden incluir relaciones entre casos de uso. Los elementos de un diagrama de casos de uso son:

- Rectángulo con esquinas redondeadas que representa la frontera del sistema.
- Óvalos etiquetados (ya sea dentro o debajo) que representan los casos de uso y se colocan dentro del rectángulo.
- Figuras delgadas que representan los actores que se colocan fuera del rectángulo.
- Líneas que son las asociaciones que unen a los actores con los casos de uso. La asociación es una relación de comunicación que muestra que el actor se comunica con el caso de uso en alguna forma (activa o pasiva).

Se inicia con un actor pensando en las transacciones que genera, se crea un caso de uso para cada una de las transacciones y se unen al actor por medio de una asociación. Usualmente debe empezarse con los actores de más alto nivel en la jerarquía, pues un actor hijo automáticamente comparte la comunicación que se haya dibujado para su actor padre, y se va trabajando hacia abajo, adicionando casos de uso especializados sólo para los actores hijo que lo requieran.

Después se debe analizar las transacciones creadas para decidir si realmente son atómicas. Esta es una parte crítica de la definición tanto del sistema como de la base de datos. Una transacción tiene una serie de operaciones que el caso de uso describe. Hacer series atómicas significa que se debe terminar la serie completa o eliminar totalmente los efectos del sistema: no se puede completar sólo parte de la serie. Finalmente, si hay actores restantes que se comunican con los casos de uso que se han definido entonces también se deben conectar. Para el sistema de información para detectives, el diagrama de casos de uso de más alto nivel sería el mostrado en la Fig. 2.4.

Para mostrar las relaciones entre casos de uso, supóngase que hay dos formas de identificar un criminal: por alias o por evidencia (huellas digitales, DNA, etc) y cuando se encuentra al criminal el sistema presenta información como nombre, alias, huellas digitales, fotos, etc. Entonces se pueden hacer dos casos de uso **Identificar con alias** e **Identificar con evidencia** y relacionarlos con el caso **Identificar Criminal** en una relación de uso (flecha que va del caso utilizador al caso utilizado):

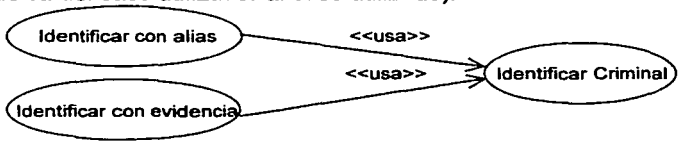


Fig. 2.2 Relación de uso

Si al hacer la búsqueda por alias, no se encuentra criminal que corresponda al alias indicado, el caso de uso **No se pudo identificar criminal** puede extender a **Identificar con alias** (flecha que va del caso extensión al caso extendido):

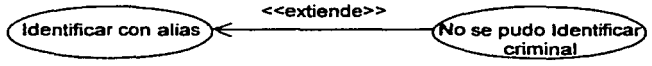


Fig 2.3 Relación de extensión

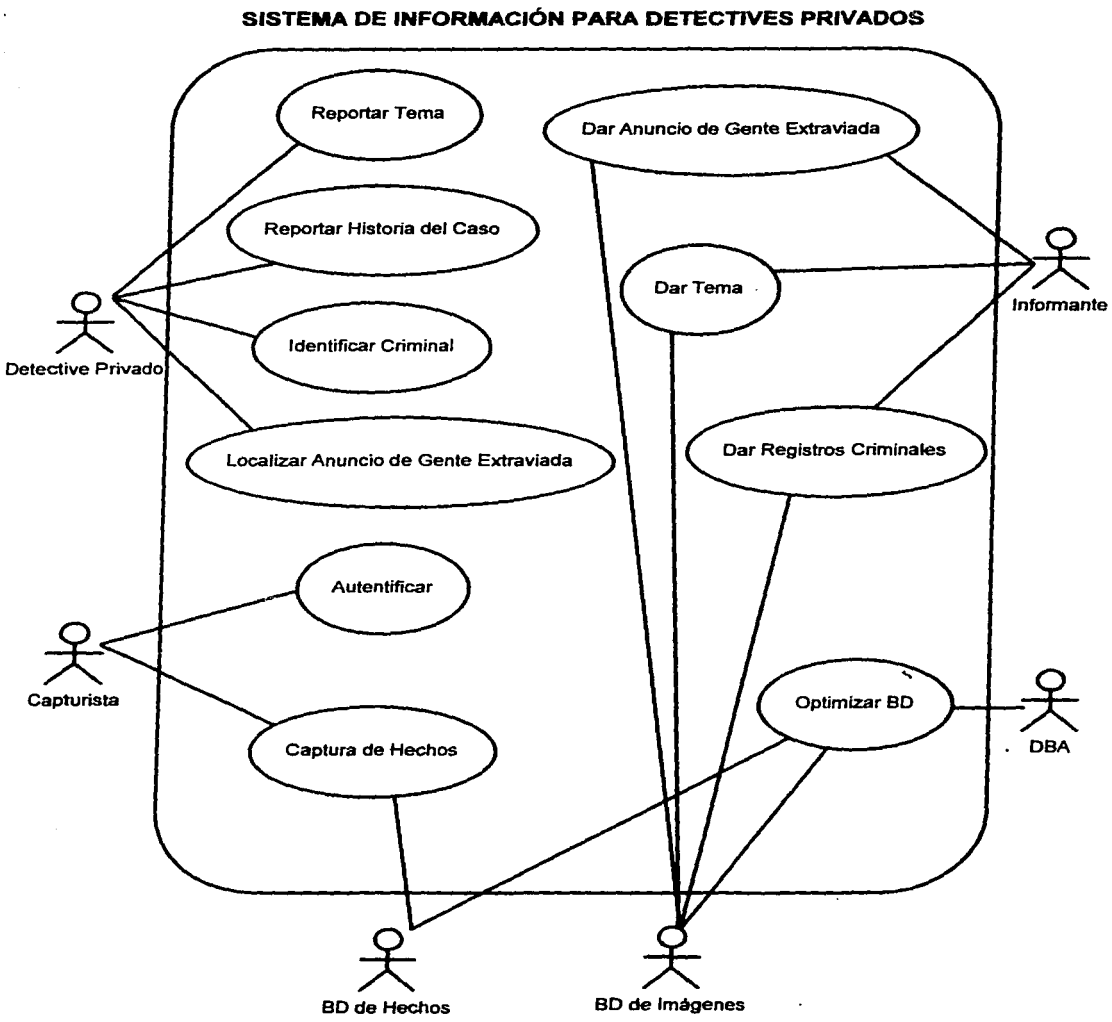


Fig. 2.4 Diagrama de casos de uso del Sistema de Información para Detectives Privados

Una vez obtenido el modelo de casos de uso más general, se requiere detallar cada caso de uso.

ELEMENTOS DE UN CASO DE USO

El objetivo principal de un caso de uso es expresar el significado y conducta del sistema que representa, esto puede lograrse con una descripción tan simple como un párrafo de texto (historia del caso de uso), y aunque siempre es necesario el criterio propio para decidir el nivel de detalle y formalidad en el caso de uso, puede resultar más expresivo y completo el incluir los siguientes elementos: Resumen, Narrativa, Diagrama de actividad UML, Tabla de datos, Resumen de reglas del negocio.

Resumen de caso de uso

Párrafo corto que describe el propósito del caso de uso, expresando la esencia, no los detalles, ya que su objetivo es proporcionar un entendimiento rápido. Se debe mencionar los actores con los que se comunica el caso de uso.

Para casos de uso que expresan puntos en común o excepciones a través de relaciones de uso o de extensión, se debe resumir el propósito y mencionar la naturaleza compartida del caso de uso o las condiciones probables que conducen al caso. Hay dos puntos que deben satisfacerse: El resumen debe indicar que la transacción no es una transacción completa en sí misma y los futuros usuarios (en el mismo proyecto o en uno posterior) deberán ser capaces de entender lo que el caso de uso proporciona para reutilizar. Considerando el modelo de casos del Sistema de Información para Detectives presentado en la Fig. 2.4, el **resumen** del caso de uso **Identificar Criminal** podría ser el siguiente:

> El caso de uso **Identificar criminal** proporciona conducta compartida para casos de uso más específicos: **Identificar con alias** o **Identificar con evidencia**. El caso de uso inicia con un identificador de criminal generado por el caso de uso relacionado. Después despliega el nombre del criminal, una foto, una lista de alias, una lista de documentos de identificación y un breve resumen de la historia criminal del individuo.

Narrativa

Usualmente toma la forma de una lista numerada de pasos que se alternan entre los actores. No es necesario especificar interfaces de usuario o detalles del sistema, esto permite que los requerimientos enfrenten sólo la conducta genérica de la aplicación, dejando a la etapa de diseño las decisiones de interfaz con el usuario. Si el caso de uso hace referencia explícita a la base de datos se puede expresar el paso usando un lenguaje de acceso a la BD en forma general, para esto, los diagramas de estructura estática deberían elaborarse de manera paralela e iterativa con los casos de uso a fin de contar con un esquema básico con nombres de objetos, tablas y atributos.

La narrativa debe referir las relaciones de uso directamente en los pasos que utilizan la conducta compartida. En cuanto a las relaciones de extensión sólo debe identificarse el punto donde aparece la condición de extensión pues la última parte de la narrativa es una sección de extensiones, allí se indican las relaciones de extensión y las condiciones bajo las cuales ocurren. Continuando con el caso de uso **Identificar criminal**, la **narrativa** sería la siguiente:

1. El caso de uso **Identificar criminal** inicia cuando recibe un identificador de criminal de otro caso de uso que lo utiliza.
2. El sistema busca nombre del criminal, foto, lista de alias, lista de documentos de identificación y resumen de la historia criminal correspondientes al identificador.

3. El sistema despliega el nombre del criminal, una foto, una lista de alias, una lista de documentos de identificación y un breve resumen de la historia criminal del individuo.

Extensiones

1. El caso de uso **No se encontró objeto** extiende a **Identificar criminal** en el paso 3 cuando no se encuentra cualquiera de los elementos buscados (nombre, foto, lista de alias, lista de documentos de identificación, resumen de la historia criminal) que correspondan al identificador de criminal dado.

Diagrama de Actividad UML

Se utiliza para representar el control lógico del caso de uso en un formato gráfico. No maneja situaciones con eventos externos o asíncronos. Los diagramas de actividad reflejan dos estructuras de control básicas: secuencia y condición. La secuencia es la conexión dirigida entre los elementos; una condición es el equivalente a una instrucción IF. También se puede representar la iteración a través de una secuencia que conecte a un elemento de una secuencia anterior. El diagrama de actividad sirve como modelo de prueba para el caso de uso.

Elementos del Diagrama de Actividad UML

1) Estados de acción: Rectángulos con esquinas redondeadas que representan una acción sin transiciones internas.
2) Transiciones: Flechas dirigidas que conectan los estados de acción.
3) Condiciones: Etiquetas en las transiciones o rombos sobre la transición.

Los estados de acción corresponden a los procesos en un diagrama de flujo, las transiciones al flujo de control. El formato de decisión es más parecido al del diagrama de flujo. Las condiciones generalmente deben ser booleanas y no pueden tener eventos asociados. Debido a que SQL y las bases de datos en general, soportan valores nulos en una lógica tri-valuada, tiene sentido el permitir este tipo de lógica para las condiciones en casos de uso basados en procesamiento SQL. En los estados de acción se puede utilizar un lenguaje de acceso a la BD (excepto si la proposición es muy larga), pues así se expresa clara y directamente el contenido del paso.

Siguiendo con el ejemplo, el **diagrama de actividad** para el caso **Identificar criminal** se puede ver en la Fig. 2.5 e indica que los pasos 1 y 2 son secuenciales y hay una condición que de cumplirse continua con el paso 3a, en caso contrario se realiza el paso 3b.

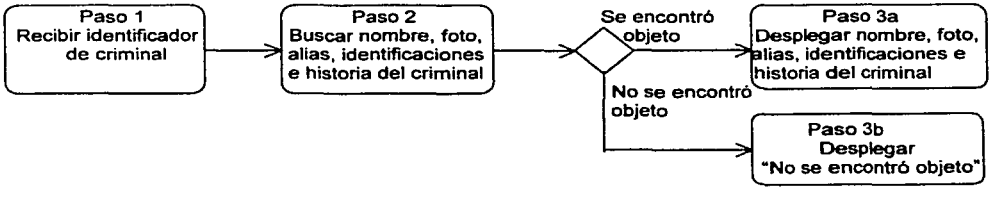


Fig 2.5 Diagrama de Actividad del Caso de Uso Identificar Criminal

Tabla de datos y resumen de reglas del negocio

En la parte final del caso de uso se pueden expresar los detalles de los requerimientos de la base de datos. La narrativa y el diagrama de actividad refieren de manera informal datos y restricciones; en esta parte se pueden reunir todos los datos requeridos para

mostrar el impacto completo del caso de uso sobre la base de datos. Cualquier dato, al cual hagan referencia las sentencias SQL en la narrativa o el diagrama de actividad, deberá aparecer aquí, organizado dentro de su tabla, tipo o clase.

Las reglas del negocio son restricciones sobre los datos u operaciones. Se deben especificar estas reglas en el caso de uso para asegurar que en el diseño se consideren y es muy útil ponerlas en forma resumida a fin de que se entiendan. El diseño de la base de datos debe reflejar completamente las reglas que los casos de uso imponen sobre los datos,. Algunas veces las reglas del negocio de diferentes casos pueden entrar en conflicto, haciendo necesario el diseñar capas de negociación para manejarlo. Si se quiere, se pueden expresar las reglas en SQL o en un lenguaje lógico equivalente, pero usualmente es mejor hacerlo en un formato de lenguaje natural que se pueda entender fácilmente, dejando al modelo de clases expresar formalmente las restricciones.

La **tabla de datos** para el caso **Identificar criminal** podría contener:

Tabla	Columna	Comentario
Persona	IdPersona	Identificador único para una persona
	Nombre	Nombre completo de una persona
Alias	Alias	Alias del nombre de una persona

Resumen de reglas del negocio.
Una persona puede tener varios alias. Diferentes personas pueden usar un alias. Un alias debe referirse al menos a una persona.

Una vez que se tienen detallados los principales casos de uso que conforman el sistema, es necesario realizar una revisión, que como con cualquier prueba no es de una sola vez. Se inicia presentando los casos de uso a los usuarios a fin de lograr una retroalimentación con respecto a sus requerimientos. Mientras se va procediendo al diseño y la implementación, se pueden encontrar partes que requieren revisión, inclusión o formas diferentes de ver el problema, esto provoca más entrevistas con el usuario y observaciones. El principal riesgo es que los requerimientos establecidos no reflejen correctamente las necesidades reales de los inversionistas. La verificación de requerimientos significa reducir este riesgo por debajo de la tolerancia de error. En la práctica es obtener el acuerdo de los inversionistas en que los casos de uso reflejan lo que se espera del sistema.

Al realizar una reunión de revisión de requerimientos es necesario que participen tanto los diseñadores como los usuarios, también es conveniente considerar la presencia de un experto en la materia y de los inversionistas. La meta de la revisión es descubrir problemas en los requerimientos y por lo tanto es importante considerar los siguientes aspectos: Completez, Exactitud, Claridad, Consistencia, Relevancia, Pruebas, Seguimiento, Factibilidad, Necesidad, Adaptabilidad. Al examinar los casos de uso en estos aspectos se puede encontrar un gran número de problemas. El nivel de resolución de cada problema depende tanto de su contribución al riesgo total de falla como de la tolerancia que se tenga para ese riesgo.

2.3 MODELO DE DATOS

DIAGRAMAS DE CLASE

En UML una **clase** se define formalmente como la descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semánticas [2.3]. Una clase puede usar un conjunto de interfaces para especificar colecciones de operaciones que proporciona a su ambiente.

Es necesario aclarar el significado especifico de algunas palabras de la definición de clase:

Atributo: Propiedad que cualquier ejemplar de la clase (objeto) puede tener. Es muy similar a la definición relacional de atributo.

Operación: Un servicio que puede ser requerido por cualquier objeto.

Método: Es la implementación de la operación. Especifica el algoritmo o procedimiento que realiza los resultados de una operación.

Interfaz: Colección de operaciones que puede ser usada para la definición de un servicio ofrecido por un objeto.

La definición de clase es casi la misma que la de entidad en ER dado que las notaciones orientadas a objetos en las cuales esta fundado UML fueron desarrolladas de las notaciones ER. Las diferencias surgen principalmente en el modelado de operaciones y relaciones

Un clasificador es un clase abstracta UML que incluye las diferentes subclases que se pueden utilizar para clasificar objetos en cualquier forma por su estructura y conducta. Clases, tipos, interfaces, subsistemas, paquetes y bases de datos son tipos de clasificadores [2.4]

Un modelo en UML es un paquete que contiene una representación completa de un sistema desde una perspectiva de modelado dada. Se pueden tener diferentes modelos del sistema utilizando diferentes perspectivas o niveles, por ejemplo se puede tener un modelo de análisis consistente de casos de uso y un modelo de diseño formado por subsistemas y clases.

Un diagrama de clase es un diagrama de estructura o estático con el cual se modela la estructura de un sistema de clases, el diagrama de clases no sólo considera clases, también puede modelar interfaces, relaciones y hasta ejemplares individuales de clases, por lo que un nombre alternativo es: diagrama estructural estático.

Antes de detallar los diagramas de clase es necesario entender los componentes que conforman al sistema como un todo y estos son los paquetes. Un paquete en UML es un grupo de elementos modelo. Un subsistema en UML es un tipo de paquete que representa la especificación y la realización de un conjunto de conductas [2.5]. La especificación consiste en un conjunto de casos de uso y sus relaciones, operaciones e interfaces. La realización es un conjunto de clases y otros subsistemas que proporcionan conducta específica. Usualmente un subsistema agrupa varias clases, aunque puede haber subsistemas con una sola clase o subsistemas que no tienen clases sólo interfaces. Los subsistemas forman la base del diseño del sistema.

En UML existen otros tipos de paquete identificados por la adición del estereotipo al nombre del paquete. Un estereotipo es una frase entre << >>, que se pone a un símbolo para representar una extensión oficial de la semántica de UML, por ejemplo:

- << Sistema >>: El paquete que contiene todos los modelos del sistema
- << Fachada >>: Paquete que consiste sólo de referencias a otros paquetes.
- << Estructura >>: Paquete que representa una plantilla extensible a usar en un dominio específico.
- << Resguardo >>: Paquete que contiene sólo una interfaz pública, representando trabajo que aún no se ha completado o que se difirió por alguna razón.

El concepto de arquitectura de sistemas relacionado con paquetes ayuda a estructurar el sistema para una reutilización óptima, siendo el paquete el centro de la creación de componentes reutilizables. La arquitectura de un sistema debe pensarse como un conjunto de paquetes y sus interfaces, no como una colección de entidades interrelacionadas. Este aspecto de empaquetamiento para diseño de bases de datos es la diferencia entre el método ER y los métodos orientados a objetos. Las bases de datos llegan a ser paquetes en algún nivel. El dividir el sistema en varias bases de datos da la opción de modular el sistema para reutilizarlo.

Una forma adecuada de enfrentar el empaquetamiento de las clases es a través de los casos de uso. Para empezar el desarrollo de las clases se deben buscar objetos y sus atributos en los casos de uso. Se empieza a construir subsistemas agrupando los casos de uso que afectan objetos similares. Dado que los casos de uso corresponden a transacciones, este método garantiza que las transacciones se centren dentro de subsistemas en vez de estar esparcidas por todo el sistema. Durante la construcción de los casos de uso se tienen algunas ideas acerca de las clases que se necesitarán, del tipo de transacciones del sistema y debe ser posible el imaginarse unos cuantos paquetes que formarán una base para la construcción del modelo de arquitectura del sistema.

Tomando el ejemplo del sistema de información para detectives, el diagrama de paquetes inicial sería el mostrado en la Fig. 2.6

Cada paquete se representa por un símbolo de archivo etiquetado con el nombre del paquete y el estereotipo apropiado (en este caso <<Subsistema>>). Las dependencias entre paquetes se muestran con flechas punteadas (que salen del paquete dependiente); por ejemplo, en la Fig. 2.6 se observa que el subsistema **Consulta** depende de los subsistemas **Criminal** y **Organización Criminal**. Como se puede notar, un diagrama de paquetes muestra la estructura total del sistema.

Mientras se avanza en la definición de detalles del sistema, los paquetes adquieren detalles internos tales como clases, interfaces y colaboraciones; al final del diseño, se debe tener un conjunto de paquetes lo más independientes uno de otro y que interactúan a través de interfaces reutilizables bien definidas. Cada paquete debe estar dividido en un conjunto de diagramas de clase que se puede usar para documentar y dirigir la construcción del código del sistema, incluyendo la creación de la base de datos y el código de manipulación. Aunque el diseño arquitectónico del sistema es diferente al diseño de la base de datos, finalmente los subsistemas y paquetes tienen efectos directos sobre la estructura fundamental de la base de datos.

