UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Modelado de un sistema de control vehícular

T E S I S

QUE PARA OBTENER EL TÍTULO DE :

ACTUARIO

P R E S E N T A

NOMBRE DEL ALUMNO :

ARMANDO RAMOS AGUILAR

TUTORA :

MARÍA GUADALUPE ELENA IBARGÜENGOÍTIA GONZÁLEZ

2008

AGRADECIMIENTOS

A mi madre por darme la vida que tengo y porque no permitió que se extinguiera la flama del saber.

A mi esposa María de Jesús Díaz Torres y a mis hijos Yazmín Itzel y José Armando Ramos Díaz que sin su amor y cariño solo sería la hoja que mueve el viento.

Al Fís. Efraín López Sansalvador y al LIc. Arnulfo Santander Ávila quienes siempre me han brindado su mano franca, símbolo de una amistad desinteresada.

A mi asesora Mtra. María Elena Guadalupe Ibargüengoítia González por su paciencia, orientación y motivación para llevar a cabo el presente trabajo. No hay palabras para agradecer su honorable labor educativa.

A mi Facultad de Ciencias por darme la oportunidad de usar sus pupitres para escuchar a sus maestros.

Gracias.

Í N D I C E

Cap. 4.- Diseño de bases de datos.

Cap. 5.- Desarrollo de la aplicación.

Cap. 6.- Construcción de bases de datos.

Cap. 7.-Diseño de la Aplicación.

Introducción

Para realizar una aplicación de software intervienen personas, procesos, proyectos y dan como resultado un producto.

En la Ingeniería de software se aplican procesos para el desarrollo de software. Los distintos procesos existentes en cascada, en espiral, etc. están formados por fases; son usados de forma iterativa repitiendo varias veces las partes del proceso hasta completar la solución de una aplicación.

Cuando se realiza una aplicación, sin conocimientos técnicos, la experiencia nos indica actividades que se van adquiriendo con el tiempo : la forma de tomar datos, de ubicar la información, de dividir la aplicación en partes, etc., sin embargo, estas actividades se parecen a lo que propone la Ingeniería de software, que está basada en buenas prácticas de experiencias.

El objetivo del presente trabajo, es desarrollar una aplicación requerida siguiendo un proceso en espiral con las fases de análisis de requerimientos y de diseño implementada con programación estructurada usando herramientas de modelado como es UML.

La Dirección General de Servicios Generales (DGSG) tiene a su cargo, entre otros, la asignación y estado de vehículos de la UNAM. El control de los mismos tiene que ver con movimientos en varios departamentos : Administrativo, de Inventarios, de Taller, etc. que llevan actividades de aseguramiento, verificación vehicular, tenencia, asignación a dependencias, reparación de vehículos en taller, condición del vehículo, etc. Mismas que se comunican por medio de un documento llamado cuadrado en donde se asienta cualquier movimiento que tiene la unidad, desde que la adquiere la UNAM y hasta que se realiza su baja para ser subastado.

Durante el uso de las unidades vehiculares se cuenta con varias situaciones críticas como son :

- Que los movimientos o control están en papel.
- Que no se cuenta con respuesta oportuna a las aseguradoras cuando sufren un evento los vehículos.
- El seguimiento de una unidad vehicular está en papel.
- Se duplica el trabajo cuando se requieren datos de unidades.

Para su control se llevo acabo un sistema que dio respuestas oportunas a éstas y otras problemáticas, como son la comunicación entre las partes, la agilidad de reportes para pagos, mantenimientos, etc.

Se buscó el intercambio de información sólo con las partes involucradas y respaldándola en un solo lugar al que tuvieron acceso los interesados del seguimiento, agilizando el tiempo de respuesta y dándole fluidez y confianza en el manejo de la información que se automatizo. Siendo estas últimas cuestiones críticas en las labores administrativas.

En la automatización se aplicaron herramientas de UML para el modelado del sistema que permitió llevar el control de las unidades identificando primero al personal que

maneja la información. Por último se realizó el planteamiento de la arquitectura y diseño del software y se termina con el pseudocódigo del sistema.

Este trabajo se divide en dos partes. La primera es el marco teórico de la Ingeniería de software comprendiendo los primeros cuatro capítulos. La segunda parte, compuesta por los siguientes tres capítulos, se refiere al desarrollo del sistema de control vehicular.

Finalmente se presentan las conclusiones.

Capítulo 1.-El proceso de desarrollo de software.

1.1. Proceso de desarrollo de software.

La Ingeniería de software se ocupa del estudio del proceso de construir aplicaciones para otros y que satisfacen los requerimientos de funcionalidad y desempeño que solicitan.

Un proceso de software es un conjunto de actividades y resultados asociados para producir un software [Somm].

Los diferentes procesos organizan estas actividades en diversas formas y se describen a diferentes niveles de detalle. Su organización varía en el resultado de cada actividad [Somm].

Con el crecimiento de la computación, el alcance de las aplicaciones de software se ha expandido sin precedentes y dentro de este contexto, la programación es sólo una parte del proceso de construcción de aplicaciones [Brau].

Una aplicación se crea por una necesidad y para que tenga un uso significativo y dada la complejidad de las aplicaciones, se deberán hacer las siguientes actividades [Brau] :

- Entender la naturaleza y el alcance del proyecto.
- Documentar desde el principio el proyecto y crear un plan de trabajo.
- Capturar los requerimientos de la aplicación.
- Llevar a cabo el diseño y la implementación del producto.
- Probar el producto inicial y final.
- Continuar con el mantenimiento del mismo una vez entregado el producto.

El ingeniero de software desempeña roles como implementador de código, documentador de las etapas de desarrollo, ubicación o acotamiento de necesidades y requerimientos, etc.

1.2. Fases del proceso de la ingeniería de software.

Con el tiempo han dominado diferentes tendencias de la ingeniería de software ya sea por el acelerado crecimiento de la computación (p.e. el crecimiento en aplicaciones, el uso de Internet, etc.) o por el florecimiento de nuevas herramientas y paradigmas (p.e. orientación a objetos). Sin embargo, las actividades básicas requeridas para la construcción de software han permanecido estables.

Actividades Básicas :

- El **análisis de requerimientos**, que consiste en reunir y establecer las necesidades del producto y las respuestas (como informes de texto) del software.
- El **diseño**, etapa del software que describe la estructura interna, es decir, la arquitectura y el diseño detallado del producto. Traduce los requerimientos en una representación de software. Se apoya con diagramas.
- La **implementación**, es el uso del lenguaje utilizado para programar el producto.
- La **integración**, es el proceso de ensamblar las partes del todo.

- Las **pruebas**, realiza las pruebas de funcionalidad de las partes y del todo de la aplicación.
- El **mantenimiento**, es el fin del proceso para reparar y modificar la aplicación para que el producto siga siendo útil.

El proceso bajo el que se desarrollan los proyectos es un factor importante para administrar su complejidad [Brau]. Existen varios procesos alternatívos : en cascada, en espiral, por incrementos y unificado.

1.3. Proceso en cascada.

Sin duda el más antiguo, consistente en una secuencia de actividades o etapas de desarrollo en la cual no se debe iniciar una actividad sin antes terminar la anterior.

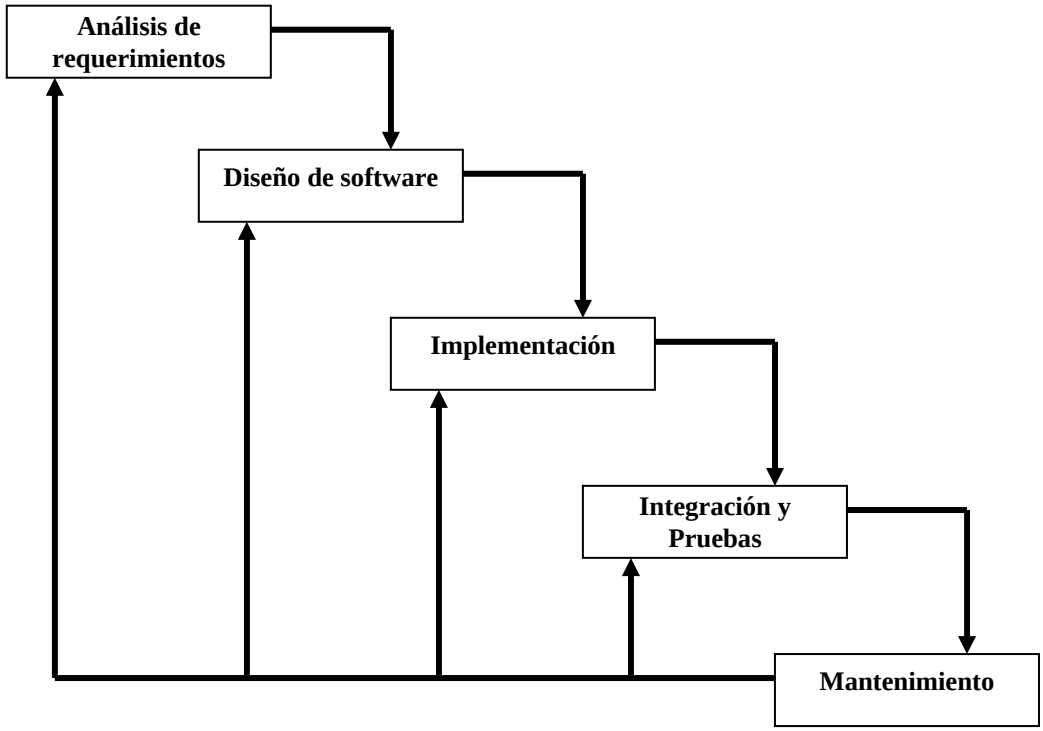


Fig. 1.1. Modelo de cascada (Somm).

La complejidad de aplicaciones conllevan que rara vez se realice un proceso en cascada puro, excepto cuando las aplicaciones son pequeñas o cuando ya se tienen productos muy similares al que se desarrollará [Brau][Somm].

El proceso en cascada es la base o punto de referencia para la mayoría de los procesos.

1.4. Proceso en espiral.

Este proceso reconoce la necesidad de pasar por la secuencia : análisis de requerimientos, diseño, implementación y pruebas más de una vez [Brau][Budg]. Existen razones muy importantes por ejemplo : la necesidad de eliminar riesgos, los prototipos del producto para retroalimentar al cliente, la aparición de nuevas necesidades, ideas no establecidas claramente, etc.

La idea es construir cada versión sobre el resultado de la anterior. Realizando un proceso repetitivo o ciclo cuya forma es una trayectoria en espiral donde se depura el producto [Brau].

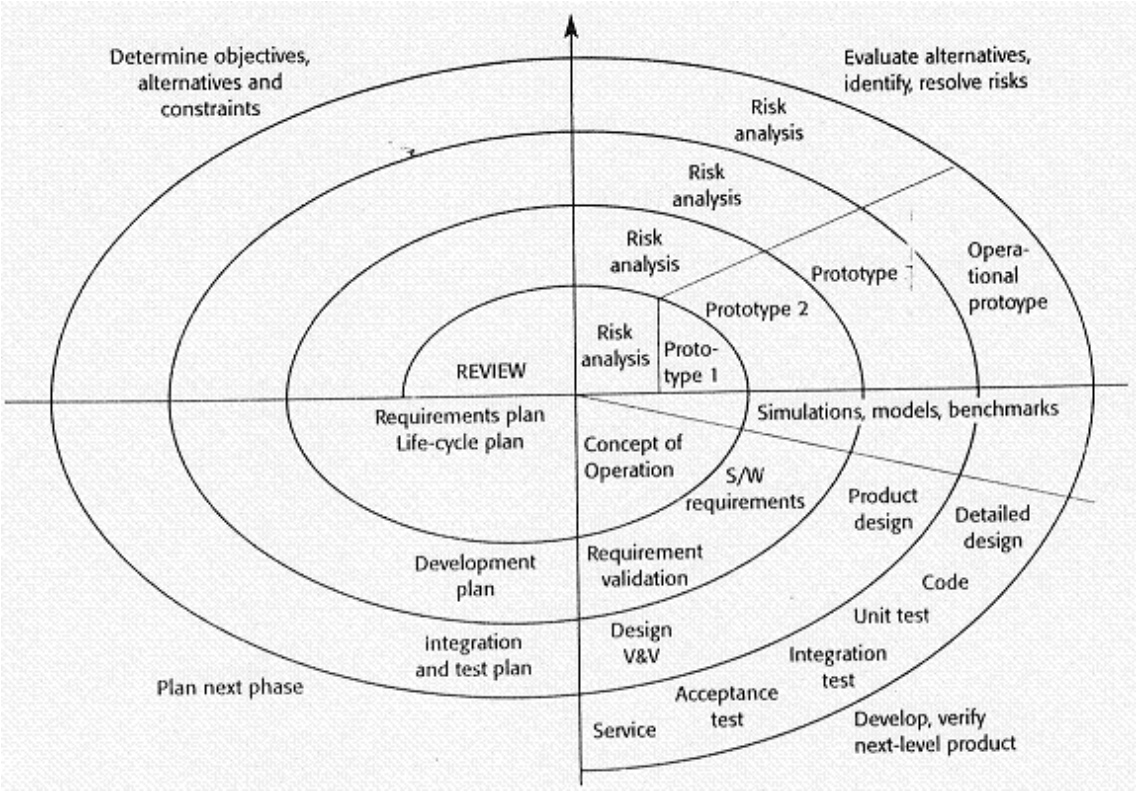


Fig. 1.2. Modelo en espiral de Boehm (1988 IEEE).

Debido a la flexibilidad de sus etapas, casi siempre, se aplica repetidas veces a una aplicación [Somm] .

Cada repetición o ciclo se compone de cuatro sectores [Somm][Brau][Bugd] :

- **Objetivos**.- Determina objetivos para la etapa del proyecto. Se identifican alternativas, estrategias, identifica producto.
- **Riesgos**.- Para cada etapa se identifican riesgos y define la estrategia para reducirlos.
- **Desarrollo y validación**.- Después de evaluar los riesgos, se elije un modelo de desarrollo del sistema.
- **Planeación**.- Se revisa el proyecto y se toma una decisión de continuar con la depuración del siguiente ciclo.

El proceso en espiral se ajusta al avance de los proyectos típicos sin embargo, requiere una administración más cuidadosa que en cascada debido al hecho de que la documentación debe ser consistente siempre que el proceso termina una iteración completa. En particular, el código debe corresponder al diseño documentado y debe satisfacer los requerimientos que se documentan.

La coordinación de la documentación se vuelve una carga especial cuando se pretende optimizar la productividad de un equipo de trabajo que inicia una nueva iteración antes de que haya terminado la anterior.

1.5. Proceso por incrementos.

Cuando las iteraciones producen incrementos en las funciones del producto y cada iteración agrega una nueva capacidad a la anterior, se le dá el nombre de Proceso por incrementos [Somm][Brau].

Este proceso es viable en las últimas etapas del proyecto, cuando el producto está en mantenimiento o cuando el producto propuesto es muy similar al producto desarrollado antes.

A fin de manejar este proceso, la arquitectura debe haberse establecido con claridad y el sistema de documentación debe estar bien sincronizado. El desarrollo por incrementos propone un intervalo de tiempo en el que el proyecto se actualiza y funciona mejor si el incremento n+1 inicia más o menos después que se determina la base del incremento n.

El acercamiento por incrementos fue sugerido como un medio de reducir el trabajo extra en el desarrollo del proceso y da al cliente oportunidad de reajustar decisiones en sus requerimientos detallados [Somm].

1.6. Proceso unificado.

Debido a que los enfoques iterativos repiten todas las partes del proceso en cascada, puede ser complicado describirlos [Brau], es aquí donde este proceso intenta resolver este problema con una clasificación de cuatro iteraciones :

- Iteración de **concepción** : iteración preliminar con los interesados, clientes, usuarios, etc., cuya finalidad es construir un prototipo preliminar que se pueda usar para discutir y determinar el alcance del proyecto. Enfocado al análisis de requerimientos.
- Iteración de **elaboración** : establece la base de la arquitectura y el plan de construcción que se desea y que se necesita. Toma en cuenta análisis de requerimientos, base de diseño e implementación.
- Iteración de **construcción** : da como resultado la capacidad operativa para establecer el producto básico. Confecciona el software a lo largo de una serie de iteraciones produciendo la afinación del prototipo. Su propósito es reducir los riesgos [Fowl]. Abarca diseño e implementación.
- Iteración de **transición** : prepara la aplicación para liberar el producto e incluye implementación y pruebas.

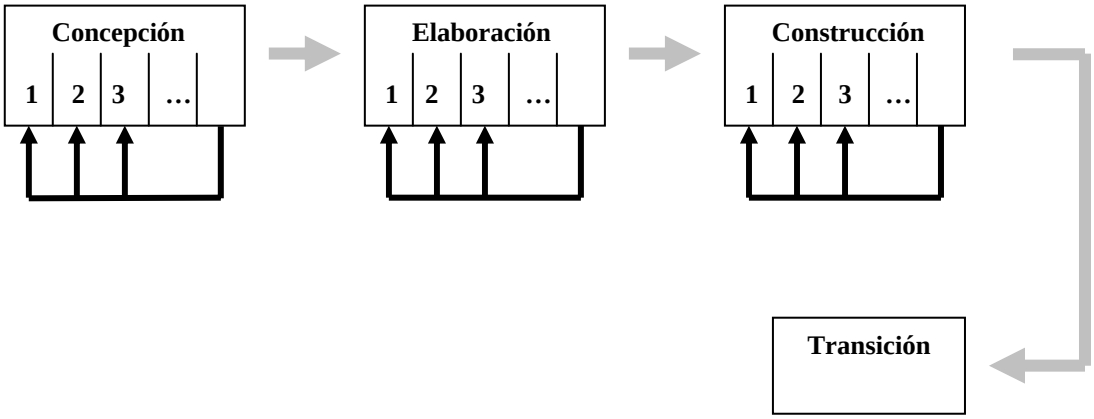


Fig. 1.3. Proceso unificado.

Capítulo 2.-Fase de análisis de requerimientos.

2.1. Introducción.

Las personas que tienen intereses en los resultados de una aplicación se llaman interesados. Como individuo o grupo, constituyen al "cliente" de la aplicación [Brau]. La aplicación es un software desarrollado en algún lenguaje de programación por un "desarrollador". Ya desarrollado el software existe una persona que va a trabajar con el, esta persona es el usuario del sistema que puede ser el cliente u otra persona a quien se le delega la actividad.

Para alcanzar las necesidades y deseos del cliente se deben captar y concretar los requerimientos. Estos requerimientos deben contestar *"qué" se supone debe hacer una aplicación, sin intentar (por lo común), decir el "cómo" lograrlo [Brau]*.

Lo que debe hacer una aplicación es expresada por medio de requerimientos. El requerimiento debe, entre otros puntos [Brau]:

- Expresarse de modo ordenado.
- Ser sencillo.
- Numerarse.
- Tomarse en cuenta en el diseño y el código.
- Probarse aislado y junto con otros requerimientos.

Para un requerimiento pueden existir necesidades a un nivel subsecuente, cuando dicha necesidad se compone de otras más detalladas.

2.2. Objetivos del análisis de requerimientos.

Una aplicación es un software que responde a una necesidad que busca satisfacer a un cliente[Brau][Somm].

El proceso de entender y documentar este software es llamado *análisis de requerimientos.* En el análisis de requerimientos debe haber comunicación entre el cliente y quien lo desarrolla. Otro objetivo es el acuerdo, entre ambas partes, del alcance del software.

El cliente expresa deseos y necesidades de una aplicación, mientras que la forma y estructura específica de la misma dependen del desarrollador.

Definir quién es el interesado principal de los requerimientos se divide en dos niveles:

> El primer nivel llamado "requerimientos C" o del cliente documenta de manera primaria los deseos y necesidades de éste, y observa secundariamente al desarrollador. En el segundo nivel se documenta primariamente los requerimientos de manera específica y estructurada del desarrollador y en forma secundaria al cliente y son llamados "requerimientos D" [Brau].

Los dos niveles representan el análisis de requerimientos de una aplicación.

2.3. Descripción de los requerimientos C.

En la descripción de los requerimientos C surge el "concepto de operación" de una aplicación, que es la visión primaria que tiene el cliente de cómo operará la aplicación para cubrir sus necesidades. El equipo del cliente en su primer contacto tiene visiones distintas que pueden conducir a aplicaciones distintas y es responsabilidad del desarrollador aclarar este concepto [Brau].

Un mapa conceptual para el análisis de requerimientos propone :

- Identificar al "cliente".
- Entrevistar a representantes del cliente.
- Escribir los requerimientos C en forma de documento estándar.
- Construir los casos de uso.
- Inspeccionar los requerimientos C.

No se debe perder de vista que los requerimientos se expresan en una "interación entre la aplicación y el usuario" [Brau].

2.4. Requerimientos D.

La parte del análisis de requerimientos donde se documenta de manera especifica y estructurada la funcionalidad de una aplicación se llama "requerimientos D o del desarrollador" [Brau]. Deben ser consistentes y responder a los requerimientos C o del cliente. Idealmente se inicia escribiendo las pruebas de los requerimientos en el momento en que surgen y, aunque son principalmente del y para el desarrollador, el cliente revisa las pruebas y los requerimientos para ser validados. Cada requerimiento se numera y rastrea durante su implementación.

Para la documentación de los requerimientos D se realizan varias actividades [Brau]:

a) Seleccionar la organización de requerimientos D.
b) Crear diagramas de secuencia a partir de casos de uso.
c) Obtener los requerimientos D a partir de requerimientos del cliente.
d) Inspeccionar.
e) Validar.
f) Liberar con el cliente.

2.4.1. Tipos de Requerimientos D.

Este tipo de requerimientos se clasifica de manera que guíe el desarrollo y el proceso de pruebas [Somm][Brau].

Su clasificación aunque se aplica a ambos tipos de requerimientos, es más significativa al escribir los requerimientos D, porque ayudan a obtener información del cliente acerca de la aplicación general

Requerimientos Funcionales.

Especifica los servicios que debe proporcionar la aplicación o en otras palabras la funcionalidad de la aplicación. El requerimiento : *"La aplicación debe calcular el valor del portafolio de inversión del usuario"* , especifica un servicio al usuario.

Requerimientos no Funcionales.
Son aquellos requerimientos que no especifican un servicio al usuario como :
"La aplicación debe terminar el cálculo del valor actual de un vehículo en menos de un segundo"

Ejemplos de este tipo de requerimientos son :

- **Requerimientos de desempeño**. Especifican restricciones de tiempo que deben tomarse en cuenta en la aplicación, recordando tiempos límite de terminación de la misma. Es una parte crítica del desarrollo porque responde al límite de tiempo para ejecutar la aplicación y se debe tener en cuenta el uso de RAM, el almacenamiento en disco, etc. Un requerimiento de desempeño sería : *"La verificación de existencia de cualquier unidad vehicular no debe ser mayor a 1 minuto"*
- **Confiabilidad y disponibilidad.** Cuantifican lo poco probable de las aplicaciones perfectas, asumiendo un grado de disponibilidad de la aplicación para los usuarios.
  "La aplicación debe estar disponible en todo momento y se deben permitir fallas de aplicación en no más de 2 al mes"
- **Manejo de errores.** Plantea cómo debe responder la aplicación a los errores en su entorno, tomando en cuenta que los usuarios manejan situaciones excepcionales al manejar las aplicaciones, informando de los defectos que no se puedan manejar.
- **Requerimientos de interfaz.** Describen el formato con el que la aplicación se comunica con el usuario. *"Para evitar errores de captura en el tipo de vehiculo, al usuario se le debe desplegar un menú con los tipos de vehiculos comerciales existentes"*.
- **Restricciones.** Las restricciones describen los límites o condiciones para diseñar o implementar la aplicación. Estas restricciones sólo especifican las condiciones que el cliente impone al proyecto o al entorno.

Requerimientos Inversos.

Estos requerimientos establecen lo que no debe hacer el software. Se seleccionan los que aclaran los requerimientos y se eliminan aquellos posibles de malos entendidos.

2.4.2. Cualidades deseables de los requerimientos D.

Las cualidades que deben tener los requerimientos D ayudan a cubrir los detalles faltantes en las aplicaciones. Los requerimientos D en particular, deben ser completos y consistentes, y cada uno debe poderse rastrear en el diseño [Brau].

Rastreo de requerimientos funcionales.

Los requerimientos D deben ser completos y consistentes. Debe haber una correspondencia de cada requerimiento con las partes relevantes del diseño y la implementación (rastreabilidad).

Una manera que ayuda a lograr esto es mapear cada requerimiento con una o más función(es) especifica(s) en el lenguaje de desarrollo.

Un documento utilizado es la matriz de rastreabilidad de requerimientos (pag. 189 [Brau]), en la que un requerimiento etiquetado con un número esta implementado por la acción de una o más funciones. Al avanzar el proyecto, el documento de requerimientos debe mantenerse consistente con el diseño.

Comprobación y no ambigüedad.

Debe ser posible validar un requerimiento cuando se prueba que se ha implementado de manera apropiada. Los requerimientos que se pueden probar se llaman probables y sus opuestos no probables, sin embargo existen requerimientos no probables que se pueden reestructurar para poder comprobarse.

Si un requerimiento es claro entonces no es ambiguo por lo que se puede implementar para que funcione como se espera.

Prioridad.

Para llevar en la práctica una aplicación planeada a tiempo se aplica la técnica de organizar los requerimientos en tres categorías : esenciales, déseables y opcionales.

Las esenciales son las que se incluyen en el proyecto de manera natural, pero las deseables y las opcionales son las que indican hacia donde se dirige la aplicación en el futuro.

Condiciones de error.

Contesta a la pregunta : ¿Qué pasaría si un requerimiento ocurriera en circunstancias equivocadas? Un requerimiento que no está completo no toma en cuenta las condiciones de error. Una falta de condiciones de error en las especificaciones de los requerimientos resalta de manera especial cuando se prueba la función.

Consistencia.

Un conjunto de requerimientos es consistente si no hay contradicciones entre ellos. Cabe resaltar que una aplicación donde crece el número de requerimientos hace más difícil de asegurar la consistencia. La organización de requerimientos D ayuda a evitar inconsistencias mediante la organización por clases.

