8

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES
"CAMPUS ARAGÓN"

MANIPULACIÓN DE CÓDIGOS DE BARRAS POR MEDIO DE UNA PALM EN UNA INSTITUCIÓN BANCARIA

T E S I S

QUE PARA OBTENER EL TITULO DE :

INGENIERO EN COMPUTACIÓN

P R E S E N T A :

JORGE ADRIÁN CONTRERAS GAMBOA

ASESOR: ING. ROBERTO BLANCO BAUTISTA

MÉXICO

2002

PAGINACION DISCONTINUA

A mi mamá y mi abuelita:
Por todo su amor, comprensión, paciencia, apoyo y muchísimas cosas mas.

A mi papá, mi tía Alma Rosa y mis hermanos:
Por todo su amor, apoyo, preocupación y por todos los buenos consejos que me han dado.

A mis demás tíos y familiares:
Por todo el apoyo y aliento para que yo pudiera terminar mis estudios.

A mi novia:
Por todo su amor, apoyo, comprensión y toda su ayuda.

A todos mis amigos:
Por todo el interés mostrado para la culminación de mis estudios y por compartir conmigo tantos bellos momentos.

MANIPULACIÓN DE CÓDIGOS DE BARRAS POR MEDIO DE UNA PALM EN UNA INSTITUCIÓN BANCARIA

Capitulo 2
BASES DE DATOS RELACIONALES

Capitulo 3
PLATAFORMA PALM

Capitulo 4

BASE DE DATOS ULTRALITE

Capitulo 5

CODEWARRIOR

INTRODUCCIÓN

Objetivo

El objetivo de este trabajo de tesis es la creación de un sistema para llevar un control de las operaciones que se realizan con los billetes en una institución bancaria, utilizando a un dispositivo portátil que cuente con un sistema operativo que se permita la creación de aplicaciones personalizadas como lo es el SPT1700 que cuenta con un sistema operativo Palm OS. Cabe mencionar que el sistema propuesto en este proyecto de tesis, únicamente aplicara para billetes quedando las monedas fuera de su alcance, y dando oportunidad para que otro sistema sea elaborado para el control de las mismas.

El sistema constará de una aplicación creada en un SPT1700, que es un dispositivo con sistema operativo Palm OS y que cuenta con la característica de poder leer códigos de barras. La aplicación se desarrollará con la herramienta CodeWarrior, en conjunto con una base de datos de UlraLite que es especial para dispositivos portátiles.

El objetivo es que los billetes sean almacenados en unidades de empaque, donde cada una esta identificada con un código de barras. Con el dispositivo portátil SPT1700 se leerán los códigos de barras y se almacenarán en la base de datos UltraLite para después ser sincronizados con la base de datos consolidada que se encuentra en una computadora de escritorio. Con la aplicación desarrollada en el SPT1700 se llevará el control de las operaciones que se pueden hacer con los billetes en las unidades de empaque como armado, traspaso, cambio de estado físico, etc.

Justificación

El sistema deberá tener alcance en lugares donde no se pueden instalar computadoras de escritorio y que es donde se llevan acabo las operaciones con los billetes, las bóvedas. Es por esto que se utiliza un dispositivo portátil como el SPT1700 que no utiliza ningún tipo de cableado para poder funcionar. Además, éste dispositivo deberá contar con un sistema operativo que permita el desarrollo de aplicaciones personalizadas para poder desarrollar

la aplicación que llevará el control de las operaciones realizadas con las unidades de empaque y los billetes. El dispositivo deberá contar también con un lector de códigos de barras, ya que las unidades de empaque estarán identificadas por éste tipo de códigos.

El desarrollo de una aplicación personalizada es para poder llevar el control de las operaciones que se realizan con las unidades de empaque ya que éstas serán el contenedor de otras unidades de empaque. Las unidades de empaque son de varios tipos que van desde un mazo hasta una plataforma, donde, un mazo estará formado por 1000 billetes de una misma denominación, un paquete estará formado por 5 mazos si son billetes pequeños como los de $ 20 o $ 50, y de 4 si son billetes grandes como los de $ 100, $ 200 y $ 500. Una bolsa estará formada por 5 paquetes, las bolsas, a su vez, formarán a los contenedores en un conjunto de 5 y dos contenedores formarán una plataforma. Es por esto que será necesario el uso de una aplicación que permita realizar operaciones con estas unidades de empaque, como por ejemplo, armar un paquete a partir de un conjunto de mazos.

Se ha pensado que el desarrollo de una aplicación, también, daría la posibilidad de que se pudieran hacer varias lecturas de los códigos de las unidades de empaque de tal manera que se pueda realizar mas de una operación con estas unidades para que después se transfiera la información (todos los códigos leídos y las operaciones realizadas con ellos) a una base de datos en una computadora de escritorio, que a su vez, puede transferir su información a un base de datos en un servidor para que esta pueda ser accedida desde cualquier otro lugar, como por ejemplo un bóveda que se encuentre en alguna sucursal de esta institución bancaria.

Hipótesis

Una institución bancaria requiere tener mayor control en el manejo de los billetes para esto necesitará cambiar la forma de almacenarlos y manejarlos. Los billetes estarán ahora contenidos en unidades de empaque las cuales estarán identificadas por códigos que

llevarán en etiquetas pegadas y estarán representados por códigos de barras. Los códigos de barras serán leídos por el dispositivo portátil SPT1700 el cual contendrá una aplicación que manejará y almacenará dichos códigos. Con esta aplicación se podrán realizar operaciones con las unidades de empaque que contienen a los billetes.

Una vez realizadas las operaciones con las unidades de empaque, y los códigos de dichas unidades estén almacenados en el SPT1700, entonces, se realizará la sincronización con un computadora de escritorio, en donde se transferirán los datos que contenga el dispositivo a la computadora de escritorio. De esta manera, las operaciones registradas por todos los dispositivos quedarán registradas también en una computadora de escritorio que, a su vez, podrá transferir los datos a un servidor.

Se pretende que con este sistema se tenga un control sobre los billetes y las unidades de empaque que los contengan, que se manejan y almacenan en una institución bancaria, de tal manera que se pueda saber cuando se arma un contenedor que será transportado a una bóveda de una sucursal. Se tendrá el control de las unidades que forman a ese contenedor, de la denominación que tienen los billetes que lo forman, de que bóveda parte y a que bóveda llega, y se podrá verificar que llega a una bóveda con la misma cantidad con que partió de otra.

Con la aplicación desarrollada especialmente para éste proyecto se podrán controlar todas las operaciones que se realizan con las unidades de empaque como son armado, apertura, preparación de desarmado, traspasos y cambio de estado físico.

Descripción del capitulado

El **capitulo 1** consiste en una descripción general del proyecto de tesis, donde, primero se habla de la situación actual en una institución bancaria, para después proponer un sistema que sea capaz de llevar el control de las unidades de empaque donde se almacenarán los billetes. Además se da una explicación de lo que son los códigos de barras así como los

tipos y simbologías mas utilizadas en la actualidad. Se hace una comparación con otras técnicas de captura de información, lo cual da una idea de el porque se eligió la técnica del código de barras para dicho proyecto. Se habla de los dispositivos que existen para la lectura de los códigos de barras así como las ventajas y desventajas de éstos. También se habla de las técnicas de impresión y de los tipos de impresoras utilizados para la impresión de los códigos de barras, así como las ventajas y desventajas de cada uno de ellas.

En el **capitulo 2** se habla de bases de datos relacionales, donde en principio se explica la aparición del término Bases de datos, se habla de las características que debe cumplir una base de datos para que sea llamada como tal. También se citan las diferentes visiones de los datos que varían de acuerdo al punto de vista con que sean vistos. En este capitulo se habla también de las diferencias que existen entre una base de datos y un sistema gestor de base de datos ya que muchas veces estos términos son confundidos.

En este capitulo se habla también del modelo de datos relacional el cual esta basado en la teoría de normalización de relaciones que tiene por objeto la eliminación de los comportamientos anómalos de las relaciones y la eliminación de redundancias superfluas. Se cita también la definición de Relación, así como de los dominios y atributos, y de la intención y extensión de una relación

También se habla sobre la normalización de las relaciones y se ve que está basada en los principios de la teoría general de conjuntos. Se analizan las dependencias funcionales para después poder citar las formas normales. Y por último se explica el lenguaje relacional y se citan algunos ejemplos.

En el **capítulo 3** se habla de las características de un dispositivo que cuenta con un sistema operativo PalmOS como el SPT1700. También se habla del tipo de procesador con que cuenta un dispositivo Palm, de la pantalla, que es una parte muy importante en el

dispositivo ya que quien desee programar en una Palm tiene que considerar el tamaño de la pantalla que es muy reducida.

Así mismo, en éste capitulo se explican los modos de ejecución de un dispositivo Palm y como es que se maneja la energía. Se explica que las aplicaciones de una Palm necesitan aparte de el núcleo del sistema operativo, el kernel, de los manejadores para poder interactuar con todo, como los puertos, la memoria, el usuario, etc. Se explica como es que se maneja la memoria y las formas en que se puede reiniciar el dispositivo.

También se habla sobre la capacidad que tiene un dispositivo Palm para sincronizar su información con la de una computadora de escritorio ya que esta característica es la llave del éxito de estos dispositivos. Y por último se explican las características del SPT1700 que es un dispositivo que cuenta con un sistema operativo Palm y es el que se utiliza en este proyecto.

El **capitulo 4** trata sobre la base de datos UltraLite, cuyo objetivo son los dispositivos móviles como son los teléfonos celulares, organizadores portátiles, etc. y es una base de datos relacional ajustada, es decir, que es construida a la medida exacta de la aplicación. Se explica también en éste capitulo la arquitectura que tiene UltraLite y sus principales limitaciones. También se explica una de las características mas importantes de UltraLite, que es su capacidad para sincronizar sus datos con una base de datos que se encuentre en una computadora de escritorio; la sincronización con Mobilink.

En este capitulo también se explica como se utiliza el lenguaje SQL en bases de datos UltraLite, como son las fases de desarrollo de una aplicación que utiliza un base de datos UltraLite y se explica detalladamente el proceso de sincronización.

También se explican las condiciones que debe cumplir la base de datos de la computadora de escritorio con que se va a sincronizar la base de datos del dispositivo portátil, la base de datos UltraLite.

En el **capitulo 5** se explican las características principales de la herramienta que se utilizó para desarrollar este proyecto, CodeWarrior. Se explican las diferentes herramientas que existen para el desarrollo de software para dispositivos como la Palm, y las características mas importantes de cada una. Se mencionan las partes que forman a la herramienta CodeWarrior y se describen los términos mas utilizados por ésta.

Se describen los elementos de la interfaz de usuario y también la estructura de una aplicación en CodeWarrior, explicando como funciona cada una de las partes de ésta.

CAPITULO 1

GENERALIDADES

1.1 ANTECEDENTES

El hombre siempre ha buscado alternativas que le ayuden y faciliten la elaboración de su trabajo, por lo que ha incursionado en el campo de la computación, creando nuevos horizontes o mejorando lo que ya había realizado.

Existen empresas y almacenes que manejan grandes cantidades de información o de productos, lo que hace que el control de éstos sea muy difícil y lento; es por eso que han buscado un apoyo en la computación para obtener métodos que les permitan agilizar y controlar el flujo de la información. Para tener un control eficiente y confiable es necesario tener un sistema de identificación por medio de la codificación de información, la cual facilita el manejo de grandes volúmenes de información haciendo este trabajo más sencillo y seguro.

Las instituciones bancarias por ejemplo, manejan grandes cantidades de dinero y entre sus diferentes funciones se encuentra almacenar éste dinero, transportarlo y realizar diferentes operaciones con el. La mayoría de las instituciones bancarias almacenan el dinero en bolsas y de esta forma lo transportan y lo guardan en bóvedas. Estas bolsas contienen una cierta cantidad de dinero, que está formada por billetes de varias denominaciones, lo cual dificulta su control.

Esta forma de manejar los billetes es muy poco práctica, ya que, por ejemplo, cuando se abre una bolsa de dinero es necesario agruparlo por denominaciones para almacenarlo y para llevar el control de cuantas piezas hay de cada denominación

En la actualidad, gracias al gran desarrollo de tecnología, todas las cosas cambian para mejorar o para adaptarse a la nueva tecnología. Es por eso que en una institución bancaria existe la necesidad de almacenar los billetes ya no en bolsas sino en diferentes contenedores las cuales permitan que se pueda llevar un mejor control del dinero. Es

decir, un contenedor que esté formado por otros contenedores y cada uno contenga todos los billetes de una denominación en especial.

1.2 PROPUESTA

Actualmente podemos observar que en México se muestra un gran avance dentro del campo de la computación, tanto en equipo como en sistemas, lo que ha permitido una automatización de trabajos en empresas, las cuales se han visto beneficiadas por la seguridad, rapidez y confiabilidad del procesamiento de su información y por lo tanto de los resultados que reciben de estos mismos.

Hay una gran diversidad de entornos donde pueden encontrarse este tipo de aplicaciones, tal es el caso de supermercados y empresas que manejan inventarios, que gracias a estos sistemas tienen mayor control sobre los mismos. Los resultados obtenidos han provocado que se utilice este tipo de sistemas en otros campos, como por ejemplo, la identificación de personal dentro de una empresa en la cual colaboran una gran cantidad de empleados. Estos sistemas mejoran las medidas de seguridad en las empresas para el acceso a determinadas zonas restringidas y proporciona información para el cálculo de nóminas.

Como es sabido, se tiene un problema mientras existan varias propuestas de posibles soluciones para poder resolverlo. Si el problema tiene una solución, entonces ya no tenemos un problema sino más bien un posible proyecto trabajando para una aplicación en particular. Todo diseño tiene una función que cumplir o una necesidad que satisfacer y debe justificar por sí solo su existencia. Además, debe tener la capacidad de ser competitivo en varios aspectos, por ejemplo, costo, tecnología involucrada, mantenimiento del sistema, vida útil del equipo, tiempo de armado del sistema, posibilidad de mejorar o adaptar el sistema a futuro para mantenerse actualizada, etc., de acuerdo a los intereses y metas que se tengan en particular.

Después de analizar la situación actual se propone un sistema que permita llevar el control de las unidades de empaque[1] donde se van a almacenar los billetes, además permitirá saber cuales unidades forman a otra unidad de empaque, cuantas piezas existen de una determinada denominación, en que bóveda se encuentra un contenedor, hacia donde se va transportar un contenedor y de donde llega, solo con un código que va a identificar a cada unidad de empaque y dicho código estará en una etiqueta que tendrá pegada cada una de estas unidades. Los contenedores de dinero se denominarán unidades de empaque. Los códigos manejados para identificar a cada unidad de empaque será un código de barras.

Esto a parte de proporcionar un mejor orden, también da un mayor control de la cantidad de dinero que se transporta o se almacena en determinado lugar. Permite saber de una forma muy sencilla la cantidad de billetes que hay de cierta denominación y la suma de esa cantidad de billetes.

Además es necesario llevar un control para saber cuales unidades de empaque se encuentran en un determinado lugar. Saber que cantidad de billetes de cierta denominación se encuentran en una bóveda específica.

Las unidades de empaque están compuestas por otras unidades de empaque mas pequeñas y éstas a su vez también se encuentran formadas por unidades de empaque mas pequeñas aun. Por lo tanto, es necesario que el control que se llevará sobre estas unidades también indique cuales unidades están formando a otra unidad y cuales de estas forman otra mas.

Una de las características principales es que cada unidad de almacenamiento va identificada con una etiqueta de código de barras la cual representa un identificador, el

[1] Una unidad de empaque es un recipiente donde se almacenarán los billetes y podrá ser de diferentes tipos. También es llamada unidad de almacenamiento o contenedor, aunque para este proyecto de tesis un contenedor es un tipo de unidad de empaque.

cual contiene información como en que bóveda se encuentra y a que institución pertenece dicha bóveda, de que denominación son los billetes que componen esa unidad de empaque y de que tipo es la unidad.

Es importante mencionar que cada unidad de almacenamiento solo contiene billetes de una sola denominación y si se quisiera juntar una suma de dinero la cual requiere billetes de varias denominaciones, entonces, se puede formar una unidad de empaque con billetes de una sola denominación y otra unidad con billetes de otra denominación diferente.

Las unidades de empaque son de diferentes tipos de acuerdo a su tamaño para su fácil armado y agrupación. De esta manera tenemos que los tipos son: mazos, paquetes, bolsas, contenedores y plataformas.

Con esto se garantiza una transportación mas fácil y eficiente del dinero, ya que los contenedores están hechos de tal forma que se pueden levantar de forma sencilla utilizando un montacargas para colocarlos en un camión de transporte de valores por ejemplo.

Esta nueva forma de manejar los billetes lleva consigo la elaboración de un sistema informático, el cual permitirá el control mediante los códigos que identificarán a cada unidad de empaque.

Con un sistema como el propuesto en éste proyecto de tesis se puede tener el control de cuales unidades son las que forman a otra unidad de empaque, también se podrá saber que contenedores están en cierta bóveda[2], también se pude llevar el control al momento en que una unidad de empaque es transportada de una bóveda a otra y así saber

[2] Se entenderá como bóveda al lugar que tienen destinado las instituciones que manejan dinero para almacenarlo y resguardarlo mediante grandes mecanismos de seguridad. Regularmente las bóvedas son lugares que están blindados y que tienen el acceso restringido.

en que momento y con que cantidad de dinero parte de la bóveda origen y en que momento llega a la bóveda destino y ver si es la misma cantidad de dinero con la que partió.

Para la realización de un sistema informático hay que tomar en cuenta que la mayoría de las operaciones que se realizan con los billetes se llevan a cabo en las bóvedas o en los lugares donde se abastecería a un camión de transporte de valores para que se puedan transportar dichos billetes.

Es claro que para implantar un sistema que pueda operar en bóvedas es necesario utilizar algún dispositivo que sea portátil y que no necesite de la conexión a la corriente eléctrica o a algún computador para poder operar. Además, será necesario que sea capaz de leer etiquetas con códigos de barras, y debe ser capaz también de ser programado o de que se le pueda implantar algún sistema para poder llevar el control que es requerido. Y tomando en cuenta que una PC[3] presentaría algunas restricciones para que se pueda instalar dentro de una bóveda ya que es necesario meter algún tipo de cableado dentro de la bóveda, por lo que se tendrían que hacer perforaciones para poder pasar los cables. También existe la desventaja de que una PC no puede trasladarse fácilmente por el hecho de que tiene que estar conectada. Por ejemplo, si se lleva a cabo un operación de armado de una unidad de empaque por otras unidades de empaque mas pequeñas, se estaría restringiendo la movilidad de las personas que realicen dicha operación, obligándolas a mover dichas unidades de empaque primero cerca de la PC para que pudiera ser leído el numero que identifica a dicha unidad y después moverlas hacia el camión que las va a transportar.

También hay que tomar en cuenta que las personas encargadas del manejo de los billetes (para subirlos a un camión o para almacenarlos) en una bóveda, no son personas

[3] PC es la abreviación de Personal Computer que significa Computadora Personal.

con el conocimiento necesario para el manejo de Computadoras Personales o inclusive de un sistema que pueda manejar bases de datos.

Va ser necesario utilizar una base de datos para que se almacenen los números identificadores de las unidades de empaque. Esta base de datos va a ser alimentada por los dispositivos que realicen las lecturas de los códigos de barras. La base de datos va a ser implantada en una PC para que se pueda administrar y manejar dicha base desde un escritorio por una persona calificada en el funcionamiento de PC's.

El dispositivo que ha cumplido con las características que se requieren es el SPT1700 que es un dispositivo que cuenta con un sistema operativo PalmOS (Palm Operating System). Este dispositivo, a diferencia de cualquier otro que tenga las mismas características y que cuente con un sistema operativo en el que se le pueda instalar o desarrollar un algún sistema, tiene un diseño industrial que permite un uso rudo y que puede aguantar caídas (de no mucha altura).

El SPT1700 tiene la característica de que lee códigos de barras de diferentes tipos, y únicamente hay que configurarlo para que se puedan leer los que se requieren para este proyecto.

Por medio de los SPT1700 se leerán los códigos de barras que identifican a cada unidad de empaque. Estos códigos quedarán almacenados en una base de datos que estará implantada en el SPT1700. El manejo del lector de códigos de barras y el funcionamiento de todas las operaciones que se llevarán a cabo por medio de éste dispositivo se realizará por medio de programas desarrollados en un lenguaje de programación. La herramienta que se utilizará para el desarrollo de dichos programas será CodeWarrior el cual permite el manejo de una gran cantidad de funciones las cuales son parte de un lenguaje de programación muy potente como es C / C++.

El SPT1700 cuenta con una base[4] que es fija y que se conecta a la computadora de escritorio, como cualquier otro dispositivo Palm. Por medio de esta base se transfiere información del SPT1700 hacia la computadora de escritorio y viceversa. Es decir, el SPT1700 se colocará en su base y después se le dará la instrucción a un programa[5] para que se inicie la transferencia de información en ambos sentidos (de la PC al SPT1700 y del SPT1700 hacia la PC), a esto se le llama sincronización.

Los códigos de barras que sean leídos con el SPT1700 quedarán almacenados en una base de datos que contendrá el dispositivo. Dicha base de datos es una base de datos UltraLite, la cual se sincronizará con una computadora de escritorio en donde existe también una base de datos. Los registros nuevos que se crearon en el SPT 1700 se agregarán a la base de datos de la computadora de escritorio, la cual se llama base de datos consolidada. Los registros que existan ya en la base de datos consolidada en el momento de la sincronización simplemente se actualizarán con la información que provenga del SPT1700.

Para evitar algún tipo de conflicto, cada registro contendrá un campo de fecha y hora, el cual almacenará el tiempo en que dicho registro fue modificado por última vez y se evaluará cada que se tenga que hacer alguna actualización en dicho registro.

Las operaciones que se podrán llevar a cabo en el SPT1700 son la de armado, que consiste en leer los códigos de barras de todas las unidades de empaque que van a formar a otra unidad de empaque y la lectura del código de barras de ésta última; así se podrá conocer el código de barras del la unidad de empaque contenedora y de las unidades de empaque que la forman.

También está la operación de preparación de desarmado la cual consiste en la lectura del código de la unidad de empaque que se quiere desarmar, el sistema verificará

[4] El nombre de dicha base es: cradle o cuna.
[5] El programa encargado de realizar la sincronización es HotSync.

si contiene las unidades de empaque que conforman a la que se quiere desarmar, sino, en la siguiente sincronización el sistema obtendrá dicha información.

También se encuentra la operación de Apertura que va ligada a la operación de preparación de desarmado. La operación de Apertura consiste en leer el código de barras del contenedor que se quiere desarmar y el sistema marcará a todas las unidades de empaque que lo forman como libres y como eliminado el contenedor. Para esto el sistema necesitará que las unidades de empaque que forman al contenedor se encuentren en el dispositivo para esto será necesario que se realice primero una operación de preparación de desarmado.

Otra operación que se podrá realizar con el sistema propuesto es la de Cambio de Estado Físico, que consiste en leer el código de un contenedor y especificar el estado físico[6] al que se quiere cambiar, el sistema se encargará de validar si en la bóveda en que se encuentra dicho contenedor se puede tener un estado físico como el que se quiere establecer. Si el sistema encuentra que si es permitido, entonces, marcará al contenedor con dicho estado físico y a las unidades que la forman. Si fuera necesario el sistema hará la consulta a la base de datos para obtener las unidades de empaque que forman al contenedor. Cabe mencionar que esta operación solo se realizará con contenedores.

Una de las operaciones mas importante que se podrán realizar con el sistema propuesto es la operación de Traspaso. Esta operación se llevará a cabo en dos partes, la primera será la salida del contenedor y se realizará leyendo el código de barras del contenedor que se quiere traspasar y la bóveda destino. El sistema validará si se puede realizar el traspaso a dicha bóveda, y si fuera permitido entonces marcará a dicho contenedor como en transito. La segunda parte que es la de llegada consiste en leer nuevamente el código del contenedor pero ya en la bóveda destino, verificar de que bóveda viene y marcarlo con la nueva bóveda.

[6] El estado físico de los billetes también es controlado por el sistema, ya que no se puede dar el mismo trato a los billetes que son nuevos que a los deteriorados.

1.3 EL CODIGO DE BARRAS

El presente trabajo tratará sobre la codificación como medio de identificación de las unidades de empaque, esto se hace con el fin de proporcionar un buen servicio a los usuarios, así como controlar y cuidar las unidades de empaque de los billetes; además, es una oportunidad para introducir la aplicación de nuevas tecnologías en una institución bancaria.

El ser vecinos de una potencia en el área de la informática nos da la ventaja de tener acceso a los innovaciones tecnológicas a un costo razonable. Con la apertura comercial de México, es muy importante que conozcamos y adaptemos estas tecnologías a nuestras necesidades para eficientar las operaciones y ser competitivos en los mercados nacionales e internacionales. Debido a la rapidez con que se dan estos cambios nuestra actualización debe ser continua; dentro de las herramientas disponibles debemos seleccionar la que mejor se apegue a nuestras necesidades, esto provoca que se tenga que realizar una investigación cada vez más detallada de los productos que vayan a utilizarse para la automatización deseada.

La utilización del código de barras se seleccionó entre varias opciones disponibles, como se verá más adelante en el desarrollo de esta tesis.

El procesamiento de la información es esencial para la administración de los gobiernos, los negocios, la educación, y aún para las actividades de entretenimiento. El pronostico del tiempo, por ejemplo, el cual puede determinar los planes para el fin de semana o las vacaciones, se basa en el procesado y comunicación de información precisa. En nuestra sociedad es vital para una organización o empresa proporcionar información correcta y puntual para apoyar la toma de decisiones y otras actividades gerenciales. Como resultado del crecimiento económico y avances tecnológicos, muchas organizaciones han crecido tanto en el tamaño como en la sofisticación de sus funciones

administrativas. Mientras el volumen de procesamiento de datos crece a una rapidez sin precedentes, también crece la demanda de medios eficientes para manejarlos.

Tenemos que existen diversos métodos de captura de datos, se describirán brevemente en la Tabla Comparativa de técnicas de captura de datos.

La información escrita mediante el código de barras puede ser utilizada en una gran variedad de aplicaciones, pero principalmente se utiliza en tres sectores:

- Automatización comercial.
- Control de inventarios.
- Sistemas de control de acceso, asistencia y productividad.

Existen varios patrones internacionales referentes al código de barras, por lo tanto cuando se piensa en implantar un sistema donde los datos colectados se basen en este tipo de código, es necesario tomar en consideración algunos factores tales como:

- El tipo de dato que se va a manejar, ya sea numérico o alfanumérico, así como la cantidad de caracteres que éste contenga.

- El medio o material en que serán impresos los datos codificados, el cual deberá tener una resistencia, durabilidad, propiedades mecánicas y ópticas consistentes con el equipo de lectura que se piense utilizar, en este caso el SPT1700.

- Los métodos de impresión disponibles para el dato codificado. La técnica de impresión utilizada debe ser capaz de generar códigos dentro de las tolerancias de anchura de las barras y de las propiedades ópticas del sistema.

TABLA COMPARATIVA DE TÉCNICAS DE CAPTURA DE DATOS

	Tiempo p/reg. 20 caracteres en un campo *	Error de sustitución	Tamaño ** de la Etiqueta (20 caracteres)	Costo de la Etiqueta	Costo del equipo de lectura	Ventajas	Desventajas
TECLADO	10 SEG.	Alto	0.4' x 2.2.'	Bajo	Bajo	Bajo costo en el equipo	Requiere operador, poca flexibilidad, baja rapidez
OCR	4 seg.	Medio	0.5' x 2.5'	Bajo	Medio	Puede ser leído por el humano	No existe flexibilidad en el equipo de lectura
MICR	Normalmente escaneado por una máquina	Medio	0.5' x 2.5'	Medio	Alto	Puede ser leído por el humano	Es caro, no existe flexibilidad en el equipo de lectura
CINTA MAGNETICA	4 seg.	Bajo	0.4' x 1.0'	Medio	Medio	Grandes cantidades de datos pueden ser decodificados y cambiados	Puede ser efectuado por campos magnéticos, requiere contacto con el equipo de lectura
RECONOCIMIENTO DE VOZ	20 seg.	Alto	0.4' x 2.2.'	Bajo	Alto	No requiere de operación manual	Requiere operador, equipo costoso y no es general para otras aplicaciones
DISPOSITIVO DE VIDEO	Normalmente escaneado por máquina	Depende de la técnica de mercado	Variable	Variable	Muy alto	Puede formar parte del sistema de inspección	Equipo muy caro y tiene una aplicación específica
RADIO FRECUENCIA	2 seg.	Bajo	1.0' x 1.5'	Alto	Alto	Las etiquetas no necesitan ser variables	Las etiquetas son caras
CODIGO DE BARRAS	4 seg.	Bajo	0.6' x 2.5' ***	Bajo	Bajo	Flexibilidad de impresión y equipo de lectura	

* Se asume que se utiliza un operador
** Se asume que el tamaño mas pequeño de la etiqueta es apropiado para las aplicaciones generales.
*** Se asume que se utiliza una sola columna en el código de barras

TABLA 1.1 ESTA TABLA COMPARATIVA DE TÉCNICAS DE CAPTURA NOS MUESTRA EL PORQUE DE ESCOGER CODIGO DE BARRAS.