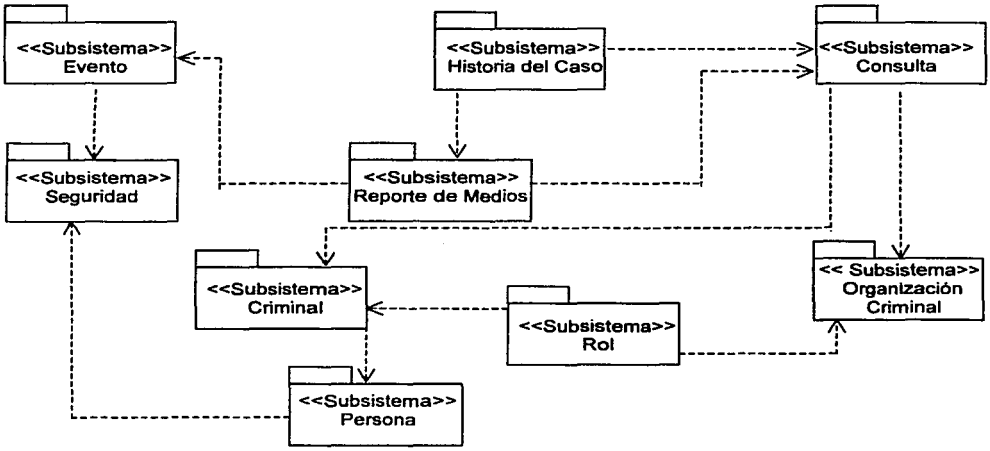


Fig 2.6 Diagrama de Paquetes del Sistema de Información para Detectives

La estructura de la base se diseña al elaborar clases con sus atributos. Usualmente cada uno de los subsistemas debe tener su propio diagrama de clase. El símbolo para una clase es un rectángulo con tres divisiones. La división superior contiene el nombre de la clase, la siguiente división contiene una lista de atributos, la división inferior contiene una lista de operaciones. En las tres divisiones se pueden tener propiedades del contenido respectivo. Las divisiones de los atributos y de las operaciones se pueden esconder y en este caso no es necesario dibujar la línea de división. Usualmente se esconden la mayoría de los detalles cuando se utiliza la clase como referencia en otro diagrama.

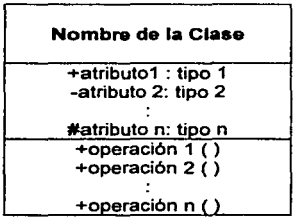


Fig. 2.7 Símbolo de una clase

El nombre de la clase debe ser único dentro del paquete. Se pueden tener clases con el mismo nombre en diferentes paquetes. Para referir a una clase de otro paquete, se precede el nombre de la clase con el del paquete en el formato "paquete::clase". La división superior puede contener también un estereotipo. Por ejemplo si se quiere que una clase sea persistente, se debe poner el estereotipo <<Persistente>>, lo cual indica que el sistema no destruye el estado de un ejemplar cuando destruye el ejemplar. Una clase

<<Persistente>> se convierte después a la implementación básica en este caso a una tabla relacional.

Una clase abstracta no tiene ejemplares. Principalmente existe para representar una abstracción común que varias subclases comparten. UML sugiere que el símbolo de la clase muestre la abstracción usando itálicas en el nombre de la clase. También se puede desplegar la propiedad entre llaves debajo del nombre de la clase: {abstracta}.

Los atributos tienen varios calificadores que se pueden poner o suprimir: estereotipo, visibilidad, tipo de dato, restricciones y propiedades. La sintaxis para la expresión de un atributo es:

estereotipo visibilidad nombre : tipo = valor inicial [cadena de propiedades]

Hay tres tipos de visibilidad y cada una se representa por un símbolo:

+ Pública: Cualquier otra clase puede examinar o cambiar directamente el valor del atributo.
# Protegida: Sólo los métodos de la clase o de su subclase pueden examinar o cambiar directamente el valor del atributo.
- Privada: Sólo los métodos de la clase (pero no de clases herederas) pueden examinar o cambiar directamente el valor del atributo.

Una práctica de diseño es hacer todos los atributos protegidos o privados, permitiendo el acceso directo sólo dentro de la jerarquía de clase, respectivamente. En implementaciones relacionales no es posible representar la visibilidad pues generalmente estos SABD sólo proporcionan atributos públicos, por lo tanto no hay encapsulación, pero ya que el diseño de la clase es tanto para la base de datos como para el modelo dominio del sistema, Muller [2.6] aconseja la visibilidad protegida como una mejor opción en clases persistentes reutilizables, particularmente para estos manejadores.
La expresión del tipo de dato es dependiente del lenguaje. Para el caso general de diseño UML se recomienda especificar atributos sólo con los tipos de dato primitivos del lenguaje. El valor inicial es el valor que el sistema proporciona automáticamente al atributo cuando se crea un ejemplar de la clase.

La división inferior del rectángulo de la clase contiene las operaciones. Se pueden especificar operaciones con el tipo de dato que devuelven. Actualmente los SABD relacionales cuentan con procedimientos almacenados por lo que es importante considerar la especificación de las operaciones en el modelo de datos. Esta es la sintaxis para indicar una operación:

estereotipo visibilidad nombre (lista de parámetros): expresión de tipo regresado { propiedad}

Los tres estereotipos para las operaciones en UML son: <<crear>>, <<destruir>> y <<señal>>. Los dos primeros representan constructores y destructores. La visibilidad toma los mismos valores que para los atributos. Si se declara un constructor protegido en vez de uno público significa que los otros objetos no pueden construir el objeto sin algunas propiedades especiales de acceso dadas. En la base de datos esto significa que las aplicaciones no pueden adicionar renglones a la tabla. El estereotipo <<señal>> tiene un uso especial para el modelado de datos que posteriormente se explica.

La lista de parámetros es una lista de variables y tipos de dato en una sintaxis de lenguaje específico, así como la expresión de tipo regresado. El nombre, la lista de parámetros y la expresión de tipo regresado conforman la firma (signature) de la operación la cual debe ser única dentro del alcance de la clase. Si aparece una segunda operación con la misma firma, UML la considera un método de la operación definida por la primera firma encontrada. La lista de parámetros se especifica con la siguiente sintaxis:

tipo : tipo-expresión = valor por omisión

El tipo es una de las diferentes posibilidades que describen el acceso de la operación al valor del parámetro:

- in: La operación puede leer el valor pero no puede cambiarlo (paso por valor)
- out: La operación puede cambiar el valor pero no puede leerlo o usarlo.
- inout: la operación puede tanto leer el valor como cambiarlo (paso por referencia)

El valor por omisión es un valor que el parámetro toma si no se pasa parámetro cuando se llama la operación.
Se puede marcar una operación con la propiedad {query} para indicar que la operación no tiene efectos laterales, es decir que no cambia el estado del sistema. Si se transforma la operación en procedimiento almacenado, esta indicación es útil en la implementación. Cualquier método que implemente la operación debe satisfacer la restricción.

La implementación de la operación se puede realizar como conducta de aplicación o como conducta de servidor. Las operaciones se convierten en conducta de aplicación al poner los métodos en las clases que son parte del modelo dominio de la aplicación, así la conducta está asociada con objetos persistentes y se ejecuta como parte de la aplicación. La conducta de servidor es conducta asociada con objetos persistentes que se ejecuta en el servidor de base de datos. Hay tres tipos de conducta de servidor: métodos, procedimientos almacenados y triggers, siendo el método el valor por omisión. El estereotipo <<señal>> indica que la operación se implementará como un trigger. Para indicar conducta de servidor se utiliza una extensión de UML: {format}; por ejemplo

{format = procedimiento_almacenado} ObtieneNombre

significa que ObtieneNombre se implementará como procedimiento almacenado.

También se le puede dar a una operación la propiedad {abstracta}, esto puede señalarse italizando el nombre de la operación. Una operación abstracta no tiene un método correspondiente. Las operaciones abstractas se utilizan para mostrar las interfaces de clases abstractas cuyas subclases proporcionan la implementación. El polimorfismo y ligado dinámico resuelven la llamada al método subclase apropiado en tiempo de ejecución.

Una interfaz representa una colección abstracta de operaciones las cuales son siempre públicas. Las interfaces se pueden especificar en cualquier lugar en donde se pueda especificar una clase. Esta abstracción permite definir paquetes con interfaces que abstraen la estructura interna del paquete como un conjunto de operaciones. También permite utilizar la misma interfaz en diferentes clases y paquetes. Una clase puede tener una o más interfaces, y las interfaces pueden agrupar operaciones de varias clases diferentes.

Una interfaz se representa gráficamente en forma similar a una clase, pero en la división superior se utiliza el estereotipo <<Interfaz>>. Además una interfaz no tiene atributos, sólo operaciones, y como todo es abstracto en ella, no es necesario utilizar itálicas para distinguir los elementos abstractos. A continuación se muestra la interfaz Persona:

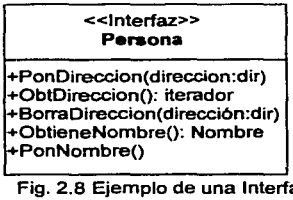


Fig. 2.8 Ejemplo de una Interfaz

Gráficamente hablando, para conectar una interfaz a un subsistema o clase, se utiliza una línea con un pequeño círculo al final y con el nombre de la interfaz como etiqueta:

Fig. 2.9 Conexión de una interfaz a un subsistema

RELACIONES ENTRE CLASES

UML tiene dos tipos de relación (conexión entre dos o más clases no necesariamente distintas) para el modelado de datos: Asociación y Generalización.

Asociación

En UML una asociación es una relación semántica entre dos o más clasificadores que implica conexiones entre sus ejemplares [2.7]. Las asociaciones muestran cómo se relaciona un objeto de una clase a otros objetos. La principal decisión que se toma cuando se crea una asociación es determinar si representa una referencia a un objeto independiente o si es una liga de propiedad.

Una asociación binaria es una asociación entre dos clases. Se pueden tener asociaciones entre más de dos clases. Gráficamente, la asociación binaria se representa como una línea conectando dos clases. La asociación ternaria y posteriores, son una serie de líneas conectando varias clases a través de un diamante central que representa la asociación.

Los nombres de las asociaciones generalmente son verbos. Frecuentemente se tienen asociaciones que sólo se nombran con "tiene", indicando que una clase tiene ejemplares de otra; en estos casos se deja la asociación sin etiqueta. Cada lado de la asociación es un rol, el papel jugado por la clase en la relación. Una asociación binaria tiene dos roles, una ternaria tres y así sucesivamente. Cada rol tiene un nombre opcional, en general son sustantivos correspondientes a los sujetos y objetos del hecho que la asociación

representa. Se puede especificar la visibilidad para el rol con los mismos símbolos que para los atributos.

El rol tiene una multiplicidad que es la propiedad que representa una regla de negocio o restricción sobre el número de objetos que participan en la asociación y se especifica con un rango de valores enteros:

- 0..* Cero o más objetos
- 0..1 Cero o un objeto
- 1..* Al menos un objeto
- 1 Exactamente uno
- * Cero o más objetos
- 2..6 Al menos dos pero no más de seis objetos
- 1,3,5-7 Al menos uno pero posiblemente tres, cinco, seis o siete objetos

El asterisco representa un número superior ilimitado. El rango es inclusive. La multiplicidad es fundamental en el modelo de datos porque se traduce directamente en la estructura de las reglas del negocio sobre llaves externas en el esquema lógico. Por ejemplo, si se especifica un rol 0..* esto se traduce como un atributo nulo en una llave externa en una base de datos relacional.

Agregación

La agregación es una asociación "parte-todo" o "parte de" entre dos clases. UML permite especificar dos tipos: agregación compartida o débil y agregación por composición o fuerte. La agregación compartida permite modelar una relación "parte-todo" en la cual un objeto es parte de otro objeto, pero su existencia no está supeditada a la existencia del otro. La agregación por composición permite modelar la relación "parte-todo" en la cual un objeto es dueño exclusivo de otro objeto.

La agregación compartida se representa gráficamente con un pequeño diamante vacío al final de la línea de asociación que llega a la clase propietaria. La agregación por composición se representa de la misma forma pero con el diamante relleno. Un ejemplo de agregación compartida: una foto puede contener un conjunto de personas, pero una persona dada puede estar en más de una foto, semánticamente no hay propiedad: la foto no es dueña de la persona, por lo que al borrar la foto no se elimina la persona de la base de datos. En cuanto a la agregación por composición: una persona puede tener uno o más documentos de identificación, y la persona es dueña de esos documentos; si la persona es eliminada de la base de datos, los documentos deben eliminarse también. Una agregación por composición debe tener una multiplicidad de 1 ó 0..1 en el rol correspondiente al dueño.

Fig. 2.10 Agregación por composición y Agregación compartida

Generalización
Una relación de generalización es una relación de orden entre un elemento general y uno más específico. El elemento específico es completamente consistente con el general y contiene información adicional. Se puede utilizar un ejemplar del elemento específico en donde esté permitido utilizar el elemento general. Las relaciones de generalización entre clases definen la jerarquía de herencia [2.8].

Por ejemplo, considerando los documentos que se tienen para identificar a una persona: acta de nacimiento, credencial de elector, licencia, pasaporte todos ellos se pueden conceptuar como una **Identificacion**, y los que tienen cierto tiempo de vigencia a su vez se pueden agrupar como **IdConExpiracion**, el diagrama para estas relaciones entre clases se muestra en la Fig. 2.11

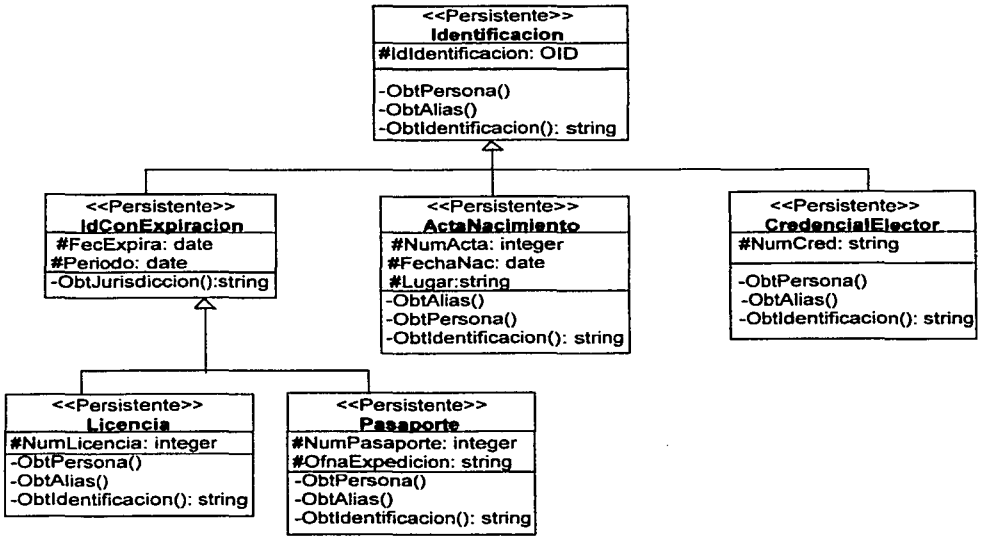


Fig.2.11 Ejemplo de Generalización

Las flechas con cabeza blanca indican las relaciones de generalización. La flecha apunta a la clase más general, llamada superclase o clase base. La flecha sale de las clases más especializadas, llamadas subclases o clases derivadas. Una clase hoja no tiene clases más especializadas. Una clase raíz no tiene clases más generales.

En UML las superclases que tienen varias subclases como **Identificacion** o **IdConExpiracion** pueden tener un discriminador indicado. Este discriminador llega a ser un atributo de la superclase que identifica la subclase, aunque no se especifique en la lista de atributos, sólo como etiqueta de la generalización.

El abstraer la estructura y conducta comunes de un conjunto de cosas diferentes en una clase más general es muy útil. Lo importante es que las subclases contienen los atributos y operaciones definidas para su superclase, esta característica es conocida como herencia por lo que se dice que las subclases heredan todo lo que tiene su superclase. La herencia es uno de los mecanismos más importantes para asegurar la reutilización de componentes.

Las subclases extienden a su superclase adicionando atributos, operaciones o asociaciones . Pero también pueden restringirla anulando conducta de su superclase con polimorfismo u operaciones virtuales. El polimorfismo es utilizar el mismo nombre para diferentes operaciones. Hay dos tipos básicos de polimofismo:

- Sobrecarga (overloading): operaciones con el mismo nombre pero con diferente firma.
- Sobreescritura (overriding): operaciones que se definen en la subclase, las cuales restringen o cambian la conducta pero tienen la misma firma.

El ejemplo más común de sobrecarga es la operación suma, que puede realizarse ya sea con enteros, números de punto flotante y hasta con cadenas. Con respecto a la sobreescritura, en el ejemplo anterior las clases abstractas **Identificacion** e **IdConExpiracion**, no tienen métodos implementados, y las clases concretas anulan las operaciones de sus superclases proporcionando implementaciones específicas.

El polimorfismo proporciona la habilidad de cambiar la conducta de una subclase, por lo que es mejor que las operaciones sobrecargadas o sobreescritas tengan semánticas similares en las diferentes clases.

Las relaciones de generalización dan significado a las clases abstractas, ya que a través de éstas se establecen las propiedades comunes de sus subclases concretas, evitando repetirlas en cada subclase. Al traducir el diseño al esquema de base de datos, éste será más flexible al disminuir la redundancia. Aunque se aumente el número de tablas en el esquema relacional, se tendrán menos problemas de normalización y una mejor administración y manejo de tablas.

Las interfaces soportan una relación UML especial: la relación "realiza", la cual es un tipo de relación de generalización que no representa herencia. Cuando se asocia una interfaz con una clase, se dice que la clase realiza o implementa la interfaz. La clase puede actuar como un subtipo de la interfaz. Se pueden instanciar objetos de la clase, asignarlos a variables de tipo interfaz y usarlos a través de esa interfaz. Esto es similar a lo que se hace con la superclase, pero se tiene acceso sólo a la interfaz, no a todas las operaciones sobre la superclase.

Las interfaces no son la forma más usada para indicar subtipos. Como regla, cuando se tienen atributos con mucho significado como propiedades comunes se debe usar la generalización, cuando se tienen sólo operaciones se debe usar interfaces.

En el modelo de datos las interfaces son útiles como una forma de expresar la relación realización, pero la traducción a los modelos lógico y físico requiere de bastante trabajo.

Clases Asociación
Algunas asociaciones contienen información no sólo acerca de las clases relacionadas, sino de la asociación en sí misma y para adicionar esa información se crea una clase asociación. Una clase asociación actúa como asociación pero tiene sus propios atributos y operaciones. En UML las clases asociación se representan como una clase ligada a la asociación con una línea punteada. Esta clase tiene el nombre de la asociación. Un ejemplo de esta clase sería el mostrado por la Fig. 2.12 que indica que la persona juega un rol, pero la asociación Juega tiene datos propios que deben almacenarse como son: fecha de inicio y fecha de fin.

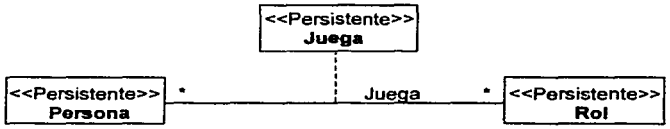


Fig. 2.12 Ejemplo de una clase asociación

La generalización y la asociación proporcionan muchas de las restricciones que requiere el modelo de datos para generar un esquema conceptual completo.

Restricciones de Identidad y Unicidad
Todo modelo de datos debe contener una manera de identificar de forma única cada objeto en la base de datos. La mayoria de los SABD proporcionan identificadores únicos, propios y ocultos para renglones y objetos, este tipo de identificación es llamada identificación implícita o basada en existencia. La alternativa es la identificación explícita o basada en valores, ya que se da a través del valor de uno o más atributos del objeto.

La identidad de objeto corresponde al concepto de llave primaria en el modelado ER. UML no proporciona ninguna forma de indicar la identidad de objeto porque asume que es una propiedad básica, implicita y automática de un objeto. Por lo tanto, para expresar el concepto en el modelado de datos se puede utilizar una extensión. Por ejemplo, en un diagrama de clase se pueden utilizar las propiedades {OID} y {OID alterno} para indicar explícitamente la identidad de objeto. El anexar la etiqueta {OID} a un atributo significa que el atributo es parte de la identidad de objeto; la etiqueta {OID alterno} indica que el atributo es parte de una llave alternativa. En SQL esto corresponde a la llave primaria y a la restricción de unicidad, respectivamente.

Sólo debe especificarse la propiedad {OID} a atributos de clases regulares, no de clases asociación. La identidad de objeto de una clase asociación siempre está implícita y consiste de la combinación de los atributos con propiedad {OID} de las clases que participan en la asociación.

Cuando se requiere hacer una identidad explícita, usualmente se adiciona un atributo con un tipo adecuado que sirva como identidad de objeto. En bases relacionales esto corresponde a un identificador que se genera tal como un tipo secuence en ORACLE. Es importante notar que al crear un identificador de objeto en una tabla, su alcance es la tabla, no la clase, por lo que si se definen varias tablas para una clase, el identificador sólo identificará de manera única los objetos en cada tabla.

En una jerarquía de generalización, la propiedad {OID} se especifica sólo en la cima de la jerarquía, pues las subclases heredan la identidad de objeto explícita de esa clase. Se pueden especificar atributos con {OID alterno} en las partes inferiores de la jerarquía.

Una clase relacionada con otra a través de agregación de composición obtiene parte de su identidad de objeto de la clase a la cual compone, y es necesario especificar atributos adicionales para identificar elementos individuales dentro del agregado.

El hacer explícita la identidad de objeto tiene la ventaja de conservar todos los atributos claros y abiertos en el diseño, haciendo más directo el traslado del modelo de datos al esquema conceptual. Por su parte la identidad implícita, esconde los detalles y permite hacer el trabajo cuando se convierte el modelo de datos al esquema conceptual, sin embargo para SABD relacionales esto requiere de mucho trabajo adicional.

Restricciones Complejas
UML proporciona un lenguaje formal para expresar las reglas del negocio que van más allá de las restricciones de integridad y de dominio: el Lenguaje de Restricciones Objeto (OCL por sus siglas en inglés). Sin embargo está permitido utilizar cualquier lenguaje como SQL, OQL, Java, Smalltalk, etc., siempre y cuando tenga sentido de acuerdo a los requerimientos. Las expresiones OCL o sustitutas se pueden utilizar en los diagramas UML en:

- Clases y tipos: para restricciones sobre todos los ejemplares de la clase o tipo.
- Operaciones: para indicar pre / poscondiciones y otras restricciones sobre la conducta de la operación
- Transiciones/Flujos: en las condiciones en diagramas de transición de estados o diagramas de actividad.