2.4.3. Métodos de organización de los requerimientos D.

La organización de los requerimientos D tiene como finalidad entenderlos como un evento unitario antes de que crezcan demasiado relacionado uno con otros de manera natural.

Al recabarse los requerimientos, estos van creciendo sin un orden y su lista no organizada presenta algunos problemas como :

- Se pueden volver poco manejables.
- Son difíciles de entender como un evento unitario.
- Que algunos requerimientos van junto con otros relacionados de manera natural.
- Es difícil localizar un requerimiento específico.

Su organización se va desarrollando de acuerdo con varios esquemas, donde tal vez sea recomendable en la organización utilizar una combinación de los mismos :

- **Por característica.** Los requerimientos se agrupan según las características observables por la aplicación. Una de sus desventajas es que no proporciona una organización sistemática puesto que permite saltar de una característica a otra de una aplicación.
- **Por casos de uso.** La idea es que los requerimientos detallados son parte de casos de uso y que muchos requerimientos ocurren de manera natural en secuencias operativas.
- **Por clase.** Los requerimientos se agrupan en clases para organizarlos o para diseñar e implementar la aplicación.
- **Por jerarquía de funciones.** Se descompone la aplicación en un conjunto de funciones de alto nivel y a su vez en subfunciones. Es una manera de poner en orden los requerimientos detallados.
- **Por estado.** Indicando los requerimientos específicos que se aplican a cada estado. Si una aplicación es un proceso, se pueden identificar los eventos que ocasionan los cambios de estados.

Una observación es que él ó los esquemas a utilizarse se relacionan con la arquitectura probable de la aplicación.

2.5. Entrevista.

El proceso de entrevista es la parte medular del análisis de requerimientos y se debe llevar a cabo con cuidado. Se debe tomar en cuenta que el cliente está compuesto por un grupo de personas y que no debe entrevistar a todos al mismo tiempo. Se debe seleccionar a una o dos personas principales (elementos clave) a quienes entrevistar primero, realizar un bosquejo de la aplicación y solicitar comentarios a los otros interesados [Leac].

No es recomendable que el desarrollador se presente solo a una entrevista, ya que por lo regular se pierden puntos que otro puede captar. La entrevista debe pensarse y prepararse con cuidado.

El entrevistador y el cliente deben llegar a una visión conjunta [Brau][Budg]. Para ello se recomienda utilizar herramientas como : casos de uso, diagramas de flujo de datos, diagramas de estado, etc. con la finalidad de ayudar al cliente y al desarrollador en la obtención de los requerimientos.

2.6. Técnicas para el análisis de requerimientos.

No es común que concuerden los interesados en el resultado de la aplicación cuando son vistos por el cliente en forma de grupo ya que cada elemento tiene una visión diferente de lo que va a hacer la aplicación. La tarea es que concuerden hacia un fin específico, ya que los conceptos de lo que se quiere y se necesita todavía están en formación en el cliente cuando el desarrollador inicia su análisis de requerimientos [Brau][Budg]. En muchas ocasiones el cliente confía en que el desarrollador le ayude a aclarar qué quiere, aunque parezca contradictorio.

Cuando hay conflicto de intereses que den como resultado requerimientos inconsistentes, se debe de atacar por medio de una conciliación entre los grupos de interesados si no, se toma el riesgo de que la aplicación quede a la deriva o se cancele [Somm]. El desarrollador puede intervenir en estas situaciones por la responsabilidad profesional que conlleva al manejar el alcance de los requerimientos para que sea factible satisfacerlos dentro de las restricciones de tiempo y presupuesto.

Un punto importante es diferenciar entre los “deseos” y las “necesidades” del cliente. En este punto el cliente puede “querer” que una aplicación realice x o y acción. Pero se puede llegar a un acuerdo y se puede observar desde el punto de vista de requerimientos C o como un elemento de diseño de un requerimiento que realizará el desarrollador (requerimientos D)[Brau][Somm].

Son técnicas para el análisis de requerimientos las siguientes :

2.6.1. Casos de uso.

Un caso de uso se define como la *“interacción entre un actor y una aplicación”* [Brau][Fowl]. Un actor tiene un nombre y el tipo de usuario que es en la aplicación.

Los actores llevan a cabo uno o más casos de uso. A su vez un caso de uso puede ser realizado por más de un actor.

Los casos de uso deben ser explícitos y detallados y pueden expresar diferentes niveles de generalidad por lo que en casos de uso extensos es recomendable descomponerlo en otros casos de uso específicos.

Desde el punto de vista del manejo de un sistema, siempre se debe ubicar quién lo usará, al que se denomina “usuario” y es una entidad externa al sistema que interacturá con él.

El modelado con casos de uso , es una herramienta que documenta el comportamiento de un sistema desde las necesidades o requerimientos tipo “C” del sistema (Fig. 2.0.). Se deben identificar a los usuarios del sistema en particular y las tareas que deben realizar con ayuda del mismo [Brau].

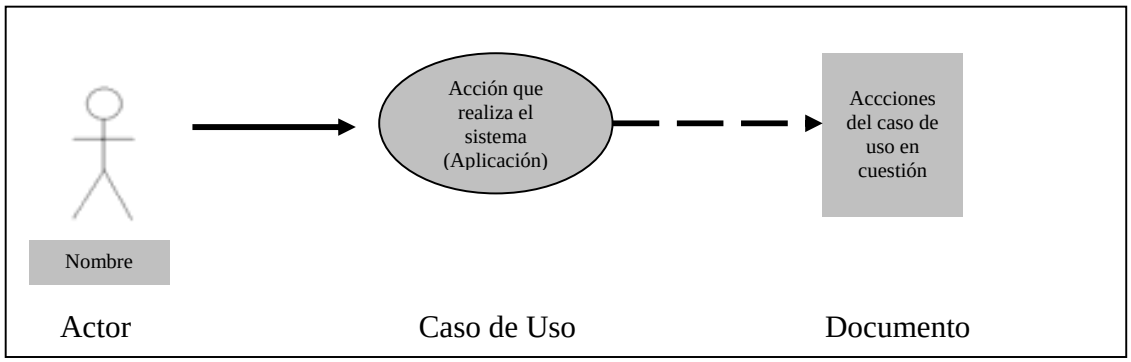


Fig. 2.0. Esquema de caso de uso.

Se construye un diagrama general de casos de uso que identifica las funcionalidades globales y los actores principales del sistema (Fig. 2.1.).

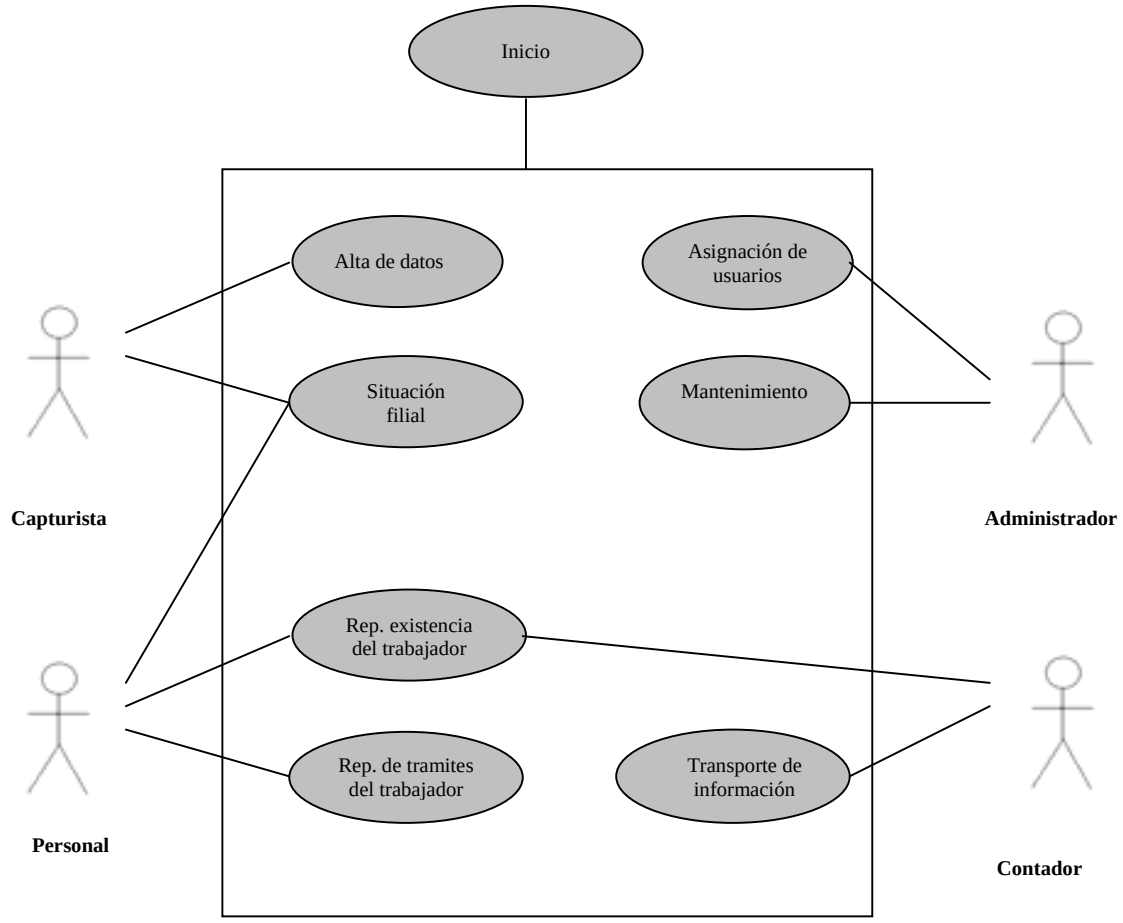


Fig. 2.1. Ejemplo de diagrama general de casos de uso.

La documentación de los casos de uso ayudan a :

- La captura de requerimientos del usuario.
- Planificar la interación que tendrá una actividad con el usuario.
- Tener un modelo del prototipo del sistema.

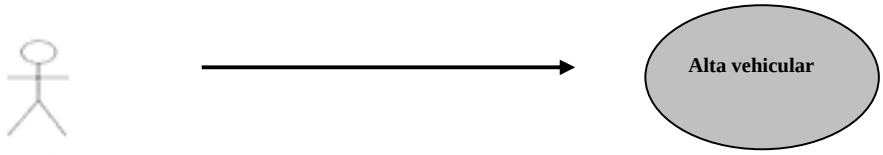
Es decir, ayudan a planificar el desarrollo del sistema basado en los requerimientos "C" y las ideas que se plasmarán en él.

La documentación del caso de uso debe ser sencilla y clara explicando la acción que realizará. Una plantilla para documentar casos de uso es (Parte de la plantilla capitulo 8 y Apéndice C de Frank Armour [Armo]) y (Fig. 2.2.) :

Encabezado.	• El texto "Caso de Uso " seguido del nombre que lo representará. • Un muñeco que represente al actor seguido de una flecha (la interacción con alguna tarea) y un ovalo que represente la tarea o acción a realizar. • El texto "Actor" seguido de su nombre.
Introductivo.	• Descripción : Qué hará el caso de uso. • Precondiciones : Necesidades que tenga el caso de uso para realizar sus acciones.
Flujo de acciones.	• Tendrá la acción del Actor versus la acción del Sistema.
Excepciones.	• Nombre de la(s) excepción(es) y su acción con un identificador referente al flujo de acciones.
Postcondición.	• Son los resultados que se obtendrán del caso de uso.

Por ejemplo si deseamos documentar la alta vehicular del actor capturista, su documento sería:

Caso de Uso : Alta vehicular.



Actor : capturista.

Introductivo.

Descripción :	El Capturista realiza la alta de una unidad vehicular.
Precondiciones :	El Capturista recibe notificación de alta vehicular bajo formato establecido. Se debe proporcionar para filtro de datos : núm. de serie, núm. de motor y/o placa de la unidad vehicular.

Flujo de acciones:

Actor		**Sistema**		
Paso	**Acción**	**Paso**	**Acción**	**Núm. Id**
1	Selecciona Alta vehicular.	2	Solicita datos filtro : placa, núm. de serie y/o núm. de motor.	
3	Escribe datos filtro proporcionados.	4	Realiza filtro de datos proporcionados.	E1
		5	Solicita datos restantes de la unidad vehicular. Proporciona opciones de : Alta vehicular y Cancelar.	
6	Escribe datos restantes del formato proporcionado.			E2
7	Elige "Alta vehicular".	8	Respado de datos de la unidad vehicular.	
		9	Abandona esta ventana y muestra el menú del Capturista.	

Excepciones :

Num. Id	**Nombre**	**Acción**
E1	Unidad existente.	El sistema detecta que ya existe esa unidad, despliega mensaje de existencia y espera datos filtro nuevamente.
E2	Dato incorrecto	El sistema proporciona menú para su elección

Postcondiciones : El capturista ha dado de alta unidad vehicular. El sistema muestra ventana de trabajo del capturista.

Fig. 2.2. Ejemplo de caso de uso.

2.6.2. Diagramas de secuencia.

Estos diagramas muestran el comportamiento y ejecución de los casos de uso [Brau][Fowl], en otras palabras muestra cómo el sistema realizará un caso de uso [Stev].

Se requiere que el caso de uso se piense en términos de objetos [1] y está compuesto de uno o más de ellos. Cada objeto tiene una vida que se representa por medio de una línea vertical con su nombre en la parte superior [Brau][Stev][Fowl].

Los diagramas de secuencia se inician por un usuario(actor) u otro objeto. Ya iniciado el diagrama, cada objeto al inicio se representa por un rectángulo, donde se señala el nombre del objeto seguido de "**:**" y el nombre de la clase que representa. Cuando solo se menciona la clase, indica que no se requieren objetos (p.e. se pueden usar métodos llamados "estáticos" o funciones). Del rectángulo de inicio del objeto se desprende una línea de vida que cuenta con al menos un rectángulo delgado y alargado que denota la ejecución de una función del objeto o clase que comunica el servicio solicitado en el mensaje de interacción entre objetos.

Los objetos se comunican por medio de una interacción que incluyen mensajes. Cada mensaje se representa por medio de una fecha (→) entre las líneas de vida de los objetos y es etiquetado por el nombre del mensaje. Esta interacción inicia el servicio al objeto que la proporciona al final de la flecha. El orden en que se dan las interacciones entre objetos es de arriba hacia abajo del diagrama.

Los mensaje que llevan información de control se muestra entre corchetes e indican cuando se envía un mensaje por medio de una condición o si el mensaje se envía a varios objetos receptores. La interacción incluyen una flecha (←) que indica el regreso del mensaje o indica uno nuevo. Se recomienda usarlos solo cuando aumentan la claridad del diagrama [Fowl].

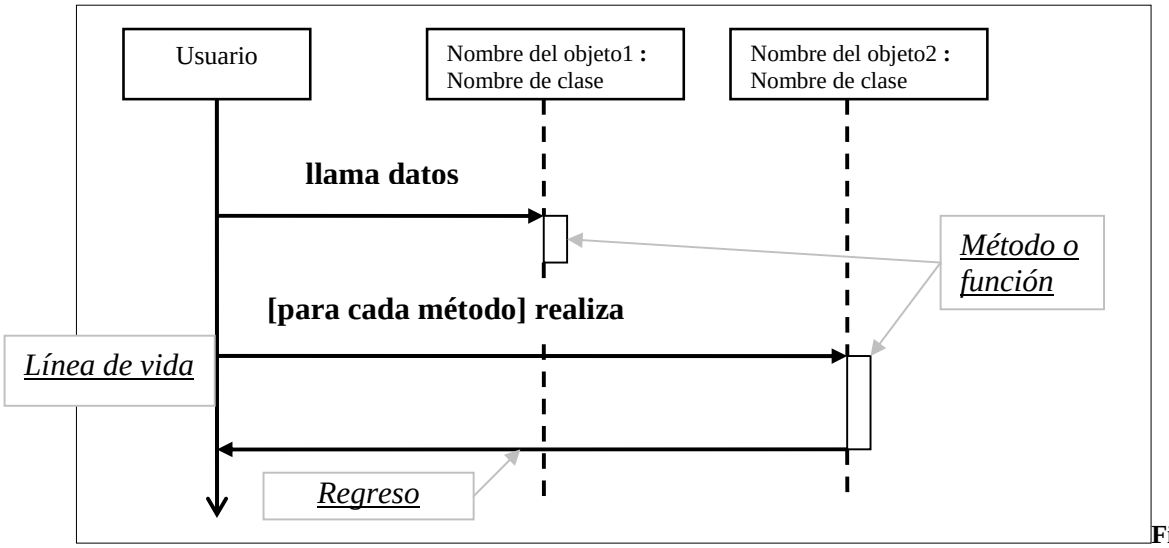


Fig. 2.3. Diagrama de secuencia.

[1] Un objeto es una cosa con la que se puede interactuar, se le puede enviar varios mensajes y éste reaccionar ente ellos.

2.6.3. Diagramas de estados.

Los diagramas de estados son una técnica para describir el comportamiento de un sistema [Fowl]. Un estado es la situación o condición actual en el que se encuentra un sistema [Brau].

Una aplicación de software se debe dividir en estados de manera que siempre esté justo en uno de ellos [Brau].

Este tipo de diagrama describe todos los estados posibles en los que puede entrar un objeto particular y la manera en que cambia el estado del objeto, como resultado de los eventos[2] que llegan a él, mostrando su comportamiento durante todo su ciclo de vida [Fowl].

Se inicia con una marca de creación, que es *un punto negro* con una flecha apuntando al estado inicial del diagrama. Este tipo de diagramas también cuenta con :

- **Estados** .- son cajas con esquinas redondeadas y que denotan la acción que se llevara a cabo en el estado mismo.
- **Actividades** .- muestra que actividad realizará el estado del objeto y se ubica en el estado mismo.
- **Transiciones** .- aparecen como flechas e indica la delegación o acción que realizara el estado.
- **Eventos** .- va en la flecha de transición por medio de mensajes.

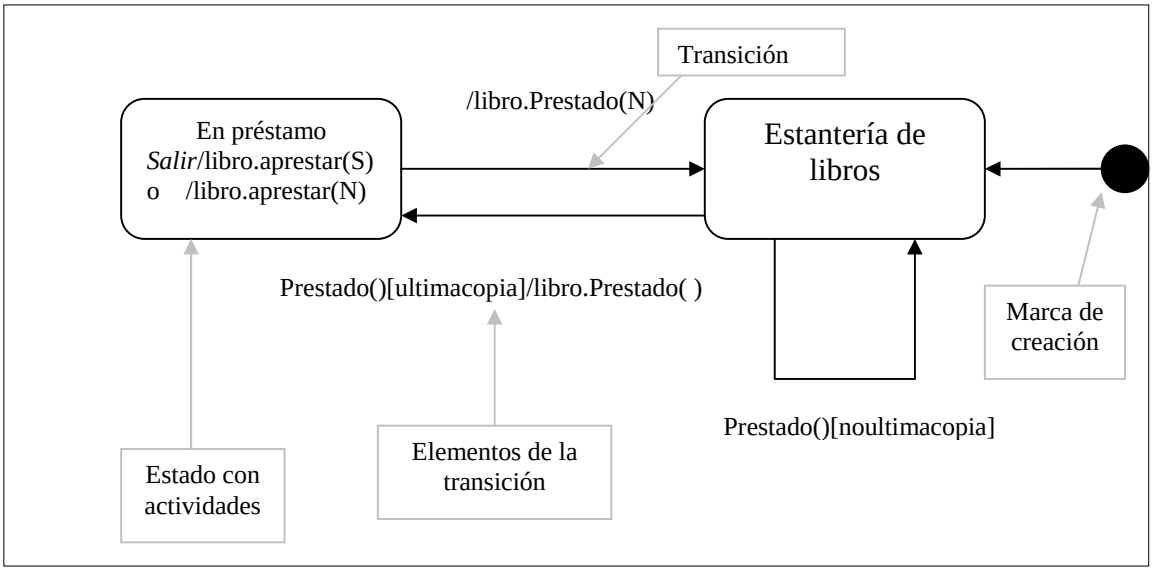


Fig. 2.4. Diagrama de estados.

[2] Un evento es una acción que provoca una transición o cambio entre estados [Scha].

2.6.4. Diagramas de flujo de datos.

Estos diagramas deben describir de forma natural el flujo de información entre los datos y sus procesos.

Se ocupan principalmente para describir el flujo sin mostrar el control (no especifica que función ocurre primero en los procesos).

Se ocupan flechas para denotar flujos de datos, círculos o rectángulos para unidades de un proceso y un par de líneas para indicar el lugar donde residen los datos (Fig. 2.5.).

Se aplica comúnmente en aplicaciones manejadas por eventos y se agregan las transiciones entre ellos por medio de fechas, mientras que los estados son rectángulos.

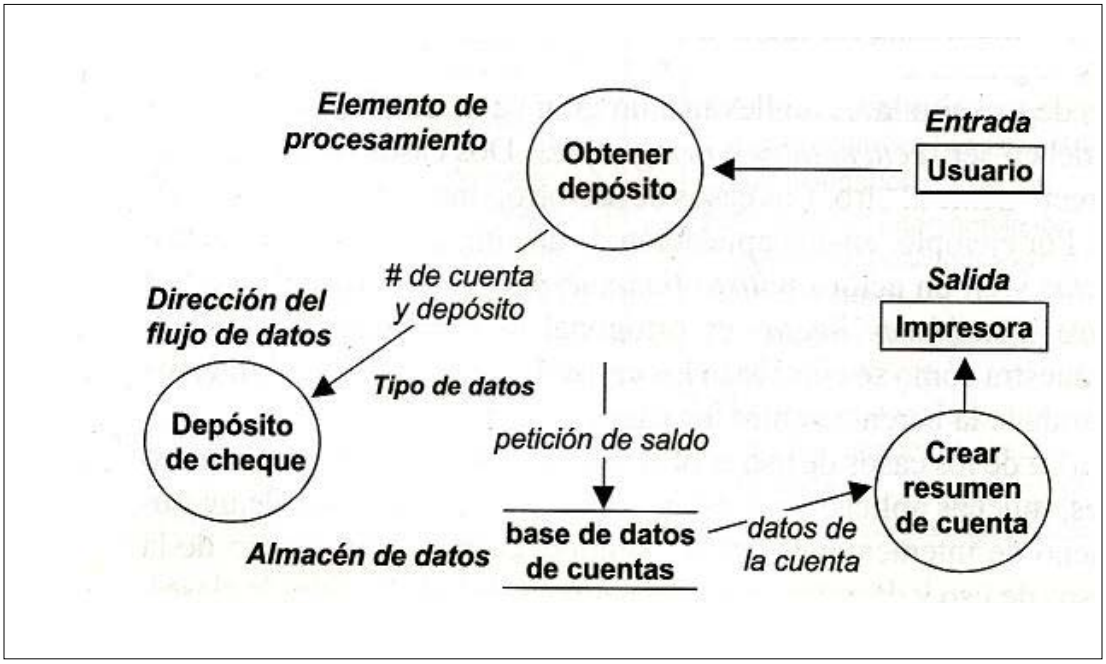


Fig. 2.5. Diagrama de flujo de datos [Brau] pág. 148.

2.7. Diseño preliminar de interfaces de usuario.

Los clientes comprenden una aplicación al visualizar su interfaz gráfica de usuario, por lo que una buena manera de ayudar a describir la aplicación es desarrollar el diseño preliminar [Brau].

Pasos para desarrollar interfaces de usuario [3]:

- **Paso 1. Conocer al usuario**. Para determinar la naturaleza de la interfaz de usuario es necesario conocer a los posibles usuarios del sistema. Para esto se

[3] Pág. 151, cap. 3 [Brau].

puede aplicar una encuesta o cuestionario que identifique el nivel de conocimiento y experiencia; las características físicas y psicológicas del usuario; y las tareas y trabajos del usuario. El usuario con menor nivel educativo, capacitación, aptitudes y motivación requiere mayor sencillez, más explicaciones y más ayuda.

- **Paso 2. Comprensión de la función de negocios.** Este paso requiere entender el propósito de la interfaz del usuario en términos del propósito global de la aplicación. Es diferente una interfaz de usuario cuya aplicación sea para inventarios de almacén que para inversiones.

- **Paso 3. Entender los principios del buen diseño de pantallas.** Se toman factores que con frecuencia se aplican al proceso de hacer una interfaz atractiva. Por ejemplo : agrupar información, asegurar consistencia, mantener simetría, mantener un balance visual y guardar una proporción.

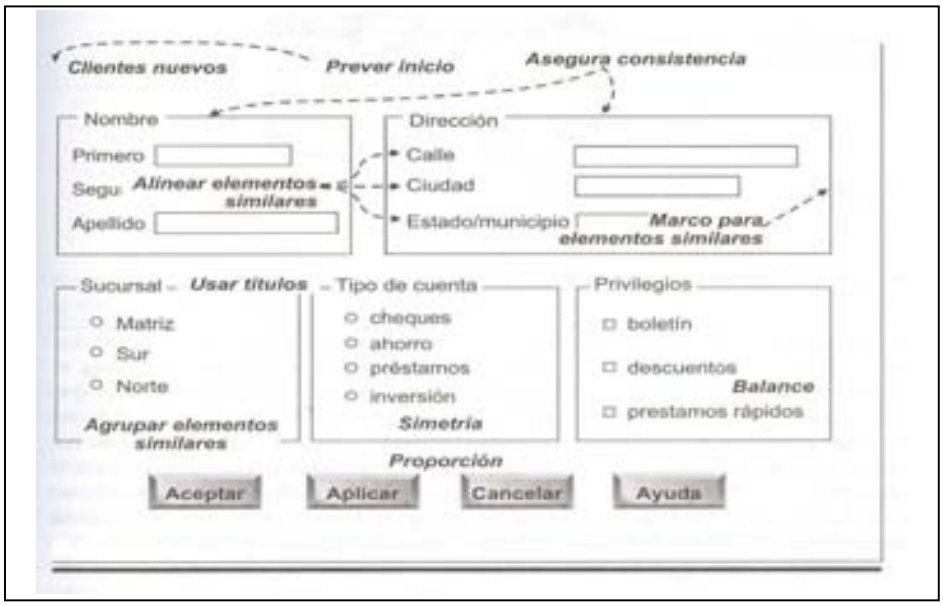


Fig. 2.6. Agrupamiento de información.