El código de barras es la forma de representar información que contiene números u otros caracteres haciendo uso de una secuencia de barras paralelas, claras y obscuras, anchas y estrechas, las cuales son leídas por medio de equipos de lectura óptica como el SPT1700. Siendo un número de identificación único para cada uno de las unidades de almacenamiento.

Es también la tecnología de identificación automática, aplicable a las personas y objetos. Teniendo como objeto la identificación y localización repetitiva de productos a nivel industrial y comercial. Logrando agilizar los procesos, evitando errores, aumentando su confiabilidad y eficiencia.

En general este sistema de identificación trata de un elemento codificado portador de la información y un elemento lector capaz de reconocer la información; ésta alimenta a un computador donde la identificación es decodificada, verificada, comparada y aceptada para luego tomar una decisión lógica. En éste caso, el elemento lector es el SPT1700 el cual contendrá una base de datos donde se almacenarán un conjunto de códigos, para después alimentar al computador[7].

Los casos de identificación de personas son: por ejemplo, acceso a una cuenta de banco, área restringida, computador, línea telefónica, empresa, casa, a los controles remotos, etc.; en cuanto a la identificación de los objetos, sobre todo los destinados a la actividad comercial donde es necesario una exacta identificación del producto que le permita conocer al industrial, comerciante, distribuidor y cliente los siguientes elementos: características del producto, origen, ubicación, destino, costo, precio de venta, verificación, control, contabilidad, administración, estadísticas e inventarios.

1.3.1 Ventajas

A continuación se explican algunas de las razones para preferir el código de barras ante cualquier otro medio de marcación y colecta de datos:

- La exactitud, precisión y confiabilidad para la recolección automática y sistematizada de información impresa, como a su capacidad de establecer lazos de intercambio y comunicación de la información únicos entre el industrial y distribuidor de productos en gran escala, para consumo masivo.

- La alta confiabilidad de los datos leídos y enviados a la computadora, gracias a que la lectura se hace por medios electrónicos y no por medios manuales, los cuales tienen un alto porcentaje de error.

[7] La forma de alimentar a la computadora de escritorio será por medio del proceso de sincronización, la cual, no se realizará por medio del HotSync, si no por medio del servidor Mobilink.

- La lectura de la información codificada, es rápida y automática que se hace por medio de lectores ópticos que envían la información a la computadora o bien la almacenan en algún dispositivo para después procesarla.

- Optimización del flujo en la producción de volumen en el manejo de documentos, facilidad de entrenamiento, mejor control y contabilidad, reducción de tiempos en las líneas de producción, reducción en los niveles de investigación de almacén y mejor visión de las existencias.

- Marcación única del producto desde la fuente primaria de producción hasta el consumidor, a quien permite saber exactamente que es lo que compra y a que precio, evitando también posibles adulteraciones.

- Información precisa de los tiempos y ciclos de producción, inspección, almacenamiento y transporte.

- Mínimo de errores en la información, ya que incluyen sistemas de autoverificación y caracteres de control dentro de si mismos, que eliminan los errores de lectura.

- Sobredimensionado vertical, que permitirá leer un código aun cuando solamente un 5% de su altura permanezca legible, ante la eventual destrucción del código impreso.

- Velocidad y eficiencia en el recepción, venta y cobranza, específicamente en las cajas de los supermercados.

- Se elimina la necesidad del remarcado de precios y/o la codificación manual e individual producto por producto, especialmente en los países de alta inflación.

- Información en tiempo real y estadísticas en general de inventario, venta y reposición de stock.

- Eliminación de errores humanos de marcación, interpretación, facturación al cliente y pérdida desconocida.

- Adaptable a la mayoría de los sistemas de embalaje, impresión y materiales de embasamiento existentes.

1.4 TIPOS Y SIMBOLOGIA

1.4.1 Código UPC

El Código Universal de Producto (UPC), fue creado y adoptado por la industria norteamericana en 1973 para su lectura en cajas registradoras de los supermercados, el cual fue desarrollado para su uso industrial.

Actualmente el código UPC es utilizado tanto en los grandes almacenes como en las pequeñas tiendas de venta al menudeo. Satisfaciendo los requerimientos particulares en las cajas registradoras de los supermercados o de las tiendas de autoservicio, llamado hoy en día punto de venta.

El código UPC es numérico, de longitud fija, simbología continua, con autochequeo y emplea cuatro bloques de elementos de diferentes anchuras. Las barras representan el 1 binario y los espacios el 0 binario.

Los dos tipos más comunes del código UPC son: el código UPC A que está formado por 12 dígitos y el código UPC E que está formado por 6 dígitos.

1.4.2 Código UPC A

El código UPC A cuenta con 12 caracteres de tipo numérico únicamente, excluyendo los caracteres de inicio / final y los identificadores centrales.

Está construido de tal forma que el primer carácter es para definir la categoría del producto (medicinal, alimenticio, etc), los siguientes cinco caracteres son para la identificación del fabricante del producto, los otros cinco caracteres identifican al producto y el último carácter es el dígito de verificación del código. Los caracteres 1 y 12 que vendrían siendo las barras de seguridad, generalmente se imprimen con una longitud mayor que el resto de las barras del símbolo. Esto se hace para maximizar autorización rastreando en un ángulo inclinado.

Cada carácter numérico se representa por 2 barras y 2 espacios, ubicados alternativamente, o sea cuatro elementos para cada carácter; el ancho y la ubicación de los elementos diferencia a un carácter de otro.

Por lo tanto, los cuatro elementos que forman a un carácter también tendrán un ancho de 7 módulos, es así que cada barra y/o espacio podrán tener un ancho como mínimo de un módulo, y como máximo de 4 módulos, siendo así un código de estructura compleja.

Estos criterios sólo se aplican a los 12 caracteres numéricos que se codifican en el sistema UPC A, y no se aplican a los separadores y zonas mudas.

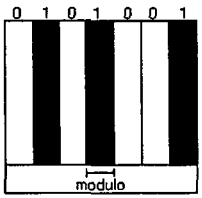


FIGURA 1.1
EN ESTA FIGURA SE MUESTRA COMO EL ANCHO DE CADA CARÁCTER ES FIJO Y MIDE 7 MODULOS.

Los primeros 6 dígitos están separados de los 6 segundos dígitos por unas barras de seguridad centrales. Las dos partes del símbolo están protegidas por dos barras izquierdas y dos barras derechas ambas de seguridad. Estas barras de seguridad pueden ser el patrón de inicio / final.

Los unos binarios que son contabilizados en la mitad izquierda, siempre suman impares y la suma de los unos de la mitad derecha siempre suman pares, de tal manera que por medio de un estudio del dato de paridad el rastreador (scanner) pueda decodificar en forma independiente la parte derecha o izquierda del símbolo identificando inmediatamente que mitad fue decodificada.

Simbología del código UPC A:

CARACTER	MITAD IZQUIERDA	MITAD DERECHA
0	0001101	1110010
1	0011001	1100110
2	0010011	1101100
3	0111101	1000010
4	0100011	1011100
5	0110001	1001110
6	0101111	1010000
7	0111011	1000100
8	0110111	1001000
9	0001011	1110100

identificador inicial: 101
identificador central: 01010

TABLA 1.2
TABLA QUE MUESTRA LA SIMBOLOGIA EN CARACTERES BINARIOS DEL CODIGO UPC A

1.4.3 Código UPC E

Esta versión también es llamada "Cero Suprimido", ya que elimina por lo menos cuatro ceros en el código. No siempre es posible su uso ya que esto dependerá del número del fabricante y el número del producto asignados.

Existen cuatro formas de supresión de ceros, dependiendo de los tipos de números que le fueran asignados al fabricante y al producto, obedeciendo a normas muy estrictas de aplicación, que determinan en cada caso, cuantos son los artículos que podrían disponer de un código reducido UPC E, por ejemplo:

- Si el número de fabricantes termina en 00, precedido por 0,1 ó 2; 1000 productos podrán ser codificados con UPC E.

- Si el número de fabricante termina en 00, precedido por 3 al 9; 100 productos podrán ser codificados.

- Si el número del fabricante termina en 0, 10 número de producto podrán ser asignados.

- Si el número del fabricante no termina en 0, sólo 5 productos podrán utilizar la versión reducida.

Aún así en todas estas condiciones, el número asignado al producto también debe comenzar con algunos ceros para que la reducción del código sea factible. Son solo 7 caracteres aunque se leerán 12, numéricos únicamente.

Los caracteres de identificación del fabricante y del producto se codifican por un método especial que permite eliminar los dígitos cuyo valor es igual a cero, la supresión de los mismos depende de su ubicación en la versión estándar UPC A.

Al igual que el código UPC A cada carácter consiste de 2 barras y 2 espacios de anchos variables. Cuenta con dos separadores laterales, eliminando el separador central, el separador derecho es diferente y le indica al rastreador (scanner) que debe decodificar un código UPC E.

El código UPC E es utilizado cuando es necesario reducir el tamaño del código por motivos de espacio y cuando la estructura original lo permite.

1.4.4 Código EAN

El código EAN (Asociación Internacional de Numeración de Artículos), es sustituto del UPC. El examinador de EAN puede decodificar UPC, pero a la inversa no es posible.

Es un código de codificación continua, longitud fija y estructura compleja. El código EAN tiene dos versiones EAN 13 y EAN 8 codificando de trece a ocho dígitos respectivamente. UPC / EAN han progresado en el mercado, utilizándose sobre todo en los supermercados al unir el rastreador (scanner) y la computadora, demostrando ser un buen conjunto para mejorar la productividad.

Este es un sistema de codificación común a varios países y productos dentro y fuera del mercado común Europeo, existe un indicativo nacional llamado "Bandera" para cada país que identifica al organismo nacional de codificación que a su vez asigna los códigos en su localidad.

1.4.5 Código 39

El código 39 fue la primer simbología alfanumérica que se desarrolló, siendo fácilmente adaptado a un gran número de aplicaciones y adoptado por un sinnúmero de industrias como el Departamento de Defensa de los Estados Unidos, así como en otras agencias del gobierno. Este código esta estandarizado por la (AIM), Automatic Identification Manufacturers, American National Standard Institute (ANSI) y Auto Industry Action Group (AIAG). Es una simbología discreta, con autochequeo y longitud variable que puede ser impresa confiablemente por una gran variedad de tecnologías.

Cada carácter del código 39 tiene cinco barras y cuatro espacios. De esos nueve elementos, tres son anchos y seis son angostos en cada carácter.

El código 39 cuenta con las siguientes características:

Conjunto de caracteres:

- 26 letras mayúsculas
- 10 dígitos
- 7 caracteres especiales
- Expandible a 128 caracteres del código ASCII con las dos marcas de inicio / final.

Longitud del símbolo: Variable
Carácter de chequeo: Opcional
Carácter de cabecera: 2 por símbolo
Tipo: Discreto
Otras características: Habilidad de concatenación
Densidad: Máxima de 9.8 c/p, cuando la impresión utiliza 7.5 mm. por dimensión

Cada símbolo del código 39 consiste de:

- Zona muda de inicio
- Carácter de comienzo
- Caracteres de datos
- Carácter de término
- Zona muda de término

Simbología del código 39

CARACTER	Código Binario	CARACTER	Código Binario
1	100100001	M	101000010
2	001100001	N	000010011
3	101100000	O	000010011
4	000110001	P	001010010
5	100110000	Q	000000111
6	001110000	R	100000110
7	000100101	S	001000110
8	100100100	T	000010110
9	001100100	U	110000001
0	000110100	V	011000001
A	100001001	W	111000000
B	001001001	X	010010001
C	101001000	6	110010000
D	000011001	Z	011010000
E	100011000	-	010000101
F	001011000	.	110000100
G	000001101	ESPACIO	011000100
H	100001100	*	010010100
I	001001100	$	010101000
J	000011100	/	010100010
K	100000011	+	010001010
L	001000011	%	000101010

El carácter tipo bold representa la barra (binario 1)
El carácter tipo normal representa el espacio (binario 0)

TABLA 1.3
ESTA TABLA ILUSTRA LOS PATRONES DE CODIFICACIÓN DEL CODIGO 39 CADA UNO DE SUS 44 ELEMENTOS TIENEN ARREGLO UNICO DE TRES ELEMENTOS ANCHOS Y SEIS DELGADOS.

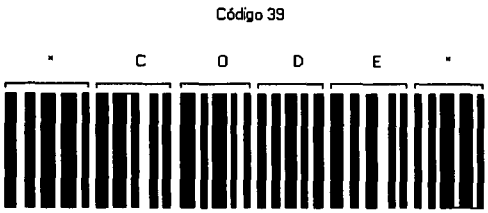


FIGURA 1.2
ESTE ES UN EJEMPLO DEL CODIGO 39 CODIFICANDO EL DATO "CODE", CADA CARÁCTER ESTA SEPARADO DE SU VECINO POR UNA BRECHA DE TOLERANCIA INSIGNIFICANTE QUE NO CONTIENE INFORMACIÓN.

Codificación del código 39:

Cada carácter del código 39 consiste de 5 barras y 4 espacios, 3 de los elementos son anchos y 6 delgados. El caracter asterisco es utilizado exclusivamente como marca de inicio / final.

Caracter de chequeo:

Es un caracter opcional en el código 39, para aplicaciones que requieren altos niveles de seguridad. Cuando es utilizado, el carácter de chequeo es posicionado entre el carácter final del dato y el carácter de termino.

1.4.6 Intercalado 2 de 5

Se utiliza principalmente en la identificación de productos y contenedores por la Industria para almacenamiento y distribución.

Es una simbología de código de barras con un conjunto de caracteres numéricos, caracteres inicial / final. El nombre entrelazado 2 de 5 deriva del método utilizado para formar pares de caracteres. En el símbolo, dos caracteres son apareados juntos utilizando

barras para representar el primer carácter y espacios anchos para representar al segundo. Cada carácter (del 0 al 9) consta de dos elementos anchos y tres angostos, o sea un total de cinco barras o espacios. Las barras representan los caracteres en la posición impar de la cadena de números y los espacios representan los caracteres en la posición par de tal forma que están intercalados.

El intercalado requiere que el símbolo codificado contenga números pares, si un número impar de caracteres ha sido codificado, un cero es agregado.

Entre sus características principales se encuentra que es un código de caracteres codificables únicamente numéricos, tiene una forma de decodificación bidireccional con autochequeo de caracteres, la zona muda superior es de 1.1 mm. y además los caracteres inicial / final son únicos.

La señal de inicio se compone de una barra angosta, un espacio angosto, una barra angosta y un espacio angosto, que en el sistema binario representarían (0,0), mientras que la señal de final se compone de una barra ancha, un espacio angosto y una barra angosta, que en binario se representaría como (1,0).

Los elementos delgados sean barras o espacios representan el cero binario y los elementos anchos sean barras o espacios representan el uno binario. Es continuo porque tanto las barras como los espacios forman parte del código.

Simbología del intercalado 2 de 8:

Caracter	Código
0	00110
1	10001
2	01001
3	11000
4	00101
5	10100
6	01100
7	00011
8	10010
9	01010

Inicio 0000
Final 100

TABLA 1.4
OBSERVANDO ESTA TABLA VEREMOS QUE EL NOMBRE INTERCALADO SE DERIVA DE FORMAR PARES DE CARACTERES.

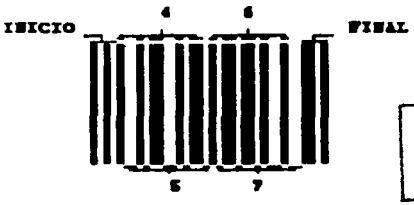


FIGURA 1.3
ESTE ES UN EJEMPLO DE EL CODIGO INTERCALADO 2 DE 5

1.4.7 CODABAR

Originalmente desarrollado en 1972 y adoptado por la Comisión Americana de Bancos de Sangre como el estándar para identificar las bolsas de sangre; en la actualidad se utiliza comúnmente en la bibliotecas, librerías, otras aplicaciones médicas y en aplicaciones de paquetería Express aérea.

El Codabar es una simbología discreta de longitud variable, con auto chequeo bidireccional y es numérico extendido. Contiene un conjunto de 20 caracteres los números del 0 al 9 y los caracteres: $,:,/,.,+ y -. Hay cuatro diferentes caracteres de inicio y final: A, B, C y D.

Cada carácter consiste de siete elementos, cuatro barras y tres espacios con dos o tres elementos que pueden ser anchos (uno binario) y el resto angosto (cero binario). Los caracteres de datos están delimitados por caracteres de inicio y final. La dimensión de las barras y espacios de cada carácter está determinada por American National Standard Institute (A.N.S.I.).

Codabar utiliza tres esquemas diferentes de codificación de caracteres:

1. Los dígitos del cero al nueve (0-9), los caracteres "$" y "-" son impresos con una barra ancha y un espacio ancho; todos los demás caracteres o elementos son angostos.

2. Los cuatro caracteres especiales (":", "/", "." y "+") son codificados con tres barras anchas y sin espacios anchos.

3. los cuatro caracteres de inicio / final (A, B, C y D) son codificados con una barra ancha y dos espacios anchos.

Cuando se imprime en este estilo cada carácter tiene la misma anchura. Muchas de las dimensiones publicadas difieren sólo por algunos diez milésimos de pulgada.

Simbología del Codabar

Caracter	Binario	Caracter	Binario
0	0000011	-	0001100
1	0000110	$	0011000
2	0001001	:	1000101
3	1100000	/	1010001
4	0010010	.	1010100
5	1000010	+	0010101
6	0100001	A	0011010
7	0100100	B	0101001
8	0110000	C	0001011
9	1001000	D	0001110

1 binario es ancho
0 binario es delgado
tipo bold son barras
tipo normal es espacio

TABLA 1.5
ESTA TABLA MUESTRA TODA LA SIMBOLOGIA DEL CODIGO CODABAR

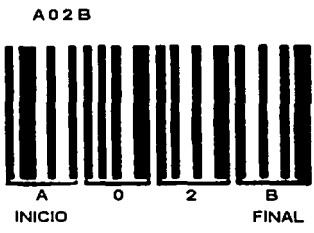


FIGURA 1.4
EJEMPLO DEL CODIGO CODABAR.

1.4.8 El tipo de código utilizado en el proyecto

El tipo de código de barras que se utilizará en éste proyecto es un código que está formado por 17 caracteres y números. En éste código se muestra información acerca de la unidad de empaque que esté identificando. Primero muestra la institución en la cual se forma la unidad de empaque, después, un código que se refiere al tipo de espécimen que contiene esa unidad, después, el tipo de unidad de empaque de que se trata, también muestra un conjunto de caracteres que forman un identificador de la unidad de empaque y por último un digito verificador.

Tenemos que los primeros 5 caracteres son los que muestran la institución, los 3 caracteres siguientes se refieren al tipo de espécimen que contiene la unidad, después aparece el tipo de unidad de empaque formado por 2 caracteres, enseguida el identificador de la unidad de empaque que llamaremos consecutivo y que está formado por 6 caracteres, y por último tenemos el digito verificador que será de un solo carácter.

Para facilitar la lectura del código a los operadores la etiqueta también cuenta con la institución en letra, el espécimen que contiene la unidad de empaque y el tipo de unidad que tiene la etiqueta.

FIGURA 1.5

1.5 LECTORES

1.5.1 Equipo de lectura

Son dispositivos para extraer información codificada en marcas ópticas en un símbolo de código de barras y convertidos en datos digitales compatibles para la computadora, pudiendo ser locamente almacenada para ser descargada más tarde, o puede interactuar con algún programa de aplicación residente en el lector.

Para decodificar la información de un símbolo de código de barras tiene que ejecutar cinco funciones:

1) Determinar la anchura de cada uno de los espacios y de las barras de los símbolos.

2) Cuantificar la anchura de los elementos en un número de niveles apropiados para la simbología a utilizar (niveles 2 para el código 2 de 9, interleaved 2 de 5 y codabar, nivel 4 para UPC / EAN, etc.).

3) Asegurar que la cuantificación de la anchura de elementos es consistente con todas las reglas de codificación de anchura de los elementos, es decir, con un

atabla donde se especifican los valores para esa simbología y determinar el dato codificado.

4) Si es necesario, invertir el orden de los datos. La dirección de la lectura es determinada por un examen de los caracteres de inicio / final.

5) Confirmar que existe una zona muda en ambas terminaciones de la simbología.

El lector de código de barras puede ser considerado como dos elementos por separado y son:

Dispositivo de entrada y Decodificador. Estos elementos pueden estar separados en forma física o por una simple unidad.

1.5.2 Dispositivo de entrada

Es una unidad que emplea técnicas electro-ópticas para rastrear o examinar el símbolo de códigos. El rastreador movible esta provisto para el movimiento manual del operador, por un mecanismo interno en el rastreador o por el movimiento de símbolos pasados al dispositivo de entrada.

Un dispositivo de entrada usualmente es un sistema activo, este ilumina el símbolo con energía luminosa y examina la cantidad de luz reflejada por un área localizada del símbolo.

1.5.3 Decodificador

El decodificador es la parte que analiza lo que recibe de la lectura del código o símbolo, produciendo la entrada a dispositivos periféricos. El resultado de la

decodificación es transmitido a la computadora cargando locamente o remotamente para las aplicaciones residentes programadas.

El proceso de la codificación funciona con una aplicación de software o con un microprocesador, pero la mitad de este se hace manual, siguiendo los pasos:

1) Determinación del dispositivo de entrada y salida.

2) Determinación de la simbología, para decidir el tipo de técnica a utilizar y la distancia adecuada para la lectura.

3) Determinación del ancho, angosto y tamaño del símbolo.

4) Decodificar el símbolo de acuerdo con los estándares ya establecidos.

5) Leer el símbolo de derecha a izquierda utilizando un algoritmo para confirmar la decodificación y detectar si existe algún error.

6) Eficientar la confirmación y validación con:

- Reducción de ruidos en la lectura.
- Confirmación de zonas mudas.
- Chequeo de caracteres.

7) Transmisión correcta de la secuencia de caracteres, emitiendo junto con este los de chequeo.

Los decodificadores se dividen de acuerdo a la operación y la comunicación en:

- *Decodificadores en línea.* Decodifican directamente al equipo, el cual generalmente es una computadora.

- *Decodificadores portátiles.* Contienen una pantalla y una batería recargable para facilitar la portabilidad. Este puede transmitir remotamente o ir almacenando los datos para después transferirlos a la computadora como es el caso del dispositivo SPT 1700.

- *Decodificadores inalámbricos portátiles.* Son los más recientes, portátiles y pueden ser remotos, pero con la diferencia de que pueden trabajar en frecuencias VHF y UHF entre los 1200 a 4800 baudios de potencia.

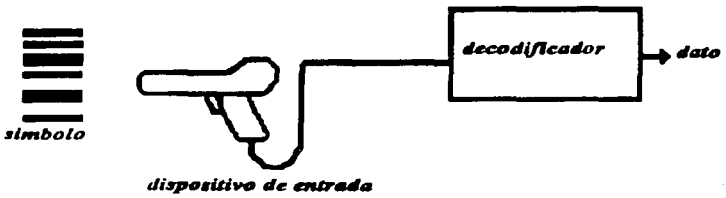
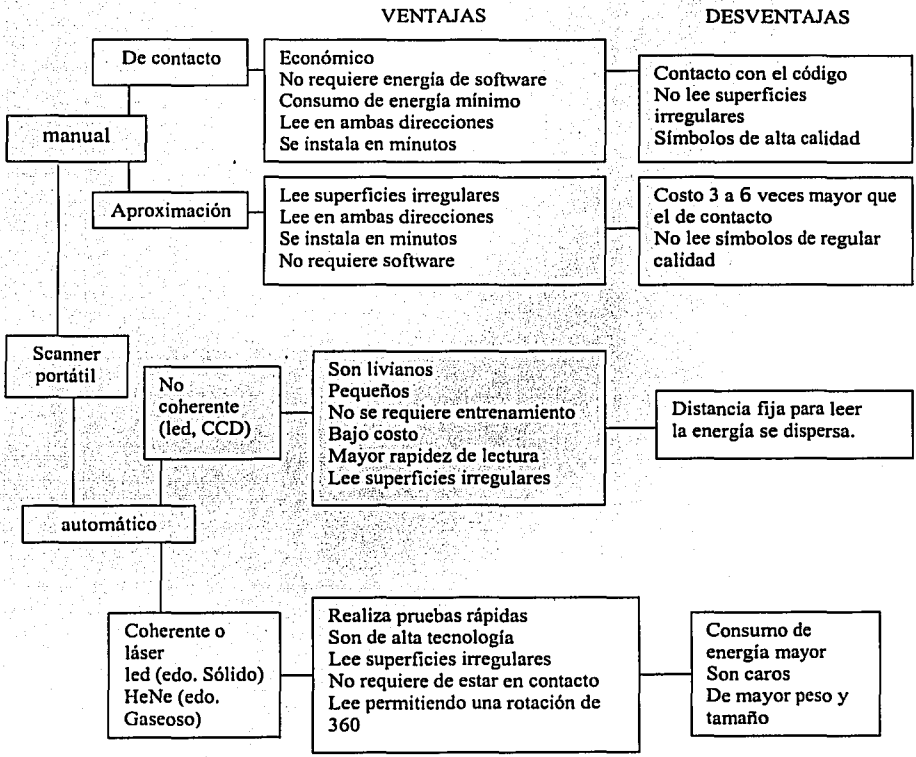


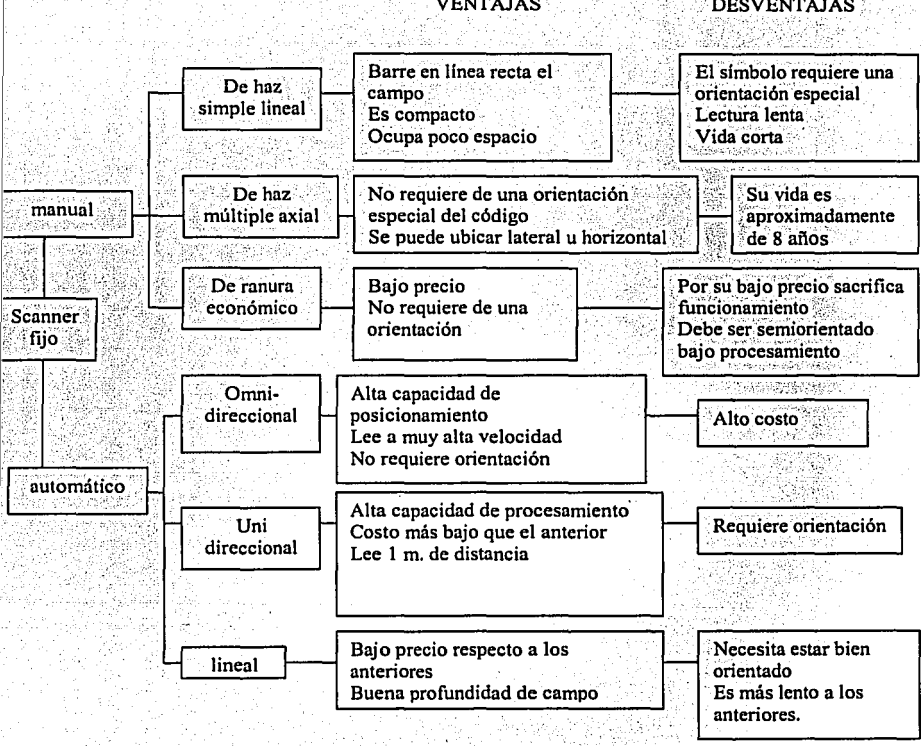
FIGURA 1.6
EN ESTA FIGURA PODEMOS OBSERVAR LOS DOS ELEMENTOS SEPARADOS DEL EQUIPO DE LECTURA.

División de equipos lectores

SCANNER	DE HAZ FIJO	DE HAZ MOVIL
MANUAL	CONTACTO NO CONTACTO	NO CONTACTO
FIJO	NO CONTACTO	NO CONTACO

1.5.4 Ventajas y Desventajas




VENTAJAS
DESVENTAJAS
manual
Scanner fijo
automático
De haz simple lineal
Barre en línea recta el campo
Es compacto
Ocupa poco espacio
El símbolo requiere una orientación especial
Lectura lenta
Vida corta
De haz múltiple axial
No requiere de una orientación especial del código
Se puede ubicar lateral u horizontal
Su vida es aproximadamente de 8 años
De ranura económico
Bajo precio
No requiere de una orientación
Por su bajo precio sacrifica funcionamiento
Debe ser semiorientado bajo procesamiento
Omni-direccional
Alta capacidad de posicionamiento
Lee a muy alta velocidad
No requiere orientación
Alto costo
Uni direccional
Alta capacidad de procesamiento
Costo más bajo que el anterior
Lee 1 m. de distancia
Requiere orientación
lineal
Bajo precio respecto a los anteriores
Buena profundidad de campo
Necesita estar bien orientado
Es más lento a los anteriores.

1.6 IMPRESORES

1.6.1 Impresión del código de barras sobre etiquetas

La impresión del código de barras cae en dos clasificaciones:

- **Impresión "fuera de sitio".** Se refiere a las tecnologías que son usadas para reproducir símbolos en código de barras para uso subsecuente. La producción es generalmente hecha en una localidad diferente de donde los símbolos serán usados: a menudo la generación del símbolo se realiza contratando una organización especializada en la tecnología. La impresión "fuera de sitio" es generalmente usada para crear volúmenes medios a grandes de símbolos idénticos o en serie. Además del tiempo de separación entre la impresión del símbolo y su uso, éste es escrito como un proceso serie o de lote.