La notación UML para las restricciones es una nota ligada al objeto restringido. El siguiente diagrama muestra una restricción invariante sobre la clase **OrganizacionCriminal**:

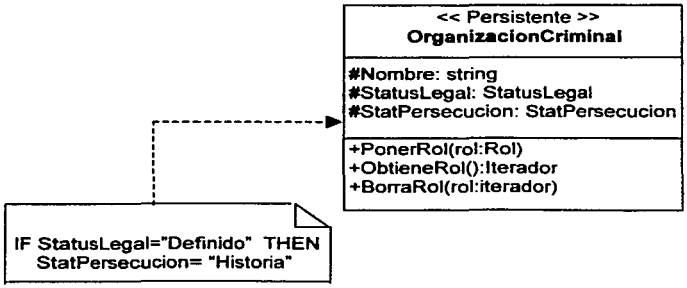


Fig 2.13 Representación gráfica de una restricción compleja

Hasta aquí se han mostrado los elementos de UML necesarios para el modelado de datos y el diseño de la base.

2.4 REUTILIZACIÓN Y EVALUACIÓN DEL DISEÑO

Uno de los objetivos principales de los modelos orientadas a objetos y por consecuencia del UML es la generación de diseños y sistemas que puedan ser reutilizables. Aunque la constante en el diseño de software es tener cambio tras cambio en la forma de hacer las cosas, los diseñadores han empezado a aceptar que existen aspectos del diseño que no deberían cambiar o deberían cambiar muy poco, lo cual genera reutilización.

Un patrón de diseño es un sistema reutilizable que describe un problema de diseño y su solución. Los patrones de diseño ayudan a establecer un contexto para la solución de un problema. Existen dos tipos de patrones de diseño de interés para los modeladores de datos: patrones abstractos y patrones de análisis. Un patrón abstracto contiene clases y relaciones describiendo una solución genérica a un problema genérico. Un patrón de análisis es un modelo de objetos concretos de dominio específico generalizado para reutilizarse en diferentes situaciones. Los patrones de diseño proporcionan una ayuda en el modelado de datos[2.9].

Para determinar si el diseño logrará que el producto final cubra los principales objetivos generados por el análisis de requerimientos los puntos principales que requieren medirse son: tamaño del modelo, complejidad del modelo, cohesión, acoplamiento y potencial de reutilización tanto del modelo como del sistema que generará, y para que la evaluación sea realista debe utilizarse la métrica apropiada.

Aunque siempre es difícil dimensionar una base de datos, aún cuando ya se tiene, en la etapa de modelado lo mejor es una métrica de magnitud de orden basada en el número de clases y el número estimado promedio de objetos persistentes por clase. Sin embargo si se desea estimar el esfuerzo, Muller propone la métrica de punto de función la cual mide la funcionalidad de un sistema de software[2.10].

No existe una medida aceptada para dimensionar la complejidad del modelo de datos, en este caso lo más útil es la intuición pues los diseñadores de bases de datos saben cuando se complican los diseños. Lo mejor es elaborar el modelo de datos tan simple como sea posible, lo cual no significa tener unas cuantas clases, sino expresar la semántica en forma clara, no importando que ello signifique mayor número de clases.

Para el diseñador de base de datos existen dos tipos de cohesión que son de su interés: la cohesión abstracta de un clasificador y la cohesión estructural de un conjunto de clasificadores relacionados. La cohesión es importante porque implica reutilización.

La cohesión abstracta de un clasificador en el modelo de datos es el grado en el cual el clasificador requiere de sus componentes individuales para mantener su abstracción. Lo que se quiere medir es cómo se integran los diferentes componentes para darle significado a la clase. Aunque no existe una forma directa de medir este aspecto, la intuición puede ayudar a estimar la cohesión de una clase. Dada una clase, cuando se tienen grupos de atributos que se relacionan a cosas diferentes, se puede intentar separar la clase para ver si los diferentes componentes pueden trabajar como clases separadas. Esto es útil en la creación de jerarquías de herencia, cuando se separan elementos individuales que aplican sólo bajo ciertas circunstancias. Un ejemplo clásico es cuando un atributo es nulo por definición si una cierta condición lógica es verdadera. Al crear una subclase y poner ese atributo en ella, frecuentemente se elimina la necesidad de hacer nulo el atributo.

La cohesión estructural es el grado en el cual la estructura de los objetos produce resultados con significado en las consultas. Una dependencia es una restricción de inferencia sobre un conjunto de atributos. Una dependencia funcional es una inferencia de un solo atributo: si se conoce el valor del atributo A entonces se conoce el valor del atributo B. Una dependencia multivaluada es una restricción de dos atributos: si se conocen los valores de A y B entonces se conoce el valor de C. De forma general una dependencia de unión es una restricción n-valuada

Para reducir la cohesión estructural y dado que las bases de datos relacionales no tienen el concepto de atributos multivaluados (arreglos, estructuras y clases anidadas), no deben utilizarse datos estructurados al modelar las clases. Los datos estructurados deben dividirse en clases separadas y utilizar asociaciones para representar la estructura multivaluada. Esta solución mapea directamente en la normalización; así se puede evaluar la cohesión estructural del modelo en :

1. Primera forma normal. Ninguna clase contiene un atributo de tipo estructurado.
2. Segunda forma normal. Ninguna clase contiene un atributo que no sea totalmente dependiente de la identidad de objeto de la clase
3. Tercera forma normal. Ninguna clase contiene una dependencia funcional que no sea de la identidad de objeto completa.
4. Cuarta forma normal. Ninguna clase contiene dependencia multivaluada que no sea de la identidad de objeto completa.
5. Quinta forma normal. Ninguna clase contiene una dependencia de unión que no sea de la identidad de objeto completa.

Cuando se utilizan lenguajes de modelado como UML, se tiende a obtener bases de datos completamente normalizadas inclusive en la quinta forma normal, porque el modelo refleja semánticamente clases y relaciones.

En términos de base de datos, el no tener acoplamiento significa que una clase dada no tiene ninguna asociación con otra clase y tampoco hereda de ninguna otra. Los métodos de la clase no se refieren a datos de otras clases en ninguna forma.
El acoplamiento de datos limita el acceso a valores persistentes (tipos de dato primitivos) encapsulándolos en métodos. Una clase muestra acoplamiento de datos cuando accede esos datos a través de la llamada al método, no por asociación o herencia.

Una clase que obtiene acceso a datos estructurados(tipos registro, arreglos u objetos) a través de los métodos muestra acoplamiento marcado. En bases de datos relacionales significa acceder a los datos a través de procedimientos almacenados que devuelven cursores o datos estructurados.

El acoplamiento de control es orientado a procedimientos, no a los datos. En el grado en que el lenguaje de base de datos soporte control procedural se puede tener acoplamiento de control en las instrucciones. Por ejemplo, en una base de datos relacional al crear una instrucción SQL parametrizada que utiliza el parámetro en una cláusula WHERE.
El acoplamiento común se refiere al uso de datos globales. Al utilizar una base de datos se están usando datos globales y por lo tanto se incrementa el nivel de acoplamiento a común para cualquier clase que refiera datos en la base. El control de visibilidad es parte del mecanismo de seguridad de la base de datos, no del mecanismo de visibilidad de programación. Con un esquema relacional sólo hay opción de acceso de datos global.

El acoplamiento de contenido se refiere directamente a datos. Las bases de datos relacionales no encapsulan los datos, por lo que SQL los refiere directamente.

El acoplamiento se relaciona fuertemente con el mantenimiento y la reutilización. Entre más baja sea la escala de acoplamiento, será más fácil mantener el sistema y se podrá reutilizar bajo diferentes escenarios. Entre menos acopladas estén las clases, el impacto de un cambio en alguna clase será menor en el resto del sistema.

En el diseño orientado a objetos hay tres formas de limitar el acoplamiento:

1. Cuidar la encapsulación. Ocultar lo más que se pueda: Cualquier objeto debería acceder directamente sólo a objetos que estén conectados a él mediante asociación o generalización. Aunque SQL es un lenguaje sin encapsulación se pueden utilizar procedimientos almacenados para encapsular tablas y utilizar vistas para limitar la exposición de datos.
2. Utilizar la herencia de manera restringida. Si una clase se refiere directamente a un miembro en su superclase, esto es acoplamiento de contenido. Cuando se cambia la superclase se necesita cambiar todas las subclases que la utilizan
3. Evitar el acoplamiento de control utilizando polimorfismo donde sea posible.

La reutilización es la habilidad de usar otra vez una base de datos fuera del sistema de aplicación que la creó y la usó. El potencial de reutilización es la probabilidad de reutilizar la base de datos. Para desarrollar bases de datos reutilizables es necesario considerar los usuarios potenciales y sus requerimientos. Aunque es imposible identificar todos los requerimientos pues siempre habrá algo nuevo, se pueden aplicar principios y patrones generales del diseño orientado a objetos que hacen adaptable y extensible un sistema:

- Parametrizar el sistema
- Permitir que los usuarios establezcan opciones a través de almacenamientos de datos persistentes.
- Hacer clases fáciles de reutilizar a través de la herencia.
- No hacer demasiadas suposiciones de lo que la gente quiere hacer.

Se requiere una política de certificación para certificar una base de datos como reutilizable. Primero es necesario establecer el nivel de riesgo del sistema lo que significa certificar que el sistema puede lograr su objetivo dentro de un cierto nivel de riesgo. Como segunda parte es necesario proporcionar una definición clara de las metas y estructura del sistema, lo cual incluye la publicación de la visión, misión y objetivos operacionales de los componentes de la base de datos. El modelo de datos y el diseño del esquema proporcionan una descripción completa de la base de datos a los usuarios potenciales futuros.

La certificación debe establecer confianza para el usuario futuro indicando la responsabilidad que el diseñador original asume y las acciones que éste realizará cuando haya problemas, lo cual se traduce en un "contrato", en donde se garantice la calidad y el desempeño de la base de datos. El último paso es el mantenimiento que se realizará sobre el sistema reutilizable pues el mantenimiento soporta las garantías.

3. CONVERSIÓN DEL MODELO DE DATOS UML A UN ESQUEMA DE BASE DE DATOS RELACIONAL

Antes de caer en la tentación de ir directamente del modelo de datos a la definición del esquema físico de la base de datos, es importante considerar algunos aspectos que impactan más de lo que se piensa en el resultado final de un proyecto.

Los diseños de bases de datos no resultan sólo del trabajo innovador del diseñador en turno, más bien son el resultado del esfuerzo de otros diseñadores, de conocer el sistema existente y de la cultura de desarrollo en la cual se llevan a cabo. La cultura es la colección de sistemas trasmitidos socialmente que establecen la forma de hacer las cosas y es uno de los elementos más útiles al realizar un proyecto, siendo imposible ignorarla sin tener problemas.

Existen cinco puntos que establecen la cultura de una organización de desarrollo de software:

1. Normas: Estándares y políticas compartidas que la organización utiliza para establecer los límites de lo que está permitido y lo que no.
2. Valores, actitudes y creencias. Los valores son los principios sociales de la organización. Las actitudes son las formas en que la gente piensa acerca de hechos en la organización. Las creencias son las cosas que se consideran verdaderas acerca de la organización o del ambiente en la cual se encuentra.
3. Rituales: Conductas sociales repetidas que establecen la rutina que debe seguirse
4. Folklore: Conjunto de historias y símbolos acerca de la organización.
5. Lenguaje común: Lenguaje especial que desarrolla la organización para describir cosas de interés a sus miembros.

La cultura es aún más importante cuando se realiza un nuevo diseño mientras se sigue utilizando el sistema anterior, y sobretodo si aún proporciona valor al negocio. Una parte esencial es la decisión de renovar el sistema existente o hacer uno nuevo. Si la decisión es renovarlo, se requiere conocer más acerca de la cultura para poder entender tanto el alcance del sistema existente como el alcance de los cambios a realizarse.

El realizar un nuevo sistema no significa ignorar lo que se hizo antes, pues el sistema anterior constituye un bien intelectual para la institución y es un buen comienzo para conocer las necesidades de la organización. Tampoco es correcto suponer que el sistema anterior expresa los requerimientos de los usuarios. Es necesario desarrollar los alcances del nuevo sistema basándose tanto en los requerimientos del sistema como en la cultura.

Independientemente de la elección que se haga en cuanto a renovar el sistema o desarrollar uno nuevo, la transición del modelo de datos a un esquema es un proceso crítico en el diseño de la base de datos.

3.1 CONVERSIÓN DEL MODELO DE DATOS

La estructura de una base de datos relacional es muy simple, pero esta simplicidad dificulta el proceso de representar objetos complejos. El objetivo de utilizar un modelo de datos UML para después trasladarse a un esquema relacional es que la simplicidad de la tabla relacional trabaje a favor del desarrollador.

El paquete, y su especialización, el subsistema proporcionan espacios nombrados organizados para la arquitectura del sistema. Los paquetes son un concepto nuevo en las bases de datos relacionales. Cuando se diseña un sistema por medio de subsistemas, se busca obtener un sistema en partes reutilizables que son independientes entre sí tanto como sea posible.

Muchos diseñadores consideran los esquemas relacionales como un depósito global de datos, debido a las limitaciones tecnológicas que tenían anteriormente y a la cultura que se creó por tales limitaciones. Pero al utilizar paquetes para generar reutilización a través del encapsulamiento este problema se reduce pues los nombres de espacios se usan para organizar cuidadosamente el acceso.

El SQL estándar proporciona una capacidad básica para nombrar espacios con el nombre del esquema. Un esquema ANSI es un descriptor que incluye un nombre, un identificador de autorización y todos los descriptores de un conjunto de componentes (tablas, vistas, dominios, aserciones, privilegios, conjuntos de caracteres, comparaciones o traslaciones). Ningún SABD relacional implementa de manera completa todas las características del estándar; sin embargo, es posible establecer espacios nombrados de subsistemas y paquetes separados haciendo uso de diferentes aspectos del SABD que se utilice, y aunque no se obtiene una representación completa del paquete, se logra más encapsulación que con una implementación usual.

CONVERSIÓN DE CLASES.
Después de establecer los nombres de espacios, lo siguiente es la transformación de las clases. Cada clase se convierte en una tabla en la base de datos relacional. Se puede utilizar el mismo nombre de la clase para la tabla. Es importante tener presente la restricción del SABD utilizado en cuanto a la longitud del nombre para tablas y columnas.

La sintaxis UML para la definición de un atributo es:

estereotipo visibilidad nombre : tipo = valor inicial { propiedad }

Cada atributo se convierte en una columna en la tabla. En manejadores relacionales no hay estereotipos para columnas y la visibilidad siempre es pública, así que esta parte se puede ignorar. En cuanto al tipo, a pesar de que el estándar ANSI SQL-92 incluyó el concepto de dominio como un conjunto de valores permisibles, ninguno de los principales SABD relacionales implementa la instrucción CREATE DOMAIN. Pero, aún cuando se tuviera, es necesario tener una forma de relacionar los tipos UML con los tipos de dato SQL que maneja el SABD a utilizar. Antes de convertir las clases, es útil tener una transformación que indique el mapeo de los tipos UML a los tipos SQL del SABD. En caso de utilizar varios SABD se debería hacer una transformación para cada uno de ellos. Debe tratarse de que las transformaciones sean simples, por ejemplo un tipo cadena en UML podría traducirse como un VARCHAR(254) en SQL. Para tipos enumerados, y sobre todo si no se tienen dominios, se puede crear una tabla para el tipo y poner los valores en ella, pues de esta forma se podrían adicionar nuevos valores cuando se requiera.

Los manejadores relacionales dependen completamente de identidad explícita, por lo que no puede existir un identificador de objeto que no sea un valor de una columna de la tabla, ni puede haber apuntadores a valores o renglones. Transformar la identidad explícita de atributos marcados con la propiedad {OID} significa poner los atributos como columnas y

establecer la restricción PRIMARY KEY sobre tales columnas. La transformación de la identidad implícita a identidad relacional explícita es un poco más difícil pues las herramientas difieren dependiendo del SABD. El método más simple es adicionar una columna de tipo entero a la tabla. Para los atributos marcados con {OID alterno}, sólo es necesario crear el (los) atributo(s) como columna(s) y poner restricciones de unicidad. Si el atributo no tiene la propiedad {nulo} significa que se debe poner NOT NULL en la definición de la columna correspondiente.

En la instrucción CREATE TABLE no hay nada con respecto a las operaciones, sin embargo la mayoría de los SABD relacionales tienen algunas posibilidades para representar conducta en el esquema a través de procedimientos almacenados.

Es importante darse cuenta de la diferencia entre el servidor de base de datos y el código de aplicación. Particionar una aplicación es el proceso de decidir entre poner conducta en el servidor de base de datos, en el servidor de aplicaciones o en el cliente. No es conveniente tener en el servidor de base de datos operaciones que requieren una referencia a una dirección u objeto en memoria, ni operaciones que acceden los datos atributo por atributo. La verificación extensiva de errores en memoria debe dejarse al cliente o al servidor de aplicaciones. El único manejo de errores que debe realizar el servidor de base de datos es el relacionado con la estructura de la base de datos (verificación de reglas del negocio, errores de almacenamiento físico, etc.). Esto aplica especialmente a código que pertenece a un trigger, dado que este se ejecuta cuando sucede un evento indicado en la base de datos.

Las operaciones que se recomienda poner en el servidor de base de datos son:

- Consultas que regresan un conjunto de valores.
- Inserciones, actualizaciones o borrado de datos para un renglón no para un único valor.
- Verificación de reglas.
- Operaciones encapsuladas. Operaciones que llaman a otras operaciones de base de datos, tales como procedimientos almacenados proporcionados por el vendedor.
- Verificación de errores. Operaciones que verifican diferentes estados de error y excepciones.

Las transacciones generalmente suceden en la aplicación, no en el servidor de base de datos, por lo que las instrucciones de transacción generalmente no se ponen en los procedimientos almacenados.

Se pueden utilizar procedimientos almacenados y subsistemas para encapsular los objetos. Por ejemplo, se pueden crear las tablas básicas y los subsistemas en un esquema que pertenezca a un usuario específico, "el dueño" de dicho esquema y otorgar los privilegios apropiados a los usuarios que necesitan acceder los objetos de la clase, así cuando se accede a la aplicación con un usuario permitido, el código llama a los procedimientos en lugar de utilizar SQL para consultar o manipular las tablas.

La mayoría de los SABD relacionales no soportan polimorfismo. El ligado dinámico, la sobreescritura (overriding) y las operaciones virtuales no existen en el mundo relacional, por lo que Muller [3.1] aconseja renombrar, con nombres únicos, las operaciones de sobreescritura y llamarlas en los lugares apropiados, pues aunque esto signifique más

código condicional para determinar qué operación llamar, es mejor a tener problemas con el mantenimiento de un código complicado que simule el polimorfismo.

Si la operación UML tiene un estereotipo <<señal>> entonces corresponde a un trigger. Al transformar la operación <<señal>> en un trigger de base de datos, es necesario utilizar el lenguaje disponible en el SABD o un lenguaje de programación. Es importante utilizar la documentación que tenga el SABD en relación con el manejo de triggers para determinar la manera más conveniente de realizar la transformación.

CONVERSIÓN DE INTERFACES.
Ya que una interfaz sólo contiene operaciones únicamente es necesario implementar los procedimientos almacenados correspondientes, siempre y cuando sean adecuados para ejecutarse en el servidor de base de datos.

CONVERSIÓN DE ASOCIACIONES.
Una base de datos relacional contiene tablas, no asociaciones, por lo que las asociaciones deben integrarse a las tablas a través de columnas especiales. En la conversión de una asociación binaria a un esquema relacional hay un concepto esencial: la llave externa. Cada asociación binaria se convierte en una restricción REFERENCES o FOREING KEY en la definición de la tabla, pero lo importante es la creación de las columnas llave externa sobre las cuales se pondrán esas restricciones. Esta es la forma de representar las asociaciones y su semántica en el modelo relacional.

El modelo UML no especifica atributos llave externa, pues están representados como asociaciones binarias. Estas asociaciones tienen dos roles y dos multiplicidades las cuales controlan la transformación que se debe realizar para adicionar columnas como llave foránea. Para determinar la estructura de la transformación se debe considerar las multiplicidades en forma individual y combinando ambos roles. Las características de las multiplicidades se dividen en dos aspectos de interés para la transformación relacional: en qué tabla generar columnas y cómo generar las restricciones de nulidad sobre las columnas.

Si la multiplicidad contiene un máximo de 1 entonces ese rol corresponde a una columna o columnas llave externa a crear en la tabla ligada al otro rol. Hay algunas variantes, por ejemplo cuando la llave primaria de una tabla depende de la llave primaria de otra. Por ejemplo, la clase **Persona** tiene una asociación binaria con **Identificacion** (ver Fig. 2.10), pero esta asociación es de agregación por composición, pues una persona es dueña de sus identificaciones. La llave primaria de la tabla **Persona** es **IdPersona** por lo que esta columna debe crearse en la tabla **Identificacion** como una llave externa a la tabla **Persona**.

```
CREATE TABLE Identificacion (
    IdPersona Integer REFERENCES Persona,
    IdIdentificacion INTEGER,
  CONSTRAINT Idenficacion_PK PRIMARY KEY (IdPersona, IdIdentificacion)
```

Si un objeto identificación fuera independiente de la persona, sería suficiente con **IdIdentificacion** como llave primaria. Si cada identificación perteneciera a una sola persona en el tiempo, el rol persona tendría una multiplicidad de 0..1, por lo que **Identificacion** tendría una columna **IdPersona** que no sería parte de la llave primaria:

```
CREATE TABLE Identificacion (
    IdPersona Integer REFERENCES Persona,
    IdIdentificacion INTEGER PRIMARY KEY)
```

Si la multiplicidad contiene un cero o lo implica (0..*, *, 0..1) entonces se pueden tener nulos en la llave externa.