- **Paso 4. Seleccionar el tipo adecuado de ventanas.** Su propósito es manejar ventanas específicas para el manejo de información ayudando al usuario y evitando errores de captura de información. Por ejemplo : desplegar una ventana de propiedades de la entidad; obtener información adicional para realizar una tarea; proporcionar información; presentar un conjunto de controles; o ampliar información.

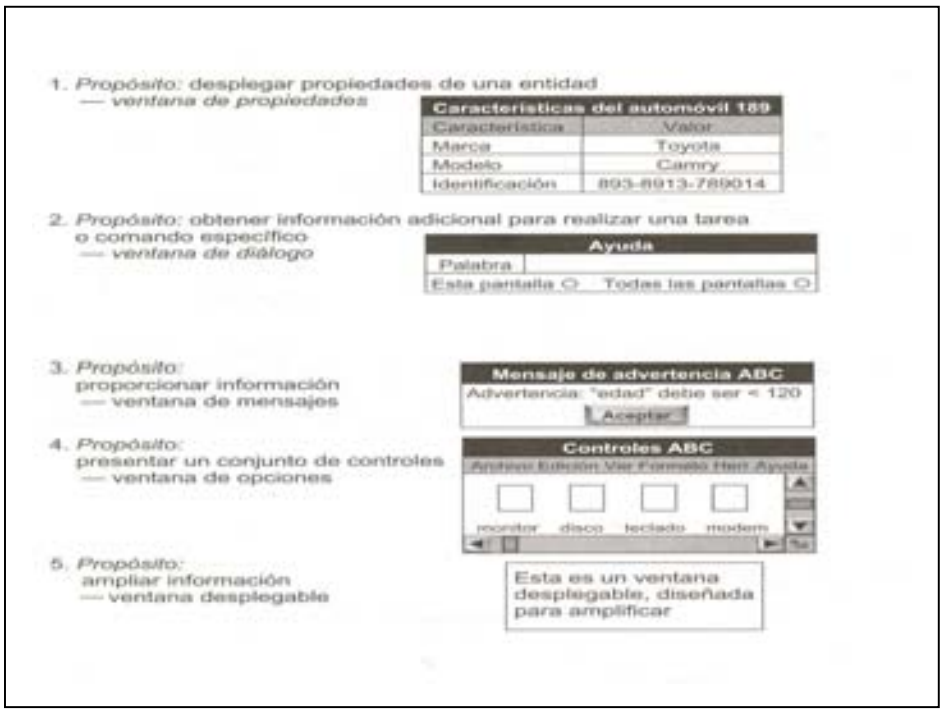


Fig. 2.7. Selección de ventanas.

- **Paso 5. Desarrollar menús del sistema.** Los usuarios requieren una ayuda estable para las aplicaciones. Esto puede proporcionarse con un menú principal constante y no debe contar con muchos elementos.

- **Paso 6. Seleccionar los controles basados en dispositivos adecuados.** Estos controles son los medios físicos por los que los usuarios comunican sus deseos respecto de la aplicación. Se incluyen joysticks, trackballs, tabletas de gráficos, pantallas de contacto, ratones, micrófonos y teclados.

- **Paso 7. Seleccionar los controles de pantalla.** Son controles en pantalla como iconos, botones, cajas de texto, botones de selección o cuadros de activación. Deben tener consistencia, simetría y balance.

- **Paso 8. Organizar y distribuir pantallas.** Se busca como se puede visualizar de la mejor manera posible la información por pantallas. Pueden ser pantallas superpuestas, en mosaico o cascada (Fig. 2.8.).

- **Paso 9. Elegir los colores adecuados.** Se usan colores que puedan resaltar las pantallas con buen gusto. Una forma es observar los colores utilizados por procesadores o editores ya establecidos. Se recomienda simetría en los colores usados para marcos, menús, datos, fondos de pantalla, etc.

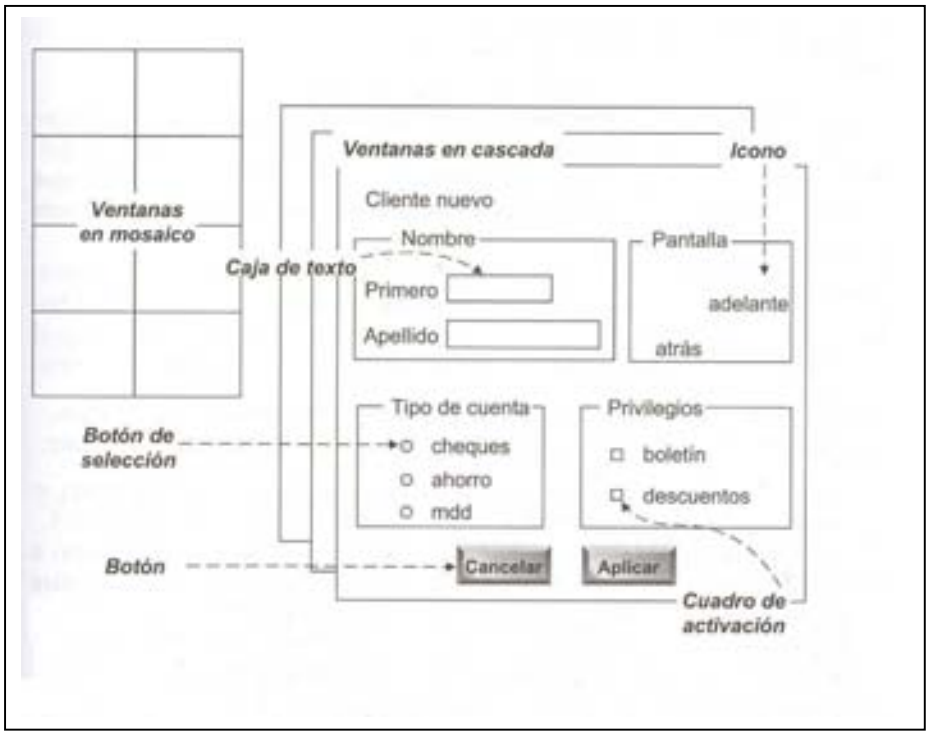


Fig. 2.8. Organización y distribución de pantallas.

- **Paso 10. Crear iconos significativos.** Estos iconos ayudan a la interfaces y la muestra o solicitud de información en pantallas.

- **Paso 11. Proporcionar mensajes, retroalimentación y guía efectivos.** El usuario debe tener una guía de acción por medio de mensajes para el uso de la aplicación. Debe ser concisa, clara y apropiada al lenguaje del usuario. Es posible una diálogo de ayuda sobre que hacer en ciertos casos.

Capítulo 3.- Fase de diseño.

3.1. Objetivos.

Una vez que se han establecido los requerimientos de software, la fase de desarrollo comprende dos pasos : diseño arquitectónico y el diseño detallado. Cada paso transforma la información de forma que finalmente se obtiene un software validado por un usuario.

En la primera etapa, los requerimientos del programa (manifestados por el dominio de la información en lo requerimientos “C” y “D”)[Brau], alimentan el paso del diseño. Usando una de las distintas metodologías del diseño se realiza : el diseño de datos (se enfoca sobre la definición de la estructura de datos), el diseño arquitectónico (define las relaciones entre los principales elementos estructurales del programa) y el diseño detallado (transforma los elementos estructurales en una descripción algorítmica del código).

Enseguida se integra y valida el software codificándolo y se generan módulos del programa para finalmente poner a prueba el código del software.

La fase del diseño absorbe tres cuartas partes del proceso del software y es aquí donde se toman decisiones que afectarán finalmente el éxito de la implementación del programa y su facilidad con que el programa será mantenido. Si el diseño genera un sistema inestable, este fallará cuando se realicen cambios pequeños.

Para evaluar la calidad del diseño se establecen criterios para un buen diseño [Pres] :

- Exhibir una organización jerárquica que haga un uso inteligente del control entre los elementos del software.
- Ser modular; esto es, el software debe estar particionado lógicamente en elementos que realicen funciones y subfunciones específicas.
- Contener una representación distinta y separable de los datos y los procedimientos.
- Conducir a módulos que exhiban características funcionales independientes.
- Derivarse usando un método repetible (en cascada, en espiral, etc.) que esté conducido por la información obtenida durante el análisis de requerimientos del software.

El diseño del software se realiza en dos pasos. El diseño arquitectónico se refiere a la transformación de los requerimientos en datos y arquitectura del software. En la segunda, el diseño en detalle se enfoca hacia los refinamientos que conducen a una estructura de datos detallada y a representaciones algorítmicas del software. Termina con un bosquejo completo para la etapa de la programación [Pres][Brau].

3.2. Arquitectura.

La arquitectura de software consiste en un proceso de partición, que relaciona los elementos de una solución del software, con partes de un problema del mundo real

definido implícitamente durante el análisis de requerimientos [Brau][Pres]. Es decir divide la aplicación para diseñarse e implementarse en partes (módulos) y después ensamblarse.

Para cada desarrollo de software pueden existir varias arquitecturas adecuadas a elegir, decidir cual es la mejor depende de las metas. Suele ser difícil satisfacer todas las metas, ya que un diseño satisface alguna y otras no.

Las metas más importantes que deben satisfacerse en una arquitectura son [Brau] :

- ***Extensión.***-Describe el grado en el que se desea introducir nuevas características de la aplicación, debe facilitar la adición de estas últimas implicando la introducción de más abstracción en el proceso.
- ***Cambio.***-Facilita los cambios en los requerimientos durante el proceso, sin perder de vista que diseñar para un cambio es diferente que diseñar para la extensión. Aunque las técnicas sean similares.
- ***Sencillez.***-Es la meta con mayor dificultad de satisfacer. En el proceso se busca la fácil comprensión e implementación del mismo, es decir, la arquitectura de diseño que permita extensiones y cambios.
- ***Eficiencia.***-Referente al uso del CPU y memoria. Lograr alta velocidad en ejecución.

3.2.1. Arquitectura de capas.

Una capa de una arquitectura es una colección coherente de artefactos de software, casi siempre un paquete de clases. En su forma más común, una capa usa cuando mucho otra capa y es usada por otra capa. Construir aplicaciones capa por capa puede simplificar mucho el proceso. Algunas capas, como marcos de trabajo, pueden servir para varias aplicaciones. Un ejemplo común de capas es la arquitectura cliente-servidor.

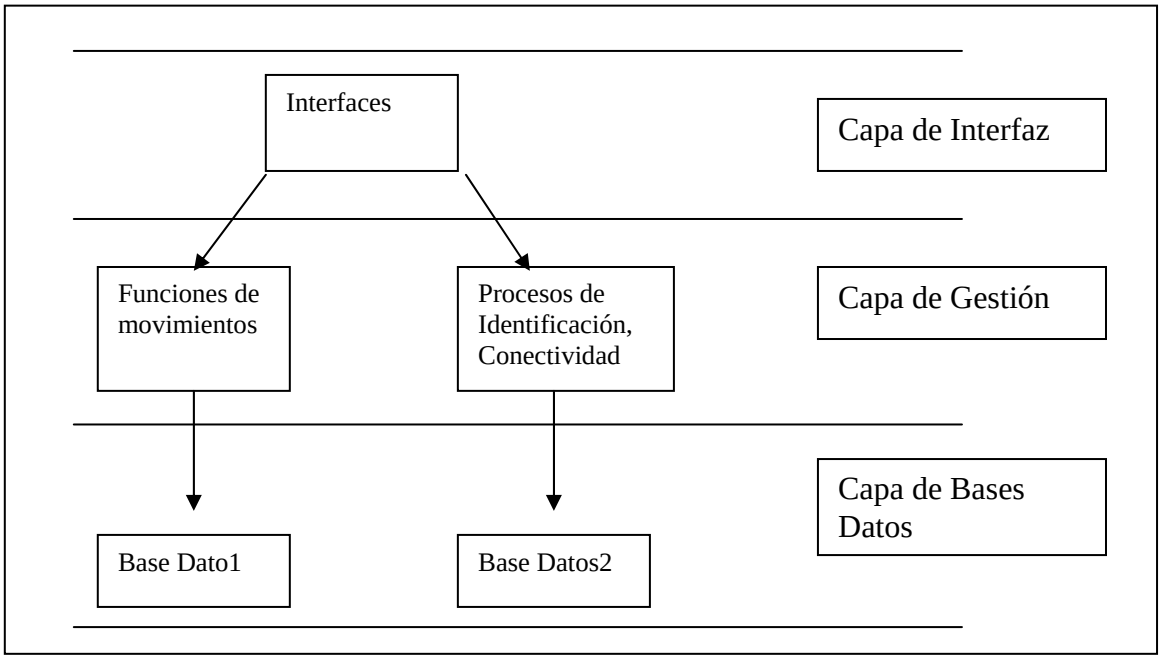


Fig. 3.1. Arquitectura de capas.

3.2.2. Módulos.

Un sistema se debe pensar como un conjunto de módulos e identificar dependencias entre ellos. Un módulo es cualquier elemento del sistema identificable y que tiene sentido por sí mismo [Stev][Brau]. Como módulos podemos nombrar :

- Rutinas.
- Bibliotecas de funciones.
- Programas independientes.

No todos los módulos son iguales pero un sistema desarrollado por módulos debería tener un mantenimiento sencillo y confiable en lo posible [Stev].

Las aplicaciones modulares tienen el potencial de reuso en sus partes. Una aplicación es modular cuando es sencillo identificar y reemplazar sus partes [Brau]. Permite, aplicando un lenguaje de programación, dar mejor soporte y mantenimiento del mismo.

3.2.3. Componentes.

Los componentes son entidades de reutilización y sustitución que no requieren conocimiento del software que las usa [Brau][Stev].

Los componentes son entidades que complementan las necesidades del sistema en su totalidad; al momento de construirlos son conectables y compatibles uno con otros.

Hay varios tipos de componentes, cada uno con su correspondiente tipo de dependencia. En el proceso de compilación de una aplicación, serían :

- Código fuente cuando el código de una clase que depende de otro componente esté disponible para la compilación.

- Código binario cuando existe una biblioteca que depende de cualquier código objeto con el que se tiene que enlazar para formar un programa ejecutable.

- Una aplicación ejecutable que puede depender de otros programas para interactuar con ella en tiempo de ejecución.

Los componentes pueden depender unos de otros; estas dependencias se representan utilizando fechas punteadas.

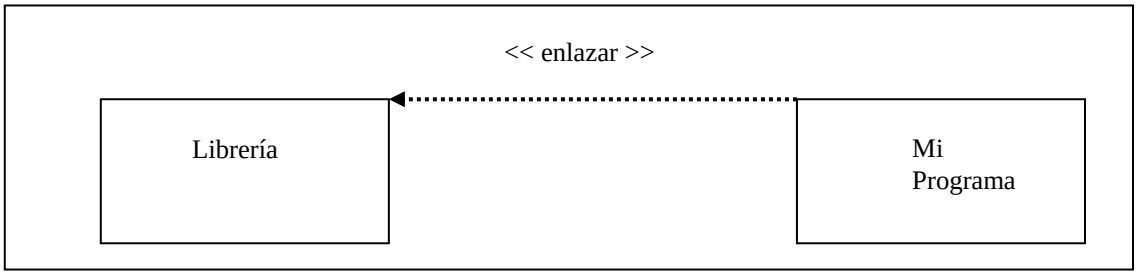


Fig. 3.2. Componentes.

3.2.4. Subsistemas.

Cuando los sistemas son grandes se recomienda dividirlos en subsistemas más pequeños y manejables con la implementación de componentes [Scha][Stev].

Las razones por las que se deciden los subsistemas son que es mas fácil implementar subsistemas pequeños que un sistema grande y, segundo, si los subsistemas que se van a implementar son independientes, pueden implementarse en los equipos que trabajan en paralelo [Scha].

Los subsistemas tiene una parte de especificación y otra de realización. En la primera se incluyen los casos de uso, que describen las operaciones que pueden hacerse con el subsistema sin revelar nada de la estructura del mismo; un subsistema incluye interfaces. En el segundo se incluyen clases y otros subsistemas, proporcionando la funcionalidad de los casos de uso [Stev].

Existe una correspondencia entre los componentes y los subsistemas, ya que un componente es una implementación según el diseño dado por la parte de realización de un subsistema.

3.3. Fundamentos del diseño.

Durante el tiempo se han establecido un conjunto de conceptos fundamentales en el diseño de software [Pres] .

Refinamiento. Se comienza con una declaración de la función (o descripción de la información) que es definida a un nivel superior de abstracción. La declaración describe la función o información conceptualmente, pero no da información sobre el funcionamiento interno de la función o la estructura interna de la información. El refinamiento da más y más detalles conforme se producen los sucesivos refinamientos.

Arquitectura de software. Se deriva mediante un proceso de partición, que relaciona a los elementos de una solución del software, con partes de un problema del mundo real definido implícitamente durante el análisis de requerimientos. La solución ocurre cuando cada parte del problema se resuelve mediante uno o más elementos del software. Una metodología de diseño de software puede usarse para derivar estructuras, pero debido a que cada una se basa en un concepto fundamental diferente, cada método de diseño dará como resultado una estructura diferente, para el mismo conjunto de requerimientos del software.

Estructura del programa. Representa la organización (frecuentemente jerárquica) de los componentes del programa (módulos) e implica una jerárquica de control. No representa aspectos procedimentales del software, tales como secuencia de procesos, ocurrencia de decisiones o repetición de operaciones. Es frecuente mostrar un diagrama de estructura como un diagrama en árbol descendiente donde se identifican profundidad y anchura, y abanico de salida y de entrada para los módulos que forman el árbol.

Estructura de datos. Es una representación lógica entre elementos individuales de datos. Debido a que la estructura de la información afectará invariablemente al diseño procedimental final, la estructura de datos es tan importante como la estructura del programa en la representación de la arquitectura de software. La estructura de datos dicta la organización, métodos de acceso, grado de asociatividad y alternativas de procesamiento para la información.

Procedimientos de software. Se enfoca sobre los detalles de procesamiento de cada módulo individualmente. El procesamiento debe dar una especificación precisa del procesamiento, incluyendo secuencia de sucesos, puntos de decisiones exactos, operaciones repetitivas e incluso organización/estructura de los datos. El procesamiento indicado por cada módulo debe incluir una referencia a todos los módulos subordinados del módulo que se describe.

Modularidad. Es el atributo más sencillo del software que permite a un programa ser manejable intelectualmente. El software se divide en elementos con nombres y direcciones separadas, llamados módulos que se integran para satisfacer los requerimientos del problema. Se debe tener cuidado al modularizar y evitarse una gran o pequeña modularidad, buscando que el tamaño de un módulo dependa de su función y aplicación [Brau].

Abstracción. Cuando se considera una solución modular a cualquier problema, pueden formularse muchos niveles de abstracción. En el nivel superior de abstracción, se establece una solución en términos amplios usando el lenguaje del entorno del problema. En los niveles inferiores de abstracción se toma una orientación más procedimental. La terminología orientada al problema se acompaña con una terminología orientada a la implementación, en un esfuerzo para establecer una solución. Finalmente, en el nivel más bajo de abstracción, se establece la solución de forma que pueda implementarse directamente. Cada paso de un proceso de ingeriería de software es un refinamiento del nivel de abstracción de la solución del software. Durante el análisis de los requerimientos del software, se establece la solución en términos de lo que es familiar al entorno del problema. Conforme nos movemos desde lo preliminar al diseño de detalles, se reduce el nivel de abstracción. Finalmente, se alcanza el nivel más bajo de abstracción cuando se genera el código fuente.

Ocultamiento de información. El principio de ocultación de información, sugiere que los módulos se caractericen por decisiones de diseño que oculten detalles de unos a otros. En otras palabras, los módulos deben especificarse y diseñarse de forma que la información (procedimientos y datos) contenida dentro de un módulo sea inaccesible con otros módulos que no necesiten tal información. La ocultación implica que una modularidad efectiva puede lograrse definiendo un conjunto de módulos independientes, que se comuniquen con otros sólo por la información necesaria para que se realice la función de software. El uso de ocultación de la información presenta ventajas cuando es necesario realizar modificaciones durante las pruebas y el mantenimiento del software.

3.3.1. Independencia funcional.

La independencia funcional se adquiere desarrollando módulos con una clara función, de tal forma que cada módulo se enfoque a una subfunción específica de requerimientos

y tenga una interfaz sencilla cuando se ve desde otras partes de la estructura del software.

El software con una efectiva modularidad, es decir, módulos independientes, es fácil de desarrollar porque su función puede ser compartida y las interfaces son fáciles de mantener y probar [Brau][Stev].

La independencia se mide usando dos criterios cualitativos : cohesión (medida de la fuerza funcional relativa a un módulo) y acoplamiento (medida de la interdependencia relativa entre módulos) [Brau][Pres].

Cohesión. La cohesión es una extensión del concepto de ocultación de la información. Un módulo cohesivo ejecuta una tarea sencilla en un procedimiento y requiere poca interacción con procedimientos que se ejecutan en otras partes de un programa.

Siempre se busca conseguir una gran cohesión aunque un punto medio es frecuentemente aceptable. La escala de la cohesión no es lineal. Esto es, una cohesión baja es mucho peor que la del rango medio, la cual es casi tan buena como una alta cohesión. El diseñador debe evitar las cohesiones de bajo nivel al diseñar módulos.

Los niveles moderados de cohesión están relativamente cercanos a otros en su grado de independencia modular. Cuando los elementos de procesamiento de un módulo están relacionados y deben ejecutarse en un orden específico, existe cohesión procedimental. Cuando todos los elementos de procesamiento se concentran sobre un área de una estructura de datos, se presenta una cohesión de comunicación. Una alta cohesión se caracteriza por un módulo que ejecute una tarea procedimental distinta. Se busca una cohesión alta y reconocer la cohesión baja de forma que el diseño de software pueda modificarse para que contenga una mayor independencia funcional.

Acoplamiento. El acoplamiento es una medida de la interconexión entre módulos en la estructura del programa. La interfaz entre módulos es el punto en el que se hace una entrada o referencia a un módulo y los datos que pasan a través de la interfaz.

En el diseño de software, se busca el más bajo acoplamiento posible. La conectividad sencilla entre módulos da como resultado un software que es más fácil de comprender y menos propenso a la propagación de errores. A niveles moderados, el acoplamiento se caracteriza por el paso de control entre módulos. Por otro lado, niveles relativamente altos de acoplamiento se producen cuando los módulos están ligados a un entorno externo al software.

El acoplamiento externo (módulos a dispositivos o formatos) es esencial, pero debe limitarse a un pequeño número dentro de una estructura. Se presenta también un alto acoplamiento cuando varios módulos referencian a un área de datos global. Estos modos de acoplamiento se presentan debido a decisiones de diseño que se hacen cuando se desarrolla la estructura y deben evitarse [Pres][Stev].

La modulación efectiva se logra al maximizar la cohesión y minimizar el acoplamiento haciendo posible descomponer tareas complejas en otras más sencillas [Brau].

Durante la necesidad de modificaciones de las aplicaciones, es más sencillo modificar las arquitecturas de bajo acoplamiento y alta cohesión, ya que los cambios tienden a tener efectos locales sobre ellos.

3.4. Herramientas del diseño.

Las herramientas del diseño proporcionan una guía de pasos de lo que debe hacerse para llevar a cabo un diseño detallado. Proporciona funciones específicas y describe la correspondencia de datos a funciones.

3.4.1. Diagramas de secuencia detallados.

En esta parte a los diagramas de secuencia obtenidos a partir de los casos de uso se les proporcionarán las clases involucradas con los métodos requeridos para ejecutar las secuencias.

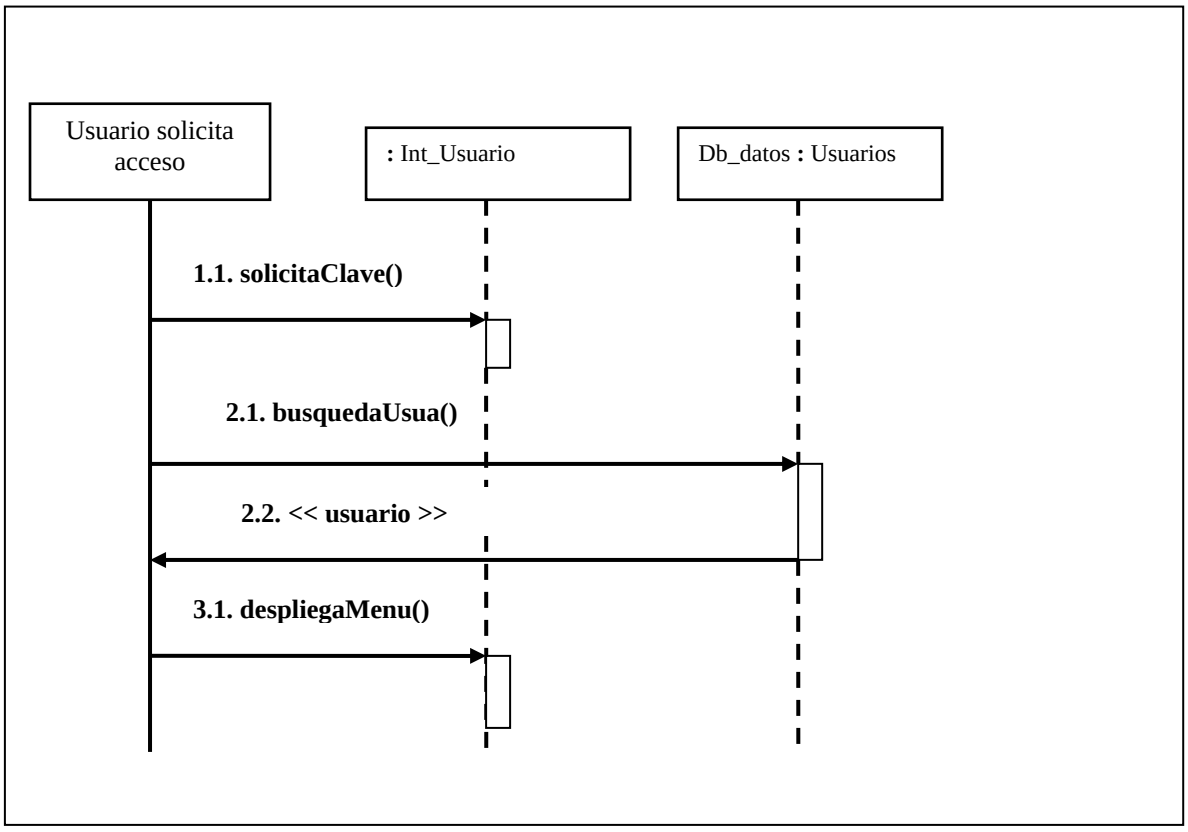


Fig. 3.3. Diagrama de secuencia para solicitar acceso.