- **Impresión "en sitio".** Las técnicas de impresión "en sitio" son usadas para crear símbolos en código de barras a la hora y lugar que ellas serán usadas. El dato codificado en cada símbolo puede ser diferente y es introducido por cualquier vía, ya sea, por un teclado local o por una computadora adjunta o acoplada. Las impresoras "de sitio" son a menudo referidas como impresoras de demanda por su habilidad para producir especialmente codificaciones de código de barras en demanda.

Las impresoras "de sitio" pueden también ser usadas para producir grandes cantidades de símbolos idénticos o en serie.

1.6.2 Impresión de texto sobre etiquetas autoadhesivas

Los códigos de barras no siempre pueden o deben imprimirse directamente sobre el substrato de un artículo o producido por diferentes motivos, y en estos casos es necesario imprimir los símbolos sobre etiquetas autoadhesivas[8].

Los motivos para entrar en este sistema son generalmente:

a) Necesidad de volver a codificar el envase de un producto erróneamente o mal codificado.

b) Necesidad de codificar pequeñas producciones que no justifican la impresión del código en el envase impreso.

c) Necesidad de codificar individualmente productos distintos que utilizan un mismo patrón, tipo de embalaje o distintas formas de presentación de un mismo producto.

d) Necesidad de codificar en el punto mismo de distribución o supermercado, productos sin código, ofertas de productos agrupados, distintos tamaños y/o medidas o colores de prendas de vestir y textiles en general.

e) Necesidad de traducir información de códigos de símbolo (barras) en pequeñas cantidades, generando o no numeración automática de artículos, sea en circuito cerrado en empresa o para distribución y consumo, también cuando se requiere frecuentemente cambios en la codificación.

[8] Estas etiquetas autoadhesivas son utilizadas en este proyecto de tesis.

f) Necesidad de imprimir distintos tipos de códigos según el uso o destino del producto, especialmente en los insumos industriales y materiales semi-elaborados según especificaciones de codificación entre industrias.

g) Necesidad de códigos muy pequeños y de alta calidad, en cantidades limitadas.

h) Empresas de servicios, fabricantes de etiquetas que ofrecen a sus clientes los servicios de impresión, preimpresión y precodificación para cualquiera de las situaciones anteriores.

Para cualquiera de estas situaciones existen básicamente ocho sistemas de impresión de código de barras sobre etiquetas, sean estas autoadhesivas o no, estos sistemas difieren entre si en la inversión inicial, en el costo unitario de impresión, la velocidad, calidad y finalmente en la compatibilidad con el sistema de computación existente, con la existencia del papel habitual, con el tipo y formato de las demás etiquetas en producción, etc.

La empresa que opte por utilizar alguno de estos sistemas para codificar sus productos, debe tener siempre presente todas las normas sobre el diseño del código que también se aplican a las etiquetas, por ejemplo, la ubicación del código en el envase, color, contraste y control de calidad.

En la mayoría de los sistemas de impresión de códigos sobre etiquetas, la impresión se limita generalmente a las barras negras, utilizando como espacios el fondo natural de la etiqueta que obviamente deberá ser blanco, el fondo sólo podría colorearse de rojo, anaranjado o amarillo siempre y cuando se verifique previamente que el contraste y reflectancia obtenidos concuerda con las especificaciones.

Las etiquetas deben cumplir con todas las especificaciones del código de barras y esto incluye lógicamente a las zonas mudas derecha e izquierda, que deben ser respetadas y de ser posible, ampliadas, especialmente cuando las etiquetas son preimpresas.

Es muy común el uso de este tipo de etiquetas donde previamente se imprime el patrón o muestra con el nombre del cliente, o leyendas como "gracias por su compra", o similares; generalmente en varios colores y dejando el espacio necesario para imprimir posteriormente las barras del código y otras informaciones como fecha, peso, precio unitario y total, etc.

CAPITULO 2

BASES DE DATOS RELACIONALES

2.1 BASES DE DATOS

El término de *Bases de Datos* no apareció hasta mediados de los años sesenta, época en la cual la información era representada haciendo uso de un conjunto de ficheros, generalmente planos. Estos ficheros no estaban relacionados entre sí, y los datos almacenados representaban las relaciones existentes en la información que representaban mediante referencias simbólicas o físicas. La redundancia era grande y la integridad de la información representada dejaba mucho que desear.

Aún así, muchos desarrolladores de software *"bautizaban"* a sus sistemas de ficheros[1] como *Bases de Datos,* sin preocuparse de que cumplieran o no una serie de propiedades que deben acompañar al uso de este término. Como los requerimientos cambian con el tiempo, la información a ser tratada en cada problema cambia y, por tanto, es necesario, de alguna manera, independizar la estructura de la información de los procedimientos encargados de su tratamiento, si no se estaría siempre avocado a la dedicación de una gran cantidad de esfuerzo a la modificación de todos aquellos procedimientos encargados del mantenimiento de la información.

Cuando se reconoce que los sistemas evolucionan y, que por tanto, la información y la estructura de la misma no es estática sino que ha cambiado con el tiempo, es cuando aparece el concepto de las *Bases de Datos.* Si se desea que cualquier modificación en la cantidad, contenido y estructura de la información que se desea mantener acerca de un determinado problema no afecte a los procedimientos desarrollados previamente para el mantenimiento de la misma, es necesario tener en cuenta que existe una *independencia de los datos con respecto a los procedimientos.*

La independencia de los datos con respecto a los procedimientos supone, que la visión conceptual de los datos no tiene porque ser la misma que la visión física de los

[1] El término fichero es la forma antigua de referirse a lo que ahora es llamado archivo.

mismos (la estructura de los archivos utilizados para su almacenamiento). Si los procedimientos encargados del mantenimiento de la información sólo *"ven"* la estructura física de los datos y si ésta se realiza a nivel de ítem de datos, un cambio en la visión conceptual no tienen porqué afectar, en principio, a estos procedimientos.

2.1.1 Características de las bases de datos

La información que forma parte de una base de datos puede organizarse de múltiples formas pero con independencia de la arquitectura de la base de datos, ésta debe cumplir una serie de características para ser considerada como tal, algunas de las cuales se describirán a continuación:

Versatilidad para la representación de la información: si bien la información que forma parte del dominio de un problema es única y caracteriza a ese problema o sistema, pueden existir diferentes visiones de esa información, visiones parciales en las que sólo se tiene en cuenta parte del dominio del problema o visiones globales que observan el problema desde diferentes puntos de vista.

Si se considera que un procedimiento *"ve"* la información que maneja como un registro, la organización de la información en la base de datos debe permitir que diferentes procedimientos puedan construir diferentes registros a partir de la información existente en la base de datos. Estos registros (lógicos) estarán formados por ítems de datos que forman parte del dominio del problema y que son derivados del conjunto de los ítems de datos existentes en ese problema y, además, cada uno de estos registros lógicos construidos por los procedimientos deben ser independiente de los registros físicos existentes en la base de datos para almacenar la información.

Desempeño: las bases de datos deben asegurar un tiempo de respuesta adecuado en la comunicación hombre-máquina, permitiendo el acceso simultaneo al mismo o distinto conjunto de ítems de datos por el mismo o distinto procedimiento.

Mínima redundancia: Una de las principales razones por las que surgió la tecnología de las bases de datos fue el evitar la alta redundancia que se presentaba en los sistemas de procesamiento de la información debido al uso de archivos con estructuras planas. Sin embargo, las bases de datos no evitan totalmente la redundancia en la información debido a que es necesario representar todas las razones que existen entre las entidades que forman parte del dominio del problema.

La existencia de redundancia es nefasta debido a la posibilidad de inconsistencia en la información almacenada en la base de datos. La redundancia implica la existencia de varias copias de un mismo ítem de datos las cuales pueden, en un momento dado tener distintos valores.

Un objetivo principal de las bases de datos es eliminar la redundancia siempre que ello implique una complejidad de la misma y una disminución en el desempeño. Si existen diversas copias de un mismo ítem de datos, es necesario establecer procedimientos que garanticen la consistencia de la información cuando estas copias sean utilizadas por diferentes procedimientos al mismo tiempo. Por otro lado, si sólo existe una copia de un ítem de datos es necesario establecer procedimientos que permitan el acceso a esta copia por varios procedimientos, para garantizar un desempeño aceptable, lo que complica el software de gestión y la representación del dominio del problema en la base de datos.

Capacidad de acceso: los usuarios de la base de datos solicitan a esta continuamente información sobre los datos almacenados. Estas peticiones a la base de datos, que pueden ser conocidas o no cuando se diseñó la misma, solicitan información correspondiente a distintos ítems de datos, así como sus relaciones, representadas en la base de datos de múltiples formas.

Una base de datos debe ser capaz de responder, en un tiempo aceptable, a cualquier consulta sobre la información que contiene, sin restricciones graves en cuanto a

los ítems, relaciones, formato, etc., solicitados en la misma, y respondiendo al usuario rápidamente.

Esta característica va a depender directamente de la organización física de los datos en la base de datos. Una organización física *"muy completa"* garantiza una respuesta rápida a la consulta aunque requiere un mayor coste computacional en actualizaciones.

Simplicidad: La base de datos representa el dominio de un problema que se necesita tratar computacionalmente. La naturaleza de este problema puede ser muy variada, y por tanto, existir en el mismo un número de objetos variables que se relacionan de múltiples formas. Es por ello la naturaleza del problema, un factor de complejidad de partida de las bases de datos que es conveniente eliminar para que se garanticen otras de las características que se le requieren.

Las bases de datos deben estar basadas en representaciones lógicas simples que permitan la modificación de los requerimientos en el problema, de tal forma que la inclusión de nuevos ítems y relaciones no ocasione una complejidad excesiva.

Integridad: La integridad de una base de datos hace referencia a la veracidad de los datos almacenados con respecto a la información existente en el dominio del problema que trata la misma. Como los datos de la base de datos son manejados por muchos usuarios haciendo uso de muchos procedimientos que tratan los mismos datos de muchas formas, es necesario garantizar que estos datos no sean destruidos ni modificados de forma anómala.

Durante el procesamiento se pueden producir fallos de muy diversa naturaleza: errores del sistema general, del hardware, software, etc. Así, los procedimientos que manejan la información deben asegurar que el sistema pueda garantizar la integridad de la

información a pesar de los errores que se puedan producir, temporalmente, a causa de los fallos con independencia de su naturaleza.

Seguridad y Privacidad: La seguridad de una base de datos hace referencia a la capacidad de ésta para proteger los datos contra su pérdida total o parcial por fallos del sistema o por accesos accidentales o intencionados a los mismos. Mientras que la privacidad de una base de datos hace referencia a la reserva de la información de la misma a personas no autorizadas. Deben existir en la base de datos tanto procedimientos de recuperación de la información, como procedimientos que supervisen el acceso a los datos por los usuarios de la base de datos.

Afinación: La afinación hace referencia a la organización física de la información de la base de datos, la cual determina directamente el tiempo de respuesta de los procedimientos que operan sobre la misma.

Si una de las características que debe tener una base de datos es un buen desempeño, la organización física de los datos debe ser tal que ésta pueda ser alcanzada. Pero la base de datos evoluciona con el tiempo, el volumen de información va haciéndose cada vez mas importante y, por añadidura, tanto los ítems de datos como las relaciones entre ellos pueden ampliarse y modificarse.

Interfaz con el pasado y futuro: Es aceptado que el problema cambia evolucionando con el tiempo, las necesidades de la organización cambian continuamente y, por lo tanto, cambia la información correspondiente al subsistema o dominio del problema de la misma. Una base de datos debe estar abierta a estos cambios de forma que no afecten, o afecten lo mínimo posible, a los procedimientos existentes para manejar la información que mantiene.

2.2 EL MODELO DE DATOS RELACIONAL

Fue *E.F. Codd* quien desarrolló en *IBM-San José (California)* el modelo de datos relacional. Este modelo está basado en conceptos muy sencillos teniendo asociada la teoría de normalización de relaciones que tiene por objeto la eliminación de los comportamientos anómalos de las relaciones durante los procesos de manejo de la información que representan y la eliminación de redundancias superfluas, facilitando así, la comprensión del esquema en cuanto a las relaciones semánticas existentes entre los objetos del dominio del problema.

El modelo relacional es un modelo lógico de datos. Una base de datos relacional puede ser estructurada físicamente de múltiples formas, la representación física deberá satisfacer y representar, de alguna forma las relaciones y restricciones lógicas del esquema relacional.

El modelo relacional propone una representación de la información que:

- Origine esquemas que representen fielmente la información, los objetos y relaciones entre ellos existentes en el dominio del problema.

- Pueda ser entendida fácilmente por los usuarios que no tienen una preparación previa en esta área.

- Haga posible ampliar el esquema de la base de datos sin modificar la estructura lógica existente y, por tanto, sin modificar los programas de aplicación.

- Permita la máxima flexibilidad en la formulación de los interrogantes previstos, y no previstos, sobre la información mantenida en la base de datos.

2.2.1 Terminología del modelo relacional

Una de las formas más naturales, y por tanto fácilmente entendibles por el usuario, de representar la información es en base al uso de una representación tabular plana de la misma. Una tabla bidimensional mediante la cual se representen tanto los objetos como las relaciones entre ellos existentes en el dominio del problema.

Una tabla es una matriz rectangular que puede ser descrita de forma simple matemáticamente y que posee las siguientes propiedades:

1) Cada entrada de la tabla, es decir, cada elemento de la matriz rectangular, representa a un ítem de datos elemental.

2) Una tabla es homogénea por columnas; es decir, todos los ítems de datos elementales de una columna (en todas las filas) son de la misma clase y, por tanto, están definidos en el mismo dominio de datos y representan una misma propiedad o característica en el dominio del problema.

3) Cada columna de la tabla tiene asignado un nombre único en el conjunto de columnas de esa tabla, aunque pueden existir tablas diferentes con columnas de igual nombre.

4) Para una tabla todas las filas son diferentes, no se admiten filas duplicadas.

5) Tanto las filas como las columnas pueden ser consideradas en cualquier secuencia sin afectar, por ello, ni al contenido de la información ni a la representación semántica de la misma.

Como cualquier modelo de datos, el modelo relacional, introduce su propia terminología para nominar los objetos y elementos utilizados por el modelo para representar el dominio de la información. Por ejemplo, a una tabla o matriz rectangular

se le denomina *relación*, a las filas de la misma se les denomina *tuplas* y al conjunto de sus columnas *dominio* de la relación. Así, una base de datos relacional estará formada por un conjunto de relaciones.

2.2.2 Relación

Dada una serie de conjuntos D_1, D_2, D_3 ...D_n, no necesariamente distintos, se dice que R es una relación entre estos n conjuntos si es un conjunto de n tuplas ordenadas (d_1, d_2, d_3,...,d_n) tales que $d_1 \in D_1$, $d_2 \in D_2$, $d_3 \in D_3$, ... , $d_n \in D_n$. A los conjuntos D_1, D_2, D_3,...,D_n se les denomina dominios de R, y el valor de n es el grado de la relación R.

ALUMNO

Matricula	nombre	apellidos	curso	Nota
3456	José	Pérez de la Lastra	1	5.25
0101	María	Antúnez Sastrez	1	7.80
8743	Lourdes	Sánchez Argot	1	4.50
1234	Antonio	Soria Madrid	3	6.35
5674	Luis	González Silos	2	3.20
0678	Pilar	Alcántara Badajoz	2	5.50
0345	Dolores	Almiz Márquez	3	7.30
2985	Manuel	Rivas Fuentes	3	3.50

Tabla 2.1: Ejemplo de relación.

Un ejemplo de relación que se muestra en la Tabla 2.1. Se trata de una tabla denominada *Alumno*, en cuyo ejemplo están presentes ocho tuplas. La relación es de grado cinco. Los cinco dominios son conjuntos de valores que representan, respectivamente el número de matricula de los alumnos, el nombre, los apellidos, el curso en el que están matriculados, y la nota obtenida por los alumnos.

El dominio correspondiente a la *nota* es el conjunto de todas las notas posibles que pueden ser asignadas a los alumnos aunque, como se observa en el ejemplo no todos los

valores posibles tienen que estar presentes en un momento dado en una relación en la que exista ese dominio.

Al número de tuplas de una relación en un instante dado se le denomina *cardinalidad* de la relación. Así, la relación *Alumno* tiene una cardinalidad de ocho. Al número de columnas de una relación se le denomina grado de la relación. Así, la relación *Alumno* tienen un grado de cinco.

Mientras la cardinalidad de una relación depende del momento en que ésta sea considerada, el grado de una relación es independiente del tiempo. El grado de una relación hace referencia al número de dominios que define la relación y éstos son independientes del momento en que ésta se considere. Dependiendo del grado de la relación éstas se denominan: unarias, binarias, terciarias, etc. Se puede introducir, en estos momentos, otra definición equivalente de relación como sigue:

Dados una serie de conjuntos D_1, D_2, D_3, ... ,D_n, no necesariamente distintos, se define el producto cartesiano de estos conjuntos y se denota por $D_1 \times D_2 \times D_3 \times ... \times D_n$, como un nuevo conjunto formado por todas las tuplas ordenadas posibles (d_1, d_2, d_3, ..., d_n), tales que $d_1 \in D_1$, $d_2 \in D_2$, $d_3 \in D_3$, ..., $d_n \in D_n$.

Se dice que R es una relación sobre los conjuntos D_1, D_2, D_3, ..., D_n si es un subconjunto del producto cartesiano de los mismos.

Como se puede observar no se define ningún orden de los elementos que forman la relación (las tuplas), puesto que una relación es un conjunto y los conjuntos no son ordenados. Si bien, desde el punto de vista de la teoría de conjuntos, el orden de las columnas si influye (el orden en que se realiza el producto cartesiano de los conjuntos determina la estructura de los elementos del nuevo conjunto; es decir, el conjunto $D_1 \times D_2$ es distinto del conjunto $D_2 \times D_1$), no así en las relaciones. La razón es simple, en una relación cada columna tiene asignado un nombre único en el contexto de la relación, y por

tanto, es posible distinguir el significado de los elementos constituyentes del conjunto, con independencia del orden de consideración de los mismos.

2.2.3 Dominios y atributos

Es importante aclarar desde el principio la diferencia existente entre dominio y atributo. Un atributo representa el uso de un dominio para un determinada relación; es decir, un atributo porta un significado semántico a un dominio. Mientras que un dominio es un conjunto homogéneo definido mediante el uso de la abstracción en base a otro conjunto.

Para aclarar esta diferencia, para el ejemplo de la relación mostrada en la Tabla 2.1., se van a asignar nombres diferentes a los dominios y a los atributos, de la forma siguiente:

Define Dominio	Expediente	Entero (4)	**Fin definición**
Define Dominio	Primer nombre	Carácter (15)	**Fin definición**
Define Dominio	Final nombre	Carácter (40)	**Fin definición**
Define Dominio	Estudios	Entero (2)	**Fin definición**
Define Dominio	Nota	Real (4)	**Fin definición**

Define Relación *Alumno*

(matricula #:	**Dominio**	Expediente
Nombre:	**Dominio**	Primer nombre
Apellidos:	**Dominio**	Final nombre
Curso:	**Dominio**	Estudios
Nota:	**Dominio**	Nota)

Se han definido, en este ejemplo, cinco dominios, y en base a ellos son definidos los cinco atributos de la relación. Se observa que el dominio puede o no tener asignado el mismo nombre que el atributo mediante el cual se define la relación. De igual forma, puede definirse más de un atributo sobre un mismo dominio.

2.2.4 Intención y extensión de relaciones

Una relación en una base de datos relacional tiene dos componentes: la intención o comprensión y la extensión, aunque es usual referirse a los dos componentes al mismo tiempo cuando se utiliza el término relación.

La intención de una relación hace referencia a la estructura estática del objeto del mundo real, el cual es representado mediante la relación. Es decir, la intención va a ser siempre invariablemente con el tiempo y se trata de la definición de las propiedades (atributos) del objeto del mundo real representado por la relación, de las cuales cada una está definida en su correspondiente dominio de datos.

Por otro lado, la existencia de una relación depende del momento específico en el cual la relación es tenida en cuenta, y hace referencia al conjunto de tuplas que forman parte de la relación en un instante dado. La extensión de una relación representa a cada uno de los objetos (tuplas), pertenecientes a un mismo tipo (relación), existentes en el dominio del problema en un momento dado.

2.2.5 Claves de las relaciones

Es usual que en el conjunto de atributos que forman parte de la intención de una relación específica, uno o un conjunto de ellos tengan la propiedad de tomar valores únicos en el dominio del problema para cualquier extensión de esa relación y, por tanto, tengan la facultad de identificar sin ambigüedad y de forma única a una, y sólo una, de las tuplas de esa relación.

A los atributos, tal vez compuestos, que satisfacen la propiedad de identificación única de las tuplas de una relación se les denomina *claves candidatas de la relación.*

Toda relación, por definición, debe tener clave candidata. Una clave candidata puede estar formada por uno o un conjunto de atributos. Si por definición, el modelo relacional, no pueden existir tuplas duplicadas en una relación, al menos la agregación de todos los atributos de una relación cumplirá la condición de clave candidata para esa relación.

De entre todas las claves candidatas de una relación, en la definición del esquema se deberá especificar cuál de ellas se considera como *clave primaria o principal,* denominándose al resto de las claves candidatas como *claves alternas.* La distinción entre clave primaria y alterna va a tener influencia sobre la implementación física de la relación y sobre los procesos de acceso a la información mantenida en la misma, pero no sobre la semántica de la relación.

2.3 NORMALIZACION DE RELACIONES

Como se ha descrito, el modelo relacional está soportado sobre una teoría de igual nombre basada en los principios de la teoría general de conjuntos. Si bien una relación representa a un conjunto, y como tal puede y debe ser considerada, no todos los conjuntos pueden ser considerados en un esquema relacional para satisfacer en la base de datos los siguientes objetivos:

1) No existencia de redundancias superfluas, aminorando el espacio requerido para el almacenamiento de la información y, por tanto, reduciendo posibles problema de integridad en la información almacenada en la base de datos.

2) Aumentar el desempeño de las operaciones de actualización de la base de datos.

3) Representar de forma coherente los objetos y relaciones existentes en el dominio del problema y cuya información es almacenada en la base de datos.

4) Aumentar el desempeño y garantizar la fiabilidad de las interrogaciones sobre la información mantenida en la base de datos.

Para satisfacer estos objetivos, las relaciones que forman parte de un esquema relacional deben satisfacer una serie de reglas que restringen el universo de relaciones y conjuntos que pueden ser considerados en un esquema relacional.

A este conjunto de reglas se les denomina *Reglas de normalización de relaciones* y a la teoría en la que se basan se le denomina *Teoría de normalización de relaciones.*

2.3.1 Dependencias funcionales

La normalización de relaciones está basada en otra teoría, la *teoría de las dependencias,* la cual se centra en el estudio de las dependencias que presenta cada atributo de una relación con respecto al resto de atributos de la misma relación.

Dada una relación R se dice que el atributo R.y $\in$ *R es funcionalmente dependiente de otro atributo R.x* $\in$ *R y se expresa de la forma R.x* $\rightarrow$ *R.y si, y sólo si, cada valor de R.x tiene asociado a él exactamente un valor de R.y para cualquier extensión de la relación R.*

Se puede apreciar lo siguiente en la definición anterior:

- El concepto de dependencia funcional tiene en cuenta a atributos de una misma relación y no a atributos de relaciones diferentes.

- El concepto de dependencia funcional hace referencia a la relación funcional que pueda existir entre parejas de atributos de una relación.

- El concepto de dependencia funcional es independiente del estado o extensión de una relación y, por lo tanto, es intrínseco a la intención de una relación.

- La definición anterior no hace referencia explícita a la complejidad de los atributos dependientes, por lo que:

  - Los atributos funcionalmente dependientes de una relación pueden ser simples o estar formados por la agregación de varios atributos, R.x y/o R.y pueden ser de la forma:

$$R.x \equiv \{R.a, R.b, ..., R.n\}$$

  Para R.a, R.b,..., R.n $\in$ R.

  - Los atributos funcionalmente dependientes pueden ser, o no, atributos que formen parte de la clave primaria o de alguna clave candidata de la relación.

- En la definición anterior no se hace referencia alguna al número de veces (número de tuplas) en las que el atributo R.x tiene un mismo valor.

Se puede apreciar que la definición anterior de dependencia funcional no impone la restricción de que no puedan repetirse los valores del atributo R.y para distintos valores del atributo R.x.

Si en la relación *Alumno* se considera que la agregación de los atributos *nombre y apellidos* puede ser una clave candidata de la relación, se observa que también existirá una dependencia funcional entre el atributo *matricula#* y el agregado (nombre + apellidos). Puesto que para un valor dado de matricula# siempre se tendrá asociado un único valor del agregado *(nombre + apellidos).* Pero además, como el agregado *(nombre + apellidos)* es una clave candidata de la relación *Alumno,* también se cumple que para un

valor dado del agregado *(nombre + apellidos)* se tendrá asociado siempre un único valor del atributo *matricula#*. Se dice en estos casos que existe una dependencia funcional mutua entre los atributos R.x y R.y de la relación R, y se representa de la forma R.x $\leftrightarrow$ R.y.[9]

Se puede introducir, en estos momentos, una definición algo más amplia de dependencia funcional, de la forma:

Dada una relación R, se dice que el atributo R.y $\in$ R es funcionalmente dependiente de otro atributo R.x $\in$ R, y se expresa en la forma R.x $\rightarrow$ R.y si, y sólo si. siempre que dos o más tuplas de R coincidan en sus valores de R.x, también, para esas tuplas, existirá una coincidencia en los valores del atributo R.y.

2.3.2 Representación textual de las dependencias funcionales

Formato general

R.x $\rightarrow$ (R.a, R.b, ..., R.n)

denotando que los atributos R.a, R.b, ..., R.n son funcionalmente dependientes del atributo R.x.

2.3.3 Representación gráfica de las dependencias funcionales

Formato general

[9] El signo + se ha utilizado para representar la agregación de atributos.

Es conveniente introducir en estos momentos el concepto de *Dependencia funcional completa*, de la forma:

Se dice que el atributo R.y ∈ R es funcionalmente dependiente y de forma completa de otro atributo R.x ∈ R, si y sólo si depende funcionalmente de R.x y no de ningún subconjunto de los atributos que formen parte del atributo R.x.

Según esta definición se puede apreciar que:

- Para que una dependencia funcional no sea completa, el atributo R.x debe ser un agregado formado por la concanetación de varios atributos pertenecientes a la relación.

- Que el atributo R.y sea simple o compuesto no tiene relevancia para que la dependencia funcional R.x → R.y sea completa o no.

En la relación *Alumno* se podría considerar que existe una dependencia entre el atributo nota y el agregado *(matricula# + curso)*, *(Alumno.(matricula#, curso) → Alumno.nota)*; pero esta dependencia funcional no es completa puesto que el atributo nota depende también funcionalmente del atributo *matricula#*.

2.3.4 Reglas de normalización

La teoría de la normalización está basada en la aplicación de una serie de reglas a las que se les denomina *Reglas de normalización*. Se dice que una relación está en una determinada forma normal si satisface un cierto conjunto específico de restricciones impuestas por la regla de normalización correspondiente. La aplicación de una regla de normalización es una operación que toma una relación como argumento de entrada y da como resultado dos o más relaciones.

De esta forma, la sucesiva aplicación de las reglas de normalización va a dar lugar a la generación de un número mayor de relaciones que formen parte del esquema relacional, y desde un punto de vista sólo lógico, una redundancia de los atributos considerados en el esquema. La aplicación sucesiva de las reglas de normalización restringe, por tanto, el número de relaciones que la satisfacen. Por regla general se dice que un esquema relacional es consistente si las relaciones satisfacen al menos la forma normal de *Boyce-Cood.*

2.3.5 La primera forma normal FN1

Una relación R satisface la primera forma normal (FN1) si, y sólo si, todos los dominios subyacentes de la relación R contienen valores atómicos.

La aplicación de esta regla es fácil y directa para cualquier relación, y este proceso se realiza de forma automática en el proceso de análisis del dominio del problema. Simplemente consiste en descomponer aquellas tuplas en las que los atributos tengan más de un valor en tantas tuplas como valores estén presentes. De hecho, es una restricción innata al propio modelo relacional. El que una relación se encuentre en FN1 no es condición suficiente, aunque sí necesaria, para garantizar la consistencia del esquema racional.

2.3.6 La segunda forma normal FN2

Una relación R satisface la segunda forma normal (FN2) si, y sólo si, satisface la primera forma normal y cada tributo de la relación depende funcionalmente de forma completa de la clave primaria de esa relación.

Para explicar la naturaleza de esta forma normal se va a analizar una nueva relación denominada *Matricula,* como se muestra a continuación:

Esquema-1

Matricula (*dni, asignatura#, apellidos, nombre, nota, curso, aula, lugar)*

- El atributo *asignatura#* representa la identificación de las asignaturas en las que se encuentra matriculado cada uno de los alumnos.

- El atributo *aula* representa las aulas en las que se imparte la docencia de las asignaturas.

- El atributo *lugar* representa los lugares de estudio en los que se imparte la docencia correspondiente a las asignaturas.

- El atributo *curso* representa el curso en el que se imparte la docencia de una asignatura. Por tanto, se va a suponer que existe una dependencia funcional entre los atributos *asignatura# y curso*, de a forma: *Matricula.asignatura#* $\rightarrow$ *Matricula.curso,* que representa que si bien en un curso se puede impartir docencia para varias asignaturas, una asignatura está asignada a la docencia de un único curso.