En las tablas los roles con una multiplicidad máxima de 1 pueden llegar a ser el nombre de la llave externa. El utilizar el nombre del rol o el nombre de la llave primaria para nombrar la llave externa depende del criterio propio del diseñador en cuanto a querer dar significado a la asociación. Hay un caso particular en el cual, al hacer la transformación al esquema relacional, es imposible utilizar el nombre de la llave primaria para la llave externa: una asociación de agregación por composición recursiva, como la que se da con la clase **Organizacion**:

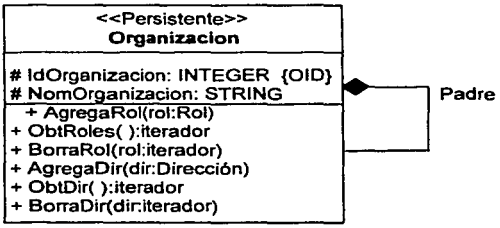


Fig. 3.1 Agregación por Composición Recursiva

La transformación directa de esta asociación a una tabla sería:

```
CREATE TABLE Organizacion(
    IdOrganizacion INTEGER PRIMARY KEY,
    NomOrganizacion VARCHAR(100) NOT NULL,
    IdOrganizacion INTEGER REFERENCES Organizacion)
```

El principal problema es que en una tabla no se pueden tener nombres duplicados para las columnas, además **IdOrganizacion** como nombre de la llave externa, no proporciona el significado adecuado para el usuario. Una solución es renombrar la llave externa con el nombre del rol:

```
CREATE TABLE Organizacion(
    IdOrganizacion INTEGER PRIMARY KEY,
    NomOrganizacion VARCHAR(100) NOT NULL,
    Padre INTEGER REFERENCES Organizacion)
```

Cuando se tiene una asociación calificada, que es la combinación de un tipo de dato enumerado y una asociación, significando que el objeto calificado tiene una liga para cada valor del tipo, el convertirla al esquema relacional requiere de una llave externa y de una columna calificador que especifica el tipo de objeto. La Fig. 3.2 muestra la asociación calificada **Identificación-Persona**, indicando que una persona tiene diferentes tipos de **IdDocumento**.

Fig. 3.2 Asociación calificada

```
CREATE TABLE Identificacion (
    IdPersona Integer REFERENCES Persona,
    IdIdentificacion INTEGER,
    IdDocumento INTEGER NOT NULL REFERENCES Documento
    CONSTRAINT Idenficacion PK PRIMARY KEY (IdPersona, IdIdentificacion)
```

Si la clase calificada tiene requerimientos específicos, es necesario adicionar restricciones o triggers que los establezcan. Por ejemplo, si una persona debe tener un acta de nacimiento y un pasaporte, se necesitará un trigger que verifique la existencia de dos renglones en **Identificacion**: uno con el **IdDocumento** correspondiente a acta de nacimiento y otro con el **IdDocumento** correspondiente a pasaporte para la misma persona.

Otra forma de transformar una asociación calificada es agregando columnas paralelas en la tabla calificada. Utilizando el mismo ejemplo, se añadirían las columnas **ActaNacimiento** y **Pasaporte** a la tabla **Persona**, las cuales serían llaves externas a las tablas **ActaNacimiento** y **Pasaporte** respectivamente.

CONVERSIÓN DE GENERALIZACIONES

Existen dos formas para realizar la conversión de una generalización al modelo relacional:

1. **Mapeo directo** : Se crea una tabla para cada clase en la jerarquía de herencia, incluyendo las clases abstractas; después se crean las llaves externas para cada relación de generalización. Si la llave primaria de la superclase consta de varias columnas, se deben crear esas mismas columnas en las subclases. Tomando como ejemplo la jerarquía de herencia de una **Identificacion** mostrada en la Fig 2.11 la tabla para ActaNacimiento se crearía así:

```
CREATE TABLE ActaNacimiento(
    IdPersona INTEGER NOT NULL REFERENCES Identificacion
    IdIdentificacion INTEGER NOT NULL REFERENCES Identificación,
    NumActa INTEGER NOT NULL UNIQUE,
    FecNac DATE
   CONSTRAINT ActaNacimiento_PK PRIMARY KEY (IdPersona, IdIdentificacion)
```

Como la clase **Identificacion** tiene una asociación de agregación por composición con la clase **Persona**, la llave primaria contiene a **IdPersona** que es la llave primaria de **Persona**. La restricción de llave externa sobre **IdPersona** e **IdIdentificacion** representa parcialmente la relación de generalización, más no completamente pues si se inserta un renglón en **ActaNacimiento**, el **IdPersona** y el **IdIdentificacion** de ese renglón debe corresponder a un renglón en **Identificacion**; pero si se inserta un renglón en **Identificacion** no hay nada que ligue el **IdIdentificacion** de ese renglón con alguna tabla subclase. Esto sucede cuando la clase padre es una clase abstracta. Para reforzar la abstracción de una clase es necesario un proceso para asegurar que al insertar un renglón a su tabla correspondiente, también se inserte un renglón en cualquiera de las subclases de esa clase abstracta. Esto podría realizarse mediante un trigger aunque para bases de datos pequeñas Muller recomienda hacerlo al nivel de la aplicación y no en el servidor de base de

datos[3.2]. La idea principal en la generalización es la herencia y para representarla en esta forma de conversión es necesario realizar operaciones de unión y join.

2. **Extensión (spreading).** En este caso, se crean los atributos y operaciones de la superclase en las subclases para enfrentar la herencia. La ventaja es que no es necesario referirse a la superclase para obtener valores o verificar existencia, eliminando así triggers y operaciones de unión y join lo cual reduce la complejidad tanto en el esquema como en el código de aplicación. Pero se adquiere redundancia en los datos debida a la denormalización del diseño del esquema de la base, lo cual puede generar inserciones, actualizaciones o borrados anormales.
   La transformación de la clase **Pasaporte** mediante este método sería:

```
CREATE TABLE Pasaporte (
    IdPersona INTEGER NOT NULL REFERENCES Identificacion,
    IdIdentificacion INTEGER NOT NULL REFERENCES Identificacion,
    FecExpiracion DATE NOT NULL CHECK (FecExpiracion > FecEmision),
    FecEmision DATE NOT NULL,
    NumPasaporte INTEGER NOT NULL UNIQUE,
    OficEmisora VARCHAR(100) NOT NULL,
    CONSTRAINT Pasaporte_PK PRIMARY KEY (IdPersona, IdIdentificacion))
```

   En lugar de referirse a la tabla **IdenConExpiracion** para obtener **FecExpiracion** y **FecEmision** este diseño copia esas columnas en la tabla **Pasaporte** (y en todas las subclases de **IdenConExpiracion**). De esta manera, **Pasaporte** tiene todos los datos disponibles y cuando se requiera información acerca del pasaporte sólo será necesario acceder a una tabla. Sin embargo cuando se actualice **FecExpiracion** o **FecEmision** de un pasaporte se deben actualizar las mismas columnas en el renglón de la superclase que corresponde al pasaporte. Esto implica un trigger o código adicional en la aplicación.

La decisión sobre qué método utilizar debe hacerse considerando el impacto sobre la calidad del sistema total. Muller recomienda utilizar la extensión cuando se tienen jerarquías de varios niveles y el mapeo directo cuando la jerarquía tiene uno o dos niveles, así como documentar cuidadosamente el método utilizado para evitar confusiones[3.3].

CONVERSIÓN DE ASOCIACIONES TERNARIAS Y DE CARDINALIDAD SUPERIOR

Cuando se tienen una asociación con cardinalidad de tres o superior se debe crear una tabla separada para la asociación. Considerando el ejemplo mostrado en la Fig. 3.3, la asociación Juega es un diamante que une a **Rol**, **Persona** y **Organizacion**. La **Persona** juega un **Rol** en la **Organizacion**. Esto se convierte en la tabla **Juega** la cual consiste de las columnas que son llave primaria en las tablas asociadas:

```
CREATE TABLE Juega(
    IdPersona INTEGER REFERENCES Persona,
    IdOrganizacion INTEGER REFERENCES Organización,
    IdRol INTEGER REFERENCES Rol
    CONSTRAINT Juega_PK  PRIMARY KEY (IdPersona, IdOrganizacion, IdRol))
```

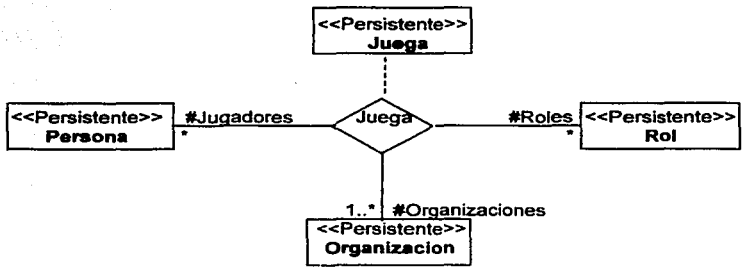


Fig. 3.3 Ejemplo de asociación ternaria

CONVERSIÓN DE CLASES ASOCIACIÓN

Cualquier asociación puede tener una clase asociación (association class) y atributos propios de la asociación en sí misma. Una clase asociación se transforma en una tabla. En el ejemplo anterior la clase **Juega** está conectada al diamante de la asociación Juega. Esta clase **Juega** contiene atributos de la asociación con los cuales se extiende la tabla **Juega** :

```
CREATE TABLE Juega(
    IdPersona INTEGER REFERENCES Persona,
    IdOrganizacion INTEGER REFERENCES Organizacion,
    IdRol INTEGER REFERENCES Rol
    Permanencia INVERVAL DAY NOT NULL,
    FechaInicio DATE NOT NULL,
    FechaFin DATE NOT NULL,
    FinRelacion INTEGER REFERENCES FinRelacion
        CHECK (FinRelacion IS NOT NULL OR
                (FinRelacion IS NULL AND FechaFin IS NULL))
  CONSTRAINT Juega_PK  PRIMARY KEY (IdPersona, IdOrganizacion, IdRol))
```

CONVERSIÓN DE AGREGACIONES POR COMPOSICIÓN

La composición establece que el objeto compuesto es dueño de otro objeto y que ningún otro objeto puede ligarse a él. Esta forma de agregación corresponde directamente a una llave externa en la tabla dependiente con acciones de actualización y borrado. Por ejemplo, cuando se borra un renglón en la tabla dueña, deben borrarse los renglones asociados a su llave primaria en todas las tablas dependientes, esto es un borrado en cascada.

El valor de una llave primaria no debería cambiar, pero en el caso de que sea necesario modificarlo, es en la agregación por composición dónde se debe tener mayor cuidado para no generar problemas de integridad.

La clase **Pasaporte** se transformaría a tabla como sigue:

```
CREATE TABLE Pasaporte(
    IdPersona  INTEGER NOT NULL REFERENCES Identificacion
      ON DELETE CASCADE  ON UPDATE CASCADE,
    IdIdentificacion INTEGER NOT NULL REFERENCES Identificacion
      ON DELETE CASCADE ON UPDATE CASCADE,
    NumPasaporte INTEGER NOT NULL UNIQUE,
    OfiEmisora CHAR(100) NOT NULL,
    CONSTRAINT Pasaporte_PK  PRIMARY KEY(IdPersona, IdIdentificacion))
```

Como la llave externa es parte de la llave primaria de la tabla dependiente, la acción cascada puede derivarse a otras tablas que tienen llaves externas a esa llave foránea. Es posible el utilizar triggers para mantener la integridad en lugar de ocupar la sintaxis expuesta, pero es muy importante no descuidar este aspecto pues se podrían generar errores en transacciones que impliquen actualización o borrado en tablas que tienen agregaciones por composición.

CONVERSIÓN DE LAS RESTRICCIONES COMPLEJAS

Usualmente, una restricción sobre una clase es una expresión que afirma una invariante de clase (class invariant) sobre los atributos de la clase. Una invariante de clase es una expresión lógica que un objeto de la clase debe satisfacer siempre, es decir, la validez de la expresión no varía con el estado del objeto.

Al transformar la clase en tabla se requiere expresar las invariantes de clase en el esquema. Existen dos opciones: restricciones de verificación (CHECK) y triggers.

Una restricción de verificación permite ligar una expresión SQL a una tabla o columna. Se debe asegurar el manejo correcto de nulos, ya que las expresiones SQL pueden evaluar a verdadero, falso o nulo.
Existen algunas restricciones en SQL sobre la condición:

- No se pueden utilizar funciones de agregación, a menos que se indique en una subconsulta.
- No se puede referir a información que cambia dinámicamente, como una sesión específica, y por consiguiente no se pueden utilizar funciones definidas por el usuario.

Hay dos formas de utilizar las restricciones de verificación: a nivel columna o a nivel tabla. En cualquier caso la restricción es evaluada cuando la instrucción SQL se completa. Cuando una restricción se refiere a dos o más tablas no se puede expresar este tipo de restricción en una restricción de verificación porque no es una invariante de clase sobre una única tabla, por lo que es necesario utilizar procedimientos almacenados o código de aplicación. Dadas las limitaciones de las restricciones de verificación, es muy común el tener que construir las restricciones invariantes de clase como triggers.

Cuando se define el sistema a través de varios esquemas, algunas veces se tienen restricciones que se extienden de un esquema a otro. El tipo más común es la restricción de llave externa que hace un subsistema dependiente de otro. Una invariante de sistema es una expresión lógica que es verdadera para el sistema como un todo.

RESUMEN DE LA CONVERSIÓN DEL MODELO DE DATOS UML AL ESQUEMA RELACIONAL

Paso	Elemento UML	En Base de Datos Relacional
1	Clase	Tabla
2	Atributo	Columna
3	Tipo de atributo	Tipo de dato
4	Propiedad {nullable} en el atributo	La columna acepta nulos
5	Atributo con valor inicial	Columna con cláusula DEFAULT
6	Atributo con indicación {OID alterno}	Adicionar esas columnas a la restricción de unicidad
7	Clases sin generalización con identidad implícita	Crear una columna para llave primaria
8	Subclases	Agregar la llave primaria de cada clase padre a la restricción de llave primaria y llave externa
9	Clases asociación	Hacer tabla y agregar la llave primaria de cada tabla que participe a la restricción de llave primaria y llave externa.
10	Restricciones complejas	Agregar CHECK para cada restricción explícita
11	Asociaciones con multiplicidad 0..1, 1.	Crear columnas llave externa en la tabla referenciada
12	Agregación por composición	Crear la llave primaria de la tabla dueña como llave externa en la tabla dependiente (con opción en cascada). Agregar columna adicional para la llave primaria.
14	Asociaciones muchos a muchos y asociaciones ternarias sin clases asociación.	Tabla con restricciones de llave primaria y llave externa de las tablas que participan

Tabla 3.1 Conversión del Modelo de Datos UML al Esquema Relacional

Lo importante en la conversión del modelo de datos UML al esquema relacional es ser muy sistemático en el método. Si se entiende como mapear cada concepto UML en conceptos relacionales es posible representar fácilmente la mayoría de los aspectos del modelo.

3.2 DENORMALIZACIÓN

El afinamiento (tunning) del desempeño de una base de datos relacional requiere de un buen análisis de datos, el conocimiento claro y amplio de el SABD a utilizar y la comprensión de cuando utilizar datos denormalizados. La denormalización de datos debe considerarse cuando se tiene un problema de desempeño y debe ser la última opción, sobre todo si existen varias aplicaciones que comparten las estructuras de los datos. No confundir la denormalización con la introducción de redundancia en el esquema. Los siguientes puntos no implican denormalización:

- Uso de grupos repetidos.
- Almacenamiento de agregados. Por ejemplo el tener una columna para guardar el total de personas en una organización en la tabla Organización.
- Columnas con valores nulos para cosas que en realidad no existen. Generalmente esto es el resultado de combinar una superclase y una subclase.

A menos que se incluya una dependencia que no sea sobre la llave primaria completa, la tabla permanece normalizada totalmente.

4. APLICACIÓN DE LA METODOLOGÍA

4.1 ANÁLISIS DE REQUERIMIENTOS

RECOLECCIÓN DE REQUERIMIENTOS (DESCRIPCIÓN DEL PROBLEMA)

Una compañía aseguradora dedicada al ramo de seguros de vida requiere un sistema automatizado que le permita realizar la emisión y seguimiento de las pólizas de los seguros de vida individual que sus agentes comercializan entre los posibles contratantes (personas y empresas) que los solicitan. La compañía aseguradora tiene una oficina matriz y varias oficinas regionales y todo agente pertenece a alguna oficina. Existen algunos procesos que requieren una opinión técnica autorizada por ejemplo: determinar la aceptación del riesgo, generación de valores técnicos, valuación actuarial de pólizas, etc., por lo que es necesario el tener un acceso restringido por responsable de acuerdo a sus capacidades. Asimismo, debido a lo delicado de la información, se requiere tener un registro de quién realizó algún proceso. Una póliza de seguro de vida individual tiene como característica importante el que su tiempo de vida generalmente es mayor a un año, por lo que es necesario el tener el registro de movimientos con fechas para una mejor administración de la póliza.

El sistema debe contar con las siguientes funciones:

- Generar cotizaciones del seguro elegido a fin de que el contratante tenga conocimiento del costo y los beneficios que obtendría.
- Realizar la emisión de la solicitud y documentos correspondientes, teniendo un control que permita determinar en qué situación se encuentra la solicitud del seguro requerido y la viabilidad de otorgamiento del mismo en base al dictamen médico y técnico y al reaseguro
- Determinar el reaseguro automático de las pólizas y permitir el registro del reaseguro facultativo en base a los contratos que se tienen con los reaseguradores.
- Registrar los pagos realizados por el contratante al área de cobranza y efectuar el proceso de cancelación o conversión de pólizas por falta de pago.
- Generar los endosos debidos a cambios a la póliza solicitados por el asegurado o por el contratante tales como: cambios a los datos generales (datos personales del asegurado y sus beneficiarios, forma de pago), cambio de planes y/o coberturas, aumento o disminución de suma asegurada, conversión de seguro, rehabilitación, cancelación .
- Generar el aniversario para pólizas de más de un año, la renovación para pólizas que terminan su vigencia y los documentos para pagos subsecuentes de pólizas con forma de pago fraccionada.
- Controlar administrativamente los siniestros reportados para brindar un respaldo efectivo a los asegurados y beneficiarios.
- Proporcionar la información solicitada tanto por áreas internas (contabilidad, cobranzas, etc.) como por instituciones externas de control (Comisión Nacional de Seguros y Fianzas (CNSF), Asociación Mexicana de Instituciones del Seguro (AMIS)).
- Ya que el mercado de seguros tiene condiciones cambiantes, se debe considerar el tener catálogos para algunos datos como recargos, tipos de cambio, planes, coberturas y cualquiera otros que requieren modificarse en el tiempo a fin de darle flexibilidad al sistema.

Existen ciertas reglas generales que deben cumplirse para poder asegurar a una persona:

1. Que la edad esté dentro del rango de edades permitidas del seguro a solicitar.
2. Que la suma asegurada solicitada esté dentro del rango establecido por la compañía.
3. Que la subnormalidad y la extraprima ocupacional no excedan los límites máximos aceptados por la compañía.
4. Si el asegurado tiene más de una póliza en la compañía, el total de sus sumas aseguradas no debe exceder el máximo aceptado por la compañía.

MODELADO DE REQUERIMIENTOS CON CASOS DE USO

De la descripción anterior se pueden identificar los siguientes actores:

- Agente: Se encarga de promover y vender los seguros a los contratantes.
- Contratante: Contrata el seguro y por lo tanto es quien realizará los pagos de la póliza y puede solicitar cambios.
- Asegurado: Es el objeto del seguro, ya que si no hay asegurado no hay póliza.
- Responsable: Principal usuario directo del sistema pues se encarga de introducir los datos para la mayoría de los procesos. Hay dos tipos de responsable: médico y técnico.
- Reasegurador: Acepta o rechaza los riesgos que la compañía por sí sola no puede aceptar.
- Cobranza: Registra los pagos que el contratante hace a la póliza .
- Contabilidad: Requiere información para el registro contable de las operaciones realizadas para el ramo de vida individual (cifras de emisión, reaseguro, valuación, siniestros etc.)
- Instituciones Externas: Solicitan información ya sea para evaluación de la compañía o para estadísticas que ayuden a conocer más sobre el ramo de vida individual.
- Beneficiario: Recibe la indemnización en caso de siniestro por muerte.

Gráficamente, el modelo jerárquico de actores es el mostrado en la Fig. 4.1

Una vez realizado el modelo de actores, se procede a elaborar el diagrama general de casos de uso para el sistema al cual llamaremos SASVI (Sistema de Administración del Seguro de Vida Individual).

De la recolección de requerimientos se observa que el sistema debe considerar las siguientes transacciones:

1. Cotizar seguro
2. Controlar y emitir solicitud
3. Determinar vigor de póliza
4. Reasegurar póliza
5. Renovar póliza
6. Endosar póliza
7. Registrar siniestro
8. Valuar Póliza
9. Mantener catálogos

En la Fig. 4.2 se presenta el diagrama general de casos de uso para el SASVI; las asociaciones se pusieron en color para mejorar su visualización debido a que hay varios entrecruzamientos.