Este refinamiento se puede apoyar en tres pasos [Brau] :

1.- Comenzar con los diagramas de secuencia construidos para los requerimientos detallados o la arquitectura correspondiente a los casos de uso.
2.- Introducir casos de uso adicionales, para describir cómo interactúan las partes del diseño con el resto de la aplicación.

3.- Proporcionar diagramas de secuencia con detalles completos.

- Especificar las funciones o métodos exactos y sus clases.
- Seleccionar los nombres de las funciones específicas en lugar del lenguaje natural.

Al terminar se conocen las funciones o métodos requeridos para ejecutar el caso de uso en cuestión y se pueden indicar en el modelo del objeto.

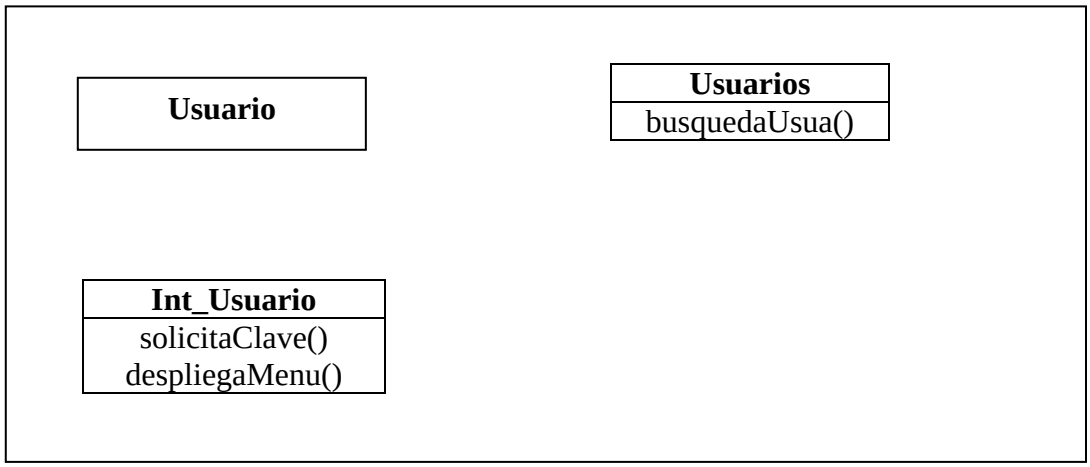


Fig. 3.4. Funciones del caso de uso acceso.

3.4.2. Diagramas de flujo de datos detallados.

Describen con detalle la relación entre funciones y datos. Se debe llegar al nivel más bajo de procesamiento ya que puede generar otras funciones a la relación inicial.

El refinamiento de los diagramas de flujo de datos se puede llevar a cabo siguiendo los pasos :
1.- Colectando los diagramas de flujo de datos construidos para los requerimientos.
2.- Introducir diagramas de flujo de datos para explicar los flujos de datos y el procesamiento.
3.- Proporcionar todos los detalles de los diagramas de flujo de datos.

- Indicar la naturaleza del procesamiento en cada nodo.
- Indicar el tipo de datos transmitidos.
- Expandir los nodos de procesamiento en el diagrama de flujo de datos si la descripción del proceso requiere más detalle.

3.4.3. Algoritmos.

Identificadas las funciones que se van a realizar bajo algún lenguaje de programación, se necesita describir el proceso por medio de un algoritmo [1] que se usará de manera que el código fuente se apegue a este. La ventaja de realizar este paso sirve para detectar los defectos antes de que se conviertan en defectos de código.

[1] Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

3.4.3.1. El pseudocódigo.

Un pseudocódigo es un medio para expresar un algoritmo como texto sin tener que especificar los detalles del lenguaje de programación [Brau].

El pseudocódigo para un método debe verificarse contra los requerimientos correspondientes. Definiendo las funciones, pasos o procesos que se implementarán para describir una solución del requerimiento.

Las características a cumplir son [Pres] :

- Debe ser fácil de entenderlo y es parecido al lenguaje que posteriormente se utilizará para codificarlo.
- Es preciso para representar la solución de algoritmos de forma detallada.
- Es independiente del lenguaje de programación a utilizar.

Un requerimiento contendrá criterios y actividades a realizarse.

Inicia con actividad de definir variables.
Asignación de valores de inicio.
Si condicion a1 es verdadera
  Se realiza proceso n1
  Proporciona respuesta afirmativa
Si no
  Se realiza proceso m1
Fin de condicion a1

Proceso m1
  Realiza funcion de valoración de datos.
Fin de proceso m1

Proceso n1
  Realiza proceso o funcion de datos.
Fin de proceso n1

Capítulo 4.- Diseño de bases de datos.

4.1. Introducción.

Los datos que ha expresado el hombre a través del tiempo por medio de imágenes, gráficos y otros medios, han de ser interpretados para que se conviertan en información. En informática para facilitar la interpretación de datos, surgen modelos de datos como instrumentos que ayudan a incorporar su significado, apoyados de la abstracción como un proceso mental capaz de ocultar detalles y fijarse en lo esencial, buscando propiedades comunes de un conjunto de objetos y ayudando a la comprensión del mundo real [Cast].

Los modelos de datos proporcionan mecanismos de abstracción que permiten la representación de aquella parte del mundo real cuyos datos nos interesa registrar. Dicha representación se concibe en dos niveles : el de las estructuras (esquemas) que hacen posible la representación de la información, y el de la información (base de datos) en sí misma.

La ANSI (American Nacional Standard Institute), propone la arquitectura de una base de datos en tres niveles de abstracción. Global, que contiene una representación de los datos de una organización; externo, en el que los datos (o conjunto de ellos) se describen para atender las necesidades de uno o varios procesos o grupos de usuarios en particular; e internos, que describen las características de los datos tal como han de encontrarse almacenados físicamente.

Según el nivel de abstracción de la arquitectura, el modelo que permitirá su descripción será un modelo global, externo o interno. El modelo global y el modelo externo describen aspectos lógicos de los datos; los internos son propios de cada producto comercial y describen aspectos más cercanos a la máquina donde se almacenan los datos.

Los modelos globales se clasifican a su vez, en modelos conceptuales que facilitan la descripción global del conjunto de información de la empresa al nivel más próximo al usuario, manejando conceptos cercanos al mundo real como entidades, atributos, interrelaciones, etc.; y los modelos convencionales que están orientados a describir los datos a nivel lógico por lo que maneja conceptos de tablas o relaciones de tablas.

4.2. Termino esquema, ejemplar.

La descripción de un cierto mundo real por medio de un modelo de datos dá como resultado un esquema [Cast]. Esquema es la descripción de la estructura de la base de datos y ejemplar del esquema son los datos que en un determinado momento se encuentran almacenados en el esquema.

Un esquema es invariante al tiempo hasta que cambien el mundo real (estructura de la base de datos) mientras que los ejemplares son distintos en el transcurso del tiempo (un dato se inserta, se borra o se modifica).

En los lenguajes de programación existen variables que tienen un tipo, un contenido y en cierto momento adquieren un valor [Diet]. En las bases de datos, en forma similar se habla de una variable de base de datos; cuyo tipo es el esquema y su contenido todos los posible valores del esquema; para que su valor en un momento determinado sea el ejemplar del esquema. En forma abstracta una base de datos son todos los posibles ejemplares que debe contener [Cast].

4.3. Modelo entidad/relación (E/R).

Propuesto por Meter P. Chen en 1976 y 1977, es uno de los modelos conceptuales que se utiliza como herramienta del diseño de bases de datos. Su definición en 1976 por Chen dice [Cast]: "el modelo E/R puede ser usado como una base para una vista unificada de los datos", adoptando "el enfoque más natural del mundo real que consiste en entidades e interrelaciones". Posteriormente se han hecho aportaciones por lo que no se puede considerar que exista un único modelo E/R. Sin embargo, "el modelo E/R permite al diseñador concebir la base de datos aun nivel superior de abstración, aislándolo de consideraciones relativas a la máquina y a los usuarios en particular, y centrándolo en un plano en el que la información desempeña un papel fundamental" [Cast].

4.3.1. Elementos del modelo E/R.

En el modelo E/R se distinguen los siguientes elementos :

Entidad. Una entidad es cualquier objeto (real o abstracto) que existe en la realidad y acerca del cual queremos almacenar información en la base de datos. En el sentido abstracto, se define tipo de entidad como la estructura genérica que describe un conjunto de entidades, mientras que entidad es cada uno de los ejemplares de ese tipo de entidad [Cast][Diet].

Cada tipo de entidad tiene asociado un predicado que lo define por así decirlo, p.e. El tipo de entidad "Dependencias" tiene asociado el predicado "Dependencias de la universidad que pueden solicitar un vehículo de transporte". Se dice que una entidad forma parte de un tipo de entidad si cumple el predicado asociado al correspondiente tipo de entidad. En este caso, "Facultad de Ciencias", sería el ejemplar o entidad del tipo de entidad "Dependencias".

Existen dos tipos de entidades [Cast]:

- Regulares que son aquellas cuyos ejemplares tienen existencia por sí mismos. Se representan gráficamente por medio de un rectángulo etiquetado en el interior con el nombre del tipo de entidad, y
- Débiles en las cuales la existencia de un ejemplar depende de que exista un cierto ejemplar de otro tipo de entidad y que la desaparición del segundo provoca la desaparición de todos los primeros. Su representación gráfica son dos rectángulos concéntricos con su nombre en el interior.

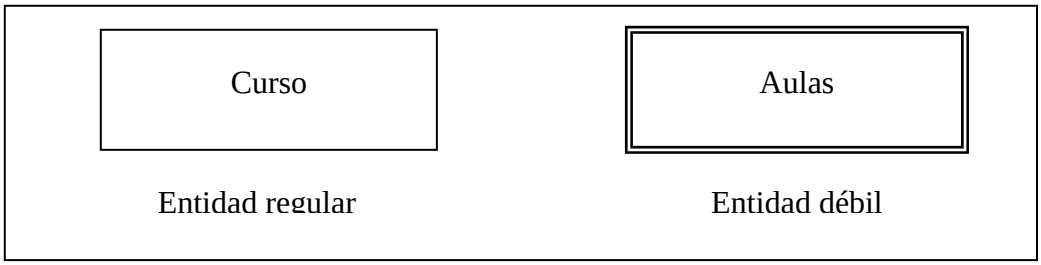


Fig. 4.1. Representación de entidades regula y débil.

Interrelación. Es aquella asociación, vinculación o correspondencia entre entidades. De la misma manera que sucede con entidad, se denomina tipo de interrelación a la estructura genérica que describe un conjunto de interrelaciones, e interrelación será cada uno de los ejemplares concretos [Diet]. El tipo de interrelación se representa gráficamente por medio de un rombo etiquetado con el nombre de la interrelación unido mediante arcos a los tipos de entidad que asocia. Entre dos tipos de entidad puede existir más de un tipo de interrelación [Ador].

Elementos de un tipo de interrelación.

En un tipo de interrelación se pueden distinguir los siguientes elementos:

- **Nombre**: Es la etiqueta que lo distingue unívocamente del resto y mediante el cual ha de ser referenciado. Por lo tanto, siempre ha de aparecer el nombre en el rombo.

- **Grado**: Es el número de tipos de entidad que participan en un tipo de interrelación. Por ejemplo, un tipo de interrelación puede asociar dos tipos de entidad distintas (grado 2). Un caso particular pude asociar un solo tipo de entidad consigo misma (grado 2) y son llamadas reflexivas, en las que asociamos un tema con otros, reflejando la posibilidad de que uno de ellos esté compuesto por subtemas.

- **Tipo de Correspondencia**: Es el número máximo de ejemplares de un tipo de entidad que pueden estar asociados, en una determinada interrelación, con un ejemplar de otro tipo. Gráficamente se etiqueta con 1:1 (relación uno a uno), 1:N (relación uno a muchos) o N:N (relación muchos a muchos) según corresponda al lado de la interrelación o bien dándole orientación al arco de unión en el sentido 1 a N ó N a N mediante una punta de flecha.

- **Papel**: Es la función que cada uno de los tipos de entidad realiza en el tipo de interrelación; se representa poniendo el nombre del papel en el arco que une cada tipo de entidad con el tipo de interrelación. Cuando no existe ambigüedad se suele prescindir de representar el papel.

Cardinalidad de un tipo de entidad. La cardinalidad de un tipo de entidad se define como el número máximo y mínimo de ejemplares de un tipo de entidad que pueden estar interrelacionados con un ejemplar de otro u otros tipos de entidad que participan en el tipo de interrelación [Cast]. Se representa por medio de etiquetas (0,1),(1,1),(0,n) ó

(1,n) según corresponda al lado de los tipos de entidades asociados por el tipo de interrelación.

Si los dos tipos de entidad asociados por un tipo de interrelación los llamamos E1 y E2 a sus subconjuntos, podemos decir que su interrelación compuesta por [E1(1,1):E2(0,n)] se puede leer como [todo ejemplar de E2 pertenece a un único ejemplar de E1 : que existen ejemplares de E2 que no tienen asignado un ejemplar de E1].

Dominio y valor. Un tipo de entidad o interrelación tiene características o propiedades que toman valores para cada ejemplar de éstas. El conjunto de posibles valores que puede tomar una cierta característica es el dominio. Un dominio tiene asociado un predicado por ejemplo el dominio "Idiomas" ("inglés, español, italiano, francés") son los valores que puede tomar la característica "Idiomas".

Un dominio puede definirse por intensión si específica el tipo de datos (pe. Carácter de 30 para el dominio "Idiomas" o fecha para "Fecha_ingresos"); o por extensión, cuando se declara el valor de cada elemento del dominio :"inglés","español","italiano","francés". El dominio se representa por medio de un círculo u óvalo etiquetado con su nombre.

Atributo. Cada una de las propiedades o características que tiene un tipo de entidad o un tipo de interrelación se denomina atributo y estos toman valores de uno o varios dominios [Diet]. Por lo que el atributo le da una determinada interpretación al dominio en el contexto de un tipo de entidad o de un tipo de interrelación. Un atributo en un tipo de entidad toma valores sobre todos los posibles subconjuntos de un dominio o de un conjunto de dominios. Su representación gráfica consiste en cualificar con su nombre el arco que une el dominio con el tipo de entidad o de interrelación. Sin embargo, para simplificar la representación gráfica y siempre que coincida el nombre del dominio con el del atributo, será suficiente con el círculo u óvalo, eliminando el nombre del atributo.

Entre todos los atributos de un tipo de entidad debemos elegir uno o varios que identifiquen unívocamente cada uno de los ejemplares de ese tipo de entidad. A este atributo o conjunto de atributos se denomina atributo identificador candidato y los atributos que lo componen deben ser mínimos en el sentido de que la eliminación de cualquier de ellos le haría perder su carácter identificador. De los atributos identificador candidato se elige uno que represente unívocamente al atributo siendo identificador principal (IP) y el resto serán identificadores alternativos (IA).

Un atributo que toma para cada ejemplar de entidad un único valor de cada dominio subyacente (un libro tiene un único título, un único ISBN, etc.) se llama univaluado, en contraposición los multivaluados son los atributos que pueden tomar más de un valor de su dominio subyecente (un curso pude impartirse en más de un idioma o un profesor puede tener más de un teléfono). Por otro lado puede obligarse a un atributo de un tipo de entidad a que tome, como mínimo, un valor del domino subyacente para cada ejemplar de entidad, por lo que su valor de este atributo es obligatorio (no pude ser nulo) para todo ejemplar de la entidad, siendo opcional cuando se puede tomar de un conjunto de dominio.

4.3.2. Dependencia en existencia y en identificación.

Cuando una interrelación asocia a dos tipos de entidad en donde alguna es débil, se dice que la interrelación es débil en otro caso es una interrelación regular. Los ejemplares dentro del tipo de interrelación se ven afectados por la dependencia en existencia y la dependencia en identificación.

Hay dependencia en existencia cuando los ejemplares de un tipo de entidad (entidad débil) no pueden existir si desaparece el ejemplar del tipo de entidad regular del cual dependen (el rombo se etiqueta con una E). Es dependencia en identificación cuando además de cumplir la condición anterior, los ejemplares del tipos de entidad débil no se pueden identificar por sí mismos y exigen añadir el identificador principal del tipo de entidad regular del cual dependen (el rombo se etiqueta con una ID).

4.3.3. Interrelación redundante.

Una interrelación es redundante cuando su eliminación no implica pérdida de semántica porque existe la posibilidad de realizar la misma asociación de ejemplares por medio de otras interrelaciones. Estas redundancias forman parte, por lo general, de ciclos en el modelo E/R. Un ejemplo representativo de [Cast] es el de la imagen Fig. 4.2., en la que se dá un ciclo entre las entidades profesor, curso y departamento y se supone que un profesor sólo puede impartir cursos de doctorado que estén adscritos al departamento al que él pertenece; luego entonces, si se conocen los cursos de doctorado que imparte un profesor y el departamento al que está adscrito cada curso, se deduce a qué departamento pertenece dicho profesor, así también, dado un departamento, si sabemos qué cursos de doctorado tiene adscritos y los profesores que imparten dichos cursos, se deduce qué profesores pertenecen a dicho departamento, por lo que la interrelación pertenece entre las entidades profesor y departamento es redundante y su eliminación no produce pérdida de información.

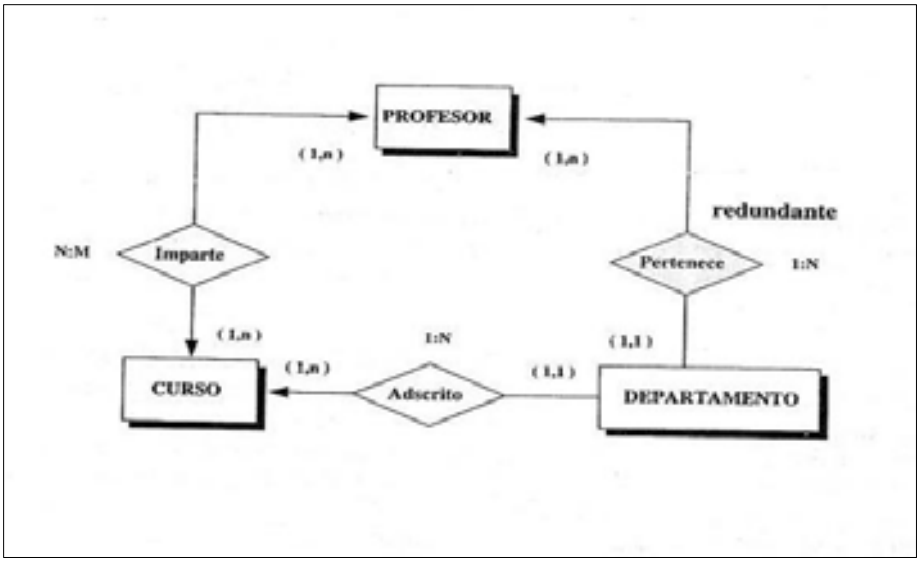


Fig. 4.2. Interrelación redundante.

4.4. Diseño lógico estándar.

A partir del esquema conceptual resultante del modelo E/R y teniendo en cuenta los requisitos de proceso y de entorno, se elabora un esquema lógico estándar (ELS), que se apoya en un modelo lógico estándar (MLS). El ELS se puede describir utilizando el lenguaje estándar del modelo de datos SQL.

Para el modelo de datos relacional se utiliza un representación gráfica llamada grafó relacional. Donde, un grafo está compuesto de un conjunto de nodos multiparticionados, donde cada nodo representa un esquema de relación o sea, una tabla de la base de datos. Para cada tabla, como mínimo, ha de aparecer su nombre y sus atributos indicando su clave primaria subrayada con trazo continuo y sus claves ajenas por trazo discontinuo. Se dibuja además un conjunto de arcos que conectan los atributos que constituyen la clave ajena con la tabla referenciada, permitiendo así que el usuario entienda los campos clave que comparten dominios comunes. La fecha parte del arco y señala con su punta a la tabla referenciada.

4.4.1. Principios de transformación del esquema conceptual al lógico estándar.

Existen tres reglas básicas para convertir un esquema del modelo E/R al relacional :

1. Todo tipo de entidad se convierte en una relación.
2. Todo tipo de interrelación N:M se transforma en una relación.
3. Para todo tipo de interrelación 1:N se realiza lo que se denomina propagación de clave o bien se crea una nueva relación.

Se observa que en el paso del modelo E/R al relacional, se pierde semántica, puesto que tanto las entidades como las restricciones se transforman en relaciones, ya que en el modelo relacional sólo existe la relación para representar ambos tipos de objetos. La pérdida de semántica no implica, sin embargo, un peligro para la integridad de la base de datos, ya que se definen restricciones de integridad que aseguren la conservación de la misma [Cast].

4.4.2. Reglas concernientes al modelo básico.

1.- Transformación de dominios. En el modelo relacional estándar un dominio es un objeto más, propio de la estructura del modelo que, como tal, tendrá su definición concreta en el ELS que se elija. Se crea el dominio observando un conjunto de valores para el atributo que se observa. Por ejemplo, que un empleado tenga nombre formado por caracteres de la “a” a la “z” y cuya longitud es de máximo 50.

Su modelo relacional : CREATE DOMAINE nombre_atributo AS tipo_longitud;

modelo E/R	Lenguaje lógico estándar (LLS)
“nombre” 	CREATE DOMAINE nombre AS char(50);

Fig. 4.3. Representación de un atributo en el LLS.

2.- Transformación de entidades. El MLS posee el objeto relación o tabla mediante el cual se representa a la entidad por lo que cada tipo de entidad se convierte en una relación o tabla. Por su pare en el modelo relacional : CREATE TABLE nombre_entidad

3.- Transformación de atributos de entidades. Cada atributo de una entidad se transforma en una columna de la relación a la que ha dado lugar la entidad. En el modelo relacional :

Nombre_Entidad(identificador principal, identificador alternativo, no identificadores)

Se divide en tres subreglas :
I.- Atributos identificadores principales(AIP). Estos atributos pasan a ser la clave primaria de la relación y en el lenguaje lógico estándar (LLS): PRIMARY KEY (atributo identificador principal) luego de la definición de la tabla.
II.- Atributos identificadores alternativos(AIA). Son soportados directamente por el modelo relacional y la transformación es directa. Si se desea que estos atributos no tomen valores nulos habrá que indicarlo. El LLS utiliza la cláusula UNIQUE para estos objetos.
III.- Atributos no identificadores. Pasan a ser columnas como los anteriores de la relación, las cuales tienen permitido tomar valores nulos a no ser que se indique lo contrario. Quedando en el LLS :

CREATE TABLE Entidad(AIP, AIA, Atributos no identificadores PRIMARY KEY(AIP) UNIQUE(AIA));

4.- Transformación de interrelaciones. Dependiendo del tipo de correspondencia de la interrelación variará la manera de realizar la transformación al esquema relacional, y se desglosa en tres subreglas:

I. Interrelaciones N:M. Un tipo de interrelación N:M se transforma en una relación que tendrá como clave primaria la concatenación de los AIP de los tipos de entidad que asocia. La siguiente figura muestra la asociación que existe entre los profesores y los cursos que imparten, apareciendo una relación cuya clave primaria está compuesta por la concatenación del código del profesor y el código del curso.

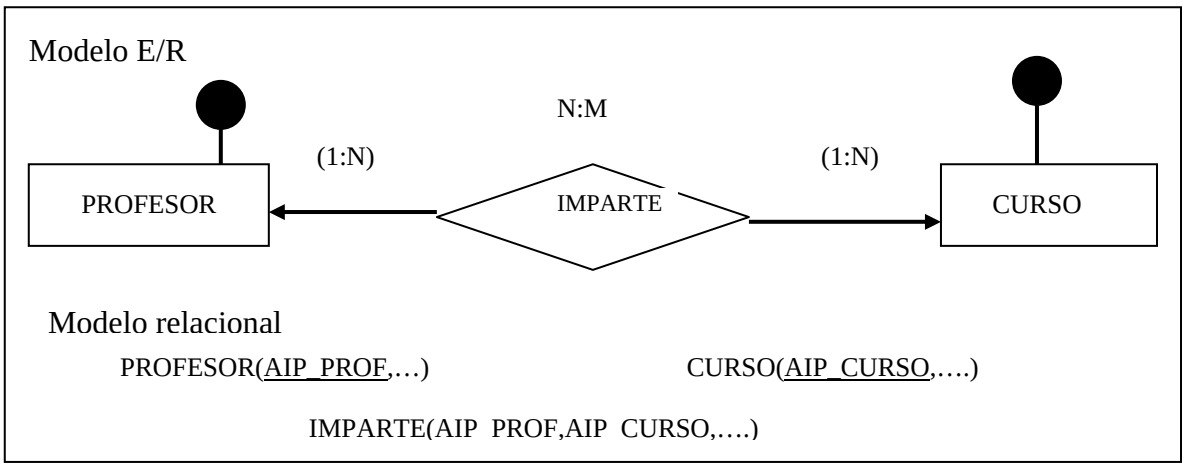


Fig. 4.4. Interrelación N:M entre entidades.