Se puede apreciar que esta relación no se encuentra en FN2 puesto que existe una dependencia funcional no completa entre atributos de la relación que no forman parte de la clave (atributos no primos) y la clave de la relación.[10]

La relación **Matricula** presenta una serie de inconvenientes debidos a que no satisface la FN2, inconvenientes que se ponen de manifiesto en:

Inserción de tuplas: se aprecia que no se pueden conocer las asignaturas que se imparten en un curso hasta que no exista algún alumno matriculado en esas asignaturas.

[10] A los atributos que forman parte de la clave de una relación se les denomina atributos primos, y a los que no forman parte de la clave, atributos no primos.

Por ejemplo, si la asignatura '4' se impartiera en el curso '3', esta información no podría ser conocida hasta que algún alumno se hubiera matriculado en esta asignatura. Debido, simplemente, a que en caso contrario se violaría la integridad de la clave al existir valores nulos en la misma.

Borrado de tuplas: de igual forma se aprecia que si se eliminan las tuplas correspondientes a los alumnos matriculados en una asignatura, se pierde la información que representa que una asignatura forma parte de la docencia de un curso.

Actualización de tuplas: si se realiza un cambio de asignación de curso para una asignatura, sería necesario actualizar todas las tuplas correspondientes a todos los alumnos matriculados en esa asignatura. Esto es debido a que existe una gran redundancia de información, puesto que en la relación para cada alumno de una asignatura se almacena el curso en que se imparte, mientras que esta información es independiente del alumno que está matriculado en esa asignatura, dependiendo únicamente de la asignatura.

Para eliminar todos estos inconvenientes es necesario realizar un proceso de descomposición de la relación Matricula en dos nuevas relaciones, las cuales satisfacen la segunda forma normal. Estas relaciones quedarían con la siguiente estructura:

Esquema-2
Imparte *(asignatura#, curso)*
Matricula-2 *(dni, asignatura#, apellidos, nombre, nota, aula, lugar)*

Se aprecia que:

- La relación *Imparte* se encuentra en FN2. La clave de esta relación es el atributo *asignatura#*, y el único atributo no primo *(curso)* depende funcionalmente de forma completa de la clave de la relación.

- Al eliminarse el atributo *curso* en la relación *Matricula-2*, se ha eliminado la dependencia funcional no completa entre el atributo curso y la clave de la relación (la existente entre los atributos *asignatura# y curso)* y, por lo tanto, los problema que causaba esta dependencia en los procesos de mantenimiento de la información para esta relación.

- No se ha producido pérdida de información, puesto que el atributo *asignatura#* debe definirse como clave foránea de la relación *Imparte.* De esta forma, el atributo *asignatura#* en la relación *Matricula-2* para cualquier tupla, sólo podrá tomar valores existentes en alguna tupla de la relación *Imparte.* Así, en base a esta referencia podrá conocerse en cada momento en qué curso es impartida cada asignatura para cada uno de los alumnos matriculados.

Esquema-3
Imparte (*asignatura#, curso)*
Alumno-2 (*dni, apellidos, nombre)*
Matricula-3 (*dni, asignatura#, nota, aula, lugar)*

2.3.7 La tercera forma normal FN3

Una relación R satisface la tercera forma normal (FN3) si, y sólo si, satisface la segunda forma normal y cada atributo no primo de la relación no depende funcionalmente de forma transitiva de la clave primaria de esa relación. Es decir, no pueden existir dependencias transitivas entre los atributos que no forman parte de la clave primaria de la relación R.

Se puede observar que la relación *Matricula-3,* la cual se encuentra en FN2 sigue presentando problemas en los procesos de manipulación de la misma. Los problemas son debidos a que existe una dependencia entre los atributos aula y lugar.

Si se considera, como es lógico, que cada aula se encuentra ubicada físicamente en un único lugar, y que la docencia de una determinada asignatura -para un determinado conjunto de alumnos; es decir, diferentes alumnos matriculados en una asignatura pueden recibir docencia en aulas diferentes (grupos de clase)- se imparte en una única aula, se observa que existe una dependencia funcional entre los atributos no primos *lugar y aula,* además de las dependencias funcionales completas entre los atributos lugar y aula con la clave de la relación *Matricula-3.*

La existencia de estas dependencias entre los atributos no primos ocasiona problemas en la manipulación de la relación *Matricula-3,* problemas en:

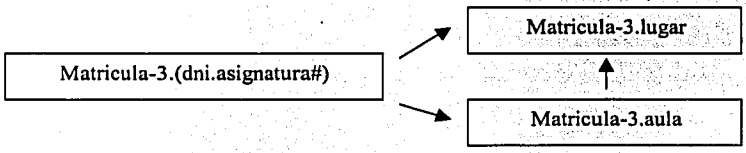
Inserción de tuplas: se observa que no se pueden conocer las aulas de cada uno de los lugares utilizadas para la docencia de asignaturas hasta que no haya algún alumno matriculado en esa asignatura.

Borrado de tuplas: una vez borrados todos los alumnos matriculados en una asignatura, se pierde la información correspondiente a las aulas de cada lugar utilizadas para la docencia.

Modificación de tuplas: existe una gran redundancia de información puesto que se almacena el lugar en el que se encuentra ubicada un aula para cada uno de los alumnos que cursan una asignatura que se imparte en dicha aula.

Estos problemas se deben, no a la presencia de una dependencia funcional no completa, sino a la presencia de un dependencia funcional transitiva. La relación *Matricula-3* presenta las siguientes dependencias:

Matricula-3 (dni, asignatura#) → Matricula-3. aula
Matirucla-3 (dni, asignatura#) → Matricula-3. lugar
Matricula-3 (aula) → Matricula-3.lugar



Y, como se observa, el atributo *lugar* es dependiente de la clave de la relación de forma transitiva, debido a que es dependiente funcionalmente de otro atributo no primo (aula), el cual depende también de la clave de la relación.

Para eliminar los problemas que ocasiona en la relación *Matricula-3* la existencia de esta dependencia funcional, esta relación debe descomponerse en dos relaciones, quedando el esquema de la forma:

Esquema-4
Imparte (*asignatura#, curso)*
Alumno-2 *(dni, apellidos, nombre)*
Ubicación *(aula, lugar)*
Matricula-4 (***dni, asignatura#****, nota, **aula)***

Donde se observa que la relación *Matricula-4* se encuentra en FN3, puesto que todas las dependencias funcionales existentes son completas y entre atributos no primos y la clave de la relación.

Se ha generado una nueva relación, la relación *Ubicación,* para la cual el tributo aula es clave, y tiene un único atributo no primo, lugar, el cual depende funcionalmente de la clave, por lo que también se encuentra en FN3.

De nuevo, el atributo aula de la relación *Matricula-4* deberá definirse, en el esquema, como clave foránea de la relación *Ubicación,* de forma que para cualquier tupla de la relación *Matricula-4* este atributo sólo pueda tomar valores nulos o bien igual a valores existentes en alguna tupla de la relación *Ubicación.*

2.3.8 La forma normal de Boyce-Codd FNBC

La forma normal FNBC es conceptualmente distinta y mucho más sencilla, a la segunda y tercera forma normal. Las relaciones que satisfacen la restricción impuesta por la forma normal FNBC satisfacen la FN2 y FN3. La FNBC se basa en el concepto de *Determinante funcional,* y está soportada en las características de las claves candidatas de las relaciones.

Se denomina determinante funcional a uno o un conjunto de atributos de una relación R del cual depende funcionalmente de forma completa algún otro atributo de la misma relación.

Se puede expresar la forma normal de Boyce-Codd de la forma:

Una relación R satisface la forma normal de Boyce-Codd (FNBC) si, y sólo si, se encuentra en FN1, y cada determinante funcional es una clave candidata de la relación R.

Del análisis del *Esquema-4* se puede deducir que todas las relaciones se encuentran en FNBC, puesto que:

- Los únicos determinantes funcionales son las claves para cada una de las relaciones, puesto que en ninguna relación existen claves alternativas.

- En todas las relaciones, las únicas dependencias funcionales son las existentes entre los atributos no primos de cada relación y la clave de la misma.

Pero se pueden dar casos en los que una relación se encuentre en FN3 pero no satisfaga la FNBC, puesto que la FNBC es más restrictiva que la FN3.

Por ejemplo, consideremos de nuevo el *Esquema-1*, pero en este caso se va a realizar la consideración de que el agregado apellido, nombre, *asignatura#* es una clave candidata de la relación *Matricula*, quedando el esquema siguiente:

Esquema-5

Matricula-5 (*dni, asignatura#, apellidos, nombre, nota, curso, aula, ,lugar).*

En la cual existen dos determinantes funcionales, cada uno de ellos compuesto y formado por los agregados: *dni, asignatura#* y *apellidos, nombre, asignatura#,* para lo cual es necesario suponer que no existen dos alumnos con el mismo nombre completo matriculados en la misma asignatura.

Las dependencias existentes en la relación *Matricula-5* son:

Matricula-5.asignatura#	→	*Matricula-5.curso*
Matricula-5 (dni, asignatura#)	→	*Matricula-5.aula*
Matricula-5 (dni, asignatura#)	→	*Matricula-5.lugar*
Matricula-5 (dni, asignatura#)	→	*Matricula-5.nota*
Matricula-5 (apellidos, nombre, asignatura#)	→	*Matricula-5.aula*
Matricula-5 (apellidos, nombre, asignatura#)	→	*Matricula-5.lugar*
Matricula-5 (apellidos, nombre, asignatura#)	→	*Matricula-5.nota*
Matricula-5 .aula	→	*Matricula-5.lugar*
Matricula-5 (dni, asignatura#)	↔	*Matricula-5.(apellidos, nombre, asignatura#)*

Como se puede observar existen dependencias entre atributos que no son determinantes funcionales y que es necesario eliminar. Por tanto, la relación *Matricula-5* debe ser descompuesta, quedando el nuevo esquema de la forma:

Esquema -6
Imparte (*asignatura#, curso)*
Ubicación (*aula, lugar)*
Matricula-6 (*dni, asignatura#, apellidos, nombre, nota, aula)*

Se puede observar que, en base a la definición de la FNBC, en el esquema-6 todas las relaciones se encuentran en FNE, simplemente por la consideración de los determinantes funcionales y las dependencias que otros atributos de la relación mantienen con ellos. Se han eliminado directamente las dependencias funcionales no completas y las dependencias transitivas.

Ahora bien, las relaciones *Imparte y Ubicación* se encuentran además en FNBC, pues sólo existe un determinante funcional y un atributo dependiente del mismo de forma completa, pero la relación *Matricula-6* aunque se encuentra en la FN3 no satisface la FNBC.

Se puede observar que en esta relación existe una dependencia funcional que no ha sido considerada hasta el momento. Se trata de la dependencia funcional mutua existente entre el atributo *dni* y el agregado (*apellidos + nombre).*

$$Matricula\text{-}6.dni \leftrightarrow Matricula\text{-}6.(apellidos, nombre)$$

En las dos claves candidatas está presente un atributo común, el atributo *asignatura#,* el cual forma parte de los dos determinantes funcionales, por lo que ambas

claves se encuentran traslapadas[11]. La presencia de claves traslapadas puede ocasionar serios problemas en el mantenimiento de la información que hay que resolver.

La relación *Matricula-6* sería conveniente descomponerla en dos relaciones que satisfagan la FNBC, quedando el nuevo esquema de la forma:

Esquema-7
Imparte *(asignatura#, curso)*
Ubicación *(aula, lugar)*
Alumno-3 *(dni, apellidos, nombre)*
Matricula-7*(dni, asignatura#, nota, aula)*

Como se puede observar es similar al *Esquema-4* obtenido previamente pro aplicación de las formas normales FN1 a FN3. Si bien en este caso al considerar el agregado formado por los atributos *(apellidos, nombre)* como identificador alternativo de los alumnos, será necesario definir en el esquema a este agregado como clave alterna de la relación *Alumno-3, para* así hacer cumplir la dependencia funcional mutua existente entre este agregado y el atributo *dni.*

2.4 OTROS TIPOS DE DEPENDENCIAS

Se han considerado hasta este momento las dependencias que puedan estar presentes en una relación R y por las cuales los valores que toma un atributo R.x de esta relación determinan, para cada valor, el valor que puede tomar otro atributo R.y de la misma relación R; es decir, dependencias funcionales de la forma R.x $\rightarrow$ R.y.

Pero pueden considerarse otros tipos de dependencia funcionales en las que el valor de R.x no tenga que determinar un único valor de R.y sino un conjunto bien

[11] Dos claves candidatas se dice que están traslapadas si cada una de ellas está formada por dos o mas atributos y alguno de ellos es común a ambas.

definido de posibles valores que, a su vez, pueda depender o no de los valores que tome otro atributo R.z de la misma relación R.

La existencia de este tipo de dependencias ocasionan problemas de redundancia y de manipulación de la información que mantiene la relación R y, por tanto, es necesario aplicar alguna regla que elimine estas dependencias. Éste es el fin de las reglas de normalización FN4 y FN5.

Si bien estos nuevos tipos de dependencias pueden existir entre cualesquiera de los atributos de una relación R, es más usual que estén presentes cuando los atributos implicados forman parte de la clave primaria de la relación. La razón es bien sencilla, si son otros atributos los implicados, por aplicación de las reglas de normalización estudiadas hasta el momento, estas dependencias han debido ser eliminadas y, sin embargo, como se ha podido observar, las reglas FN1 a FNBC no tienen en cuenta las dependencias existentes entre los atributos que forman parte de la clave primaria de una relación cuando esta clave está formada por un agregado de datos.

2.4.1 La cuarta forma normal FN4

La regla de normalización FN4 está basada en la eliminación de las relaciones de un tipo especial de dependencias denominadas *Dependencias multivaluadas* (DMV), las cuales fueron consideradas por primera vez por *Fagin y Zamiolo* en 1977.

Dada una relación R, se dice que el atributo R.y $\in$ R depende de forma multivaluada de otro atributo R.x $\in$ R, o que R.x multidetermina a R.y, y se expresa de la forma R.x $\rightarrow\!\!\!\!\rightarrow$ R.y si, y sólo si, cada valor de R.x tiene asignado un conjunto bien definido de valores de R.y y este conjunto es independiente de cualquier valor que tome otro atributo R.z $\in$ R, el cual depende del valor de R.x.

Puede apreciarse en la definición de dependencia multivaluada que para que este tipo de dependencia esté presente en una relación, ésta, al menos, debe estar formada por tres atributos.

Las dependencias multivaluadas representan la independencia existente entre dos conjuntos R.y y R.z, la cual está correlacionada por la dependencia que tiene cada uno de estos conjuntos con el conjunto R.x del cual dependen ambos de forma multivaluada.

Así, es fácil demostrar que para que en una relación R esté presente la dependencia multivaluada R.x $\rightarrow\!\!\!\rightarrow$ R.y es condición necesaria que además esté presente la dependencia multivaluada R.x $\rightarrow\!\!\!\rightarrow$ R.z. Las dependencias multivaluadas nunca están presentes solas en una relación, sino que siempre están presentes en parejas, por lo que es usual representarlas de la forma: R.x $\rightarrow\!\!\!\rightarrow$ R.y/R.z, significando que: para cada valor de R.x no existe un único valor de R.y asociado (se trataría de una dependencia funcional), sino un conjunto bien definido de valores independiente del valor de R.z para ese valor de R.x que sólo depende del valor de R.x.

Se puede ahora definir la FN4 de la forma siguiente:

Una relación R está en FN4 si, y sólo si, siempre que exista una dependencia multivaluada en R de la forma R.x $\rightarrow\!\!\!\rightarrow$ R.y, todos los demás atributos de R son funcionalmente dependientes de R.x.

Es decir, para que una relación satisfaga la FN4, las únicas dependencias admitidas son las dependencias funcionales de la forma R.x $\rightarrow\!\!\!\rightarrow$ R.z donde R.x es un determinante funcional de R, y R.z es cualquier otro atributo de R.

Hay que observar que las dependencias multivaluadas son una generalización de las dependencias funcionales o, lo que es lo mismo, una dependencia funcional es un caso

particular de una dependencia multivaluada en la que se considera en lugar de *"un conjunto bien definido"* de valores a *"un solo valor"*.

2.4.2 La quinta forma normal FN5

Hay ocasiones en las que la descomposición de una relación en dos nuevas relaciones no puede realizarse sin pérdida de información y sin embargo, es necesario realizar un proceso de normalización de una relación dada, pues están presentes una serie de dependencias que ocasionan, al menos, problemas de redundancias superfluas en esa relación.

En una relación R puede estar presente un caso especial de dependencias llamadas *Dependencias de Reunión*, las cuales son la base de la aplicación de la FN5. Las dependencias de reunión son difíciles de detectar, por tanto, la aplicación de esta regla de normalización no es muy usada, además de cuestionarse la mejora que puede introducir la aplicación de la misma si se tiene en cuenta la complejidad que añade al nuevo esquema relacional por el número de relaciones nuevas que introduce.

Dada una relación R de esquema R(a1, a2,...,an), se dice que existe una dependencia de reunión si, y sólo si, la relación R puede ser construida a partir de la reunión natural de las relaciones R1, R2,...,Rn obtenidas por la proyección de R sobre los atributos a1, a2,...,an, respectivamente.

Puede ser definida en estos momentos la regla de normalización FN5 de la forma siguiente:

Una relación R satisface la FN5, también denominada forma normal de proyección (FNPR) si, y sólo si, toda dependencia de reunión en R está implicada por las claves candidatas entre si y no por cualquier otro atributo de R forme o no parte de las claves candidatas.

2.5 EL ALGEBRA RELACIONAL. SQL

Al mismo tiempo que E.F. Codd propuso el modelo relacional como una nueva forma de representar la información para su tratamiento, se propusieron dos planteamientos o visiones diferentes, desde el punto de vista matemático, para el acceso y manipulación de la información representada bajo este modelo: la visión conjuntivista y la visión predicativa de los datos.

En base a cada una de estas visiones se han ido desarrollando lenguajes que están basados, respectivamente, en el álgebra y en el cálculo de predicados.

Codd, cuando propuso el modelo relacional, sentó las bases de un lenguaje algebraico denominado SQUARE para la manipulación de los datos. Este lenguaje fue mejorándose y ampliándose, hasta que se denominó posteriormente SEQUEL y, por último, en 1976, SQL (debido al sonido de la pronunciación del nombre primitivo).

Durante la década de los setenta tanto el modelo relacional como el lenguaje propuesto por Codd estuvieron siendo validados, periodo en el cual una serie de firmas empezaron a comercializar productos basados en los mismos. No fue hasta los primeros años de la década de los ochenta cuando IBM comercializó su primer producto relacional, el SQL / DS y el DB2, más adelante. Y fue en esta década cuando ANSI publicó los primeros estándares relacionales.

Hoy en día existe en el mercado un gran número de Sistemas Gestores de Bases de datos (SGBD) relacionales cuyo lenguaje de manipulación de datos es el SQL con más o menos mejoras introducidas particularmente por los fabricantes. SQL se ha convertido en un estándar gracias a las mejoras que ha experimentado en los últimos años que le han conferido una gran potencia y sencillez de uso y aprendizaje.

2.5.1 El lenguaje relacional

SQL es el lenguaje estandarizado y de uso universal utilizado por los SGBD basados en el álgebra relacional. Como cualquier lenguaje de un SGBD, SQL cubre tres aspectos bien diferenciados: *la descripción, la manipulación y el control y seguridad de los datos*. Para cada uno de estos aspectos SQL cuenta con un conjunto de verbos que realizan funciones específicas con la información.

Si bien SQL es un lenguaje interactivo, las sentencias SQL pueden ser incorporadas a las sentencias de programas realizadas con lenguajes *host* como COBOL y C.

2.5.2 Definición de esquemas relacionales

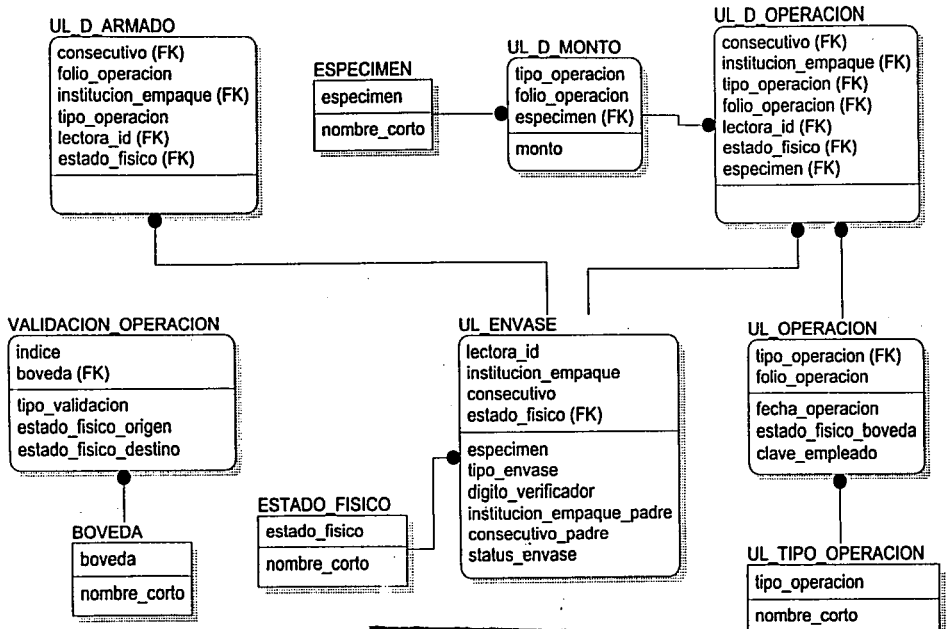
Un esquema de una base de datos relacional está formado, básicamente, por la definición de un conjunto de tablas (relaciones).

Cada tabla debe tener un nombre único en el esquema y estar definida en base a la especificación de un conjunto de atributos, las columnas de la tabla, que representan a propiedades del objeto del mundo real representado por la tabla.

Cada atributo debe tener un nombre único para una tabla y estará definido en un dominio de datos preestablecido. De entre todos los atributos que forman la definición de una tabla, uno o un conjunto de ellos se elegirán como clave principal de la misma.

Opcionalmente, en la definición de una tabla se especificarán las claves foráneas; es decir, aquellos atributos de la tabla que están definidos en el mismo dominio y representan la misma propiedad que la clave principal de alguna otra tabla que forma parte del esquema de la base de datos. A continuación se presenta el esquema de la base de datos relacional:

DIAGRAMA DE LA BASE DE DATOS



SQL utiliza el verbo **CREATE TABLE** para la definición de las tablas del esquema relacional. El formato general es el siguiente:

```
CREATE TABLE  Nombre Tabla
(nombre-atributo-1:      DOMINIO (opción),
nombre-atributo-2:       DOMINIO (opción),
.....                    .....
nombre-atributo-n:       DOMINIO (opción),
PRIMARY KEY              (lista-atributos),
FOREIGN KEY              (lista-atributos),
REFERENCES               NombreTabla(Clave-principal) );
```

Para cada atributo es necesario especificar un DOMINIO en el cual pueda tomar valores esa propiedad del objeto. La definición de un dominio comprende:

1. La definición de un tipo de datos. Los tipos de datos en los cuales se especifican los dominios están predeterminados por los SGBD.

2. En la definición de los atributos se pueden especificar algunas restricciones u opciones para ellos. Estas restricciones se pueden especificar directamente en la definición de la columna de la tabla, o más fácilmente haciendo uso de las cláusulas CONSTRAINT.

Un CONSTRAINT es un cuerpo de definición en el esquema que permite definir las restricciones de integridad de la información definida en el mismo. La sintaxis general es la siguiente:

CONSTRAINT nombre-constraint
Cuerpo-constraint,

donde el nombre-constraint debe ser único en el esquema, y en el cuerpo-constraint pueden aparecer varias cláusulas.

3. La definición de una tabla del esquema relacional puede ser modificada haciendo uso del verbo ALTER TABLE, cuya sintaxis es la siguiente:

ALTER TABLE nombre_tabla
cuerpo_modificación;

pudiendo realizarse modificaciones sobre los siguientes objetos del esquema, entre otros:

- Añadir una columna o atributo
  ADD (nombre atributo DOMINIO (opción)).

- Añadir un CONSTRAINT
  ADD CONSTRAINT (definición del constraint).

- Activar, desactivar o borrar un CONSTRAINT
  ENABLE, DISABLE, DROP CONSTRAINT

En la definición de los esquemas relacionales son muy importantes los índices. Un índice es un objeto de la base de datos que contiene una entrada para cada valor que aparece en una columna (o conjunto de ellas) de una tabla, proporcionando accesos rápidos y directos a las tuplas de la tabla indexada.

Sin embargo, un índice puede aminorar los desempeños de las operaciones de inserción, borrado y modificación que afectan a las columnas por las cuales se ha generado el índice debido a que se debe gestionar tanto los índices como la información de la tabla indexada.

Así, cuando se insertan inicialmente valores de una tabla, es generalmente más rápido crear una tabla, insertar las tuplas y posteriormente crear el índice. Si se crea el

índice antes de la inserción de las tuplas. ORACLE, por ejemplo, debe actualizar el mismo tras cada inserción.

Para crear un índice se utiliza la siguiente sintaxis:

CREATE INDEX nombre-índice **ON**
Nombre-tabla (lista-campos);

Otro elemento de interés y necesario para la definición de la base de datos son las VISTAS. Una vista VIEW es una tabla lógica que permite acceder a la información de otras tablas o vistas.[12]

Una vista permite además proporcionar un nivel adicional de seguridad a las tablas del esquema, restringiendo el acceso a un conjunto predeterminado de tuplas y/o columnas de las tablas base en las que se apoya.

Por otra parte, una vista permite ocultar el nivel de complejidad y de los datos, proporcionando una visión lógica externa de los mismos más simple y próxima al usuario.

Para crear una vista se utiliza la siguiente sintaxis

CREATE VIEW nombre-vista **AS.**
sentencia SELECT;

donde sentencia SELECT es una expresión bien formada del DML, cuya sintaxis se describirá mas adelante.

[12] El término lógica quiere decir que una vista no contiene información por sí misma, sino que su información está basada en la que contiene otras tablas a las que se les denomina tablas base.

2.5.3 Manipulación de la información

La manipulación de la información de una base de datos comprende operaciones de inserción, modificación, borrado y consulta de los datos almacenados en la misma. SQL incorpora verbos para cada una de estas operaciones, con una sintaxis clara y sencilla, a la vez que potente, puesto que permite la anidación de sentencias en las que aparecen más de un verbo SQL que realizan la mismo o distinta operación.

2.5.4 Consulta de la información

Consultar información de una base de datos supone el acceso seleccionado de los datos existentes para un subconjunto de los atributos que formar parte del esquema de la misma, obteniendo como resultado una nueva tabla, un valor o conjunto de valores. SQL aporta una gran potencia a la vez que sencillez, en los procesos de acceso a la información existente en la base de datos. El formato general de esta operación es el siguiente:

```
SELECT    [DISTINCT / ALL] lista-atributos-1
          [INTO] cursor
          FROM lista-tablas
               WHERE Condición-1
          GROUP BY lista-atributos-2
               HAVING Condición-2
          ORDER BY lista-atributos [ASC  DESC];
```

La *lista-atributos-1* representa a aquel conjunto de atributos sobre los cuales se desea realizar una operación de proyección. Si no se quiere realizar una proyección, este parámetro se sustituye por el modificador *, que representa a todos aquellos atributos de todas las tablas que intervienen en la operación de consulta.

Las palabras DISTINCT acompañando a algún atributo, indica que no podrán existir valores duplicados para ese tributo en las tuplas de la relación resultante. Por

defecto, en el resultado de una consulta pueden existir valores duplicados para los atributos seleccionados (ALL). En el apéndice A se muestra toda las funciones con las cuales se manipula la información de la base de datos y podemos encontrar varios ejemplos de esta y otras sentencias como la de inserción, actualización, borrado, etc.

El verbo INTO permite asignar una variable definida por el usuario para el control y acceso de las tuplas resultado de l operación de consulta.

Las tablas que van a ser manejadas en la operación de consulta son especificadas mediante la cláusula FROM, detallando cada una de las tablas separadas por comas.

Si se especifica la cláusula WHERE la operación de consulta realiza una selección de las tuplas resultante en base a la expresión específica en la *condición-1*. Esta expresión puede estar formada por otra instrucción SELECT de forma que se pueden anidar las consultas, teniendo en cuenta que son las consultas más internas las que se realizan en primer lugar.

Mediante las cláusulas GROUP BY pueden construirse grupos de tuplas en base a la *lista-atributos-2*. La agrupación permite la recuperación de una sol tupla por cada uno de los grupos construidos en base a cada uno de los valores de los atributos especificados en la *lista-atributos-2*.

La cláusula HAVING permite restringir el conjunto de grupos que pueden ser formados por la cláusula GROUP BY. Como se puede apreciar tiene la misma función que la cláusula WHERE.

Por último, la cláusula ORDER BY tiene como función proporcionar un orden de las tuplas obtenidas como resultado de la consulta. En esta cláusula se indican los atributos que van a determinar el orden de las listas. Si el orden será ascendente *(ASC)* o descendente (*DESC).*

2.5.5 Inserción de la información

La inserción de la información consiste en la incorporación de nuevas tuplas a tablas de la base de datos. El formato general de esta operación en SQL es:

INSERT [INTO] NombreTabla [lista-atributos]
VALUES (lista-valores);

La lista de valores puede introducirse directamente en la expresión o bien puede obtenerse a partir de la información existente en la base de datos mediante la inclusión de una consulta haciendo uso del verbo SELECT.