MODELO DE ACTORES DEL SISTEMA DE ADMINISTRACIÓN DEL SEGURO DE VIDA INDIVIDUAL

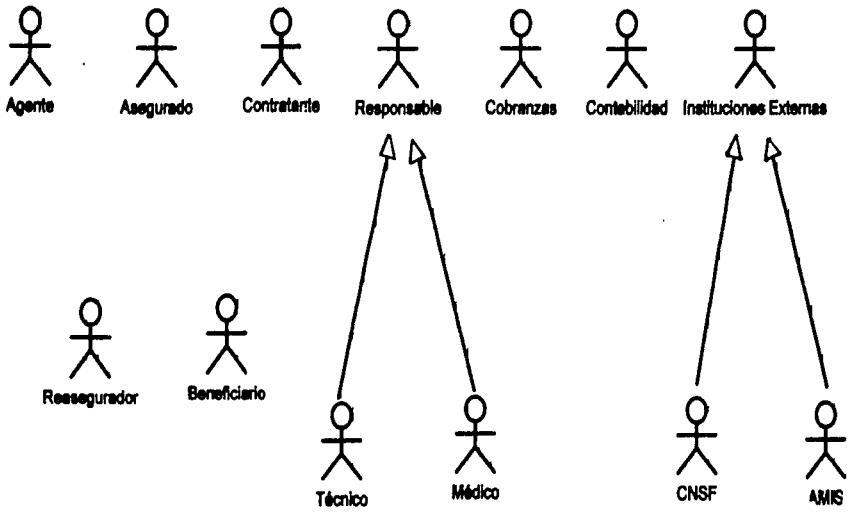


Fig. 4.1 Modelo de Actores del SASVI

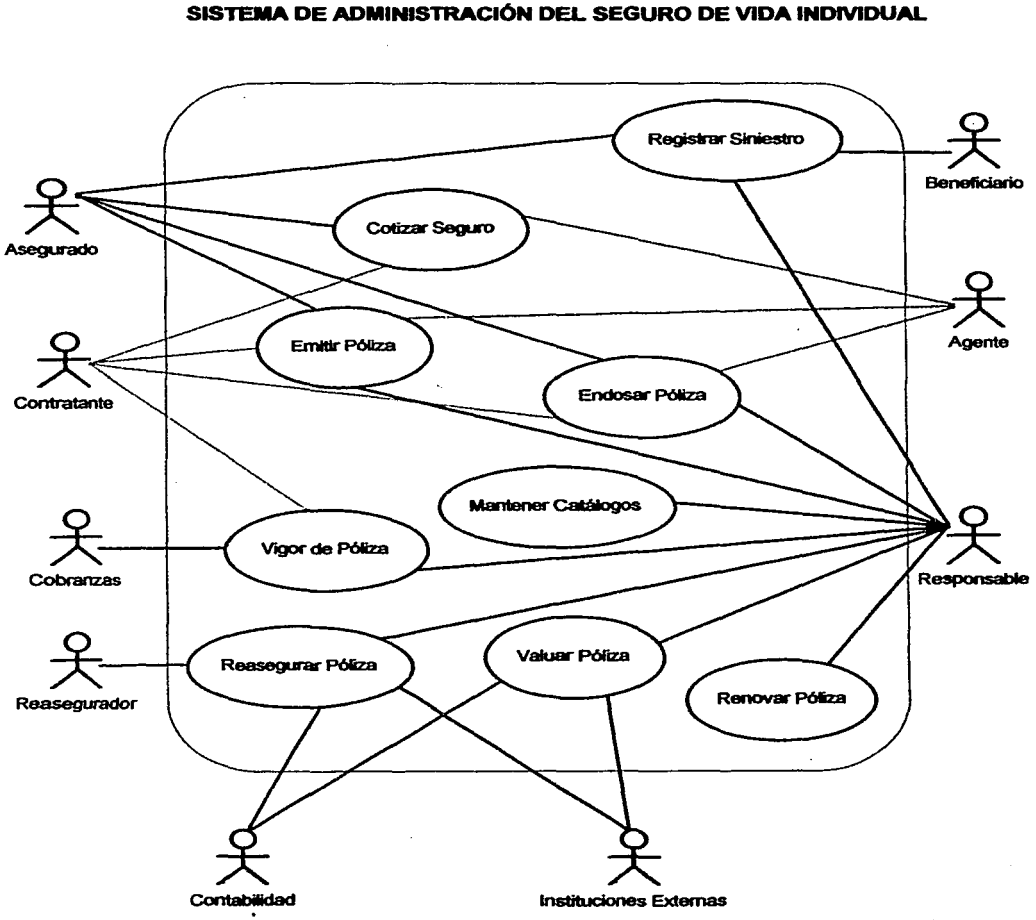


Fig. 4.2 Diagrama General de Casos de Uso del SASVI

A continuación se detalla cada uno de los casos presentados en la Fig. 4.2. Algunos necesitan casos de uso extensión, los cuales se detallan inmediatamente después del caso de uso al cual extienden. Debido a la interrelación que muestra el diagrama entre actores y casos, en cada caso de uso se especifican los actores que interactúan para mayor claridad.

Caso de uso: Cotizar Seguro
Actores: Contratante, agente, responsable
Resumen.
Genera un reporte de cotización del o los seguros solicitados por el contratante al agente. Cada reporte contiene la información acerca del seguro solicitado(plan básico y coberturas adicionales), plazo de pago, forma de pago, vigencia del seguro, supuesto asegurado, prima a pagar, fecha de elaboración.
Narrativa

1. El contratante solicita una cotización al agente o responsable.
2. El agente o responsable informa de los seguros que puede cotizar de acuerdo a las necesidades del contratante.
3. El contratante elige un seguro a cotizar.
4. El agente o responsable captura en el sistema los datos necesarios para generar la cotización.
5. El sistema calcula las primas y genera un reporte a pantalla y/o impresora por cada uno de los seguros elegidos.
6. El agente o responsable entrega el reporte de cotización al contratante.

Reglas del Negocio

a) Los datos necesarios para generar una cotización son: edad o fecha de nacimiento del prospecto de asegurado, condición de fumador, suma asegurada, plan básico y forma de pago.
b) Toda cotización debe tener una y sólo una cobertura básica (plan)
c) La cotización puede o no tener coberturas adicionales.
d) Solo puede incluirse una cobertura adicional por accidente.
e) Si la forma de pago elegida es fraccionada (pagar en partes) debe considerarse el recargo por pago fraccionado.
f) Debe hacerse notar el derecho de póliza que se cobrará en caso de que se emitiese una póliza con los datos indicados en la cotización.

Caso de uso: Emitir Solicitud
Actores: Contratante, agente, responsable
Resumen.
A partir de una solicitud de seguro, presentada por el agente o por el contratante, el responsable genera en el sistema pólizas nuevas y sus documentos correspondientes, almacenando toda la información requerida para el control de la solicitud y la generación de la póliza.
Narrativa

1. El contratante solicita una póliza de seguro al agente.
2. El agente y/o el contratante requisita la solicitud y la entrega al responsable.
3. El responsable captura en el sistema los datos de la solicitud.
4. El sistema determina si la solicitud está completa y los requisitos de aseguramiento a solicitar.
5. El sistema determina la subnormalidad por peso y tipo de dictamen para la solicitud.
6. El sistema determina el tipo de reaseguro para la solicitud.

7. El responsable genera la nueva póliza y sus recibos en el sistema para cada solicitud dictaminada, e imprime los documentos correspondientes: carátula de póliza, recibo, valores garantizados.
8. El responsable entrega al agente los documentos impresos.
9. El agente entrega al contratante o asegurado los documentos impresos.

Extensiones

1. El caso de uso **Solicitar datos y requisitos** extiende a **Emitir Solicitud** en el paso 4 cuando los datos de la solicitud están incompletos y/o el sistema determina solicitar requisitos de aseguramiento en base a suma asegurada, edad y riesgos indicados
2. El caso de uso **Dictaminar solicitud** extiende a **Emitir Solicitud** en el paso 5 cuando la solicitud no tiene dictamen normal.
3. El caso de uso **Solicitar Reaseguro** extiende a **Emitir Solicitud** en el paso 6 cuando el tipo de reaseguro para la solicitud es facultativo.

Reglas del Negocio

a) Un asegurado puede tener más de una solicitud de seguro.
b) Debe considerarse el catálogo de requisitos de aseguramiento para determinar automáticamente los requisitos a solicitar en base a suma asegurada y edad.
c) Una solicitud debe tener sólo una cobertura básica (plan) y puede tener una o más coberturas adicionales.
d) Una solicitud solo puede tener una cobertura adicional por accidente.
e) Inicialmente una solicitud sólo puede tener un tipo de dictamen de cualquiera de estos: Normal, Médico o Técnico.
f) El tipo de dictamen de una solicitud es normal si:
   - La subnormalidad por peso es igual a cero.
   - La suma asegurada es menor que la retención
   - El asegurado no tiene otros seguros en la misma compañía o en otra.
   - Se contestó negativamente todo el cuestionario de salud (excepto la condición de fumador)
   - El asegurado no practica actividades peligrosas.
g) Una solicitud requiere dictamen médico siempre que la subnormalidad por peso sea mayor que cero, se conteste afirmativamente cualquiera de las preguntas del cuestionario de salud (excepto condición de fumador) o se le soliciten requisitos de aseguramiento.
h) El tipo de dictamen de una solicitud es sólo técnico si el asegurado contestó afirmativamente la pregunta de actividades peligrosas y no cumple con las condiciones para requerir dictamen médico.
i) El tipo de reaseguro inicial para una solicitud es:
   - Ninguno: si la suma asegurada es menor o igual a la retención.
   - Automático: si la suma asegurada es mayor a la retención, pero menor o igual a la capacidad de la compañía.
   - Facultativo: si la suma asegurada es mayor a la capacidad.

Caso de uso: Solicitar datos y requisitos

Actores: Responsable, agente, asegurado.

Resumen

Inicia cuando en el caso de uso **Emitir Solicitud** el sistema determina que la solicitud está incompleta (le faltan datos) o se requieren requisitos de aseguramiento en base a suma asegurada, edad y riesgos indicados. El responsable puede generar la carta de petición de datos y requisitos y puede registrar los documentos entregados.

Narrativa

1. El sistema indica las solicitudes incompletas o que requieren requisitos de aseguramiento.
2. El responsable genera la carta de petición de datos y requisitos y la envía al agente o asegurado.
3. El agente o asegurado entregan los datos y requisitos solicitados.
4. El responsable registra en el sistema la entrega de los datos y requisitos solicitados.

Caso de uso: Dictaminar solicitud

Actores: Responsable médico, responsable técnico, asegurado,

Resumen

Este caso de uso inicia cuando en el caso de uso **Emitir Solicitud** el sistema determina que la solicitud requiere dictamen médico o técnico.

Permite registrar las calificaciones (dictamen) que el médico y/o el técnico asignan a cada una de las coberturas que se solicitan. También permite solicitar reaseguro facultativo.

Narrativa

1. El sistema indica las solicitudes que requieren dictamen médico y/o técnico.
2. El responsable revisa qué requisitos se han entregado y puede solicitar requisitos adicionales en base a su criterio.
3. El asegurado entrega los requisitos solicitados.
4. El responsable médico determina y registra sus calificaciones para cada cobertura.
5. El responsable técnico determina y registra sus calificaciones para cada cobertura.
6. El sistema determina el nuevo estatus de la solicitud.

Extensiones

1. El caso de uso **Solicitar reaseguro** extiende a **Dictaminar solicitud** en el punto 5 cuando el responsable técnico determina que la solicitud requiere reaseguro facultativo

Regla del Negocio

Toda solicitud que tenga dictamen médico posteriormente debe realizársele dictamen técnico.

Caso de uso: Solicitar Reaseguro

Actores: Responsable, reasegurador.

Resumen

Este caso de uso inicia cuando el responsable técnico determina que la solicitud requiere reaseguro facultativo. El responsable genera y envía una carta solicitando la aceptación del riesgo a los reaseguradores. El responsable registra la aceptación o rechazo.

Narrativa

1. El responsable busca en el sistema la solicitud que requiere reaseguro facultativo y que aún no ha enviado a reaseguro.
2. El responsable genera una carta de solicitando la aceptación del riesgo para cada uno de los reaseguradores con los que se tiene contrato y la envía.
3. El reasegurador informa el rechazo o la aceptación de la solicitud de reaseguro, así como las calificaciones que le asignó a las coberturas de la solicitud aceptada.
4. El responsable registra en el sistema la aceptación o rechazo de la solicitud de reaseguro.

Ya que el caso de uso **Emitir Solicitud** se extendió con diversos casos de uso, a continuación se presenta el diagrama de actividad correspondiente para una mejor visualización en conjunto:

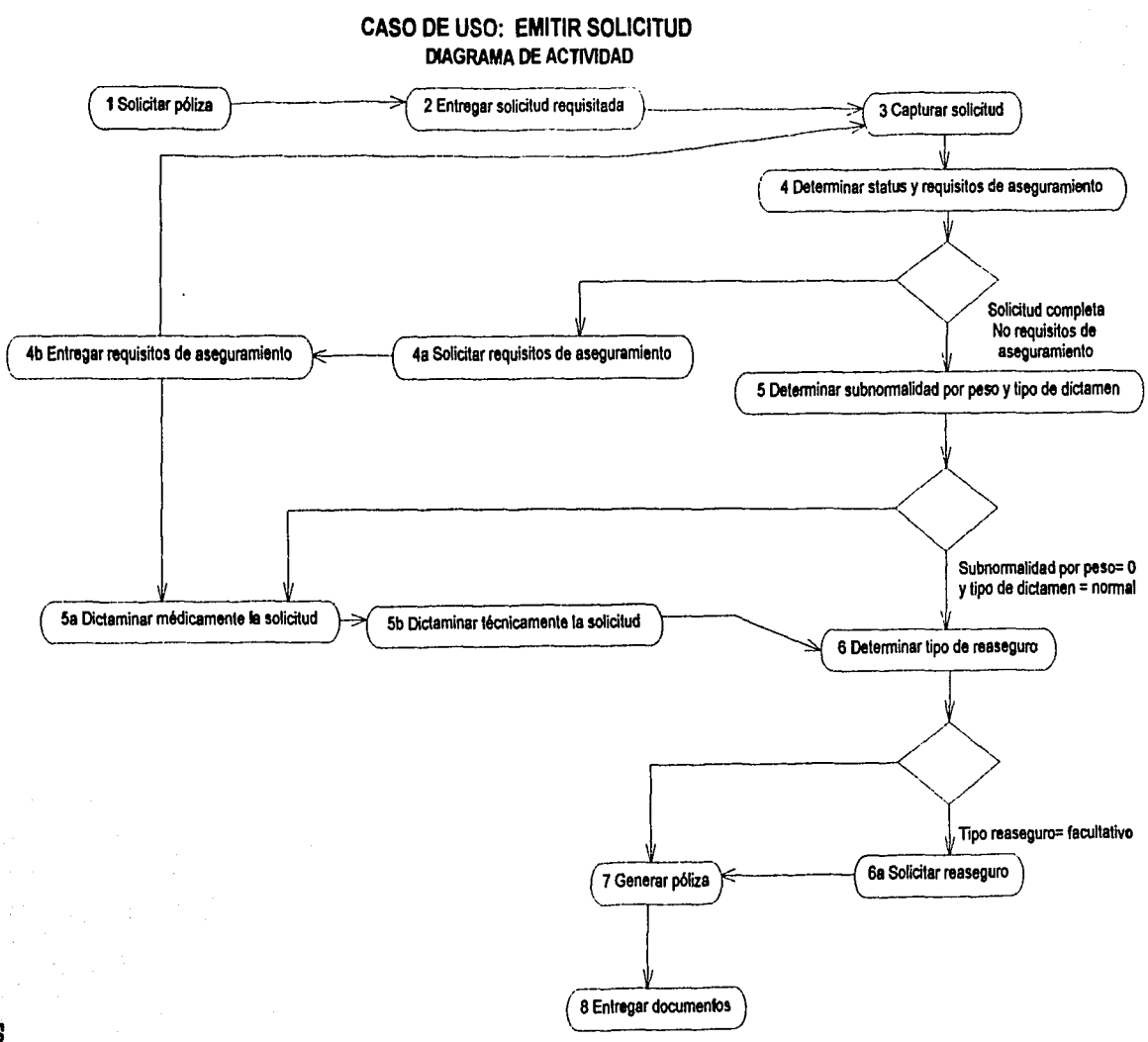


Fig. 4.3 Diagrama de Actividad del Caso de Uso Emitir Solicitud

Caso de uso: Vigor de Póliza
Actores: Contratante, cobranzas, responsable, asegurado.
Resumen
El responsable registra los pagos realizados por el contratante a su póliza y el sistema determina el nuevo estatus de la póliza con relación a sus recibos y produce el reporte general correspondiente y un reporte individual para cada póliza que se cancela o prorroga para informar al asegurado.
Narrativa

1. El contratante realiza el pago a su póliza al área de cobranzas (a través del agente o directamente).
2. El área de cobranzas registra el pago del recibo correspondiente e informa al responsable de dicho pago.
3. El responsable registra el pago al recibo correspondiente.
4. Periódicamente el responsable realiza la cancelación o conversión de las pólizas que no han recibido el pago correspondiente con el caso de uso Determinar estatus.
5. El sistema genera un reporte general de las pólizas canceladas o prorrogadas.

Caso de uso: Determinar Estatus
Actores: Responsable
Resumen
Este caso de uso extiende a otros cuando el responsable requiere dar un nuevo estatus a las pólizas en base a sus recibos.
Narrativa

1. El responsable indica la fecha a considerar.
2. El sistema selecciona las pólizas vigentes.
3. El sistema selecciona para cada póliza los recibos pendientes de pago con fecha de inicio de vigencia menor a la fecha indicada.
4. El sistema determina el nuevo estatus del recibo
5. El sistema determina el nuevo estatus de la póliza.
6. El sistema registra el nuevo estatus de la póliza y el recibo

Extensiones

1. El caso de uso **Cancelar Póliza** extiende a **Determinar Estatus** en el paso 5 cuando el nuevo estatus de la póliza es Cancelada
2. El caso de uso **Prorrogar Póliza** extiende a **Determinar Estatus** en el paso 5 cuando el nuevo estatus de la póliza es Prorrogada.

Reglas del Negocio

a) Un recibo pendiente de pago se cancela por falta de pago cuando su fecha inicio de vigencia más el número de días para aceptar pago (determinados por la compañía) es menor o igual a la fecha indicada para determinar estatus
b) Una póliza se cancela por falta de pago cuando el recibo correspondiente se canceló y la póliza aún no cumple tres años con pagos.
c) Una póliza se prorroga por falta de pago cuando el recibo correspondiente se canceló y la póliza ha sido pagada durante tres años (tiene derecho a valores garantizados).

Caso de uso: Cancelar Póliza
Actores: Asegurado
Resumen
Este caso de uso inicia cuando otro caso de uso determina el estatus cancelada para la póliza. El sistema cancela las coberturas tanto básicas como adicionales y genera un reporte de aviso de cancelación para el asegurado.

Caso de uso: Prorrogar Póliza
Actores: Asegurado
Resumen
Este caso de uso inicia cuando otro caso de uso determina el estatus prorrogada para la póliza. El sistema calcula el tiempo de prorroga para la cobertura básica más antigua de la póliza (la que se emitió primero), cancela las coberturas adicionales por invalidez, cambia el estatus de las coberturas básicas y genera una carta de aviso de conversión a prorrogado para el asegurado.

Caso de uso: Reasegurar Póliza.
Actores: Responsable, reasegurador
Resumen
El responsable asigna distribución de reaseguro a las pólizas que tienen reaseguro facultativo y puede consultar o modificar el tipo de reaseguro para las pólizas con reaseguro automático. Genera reportes de la distribución de las pólizas en reaseguro y documentos informativos para pagos al reasegurador (bordereaux).
Narrativa
1. El responsable consulta las pólizas que tienen reaseguro.
2. El responsable elige una póliza para asignarle la distribución de reaseguro facultativo.
3. El responsable registra la asignación de reaseguro facultativo.
4. Periódicamente el responsable genera los reportes de documentos para pagos al reasegurador.

Reglas del Negocio
Una póliza puede tener reaseguro automático y facultativo a criterio del responsable.

Caso de uso: Renovar Póliza.
Actores: Responsable, asegurado, contratante
Resumen
El responsable busca y registra el porcentaje de incremento de suma asegurada por aniversario o el rechazo dependiendo de lo indicado por el contratante. El responsable indica el periodo a procesar y utiliza los casos de uso Aniversario de póliza para pólizas que cumplan un año más y el caso de uso Renovación de póliza para pólizas que se encuentren en el último año de vigencia y el caso de uso Generar Recibos para pólizas con forma de pago fraccionado.
Narrativa
1. El responsable busca las pólizas para las cuales el contratante solicita un porcentaje de incremento de suma asegurada por aniversario diferente al que otorga la compañía.
2. El responsable registra para cada póliza encontrada el incremento solicitado o el rechazo de éste.
3. El responsable indica el periodo a procesar y el tipo de proceso.

4. Una vez realizado el proceso indicado el responsable puede generar un reporte de las pólizas procesadas.
5. El responsable imprime los reportes generados por el proceso elegido.

Extensiones

1. El caso de uso **Aniversario de póliza** extiende a **Renovar póliza** en el punto 3, si el proceso elegido es Aniversario.
2. El caso de uso **Renovación de póliza** extiende a **Renovar póliza** en el punto 3, si el proceso elegido es Renovación.
3. El caso de uso **Generar Recibos** extiende a **Renovar póliza** en el punto 3, si el proceso elegido es Generar Recibos.

Caso de uso: Aniversario de Póliza.

Resumen

Este caso extiende a **Renovar Póliza** cuando el responsable indicó que el tipo de proceso es Aniversarios. El sistema busca las pólizas vigentes que cumplen un año más y genera los recibos y documentos correspondientes.

Narrativa

1. El sistema busca las pólizas vigentes que cumplen un año más.
2. El sistema determina, para cada póliza encontrada, las nuevas coberturas a otorgar.
3. El sistema determina la vigencia de las coberturas adicionales de los planes emitidos anteriormente.
4. El sistema genera los recibos y documentos correspondientes para cada póliza encontrada.

Extensiones.

1. El caso de uso **Generar coberturas** extiende a **Aniversario de Póliza** en el paso 2 cuando la póliza tiene aceptación de incremento de suma asegurada por aniversario y porcentaje de incremento de suma asegurada diferente de cero, por lo tanto se le generan nuevas coberturas.

Reglas del Negocio

a) Para dar incremento a la suma asegurada al aniversario, la póliza debe tener esa indicación.
b) Si el asegurado o contratante ha rechazado consecutivamente más del límite máximo de rechazos de incremento a la suma asegurada por aniversario que establece la compañía, entonces ya no se le otorga el incremento automático.

Caso de uso: Generar Coberturas

Resumen

Este caso de uso inicia cuando otro caso de uso requiere la generación de nuevas coberturas. El sistema busca la cobertura básica correspondiente a la edad del asegurado y el tipo de cobertura inicial contratada. También determina el otorgamiento de las coberturas adicionales ya contratadas para la nueva cobertura básica.

Caso de uso: Renovación de Póliza.