Su LLS es :

CREATE TABLE IMPARTE(AIP_PROF,AIP_CURSO, …, PRIMARY KEY (AIP_PROF,AIP_CURSO) FOREIGN KEY (AIP_PROF) REFERENCES PROFESOR ON DELETE CASCADE FOREIGN KEY (AIP_CURSO) REFERENCES CURSO ON DELETE CASCADE)

II. Interrelaciones 1:N. Existen dos soluciones para la transformación.

a) Propagar los AIP del tipo de entidad que tiene de cardinalidad máxima 1 a la que tiene N, es decir en el sentido de la flecha desapareciendo el nombre de la interrelación, con lo cual se pierde semántica.
b) Transformarlo en una relación, como si se tratara de una interrelación N:M, sin embargo la clave primaria de la relación creada es sólo la clave primaria de la tabla a la que le corresponde la cardinalidad N.

III. Interrelaciones 1:1. Una interrelación de tipo 1:1 es un caso particular de una N:M o también de una 1:N por lo que no hay regla fija para la transformación de este tipo de interrelación al modelo relacional pudiéndose aplicar la reglas 4.I. ó 4.II. Los criterios para aplicar una u otra regla (para propagar la clave) se basan en las cardinalidades mínimas, en recoger la mayor cantidad de semántica posible, evitar los valores nulos o en motivos de eficiencia. Tres ejemplos para seguir la pauta proporciona [Cast] :

a) Si las entidades que se asocian poseen cardinalidades (0,1) puede ser conveniente transformar la interrelación 1:1 en una relación.
b) Si una de las entidades que participa en la interrelación posee cardinalidades (0,1), mientras que en la otra son (1,1), conviene propagar la clave de la entidad con cardinalidades (1,1) a la tabla resultante de la entidad de cardinalidades (0,1).
c) En el caso de que ambas entidades presenten cardinalidades (1,1), se puede propagar la clave de cualquiera de ellas a la tabla resultante de la otra, teniendo en cuenta en este caso los accesos más frecuentes y prioritarios a los datos de las tablas.

5.- Transformación de atributos de interrelaciones. Si la interrelación se transforma en una relación, todos sus atributos pasan a ser columnas de la relación.

En caso de que la interrelación se transforme mediante propagación de clave, sus atributos migran junto con la clave a la relación que corresponda.

6.- Transformación de restricciones. En cuanto a las restricciones de usuario, existen ciertas cláusulas en el LLS que pueden recogerlas. Por ejemplo, podemos restringir un rango determinado los valores de un dominio a través de la cláusula BETWEEN, o bien determinar por enumeración los valores que puede tomar una columna en una tabla como se observa en la regla 1.

Capítulo 5.- Desarrollo de la aplicación.

5.1. Planteamiento del problema.

La Dirección General de Servicios Generales (DGSG) de la UNAM tiene a su cargo la adquisición de unidades vehiculares y su asignación a otras dependencias de la UNAM cuando lo solicitan.

Cuando se asigna una unidad vehicular a otra dependencia, ésta cubre los gastos de mantenimiento y circulación como : seguro, verificación, gasolina, taller, etc.

Uno de los problemas a los que se enfrenta la DGSG es la falta de información en forma ágil de las unidades vehiculares con que cuenta y muchas veces se traslapa la documentación de sus unidades, ocasionando problemas de inconsistencia.

Algunos de los problemas a los que se enfrenta, son :

- Cuando alguna unidad tiene una eventualidad de accidente, no se sabe si está asegurado hasta ver su expediente.
- Se canaliza su baja cuando el vehículo está en buenas condiciones y circulando.
- Cuando la unidad está a préstamo en otra dependencia, no se sabe su condición al pasar algún tiempo.
- En algunas ocasiones se ha pagado el seguro cuando la unidad ya fue dada de baja y hasta rematada por Patrimonio.
- Al finalizar un periodo, por ejemplo el año, se tiene que “rastrear” a las unidades vehiculares para generar informes (de verificación vehiculares o administrativos).
- Cuando se requiere saber de una unidad vehicular, se tiene que realizar un seguimiento hacia atrás en expediente(s).
- El respaldo de movimientos de una unidad (su histórico) se lleva en papel.

Por lo que surge la necesidad de un sistema que ayude a controlar la información de las unidades vehiculares. Aplicaré el proceso en espiral siguiendo las fases descritas en los capítulos 3 y 4.

Durante las primeras entrevistas con el cliente, se identifica la información que se necesita para satisfacer sus necesidades, en este caso el control vehicular. El diseñador se pregunta cómo se puede llevar a cabo y se cuestiona al cliente sobre cómo la maneja administrativamente estableciendo un diálogo interactivo, con esto se va obteniendo la información que se necesita.

Se propone identificar a las personas administrativas que están relacionadas en el manejo de la información y control de la misma. Estas personas deberán tener el control de las unidades vehiculares con información actualizada y registrar cualquiera de los movimientos en un medio electrónico.

Cualquier movimiento o actualización interna de información de las unidades se realiza por medio de un oficio llamado “cuadrado”, que al llegar al departamento de inventarios se le asigna un control formado por cuatro dígitos del año, un “/” y tres dígitos que son consecutivos (2006/025). Dicho cuadrado también sirve de apoyo a cualquier solicitud externa de datos por parte de algún administrativo que lo requiere.

5.2. Identificación de actores.

Por medio de entrevistas con el personal de la DGSG, se busca quién interviene en el manejo de la información o para qué se necesita : quién emite formatos, quien necesita ubicar unidades, quien genera reportes, que unidades están a préstamo o en taller, verificación vehicular, gastos por aseguramiento, bajas, etc.

Se identifica como fluye la información, identificando ***quienes*** tienen participación en ella.

- Cuando se necesita reparar un vehículo se comunica al ***Contador***, quien realiza un oficio para informar al ***Jefe de inventarios*** y al ***Jefe del taller***.
- El ***Jefe de inventarios*** es la persona encargada de controlar la información de los vehículos, respaldarla en archivo y tenerla actualizada.
- El ***Jefe del taller*** informa si tiene reparación o se propone para dar de baja.
- El ***Coordinador*** solicita un pago a las dependencias que tienen asignado el vehículo.
- El ***Coordinador*** solicita a ***Patrimonio*** el presupuesto anual para la adquisición de vehículos (Se tiene que enviar un reporte semestral y anual a ***Patrimonio***, sobre el parque vehicular existente).
- El ***Contador*** solicita actualizar y corregir los datos de los vehículos al ***Jefe de inventarios*** quién anexa al oficio de solicitud el número del cuadrado.
- El ***Coordinador*** le solicitan reportes al ***Jefe de inventarios***.
- La J***efatura de inventarios*** respalda la información de cualquier movimiento de un vehículo.
- Se solicitar a ***Patrimonio*** los pagos por tenencia, verificación, etc.

Se hace una lista de quiénes participan y se captan sus funciones y acciones administrativas. Junto con el cliente se depura esta lista (p.e. el Contador y el Coordinador pueden desempeñar la misma acción dentro del sistema o, también no siempre captura la información el Jefe de inventarios) y se define el nombre que deberá tener en la documentación del sistema.

Es decir, se identifican a los actores :

- Coordinador y/o Contador serán el Administrador del sistema,
- Jefe de inventarios será el Personal de inventarios en el sistema, y
- Quien captura la información será el Capturista en el sistema.

Actor.	**Responsabilidad.**
Capturista	Persona encargada de registrar la información y tenerla actualizada. Tiene a su cargo el registro y realiza cualquier movimiento de la información. La información es obtenida bajo un formato y oficio que deben quedar registrados (cuadrados).
Personal de inventarios	Persona responsable de conocer la condición y ubicación vehicular. Genera reportes de existencia, situación y ubicación vehicular, así como grupos de asegurados. Canaliza formatos del movimiento vehicular al Capturista y supervisa el movimiento de estos.
Administrativo	Persona responsable del control vehicular. Necesita la obtención de ingresos para tener activas a la unidades, adquirirlas o darlas de baja. Necesitan saber a quien esta asignada un vehículo y la situación del vehículo. Emite oficio de movimientos vehiculares, de asignación de usuarios y transporte de información.

Cuadro 5.1. Definición de actores y sus responsabilidades.

5.3. Identificación de casos de uso.

Los actores anteriores serán los usuarios del sistema.

De las actividades recabadas durante las entrevistas se resumen y se identifican actividades comunes que serán los casos de uso de los actores. Con el cliente se definirá el nombre con el que se identificarán los casos de uso que formarán parte del Menú del sistema y de la Interfaz del usuario.

Actividades comunes.	**Nombre del casos de uso.**
Capturar los datos de una unidad nueva.	Alta vehicular.
Estado de los vehículos.	Situación vehicular.
Corregir datos.	Modificación de datos.
Emitir reportes de unidades.	Reporte de existencia vehicular.
Emitir información por medios electrónicos.	Respaldo de información.
Solicitar un pago de las unidades asignadas.	Reporte por trámite vehicular.
Informe de las unidades asignadas a dependencias externas.	Reporte de existencia vehicular.
Emitir información de unidades.	Transporte de información.

Cuadro 5.2. Actividades y casos de uso.

Con estos casos de uso se agrupan los actores definidos y se definirá al actor faltante, en este caso, el Administrador del sistema.

Actor : Capturista.	**Actor : Personal de Inventarios.**
Realiza :	Solicita y/o Realiza
• Alta vehicular. • Modificación de datos. • Situación vehicular.	• Reporte por tramite. • Reporte existencia vehicular. • Histórico de unidad.
Actor : Administrador.	**Actor : Administrador del sistema.**
Solicita :	Realiza
• Reporte existencia vehicular. • Histórico de unidad. • Transporte de información.	• Respaldo de información.

Cuadro 5.3. Actores y actividades.

Teniendo los actores y sus casos de uso se puede construir el diagrama general de casos de uso (Fig. 5.4.).

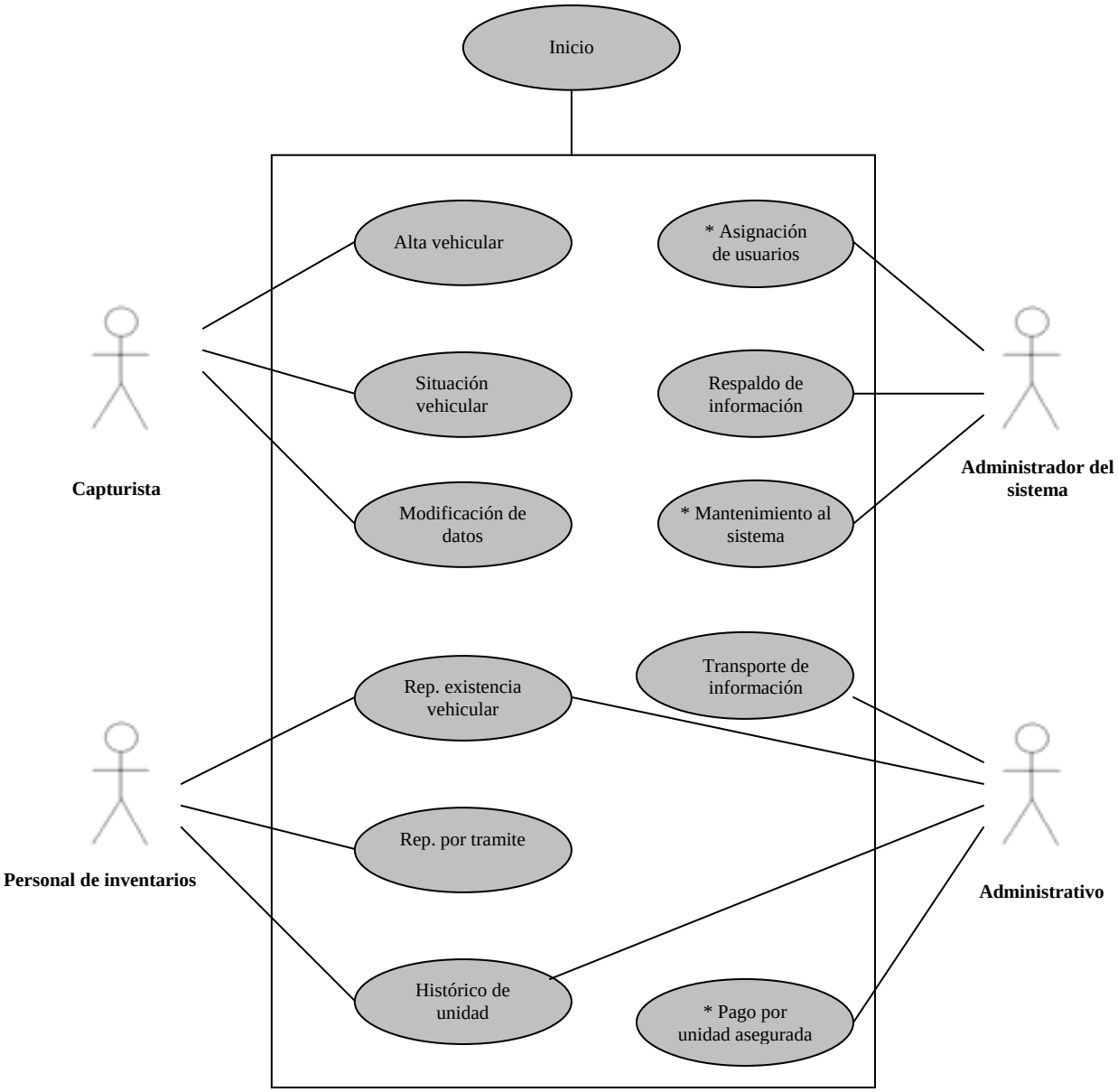


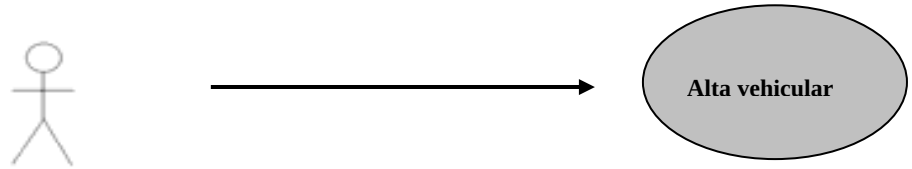
Fig. 5.4. Diagrama general de casos de uso.
(Los marcados con * surgen más adelante)

5.4. Detalle de los casos de uso.

Para los casos de uso, se detallan las actividades del caso de uso escribiendo en pasos sencillos la idea general de cómo cada actor interactuará con el sistema.

El detalle del caso de uso : Alta vehicular sería :

Caso de Uso : Alta vehicular.



Actor : Capturista.

Descripción :	El Capturista realiza la alta de una unidad vehicular.
Precondiciones :	El Capturista recibe notificación de alta vehicular bajo formato establecido. Se debe proporcionar para filtro de datos : núm. de serie, núm. de motor y/o placa de la unidad vehicular.

Flujo de acciones :

Actor		**Sistema**		
Paso	**Acción**	**Paso**	**Acción**	**Excepción**
1	Selecciona Alta vehicular.	2	Solicita datos filtro : placa, núm. de serie y/o núm. motor.	
3	Escribe datos filtro proporcionados.	4	Realiza filtro de datos proporcionados.	E1
		5	Solicita datos restantes de la unidad vehicular. Proporciona opciones de : Alta vehicular y Cancelar.	
6	Escribe datos restantes del formato proporcionado.			E2
7	Elige “Alta vehicular”.	8	Respado de datos de la unidad vehicular.	
		9	Abandona esta ventana y muestra el menú de trabajo del Capturista.	

Excepciones :

Num. Id	**Nombre**	**Acción**
E1	Unidad existente.	El sistema detecta que ya existe esa unidad, despliega mensaje de existencia y espera datos filtro nuevamente.
E2	Dato incorrecto	El sistema proporciona menú para su elección

Postcondiciones : El capturista ha dado de alta unidad vehicular. El sistema muestra ventana de trabajo del capturista.

Fig. 5.5. Caso de uso Alta vehicular.

Es importante que el cliente comprenda el alcance del sistema y los usuarios su función dentro de él.

En este caso, cada actor tendrá acceso, por medio del caso de uso : Inicio, a un subsistema del todo.

5.5. Prototipo de pantalla.

Identificados los casos de uso por actor se realizan las pantallas que mostrará el sistema. El actor Capturista, como se había dicho, mantiene al día la información de unidades, por lo que en el sistema deberá tener una pantalla para cada caso de uso : Alta vehicular, Situación vehicular, etc.

Al revisar los casos de uso, se sugiere el menú de cada actor y las pantallas necesarias.

Para el actor Capturista se tendrá :

Actor : Capturista.

Caso de Uso	Menú	Pantalla(s)
Alta vehicular	Alta vehicular	Alta vehicular
Situación vehicular	Situación vehicular	Situación vehicular – activo Situación vehicular – taller Situación vehicular – póliza Situación vehicular – baja Situación vehicular – ubicación
Modificación de datos	Modificación de datos	Modificación de datos
	Salir del sistema	

Para formar las pantallas se recurre a la información de los requerimientos y a la documentación de los casos de uso.

¿Qué datos se manejarán para solventar las actividades de los casos de uso?
En el caso de uso "Alta vehicular", maneja datos para filtrar información, datos de la unidad y datos para generar su histórico, que son las actividades de este caso de uso.

Las pantallas se estructuran en secciones de información. Por ejemplo la Alta vehicular tiene una sección donde pide datos de identificación de la unidad, la segunda sección proporciona los datos restantes y la tercera sección proporciona los movimientos de la unidad que se hayan realizado.

Las pantallas se proponen al usuario y se inspecciona su manejo por ellos, la pantalla Alta vehicular se propondría de la elección del menú del capturista :

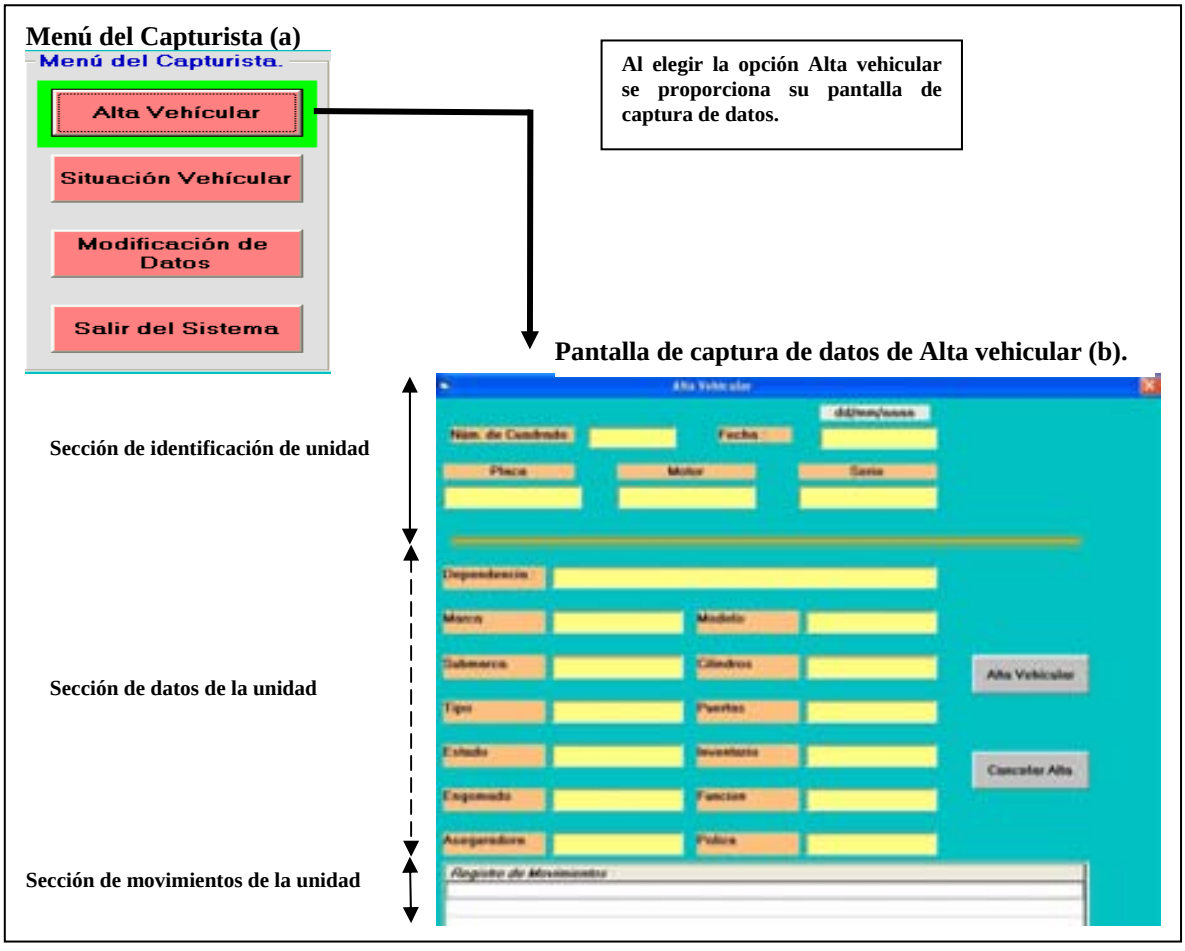
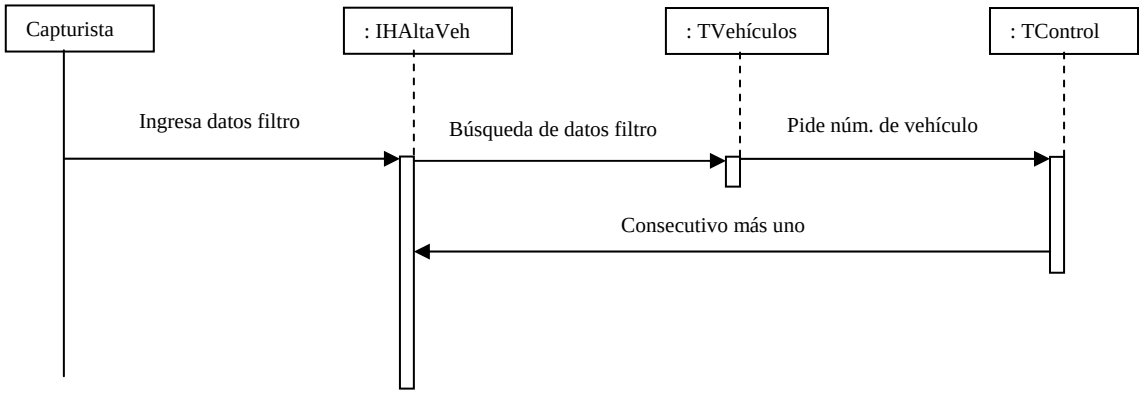


Fig. 5.6. Pantalla de Alta vehicular.

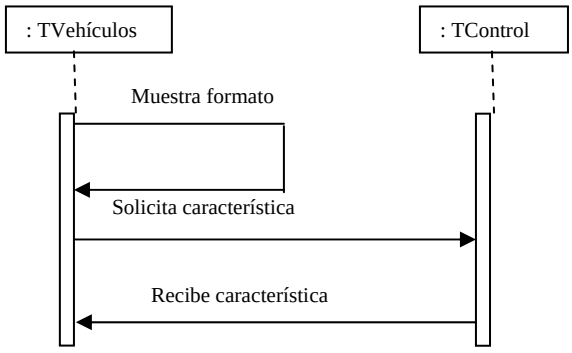
5.6. Diagramas de secuencia.

Teniendo los diagramas de flujo de datos y los casos de uso, se enfoca la atención en uno de ellos, que es un subsistema de algún actor. Supongamos que el caso de uso inicio nos ubicará en el subsistema de un actor y que de las distintas opciones que tenga el actor se muestren por un menú, para que el elija la apropiada (únicamente se realiza para entrar el diagrama de flujo y el caso de uso respectivo). Se elije la opción del caso de uso Alta vehicular, su diagrama de secuencia se estructura de la siguiente manera :

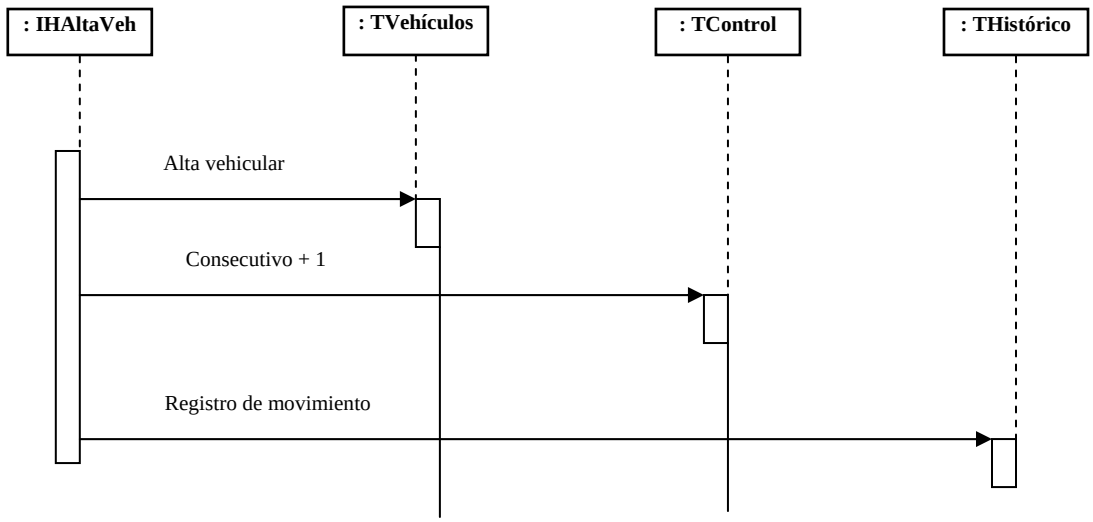
Para Alta vehicular del capturista se tienen que satisfacer los pasos 1 a 4 del caso de uso en cuestión.