Si los atributos no se especifican en la cláusula INSERT se consideran, por defecto, todos los atributos de la tabla. Y si no se asigna un valor a un determinado atributo, por defecto, éste toma el valor nulo. Existen varias funciones en el apéndice A que usan esta sentencia.

2.5.6 Modificación de la información

La modificación de la información consiste en la actualización de los valores de los atributos para una o varias tuplas de un tabla. El formato general de esta operación en SQL es:

UPDATE NombreTabla
SET nombre-atributo-1 = valor
SET nombre-atributo-2 = valor
.....
SET nombre-atributo-n = valor
WHERE *Condición;*

Mediante esta instrucción son modificados los valores de los atributos especificados de la tabla al valor indicado en la cláusula. Este valor puede ser obtenido, al igual que con el verbo INSERT, de los valores existentes en la base de datos mediante la inclusión de una consulta a la misma. Opcionalmente, puede incluirse la cláusula WHERE, la cual permite que sólo se modifiquen los valores de los atributos cuando se cumpla la condición especificada en esta cláusula.

2.5.7 Borrado de la información

El borrado de la información consiste en la eliminación de una o varias tuplas de una tabla que satisfagan o no una condición. La tabla no es borrada del esquema de la base de datos, únicamente son borradas las tuplas de la misma. Para la eliminación de la tabla es necesario utilizar un verbo del DDL de SQL, éste es el verbo DROP[14]. El formato general de esta operación en SQL es:

DELETE [FROM] NombreTabla
WHERE Condición;

En el apéndice A se pueden encontrar varios ejemplos de estas sentencia donde se manipula la información

[14] Este verbo, aplicado con sus diferentes operandos es utilizado por Sybase para la eliminación de cualquier objeto del esquema.

CAPITULO 3

PLATAFORMA PALM

3.1 EL PROCESADOR PRINCIPAL

El procesador principal en los modelos actuales de Palm es el Motorola MC68328 DragonBall, el cual es una versión de bajo consumo del procesador MC68000, es un chip popular usado en computadoras como la Macintosh original y la Amiga. El 68328 y el 68000 comparten el mismo juego de instrucciones, pero el procesador DragonBall está diseñando específicamente para el mercado de dispositivos portátiles.

Comparado con el procesador de una computadora de escritorio, el DragonBall es lento, porque un procesador mas rápido requiere mas poder para funcionar, y el bajo consumo de poder es la característica principal en el diseño de la plataforma Palm. De hecho, Palm computing recomienda que si se necesita hacer algún procesamiento intenso, éste se haga en una computadora de escritorio para que la Palm no gaste tanto poder y que la respuesta en la Palm sea siempre eficaz.

3.2 LA PANTALLA

La pantalla en una Palm es actualmente de 160 pixeles[1] de ancho por 160 pixeles de ancho, y las aplicaciones deben tener una interfaz que cumpla con esta restricción.

Un digitalizador hace que la pantalla sea sensible al toque. El sistema operativo transforma movimientos de la pluma y los toques[2] (taps) en la pantalla a eventos de alto nivel que una aplicación puede procesar.

[1] Unidad con la que se mide la resolución de un monitor.
[2] Al toque en la pantalla de una Palm con un stylus el cual es el aditamento que tiene la Palm para que se le introduzca información, se le llama tap y es similar a lo que sería un clic de un mouse para una PC.

3.3 MODOS DE EJECUCIÓN

Aparte de su característica de bajo consumo, existe otra característica muy importante en los dispositivos Palm que es su manejo de la energía mediante los diferentes modos de ejecución. Un dispositivo Palm tiene tres modos de ejecución:

Sleep mode. Ocurre cuando no ha habido actividad del usuario en el dispositivo por un cierto número de minutos. La cantidad de minutos utilizada la mayoría de las veces son dos minutos, pero se puede cambiar el periodo de espera a uno o tres minutos usando la aplicación Preferencias. También se pasa a éste modo cuando se apaga el dispositivo usando el botón de poder. En este modo el procesador, la pantalla y la mayoría de las partes de hardware no reciben nada de energía excepto los chips de memoria, el reloj de tiempo real y algunos circuitos de bajo nivel. Presionando el botón de poder se restaura la energía al procesador, la pantalla y las otras partes de hardware. Pero también existe otra forma de que se vuelva a energizar toda la Palm por medio del reloj, ya que se vuelve a energizar en un predeterminado tiempo.

Doze mode. Ocurre cuando una aplicación esta esperando a que el usuario dé una entrada a la Palm, ya sea que escriba algo o que de un toque en la pantalla, etc.

Running mode. Ocurre cuando el procesador esta ejecutando instrucciones, y es también el modo en el que se consume la mayor cantidad de energía, y es por esto que el dispositivo regresa al modo anterior, o sea al doze mode, tan pronto como sea posible para conservar la mayor cantidad de energía.

En realidad la fuente de poder, o sea, la alimentación de energía nunca se apaga, a menos que se quiten las baterías y se dejen que los capacitores[3] internos se descarguen

[3] También llamados condensadores

completamente. Estos capacitores son los que mantienen a la Palm cargada mientras se cambias las baterías.

La aplicación activa esta simplemente en un estado de suspensión cuando la Palm está en el modo *sleep mode*, lista para regresar a donde se quedó cuando la energía se reactivada y alimente el procesador.

3.4 EL SISTEMA OPERATIVO

El sistema operativo Palm (Palm OS) es el que provee el ambiente para la ejecución de la aplicación y consiste de dos partes: el kernel y los manejadores o controladores.

3.4.1 El kernel

El kernel es el núcleo del sistema operativo Palm (PalmOS). Este provee la interfaz entre el hardware y el resto del sistema operativo, y también maneja los múltiples hilos (Threads) de ejecución. También es el encargado de atrapar las interrupciones cuando el usuario toca la pantalla con la pluma[4]. El kernel en la actualidad es una licencia de Kadak Products Ltd.

Aunque el sistema operativo Palm usa varios Threads internamente, las aplicaciones no pueden crear o usar algunos otros Threads que no sean los que están destinados para ellas. Y solamente se pueden crear aplicaciones que utilicen únicamente un solo Thread, esto significa que se tienen que realizar largas operaciones en pequeñas cantidades para evitar la apariencia de que el dispositivo esta suspendido.

[4] El objeto con el que se hacen los toques en la pantalla en un dispositivo Palm comúnmente se le llama pluma aunque su nombre correcto es stylus.

Aunque las aplicaciones son de un solo Thread y solo una aplicación aparece activa al usuario, una segunda copia de la aplicación activa algunas veces corre en otro Thread pero en un ambiente mas limitado.

3.4.2 Los manejadores

Aparte del kernel, el sistema operativo Palm consiste de una serie de manejadores los cuales son una colección de API's agrupados por función. Las aplicaciones usan los manejadores para obtener recursos del sistema e interactuar con el usuario. Los manejadores pueden ser agrupados como sigue:

Los manejadores de la interfaz de usuario que controlan toda la interacción con el usuario. Esto incluye funciones generales para dibujar en la pantalla así como funciones relacionadas a elementos específicos de la interfaz de usuario así como campos o listas.

Los manejadores del sistema que controlan todo lo que no tiene que ver con la interacción con el usuario, incluyendo el acceso con los contadores de tiempo y a la cola de eventos del sistema.

Los manejadores de la memoria que controlan el acceso a la memoria y el manejo de las base de datos del sistema operativo Palm y de los recursos de estas.

Los manejadores de comunicación que controlan la comunicación entre el dispositivo Palm y el mundo externo usando el puerto serial o el puerto infrarrojo.

3.4.3 Las versiones del sistema operativo Palm

Como los dispositivos Palm han evolucionado, Palm computing ha sacado nuevas versiones del sistema operativo. El sistema operativo Palm 4.0 es el que se usa en las Palm m500 o m505, la versión 3.2 es el usado en las Palm VII. La Palm IIIx y la palm V

usan el sistema operativo 3.1, mientras que la Palm III usa la versión 3.0. El modelo original que es la Pilot 1000 y el Pilot 5000 usaban el sistema operativo 1.0. Los modelos PalmPilot Personal y el PalmPilot Professional usan el sistema operativo 2.0. Todos los modelos Palm anteriores al Palm III pueden ser actualizados a la versión de sistema operativo 3.2 ordenando una nueva tarjeta de memoria a Palm Computing. La actualización también agrega mas memoria RAM y agrega algunas nuevas capacidades como la transferencia por el puerto infrarrojo que no lo tienen esos modelos. Los modelos mas recientes desde la Palm III pueden ser actualizados simplemente bajando una versión mas reciente del sistema operativo de Internet a la memoria flash del dispositivo.

Palm computing ha hecho su mejor esfuerzo para que cada versión del sistema operativo Palm sea compatible. Las aplicaciones escritas para el sistema operativo versión 1.0 trabajará igual de bien en dispositivos que tengan las versiones de sistema operativo 2.0 o 3.0. Aplicaciones para sistema operativo Palm 2.0 trabajan perfectamente en el sistema operativo 3.0 pero pueden no correr bien en algún dispositivo que tenga sistema operativo 1.0.

En algunos casos una aplicación que fue desarrollada en un sistema operativo mas antiguo no corre adecuadamente en una versión mas reciente porque la aplicación fue hecha considerando algunos errores de esa versión de sistema operativo y la versión mas reciente ya esta corregida.

Ahora que si la aplicación fue desarrollada en una de las versiones mas recientes del sistema operativo, pero esta aplicación no utiliza las nuevas características de esa nueva versión, entonces, si podrá ejecutarse correctamente en una versión mas antigua. Pero, entonces, surge la siguiente pregunta ¿Cómo se puede saber si una aplicación esta utilizando una de las características específicas de la nueva versión?. La manera mas segura de saberlo es ejecutando la aplicación en la versión 1.0 o 2.0 del sistema operativo o usando el emulador de sistema operativo (del cual se hablará mas adelante). O también

se puede compilar la aplicación utilizando una versión mas antigua del software del kit de desarrollo del sistema operativo. El problema con esta última opción y con la de ejecutar la aplicación en una versión mas antigua es que resulta difícil poder conseguir la versión mas antigua del sistema operativo o del software del kit de desarrollo. Entonces, podemos decir que lo mejor sería que se supiera cuales son las características que se agregaron a la nueva versión del sistema operativo.

Por ejemplo, estas son algunas características que se agregaron a la versión 3.0 en comparación con la versión 2.0:

- Se agregó mas memoria dinámica, de 64K a 96K.
- Se agregaron nuevas funciones para el soporte de transferencia por el puerto infrarrojo que lo trae la Palm III.
- Se agregó soporte para archivos de sonido MIDI.
- Se agregó la habilidad para crear dinámicamente objetos de interfaz de usuario en tiempo de ejecución, en vez de que solamente sean cargados desde definiciones estáticas.
- Se agregó un archivo simple API para el manejo de largos bloques de datos.
- Se agregó un manejador de progreso para desplegar diálogos durante operaciones largas.

3.5 MEMORIA

A diferencia de las computadoras de escritorio, que pueden usar memoria virtual para incrementar la cantidad de memoria disponible para una aplicación, los chips de memoria que vienen con el dispositivo Palm son todo lo que se tiene para trabajar. Es por esto que la memoria es algo muy valioso en la plataforma Palm.

3.5.1 ROM y RAM

Los dispositivos Palm vienen con diferentes cantidades de memoria, del tipo de solo lectura (ROM) y del tipo de lectura y escritura (RAM). La memoria ROM es usada para almacenar el sistema operativo, las aplicaciones que vienen con la Palm, y algunos datos necesarios para esas aplicaciones. Los modelos mas recientes así como la Palm III utilizan memoria flash en vez de ROM, así que el sistema operativo puede ser actualizado sin remplazar chips de memoria.

La memoria RAM es donde todo lo demás es almacenado, las aplicaciones que se han instalado al dispositivo, los datos para esas aplicaciones, los valores de configuración del usuario y varias áreas del sistema de datos. Los dispositivos Palm originales vienen con muy poca memoria, 512K de memoria ROM y 128K de memoria RAM, pero la Palm III viene con 2 MB de memoria RAM.

Hay que recordar que a diferencia de las computadoras de escritorio, las Palm nunca se apagan en realidad, ya que cuando se oprime el botón de encendido el dispositivo únicamente pasa al modo de sleep mode (explicado con anterioridad), por lo que cualquier cosa que se guarde en la memoria RAM permanece ahí hasta que sea explícitamente borrado.

3.5.2 Tarjetas y heaps

Las memorias ROM Y RAM son empacadas juntas en lo que es conocido como tarjeta de memoria. Los dispositivos Palm pueden soportar múltiples tarjetas, pero solo una tarjeta es instalada en cada dispositivo. El número de tarjeta de memoria (que siempre comienza en 0) es un parámetro requerido en algunas llamadas al sistema.

La memoria en una tarjeta es dividida en heaps. La memoria ROM es considerada un solo heap. La memoria RAM consiste de al menos dos heaps, el primero de los cuales es el conocido como heap dinámico, mientras el resto son llamados heaps de almacenamiento.

El heap dinámico es utilizado por el sistema operativo y por las aplicaciones y entre otras cosas el heap dinámico almacena los siguiente:

- Datos del sistema operativo, estructuras de la interfaz de usuario y varios buffers
- La pila de la aplicación donde las direcciones de las variables locales y de las funciones son almacenadas
- Las variables globales y estáticas de la aplicación, que son conocidas como el bloque de datos global.
- El acceso dinámico a la memoria usado por una aplicación.

El tamaño del heap dinámico pude ser muy pequeño. En dispositivos con sistema operativo Palm 1.0 es de tan solo 32K. En el sistema operativo 2.0 puede ser también de 32K (PalmPilot Personal) o 64K (PalmPilot Professional), pero si existe la comunicación por medio de TCP/IP en el dispositivo, éste usa entonces 32K de memoria. En

dispositivos con sistema operativo versión 3.0 el tamaño del heap dinámico es de 96K, pero si utiliza TCP/IP y ya que esta requiere tan solo de 32K, entonces, le dejan por lo menos 64K para uso general.

Existen dos importantes diferencias entre el heap dinámico y los heaps de almacenamiento. Primero, el contenido del heap dinámico es limpiado cuando el dispositivo es reseteado. El heap dinámico no es usado para guardar información por largos periodos de tiempo. El contenido de los heaps de almacenamiento es mantenido excepto en los casos mas extremos, como cuando las baterías son removidas del dispositivo por mas de un minuto o cuando un reset duro es realizado.

Segundo, los heaps de almacenamiento son protegidos contra escritura para prevenir que una aplicación haga alguna escritura sobre otra aplicación que utilice almacenamiento permanente, mientras el contenido del heap dinámico no es protegido contra escritura, haciendo posible que algún mal comportamiento de alguna aplicación provoque que el dispositivo se trabe. En el sistema operativo versión 1.0 y 2.0 hay múltiples heaps de almacenamiento y cada uno no tiene mas de 64K de tamaño.

3.6 RESETEANDO EL DISPOSITIVO

En algún punto, una aplicación que se está desarrollando va corromper la memoria y a provocar que se trabe el dispositivo. Presionando el botón de encendido no sirve para sacar de ese estado a la Palm, ya que lo único que hace este botón es cambiar de estado al dispositivo. En casos como éste será necesario que se resetee el dispositivo manualmente.

Hay tres tipos de reset, pero, antes de explicar cada uno de ellos hay que entender que primero será necesario realizar una sincronización para que queden respaldados todos los datos del dispositivo. Los tipos de reset son los siguientes:

3.6.1 Reset suave

La forma mas amable y básica de resetear la Palm es conocida como reset suave. Un reset suave limpia el contenido del heap dinámico y reinicia el sistema operativo. Los heaps de almacenamiento, y por consiguiente todas las aplicaciones y sus datos, son intocables. Todas las aplicaciones son avisadas de que ha habido un reset.

Se realiza un reset suave usando un clip para sujetar papel y con éste se presiona suavemente en el orificio del reset en la parte trasera del dispositivo. En el modelo Palm III la parte de arriba de la pluma, la cual es llamada "stylus", se desenrosca y tiene algo que puede suplir al clip.

3.6.2 Reset suave modificado

Un paso mas allá del reset suave es el reset suave modificado, el cual también limpia el contenido del heap dinámico y reinicia al sistema operativo, pero a diferencia del reset suave, en este reset no se avisa a las aplicaciones que ha ocurrido un reset. Esto permite que el dispositivo y la aplicación vuelva a ejecutarse si es que ésta se traba mientras se procesa la notificación enviada por un reset suave. Los parches de sistema, los cuales son la primera forma de reparar los errores del sistemas operativo, no son cargados en el caso de que estos sean la causa de que la aplicación se haya trabado.

Se realiza un reset suave modificado presionando y manteniendo presionado el botón de desplazamiento hacia arriba y entonces, se realiza un reset suave con un clip para sujetar papeles.

Este tipo de reset se utiliza cuando el sistema se traba después de haber hecho un reset suave y se realiza inmediatamente después de éste para evitar que el sistema se trabe por completa y se tenga que realizar un reset duro.

3.6.3 Reset duro

La última forma de hacer un reset es el reset duro. Un reset duro borra el contenido del heap dinámico y todos los heaps de almacenamiento y después reinicia el sistema operativo. Cualquier aplicación que se haya instalado y todos sus datos se pierden (el dispositivo esta como cuando lo sacaron de su caja por primera vez). Realizando respaldos periódicamente va a permitir que algún momento dado se haga un reset duro y se pueda recuperar la información.

Se realiza un reset duro presionando y manteniendo presionado el botón de encendido de la Palm y presionando con un clip para sujetar papeles en el orificio destinado para el reset que por lo general se encuentra en la parte trasera del dispositivo, y el reset se realiza en el momento en el que se quita el clip del orificio y se suelta el botón de encendido que se estaba presionando. Después la Palm pregunta si se está seguro de realizar el reset duro.

El reset duro no se necesita hacerlo muy seguido, solo cuando se quiera limpiar algo y no cuando la Palm esté trabada y se quiera recuperar, ya que para esto existe el reset suave o el reset suave modificado. Una de las ocasiones en que será necesario utilizar el reset duro es cuando por alguna razón es necesario cambiar el nombre de usuario del dispositivo. La primera sincronización utilizando el programa HotSync[5] después de que se ha realizado un reset duro permite elegir cual nombre de usuario utilizará esa Palm. Pero si únicamente se tiene un solo dispositivo, el reset duro podría ser utilizado para establecer otro usuario y así probar una aplicación con varios usuarios. Nada mas que antes de realizar un reset duro hay que asegurarse que se ha hecho un respaldo de todas las aplicaciones de la Palm.

[5] En este proyecto de tesis no se utiliza el programa HotSync para realizar la sincronización sino que es por medio del servidor Mobilink.

3.7 SINCRONIZACION DE DATOS

La llave del éxito de la plataforma Palm es su capacidad para sincronizar datos entre un dispositivo como la Palm y una computadora de escritorio. La sincronización permite al usuario introducir datos una sola vez y que estos aparezcan en ambas máquinas, en el dispositivo y en la computadora de escritorio, sin importar en cual fueron introducidos. Y en efecto, ambas máquinas, la Palm y la computadora de escritorio son extensiones una de la otra.

3.7.1 La sincronización

Generalmente hablando, la sincronización de datos se refiere al intercambio y transformación de datos entre dos aplicaciones que mantienen separados su almacenamiento de datos. Las aplicaciones comparten datos aunque posiblemente estén almacenados en diferentes lugares y el proceso de sincronización guarda una copia de cada aplicación y mantiene un respaldo de los datos que siempre es actualizado.

En la plataforma Palm, la sincronización de datos ocurre entre una aplicación que corre en un dispositivo Palm y una aplicación en una computadora de escritorio llamada conducto (conduit).

La sincronización de datos no es una nueva tecnología. La mayoría de la compañías utilizan comúnmente la sincronización de datos en un solo sentido, algunas veces es llamado como replicación de datos, para bases de datos espejo, de tal manera que si un servidor de bases de datos se corrompe, el otro esta listo para tomar su lugar con un mínimo de pérdida de datos. Pero hay quienes argumentarían que la replicación de datos no es lo mismo que la sincronización de datos y se reservarían éste termino para el intercambio de información en ambos sentidos.

La sincronización de datos es también usada en automatización en ventas donde se utilizan computadoras portátiles (LapTop) y que después se tienen que sincronizar con las bases de datos principales del corporativo. Esto permite que se actualice información de algunos clientes por medio de las computadoras portátiles mientras esta desconectado de la red corporativa. Todos los cambios realizados en una computadora portátil después son transferidos y aplicados en la base de datos central en la siguiente ocasión en que la computadora personal se conecte a ka red. Desde que la sincronización de datos es utilizada de manera común en ambos sentidos, cualquier cambio realizado en la base de datos central es también transferido hacia las computadoras portátiles que lo requieran.

Hay que hacer notar que la sincronización de datos se refiere a transformación de datos así como intercambio de datos. No hay nada que diga que la sincronización entre dos fuentes de datos tienen que ser imágenes espejo la una de la otra. Una fuente de datos puede contener un subconjunto de los datos en una segunda fuente de datos, esto es lo que ocurre principalmente entre las aplicaciones Palm, por que el dispositivo Palm simplemente no tiene la memoria suficiente para guardar grandes cantidades de datos. Las dos fuentes de datos pueden también guardarlos en formatos completamente diferentes. La sincronización de la Libreta de Direcciones de la Palm, con la lista de direcciones que se guarda en la computadora de escritorio puede estar guardada por ejemplo en Lotus o en Outlook, y para esto hay que transformar los datos de la Palm en formatos de Lotus o Outlook.

Se puede pensar que la sincronización de datos en la plataforma Palm es simple, primero creando la base de datos, después estableciendo algunas banderas, y la base de datos será respaldada de forma automática en la siguiente sincronización pero en realidad no es así, ya que la sincronización es más que solo respaldar los datos.

La sincronización de datos con un dispositivo Palm es realizado por un conducto (o conduit). El conducto es una aplicación que se ejecuta en la computadora de escritorio y no en el dispositivo Palm y es invocado automáticamente como parte del proceso

HotSync. El conducto recibe, los datos modificados de la Palm y la actualiza con los datos nuevos. El conducto es una puerta hacia otra aplicación, que se comunica directamente con la aplicación (la cual puede ser el servidor de la base de datos) o que sabe como leer o escribir los archivos de datos de la aplicación. Se pueden comprar conductos para sincronizar datos de aplicaciones construidas en una Palm, como la libreta de direcciones o la agenda.

Cuando se escriben aplicaciones personalizadas en una Palm es poco probable que se encuentren conductos apropiados para hacer la sincronización de datos. El conducto tiene que saber como es el diseño de la base de datos y como comunicarse con la aplicación que corre en la computadora de escritorio y como convertir los datos al formato apropiado. Esto es el primer reto para hacer la sincronización de datos en la plataforma Palm: a menos que se pueda encontrar una alternativa apropiada, se tendrá que escribir un conducto personalizado. Esto significa que se habría de conocer una nueva herramienta o quizá aún un nuevo lenguaje de programación, que podría ser Java o C/C++.

Cuando se va a sincronizar una base de datos relacional, se tiene que conocer como es la interfase con la base de datos, como extraer los datos necesarios, y como asegurar la integridad de los datos.

3.7.2 Retos en la Sincronización

Entender la estructura de la base de datos relacional y de los datos que la aplicación necesita para extraerlos de la base de datos consolidada. Esto es necesario para cualquier aplicación con una base de datos, independientemente si es una aplicación para una Palm o no. Con la aplicación agenda, por ejemplo, la estructura de la base de datos es suficientemente sencilla, y consiste en un conjunto de tablas que contienen empleados, clientes, proyectos y entradas de tiempos. Si la base de datos no tiene la información que se requiere, se necesitará definir nuevas tablas

o modificar tablas existentes para guardar esa información. Si no se esta usando una base de datos relacional, como fuente de datos se necesita entender como esta estructura la fuente de datos.

Se necesita conocer como es la interfaz con la base de datos. El conducto necesita comunicarse con el servidor de la base de datos, para leer y escribir en la base de datos. Esto se puede hacer a través de un controlador ODBC (para lenguaje C/C++) o un controlador JDBC (para Java) . El conducto para la agenda, por ejemplo, sabe donde reside el servidor de la base de datos y que formas de comunicación soporta.

Entender como convertir la estructura de la base de datos en una base de datos Palm. El conducto que utiliza la base de datos no esta disponible en el dispositivo Palm, así que se tienen que convertir los datos en registro de la base de datos Palm, que la aplicación pueda fácilmente leer y comunicar. El conducto de la aplicación de la agenda convierte cada columna que lee en la base de datos relacional en una contraparte específica para una base de datos Palm.

Decidir que subconjunto de datos es necesario en el dispositivo. Las limitaciones de memoria y velocidad en una Palm, hacen impractico cargar el conjunto completo de datos en el dispositivo. Las consideraciones de seguridad y confidencialidad también juegan un papel importante, por ejemplo, los usuarios de la agenda pueden ver únicamente la información que es relevante para su cargo dentro de la compañía.

Asegurar la integridad de los datos, ya que los datos los usarán varios usuarios o aplicaciones. Detectar y resolver conflictos cuando se sincronizan datos por múltiples usuarios. Este problema existe en cualquier sistema multiusuario. Los usuarios que están directamente conectados al servidor de la base de datos ven la información actualizada y tienen menos conflictos que los usuarios que no están conectados. Un usuario de la agenda que no ha realizado el proceso Hotsync

(sincronización) por varios días puede haber hecho cambios sobre información desactualizada, sin saber que alguien más hizo cambios similares en la base de datos central algún día anterior. El conducto de la agenda tiene que resolver esos cambios y notificar al usuario si los cambios fueron rechazados o modificados.

Detectar y resolver conflictos cuando hay sincronización de datos con múltiples clientes. Este problema existe en cualquier sistema multiusuario, pero los usuarios que están conectados directamente a un servidor de base de datos son mas apropiados para hacer actualizaciones y evitar conflictos que los usuarios que están desconectados. Un usuario de la agenda, por ejemplo, que no realiza un proceso de HotSync por muchos días puede haber hecho cambios con información antigua (de una sincronización previa) sin saber que alguien mas realizó cambios similares en la base de datos central un día después. El conducto de la agenda tiene que resolver esos cambios y notificar a cualquier usuario si sus cambios fueron rechazados o modificados.

3.7.3 Estrategias de sincronización

Las tareas de sincronización (extracción de datos así como resolución de conflictos[6]) cambian de aplicación a aplicación dependiendo de los requerimientos y de las reglas del negocio de la organización, pero, el desarrollador es el único quien puede decidir que estrategias debe de usar. A continuación se explican las diferentes estrategias de sincronización:

Extrayendo los datos y creando subconjuntos

Una base de datos relacional contiene mas datos que los que la aplicación requiere, es decir, que la base de datos contiene tablas de sistema además de la tablas

[6] El término resolución de conflictos se refiere a que cuando se realiza una sincronización se puede dar el caso de que se transfiera un registro el cual ya exista en la base de datos, para lo cual se debe de seguir un método para resolver este problema.

usadas por la aplicación. La primera tarea para el desarrollador es hacer subconjuntos de datos de tal manera que solo se extraen los datos que la aplicación requiere y en el formato adecuado. Conceptualmente, esto se realiza mediante una sentencia SELECT que define el subconjunto, además que el software de sincronización puede presentar este conjunto de datos de forma diferente. Las estrategias de sincronización son las siguientes:

Sincronización incremental

Si es posible, la sincronización con la base de datos debe ser incremental. En otras palabras, únicamente los registros en la base de datos que son nuevos o que han cambiado desde la última sincronización deben ser transferidos al dispositivo. La mejor manera de realizar esto es insertar un campo del tipo timestamp[7] en la tabla y actualizar el valor de éste campo para un registro cada vez que es modificado. Se puede utilizar un trigger[8] que automáticamente ejecute unas sentencias SQL que actualicen el valor del campo timestamp.

La aplicación guarda el valor timestamp de la última actualización y usa ese valor para realizar partición de los datos y se forme un subconjunto excluyendo los que tengan el valor timestamp mas antiguo. Hay que resaltar que el valor timestamp no se necesita sincronizar y transferir al dispositivo.

Si la sincronización incremental no es utilizada, entonces todos los registros que coincidan con el criterio de selección participan en el proceso de sincronización. Esto es llamado algunas veces como refreshing the data (refresco de los datos) snapshot[9] synchronization. Tablas pequeñas o tablas con cambios frecuentes son buenos candidatos para el refreshing (refresco) opuesto a la sincronización incremental.

[7] El tipo de datos timestamp es utilizado por la herramienta de Sybase Adaptive Server Anywhere para almacenar datos de fecha y hora.
[8] Un trigger es un procedimiento almacenado que se ejecuta cuando se dispara un determinado evento.
[9] Es conocido como hoja de respuestas instantánea y contiene una copia de los datos resultado de una consulta.

Mientras se desarrolla la aplicación, puede ser útil encontrar una manera de realizar sincronizaciones con snapshots (snapshot synchronizations) y probar la extracción de datos y las estrategias de resolución de conflictos.

Partición de Columnas

La partición de columnas significa, seleccionar únicamente las columnas en una tabla que es requerida por la aplicación. Las otras columnas no son sincronizadas, y hay que tener en cuenta que no solo se debe hacer la selección de los campos que son desplegados (aunque eso es un buen comienzo) sino que se pueden requerir algunos otros para el funcionamiento apropiado, y en que puede ser requerida principalmente la llave primaria[10], ya que es la única forma de identificar a un registro en la tabla.