Resumen

Este caso extiende a **Renovar Póliza** cuando el responsable indicó que el tipo de proceso es Renovación. El sistema busca las pólizas vigentes cuyo fin de vigencia está dentro del periodo a procesar y genera los recibos y documentos correspondientes.

Narrativa

1. El sistema busca las pólizas vigentes cuyo fin de vigencia está dentro del periodo indicado.

2. El sistema determina, para cada póliza encontrada, las nuevas coberturas a otorgar con el caso de uso **Generar coberturas**.
3. El sistema determina la nueva fecha de fin de vigencia de la póliza en base a la nueva cobertura básica
4. El sistema genera los recibos y documentos correspondientes para cada póliza encontrada.

Caso de uso: Generar recibos
Resumen
Este caso extiende a **Renovar Póliza** cuando el responsable indicó que el tipo de proceso es Generar recibos. El sistema busca las pólizas vigentes cuya forma de pago es fraccionada, no cumplen un año más en la compañía y su fecha de fin de vigencia no está dentro del periodo a procesar y genera los recibos correspondientes.

Caso de uso: Registrar Siniestro.
Actores: Asegurado, contratante, beneficiario, responsable, contabilidad
Resumen
El responsable registra el siniestro reportado por el asegurado, contratante o beneficiario. El responsable dictamina procedencia del siniestro y reporta a contabilidad y reaseguro (en caso necesario) para realizar el pago del siniestro.
Narrativa
1. El asegurado, contratante o beneficiario reporta el siniestro al responsable.
2. El responsable verifica en el sistema que la póliza esté en vigor.
3. El responsable captura en el sistema los datos del siniestro.
4. El sistema determina si el siniestro procede.
5. El sistema genera un reporte de documentos requeridos para respaldar el siniestro.
6. El asegurado, contratante o beneficiario entrega los documentos solicitados
7. El responsable médico dictamina la procedencia del siniestro
8. El responsable técnico dictamina la procedencia del siniestro
9. El sistema genera un reporte de pago inmediato.
10. El responsable entrega el reporte de siniestros procedentes a contabilidad.
11. Contabilidad genera los cheques para cubrir el siniestro y los entrega al responsable
12. El responsable entrega los cheques al asegurado, contratante o beneficiario.

Extensiones.
1. El caso de uso **Siniestros Improcedentes** extiende a Registrar Siniestro cuando el sistema determina que el siniestro no procede.
2. El caso de uso **Informar siniestro** extiende a Registrar Siniestro cuando el responsable técnico dictamina la procedencia del siniestro.

Caso de uso: Siniestros Improcedentes
Resumen
Este caso de uso extiende **Registrar siniestro** cuando el sistema determina que el siniestro no procede. El sistema registra las causas por las que se rechaza el siniestro y el responsable puede consultar esa información y generar una carta de rechazo del siniestro.

Caso de uso: Informar siniestro
Resumen
Este caso de uso extiende a **Registrar siniestro** cuando el responsable técnico determina la procedencia del siniestro y la póliza está reasegurada. El sistema genera una carta para cada reasegurador solicitando el pago de la proporción correspondiente de los siniestros procedentes

Caso de uso: Endosar Póliza
Actores: Asegurado, contratante, responsable
Resumen.
Este caso de uso inicia cuando el asegurado o contratante solicita un cambio permitido a cualquiera de los datos de una póliza vigente o en posibilidad de rehabilitación. El responsable registra el cambio solicitado y el sistema determina si el cambio genera una modificación en primas, en cuyo caso registra la nueva prima y genera el recibo correspondiente y un reporte de endoso.
Narrativa.
1. El asegurado o contratante solicita un cambio a los datos de su póliza.
2. El responsable verifica que la póliza esté vigente o en periodo de rehabilitación.
3. El responsable registra el cambio solicitado.
4. El sistema determina si el cambio implica una modificación a la prima.
5. El sistema genera un reporte de endoso.

Extensiones
1. El caso de uso **Calcula endoso** extiende a **Endosar Póliza** en el paso 4 cuando el cambio implica modificación en la prima.

Caso de uso: Calcula Endoso
Este caso de uso extiende a **Endosar Póliza** en el paso 4 cuando el cambio solicitado genera una modificación a la prima. El sistema calcula la prima a pagar por el endoso, registra la nueva prima y genera el recibo correspondiente al endoso.

Caso de uso: Valuar Póliza
Actores: Responsable, instituciones externas, contabilidad
Resumen.
Periódicamente, instituciones externas solicitan la información sobre las reservas que tiene la compañía para su cartera de pólizas vigentes. El responsable indica una fecha de proceso para realizar la valuación. El sistema utiliza el caso de uso **Determinar estatus** para considerar sólo las pólizas realmente vigentes, calcula las reservas para cada una de éstas y genera los reportes valuación correspondientes. El responsable informa los resultados obtenidos a contabilidad y a las instituciones externas.
Narrativa.
1. Instituciones externas solicitan información sobre las reservas para pólizas vigentes a una fecha.
2. El responsable indica la fecha del proceso al sistema.
3. El sistema determina las pólizas vigentes usando el caso **Determinar status**.
4. El sistema calcula y registra las reservas tanto de coberturas básicas como de coberturas adicionales para cada póliza vigente
5. El responsable genera el reporte de la valuación realizada.
6. El responsable informa las cifras de sumas aseguradas y reservas a contabilidad y a las instituciones externas.

Caso de uso: Mantener Catálogos
Actores: Responsable
Resumen.
El responsable agrega, actualiza, consulta o elimina los registros que requiere de los diferentes catálogos necesarios para la generación adecuada de los procesos que realiza el sistema.

Tabla de datos.

Clase	Descripción
Agente	Conjunto de agentes que laboran para la compañía.
CoberturaAdicional	Coberturas adicionales (por invalidez o accidente) ofrecidas por la compañía.
Estado	Entidades federativas correspondientes a direcciones.
FormaPago	Diferentes periodicidades de pago aceptadas por la compañía.
Moneda	Monedas en las cuales se manejan las sumas aseguradas, primas y recargos.
Municipio	Municipios en los cuales están divididas las entidades federativas.
Oficina	Oficinas en las que están distribuidos los agentes.
Plan	Coberturas básicas ofrecidas por la compañía.
Tarifa	Tarifas establecidas para los planes y coberturas adicionales por invalidez para cada edad.
Reasegurador	Conjunto de reaseguradores que tienen contrato con la compañía.
Recargo	Recargos por pago fraccionado de acuerdo a forma de pago y moneda.
Responsable	Conjunto de personas encargadas y autorizadas para realizar procesos en el sistema.

Una vez detallados los casos de uso, se procede a presentar el modelado de datos iniciando con el diagrama de subsistemas y el modelo de datos con diagramas de clase.

4.2 MODELADO DE DATOS

DIAGRAMA DE SUBSISTEMAS

Ahora se elabora el diagrama de subsistemas a partir de los casos de uso que se han detallado. Para ello se buscan objetos en los casos, agrupando los que afectan los mismos objetos. Al revisar los casos de uso se puede obtener la siguiente división:

Subsistema Cotización
Casos de uso: Cotizar seguro

Subsistema Solicitud
Casos de uso: Emitir solicitud, Solicitar datos y requisitos, Dictaminar solicitud, Solicitar reaseguro

Subsistema Póliza
Casos de uso: Vigor de póliza, Determinar estatus, Cancelar póliza, Prorrogar póliza, Renovar póliza, Aniversario de póliza, Renovación de póliza, Generar coberturas, Generar recibos, Endosar póliza, Calcula endoso, Valuar póliza, Reasegurar póliza

Subsistema Siniestro
Casos de uso: Registrar siniestro, Siniestros improcedentes, Informar siniestro.

Subsistema Catálogo
Casos de uso: Mantener catálogos.

El diagrama de subsistemas se muestra en la Fig. 4.4.

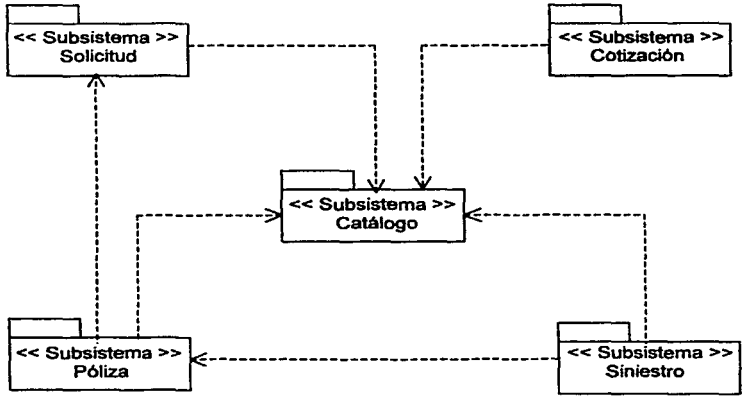


Fig. 4.4 Diagrama de Subsistemas del SASVI

En la Fig. 4.4 puede observarse que los subsistemas **Solicitud**, **Cotización**, **Póliza** y **Siniestro** dependen del subsistema **Catálogo**, esto es porque **Catálogo** contiene las clases que son requeridas para el funcionamiento adecuado de los otros subsistemas. El subsistema **Póliza** depende además de **Solicitud**. Por su parte el subsistema **Siniestro** depende también de **Póliza**.

DIAGRAMAS DE CLASE
A continuación se elabora el diagrama de clase para cada uno de los subsistemas. Antes de presentarlos, se hacen algunas aclaraciones al respecto:

- Para mayor rapidez los diagramas se elaboraron en la herramienta CASE Power Designer versión 9.0 de SYBASE, ya que dicha herramienta fue la más accesible de obtener, pero no significa que el modelado esté orientado para utilizar tal SABD.
- Dado que la indicación de la identidad de objeto (con {OID}) para los atributos, es una extensión de UML, la herramienta no lo maneja, por lo que se tuvo que indicar con texto adicional.
- La indicación {OID} se puso a los atributos que son parte de la identidad de objeto de clases que no heredan de otra clase o que no son clases asociación.
- Cuando se utilizan clases que son parte de otro subsistema, se presenta el atributo o atributos que corresponden a la identidad de objeto, sólo para una mejor

referencia y visualización (pues generalmente se suprimen las divisiones de atributos y operaciones).

- Para simplificar los diagramas, la propiedad {nullable} no se puso a ningún atributo, lo cual no significa que ninguno acepte nulos, pero esto se indicará al crear la tabla.
- Debido a la interrelación entre subsistemas, los diagramas de clase correspondientes se presentan en el siguiente orden:

  - Subsistema Catálogo
  - Subsistema Cotización
  - Subsistema Solicitud
  - Subsistema Póliza
  - Subsistema Siniestro

La cuestión medular al pasar de los casos de uso a los diagramas de clase es la identificación adecuada de los objetos y clases que se requerirán para soportar suficiente y correctamente los requerimientos de información del sistema del cual será parte la base de datos. Este punto todavía es bastante dependiente de la experiencia del desarrollador y por supuesto del nivel de conocimiento del problema, por lo que generalmente se necesita realizar mucho trabajo "invisible" antes de obtener un diagrama de clase satisfactorio. Una ventaja al utilizar el método presentado, es que es iterativo e incremental, por lo que se puede pasar de los casos de uso a los diagramas de clase y regresar para realizar las correcciones o adiciones necesarias.