El paso 5 y 6 solo es un formato que solicita datos restantes y que o se escriben o se toman de una tabla. En este caso es con la tabla TControl de la cual se obtienen las características : marca, tipo, submarca, etc de los vehículos



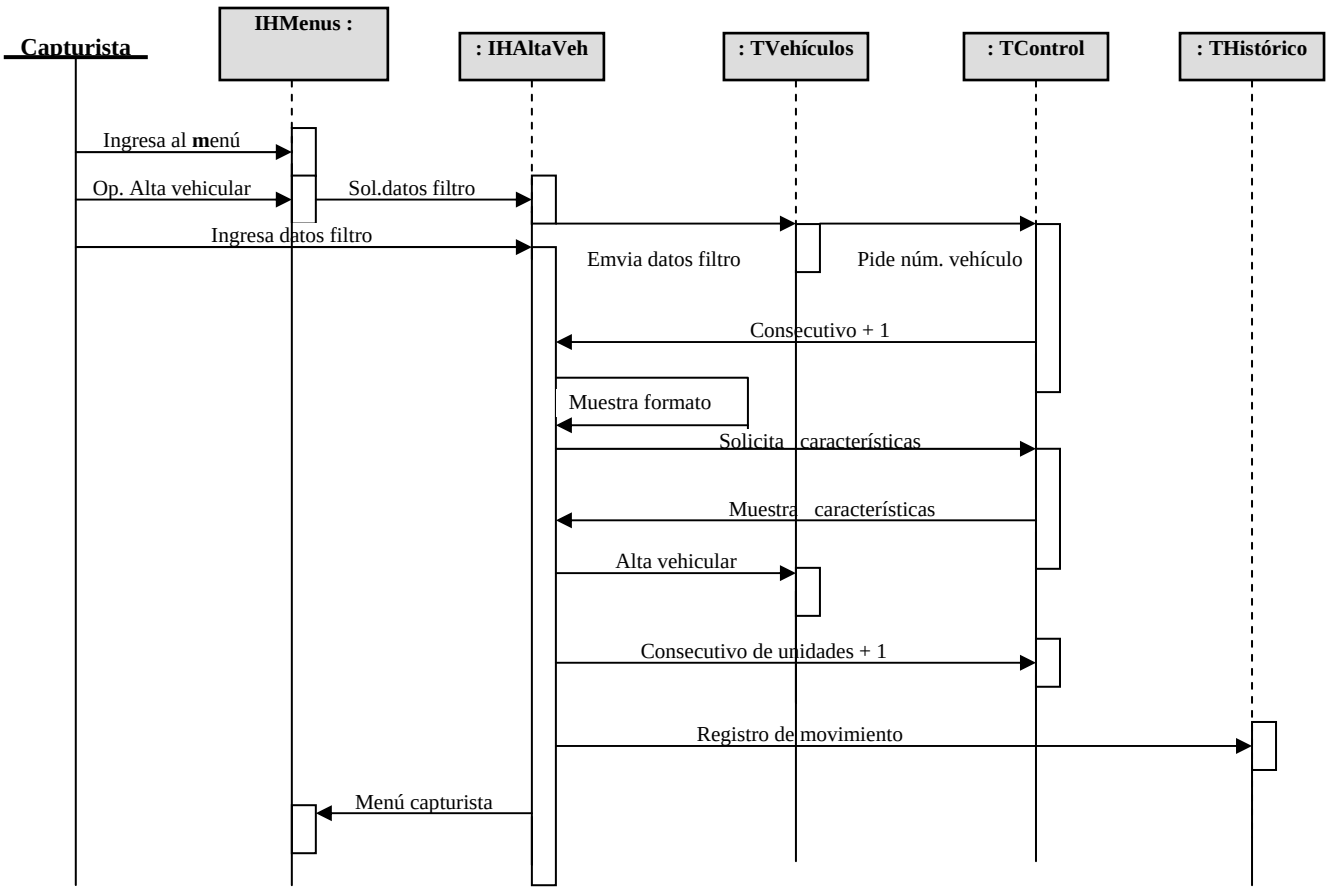
Una vez terminada la captura de los datos sólo resta respaldar la información, que son los pasos 7,8 y que afecta a las tablas : TVehículos, TControl y THistórico



Por último, se regresa al menú del capturista en el paso 9.

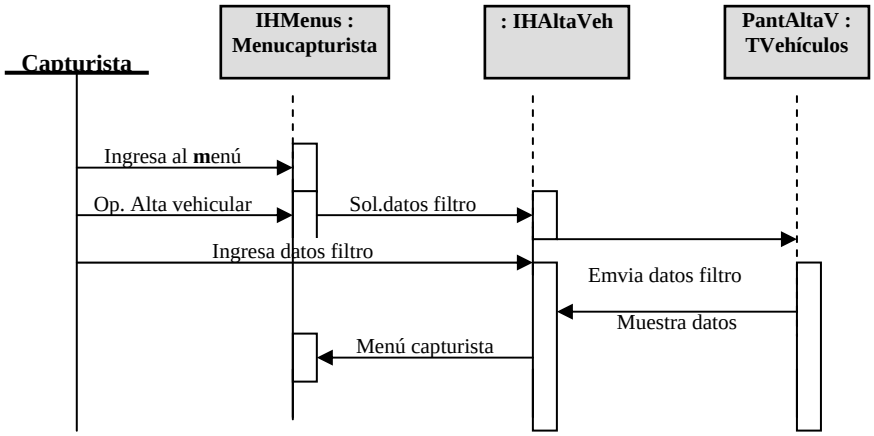
Teniendo un diagrama de secuencia para el caso de uso Alta vehicular como el que sigue :

Diagrama de secuencia.
Actor : Capturista.
Caso de Uso : Alta vehicular. Flujo normal.



Un caso anormal sería si en la búsqueda de datos filtro se localiza la unidad, entonces mostraría los datos.

Diagrama de secuencia.
Actor : Capturista.
Caso de Uso : Alta vehicular. Flujo anormal.



5.7. Diagramas de estado.

Un caso de uso está formado por una o más actividades. Una actividad tiene estados que cambian valores por medio de una transición. La transición es un evento el responsable de provocar la transición entre estados de las actividades.

En el ejemplo de Alta vehicular el capturista elige de su menú la opción Alta vehicular. Como primer paso en este caso de uso se solicitan los datos filtro de un vehículo. Cuando se elige que se verifique su existencia, la transición se realiza enviando los datos filtro capturados y los envía a otro estado de la actividad que es Existencia vehicular. Este evento proporcionará si existe o no el vehículo. Si Existe se activa el estado Mostrar datos y termina. Si no existe, se va al estado Captura de datos (que solicitará los datos restantes). Una vez terminada la captura de datos se podrá elegir alguno de dos eventos : Cancelar que origina terminar. Si la opción es Alta, se pasa a los estados Respaldo de información, Registro de movimiento y terminar (Fig. 5.7).

Diagrama de estados.
Actor : Capturista.
Caso de Uso : Alta vehicular.

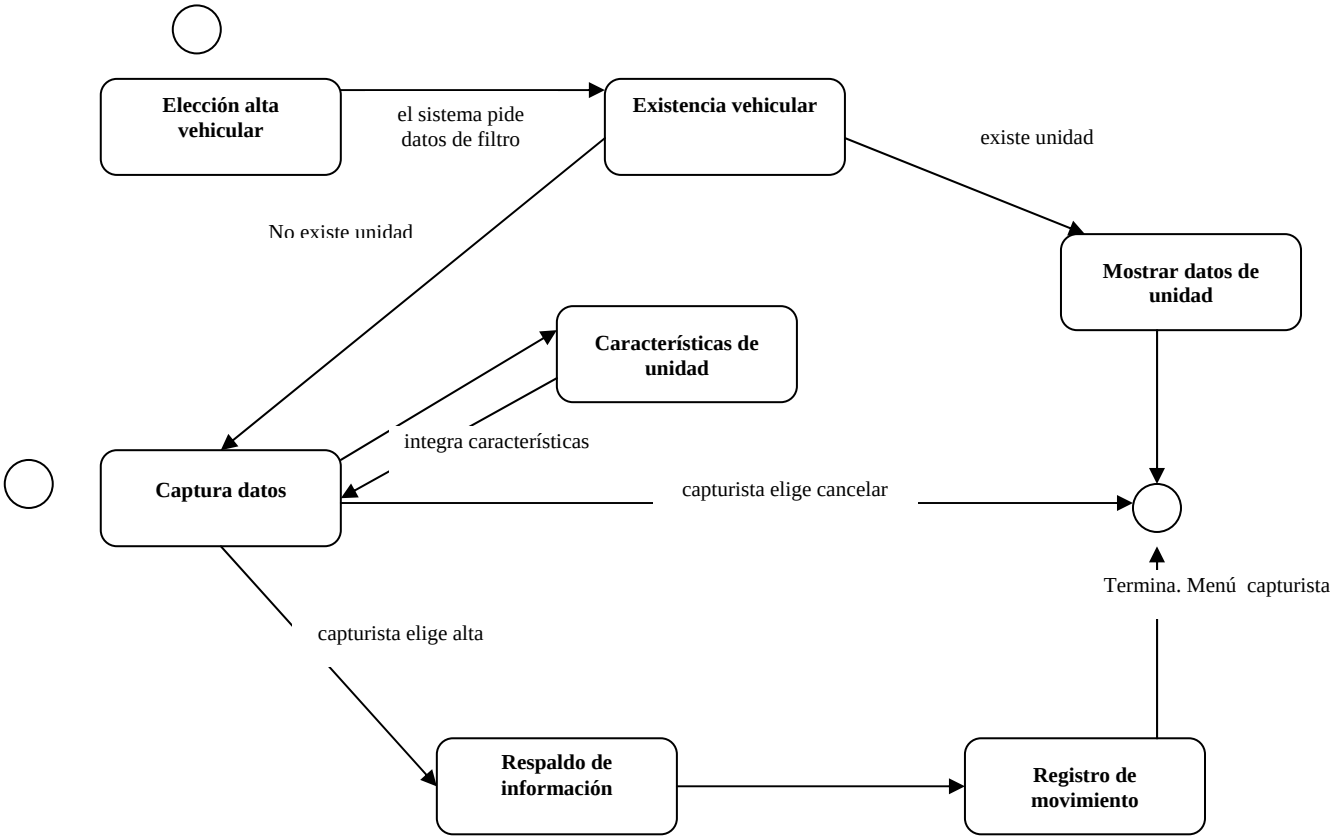


Fig. 5.7. Diagrama de estados del caso de uso Alta vehicular.

Capítulo 6.- Construcción de tablas.

6.1. Definición de base de datos.

Se investiga la información que desea manejar el cliente tanto interna como externa. Nos apoyamos en el modelo de entidad relación y el modelo lógico estándar para definir las tablas que usará el sistema.

Un archivo es un tipo de almacenamiento de información ordenada bajo un sistema operativo. Un archivo de datos o base de datos es un conjunto de una o mas tablas que permiten almacenan y manejar información referente a un elemento en particular. Cada tabla o entidad de información esta formada por registros (filas) con información identificable acerca de uno de los elementos que hace referencia y cada registro está compuesto de uno o mas campos (columnas) que son las unidades básicas de información de las bases de datos [Grav].

6.1.1. Recabar datos para ubicar entidades.

De entre las necesidades del cliente se encuentra el manejo de información. Por ejemplo, se identifica que, entre sus necesidades están las de emitir listados para verificación, para tenencia y/o para seguro vehicular, que son enviados para su tramite acompañadas de un oficio y que es información especifica de la(s) unidad(es) que se debe(n) proporcionar al "exterior" del sistema.

De las entrevistas se observa que los vehículos tienen datos que no se repiten y son únicos, estos son la placa, el núm. de serie y el núm. de motor, pero no siempre se dan todos y faltan uno u otro pero no los tres, por lo que estos datos sirven para identificar un vehículo, es decir cualquiera de ellos proporciona una unidad vehicular única. Para su control interno se agrega un número de control que se llama consecutivo que va a ser único desde que se da de alta y aún perdura cuando ya se ha subastado. Este consecutivo ayuda a llevar un histórico de movimientos (un movimiento puede ser una alta, una corrección/agregado de datos, etc.) de la unidad y es más ágil su manejo que usar el de núm. de serie (serie), núm. de motor (motor) o placa ya que o son muy grandes o no se proporcionan en el instante. Indagando la información necesaria para estos trámites y qué necesitan, se observa que es necesaria la carta factura para obtener la tarjeta de circulación de la unidad vehicular y que estos documentos tienen los siguientes datos de información :

Datos de factura :
- Folio :
- Compañía :
- Fecha de Factura :
- Valor Total :

Datos del propietario :
- Nombre :
- Domicilio : Calle, Núm., Colonia, Etc.
- Ciudad :
- C.P.:
- Estado :
- Municipio o Delegación :

Datos del vehículo :
- Tarjeta de Circulación Núm. :
- Marca :
- Submarca :
- Tipo :
- Modelo :
- Núm. De Serie:
- Núm. De Motor :
- Núm. De Puertas :
- Núm. De Cilindros :
- Color:
- Capacidad :
- Placas :
- Servicio :
- Uso :

Podemos tomar como inicio, los "Datos del vehículo" y lo proponemos como la entidad ***Vehículos*** como primer acercamiento. Tenemos las primeras propiedades o características de está entidad (***atributos***) como son : placa, serie, motor, marca, submarca, tipo, etc. Sin olvidar la propiedad consecutivo y por lo comentado se tienen los identificadores candidato de los atributos : consecutivo, placa, serie, motor. Así como el identificador principal (IP) : consecutivo y los identificadores alternativos (IA): placa, serie, motor. Por lo tanto, tenemos la primera propuesta de entidad con sus atributos :

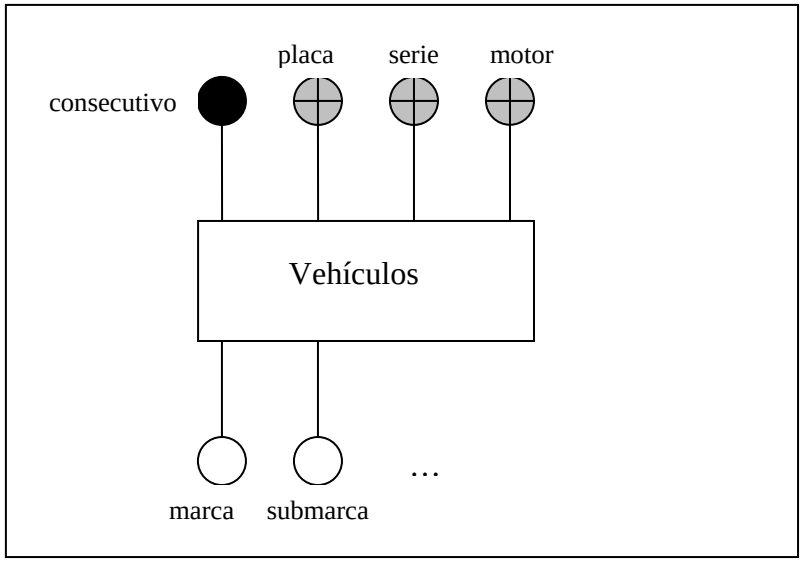


Fig. 6.1. Entidad Vehículos.

Se toma esta información como datos para una unidad vehicular y pueden surgir otros.

- Pueden haber otros datos como la Póliza de seguros para la unidad, color de engomado y aseguradora, siendo estos últimos propiedades de la entidad ***Vehículos*** pero también indica la existencia de la entidad ***Póliza*** donde tendremos las propiedades póliza y aseguradora.
- Cuando se adquiere un bien en la Universidad surge un número de control, este es el inventario, que se tomará como propiedad del vehículo.
- La DGSG asigna los vehículos para su uso a cualquier entidad de la Universidad que lo solicita, se crea la propiedad dependencia y clave de dependencia (llamada clave únicamente). Cabe notar que no siempre se proporciona quien maneja la unidad por lo que solo se hace referencia a dónde esta ubicada. Formarán entonces "Todas la dependencias de la Universidad" el ***dominio*** de posibles valores que tomarán las propiedades dependencia y clave de la dependencia.
- Dentro de la DGSG se necesita saber las condiciones en las que se encuentra la unidad, si está en taller o circulando por lo que se necesita saber su propiedad estado, que puede ser Activo, Baja, Taller, Asegurado por evento, siendo estos el ***dominio*** de los posibles valores que tomará la propiedad estado y que estará ubicado en la entidad ***Vehículos***.

- En la comunicación interna de la DGSG, cualquier movimiento a una unidad se realiza por medio de un documento oficial llamado “cuadrado” y para darle seguimiento se anexan las propiedades : cuadrado y fecha de emisión a la entidad cuadrados.

De está información se deduce que :

- La entidad ***Vehículos*** tendrá la mayor parte de información para el control de los vehículos en el sistema y que se están definiendo los ***atributos*** de la entidad así como sus posibles dominios.
- Que existen otras entidades : ***Póliza***, ***Dependencias*** y ***Cuadrados*** con algunos de sus ***dominios*** y ***atributos*** de ellas.

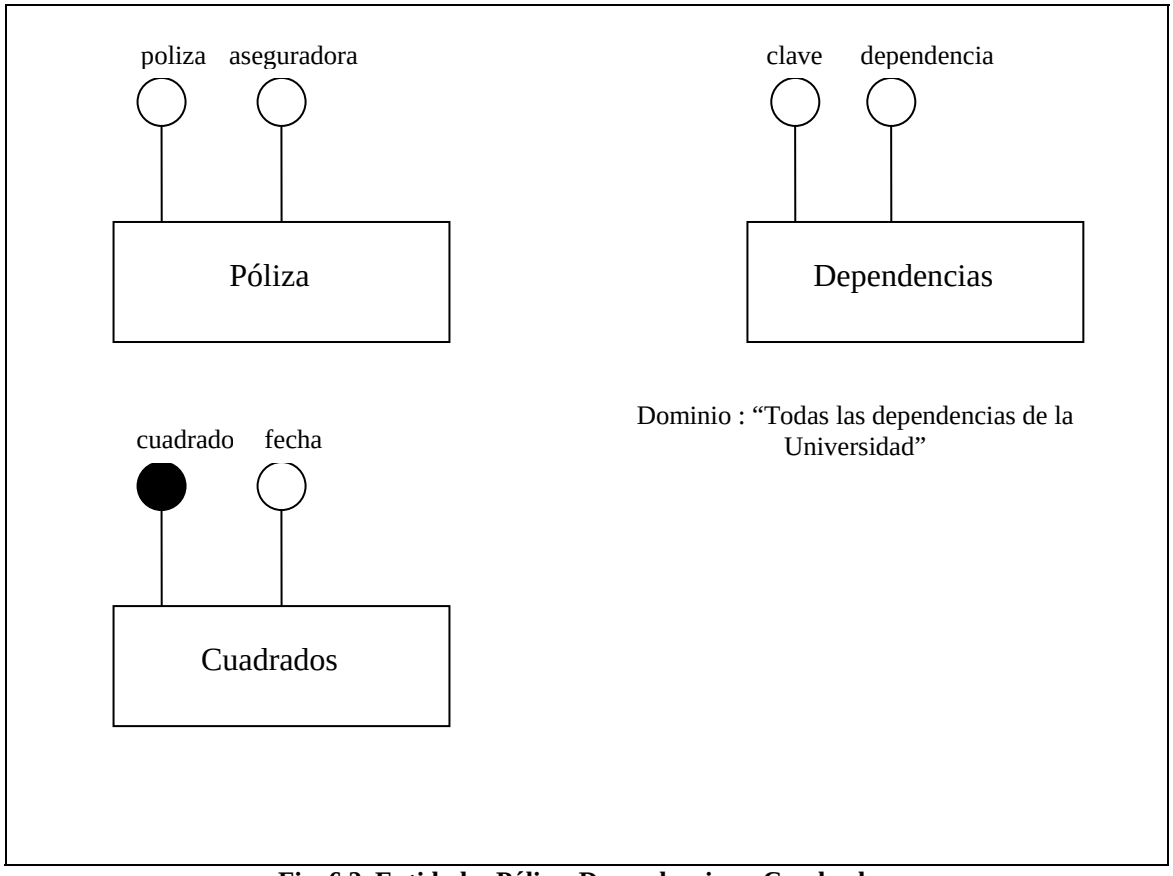


Fig. 6.2. Entidades Póliza, Dependencias y Cuadrados.

Para que al momento de capturar la información no se cometan errores de escritura, se propone identificar catálogos como el catálogo que puede formar el dominio de “Todas la dependencias de la Universidad” de la entidad Dependencias o “todas las marcas de vehículos que existen en el mercado”.

- Patrimonio Universitario proporciona catálogos de dependencias con los campos : clave de dependencia y nombre de dependencia y nos proporciona el establecimiento de la entidad Dependencias con los atributos : clave de dependencia y dependencia. Con la clave de la dependencia como IP.
- Para los trámites y para la aseguradora son necesario conocer marca, submarca, tipo de vehículos (entre otra información). Se investiga esta información en el mercado vehicular para formar el catálogo que contrendrá está información. Se crea la entidad ***Características*** con ***atributos*** textocar y característica, ***dominio***

para textocar : marca, submarca y tipo, y sus valores los que definen a cada uno en el mercado vehicular. Con IA textocar.

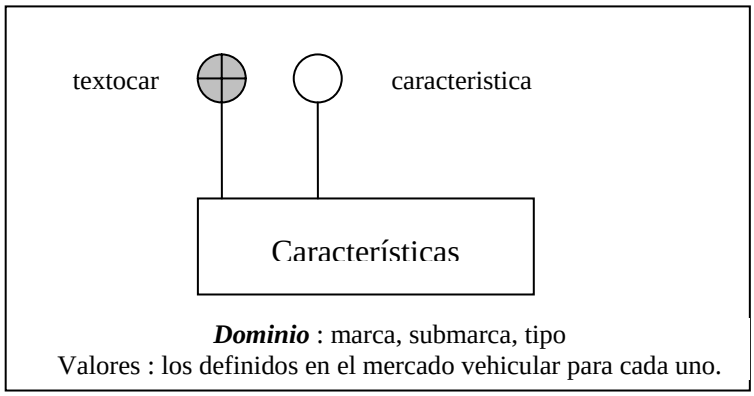


Fig. 6.3. Entidad Características.

- Como Patrimonio tramita el seguro de los autos, solo comunica el periodo de cobertura, el núm. de póliza, el nombre de la aseguradora y teléfonos de reporte, por lo que esto establece la entidad Pólizas cuya información num. de póliza será el atributo póliza y será nuestro IP de la entidad. También se integran los atributos cobertura y telrep.

Por último se estructuran otras entidades necesarias :

- Puesto que el cliente necesitará conocer cualquier movimiento en las unidades vehiculares, se crea la entidad Histórico, con las propiedades (hasta el momento):
  - textoreg que contendrá el registro del movimiento del vehiculo.
  - numcon que contendrá una copia de la propiedad consecutivo del vehículo afectado.
  - numcuadrado que contendrá el "cuadrado" que lo afecta.
- Para saber quien entra al sistema o cuál es el núm. de control (consecutivo), vehicular se crea la entidad Control con las propiedades numero que se refiere a cuando se dá de alta un vehículos nuevo en el sistema y qué número tendrá el consecutivo de control de unidades vehiculares del sistema.
- Para controlar el acceso de los usuarios surge la entidad Usuarios cuya propiedad clave_usu se refiere a "usuario" que identifica a un usuario del sistema, para lo cual contará con la propiedad nombre de usuario y módulo al que pertenece.

6.1.2. Interrelación entre entidades.

A.- Como en las necesidades del cliente es necesario saber a que dependencia es asignada una unidad vehicular, tenemos la relación entre las entidades Vehículos y Dependencias. La interrelación clave concierne a identificar la dependencia.

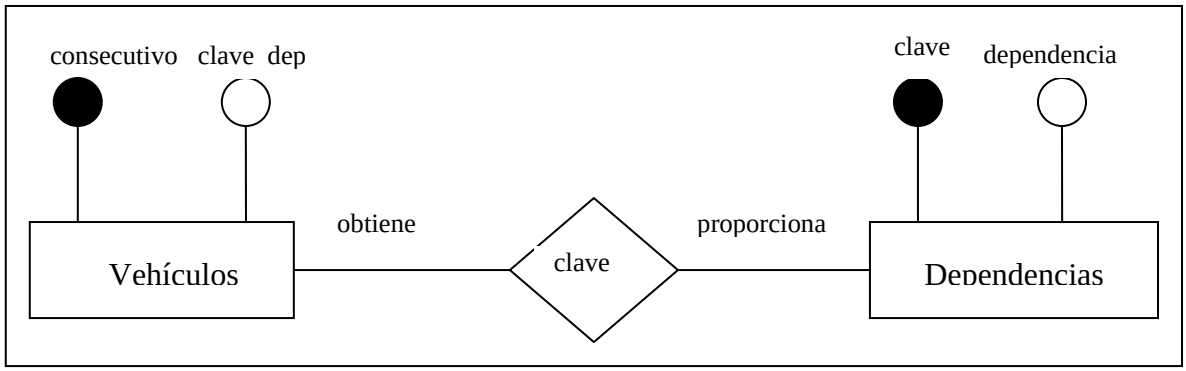


Fig. 6.4. Interrelación entre entidades Vehículos y Dependencias.

Considerando que :

1. Como el número de entidades que participan en esta interrelación son dos, su grado es 2.
2. Puesto que el numero de ejemplares de la entidad Vehículos es 1 y puede estar asociado por la interrelación con más de un ejemplar de la entidad Dependencias, la interrelación es 1:N.
3. La cardinalidad de un tipo de entidad se ve como dos subconjuntos en los cuales Vehículos (E1) puede estar interrelacionado con uno o más ejemplares de Dependencias (E2). Se obtiene la interrelación clave de Vehículos y nos dice que cada unidad vehicular pertenece a un único ejemplar de Dependencias luego entonces E1 es 1:1 y que pueden existen ejemplares de Dependencias que no tienen ningún ejemplar en Vehículos, se tiene entonces que E2 es 0:N

Se tiene que esta interrelación queda de la siguiente manera :

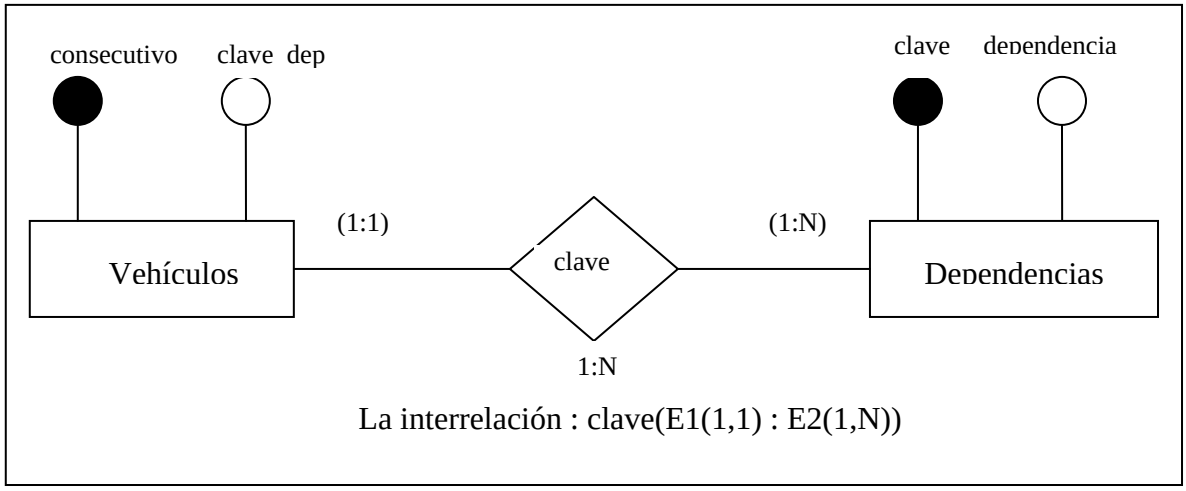


Fig. 6.5. Cardinalidad de interrelación.