Partición de registros

La partición de registros es muy similar a la partición de columnas, nada mas que ahora se hace sobre los registros y no sobre los campos. La idea es bastante simple: seleccionar únicamente los registros que son requeridos por el usuario de la aplicación.

Conversión explícita de tipos de datos

Los datos en la base de datos no siempre se encuentran en el formato mas conveniente o eficiente para la aplicación. Se pueden realizar conversiones de tipos de datos explícitamente mediante la función CAST[11] o su equivalente. Esto se puede ver cuando por ejemplo, se necesita convertir un campo del tipo carácter de longitud fija a uno del tipo carácter de longitud variable para ahorrar espacio en el dispositivo:

SELECT CAST (Direccion as VARCHAR(100)) FROM CLIENTE

[10] Es el campo o los campos que identifican de forma única a un registro en una tabla.
[11] Esta función se utiliza para que un dato tome un tipo de dato especifico.

Conversiones explicitas en campos que se pueden actualizar deben siempre ser reversibles. Esto es, que el servidor de la base de datos tiene que ser capaz de realizar conversiones implícitas en cualquier actualización para corregir el tipo de dato al correcto para ese campo si la actualización es exitosa.

3.8 RESOLVIENDO CONFLICTOS

Un conflicto en una Palm ocurre cuando dos o mas usuarios modifican los mismos datos de forma independiente cada uno. Al momento de detectar un conflicto la aplicación debe almacenar el valor antiguo y el nuevo. Cuando se realiza la sincronización, un conflicto es detectado primero comparando el valor almacenado por la aplicación con el valor actual almacenado en la base de datos. Si los dos son iguales, no ocurre conflicto alguno y el nuevo valor es guardado en la base de datos. De otro modo, otro usuario o aplicación ha modificado el valor en la base de datos y una elección debe ser hecha para saber que valor se debe mantener, el valor que se acaba de modificar en la base de datos o el nuevo valor de la aplicación o quizá una combinación de los dos.

No existe una única estrategia de resolución de conflictos que sea apropiada para todas las aplicaciones, así que en cada nueva aplicación se debe decidir como se deben manejar los conflictos.

3.8.1 Evitándose conflictos

La estrategia mas simple es evitar los conflictos. Si los usuarios no están permitidos para realizar cambios, entonces no van a existir conflictos. Se puede definir subconjuntos de datos que sean mutuamente excluyentes para cada usuario, de tal manera que un usuario no puede modificar los datos de otro usuario.

3.8.2 Un lado siempre gana

Si los conflictos no pueden ser evitados, la siguiente estrategia simple es que un lado de la sincronización cuando hay un conflicto siempre gane. Es decir, o el nuevo valor del dispositivo sobrescriba el valor actual en la base de datos, o sino el valor en la base de datos sobrescriba el valor en la aplicación. Si la base de datos siempre gana sobre el dispositivo, la aplicación necesitará informar al usuario que la actualización falló y quizá debe dar al usuario la oportunidad de volver a aplicar los cambios se es que estos tienen sentido.

3.8.3 Ambos lados ganan

Otra estrategia es permitir que los dos cambios ocurran. Las aplicaciones Palm desarrolladas ofrecen esta elección, por ejemplo, asegurando que los cambios no se pierdan cuando un conflicto ocurre. Sin embargo, cuando se actualizan datos en la base de datos relacional, esta estrategia rara vez tiene sentido, porque los procedimientos de sincronización identifican los registros en una tabla para sincronizar por la llave primaria, y esta siempre es única en una tabla.

Para que ambos lados ganen, una copia del registro, idéntico con el que ya existe, tendría que ser insertado en la tabla, y esto violaría la restricción de que la llave debe ser única. La única forma de hacer esto sería generar una nueva llave primaria para la copia del registro. Sería mejor reportar un error que tener registros duplicados.

3.8.4 Estrategias personalizadas

Si ninguna de las estrategias simples resuelven el problema, se puede implementar una estrategia propia personalizada para la resolución del conflicto. Se puede establecer que los cambios del administrador siempre sobrescriban los de sus subordinados. O se puede mandar el conflicto a un humano para que se tome una decisión.

3.9 CONDUCTOS

Una parte importante de las aplicaciones Palm es la conexión con la computadora de escritorio, y ya que los dispositivos Palm actúan como una extensión de las PC's, es crucial que la información se pueda intercambiar entre la Palm y la PC con mucha facilidad. Los conductos son el mecanismo para hacer esto.

Un conducto es código (un programa) en la computadora de escritorio que es llamado durante la sincronización HotSync para manejar el flujo de la información que va de la base de datos hacia el dispositivo y viceversa. Los conductos registran la base o bases de datos para las cuales es responsable. Nada mas que hay que hacer notar que cada base de datos debe tener únicamente un solo conducto responsable por ella.

Los conductos son creados utilizando la herramienta Conduit Development Kit para Windows (C/C++), Mac OS (C/C++) o Java.

Las aplicaciones que no tienen un conducto utilizan uno que es proveído por el sistema. Este conducto que fue creado por Palm es usado para realizar respaldos y es parte de la herramienta HotSync. Este conducto que realiza el respaldo copia los datos de la aplicación o la base de datos del dispositivo y la almacena como un archivo. Cuando se crea la base de datos es cuando se especifica el conducto para respaldos.

Durante la sesión HotSync, el conducto para respaldos es llamado por las bases de datos que no tienen su propio conducto y que han sido marcadas como que necesitan un respaldo. En este punto, el conducto copia cada registro de la base de datos y copia la información de los encabezados de la base de datos a un archivo, y este archivo es usado, si es necesario, para restaurar el estado del dispositivo Palm.

Existen conductos mas sofisticados que realizan mas funciones que únicamente copiar información para el respaldo. Los conductos pueden leer o escribir información

específica hacia o desde el dispositivo. Por ejemplo, un conducto para una aplicación de ventas puede manejar las transferencias de la PC hacia el dispositivo de las nuevas listas de precios, sobrescribiendo cualquier lista de precios existente que haya expirado; y también puede ser responsable de transferir del dispositivo hacia la PC las ordenes de venta.

Los conductos mas sofisticados son aquellos que sincronizan los registros en el dispositivo con la información de la computadora de escritorio. Estos conductos para la sincronización usualmente trabajan asignando a cada registro un único identificador (ID) y actuando sobre estos cuando un registro particular haya sido modificado.

En el ambiente Windows, un conducto es una librería dinámica enlazada, DLL (Dynamic Link Library) que es llamada cada que una sincronización HotSync ocurre.

Los conductos tienen acceso a las bases de datos en el sistema operativo Palm. El manejador del conducto maneja las complejidades de la comunicación haciendo llamadas simples a rutinas para leer y escribir registros en la base de datos. El manejador del conducto interactúa con el protocolo de comunicación.

3.10 PALM EN LA ACTUALIDAD

Existen varios modelos de Palm en el mercado, los cuales dependen principalmente de la compañía que los elabora y de las características físicas que posee cada modelo.

Existen los modelos Palm IIIe, Palm IIIc y Palm IIIxe los cuales forman la familia de Palm III los cuales cuentan con las siguientes características:

El modelo Palm IIIe cuenta con 2MB de memoria, la pantalla es monocromática. Utiliza dos baterías AAA. El sistema operativo con que cuenta este modelo es Palm OS 3.1. Dimensiones 12 x 8 x 1.7 cm.

El modelo Palm IIIxe cuenta con 8MB de memoria, la pantalla es monocromática y utiliza dos baterías AAA y tiene un sistema operativo Palm OS versión 3.1. Sus dimensiones son 12 x 8 x 1.7 y pesa 170 gramos.

El modelo Palm IIIc cuenta con pantalla a color TFT de matriz activa tiene 8 MB de memoria tiene un sistema operativo Logiciel Palm OS versión 3.5. Tiene una batería recargable de Litio la cual dura aproximadamente 2 semanas.

También existen los modelos m100 y m105 que forman la familia m100. Estos modelos cambian su diseño y cuentan con la característica de que si se le oprime un botón muestra la hora sin necesidad de prenderla. También cuenta con un block de notas, que a diferencia del block de notas tradicional donde se introduce el texto solamente por medio del área de Graffiti o por el teclado, se graba todo lo que se introduce en la pantalla. Estos modelos, aparte de las características mencionadas, cuentan con las características básicas de todos los modelos Palm. La diferencia entre la m100 y la m105 es que la primera tiene 2 MB de memoria y la segunda tiene 8 MB.

Los modelos V y Vx forman la familia Palm V la cual consta con una batería de Litio recargable, de tal manera que cuando la Palm se coloca en su base se recarga. Preparada para conectarse a Internet por medio de un modem. Tiene 8 MB de memoria y cuenta con un sistema operativo Palm OS versión 3.3. Sus dimensiones son 11.5 x 8 x 1 cm y pesa 114 gms.

También está la familia VII que la forman los modelos VII y VIIx los cuales cuentan con la función de conectarse a Internet vía comunicación celular. Hay que tomar en cuenta que dicha tecnología no se cuenta en México.

También existe la familia m500, la cual esta formada por los modelos Palm m500 y Palm m505 los cuales tienen un slot donde se conectan tarjetas que ya tienen cierta información almacenada. Esta familia de Palm ya cuenta con la última versión de Palm OS que es la 4.0. La diferencia entre los dos modelos es que el modelo m505 tiene una pantalla a color mientras que la m500 la tiene monocromática.

Todos los modelos antes mencionados tienen un sistema operativo Palm OS que varía en la versión, además contienen agenda electrónica y de direcciones, conectividad de correo electrónico con el escritorio, lista de trabajos pendientes, block de notas, control de gastos, calculadora, seguridad, juegos y sincronización local y remota con la PC

Existe el modelo SPT1500, el cual, fue realizado por la compañía "Symbol". Este modelo tiene la característica de que lee códigos de barras aparte de todas las características básicas con que cuentan todos los modelos de Palm[12].

El modelo que se utilizó para realizar este proyecto, es el modelo SPT1700, ya que cuenta con lector de códigos de barras, el cual es indispensable para poder ingresar los códigos que identifican a cada unidad de empaque, y a diferencia del modelo SPT1500 este cuenta con un diseño industrial con el cual puede aguantar un uso rudo. De igual forma, este modelo tiene un sistema Palm OS versión 3.2, y cuenta con todas las funciones de este sistema operativo y que todos los modelos Palm tiene como Bloc de Notas, Agenda, Direcciones, etc. A continuación veamos las características del modelo SPT1700.

3.10.1 El modelo SPT1700

La familia SPT1700 continúa la revolución en la informática de bolsillo, combinando lectura integrada, conectividad sofisticada y resistencia con la conocida

[12] Las características básicas de una Palm son la agenda, direcciones, bloc de notas, lista de tareas, calculadora,etc.

plataforma Palm. La serie SPT1700 ofrece capacidades reforzadas de comunicaciones de datos con un vínculo integrado a la arquitectura abierta de Spectrum24.

La familia de productos SPT1700 de numerosas características, amplía el alcance de recursos IT en una serie de industrias. El personal administrativo del almacén puede registrar fácilmente nuevos niveles de inventario en sistemas host de red a medida que se descargan los envíos. Los profesionales de servicio en la industria restaurantera, pueden realizar el registro de clientes fuera de los hoteles; y los médicos y personal de asistencia en los hospitales pueden tener a mano los historiales más recientes de los pacientes.

La atención detallada a la ergonomía y los bordes suavemente redondeados proporcionan un ajuste seguro en la mano. Con una altura de 17,8 cm, el SPT 1700 cabe perfectamente en un bolsillo y con un peso de sólo 288 gm, esta potente herramienta de productividad es perfecta para los trabajadores con movilidad. Este modelo que cuenta con la tecnología "scan engine" de vanguardia, el más pequeño, ligero e inteligente que se puede encontrar en la actualidad.

3.10.2 La SPT 1700 en entornos hostiles

Este modelo de Palm es lo suficientemente fuerte como para utilizarla en entornos industriales y de fabricación. La carcasa reforzada resiste caídas de hasta 1,2 m sobre superficie de hormigón. Probados según las normas IP54 para la protección frente a la lluvia y el polvo, estos ordenadores se pueden utilizar al aire libre en las zonas de carga y astilleros y, debido a que utilizan un diodo láser de 650 mm, podrá ver el haz de lectura incluso a plena luz del sol. Una pantalla de cristal líquido antirreflejante de alto contraste facilita la lectura de los datos con baja luz o al aire libre. Una batería de litio-ión recargable y configuraciones opcionales de cunas de una y múltiples ranuras garantizan una comodidad flexible y un alto rendimiento, independientemente de la intensidad de la tarea.

Mediante la plataforma Palm, los programadores pueden crear soluciones a partir de numerosos entornos de desarrollo estándar. Tiene un RAM estándar de 2 MB, la memoria ROM Flash es de 2 MB. Los programadores pueden utilizar herramientas de desarrollo Palm OS para escribir aplicaciones personalizadas rápida y económicamente. El juego de herramientas de desarrollo incluye Interfaces de programas de aplicaciones (API) de Symbol que controlan el scanner, y manipulan la entrada de datos y la impresión mediante los puertos serie o infrarrojos. Las herramientas de desarrollo de software, CodeWarrior, SatelliteForms, PenRight!, AvantGo y Aether ScoutWare, admiten las API de Symbol como herramientas de desarrollo de aplicaciones[13].

Los profesionales con movilidad en cualquier empresa pueden ahorrar tiempo y papeleo mediante el innovador ordenador de bolsillo SPT 1700 de Symbol Technologies. Estas son algunas de las características funcionales de éste modelo:

3.10.3 Características del modelo SPT1700

Opción de conectividad de LAN[14] inalámbrica Spectrum 24	Transmite datos a y desde los sistemas de información de red en tiempo real
La carcasa reforzada cumple las normas IP54	Soporta los entornos industriales Hostiles
"Scan engine" SE 900/SE 900HA integrado	El "scan engine" de vanguardia proporciona un rendimiento óptimo
Bateria de litio-ión Recargable	Flexible y conveniente para entornos exigentes
2MB de RAM y ROM con 8 MB de RAM y 4MB de ROM opcionales disponibles	Flexibilidad de memoria para satisfacer las necesidades de las aplicaciones
Entorno de desarrollo y sistema operativo Palm OS	Herramientas de desarrollo de fácil adquisición incluidas AdvantGo, CodeWarrior, SatelliteForms, Aether ScoutWare y PenRight!.
Admite cunas de módem y en serie de una ranura y cunas Ethernet y en serie de múltiples ranuras	Comunicaciones flexibles del host para la mayoría de los entornos del usuario y de la red

[13] La información acerca del SPT1700 fue obtenida de la página www.symbol.com
[14] LAN son las siglas de Local Area Network o red de área local.

3.10.4 Especificaciones principales de las SPT 1700/1700-2D

Dimensiones:	25,4 mm de alto x 92 mm de ancho x 177,8 mm de largo
Peso incluida la batería:	Versión por lotes: 288 gm; Versión inalámbrica: 330 gm
Especificación de caídas:	1,2 m sobre superficie de hormigón
Batería:	3,7 V, litio-ión recargable de 1400mAh
Sellado al entorno:	IP54 (polvo y lluvia)
Temperatura de funcionamiento / Temperatura de almacenamiento:	-20° a 50°C; -25° a 50°C
Humedad:	5% a 90% de humedad relativa sin condensación
Descarga electrostática:	8kVdc de aire, 4kVdc de contacto
Scan Engine:	SE 900/SE 900HS
CPU:	Motorola DragonBall
Sistema operativo:	Palm OS ™ 3.2
Memoria (versiones por lotes e inalámbrica):	SPT 1700: 2 MB/2 MB, 2 MB/8 MB, 4 MB/8 MB SPT 1700-2D: 2 MB/8 MB, 4 MB/8 MB
Desarrollo de aplicaciones:	CodeWarrior, SatelliteForms, PenRight!, AvantGo y Aether ScoutWare
Pantalla:	Pantalla LCD 160 x 160 de alto contraste y antirreflejante
Cunas:	Cuna con módem y para vehículo, cuna en serie de una y cuatro ranuras y cuna Ethernet de cuatro ranuras
Cable de carga y comunicaciones :	En serie/carga en la batería del terminal
Lector de tiras magnéticas (MSR):	Conexión en serie para la legibilidad de MSR
Cargador de baterías universal de cuatro ranuras:	Recarga varias baterías; adaptador de baterías: se utiliza con el cargador de baterías universal de Symbol
Red:	Cumple con la norma de ondas aéras IEEE 802.11 de Spectrum24 de Symbol
Técnica de dispersión:	Salto de frecuencia
Velocidad de transmisión:	1 Mbps
Antena:	Interna
Rango:	Espacio abierto: hasta 303 m, Característico: 54 a 76 m
Potencia de salida:	500 mW EE.UU.; 100 mW internacional
Rango de frecuencias:	Depende del país, normalmente 2,4 a 2,5 GHz
Seguridad eléctrica:	Certificado para UL1950, CSA C22.2 N° 950, EN60950/IEC950
EMI / RFI:	Clase B Parte 15 de la FCC, ICES-003 Clase B, Directriz sobre compatibilidad electromagnética de la Unión Europea, SMA australiana
Seguridad de láser:	CDRH Clase II, CEI Clase 2

CAPITULO 4

BASE DE DATOS "ULTRALITE"

4.1 ULTRALITE

Dada la popularidad de la plataforma Palm y la complejidad del desarrollo de conductos especializados, fue inevitable que las soluciones con una sincronización predefinida aparecieran en el mercado. Los distribuidores de bases de datos están cada vez mas ansiosos de expandir el alcance de sus sistemas de bases de datos y así estar en la pelea en esta importante área de crecimiento como lo es la plataforma Palm.

UltraLite es una tecnología de instalación y expansión de Adaptive Server Anywhere[1] cuyo objetivo son los dispositivos móviles como son los teléfonos celulares, organizadores portátiles entre otros.

El propósito de UltraLite es proveer la funcionalidad y confiabilidad de una base de datos SQL que corre en estos dispositivos y que además tiene la habilidad de sincronizar datos con una base de datos central mientras mantiene un tamaño extremadamente pequeño.

Una base de datos UltraLite es una base de datos relacional ajustada, es decir, que es construida a la medida exacta de la aplicación y con capacidad para crear funciones de sincronización. También cuenta con un diseño extremadamente pequeño que depende completamente de las capacidades y características que la aplicación requiere.

UltraLite permite que las aplicaciones en los dispositivos como Palm usen lenguaje SQL para llevar a cabo el almacenamiento de los datos, así como para extraerlos o manipularlos; además que soporta la integridad referencial, procesamiento por transacciones y unión de varias tablas. De hecho, UltraLite soporta la mayoría de los mismos tipos de datos, funciones en tiempo de ejecución y la manipulación de datos por SQL que Sybase Adaptive Server Anywhere.

[1] Es una herramienta de Sybase para manejar bases de datos principalmente locales.

Una base de datos UltraLite se sincroniza con una fuente de datos ODBC, lo que significa que puede ser usada con casi cualquier tipo de base de datos, no solamente con Sybase.

Se puede pensar que la plataforma Palm no es la mas recomendable para una base de datos relacional, y que una base de datos UltraLite debe ser o muy grande o de capacidades muy limitadas. De hecho una base de datos UltraLite es muy pequeña, porque está construida a la medida exacta de la aplicación, basada únicamente en las características que la aplicación requiere.

Las herramientas de desarrollo de UltraLite preprocesan el SQL, analizan la base de datos de referencia y generan código fuente C/C++ que implementan las operaciones SQL en la aplicación. Cuando éste código generado es compilado y enlazado con UltraLite en tiempo de ejecución, el resultado es el motor de la base de datos de la aplicación que puede ser tan pequeña que mida 50 kb. La extensión del código del motor de la base de datos UltraLite depende del número de sentencias SQL en la aplicación.

Los dispositivos de destino de UltraLite como lo es la Palm no tienen un disco duro y tienden a tener relativamente lento el procesador, por lo tanto UltraLite en tiempo de ejecución está optimizado para tener la base de datos en memoria, utilizando algoritmos y estructuras de datos que proveen un alto desempeño y bajo uso de la memoria.

4.1.1 Plataformas soportadas

- Las aplicaciones UltraLite pueden ser ejecutadas en la plataforma Palm y en la plataforma de Windows CE.

- El desarrollo de una aplicación con una base de datos UltraLite se puede realizar en sistemas operativos Windows 95/98. Hay también que señalar que las herramientas de UltraLite permiten desarrollar aplicaciones que corren en Windows NT, así que se puede probar la lógica de la base de datos en la aplicación sin descargar la aplicación UltraLite al dispositivo. Esto porque la base de datos UltraLite no es persistente en Windows NT ya que cada vez que en Windows NT la aplicación UltraLite corre, esta empieza con la base de datos vacía.

- CodeWarrior es requerido para compilar las aplicaciones para la ejecución en la plataforma Palm.

- Microsoft Visual C++ 5.0 en Windows NT usando como destino Windows CE 2.0 corriendo en dispositivos con un procesador MIPS o SH3.

- Las aplicaciones UltraLite soportan el sistema operativo Palm OS 2.0 o superior, mientras que MobiLink requiere el manejador HotSync 3.0 o superior.

4.1.2 Arquitectura de UltraLite

Productos con bases de datos SQL usan típicamente una arquitectura cliente / servidor. El servidor recibe y procesa las peticiones SQL de las aplicaciones cliente, manteniendo la base de datos de acuerdo con las peticiones hechas. El servidor se encarga de proteger la información de fallas y garantiza que transacciones completas sean recuperadas en caso de fallas, y transacciones incompletas sean deshechas.

UltraLite tiene la misma arquitectura cliente / servidor que otros sistemas de base de datos SQL. Sin embargo, el motor de la base de datos UltraLite no es un proceso separado, es decir, esta enlazado con la aplicación para formar un solo archivo ejecutable.

El soporte UltraLite de tiempo de ejecución es accedido de una DLL o de una librería estática enlazada.

El proceso de desarrollo UltraLite genera código fuente en C/C++ que es compilado con el código fuente de la aplicación. Cuando se enlaza la aplicación, se enlaza todo el código C/C++ junto con librerías de UltraLite o se importan librerías, el resultado es un solo archivo ejecutable que contiene la lógica de la aplicación y la lógica de la base de datos requerida por la misma aplicación.

Cuando es ejecutado por primera vez en un dispositivo, este archivo ejecutable automáticamente crea la base de datos UltraLite. Esta base de datos al principio esta vacía, pero se pueden agregar datos, o a través de una sincronización con una base de datos central.

UltraLite provee también protección contra fallas como otros sistemas de bases de datos SQL. UltraLite mantiene la base de datos en memoria persistente y no en el archivo. Los dispositivos no tienen unidad de disco, y por lo tanto UltraLite usa la memoria para almacenar el contenido cuando el dispositivo no esta corriendo.

Hay que mencionar sin embargo, que UltraLite no permite que el esquema de la base de datos sea modificada una vez que la aplicación es instalada y distribuida; y cuando una nueva versión de la aplicación requiere mas tablas o mas columnas, se modifica la base de datos de la computadora de escritorio y la nueva versión de la aplicación es reinstalada y redistribuida, y la base de datos UltraLite es llenada con datos nuevamente a través de la sincronización.

4.1.3 Principales limitaciones de UltraLite

Para mantener las bases de datos personalizadas tan pequeñas como sea posible, UltraLite limita lo que una aplicación puede hacer. Las limitaciones son las siguientes:

- ***No son permitidos los cambios en el esquema de la base de datos en tiempo de ejecución.*** Una aplicación UltraLite no puede alterar el esquema de la base de datos de ningún modo. Los cambios deben ser hechos en tiempo de diseño y una nueva versión de la aplicación debe ser construida para reflejar el nuevo esquema.

- ***No se tiene acceso a tablas de sistema o funciones de sistema.*** Una base de datos personalizada de UltraLite no tiene tablas de sistema y no soporta funciones de sistema.

- ***No hay SQL dinámico.*** Solamente instrucciones de SQL estático son permitidas en una aplicación UltraLite. Las instrucciones parametrizadas, en cambio, si son permitidas, así que el valor de una cláusula WHERE si puede ser especificado en tiempo de ejecución, como se muestra en las funciones del apéndice A.

Estas son principalmente las limitaciones que existen para construir una base de datos UltraLite. Existen otras limitaciones que no son tan importantes, las cuales son:

- ***No soporta nada mas que SQL estático.*** Otras interfaces como ODBC, OLEDB o ADO no son soportadas actualmente. El acceso directo a registros en memoria no es permitido tampoco.

- ***No soporta otros lenguajes.*** C/C++ es el único lenguaje que actualmente es soportado.

- ***No soporta stored procedures, variables, triggers o clases de Java.*** Dado que la base de datos UltraLite está ajustada a la aplicación, los mismos resultados son conseguidos escribiendo rutinas de librerías que son enlazadas a la aplicación.

- ***No se pueden compartir datos entre aplicaciones.*** Cada base de datos personalizada queda separada de alguna otra base de datos, aún si dos aplicaciones usan las mismas tablas, cada una requiere su propia base de datos.

- ***No hay tipos de datos de punto flotante, long varchar, long varbinary.*** Y los tipos de datos como: CHAR(N), VARCHAR(N), BINARY(N) están limitados a que N <= 2048.

- ***No hay conversión de caracteres.*** UltraLite asume que todos los caracteres son representados usando la pagina de códigos 1252, que es la pagina de códigos de Windows US, la cual es también soportada por la plataforma Palm.

4.2 SINCRONIZACION CON MOBILINK

Las bases de datos de construcción personalizada son el corazón de la tecnología UltraLite. La característica mas importante de este tipo de bases de datos es su capacidad para sincronizarse con bases de datos externas. La sincronización para las bases de datos UltraLite depende de un conducto corriendo en una computadora de escritorio. Este conducto es una interfaz entre la base de datos UltraLite y la base de datos externa utilizando el servidor MobiLink[2] de Sybase como puerta entre estas dos bases de datos.

Una aplicación UltraLite puede también conectarse directamente al servidor MobiLink usando TCP/IP o el puerto serial sin utilizar el conducto. El servidor MobiLink permite definir una extracción sofisticada de datos y estrategias de resolución de conflictos. El conducto y el servidor MobiLink maneja todo el proceso de sincronización automáticamente.

[2] Por medio de éste se realiza la sincronización para éste proyecto de tesis.

El servidor MobiLink puede sincronizar cualquier fuente de datos ODBC de bases de datos relaciónales. UltraLite incluye scripts[3] para instalar las tablas necesarias y almacenar procedimientos en bases de datos de Adaptive Server Anywhere, Sybase Adaptive Server Enterprise (formalmente llamada Sybase SQL Server), Microsoft SQL Server y Oracle.

Juntos, las bases de datos UltraLite de construcción personalizada y el servidor de sincronización MobiLink hacen fácil la construcción de completas aplicaciones con bases de datos sin tener que escribir conductos personalizados.

La mayoría de las aplicaciones para dispositivos móviles requiere que los datos se transfieran a una base de datos central, también, de la base de datos central hacia el dispositivo y ambas cosas. Esta funcionalidad es esencial para hacer la función de la aplicación efectiva con la infraestructura de la información de una organización. La tecnología de sincronización Mobilink incluida con UltraLite, esta diseñada para trabajar con el estándar de manejo de base de datos SQL de Sybase y otros productos.

UltraLite en tiempo de ejecución automáticamente guarda los cambios hechos a la base de datos UltraLite entre cada sincronización con la base de datos central. Cuando la base de datos UltraLite es sincronizada, todos los cambios realizados desde la sincronización anterior serán transferidos hacia la base de datos central.

Las bases de datos en dispositivos móviles contienen una fracción de los datos que existen en la base de datos central, la cual es llamada base de datos consolidada ya que consolida todos los datos.

Las tablas en cada base de datos UltraLite contienen un subconjunto de los registros de la base de datos central y un subconjunto de columnas de esta base de datos.

[3] El término script se refiere a un conjunto de sentencias que entiende el manejador de la base de datos y que realizan ciertas acciones en ésta.

Por ejemplo, una tabla de clientes contiene arriba de 100 columnas y 100,000 registros en la base de datos consolidada, pero la base de datos UltraLite requiere únicamente 4 columnas y 1000 registros. Mobilink permite definir el subconjunto exacto de datos que será transferido a cada base de datos remota.

La sincronización Mobilink es flexible, es decir, se define el subconjunto de datos utilizando el lenguaje nativo SQL del sistema manejador de la base de datos central. En la mayoría de los casos, las tablas en la base de datos UltraLite corresponden a tablas en la base de datos central, aunque esto no es necesariamente requerido. Se puede llenar con datos una tabla en la base de datos UltraLite con una tabla de la base de datos consolidada pero con distinto nombre, o se puede llenar con una unión de dos o mas tablas.

En la sincronización es importante transferir solo los datos que han cambiado y no hacer una copia completa de los datos. La transferencia de los datos del dispositivo hacia la PC de escritorio esta automáticamente determinada por UltraLite en tiempo de ejecución, mientras que, la transferencia de datos de la PC al dispositivo esta definida usando el lenguaje nativo SQL del sistema manejador de la base de datos central. Esto asegura que solamente los datos que necesitan ser transferidos serán transferidos.

Las bases de datos en los dispositivos frecuentemente comparten datos comunes, esto se da cuando dos o mas tablas remotas actualizan los mismos registros aunque esto puede generar conflictos. Los conflictos deben ser detectados y resueltos cuando los cambios son transferidos hacia la base de datos central. La sincronización UltraLite automáticamente detecta los conflictos y la lógica para resolverlos esta definida en SQL nativo del sistema manejador de la base de datos central.