SUBSISTEMA CATÁLOGO

DIAGRAMA DE CLASE

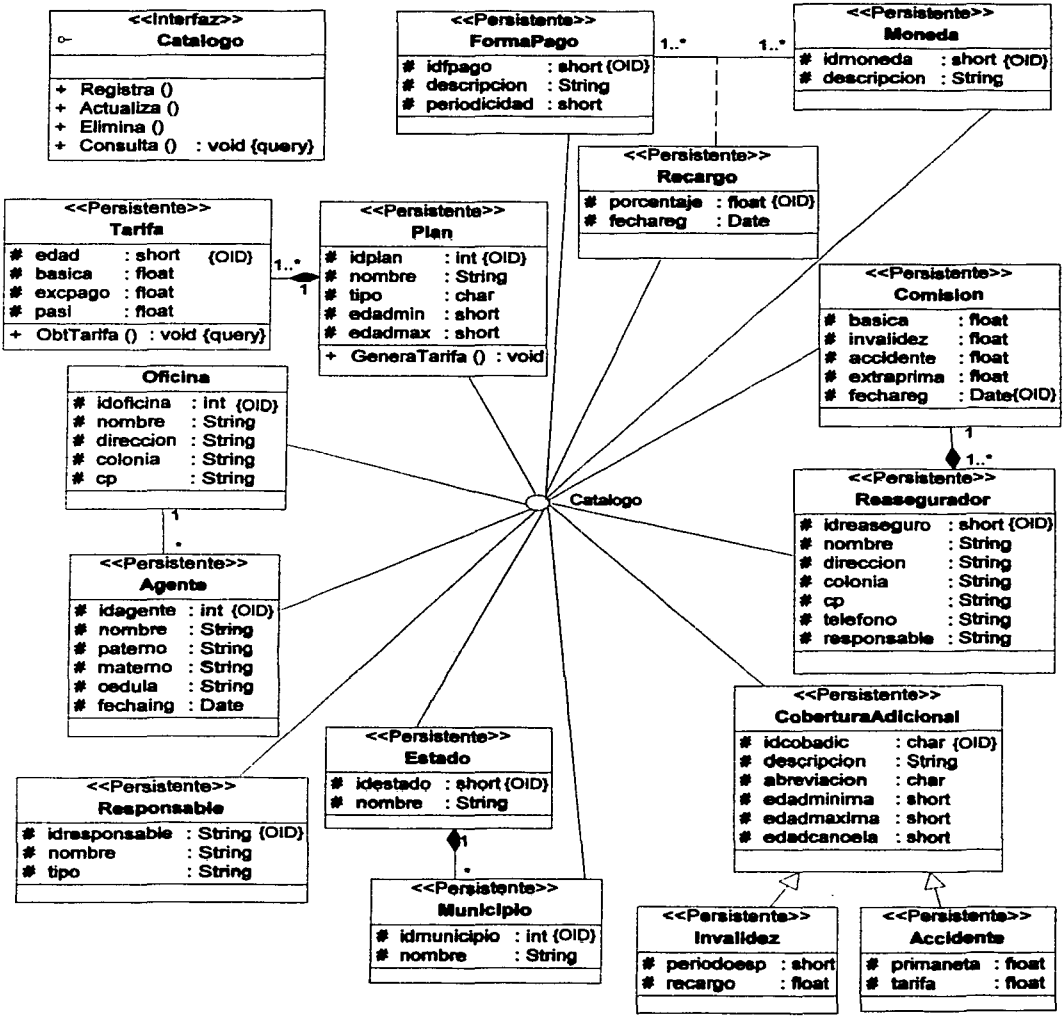


Fig. 4.5 Diagrama de Clase del Subsistema Catálogo

SUBSISTEMA COTIZACIÓN

DIAGRAMA DE CLASE

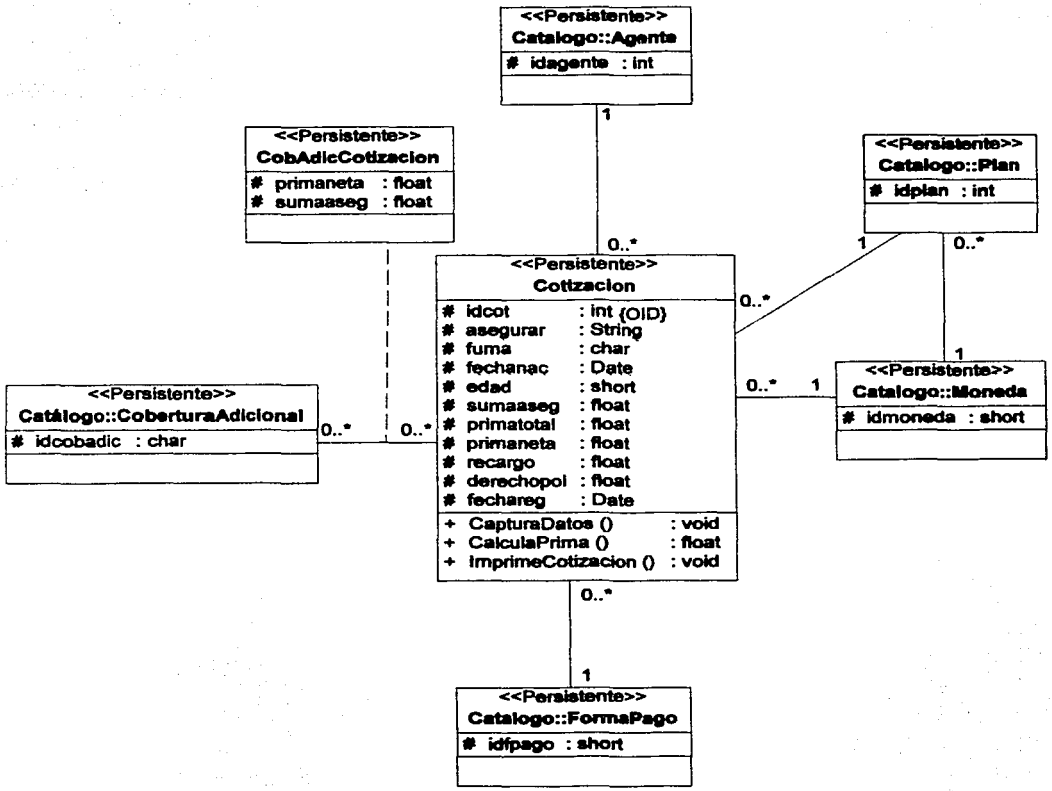


Fig. 4.6 Diagrama de Clase del Subsistema Cotización

SUBSISTEMA SOLICITUD
DIAGRAMA DE CLASE

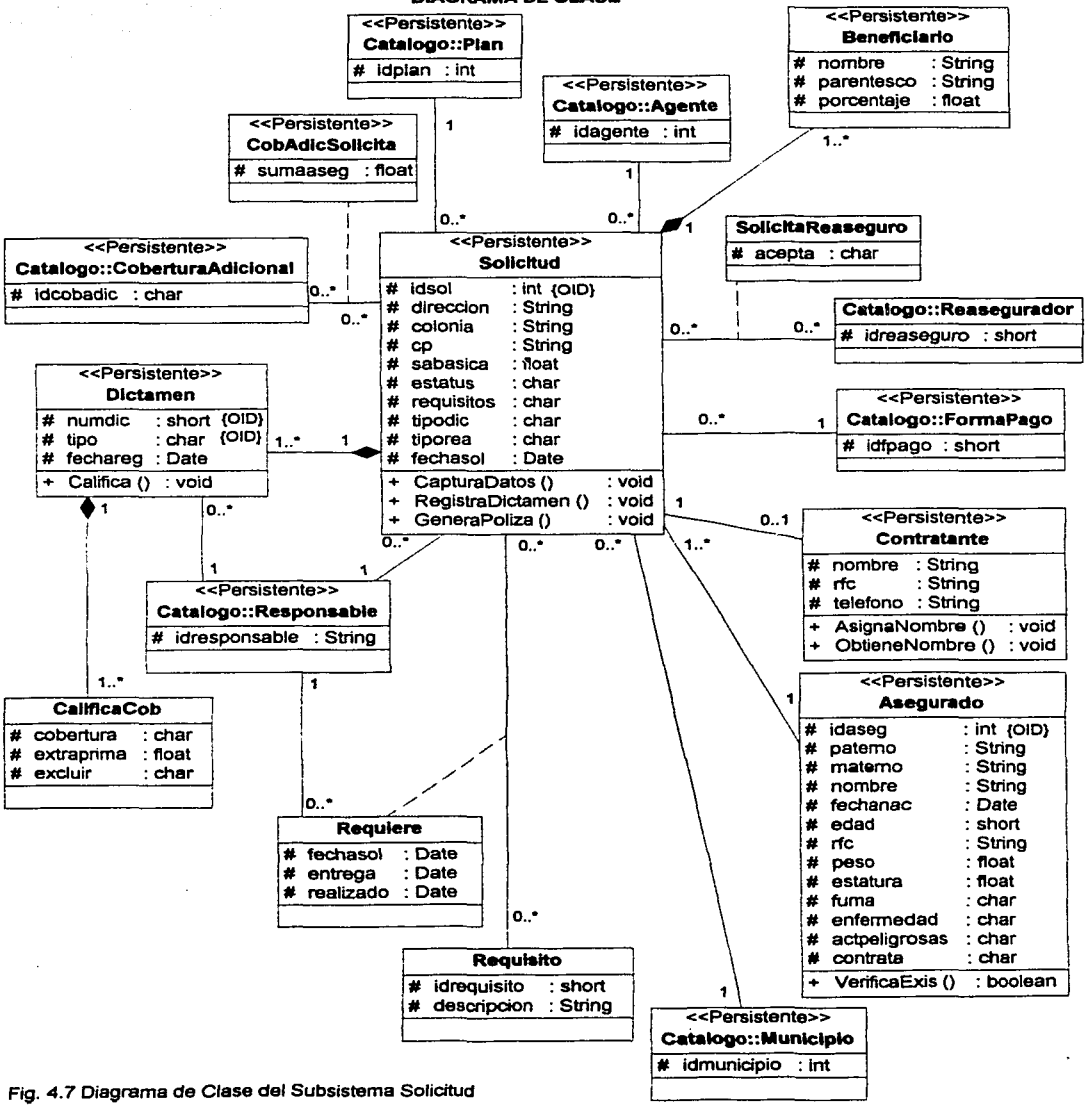


Fig. 4.7 Diagrama de Clase del Subsistema Solicitud

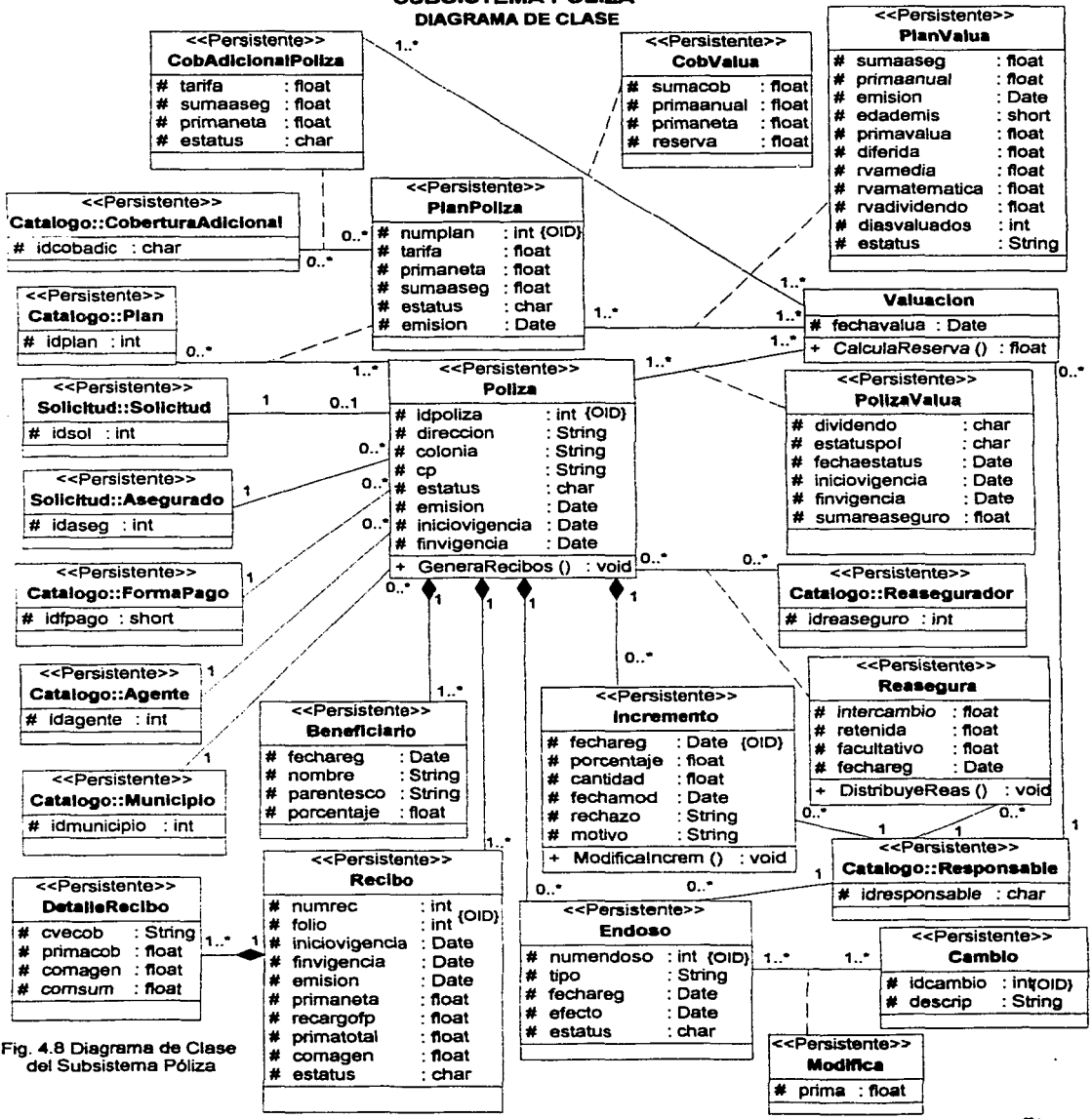


Fig. 4.8 Diagrama de Clase del Subsistema Póliza

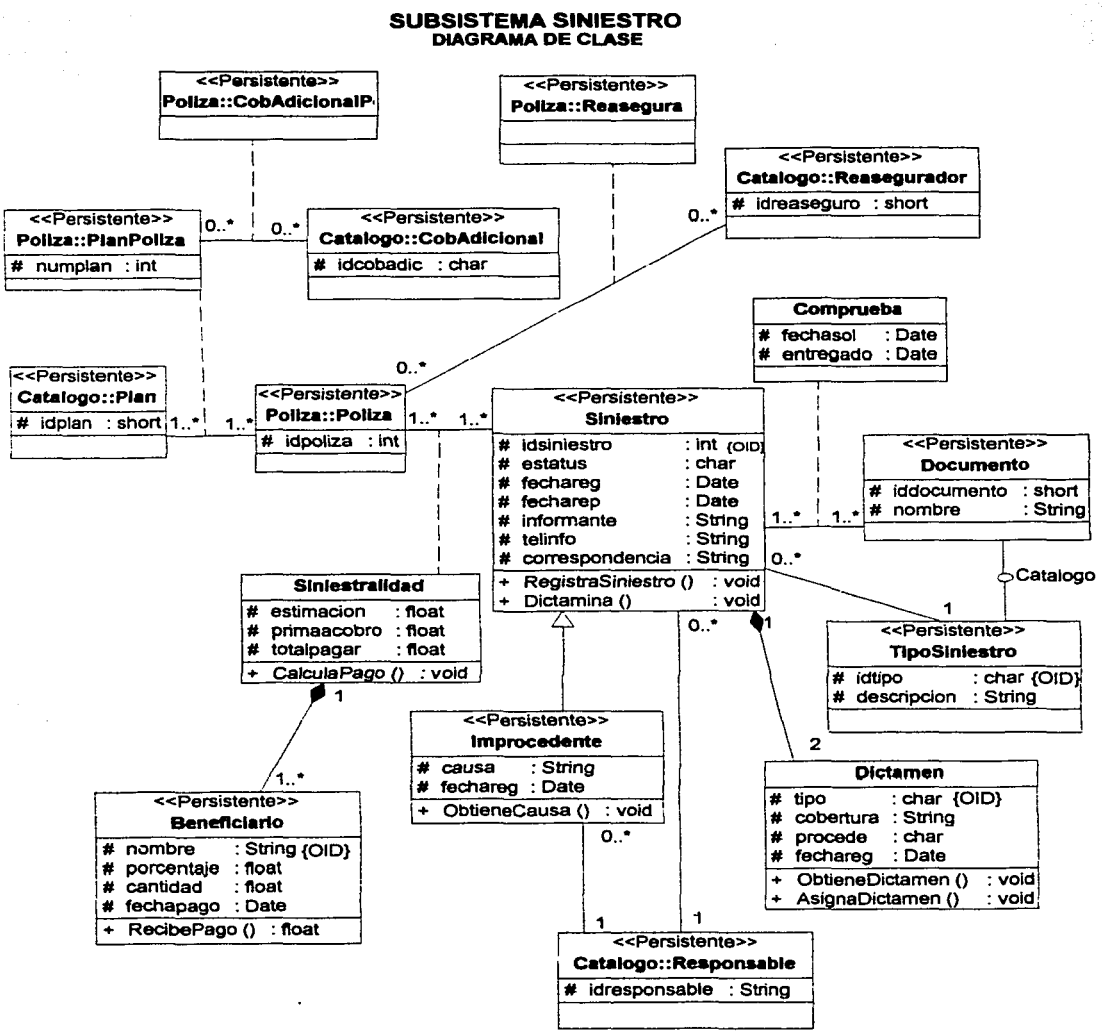


Fig. 4.9 Diagrama de Clase del Subsistema Siniestro

4.3 CONVERSIÓN DEL MODELO DE DATOS UML A UN ESQUEMA DE BASE DE DATOS RELACIONAL

A fin de que el esquema quede abierto para ser utilizado en cualquier SABD relacional, se hizo la conversión del modelo de datos UML a un esquema en una versión de estándar de SQL.

A continuación se muestra la transformación de los tipos de atributo UML (que se utilizaron) a los tipos de dato para columnas SQL :

Tipo Atributo UML	Tipo Columna SQL
Char	CHAR
String	CHAR(254)
Short	SMALLINT
Int	INTEGER
Float	FLOAT
Date	DATE

Aunque el mapeo de un tipo String de UML es a un tipo char(254) en SQL, cuando es suficiente con menos caracteres, se indica el número correspondiente.

Debido a que los procedimientos almacenados y triggers son dependientes del SABD a utilizar, no se hace el mapeo de las operaciones en el esquema.

Ahora se presenta el esquema para cada uno de los subsistemas:

- **Subsistema Catálogo:**

Primero se nombra el espacio para el subsistema:

```
CREATE  DATABASE catalogo;
```

En el diagrama de clases de este subsistema (Fig. 4.5) se presenta una interfaz llamada **Catalogo** a la cual se asocian la mayoría de las clases, esto significa que todas esas clases implementarán las operaciones indicadas en dicha interfaz; estas asociaciones con la interfaz no se mapean en el esquema por corresponder a operaciones.

Como se indicó en el capitulo anterior, las clases y sus atributos mapean directamente en tablas y columnas, pero las asociaciones y generalizaciones se transforman de manera particular. Por ejemplo, en el diagrama (Fig. 4.5) hay una asociación binaria entre la clases **Oficina** y **Agente** y es el rol que corresponde a **Oficina** el que tiene una multiplicidad máxima de uno, por lo que al convertirse en tablas, la llave primaria de **Oficina** debe adicionarse a las columnas de **Agente** como llave externa; estas tablas quedarán de la siguiente manera:

```
CREATE TABLE Oficina (
   idoficina      INTEGER  PRIMARY KEY,
   nombre         CHAR(25) NOT NULL,
   direccion      CHAR(35),
   colonia        CHAR(25),
   cp             CHAR(5) );
```

```
CREATE TABLE Agente (
   idagente        INTEGER  PRIMARY KEY,
   idoficina       INTEGER  NOT NULL REFERENCES Oficina,
   nombre          CHAR(25) NOT NULL,
   paterno         CHAR(20) NOT NULL,
   materno         CHAR(20) NOT NULL,
   cedula          CHAR(15) NOT NULL,
   fechaing        DATE  NOT NULL);
```

La Fig. 4.5 también muestra una relación de generalización entre las clase **CoberturaAdicional** y las clases **Accidente** e **Invalidez**. Dicha generalización sólo tiene un nivel, por lo que se utiliza el mapeo directo para representarla en el esquema:

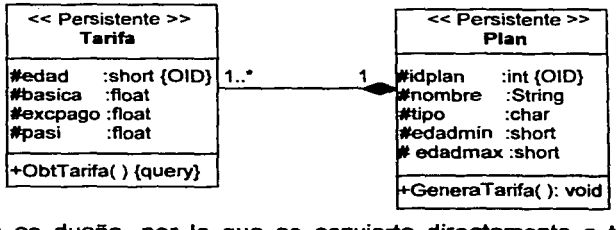
```
CREATE TABLE CoberturaAdicional
   idcobadic       CHAR PRIMARY KEY,
   descripcion     CHAR(45) NOT NULL,
   abreviacion     CHAR(4) NOT NULL,
   edadminima      SMALLINT NOT NULL,
   edadmaxima      SMALLINT NOT NULL,
   edadcancela     SMALLINT NOT NULL );

CREATE TABLE Accidente (
   idcobadic       CHAR NOT NULL REFERENCES CoberturaAdicional,
   primaneta       FLOAT NOT NULL,
   tarifa          FLOAT NOT NULL,
   CONSTRAINT Accidente_PK PRIMARY KEY (idcobadic) );

CREATE TABLE Invalidez (
   idcobadic       CHAR NOT NULL REFERENCES CoberturaAdicional
   periodoesp      SMALLINT NOT NULL,
   recargo         FLOAT NOT NULL,
   CONSTRAINT Invalidez_PK PRIMARY KEY (idcobadic) );
```

Obsérvese que a las tablas **Accidente** e **Invalidez** se les agregó la columna **idcobadic** con restricción de llave primaria y llave externa, pues corresponde a la llave primaria de la tabla **CoberturaAdicional.**

Otra relación interesante en la Fig. 4.5 es la asociación de agregación por composición que se da entre las clases Plan y Tarifa, Estado y Municipio, Reasegurador y Comisión. Tomando como ejemplo Plan y Tarifa:



La clase **Plan** es dueña, por lo que se convierte directamente a tabla, sin requerir de columnas adicionales:

```
CREATE TABLE Plan (
   idplan       INTEGER PRIMARY KEY,
   nombre       CHAR(35) NOT NULL,
   tipo         CHAR NOT NULL,
   edadmin      SMALLINT NOT NULL,
   edadmax      SMALLINT NOT NULL );
```

Por su parte, la clase **Tarifa** es propiedad de **Plan**, por lo que debe agregársele **idplan** como parte de su llave primaria y la opción de cascada para borrado y actualizaciones:

```
CREATE TABLE Tarifa (
   idplan          INTEGER  NOT NULL REFERENCES Plan
     ON DELETE CASCADE  ON UPDATE CASCADE,
   edad            SMALLINT  NOT NULL,
   basica          FLOAT,
   excpago         FLOAT,
   pasi            FLOAT,
   CONSTRAINT PK_Plan PRIMARY KEY (idplan, edad) );
```

Para las restantes agregaciones la conversión es similar:

```
CREATE TABLE Estado (
   idestado        SMALLINT  PRIMARY KEY,
   nombre          CHAR(25) NOT NULL );

CREATE TABLE Municipio (
   idestado     SMALLINT  NOT NULL  REFERENCES Estado,
     ON DELETE  CASCADE  ON UPDATE CASCADE,
   idmunicipio  INTEGER   NOT NULL,
   nombre       CHAR(35) NOT NULL,
   CONSTRAINT Municipio_PK PRIMARY KEY (idestado, idmunicipio) );

CREATE TABLE Reasegurador (
   idreaseguro     SMALLINT  PRIMARY KEY,
   nombre          CHAR(25) NOT NULL,
   direccion       CHAR(35),
   colonia         CHAR(25),
   cp              CHAR(5),
   telefono        CHAR(15),
   responsable     CHAR(30) );

CREATE TABLE Comision (
   idreaseguro     SMALLINT NOT NULL REFERENCES Reasegurador
     ON DELETE CASCADE  ON UPDATE CASCADE,
   fechareg        DATE   NOT NULL,
   basica          FLOAT NOT NULL,
   invalidez       FLOAT NOT NULL,
   accidente       FLOAT NOT NULL,
   extraprima      FLOAT NOT NULL,
   CONSTRAINT Comision_PK PRIMARY KEY (idreaseguro, fechareg) );
```

Continuando con el diagrama presentado en la Fig. 4.5, las clases **Moneda** y **FormaPago** están asociadas, pero además hay una clase asociación: **Recargo**. En este caso las

clases asociadas mapean directamente en tablas y se crea una tabla para la clase asociación, cuya llave primaria será formada por las llaves primarias de las clases asociadas:

```
CREATE TABLE Moneda (
   idmoneda        SMALLINT  PRIMARY KEY,
   descripcion     CHAR(15) NOT NULL );

CREATE TABLE FormaPago (
   idfpago         SMALLINT PRIMARY KEY,
   descripcion     CHAR(10) NOT NULL,
   periodicidad    SMALLINT NOT NULL );

CREATE TABLE Recargo (
   idmoneda        SMALLINT  NOT NULL REFERENCES Moneda,
   idfpago         SMALLINT  NOT NULL REFERENCES FormaPago,
   fechareg        DATE  NOT NULL,
   porcentaje      FLOAT NOT NULL,
   CONSTRAINT PK_Recargo PRIMARY KEY (idmoneda, idfpago, fechareg) );
```

La clase **Responsable** se convierte directamente a tabla sin mayores modificaciones:

```
CREATE TABLE Responsable (
   idresponsable   CHAR(10) PRIMARY KEY,
   nombre          CHAR(35),
   tipo            CHAR  );
```

En la presentación de los siguientes subsistemas, sólo se hacen notas para las tablas que tienen detalles especiales, pues ya se ha indicado como se debe realizarse la conversión para las asociaciones, generalizaciones y clases asociación.

- **Subsistema Cotización**

```
CREATE DATABASE cotizacion;
```

La Fig. 4.6 presenta el diagrama de este subsistema y puede observarse que la clase **Cotizacion** tiene una asociación binaria con **Agente**, **Moneda**, **FormaPago** y **Plan** que pertenecen al subsistema **Catalogo** y la multiplicidad para los roles de estas clases es de uno, por lo que al convertir la clase **Cotizacion** en tabla se adicionan las llaves primarias de las tablas correspondientes como columnas llave externa.

```
CREATE TABLE Cotizacion (
   idcot           INTEGER   PRIMARY KEY,
   idagente        INTEGER   NOT NULL REFERENCES Catalogo:Agente,
   idmoneda        SMALLINT  NOT NULL REFERENCES Catalogo: Moneda,
   idfpago         SMALLINT  NOT NULL REFERENCES Catalogo:FormaPago,
   idplan          INTEGER   NOT NULL REFERENCES Catalogo:Plan,
   asegurar        CHAR(45),
   fuma            CHAR NOT NULL,
   fechanac        DATE,
   edad            SMALLINT NOT NULL,
```

```
    sumaaseg   FLOAT NOT NULL,
    primatotal  FLOAT NOT NULL,
    primaneta   FLOAT NOT NULL,
    recargo     FLOAT NOT NULL,
    derechopol  FLOAT NOT NULL,
    fechareg    DATE NOT NULL );
```

Las clase **CobAdicCotizacion** es una clase asociación, por lo que se transforma como se indicó anteriormente.

```
CREATE TABLE CobAdicCotizacion (
    idcobadic   CHAR  NOT NULL REFERENCES  Catalogo:CoberturaAdicional,
    idcot       INTEGER  NOT NULL REFERENCES Cotizacion,
    primaneta   FLOAT NOT NULL,
    sumaaseg    FLOAT NOT NULL,
    CONSTRAINT PK_CobAdicCotizacion PRIMARY KEY (idcobadic, idcot) );
```

- **Subsistema Solicitud**

```
CREATE DATABASE solicitud;
```

Como se observa en el diagrama para este subsistema (Fig. 4.7) la clase **Asegurado** se transforma directamente en tabla:

```
CREATE TABLE Asegurado (
    idaseg      INTEGER  PRIMARY KEY,
    paterno     CHAR(25) NOT NULL,
    materno     CHAR(20) NOT NULL,
    nombre      CHAR(20) NOT NULL,
    fechanac    DATE NOT NULL,
    edad        SMALLINT NOT NULL,
    rfc         CHAR(10),
    peso        FLOAT NOT NULL,
    estatura    FLOAT NOT NULL,
    fuma        CHAR NOT NULL,
    enfermedad  CHAR NOT NULL,
    actpeligrosas CHAR NOT NULL,
    contrata    CHAR NOT NULL );
```

Al convertir en tabla la clase **Solicitud**, es necesario adicionarle como llaves externas las llaves primarias de las clases con las cuales tiene una asociación binaria y cuya multiplicidad es de 1: **Plan**, **Asegurado**, **Agente**, **Estado**, **FormaPago**, **Responsable** y **Municipio**:

```
CREATE TABLE Solicitud (
    idsol       INTEGER  PRIMARY KEY,
    idplan      INTEGER  NOT NULL REFERENCES Catalogo:Plan,
    idaseg      INTEGER  NOT NULL REFERENCES Asegurado,
    idagente    INTEGER  NOT NULL REFERENCES Catalogo: Agente,
    idestado    SMALLINT  NOT NULL REFERENCES Catalogo:Estado,
    idfpago     SMALLINT  NOT NULL REFERENCES Catalogo:FormaPago,
    idresponsable CHAR(10) NOT NULL REFERENCES Catalogo:Responsable,
```

```
idmunicipio     INTEGER    NOT NULL REFERENCES Catalogo:Municipio,
direccion       CHAR(35) NOT NULL,
colonia         CHAR(25) NOT NULL,
cp              CHAR(5) NOT NULL,
sabasica        FLOAT NOT NULL,
estatus         CHAR  NOT NULL,
requisitos      CHAR NOT NULL,
tipodic         CHAR  NOT NULL,
tiporea         CHAR NOT NULL,
fechasol        DATE NOT NULL );
```

La clase **Beneficiario** tiene una agregación por composición con **Solicitud**, por lo que la llave primaria de **Solicitud** pasa a formar parte de la llave primaria de **Beneficiario**:

```
CREATE TABLE Beneficiario (
   idsol          INTEGER  NOT NULL REFERENCES Solicitud
      ON DELETE CASCADE  ON UPDATE CASCADE,
   nombre         CHAR(50)   NOT NULL,
   parentesco     CHAR(15),
   porcentaje      FLOAT NOT NULL,
   CONSTRAINT PK_Beneficiario PRIMARY KEY (idsol, nombre) );
```

Aunque en el mundo real no es tan obvia la agregación por composición entre **Dictamen** y **Solicitud**, dada la dependencia de existencia de un dictamen con una solicitud (si no hay solicitud no hay dictamen), es la única manera en que la llave primaria de **Solicitud** sea parte de la llave primaria de **Dictamen**:

```
CREATE TABLE Dictamen (
   idsol          INTEGER  NOT NULL REFERENCES Solicitud
      ON DELETE CASCADE  ON UPDATE CASCADE,
   numdic         SMALLINT   NOT NULL,
   tipo           CHAR   NOT NULL,
   idresponsable  CHAR(10) NOT NULL REFERENCES Catalogo:Responsable,
   fechareg       DATE NOT NULL,
   CONSTRAINT PK_Dictamen PRIMARY KEY (idsol, numdic, tipo) );

CREATE TABLE CalificaCob (
   idsol          INTEGER  NOT NULL REFERENCES Dictamen
      ON DELETE CASCADE  ON UPDATE CASCADE,
   numdic         SMALLINT  NOT NULL REFERENCES Dictamen
      ON DELETE CASCADE  ON UPDATE CASCADE,
   tipo           CHAR  NOT NULL REFERENCES Dictamen,
      ON DELETE CASCADE  ON UPDATE CASCADE,
   cobertura      CHAR  NOT NULL,
   extraprima     FLOAT NOT NULL,
   excluir        CHAR NOT NULL,
   CONSTRAINT PK_CalificaCob PRIMARY KEY (idsol, numdic, tipo, cobertura) );
```