Para la interrelación entre la entidad Vehículos y las entidad Caracteristicas sucede en forma similar.

B.- La entidad Control con respecto a la entidad Vehículos es débil debido a que un ejemplar de Control depende de la existencia de un ejemplar de Vehículos (depende de la existencia de una unidad nueva). Su interrelación “obtiene” solicita el número de control para generar el consecutivo que quedará en la entidad Vehículos. Su tipo de correspondencia de la interrelación es 1:1 debido a que un ejemplar de Vehículos (en el

atributo consecutivo) solo esta relacionado con un ejemplar de Control. Sin embargo es débil porque la entidad Control lo es. Pero hay dependencia en existencia debido a que el ejemplar de la entidad Control no puede existir si desaparece el ejemplar del tipo de entidad Vehículos por lo que la interrelación está etiquetado con E. Por último como todo vehículo nuevo tiene un único identificador que lo distingue de otro la cardinalidad con respecto a Vehículos (E1) es 1:1 y como ya conocido el consecutivo se asignará a un único vehículo la cardinalidad de Control (E2) es 1:1 luego entonces la interrelación "obtiene" es : obtiene(E1(1:1) : E2(1:1)).

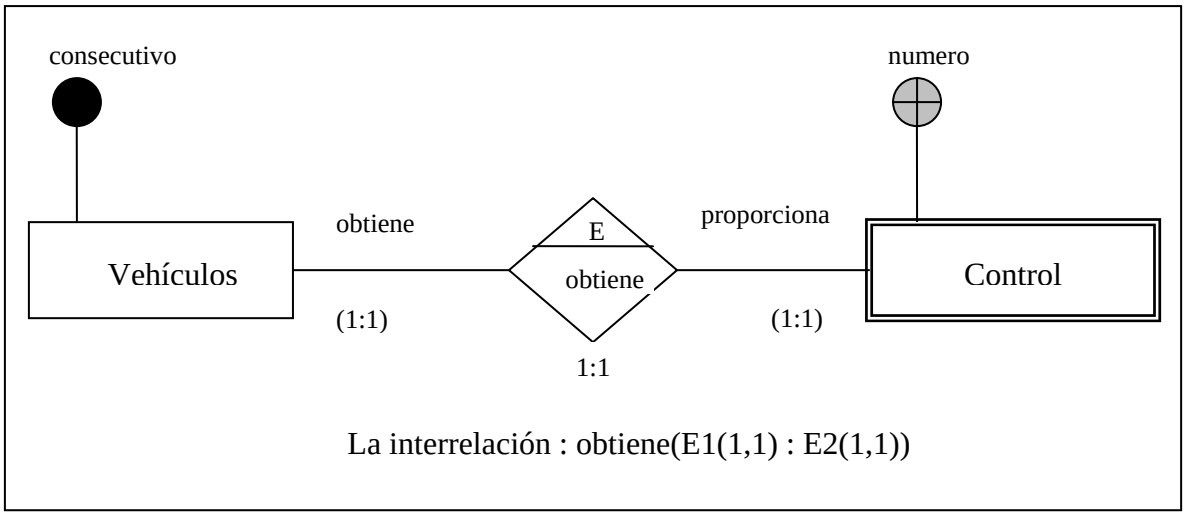


Fig. 6.6. Cardinalidad con interrelación en existencia.

6.1.3. Transformación al diseño lógico.

Siguiendo los pasos señalados en el capítulo 4 "Reglas concernientes al modelo básico". Tomamos el rubro concerniente a 6.1.2. inciso A.

1.- Transformación de entidades.

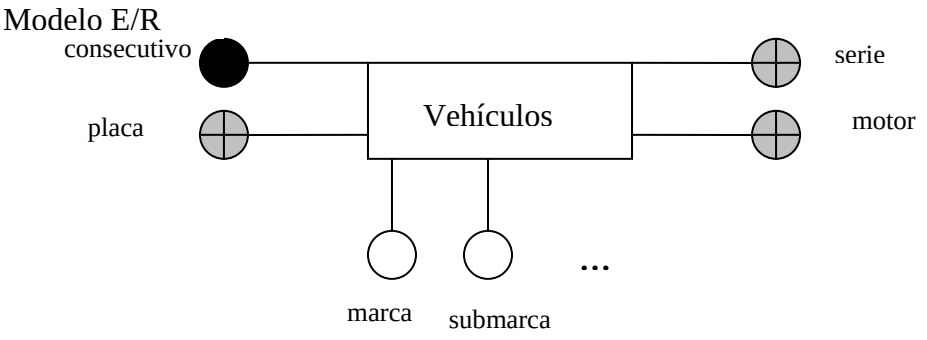
La transformación de las dos entidades es directa, es decir se transforman entidad Vehículos en tabla Vehículos y entidad Dependencias en tabla Dependencias.

2.- Transformación de atributos de entidades.

En la entidad Vehículos se identifican a todos los atributos como columnas. De estos la propiedad consecutivo será una columna que contendrá al IP y que será la clave primaria; las propiedades serie, motor y placa serán las columnas que contendrán a los IA y que serán las claves secundaria o únicas porque no tienen duplicidad de ejemplares; el resto de atributos serán columnas.

En el lenguaje lógico estándar (LLS) utilizado (SQL) y tomando 1 y 2 será :

CREATE TABLE Vehiculos(consecutivo,placa,serie,motor,marca,submarca,…., PRIMARY KEY (consecutivo),UNIQUE(placa,serie,motor));



Modelo Relacional

Tabla : **Vehículos**

consecutivo	placa	serie	motor	marca	submarca	…

Clave primaria **: consecutivo**

Vehículos (<u>consecutivo</u>, placa , serie , motor , marca , submarca, …)

Fig. 6.7. Transformación de la entidad Vehículos.

Para la entidad Dependencias se siguen las reglas 1 y 2 para generarla.

3.- Transformación de la interrelación.

Aplicando la regla 4.II.b. del capítulo 4, tenemos que se trata como una relación N:M, sin embargo la clave primaria de la relación creada es sólo la clave primaria de la tabla a la que le corresponde la cardinalidad N. y que por regla 5 del capítulo 4 pasa el atributo de la interrelación a la columna de la tabla que se formará.

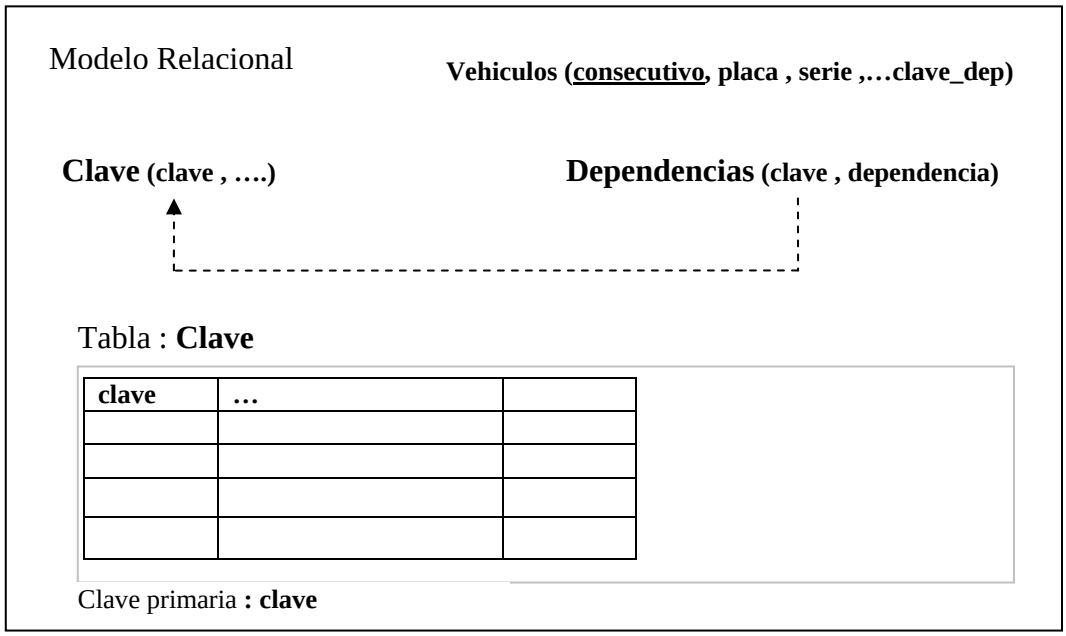


Tabla : **Clave**

clave	…	

Clave primaria **: clave**

Fig. 6.8. Transformación de interrelación.

En el lenguaje lógico estándar (LLS) utilizado (SQL), tomando 1, 2 de este apartado y regla 6 capítulo 4 el valor de clave es mayor a 10000 y menor a 99999, será :

CREATE TABLE Clave(clave,….,) PRIMARY KEY (clave) WHERE clave BETWEEN (10000 AND 99999));

4.- Transformación de dominios.

Con la información recabada, la estructura de los atributos de la entidad Vehículos, se estima cómo se deberán almacenar los datos en longitud y tipo, dándoles a algunos de ellos un margen de crecimiento futuro. Se toma en cuenta que :

- Hace 4 años el número de serie era de 15 dígitos alfanuméricos y que al día de hoy es de 19, por lo que esperando que se pueda usar un tiempo razonable el sistema, se amplia el campo serie a 25 dígitos alfanuméricos. De igual manera para el motor.
- El número de unidades que adquiere la Universidad es en promedio 50 por año, por lo que si se tienen en existencia 2700 unidades, con un entero de 6 dígitos el sistema podrá solventar el control de unidades con el campo consecutivo.
- Existen datos que no van a cambiar de tamaño o tipo por un tiempo razonable al menos que los modifique Patrimonio como es la clave de la Dependencia que es de cuatro dígitos.
- Recabando información de los nombres que les han dado a las marcas, submarcas y/o tipos a los autos las compañías automotrices y tratando de unificar los datos que se manejan para pago de tenencia, verificación, asegurar unidad, etc., se asigna una longitud de 25 caracteres alfanuméricos a los campos marca, submarca y tipo.
- Hay campos que tienen una longitud “fija” que no se rebasa como modelo, num. de puertas y num. de cilindros.
- Las placas son de 8 dígitos alfanuméricos.

Los atributos de la entidad Vehículos toman una estructura como la que sigue :

Nombre del atributo	Tipo	Longitud	Decimales
consecutivo	Entero Largo	6	
foliofactura	Alfanumérico	20	
cuadrado	Alfanumérico	8	
fechacuadrado	Fecha	8	
placa	Alfanumérico	8	
serie	Alfanumérico	25	
motor	Alfanumérico	25	
clavedep	Entero	4	0
dependencia	Carácter	50	
marca	Carácter	25	
submarca	Carácter	25	
tipo	Carácter	25	
modelo	Entero	4	
cilindros	Entero	1	
puertas	Entero	1	
uso	Carácter	30	
servicio	Carácter	30	
inventario	Entero	10	
engomado	Carácter	10	
estado	Carácter	30	
poliza	Alfanumérico	20	
aseguradora	Carácter	70	
periodopoliza	Carácter	30	

El resultado del rubro concerniente a 6.1.2. inciso A son tres tablas : Vehículos, Dependencias y Clave con sus respectivos campos.

Otras interrelaciones del sistema son :

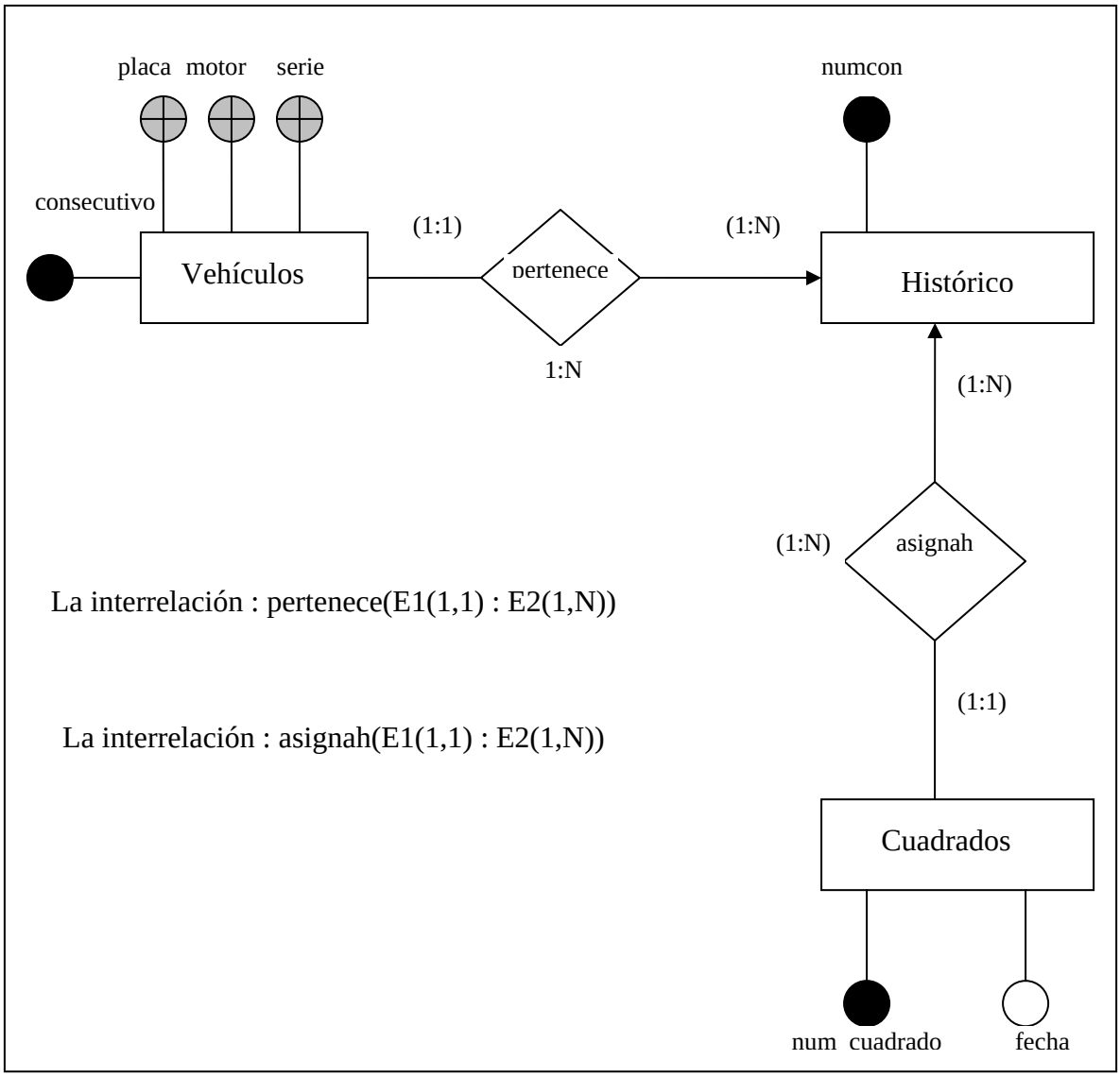


Fig. 6.9. Interrelación entre entidades Vehículos, Histórico y Cuadrados.

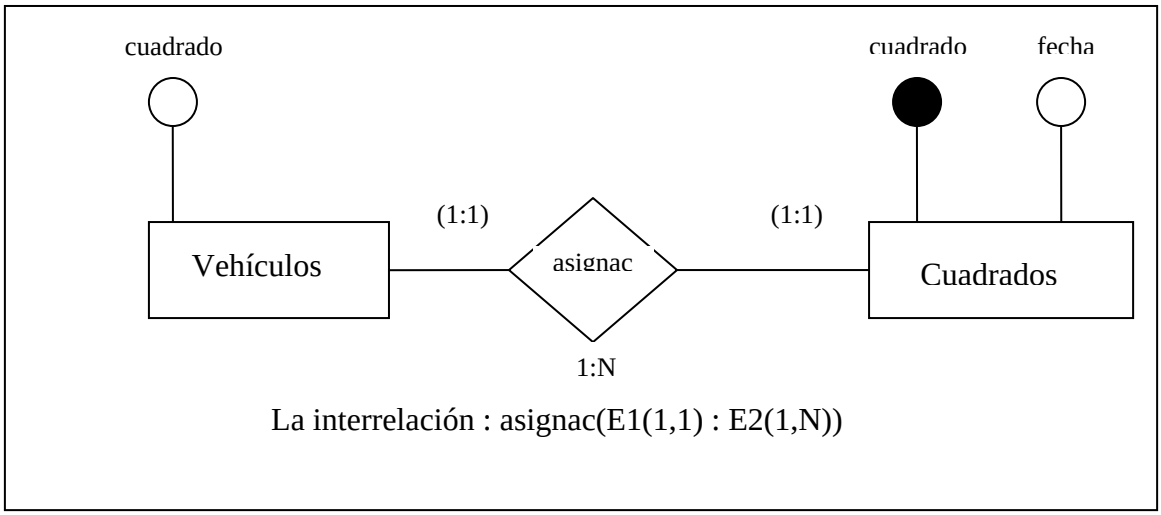


Fig. 6.10. Interrelación entre entidades Vehículos y Cuadrados.

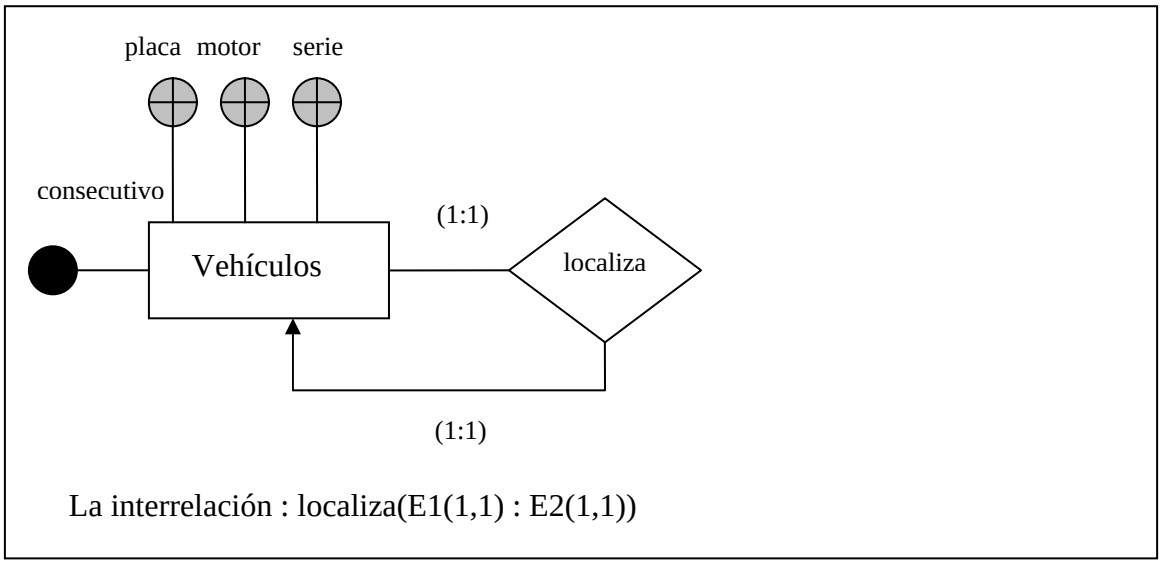


Fig. 6.11. Interrelación de entidad Vehículos consigo misma.

Capítulo 7.- Diseño de la aplicación.

Aplicando los fundamentos de diseño del capítulo 3, se obtiene la siguiente arquitectura:

7.1. Arquitectura de tres capas.

Cada usuario solicitará su acceso por medio de una clave de identificación, tendrá un menú de trabajo relacionado a los casos de uso de la Fig. 5.4. (Diagrama general de casos de uso) y podrá usar los datos de las tablas generadas en el capítulo 6.

Se utiliza una capa de interfaz donde estarán las interfaces del sistema : Interfaces de usuarios, formatos de impresión, etc. que son la forma de presentación del sistema con el usuario. Esta capa es la encargada de comunicar información y captura de datos. Una interfaz de esta capa es la de la Fig. 4.6. “Pantalla de Alta vehicular”.

La capa de interfaz se comunica con la capa de gestiones donde residen los programas que se ejecutarán. En esta capa se reciben y presentan los resultados en la capa de interfaz de :

- Las peticiones del usuario como busquedas y almacenaje de información,
- Procesos de movimientos de datos para presentar submenús como pueden ser las caracteristicas o tipos de automóviles que se podrán elegir para una unidad vehicular en el sistema.
- Procesos de asignación única a las unidades vehiculares o asignación de clave de usuario a los movimientos de datos en el sistema, y
- Procesos de filtros de información para generar reportes vehiculares.

La capa de gestión solicita a una tercera capa (capa de datos) el almacenaje o recuperación de información residente en ella. Recibe las solicitudes de almacenamiento y recuperación de información de una unidad vehicular desde la capa de gestiones.

Utilizando el modelo de capas, la arquitectura de software para el sistema vehicular tendrá las siguientes capas :

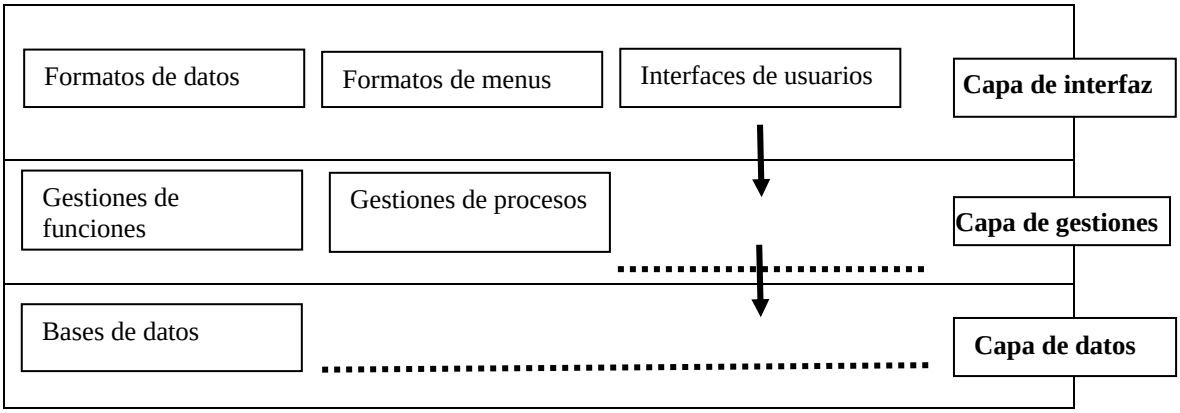
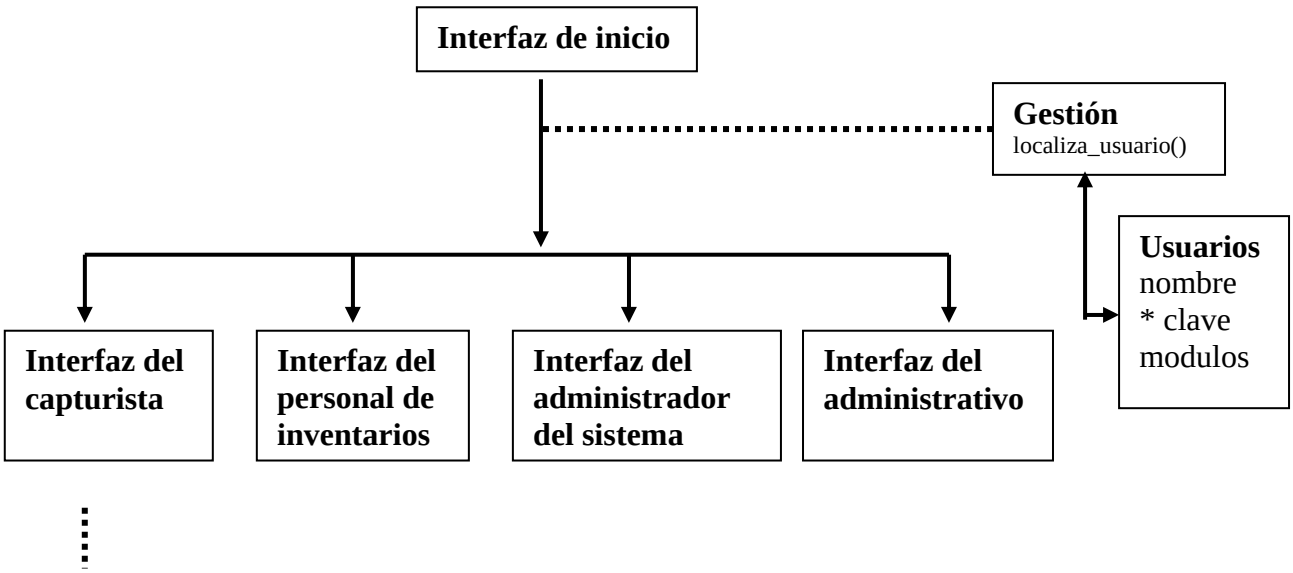


Fig. 7.1. Arquitectura de capas del sistema vehicular.