Una aplicación UltraLite se sincroniza con una base de datos central, consolidada a través del servidor Mobilink. Este servidor provee una interfaz entre las comunicaciones de la aplicación UltraLite y el servidor de la base de datos.

El proceso de sincronización se controla usando scripts. Estos scripts son sentencias SQL o procedimientos escritos en el lenguaje nativo del sistema manejador de la base de datos consolidada. Por ejemplo, se puede usar una sentencia SELECT para identificar las columnas y tablas en la base de datos consolidada que corresponden a cada columna de un registro transferido a la base de datos central de una tabla en la aplicación UltraLite.

Estos scripts están almacenados en tablas en la base de datos consolidada y cada script controla un evento particular durante el proceso de sincronización.

La sincronización puede ocurrir a través de una variedad de canales. Una opción es una conexión TCT/IP entre la aplicación y el servidor Mobilink, otras plataformas requieren una solución diferente, como la plataforma Palm que utiliza HotSync[4], TCP/IP o comunicación directa por el puerto serial de comunicaciones. Pero independientemente del canal que se utilice siempre se controla el proceso de sincronización usando los mismos scripts SQL definidos en la base de datos consolidada.

La sincronización es iniciada desde la aplicación por una llamada a la función ULSynchronize cuando se usa TCP/IP o sincronización por el puerto serial. La interfaz para HotSync es un poco diferente pero también es muy simple. En éste proyecto de tesis no se utiliza la sincronización por HotSync sino que se hace la llamada a la función ULSynchronize desde una opción del menú de la aplicación.

Hay un componente que juega un papel muy importante en el proceso de sincronización, es el servidor de sincronización Mobilink. Este servidor provee una interfaz entre la base de datos consolidada y la aplicación UltraLite, y gracias a que es un componente separado, no es parte de un producto de base de datos.

[4] Programa que se encarga de la sincronización entre el dispositivo Palm y una computadora de escritorio, pero que no es utilizado en éste proyecto de tesis.

El servidor Mobilink se conecta a la base de datos consolidada donde se almacena la información de la sincronización incluyendo los scripts de sincronización. Estos scripts son la guía para el servidor Mobilink para manejar las conexiones con usuarios remotos y para hacer coincidir datos de la base de datos remota con la base de datos consolidada.

4.2.1 Iniciando la sincronización de una aplicación UltraLite

La sincronización se inicia desde la aplicación UltraLite y como se mencionó anteriormente se inicia con una llamada a la función ULSynchronize para sincronización por TCP/IP y por puerto serial, el cual es el utilizado en este proyecto de tesis. Para la sincronización HotSync hay una pequeña diferencia en la interfaz entre la aplicación y el sistema.

En este proyecto de tesis la sincronización se llevará a cabo por medio del puerto serial. Será necesario levantar el servidor Mobilink desde la computadora de escritorio para poder realizar la sincronización, para esto se creará una opción en el menú de la aplicación del dispositivo para iniciar la sincronización.

Una vez que la sincronización es iniciada desde la aplicación o por HotSync, el servidor Mobilink y UltraLite en tiempo de ejecución controlan las acciones que ocurren durante la sincronización.

4.2.2 Identificación del usuario

Todos los dispositivos Palm o que cuentan con un sistema operativo Palm OS, están identificados con un nombre, el cual se establece cuando se acaba de adquirir el dispositivo y se está empezando a utilizar. Este identificador del dispositivo se le conoce como nombre de usuario.

Cada dispositivo debe pasar el nombre de usuario al servidor Mobilink al momento de sincronizar. El nombre de usuario es una cadena de caracteres que mide 128 caracteres como máximo. El servidor pasa el nombre de usuario como parámetro a los scripts de sincronización, haciendo posible que lo que cada script haga dependa de quien está realizando la sincronización. Se puede usar esta característica, para determinar si alguien es un gerente y si los cambios tienen precedencia sobre los de alguien mas que se este sincronizando en ese momento y se detecte un conflicto.

El nombre de usuario que se pasa al servidor Mobilink debe identificar al usuario y a la aplicación, porque solamente un conjunto de scripts de sincronización pueden ser asociados con una tabla en particular en la base de datos consolidada. Un nombre de usuario que incluye información acerca de la aplicación puede utilizar dos diferentes aplicaciones corriendo en el mismo dispositivo que tengan el mismo nombre de las tablas y permite a los scripts realizar diferentes selecciones de datos dependiendo de la aplicación.

Existe una tabla de sistema llamada ul_user que mantiene una lista de los nombres de usuario de la sincronización o sincronizaciones, y los obtiene de la llamada a la función ULSynchronize, también almacena el estado de sincronización de cada usuario en la columna commit_state. Esta información permite una apropiada recuperación si la sincronización es interrumpida.

El nombre de usuario identifica cada dispositivo remoto que se sincroniza con la base de datos central. Si una aplicación intenta la sincronización pero el nombre de usuario no existe en la tabla, el servidor Mobilink automáticamente inserta un nuevo registro, por esta razón, no se necesita hacer alguna acción especifica para darle mantenimiento a esta tabla.

4.2.3 Canales de sincronización

UltraLite ofrece tres canales de sincronización: HotSync, TCP/IP y serial. Dependiendo del canal que se seleccione se deberán pasar diferentes parámetros a las funciones ULPalmLaunch, ULPalmExit y ULSynchronize. Estas funciones son llamadas al inicio de una sincronización y son las que inician con todo el proceso de sincronización, ya que estas a su vez mandan llamar otras funciones o procedimientos que se realizan durante este proceso.

4.2.4 Como se utiliza el lenguaje SQL

Una aplicación UltraLite usa SQL propio de UltraLite para comunicarse con la base de datos UltraLite.

El preprocesador[5] SQL lee los archivos de entrada y busca las líneas que comiencen con la cadena EXEC SQL como ésta:

```
EXEC SQL INCLUDE SQLCA;
```

La cadena EXEC SQL identifica la línea como una instrucción de SQL, la instrucción debe terminar con punto y coma y pude ser dividida en varias líneas para su mejor comprensión y fácil lectura, también se pueden incluir comentarios de C o C++ entre la instrucción.

El texto que queda entre la cadena EXEC SQL y el punto y coma es la instrucción SQL que se va a ejecutar. Las instrucciones pueden ser comandos normales de SQL como UPDATE, DELETE, SELECT O INSERT; o también pueden ser comandos específicos del SQL para UltraLite así como INCLUDE O WHENEVER.

[5] Es un programa que traduce las sentencias SQL en instrucciones en lenguaje C/C++

Algo muy importante que hay que tomar en cuenta es que el preprocesador SQL no entiende ninguna instrucción de C o C++ y en particular las macros C/C++ (#define) y directivas de compilación (#if, #else, #endif, etc.) son ignorados por el preprocesador SQL. Las instrucciones SQL son procesadas en el orden en el cual son encontradas en el archivo fuente.

Un comando del SQL de UltraLite es aquel que debe definir en el archivo fuente el área de comunicaciones de SQL, el comando es SQLCA, que se usa de la siguiente manera:

EXEC SQL INCLUDE SQLCA;

El área de comunicaciones SQL define una estructura usada para comunicar errores y alguna otra información de la base de datos de UltraLite a código C/C++. Es necesario colocar la instrucción INCLUDE SQLCA hasta arriba de cada archivo SQL después de incluir los archivos de cabecera.

Ahora tenemos un ejemplo de cómo el preprocesador transforma el código en lenguaje C/C++:

```
#include <Pilot.h>
#include "ULPhoneData.h"
EXEC SQL INCLUDE SQLCA;
```

El preprocesador SQL transforma estas instrucciones en algo como esto:

```
#include <Pilot.h>
#include "ULPhoneData.h"

/* EXEC SQL INCLUDE SQLCA; */
```

```
#include "ulglobal.h"
extern SQLCA sqlca;
```

El código presentado en el apéndice A, es el código antes de ejecutar el preprocesador, es decir, que todavía no esta transformado de sentencias SQL a instrucciones de lenguaje C/C++.

4.3 LA BASE DE DATOS ULTRALITE

Para entender como trabaja UltraLite, se debe primero entender que sucede durante las fases de desarrollo, instalación y expansión de la aplicación. Durante la fase de desarrollo, las instrucciones SQL y las tablas de la base de datos requeridas por la aplicación son analizadas para generar una base de datos personalizada, es decir, a la medida de la aplicación y que es parte de ésta. Las estrategias de sincronización son también desarrolladas en este momento. En la fase de ejecución, la base de datos hecha a la medida y el servidor MobiLink trabajan juntos para manejar y sincronizar los datos de la aplicación.

4.3.1 Fase de desarrollo

El primer paso en el desarrollo de una aplicación UltraLite es definir la base de datos de referencia. Una base de datos de referencia es una base de datos de Adaptive Server Anywhere usada como una plantilla para construir la base de datos UltraLite, que es la base de datos del dispositivo.

La base de datos de referencia es usada durante la fase de desarrollo y cuando la aplicación es instalada en el dispositivo (Palm) esta base de datos no se instala, solo se utilizó su estructura. Y cuando la aplicación corre, ésta se sincroniza con la base de datos consolidada, la cual contiene las tablas y los datos para la sincronización. Se puede

utilizar únicamente una sola base de datos de Adaptive Server Anywhere que seria usada como una base de datos de referencia y la base de datos consolidada, si así se desea.

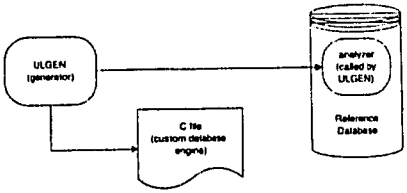
Entonces se puede decir que una base de datos de referencia modela un subconjunto de la base de datos consolidada, esto significa que hay una correspondencia de uno a uno entre las tablas y columnas de la base de datos de referencia y el subconjunto de tablas y columnas de la base de datos consolidada. La base de datos de referencia, que aunque es el modelo no tiene que ser exacta y su esquema puede ser diferente del esquema de la base de datos consolidada. Cualquier diferencia entre estos dos esquemas son manejados por los mapeos apropiados que se definan como parte del proceso de sincronización. La documentación de UltraLite recomienda que los dos esquemas sean tan similares como sea posible.

Además de modelar la estructura de la base de datos consolidada, la base de datos de referencia puede también contener datos representativos de los datos de la base de datos consolidada. Los datos representativos no son requeridos en la base de datos de referencia pero si se guardan algunos datos, éstos son usados para optimizar el diseño de una base de datos UltraLite.

Una vez que la base de datos de referencia ha sido definida el siguiente paso es escribir las instrucciones SQL que son requeridas por la aplicación, estas instrucciones son agregadas a los archivos fuentes C/C++ por medio del Adaptive Server Anywhere y operan contra el esquemas de la base de datos de referencia. Los archivos fuentes son procesados por el preprocesador de SQL, el cual almacena información acerca de las instrucciones SQL en la base de datos de referencia y también las transforma en código C/C++ remplazando todas estas instrucciones SQL con llamadas a las librerías de UltraLite.

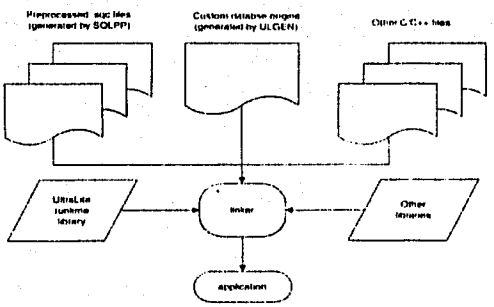
Después que los archivos fuente han sido procesados, se necesita ejecutar el analizador UltraLite que es un programa de Java que se ejecuta dentro de la base de datos

de referencia usando la maquina virtual de Java del Adaptive Server Anywhere. El analizador usa el esquema de la base de datos de referencia y la información almacenada por el preprocesador SQL para crear una base de datos hecha a la medida de la aplicación. El analizador genera un archivo fuente C que constituye el motor de la base de datos personalizada, y debe ser ejecutado cada vez que las instrucciones SQL cambian o cada que cambia el esquema de la base de datos referenciada.



Sería bueno dejar en claro que los datos que se almacenan en la base de datos de referencia son usados únicamente por el analizador para optimizar la base de datos que se va a generar que esta hecha a la medida de la aplicación, y no son usados para llenar dicha base de datos con valores iniciales; el proceso de sincronización será el que llene la base de datos cuando se ejecute la aplicación por primera vez.

Después de que se ejecutan el preprocesador y el analizador, los archivos generados y algún otro archivo fuente definido por el usuario adicional son compilados por el compilador de C/C++ y enlazados con las librerías UltraLite de tiempo de ejecución para formar una aplicación UltraLite.



Hay que tomar en cuenta que el tamaño de la base de datos final es proporcional a la complejidad de la base de datos y al número de instrucciones SQL que la aplicación ejecute.

4.3.2 Fase de instalación y expansión

Así como en la fase de desarrollo existen varios pasos, también los hay para la fase de instalación y expansión de la aplicación UltraLite

1. Preparar la base de datos consolidada
2. Instalar el conducto Mobilink
3. Ejecutar el servidor Mobilink
4. Ejecutar y sincronizar la aplicación

Una vez construida la aplicación UltraLite, es instalada en el dispositivo Palm para su ejecución. Esta aplicación corre utilizando el motor de la base de datos, leyendo y

escribiendo datos en la base de datos que se encuentra en memoria usando la interfase de SQL. La base de datos utiliza la memoria de la Palm para manejar los datos, pero los detalles son completamente ocultados de la aplicación porque la aplicación únicamente trata con la base de datos a través de la interfaz SQL. Y entre otras cosas, la base de datos guarda los cambios que son hechos a los datos para usarlos en la siguiente sincronización.

Una sincronización se puede hacer en cualquier momento que el usuario lo desee y puede ser como parte del HotSync o usando el conducto Mobilink para comunicarse con el servidor Mobilink, también puede ser independiente del HotSync utilizando el protocolo TCP/IP o por medio del puerto serial para hacer esta comunicación. Estos son los modos de comunicación que son conocidos como canales de sincronización, y fuera de algunas inicializaciones simples, la base de datos UltraLite y el servidor Mobilink se encargan de manejar el canal de sincronización sin intervención del usuario.

Algunos modelos de Palm pueden comunicarse con alguna otra Palm a través del puerto infrarrojo. Dos aplicaciones UltraLite puede intercambiar información por medio de esta forma; pero este intercambio de datos no significa sincronizar, y además pudiera causar conflictos en la sincronización. Si este intercambio de información fuera permitida por la aplicación entonces, se necesitarían desarrollar estrategias de resolución de conflictos para poder manejarlos. Por lo tanto para el proyecto actual no se desarrolló ninguna posibilidad de transferir los datos de una Palm a otra.

La sincronización siempre se hace con la base de datos consolidada, la base de datos de referencia es únicamente utilizada en la fase de desarrollo. El servidor Mobilink utiliza una conexión ODBC para la comunicación con la base de datos consolidada.

El primer paso en el proceso de sincronización es subir los cambios realizados en el dispositivo Palm al servidor Mobilink, después el servidor aplica estos cambios en la base de datos consolidada. Todos estos cambios se aplican como si fueran una sola

transacción y se debe tomar en cuenta que se pueden presentar conflictos entre los datos que se subieron y la base de datos consolidada.

Los cambios son transferidos hacia la PC como una serie de registros, y el conjunto de cambios es conocido como el conjunto de datos que se transfiere de la Palm a la PC. La estructura de un registro en el conjunto es fija y definida por el esquema de la tabla apropiada en la base de datos de referencia. Cada registro que se transfiere de la Palm hacia la PC invoca a un script predefinido en la base de datos consolidada.

Un script es un conjunto de sentencias SQL que el servidor Mobilink invoca a la base de datos consolidada para realizar una operación. En este caso, el script posiciona un cursor en la base de datos usando la llave primaria del registro y una vez posicionado, el servidor Mobilink inserta, actualiza o elimina el registro apropiado.

Después de aplicar los cambios del conjunto de datos que se transfirió de la Palm hacia la PC, el servidor Mobilink prepara el conjunto de datos que será transferido de la PC hacia la Palm. El conjunto de datos que se transfiere de la PC hacia la Palm es igual al conjunto de datos que se transfiere de la Palm hacia la PC y que consiste en el conjunto de cambios en la base de datos consolidada y que es transferido hacia la base de datos UltraLite. Y como el primer conjunto, el conjunto de datos que se transfiere de la PC hacia la Palm invoca a un script predefinido, pero este script es invocado una sola vez por tabla para regresar un conjunto resultado de cambios realizados. Cada registro en el conjunto resultado es procesado y aplicado a la base de datos UltraLite. Los cambios son aplicados como una única transacción.

Una vez que los conjuntos de datos han sido procesados, el que se transfirió de la Palm a la PC y el que se transfirió de la PC a la Palm, la base de datos UltraLite y el servidor Mobilink reconocen cada uno la recepción del conjunto de datos y se finaliza el proceso de sincronización.

Si la conexión se pierde durante la sincronización, no hay porque alarmarse, la sincronización UltraLite es muy tolerante para las fallas, es decir, los cambios que no han sido aplicados no se pierden, sino que simplemente se intentan aplicar en la siguiente sincronización. Cada etapa en el proceso de sincronización es controlado por scripts en la base de datos consolidada y no por la misma aplicación.

4.4 EL DESARROLLO DE UNA APLICACIÓN ULTRALITE

Se pueden citar una serie de pasos que serían los requeridos para construir una aplicación UltraLite para la plataforma Palm y son los siguientes:

1. Diseñar la aplicación
2. Hacer un nuevo proyecto en CodeWarrior
3. Crear las bases de datos de referencia y consolidada
4. Definir el esquema de la base de datos
5. Escribir las instrucciones SQL que permiten el acceso a la base de datos
6. Hacer la interfaz con el usuario
7. Construir e instalar la aplicación.

Como con cualquier otra aplicación, en cualquier plataforma, lo primero es plantear el problema y decidir como se va a resolver además de establecer los requerimientos del sistema. Después se necesita crear un nuevo proyecto en CodeWarrior. Los siguientes pasos explican como crear un proyecto en CodeWarrior para construir aplicaciones UltraLite.

1. **Crear un nuevo proyecto multisegmento.** Se utiliza el proyecto multisegmento de Palm OS. Las aplicaciones UltraLite no son excesivamente largas, pero muy raras veces quedan en un solo segmento.

2. **Renombrar el proyecto.** Se debe renombrar el proyecto y asignar un identificador para el proyecto que es conocido con su nombre en inglés Creator ID; y ajustar el nombre de la aplicación y el icono que se va a utilizar para llamarla.

3. **Cambiar algunos valores de compilación.** En la ventana de los valores hay que seleccionar los valores correctos para la compilación.

4. **Cambiar la librería de tiempo de ejecución estándar.** El proyecto multisegmento que se utiliza casi siempre incluye MSL Runtime Palm OS (2i).Lib, y se debe cambiar por MSL Runtime Palm OS (4i).Lib.

5. **Ajustar las rutas de acceso.** Los archivos fuente que UltraLite genera requieren acceso al encabezado de los archivos UltraLite cuando son compilados y a la librería de tiempo de ejecución de UltraLite cuando son enlazados.

6. **Agregar la librería de tiempo de ejecución de UltraLite.** Las rutinas soportadas requeridas por la base de datos UltraLite son encontradas en el siguiente archivo:

   C:\Adaptive Server Anywhere 6.0\ultralite\palm\68k\lib\ulrt.lib

   Y se tiene que agregar este archivo al proyecto

4.5 LA SINCRONIZACIÓN ULTRALITE

Lo primero que hay que entender es que la sincronización UltraLite no involucra a toda la aplicación, es decir, la sincronización ocurre entre la base de datos que esta hecha de acuerdo a las necesidades de la aplicación, o sea, la base de datos UltraLite y la base de datos consolidada. La aplicación usa los datos de la base de datos UltraLite sin saber o preocuparse de donde vienen. Esto es porque se puede desarrollar la aplicación separada de los aspectos de la sincronización.

Un segundo aspecto que hay que resaltar es que a pesar de que la sincronización se realiza entre la base de datos personalizada de la aplicación y la base de datos consolidada, conceptualmente se sincroniza la base de datos de referencia con la base de datos consolidada. La base de datos personalizada de la aplicación es derivada de la base de datos de referencia pero su estructura interna esta completamente oculta. Cuando se escribe la parte de SQL, se usan tablas y columnas de la base de datos de referencia, así cuando se escriben los scripts de sincronización, se hace un mapeo de los datos de la base de datos de referencia a tablas en la base de datos consolidada.

Así es que cuando se habla de sincronización y se nombra a la base datos remota, ésta es siempre la base de datos que se sincroniza con la base de datos consolidada, entonces, podemos decir que la sincronización ocurre en la base de datos consolidada y todo en el proceso de sincronización es relativo a la base de datos consolidada.

Todo el proceso de sincronización es controlado por scripts que se instalan en la base de datos consolidada. Los scripts son instrucciones SQL y pueden ser simples instrucciones de lenguaje de manipulación de datos (Data Manipulation Language DML)

o llamadas a procedimientos almacenados[6] (stored procedures) que realizan acciones mas complicadas.

4.5.1 Scripts y eventos

Cuando se inicia el servidor Mobilink, éste establece una o mas conexiones con la base de datos consolidada y localiza las tablas del sistema UltraLite. Los scripts de sincronización son almacenados en las tablas del sistema y no en el servidor Mobilink, pero el servidor es el que trae los scripts de las tablas del sistema para ejecutarlos.

Una vez que se establece la conexión y los scripts de sincronización son localizados, la sincronización se puede realizar. El proceso de sincronización esta dividido en varios pasos, cada uno de los cuales dispara un evento o notificación. Cuando un evento ocurre, el servidor Mobilink busca en las tablas del sistema el script asociado con el evento. Si el script es encontrado lo ejecuta, sino, se va al siguiente paso en el proceso de sincronización. Ejecutando un script en respuesta a un evento es conocido como que el evento es manejado, sin embargo, hay muchos eventos y la mayoría de ellos son opcionales y solo pocos tienen que ser manejados explícitamente. El gran numero de eventos permite controlar cada aspecto en el proceso de sincronización si así se desea.

Existen dos eventos que ocurren actualmente fuera del proceso de sincronización, el evento *begin_connection* que se dispara inmediatamente después de que el servidor Mobilink se conecta con la base de datos, pero justo antes de que la sincronización comience. El evento end_connection ocurre justo antes de que el servidor Mobilink se desconecta de la base de datos. Estos eventos son disparados por cada conexión que el servidor hace con la base de datos.

[6] Un stored procedure o procedimiento almacenado se ejecuta mediante una llamada a éste, mientras que un trigger es ejecutado cuando se dispara un evento asociado.

4.5.2 Tipos de eventos y parámetros en los scripts

Aparte de *begin_connection* y *end_connection*, los eventos pueden caer en una de las tres categorías siguientes:

- Un evento *cursor event* ocurre cuando un cursor necesita ser abierto. El script asociado es una sentencia SELECT que define el cursor.

- Un evento *table event* ocurre en varias etapas. El script asociado regresa la lógica de sincronización para una tabla remota específica.

- Un evento *connection event* también ocurre en varias etapas. El script asociado regresa la lógica de sincronización para toda la conexión.

El tipo de evento determina el tipo del script. Entonces un evento connection event que es el evento que ocurre cuando se establece una conexión, dispara un script de conexión o connection script, un evento de tabla o table script dispara un script de tabla o un table script. Algunos eventos tienen el mismo nombre pero diferente tipo, por eso es que hay que asegurarse de que evento se está hablando.

De los tres eventos, los eventos de cursor o cursor event, deben ser manejados para realizar aun la mas básica sincronización, y los eventos de conexión y de tabla (connection o table events) son usados en la resolución de conflictos y en estrategias de sincronización complejas y no son requeridos para sincronizaciones básicas.

Cuando un script es ejecutado en respuesta a un evento diferente a begin_connection o end_connection, al script se le pasan uno o mas parámetros. El script usa estos parámetros para personalizar su lógica, por ejemplo, casi cualquier script recibe el nombre del usuario como parámetro.

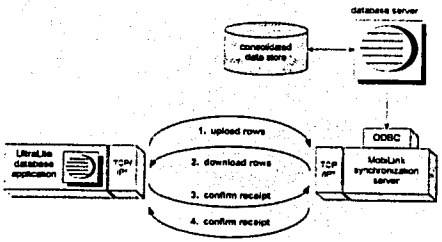
En un script los parámetros se identifican con signos de interrogación cerrados y cada uno de estos es reemplazado con el valor del parámetro antes de que el servidor Mobilink ejecute el script. Por ejemplo, tenemos el evento de conexión begin_synchronization el cual lleva como parámetro un signo de interrogación cerrado, entonces el servidor Mobilink reemplaza este signo de interrogación con el nombre del usuario y entonces ejecuta el script.

Si un evento tiene dos o mas parámetros, el orden de los parámetros determina el orden en que se harán las substituciones en los signos de interrogación en el script. La mayoría de los scripts pueden ignorar los parámetros si se quiere simplificar omitiendo cualquier signo de interrogación, sin embargo, para algunos eventos los signos de interrogación no son opcionales.

Los eventos de conexión tienen un solo parámetro, el nombre del usuario, así como los table events o eventos de tabla tienen dos parámetros: el nombre de usuario y el nombre de la tabla remota. Y los cursor events o eventos de cursor obtienen o el nombre de usuario o una lista de llaves primarias.

4.5.3 El proceso de sincronización

Para la sincronización un dispositivo remoto debe establecer una conexión con el servidor Mobilink, el cual mantiene un juego de conexiones al servidor de la base de datos, o sea a la base de datos consolidada. El proceso de sincronización ocurre en varios pasos como se muestra en el siguiente diagrama:



Para transferir los registros hacia la base de datos consolidada, las aplicaciones UltraLite, preparan y mandan los datos en conjunto, el cual contiene una lista de todos los registros que han sido actualizados, insertados o borrados, en la aplicación UltraLite desde la última sincronización. Y de manera similar para transferir dados hacia el dispositivo, el servidor Mobilink prepara y manda los datos en conjunto, mismo que contiene una lista de registros insertados, actualizados y borrados. Estos dos conjuntos de datos son parte de los pasos 1 y 2, respectivamente en el proceso de sincronización.

El proceso de sincronización se lleva a cabo de la siguiente Manera. Cada paso en la siguiente descripción esta dividida en dos partes, una parte describe las acciones de la aplicación UltraLite, y la otra describe las acciones del servidor Mobilink.

El proceso de sincronización comienza cuando un dispositivo directa (por medio de TCP/IP o conexión serial) o indirectamente (vía HotSync y el conducto Mobilink) hace la petición de sincronización.

1. La aplicación UltraLite automáticamente respalda los registros en el dispositivo remoto que han sido insertados, actualizados o borrados desde la sincronización previa y establece una conexión con el servidor Mobilink, y transfiere estos cambios.

El conjunto de datos que se transfiere a la base de datos central consiste en todos los registros insertados, actualizados y borrados.

El servidor Mobilink recibe y aplica el conjunto de datos en la base de datos consolidada, estos cambios son hechos en una sola transacción, cuando ésta termina el servidor Mobilink acepta la transacción en la base de datos consolidada

2. El servidor Mobilink prepara una lista de registros para ser insertados, actualizados o borrados en la aplicación UltraLite, ejecutando los scripts presentes en la base de datos consolidada. En ese momento los registros son enviados a la aplicación UltraLite en forma de conjunto.

   El servidor Mobilink prepara este conjunto usando una sola transacción en la base de datos consolidada, pero esta no realiza ningún cambio. Se puede usar esta transacción para mantener un histórico de que información a sido enviada a que base de datos remota

   La aplicación UltraLite se asegura que todos los cambios que mando al servidor en el conjunto han sido aplicados en la base de datos consolidada, cuando esta comienza a recibir el conjunto de datos que se le transfirió.

   La aplicación UltraLite automáticamente actualiza su registro de cuales datos no fueron sincronizados, así que los mismos datos no son enviados otra vez, entonces se procesa el conjunto de datos que se transfiere al dispositivo, borrando los registros antiguos, insertando los nuevos y actualizando los que han cambiado. Todos estos cambios se llevan a cabo en una sola transacción y al finalizar son aceptados.

3. La aplicación UltraLite manda un mensaje corto de regreso al servidor Mobilink confirmando que ha recibido y procesado todos los cambios.

El servidor Mobilink aplica la transacción en la base de datos consolidada al recibir estar confirmación.

4. El servidor Mobilink manda a la aplicación UltraLite una confirmación de que la sincronización esta completa.

   La aplicación UltraLite recibe esta confirmación y cierra la conexión.

Hay mas detalles para el proceso de sincronización pero los pasos generales son los enunciados aquí. Varios pasos en una sincronización completa pueden ser agrupados en pasos generales, pasos para subir datos y pasos para bajar datos. Cuando se menciona subir se refiere pasar datos del dispositivo hacia la PC y cuando se menciona bajar se refiere a pasar datos de la PC hacia el dispositivo o sea la Palm.

Los siguientes pasos detallados ocurren durante la sincronización:

1. Se dispara el evento begin_sinchronization (NombreUsuario)[7] y se le pasa el nombre de usuario como parámetro.

2. Por cada tabla remota que es sincronizada, se dispara el evento begin_synchronization (NombreUsuario, NombreTabla)

3. Se ejecuta un commit

4. Se procesa el conjunto de datos que se van a transmitir a la PC usando los pasos que suben los datos (descritos mas adelante)

5. Se ejecuta un commit

[7] Cuando pongo entre paréntesis la palabra NombreUsuario me refiero que cuando se hace la llamada a esa función lleva como parámetro el nombre del usuario.