La clase **Contratante** está en la misma situación que **Dictamen**, pero como una solicitud sólo puede tener como máximo un contratante, basta con **idsol** como llave primaria para la tabla **Contratante**:

```
CREATE TABLE Contratante (
   idsol          INTEGER  NOT NULL REFERENCES Solicitud,
   nombre         CHAR(50)  NOT NULL,
   rfc            CHAR(13),
   telefono       CHAR(15),
   CONSTRAINT PK_Contratante PRIMARY KEY (idsol);
```

Las siguientes tablas provienen de clases asociación, por lo que ya se sabe como se realizó su transformación:

```
CREATE TABLE CobAdicSolicita (
   idcobadic     CHAR  NOT NULL REFERENCES Catalogo:CoberturaAdicional,
   idsol         INTEGER  NOT NULL REFERENCES Solicitud,
   sumaaseg      FLOAT NOT NULL,
   CONSTRAINT PK_CobAdicSolicita PRIMARY KEY (idcobadic, idsol) );

CREATE TABLE Requiere (
   idsol          INTEGER  NOT NULL REFERENCES Solicitud,
   idrequisito    SMALLINT  NOT NULL REFERENCES Catalogo:Requisito,
   idresponsable CHAR(10) NOT NULL REFERENCES Catalogo:Responsable,
   fechasol       DATE,
   entrega        DATE,
   realizado      DATE,
   CONSTRAINT PK_Requiere PRIMARY KEY (idsol, idrequisito) );

CREATE TABLE SolicitaReaseguro (
   idsol          INTEGER  NOT NULL REFERENCES Solicitud,
   idreaseguro    SMALLINT NOT NULL REFERENCES Catalogo:Reasegurador,
   acepta         CHAR,
   CONSTRAINT  PK_SolicitaReaseguro PRIMARY KEY (idsol, idreaseguro) );
```

La clase **Requisito** se convierte a tabla directamente:

```
CREATE TABLE Requisito (
   idrequisito       SMALLINT  PRIMARY KEY,
   descripcion       CHAR(45));
```

- **Subsistema Póliza**

```
CREATE DATABASE poliza;
```

La Fig. 4.8 presenta el diagrama de clase del subsistema **Póliza**, este diagrama es el que tiene el mayor número de clases y relaciones debido a que es la póliza lo que debe administrarse durante el tiempo que esta vigente el contrato de seguro entre la aseguradora y el contratante. A las clases **Poliza** y **Valuacion** es necesario adicionarles columnas llave externa correspondientes para cada una de las clases con las que tienen una asociación binaria con multiplicidad máxima de uno.

```
CREATE TABLE Valuacion (
   fechavalua     DATE  PRIMARY KEY,
   idresponsable  CHAR  NOT NULL REFERENCES Catalogo:Responsable );
```

```
CREATE TABLE Poliza (
   idpoliza        INTEGER  PRIMARY KEY,
   idmunicipio     INTEGER   NOT NULL REFERENCES Catalogo:Municipio,
   idfpago         SMALLINT NOT NULL REFERENCES Catalogo:FormaPago,
   idaseg          INTEGER   NOT NULL REFERENCES Solicitud:Asegurado,
   idestado        SMALLINT NOT NULL REFERENCES Catalogo:Estado,
   idagente        INTEGER  NOT NULL REFERENCES Catalogo:Agente,
   idsol           INTEGER  NOT NULL REFERENCES Solicitud:Solicitud,
   direccion       CHAR(35) NOT NULL,
   colonia         CHAR(25) NOT NULL,
   cp              CHAR(5) NOT NULL,
   estatus         CHAR NOT NULL,
   emision         DATE NOT NULL,
   iniciovigencia  DATE NOT NULL);
```

Las clase **Cambio** se convierten directamente en tabla:

```
CREATE TABLE Cambio (
   idcambio     INTEGER  PRIMARY KEY,
   descrip      CHAR(25) );
```

Las siguientes tablas provienen de una asociación de agregación por composición con la clase **Poliza**:

```
CREATE TABLE Beneficiario (
   idpoliza     INTEGER  NOT NULL REFERENCES Poliza
     ON DELETE CASCADE  ON UPDATE CASCADE,
   nombre       CHAR(50) NOT NULL,
   fechareg     DATE  NOT NULL,
   parentesco CHAR(15),
   porcentaje   FLOAT,
   CONSTRAINT PK_Beneficiario PRIMARY KEY (idpoliza, nombre, fechareg) );

CREATE TABLE Endoso (
   idpoliza       INTEGER NOT NULL REFERENCES Poliza
     ON DELETE CASCADE  ON UPDATE CASCADE,
   numendoso  INTEGER  NOT NULL,
   idresponsable CHAR(10) NOT NULL REFERENCES Catalogo:Responsable,
   tipo           CHAR(1),
   fechareg       DATE NOT NULL,
   efecto         DATE NOT NULL,
   estatus        CHAR NOT NULL,
   CONSTRAINT PK_Endoso PRIMARY KEY (idpoliza, numendoso) );

CREATE TABLE Incremento (
   idpoliza       INTEGER  NOT NULL REFERENCES Poliza
     ON DELETE CASCADE  ON UPDATE CASCADE,
   fechareg       DATE  NOT NULL,
   idresponsable CHAR(10)  NOT NULL REFERENCES Catalogo:Responsable,
   porcentaje     FLOAT NOT NULL,
   cantidad       FLOAT NOT NULL,
   fechamod       DATE,
   rechazo        CHAR(2) NOT NULL,
   motivo         CHAR(20),
CONSTRAINT PK_Incremento PRIMARY KEY (idpoliza, fechareg) );
```

```
CREATE TABLE Recibo (
   idpoliza          INTEGER  NOT NULL REFERENCES Poliza
     ON DELETE CASCADE  ON UPDATE CASCADE,
   numrec            INTEGER  NOT NULL,
   folio             INTEGER,
   iniciovigencia    DATE NOT NULL,
   finvigencia       DATE NOT NULL,
   emision           DATE NOT NULL
   primaneta          FLOAT NOT NULL,
   recargofp          FLOAT NOT NULL,
   primatotal         FLOAT NOT NULL,
   comagen            FLOAT NOT NULL,
   estatus            CHAR NOT NULL,
   CONSTRAINT PK_Recibo PRIMARY KEY (idpoliza, numrec) );
```

La tabla **DetRecibo** proviene de una asociación de agregación por composición con la clase **Recibo**:

```
CREATE TABLE DetRecibo (
   idpoliza          INTEGER  NOT NULL REFERENCES Recibo
     ON DELETE CASCADE  ON UPDATE CASCADE
   numrec            INTEGER  NOT NULL REFERENCES Recibo,
     ON DELETE CASCADE  ON UPDATE CASCADE
   cvecob            CHAR(5) NOT NULL,
   primacob          FLOAT NOT NULL,
   comagen           FLOAT NOT NULL,
   comsum            FLOAT NOT NULL,
   CONSTRAINT PK_DetRecibo PRIMARY KEY (idpoliza, numrec, cvecob) );
```

Las siguientes tablas se originaron de clases asociación por lo que ya se sabe como se creó su llave primaria:

```
CREATE TABLE PlanPoliza (
   idplan            INTEGER  NOT NULL REFERENCES Catalogo:Plan ,
   idpoliza          INTEGER NOT NULL REFERENCES Poliza,
   numplan           INTEGER  NOT NULL,
   tarifa            FLOAT NOT NULL,
   primaneta         FLOAT NOT NULL,
   sumaaseg          FLOAT NOT NULL,
   estatus           CHAR NOT NULL,
   emision           DATE NOT NULL,
   CONSTRAINT PK_PlanPoliza PRIMARY KEY (idplan, idpoliza, numplan) );

CREATE TABLE CobAdicionalPoliza (
   idplan            INTEGER NOT NULL REFERENCES PlanPoliza,
   idpoliza          INTEGER NOT NULL REFERENCES PlanPoliza,
   numplan           INTEGER NOT NULL REFERENCES PlanPoliza,
   idcobadic         CHAR NOT NULL,
   tarifa            FLOAT NOT NULL,
   sumaaseg          FLOAT NOT NULL,
   primaneta         FLOAT NOT NULL,
   estatus           CHAR,
   CONSTRAINT PK_CobAdicionalPoliza PRIMARY KEY (idplan, idpoliza, numplan, idcobadic) );
```

```
CREATE TABLE Modifica (
   idpoliza      INTEGER  NOT NULL REFERENCES Endoso,
   numendoso  INTEGER  NOT NULL REFERENCES Endoso,
   idcambio     INTEGER  NOT NULL REFERENCES Cambio,
   prima         FLOAT,
   CONSTRAINT PK_Modifica PRIMARY KEY (idpoliza, numendoso, idcambio) );

CREATE TABLE Reasegura (
   idpoliza        INTEGER  NOT NULL REFERENCES Poliza,
   idreaseguro  INTEGER  NOT NULL REFERENCES Catalogo:Reasegurador,
   idresponsable CHAR(10)  NOT NULL Catalogo:Responsable,
   intercambio   FLOAT NOT NULL,
   retenida       FLOAT NOT NULL,
   facultativo    FLOAT NOT NULL,
   fechareg       DATE NOT NULL,
   CONSTRAINT PK_Reasegura PRIMARY KEY (idpoliza, idreaseguro) );

CREATE TABLE PlanValua (
   fechavalua     DATE NOT NULL REFERENCES Valuacion,
   idplan          INTEGER NOT NULL REFERENCES  PlanPoliza,
   idpoliza        INTEGER  NOT NULL REFERENCES PlanPoliza,
   numplan         INTEGER NOT NULL REFERENCES PlanPoliza,
   sumaaseg       FLOAT NOT NULL,
   primaanual      FLOAT NOT NULL,
   emision         DATE NOT NULL,
   edademis        SMALLINT NOT NULL,
   primavalua      FLOAT NOT NULL,
   diferida        FLOAT NOT NULL,
   rvamedia        FLOAT NOT NULL,
   rvamatematica FLOAT NOT NULL,
   rvadividendo   FLOAT NOT NULL,
   diasvaluados    INTEGER NOT NULL,
   estatus         CHAR NOT NULL,
   CONSTRAINT PK_PlanValua PRIMARY KEY (idplan, idpoliza, fechavalua, numplan) );

CREATE TABLE PolizaValua (
   idpoliza          INTEGER NOT NULL REFERENCES Poliza,
   fechavalua       DATE  NOT NULL REFERENCES Valuacion,
   dividendo        CHAR NOT NULL,
   estatuspol       CHAR NOT NULL,
   fechaestatus     DATE NOT NULL,
   iniciovigencia    DATE NOT NULL,
   finvigencia       DATE NOT NULL,
   sumareaseguro  FLOAT NOT NULL,
   CONSTRAINT PK_PolizaValua PRIMARY KEY (idpoliza, fechavalua) );

CREATE TABLE CobValua (
   idplan       INTEGER NOT NULL REFERENCES CobAdicionalPoliza,
   idpoliza     INTEGER NOT NULL REFERENCES CobAdicionalPoliza,
   numplan     INTEGER NOT NULL REFERENCES CobAdicionalPoliza,
   idcobadic   CHAR NOT NULL REFERENCES CobAdicionalPoliza,
   fechavalua DATE NOT NULL REFERENCES Valuacion,
   sumacob     FLOAT NOT NULL,
   primaanual  FLOAT NOT NULL,
```

```
    primaneta   FLOAT NOT NULL,
    reserva     FLOAT NOT NULL,
CONSTRAINT PK_CobValua PRIMARY KEY (idplan, idpoliza, numplan, idcobadic, fechavalua) );
```

- **Subsistema Siniestro**

```
CREATE DATABASE Siniestro;
```

El diagrama de clase para este subsistema se mostró en la Fig. 4.9. Las clases **Documento** y **TipoSiniestro** se transforman directamente en tablas:

```
CREATE TABLE Documento (
   iddocumento  SMALLINT PRIMARY KEY,
   nombre       CHAR(30) NOT NULL );

CREATE TABLE TipoSiniestro (
   idtipo       CHAR  PRIMARY KEY,
   descripcion CHAR(30) NOT NULL );
```

Al convertir la clase **Siniestro** en tabla se le agregan dos columnas llave externa pues tiene asociación binaria con multiplicidad de uno para las clases **TipoSiniestro** y **Catalogo::Responsable**.

```
CREATE TABLE Siniestro (
   idsiniestro  INTEGER  PRIMARY KEY,
   idtipo       CHAR  NOT NULL REFERENCES TipoSiniestro (idtipo),
   idresponsable CHAR(10) NOT NULL REFERENCES Catalogo:Responsable,
   estatus      CHAR NOT NULL,
   fechareg     DATE NOT NULL,
   fecharep     DATE NOT NULL,
   informante CHAR(50) NOT NULL,
   telinfo      CHAR(15) NOT NULL,
   correspondencia  CHAR(60),
   CONSTRAINT PK_Siniestro PRIMARY KEY (idsiniestro) );
```

La clase **Improcedente** hereda de **Siniestro** (relación de generalización) y como sólo es un nivel se utiliza el mapeo directo:

```
CREATE TABLE Improcedente (
   idsiniestro      INTEGER  NOT NULL REFERENCES Siniestro,
   idresponsable CHAR(254) NOT NULL REFERENCES Catalogo:Responsable,
   causa            CHAR(120) NOT NULL,
   fechareg         DATE NOT NULL,
   entregado        DATE,
  CONSTRAINT PK_Improcedente PRIMARY KEY (idsiniestro);
```

Las siguientes tablas provienen de clases asociación, por lo que ya se conoce como se convirtieron:

```
CREATE TABLE Comprueba (
   idsiniestro     INTEGER  NOT NULL REFERENCES Siniestro,
   iddocumento  SMALLINT NOT NULL REFERENCES Documento,
   fechasol        DATE NOT NULL,
   entregado       DATE,
  CONSTRAINT PK_Comprueba PRIMARY KEY (idsiniestro, iddocumento) );


CREATE TABLE Siniestralidad (
   idpoliza        INTEGER NOT NULL REFERENCES Poliza:Poliza,
   idsiniestro     INTEGER NOT NULL REFERENCES Siniestro,
   estimacion      FLOAT,
   primaacobro FLOAT,
   totalpagar      FLOAT,
   CONSTRAINT PK_Siniestralidad PRIMARY KEY (idpoliza, idsiniestro) );
```

Las clases **Dictamen** y **Beneficiario** tienen una asociación de agregación por composición con **Siniestro** y **Siniestralidad** respectivamente, por lo que quedan de la siguiente forma:

```
CREATE TABLE Dictamen (
   idsiniestro  INTEGER  NOT NULL REFERENCES Siniestro,
      ON DELETE CASCADE  ON UPDATE CASCADE,
   tipo            CHAR NOT NULL,
   cobertura    CHAR(5) NOT NULL,
   procede      CHAR NOT NULL,
   fechareg     DATE NOT NULL,
   CONSTRAINT PK_Dictamen PRIMARY KEY (idsiniestro, tipo );

CREATE TABLE Beneficiario (
   idpoliza        INTEGER NOT NULL REFERENCES Siniestralidad
     ON DELETE CASCADE  ON UPDATE CASCADE,
   idsiniestro  INTEGER NOT NULL REFERENCES Siniestralidad,
     ON DELETE CASCADE  ON UPDATE CASCADE,
   nombre       CHAR(60) NOT NULL,
   porcentaje   FLOAT NOT NULL,
   cantidad     FLOAT NOT NULL,
   fechapago    DATE,
   CONSTRAINT PK_Beneficiario PRIMARY KEY (idpoliza, idsiniestro, nombre) );
```

Aunque al utilizar una herramienta CASE, posiblemente se tenga la opción para generar automáticamente el esquema relacional, es importante realizar una revisión del esquema generado considerando lo expuesto en la Tabla 3.1 Conversión del Modelo de Datos UML a un Esquema Relacional, a fin de verificar la formación correcta de las llaves primarias y externas.

CONCLUSIONES

El modelado de los datos con la metodología propuesta apoyada en los conceptos y notación de UML mostrados, se va haciendo de manera sencilla y entendible, además de que cualquiera que esté familiarizado con notaciones Entidad-Relación en particular y de diseño de sistemas en general, puede migrar casi sin problema hacia su utilización pues la mayoría de los elementos de UML presentados, refieren muchos aspectos comprendidos en metodologías y notaciones utilizadas anteriormente, pero en una manera estándar.

No obstante que al principio puede parecer que se requiere mucho esfuerzo y trabajo para obtener un modelo de datos más completo y correcto, realmente se está haciendo una inversión que rendirá buenos resultados, pues además de que el modelo sirve para un mejor diseño de la base de datos, también queda una documentación utilizable para adiciones futuras.

UML tiene diversos puntos a favor para ser utilizado:

- Puede aplicarse a la gran mayoría de los sistemas, aunque no sean de software.
- Facilita la comunicación desarrollador-cliente.
- Tiene una notación estándar sencilla y basada en notaciones anteriores.
- Cuenta con elementos que permiten modelar un sistema en su totalidad, sin tener que ir de una notación a otra en las diferentes fases del sistema.

Particularmente en la aplicación presentada, al hacer uso de los conceptos y notación UML en el diseño de la base, se obtuvieron las siguientes ventajas

- Presentación escrita y gráfica del análisis de requerimientos en un formato estándar, fácil de entender, revisar y corregir, tanto por el desarrollador como por el usuario, sin tener que invertir demasiado tiempo.
- Posibilidad de dividir la base de datos en varios subsistemas con su correspondiente diagrama de clase, permitiendo entender mejor cada parte sin tener que enfrentar la base de datos en general.
- Posibilita la reutilización tanto parcial como total de la base de datos, ya sea para el mismo sistema o para otros.
- El modelo de datos obtenido es fácil de explicar y comprender.
- El modelo de datos es independiente del manejador a utilizar pues igualmente se puede aplicar para el diseño de bases de datos a implementarse en un SABD orientado a objetos, por lo que se logra la independencia lógica de datos.
- Evita la necesidad de aplicar el proceso de normalización de tablas al realizar la conversión de las clases al modelo relacional.

Aunque se tienen varias ventajas, nada es perfecto, por lo que a continuación se enuncian algunas desventajas:

- Para desarrolladores acostumbrados a los conceptos de entidades y relaciones (como yo) es difícil el determinar las operaciones asociadas con los objetos de las clases, pues este aspecto es nuevo con respecto a las metodologías Entidad-Relación.
- Es relativamente difícil encontrar herramientas CASE accesibles que manejen UML con apego total al estándar además de requerirse tiempo para aprender la manera correcta de utilizarlas.

Obviamente, el éxito de un diseño no sólo depende de la notación utilizada, generalmente es más importante la experiencia del desarrollador y su nivel de conocimiento del problema a diseñar, pero el contar con una notación y conceptos estándar que ayuden a modelar el problema es un soporte indispensable para generar elementos de software más flexibles y reutilizables.

Todavía existen algunos aspectos que deben trabajarse para auxiliar en el esfuerzo de diseño, como son:

- Técnica para pasar del modelado de requerimientos con casos de uso a la identificación directa de los objetos, y por ende las clases, que se utilizarán en los diagramas respectivos.
- Aunque en el mundo real no sea tan obvio algún tipo de relación entre los objetos, algunas relaciones entre clases se tienen que modelar en cierta forma (por ejemplo en asociación de agregación por composición) a fin de que al pasar del modelo de datos al esquema se pueda obtener directamente la llave primaria adecuada de la tabla dependiente.

Sin embargo, el utilizar la metodología presentada, es una buena opción para obtener un diseño con un perfil estándar que será más fácil de asimilar por futuros usuarios (desarrolladores).

BIBLIOGRAFIA

Hawryszkiewycz T., Igor
"Relational Database Design"
Prentice Hall, 1982

Korth F. Henry; Silberschatz Abraham
"Database System Concepts"
Mc Graw Hill, 1984

Date C. J.
"Introducción a los sistemas de bases de datos"
Addison Wesley Iberoamericana, 1986

Maier, David
"The Theory of Relational Databases"
Computer Science Press, 1983

Ullman, Jeffrey
"Principles of Database Systems"
Computer Science Press, 1982

"UML Notation Guide" , version 1.1
Rational Software, 1997
www.rational.com/uml

"UML Semantics", version 1.1
Rational Software 1997
www.rational.com/uml

Muller, Robert J.
"Database Design for Smarties"
Morgan Kaufmann Publishers, 1999.

Blaha, Michael R., William J. Premerlani and James Rumbaugh,
"Relational Database Design using an Object Oriented Methodology"
Communications of the ACM #31, 414-427, 1988

Booch Grady, Rumbaugh James and Ivar Jacobson,
"More than a Notation"
Sofwtare Development, 55-62, Diciembre 1996.

REFERENCIAS

[2.1] Muller, Robert J.,"Database Design for Smarties",Morgan Kaufmann Publishers, 1999, pág. 127

[2.2] Muller, Robert J.,"Database Design for Smarties",Morgan Kaufmann Publishers, 1999, págs. 76-77

[2.3] "UML Semantics", version 1.1,Rational Software 1997, www.rational.com/uml, pág.20

[2.4] Muller, Robert J.,"Database Design for Smarties",Morgan Kaufmann Publishers, 1999, pág. 129

[2.5] "UML Notation Guide", version 1.1,Rational Software 1997, www.rational.com/uml, pág. 13

[2.6] Muller, Robert J.,"Database Design for Smarties",Morgan Kaufmann Publishers, 1999, pág. 139

[2.7] "UML Semantics", version 1.1,Rational Software 1997, www.rational.com/uml, pág.17

[2.8] "UML Semantics", version 1.1,Rational Software 1997, www.rational.com/uml, pág.24

[2.9] Muller, Robert J.,"Database Design for Smarties",Morgan Kaufmann Publishers, 1999, pág. 213

[2.10] Muller, Robert J.,"Database Design for Smarties",Morgan Kaufmann Publishers, 1999, pág. 217

[3.1] Muller, Robert J.,"Database Design for Smarties",Morgan Kaufmann Publishers, 1999, pág. 297

[3.2] Muller, Robert J.,"Database Design for Smarties",Morgan Kaufmann Publishers, 1999, pág. 306

[3.3] Muller, Robert J.,"Database Design for Smarties",Morgan Kaufmann Publishers, 1999, pág. 308