Expandiendo esta arquitectura se muestran los subsistemas del Diagrama general de casos de uso Fig. 5.4. :



7.2. Diagrama parcial de la interfaz del sistema.

El diagrama del sistema vehicular esta compuesto por las interfaces de los actores del sistema vehicular del Diagrama general de casos de uso Fig. 5.4.

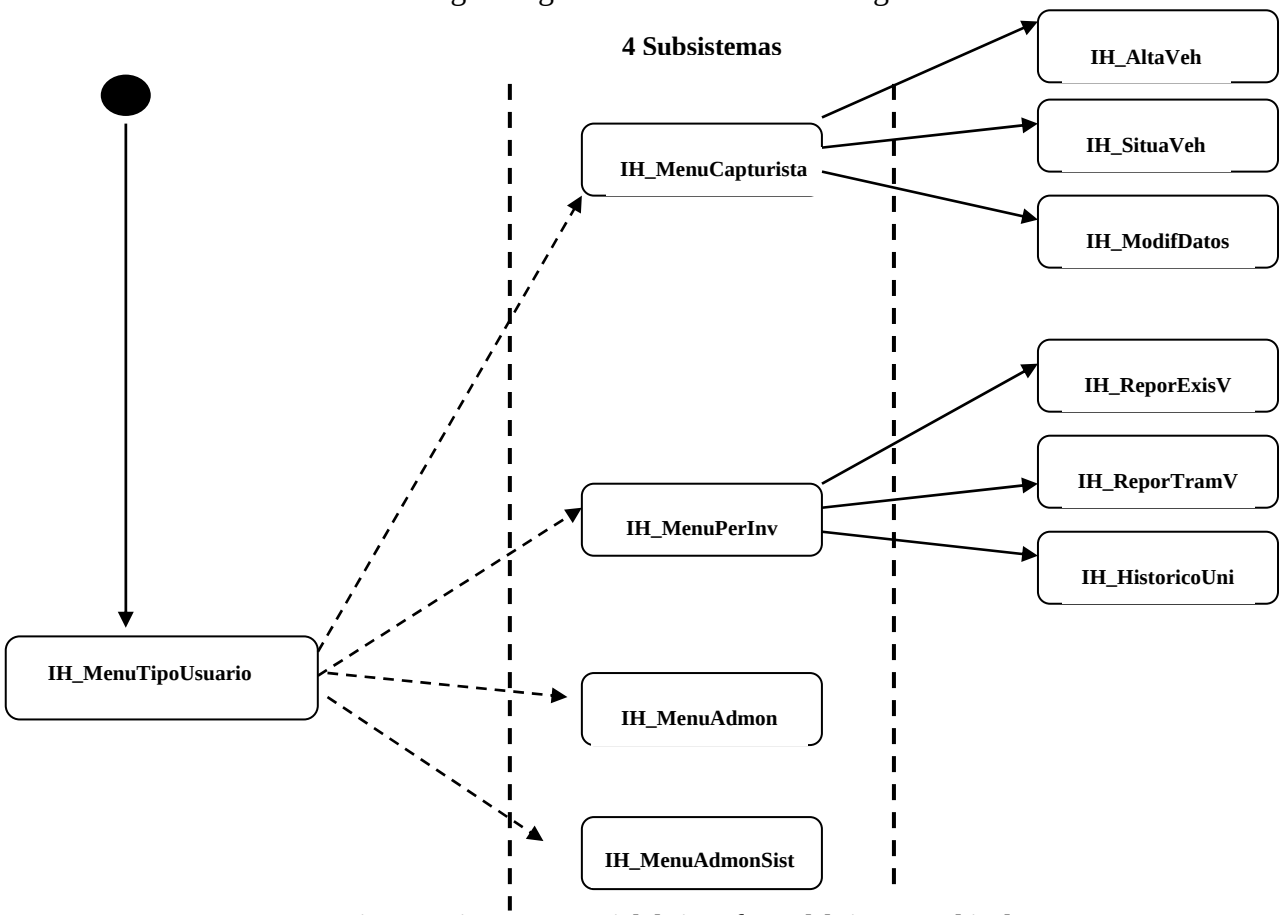


Fig. 7.2. Diagrama parcial de interfaces del sistema vehicular.

Nos proporciona los cuatro subsistemas : Capturista, Personal de inventarios, Administrativo y Administrador del sistema.

7.3. Capas del subsistema capturista.

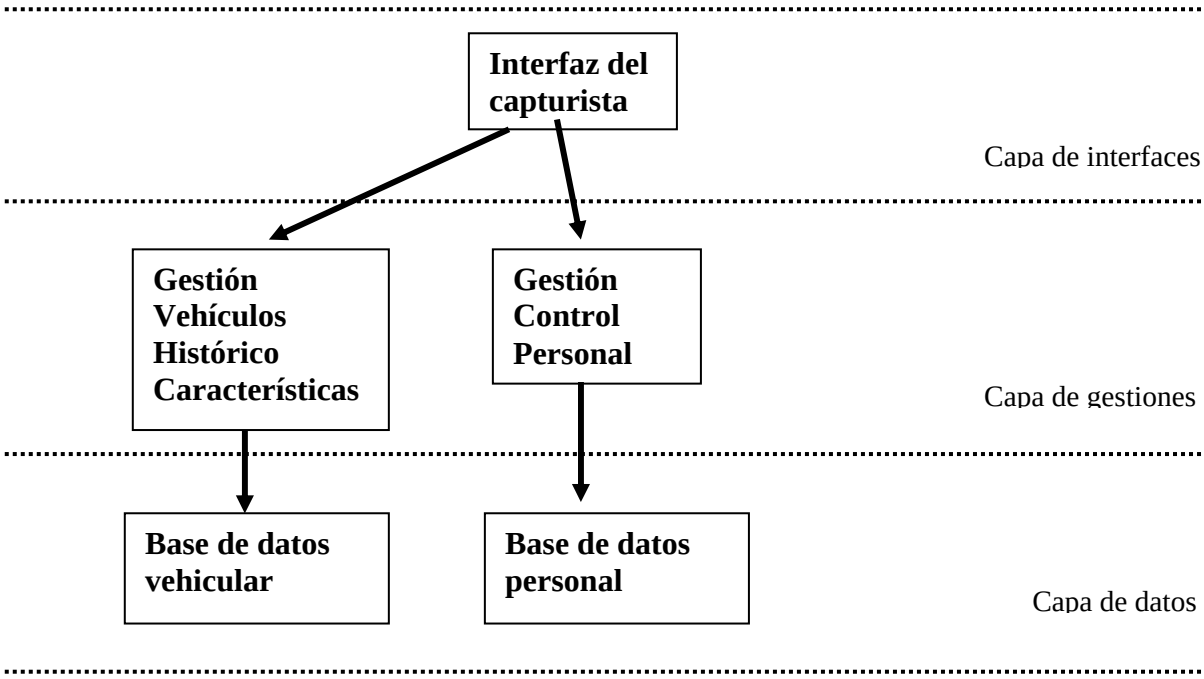


Fig. 7.3. Capas del capturista.

7.4. Identificación de funciones y datos.

Apoyado por el diagrama de secuencia detallado de los casos de uso encontramos las funciones que se usan en el proceso. En el caso del diagrama de secuencia del actor capturista, se solicitan al usuario los datos : placa, motor y serie para identificar a una unidad vehícular. Por lo que una función llamada dameDatos() con variables placa, motor y serie es activada cuando se solicita de la interfaz del menú (IHMenus) a la interfaz de alta vehicular (IHAltaVeh). Cuando se proporcionan los datos se solicita a la capa de gestiones la activación de la función buscaUni(), que solicita dichos datos a la capa datos en la tabla TVehiculos y al no encontrarlo solicita asignación de control de unidad por medio de la función asignaCon(), cuyo datos se controla en la tabla TControl. Continuando, el diagrama de secuencia detallado se expresa :

Diagrama de secuencia detallado.
Actor : capturista.
Caso de Uso : Alta vehicular. Flujo normal.

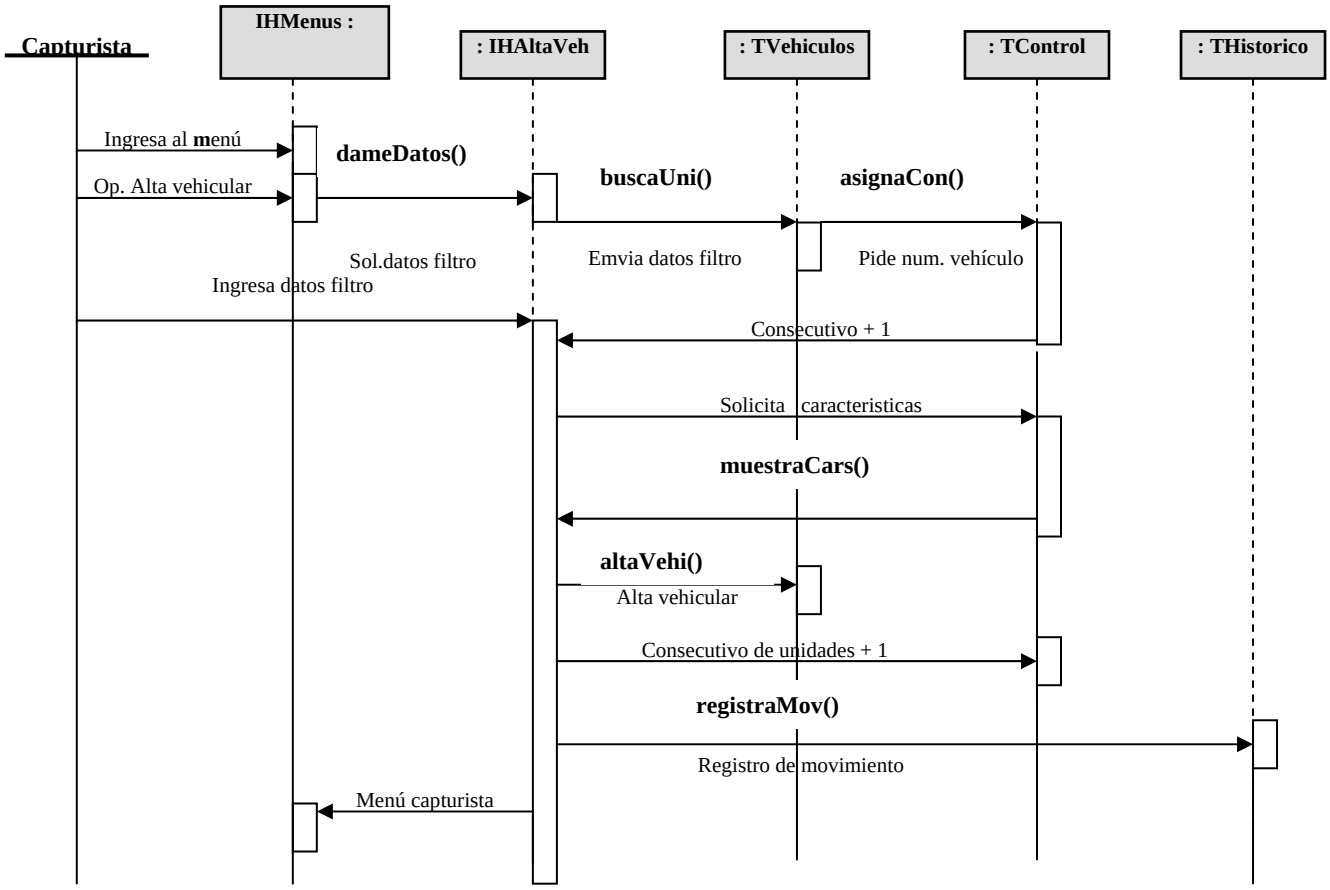


Fig. 7.4. Diagrama de secuencia detallado del caso de uso alta vehicular.

Proporcionando las funciones para este caso de uso.

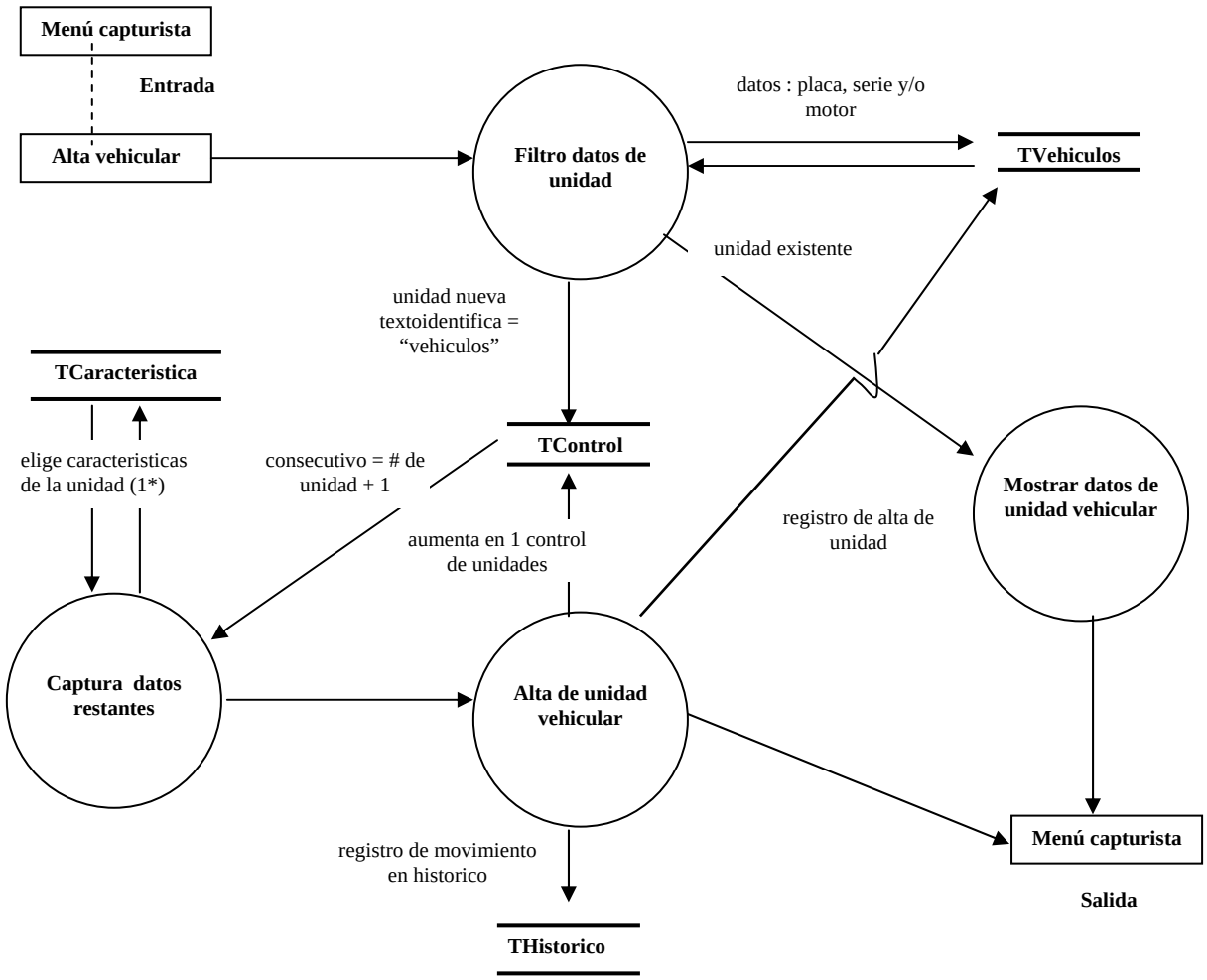
Localizadas las funciones y contando con las tablas, aplicamos los diagramas de flujo de datos a las actividades que satisfacen los casos de uso, observando el traslado de datos entre procesos (¿Qué datos se necesitan para leer/escribir?) y las unidades de almacenamiento (tablas donde reside la información).

Las actividades para el caso de uso alta vehicular (Fig. 7.4.), serían :

1. Del menú capturista se elige alta vehicular.
2. Identificados los datos filtro que identifican a una unidad, se busca su existencia en la tabla TVehiculos.
3. Si existen, se muestan los datos restantes que están almacenados en la tabla Vehículos y se regresa al menú del capturista.
4. Si no existen, entonces es una unidad nueva y se pide el consecutivo de control vehicular a la tabla TControl. Se almacena en una variable y se muestra el formato de captura de datos restantes.
5. Para llenar este formato se ligan los datos (como marca, submarca, tipo, etc.) de la tabla : TCaracterísticas y/o se capturan a mano.
6. Teniendo la captura de datos de la unidad, se da de alta. Este proceso realiza :

6.1. Aumentar en uno al campo consecutivo de la tabla TControl, que es el número que se asigna a la nueva unidad.
6.2. Registra el movimiento en la tabla THistórico.
6.3. Respalda la información de la unidad vehicular en la tabla TVehículos.
7. Regresa al menú del capturista.

Diagrama de flujo de datos detallados.
Actor : capturista.
Caso de Uso : Alta vehicular.



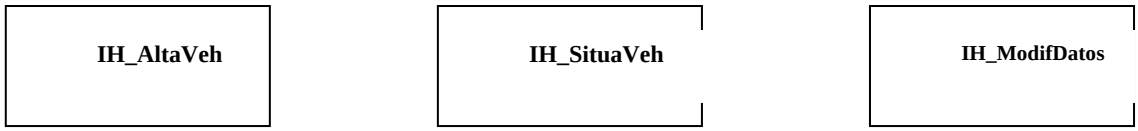
(1*) El campo textocar puede ser marca, submarca y/o tipo. Se contesta con características dependiendo de la elección.

Fig. 7.5. Diagrama de flujo de datos detallados del caso de uso alta vehicular.

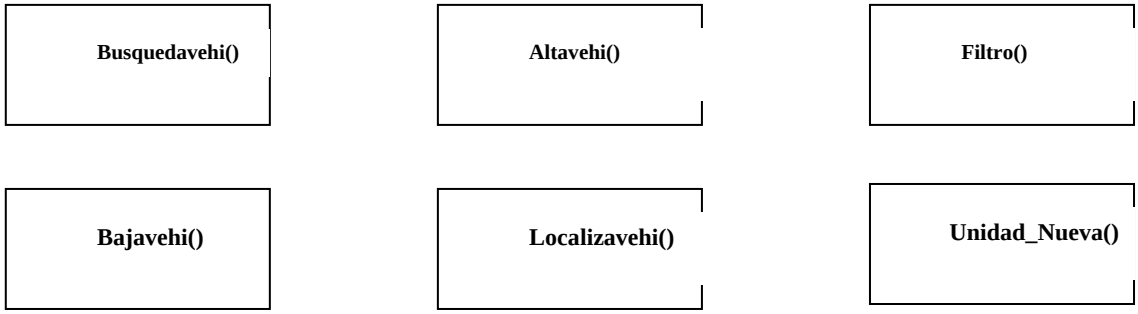
Este diagrama muestra los datos que se transfieren de proceso a proceso y se puede identificar aquellos que tienen una relación con las tablas.

7.5. Componentes del subsistema capturista.

A.- Componentes de la Interfaz del Capturista.



B.- Componentes de Gestión Vehículos.



C.- Componentes de Bases de Datos.

7.6. Detalle del caso de uso alta vehicular

Este detalle se puede llevar a cabo realizando el pseudocódigo.

El pseudocódigo para el caso de uso alta vehicular será :

Inicia programa Altavehicular
Definición de variables :
   De tipo alfanumérico : tplaca, tmotor, tserie
   De tipo númerico : ncontrol
Inicializar variables
   tplaca, tmotor, tserie = “”
   ncontrol=0
Dame datos para filtro(tplaca,tmotor,tserie)
‘Localiza número de unidad si existe y lo almacena en ncontrol’
ncontrol=**busquedavehi(tplaca,tmotor,tserie)**
Si ncontrol > 0 ‘Existe la unidad’
   Mostrar_datos(ncontrol)
Sino ‘No existe unidad, buscar consecutivo para unidad nueva’
   ncontrol**=Unidad_nueva()**
   Captura_datos(ncontrol)
Termina condición ncontrol>0

Termina programa de Altavehicular

‘******** Procesos **********’**

Proceso busquedavehi(tp,tm,ts)
‘Definición de variables’
De tipo numérico : existe

‘Inicializa variable’
existe=0
Si tp diferente a espacios ‘Proporcionaron dato de placa’
existe=**Filtro(tp,’placa’)** ‘Localiza placa de unidad’
Sino
Si tm diferente a espacios ‘Proporcionaron dato de motor’
existe=**Filtro(tm,’motor’)** ‘Localiza motor de unidad’
Sino
Si ts diferente a espacios ‘Proporcionaron dato de serie’
existe=**Filtro(ts,’serie’)** ‘Localiza serie de unidad’
Sino
Mensaje solicitando datos
Fin condición ts
Fin condición tp
Fin condición tp
Regresar existe
Fin de proceso busquedavehi

‘********’

Proceso Filtro(fdato,fcampo)
‘Definición de variables’
De tipo numérico : localizado, numerodevehiculo
‘Inicializa varibles’
localizado=0
numerodevehiculo=0
localizado=**realiza Query en tabla vehiculos bajo “fdato” y campo “fcampo”**
Si localizado > 0
numerodevehiculo=**valor del campo numvehi**
‘numvehi contiene el numero de control del vehículo para
Cualquier movimiento de éste’
Fin condición localizado
Regresa numero de vehículo
Fin de proceso Filtro

’********

Proceso Unidad_nueva()
‘Definición de variables’
De tipo numérico : unidadnueva, localizado
‘Inicializa variables’

unidadnueva=0
localizado=0
localizado=**realiza Query en tabla control bajo el campo “texto” y frase “vehiculos”**
Si localizado > 0
unidadnueva=**valor almacenado en campo “numero” aumentado en uno**
Fin condición localizado
Regresa unidadnueva
Fin de proceso Unidad_nueva
‘********’**

Conclusiones

En este trabajo se utilizó un proceso de la Ingeniería de software a un problema administrativo real para su automatización.

En la práctica, cuando nos enfrentamos a realizar aplicaciones de cómputo, aplicamos técnicas o herramientas que conocemos, aprendemos o quizá empíricamente "adaptamos". Muchas de estas formas de desarrollo nos llevan a tener aplicaciones funcionales pero sin un sustento y en muchas ocasiones sólo el código es el que está a la vista. Por lo mismo, llegan a ser engorrosas y difíciles de entender y mantener después de algún tiempo.

El uso de técnicas de la Ingeniería de software, nos propone un desarrollo disciplinado al desarrollo de una aplicación. Sus técnicas se asemejan con las que había hecho empíricamente en otras aplicaciones pero que en la aplicación de control vehicular se realiza siguiendo dos fases del proceso. En un futuro se podrá entender la forma en que fue desarrollada para poder aprovecharse y mantenerse más actual si fuera necesario.

Aunque no se resalta, las actividades como dialogo, búsqueda y estructuración de la información de la fase de análisis de requerimientos absorben más del 60% de tiempo del desarrollo de la aplicación. La comunicación y el modelado de necesidades con UML es un trabajo tedioso y laborioso en sus primeras sesiones.

La negación del personal a utilizar computadoras en sus actividades administrativas tiene como consecuencia que la solución de una aplicación para su trabajo diario, sea para el usuario, de un mínimo de esfuerzo en su acoplamiento y que su familiarización sea rápida. Por otro lado, la aplicación debe ser independiente del personal administrativo, es decir, que puede haber cambios de personal y el sistema debe seguir respondiendo a las necesidades para las que fue creado.

Una parte crítica de una aplicación es la estructuración y respaldo de la información en bases de datos. Para ello no se debe olvidar la existencia de un administrador del sistema.

Bibliografía

[Armo] Armour, Frank, Granville Millar. Advanced use case modeling : Software systems. Addison Wesley, E.U.A., 2001.

[Barr] Barroca, Leonor, John Hall and Patrick Hall. Software architectures : Advances and applications. Second Edition. Springer, E.U.A., 2000.

[Booc] Booch, Grady, James Rumbaugh and Ivar Jacobson. The unified modeling language : User guide. Massachusetts, E.U.A. : Addison-Wesley, 1997.

[Brau] Braude, J. Eric. Ingeniería de software : Una perspectiva orientada a objetos. México : Alfaomega, 2005.

[Budg] David, Budgen. Software design. England : Addison-Wesley, 1993.

[Cast] Castaño, Adoración de Miguel, Mario Piattini. Diseño de bases de datos relacionales. México, Alfaomega Grupo Editores, S. A. de C.V., 2004.

[Diet] Dietrich, Suzanne W. Understanding relational database query languages. New Jersey, U.S.A., 2001.

[Fowl] Fowler, Martin. UML : Gota a gota. Addison Wesley Longman de México, 1999.

[Gali] Galitz, W., The essential guide to user interface design : An introduction to GUI principles and techniques, Nueva York : John Wiley & Sons, 1996.

[Gram] Gram, Chistian and Gilbert Cockton. Design principles for interactive software, London : Chapman and Hall, 1996.

[Harr] Harrington, Jan L., Relational database design clearly explained. U.S.A., 1998.

[Jaco] Jacobson, Ivar, Magnus Christerson. Object-oriented software engineering : A use case driven approach (Addison-Wesley Object technology series). England : Addison-Wesley, 1994.

[Kauf] Kaufman, Terry. Information modeling and relational databases from conceptual analysis to logical design. U.S.A. 2001.

[Leac] Leach, Ronald J., Introduction to software engineering. Florida, U.S.A., CRC Press LLC, 2000.

[Pres] Presuman, S. Roger. Ingeniería del software : un enfoque práctico. Madrid,, McGraw-Hill, 1988.

[Scha] Schach, Stephen R. Análisis y diseño orientado a objetos con UML y el proceso unificado. Mc Graw Hill, 2005.

[Stev] Stevens, Perdita, Rob Pooley. Utilización de UML en ingeniería del software con objetos y componentes, Madrid, España. Addison Wesley, 2003.

[Somm] Sommerville, Ian. Software engineering, 6ta. Edition, E.U.A. Pearson Education, 2001..

[Vies] Viescas, John L. Guia completa de Microsoft Access para Windows 95. McGraw-Hill, España, 1996.