6. Se prepara el conjunto de datos que se van a transmitir a la Palm usando los pasos que bajan los datos (descritos mas adelante)

7. Se ejecuta un commit

8. Por cada tabla remota que es sincronizada, se dispara el evento end_synchronization (NombreUsuario, NombreTabla)

9. Se dispara el evento end_synchronization

10. Y por último se ejecuta un commit

Hay que resaltar que el servidor Mobilink aplica los cambios pendientes en varios puntos a través del proceso de sincronización. Ninguno de los eventos disparados por estos pasos tienen que ser manejados.

4.5.4 Pasos para transferir los datos del dispositivo hacia la PC

Los siguientes pasos ocurren cuando se procesa el conjunto de datos que se van a subir a la PC:

1. Se dispara el evento begin_upload (NombreUsuario), que como se puede apreciar lleva como parámetro el nombre del usuario.

2. Por cada tabla remota, se dispara el evento begin_upload (NombreUsuario, NombreTabla)

3. Por cada tabla remota:

    a. Se dispara el evento begin_upload_rows(NombreUsuario, NombreTabla)

b. Por cada renglón nuevo o que ha cambiado y que se ha subido a la PC desde una tabla remota

   i. Se abre el cursor definido por upload_cursor(pk1, ..., pkn), donde pk1 hasta pkn son las llaves primarias usadas por el cursor, buscando por un renglón existente que coincida con los valores de la llave primaria.

   ii. Si el renglón es nuevo se inserta usando este cursor

   iii. Si el renglón es cambiado y no hay conflictos se actualiza el renglón.

   iv. De otro modo, se maneja el conflicto abriendo dos nuevos cursores, old_row_cursor(pk1, ..., pkn) y el new_row_cursor(pk1, ..., pkn), e insertando el renglón anterior (el renglón tal y como aparece en la base de datos) usando old_row_cursor. Entonces se dispara el evento resolve_conflict (NombreUsuario, NombreTabla) para resolver los conflictos.

c. Se dispara end_upload_rows(NombreUsuario, NombreTabla)

4. Por cada tabla remota, en orden contrario

   a. Se dispara begin_upload_deletes (NombreUsuario, NombreTabla)

   b. Por cada renglón borrado de la tabla remota

      i. Se abre el cursor definido por upload_cursor(pk1, ..., pkn)

ii. Se borra el renglón usando este cursor

c. Se dispara end_upload_deletes (NombreUsuario, NombreTabla)

5. Por cada tabla remota, se dispara end_upload (NombreUsuario, NombreTabla)

6. Se dispara end_upload (NombreUsuario)

Como se puede apreciar, procesando el conjunto de datos que se van subir a la PC es mucho mas complicado que un simple conjunto de pasos. Sin embargo, el único evento que se requiere manejar es el evento upload_cursor, aunque se debe manejar también old_row_cursor, new_row_cursor, y resolve_conflict para manejar los conflictos.

4.5.5 Pasos para transferir los datos de la PC al dispositivo (Palm)

Los pasos siguientes son para producir el conjunto de datos que se va a bajar hacia el dispositivo

1. Se dispara el evento begin_download (NombreUsuario)

2. Por cada tabla remota, se dispara el evento begin_download (NombreUsuario, TablaUsuario)

3. Por cada tabla remota:

   a. Se dispara begin_download_deletes (NombreUsuario, NombreTabla)

   b. Se abre el cursor definido por download_delete_cursor (NombreUsuario) y se agregan los registros que componen el resultado al conjunto de datos que se va a bajar al dispositivo. Estos registros serán eliminados de la

tabla remota cuando el dispositivo procese el conjunto de datos que se haya bajado.

c. Se dispara end_download_deletes (NombreUsuario, NombreTabla)

d. Se dispara begin_download_rows (NombreUsuario, NombreTabla)

e. Se abre el cursor definido por download_cursor (NombreUsuario) y se agregan los registros resultantes al conjunto de datos que se bajaran al dispositivo. Estos registros serán insertados o actualizados en la tabla remota cuando el dispositivo procese el conjunto de datos que se hayan bajado.

f. Se dispara end_download_rows (NombreUsuario, NombreTabla)

4. Por cada tabla remota, se dispara end_download (NombreUsuario, NombreTabla)

5. Se dispara end_download (NombreUsuario)

6. Se espera a que el dispositivo reciba y aplique el conjunto de datos que se bajó.

7. Si el conjunto de datos se aplicó con éxito, se ejecuta un commit y se envía la aceptación al dispositivo.

8. De otro modo, se ejecuta un Rollback para deshacer los cambios.

4.5.6 La base de datos consolidada

Las aplicaciones UltraLite se sincronizan con una base de datos central llamada base de datos consolidada. El esquema de esta base puede y debe ser similar al de la base

de datos contenida en una aplicación remota, o sea, la que se encuentra en el dispositivo, de hecho, una forma de diseñar la base de datos de la aplicación es usar el esquema de la base de datos consolidada como el punto de partida. Regularmente en la base de datos consolidada se encuentran tablas que corresponden a cada tabla de la base de datos remota.

Mientras la base de datos de referencia que es usada para construir la aplicación debe ser una base de datos de Adaptive Server Anywhere. Se puede construir la base de datos consolidada usando cualquier producto que pueda usar ODBC. Se puede utilizar algún producto de Sybase como Adaptive Server Anywhere o Adaptive Server Enterprise[8], o algún otro producto de otra compañía así como Oracle o Microsoft SQL Server. Y también se puede usar una sola base de Adaptive Server Anywhere que pueda tener ambas funciones, el de base de datos consolidada y el de la base de datos de referencia.

4.5.7 Las tablas del sistema Mobilink

Las acciones del servidor Mobilink se dirigen mediante los scripts, estos scripts son almacenados en tablas que deben residir en la base de datos consolidada. Los scripts de sincronización son escritos en el lenguaje SQL de la base de datos consolidada. También existe una tabla adicional que contiene información acerca de cada usuario remoto.

Una característica importante de una aplicación UltraLite es que la sincronización entre la base de datos personalizada de la aplicación y la base de datos consolidada parece transparente. El proceso de sincronización involucra la ejecución de varios scripts por el servidor Mobilink. Estos scripts son las instrucciones SQL ejecutadas por el servidor para actualizar o consultar la base de datos consolidada y son instalados en esta base de

[8] Es una herramienta de Sybase que se utiliza para manejar bases de datos que se encuentran en un servidor.

datos junto con otra información y se almacenan en un conjunto de tablas llamadas tablas del sistema Mobilink.

Las tablas del sistema Mobilink son instaladas siempre que se crea una base de datos nueva con el Adapative Server Anywhere, pero deben ser creadas explícitamente cuando se crea la base de datos con otros sistemas.

Sybase cuenta con scripts SQL para crear las tablas del sistema Mobilink y pueden ser usados algunos de los tres siguientes sistemas de bases de datos: Sybase Adaptive Server Enterprise, Microsoft SQL Server y Oracle. Los scripts pueden ser modificados si así se desea y solo es necesario que sean instalados una sola vez en cualquier base de datos dada para crear dichas tablas.

Una vez que las tablas están creadas, entonces se escriben los scripts de sincronización. Para poder acceder a estos scripts se utiliza la versión Java de Sybase Central.

Primero es necesario conectarse a Sybase Central en donde después se hace la conexión a la base de datos consolidada. Una vez conectado, Sybase Central permite ver las tablas que son definidas en la base de datos, crear y editar los scripts de sincronización.

4.5.8 Como están relacionadas las tablas de la base de datos remota con la base de datos consolidada

Los diseños de sincronización pueden especificar alguna correspondencia arbitraria entre tablas y registros en la base de datos remota, y tablas y registros en la base de datos consolidada. La única restricción es que las columnas en la base de datos consolidada deben ser compatibles en cuanto al tipo de datos con las de la base de datos de la aplicación.

Las tablas presentes en la base de datos remota no necesitan existir en la base de datos consolidada. Los datos sincronizados en una tabla de la aplicación remota pueden ser distribuidos entre columnas en diferentes tablas y aún en diferentes bases de datos consolidadas. Estas relaciones se especifican utilizando los scripts de sincronización.

Hay tablas en la base de datos consolidada que frecuentemente tiene columnas extra que las tablas en la base de datos UltraLite no tienen y que no son sincronizadas, pero que pueden ayudar en la sincronización, por ejemplo, una columna del tipo timestamp que es un tipo de datos que guarda un valor de tiempo, o sea fecha y hora, puede identificar registros nuevos o que hayan sido actualizados en la base de datos consolidada; o en otros casos, simplemente existen columnas extra o tablas que contienen información que no es requerida en los dispositivos remotos (Palm).

4.6 EL SERVIDOR MOBILINK

Todas las aplicaciones UltraLite se pueden sincronizar a través del servidor Mobilink, y ninguno se puede conectar directamente con el servidor de la base de datos. A menos que se este usando HotSync se debe iniciar el servidor Mobilink antes de que la aplicación UltraLite haga la petición de sincronización.

El servidor Mobilink abre las conexiones vía ODBC con la base de datos consolidada, esta acepta las conexiones de las aplicaciones remotas y controla el proceso de sincronización.

Para iniciar el servidor Mobilink:

Se debe ejecutar el comando dbssrv6, localizado en el directorio win32 de la instalación del Adaptive Server Anywhere y se usa el interruptor –c para especificar los parámetros de conexión ODBC de la base de datos consolidada, o también se pueden utilizar algunos otros parámetros, pero son opcionales. Por ejemplo el comando siguiente

inicia el servidor Mobilink, identificando la fuente de datos ODBC CustDB como la base de datos consolidada:

```
Dbssrv6 -c "DSN=UltraLite Sample: UID=dba; PWD=sql"
        -o ulsync.log
        -vcr
        -x tcpip
```

Los comandos anteriores usan el interruptor -o para especificar que el archivo log debe ser nombrado ulsync.log. El interruptor –x especifica que la aplicación UltraLite se conectara vía TCP/IP, y en el caso que se quisiera utilizar una conexión de tipo serial se debe remplazar la palabra tcpip por la palabra *serial.*

Aquí se muestra como es que se inicia el servidor Mobilink antes de sincronizar la base de datos que se conecta por el DSN lsifca2000[9]:

```
dbssrv6 -x serial{port=2}  -vcrst -c "dsn=lsifca2000;uid=dba;pwd=sql" -o "ul_sifca.log"
```

4.6.1 Como se manejan los intentos de sincronización fallidos

La sincronización UtraLite es bastante tolerante, si una vía de comunicación falla durante la sincronización la base de datos UltraLite y la base de datos consolidada se quedan en un estado consistente y el valor SQLCode es establecido a SQLE_COMMUNICATION_ERROR en la aplicación UltraLite cuando la función ULSinchronize.

[9] Nombre que se le dio al DSN por el que se hace la conexión con la base de datos consolidada.

Hay tres casos que son manejados en diferentes formas:

- ***Falla durante la transferencia de datos de la Palm hacia la PC.*** Si la falla ocurre mientras se construye el conjunto de datos que será transferido hacia la PC, la base de datos UltraLite se queda exactamente en el mismo estado como al inicio de la sincronización. En el servidor Mobilink, si alguna parte del conjunto de datos ya ha sido guardada entonces se deshacen los cambios.

- ***Falla entre la transferencia de datos del dispositivo hacia la PC y la transferencia de la PC hacia el dispositivo.*** Si la falla ocurre una vez que se ha completado la transferencia de datos del dispositivo hacia la PC pero antes de que la aplicación UltraLite reciba el conjunto de datos que le será transferido. La base de datos UltraLite se queda en un estado desconocido, mientras que el conjunto de datos que ya fue previamente transferido al servidor puede ya haber sido aplicado o pudo haber ocurrido la falla antas de que fuera aplicado.

  La siguiente vez que la base de datos UltraLite se sincroniza determina el destino del conjunto de datos anterior antes de construir un nuevo conjunto de datos. Si el conjunto de datos anterior no fue aplicado, el nuevo conjunto de datos contiene todos los cambios contando los del conjunto de datos anterior.

- ***Falla durante la transferencia de datos de la PC hacia el dispositivo.*** Si la falla ocurre mientras se están aplicando los cambios del conjunto de datos que se transfirió hacia la Palm, cualquier parte de este conjunto que ya haya sido aplicado será deshecho y la base de datos UltraLite se quedará con el mismo estado que tenía antes de que se le transfiriera el conjunto de datos.

4.6.2 Como se procesa el conjunto de datos que se transfirió de la Palm a la PC

Cuando el servidor Mobilink recibe un conjunto de datos de la base de datos UltraLite en una sincronización, todo este conjunto es mantenido en memoria hasta que la sincronización se complete. Esto es hecho por tres razones:

- ***Bloqueo.*** Cuando un conjunto de datos esta siendo aplicado a la base de datos consolidada se puede encontrar algún bloqueo debido a la concurrencia con otras transacciones. Estas transacciones pueden ser de otros servidores Mobilink o transacciones de otras aplicaciones que están utilizando la base de datos consolidada. Cuando una transacción en la base de datos consolidada es bloqueada, es deshecha y el servidor Mobilink automáticamente comienza otra vez con la transacción.

  Es importante escribir los scripts de sincronización de tal manera que se trate de evitar una disputa lo mas que se pueda, ya que una disputa para ver quien se conecta con la base de datos consolidada tiene un impacto bastante significante en el desempeño de la sincronización cuando múltiples usuarios se sincronizan simultáneamente.

- ***Filtrando registros que son transferidos hacia la Palm.*** La técnica mas común para determinar que registros serán transferidos es, transferir aquellos registros que han sido modificados desde la transferencia anterior. Cuando hay una sincronización, la transferencia de datos de la Palm hacia la PC es primero y después sigue la transferencia de la PC hacia la Palm. Entonces cualquier registro insertado o actualizado durante la transferencia hacia la PC serán registros que han sido modificados desde la transferencia anterior y otra vez serían transferidos estos registros hacia la Palm. Además si tomamos en cuenta que sería difícil escribir el script download_cursor que omita estos registros para que no sean

transferidos otra vez hacia la Palm, entonces existe la función del servidor Mobilink que automáticamente filtra los registros que acaban de ser transferidos hacia la PC para que no sean transferidos hacia la Palm de nuevo.

Cuando un registro es agregado al conjunto de datos que será transferido de la PC hacia la Palm, el servidor Mobilink busca el registro en el conjunto de datos que acaba de ser transferido de la Palm hacia la PC y elimina el registro del conjunto de datos que será transferido hacia la Palm cuando lo encuentra en el otro conjunto.

- ***Procesando los registros eliminados después de inserciones y actualizaciones.*** El conjunto de datos es aplicado en la base de datos consolidada en un orden que evita violaciones de integridad referencial. El conjunto de datos es formateado de tal forma que todas las operaciones (inserciones, actualizaciones y borrados) para una sola tabla son agrupadas. Las tablas, cuando se transfiere el conjunto de datos hacia la PC, son ordenadas en base a las relaciones por llaves foráneas. Y todas las tablas en la base de datos UltraLite que son referenciadas por otra tabla en la base de datos irá antes en la transferencia del conjunto de datos que la tabla que hace la referencia.

  Cuando el conjunto de datos que se transfirió de la Palm hacia la PC es aplicado a la base de datos consolidada, las inserciones y actualizaciones son aplicadas en el orden en que aparecen en el conjunto de datos.

Cuando un registro insertado o actualizado hace referencia a otro insertado, entonces se asegura que el registros referenciado será insertado antes que el registro que hace la referencia. Las eliminaciones son aplicadas en el sentido opuesto y después de que todas las inserciones y actualizaciones hayan sido aplicadas. Un registro es borrado cuando hace referencia a otro que haya sido borrado anteriormente. Esto asegura la

integridad referencial y entonces se aplica el orden de que el registro que hace la referencia es borrado primero que el registro referenciado.

4.6.3 Resolución de conflictos

Cuando un registro es actualizado en la tabla remota, el dispositivo manda dos copias del registro al servidor Mobilink: una copia tiene los valores anteriores del registro antes de actualizarlo, mientras la segunda copia tiene los nuevos valores del registro después de la actualización. Cuando el servidor Mobilink recibe estos registros, compara el valor anterior con los valores del registro almacenados en las tablas de la base de datos consolidada. Si son iguales, el registro en la tabla de la base de datos consolidada es actualizado con los nuevos valores de la tabla de la base de datos remota y no ocurren errores. Si no son los mismos, otra aplicación ha actualizado el registro y un conflicto es detectado.

El servidor Mobilink resuelve conflictos invocando tres scripts: old_row_cursor para almacenar los valores anteriores del registro. El script new_row_cursor para almacenar los nuevos valores del registro y resolve_conflict para resolver el conflicto. Ambos scripts old_row_cursor y new_row_cursor definen cursores similares a los definidos por upload_cursor, pasando la llaves primarias como parámetros.

No hay comportamientos definidos por defecto para cualquiera de estos scripts, porque no hay modo que el servidor Mobilink o aun el analizador UltraLite puedan determinar la estrategia correcta para la resolución del conflicto.

La estrategia mas simple sería dejar ganar a un lado siempre, o sea que, o los nuevos valores de la tabla remota sobrescriben lo que ya existe en la base de datos consolidada, o los nuevos valores son ignorados y los valores actuales en las tablas de la base de datos consolidada son preparados para ser bajados hacia la tabla remota del dispositivo.

Estrategias mas complicadas usarían el nombre del usuario o algunos otros datos lógicos para determinar que lado gana y sobrescribe al otro.

Como un ejemplo de resolución de conflictos se pueden definir dos tablas globales temporales para almacenar los nuevos y los viejos valores para determinado registro. En estas tablas se almacenaría los registros únicamente en el momento de la conexión y después serían borrados al terminar ésta.

Cuando el script de resolve_conflict es llamado, éste usa uno de los nuevos registros para determinar si los nuevos valores del registro deben sobrescribir a lo que se encuentra en la base de datos consolidada. Y después las tablas temporales son limpiadas, dejándolas vacías para usarlas en el siguiente conflicto en la misma conexión.

4.6.4 Contenedor de llaves primarias

Los conflictos también pueden ocurrir si dos tablas remotas insertan nuevos registros con idénticas llaves primarias. Esto es fácilmente resuelto con un contenedor de llaves primarias. Un contenedor de llaves primarias es una tabla que asigna un número fijo de valores no usados de llaves primarias a cada usuario. La tabla contenedora es sincronizada usando una estrategia de partición, así que cada aplicación tiene acceso a sus valores de llaves primarias no usados. Cuando la aplicación inserta un nuevo registro, obtiene una llave primaria de la tabla contenedora. La llave primaria ya no se vuelve a usar así que el registro apropiado es borrado de la tabla contenedora. En la siguiente sincronización, los scripts en la base de datos consolidada revisan si la tabla contenedora necesita ser llenada con valores de llaves primarias.

4.7 MANEJANDO LOS ERRORES

Un error en los scripts de sincronización ocurre cuando una operación en el script falla cuando el servidor Mobilink lo esta ejecutando. El sistema manejador de la base de

datos regresa un valor (un código SQL) al servidor Mobilink indicando la naturaleza del error. Cada base de datos consolidada tiene su propio conjunto de valores para los códigos SQL que regresan los scripts cuando ocurre un error.

Si un error ocurre mientras el servidor Mobilink esta ejecutando un script, este invoca el script handle_error, pasándole cuatro valores: el código de error de SQL (que es un entero), el mensaje de error, el nombre de usuario de la sincronización y el nombre de la tabla (el cual puede ser un valor Nulo si no fue pasado un nombre de tabla al script original). El script handle_error invoca un stored procedure para manejar el error, como sigue:

CALL MyErrorProcedure (?, ?, ?, ?, ?)

El primer parámetro para el stored procedure es un parámetro de salida del tipo entero, usado para regresar un código de acción al servidor Mobilink. Un modo alterno para llamar el strored procedure es

? = CALL MyErrorProcedure (?, ?, ?, ?)

En este caso, el primer parámetro indica que el valor de retorno del stored procedure es el código de acción.

El servidor Mobilink actualmente entiende tres códigos de acción:

- 1000 significa ignorar el registro actual y continuar procesando con el siguiente registro.

- 3000 significa deshacer la transacción actual y cancelar lo que falta de la sincronización.

- 4000 significa deshacer la transacción actual y dar de baja el servidor Mobilink.

La acción por defecto si no hay un script definido es 3000.

Si lo que se desea es que en un script se guarde información acerca del error en la base de datos consolidada antes de regresar un código 3000 o un código 4000, entonces se debe primero deshacer la transacción actual, hacer los cambios, y aceptar entonces la transacción. De otro modo, lo almacenado será desecho cuando el manejador del error regrese el código de acción.

Algunas de la acciones que se pueden realizar cuando se esta manejando un error son:

- Detallar el error en una tabla separada

- Decirle al servidor Mobilink si se ignora el error y se debe continuar o se debe dar marcha atrás a la sincronización o inclusive dar marcha atrás a la sincronización y dar de baja el servidor Mobilink

CAPITULO 5

CODEWARRIOR

5.1 CARACTERÍSTICAS GENERALES

Como ya se ha mencionado anteriormente la herramienta CodeWarrior fue utilizada para desarrollar éste proyecto. CodeWarrior es mas que una herramienta, es el ambiente de desarrollo oficial para el sistema operativo Palm, y permite la creación de programas en ANSI C y C++ en plataformas como Windows 95/98/NT o Macintosh.

Desarrollar para el sistema operativo Palm OS es de algún modo similar a desarrollar para cualquier otra plataforma y en otros sentido un poco diferente. Dos importantes similitudes son las siguientes:

- Aplicaciones que son manejadas por eventos
- Se puede utilizar cualquier código que esté basado en el estándar C

Las diferencias son las características cruciales como el tamaño del dispositivo y el propósito. Estos incluyen como el Palm OS maneja:

- Los requerimientos de memoria
- El almacenamiento de la misma aplicación así como sus datos
- La conectividad del dispositivo con una computadora de escritorio

Lo mas importante que se debe recordar es que las relaciones entre el dispositivo y el sistema operativo son extremadamente limitadas. Todo has sido construido bajo la premisa que el dispositivo es una extensión de la computadora de escritorio.

La interfaz de usuario permite únicamente una aplicación ejecutándose a la vez. De tal manera, que cuando la aplicación es abierta o ejecutada, ésta tiene el control de la pantalla completa.

Todas las aplicaciones corren en una interfaz de usuario de un solo Thread y no pueden ser Multi-Thread por si mismas.

5.1.1 Como se maneja la memoria

La memoria es manejada en un inusual estilo. La memoria RAM en un dispositivo Palm OS es usada por dos propósitos:

Memoria para acceso dinámico. Esta es la memoria que cualquier aplicación o inclusive el sistema necesita mientras están corriendo. También incluye la pila de almacenamiento que la aplicación requiere. Cuando ocurre un reset, esta memoria se limpia. Esta memoria es similar a lo que es la memoria RAM en un sistema operativo tradicional.

Memoria para almacenamiento permanente. Esta incluye las aplicaciones que se han ido instalando, así como los datos que el usuario utilizará en el dispositivo. Las tareas, nombres, direcciones, notas y números telefónicos son los que utilizan esta memoria. Cuando ocurre un reset, ésta memoria no se borra. Esta memoria es similar a lo que son los archivos en un disco duro en un sistema operativo tradicional.

Para ambos tipos de memoria, el acceso es hecho como pedazos de memoria llamados chunks. El almacenamiento permanente contiene bases de datos, con pedazos de memoria relacionados (chunks) que son contenidos en una sola base de datos. Por ejemplo, todos las notas son almacenadas (cada una como un pedazo separado) en una

sola base de datos. Y también, por ejemplo, otra base de datos contiene todos los registros de la aplicación "Directorio[1]".

5.2 DESARROLLO PARA DISPOSITIVOS PORTATILES

Existen muchas herramientas de desarrollo disponibles para la plataforma Palm. Estas herramientas permiten escribir el código en lenguaje C y que están basados en paquetes que ya tienen algunas funciones y únicamente necesitan una pequeña cantidad de scritping (código). De ésta gama de posibilidades, se puede elegir la herramienta correcta para el tipo de aplicación que se quiere crear.

Para el desarrollo de una aplicación Palm, se puede escribir el código en plataformas como Windows 95/98/NT, Unix o Macintosh. El ambiente de desarrollo oficial para Palm, CodeWarrior, está disponible para ambos, Windows y Macintosh. Programadores para Unix y Windows tienen acceso a un conjunto de herramientas disponibles como son GNU, compilador C o GCC y hay dos paquetes de desarrollo de formas basado en Windows, y por último se puede programar en 68K assambler o usar una propiedad de lenguaje llamado CASL.

5.2.1 GCC

Existe una amplia tradición en la comunidad de desarrollo de software donde las herramientas, incluyendo compiladores deben ser gratuitos. Una figura sobresaliente en la búsqueda de herramientas de programación gratuitas es la Free Software Foundation (Fundación de software gratuito). Voluntarios de esta fundación son los que han sido responsables de crear algunos de los mejores compiladores que existen. Uno de los mejores que han sido creados por este esfuerzo es GCC (el compilador GNU C) un

[1] Es una de la aplicaciones básicas que contiene una Palm y es utilizada para guardar direcciones y teléfonos.

compilador general de C/C++. Este compilador es uno de los mas amplios compiladores usados en Unix y que también es usado en la plataforma Win32.

Los voluntarios de esta fundación han creado compiladores para varias plataformas y ponen disponible el código fuente bajo la condición de que cualquier modificación debe también ser distribuida gratuitamente.

Cuando se puso a la venta uno de los primeros modelos de Palm que fue la Pilot 1000, el único ambiente de desarrollo fue CodeWarrior que corría únicamente en sistema operativo Mac. Muchos programadores de Unix y Windows querían desarrollar aplicaciones para Palm, pero no estaban dispuestos a comprar una Macintosh, entonces, una colección de herramientas fue puesta a la venta en lo que es llamado oficialmente como GNU PalmPilot SDK, sin embargo, a esta colección completa se le conoce como GCC.

5.2.2 GNU PalmPilot SDK

Esta colección de herramientas SDK es lo que permite crear aplicaciones Palm OS en C/C++ para Unix o Windows, que incluye:

GCC. La mas importante de estas herramientas es el compilador GNU C, que es llamado propiamente GCC y que compila código C/C++ a Motorola 68K

GDB. Otra herramienta muy importante es el debugger a nivel de código fuente, que es llamado GDB.

PilRC. Este recurso de compilación crea recursos Palm de descripciones textuales de los recursos. Estos archivos de texto contienen descripciones de los recursos y tienen extensión .RCP

PilrcUI. Esta aplicación lo que hace es desplegar los archivos RCP gráficamente, y muestra como es que se ven estos archivos en el portátil.

Copilot. Esta aplicación emula el dispositivo Palm en nivel de hardware. Requiere una imagen ROM de un dispositivo Palm actual y actúa casi como una Palm.

Existen muchas fuentes en la red (Internet) para GCC, dependiendo si se quiere GCC para Unix o para Windows. Se pueden conseguir todas las partes de una vez, pero para bajarlo de Internet se necesitará una buena cantidad de tiempo, ya que es un gran paquete de casi 15 MB. El lugar en el que se piensa primero para bajar GCC de Internet es *http://www.palmcentral.com.*

Si se utiliza GCC, es necesario pensar en que es lo que se debe hacer con la parte del conducto (conduit) cuando se desarrolle una aplicación. Se tienen dos opciones, se puede comprar el Kit de herramientas para conductos SDK, o se puede usar el simple conducto de respaldo[2] (backup) que Palm ya tiene.

5.3 AMBIENTES DE DESARROLLO PARA PALM

5.3.1 Assembler SDK (ASDK)

Esta herramienta para el desarrollo del software permite el desarrollo de aplicaciones en ensamblador Motorola 68K. Incluye ensamblador Pilot que para mucha gente esto sería una completa agonía, pero aparentemente, algunos desarrolladores disfrutan escribir aplicaciones en lenguaje ensamblador. Pero la ventaja de esto es que es gratuito .

[2] Cuando se realiza una sincronización por medio de HotSync se respaldan todas las aplicaciones que estén instaladas en el dispositivo automáticamente, sin tener que configurar nada.

5.3.2 JUMP

Este ambiente novedoso permite escribir aplicaciones en Java usando una librería de clase Palm y el ambiente de desarrollo Java favorito. Jump entonces compila los archivos resultantes en Java (.class) en código Motorola 68K. Jump incluye una librería muy pequeña que es usada en tiempo de ejecución y que provee soporte Java como una colección.

El único aspecto decepcionante de Jump es que el sistema operativo Palm no es completamente soportado, por ejemplo, cualquier llamada a una función que deba regresar algún valor como un parámetro, no funcionaría en Jump.

El que se puede considerar como padre de este ambiente de desarrollo es Greg Hewgill y se puede obtener información de Jump en el sitio http://www.hewgill.com. Este ambiente de desarrollo es gratuito, y el código fuente también es proveído junto con la información de este ambiente

5.3.3 CASL

Este paquete comercial provee soporte para la plataforma. Se escribe una aplicación en lenguaje CASL una sola vez y después se distribuye en el sistema operativo Palm o en otro sistema operativo.

Este ambiente ofrece una gran facilidad para la dispersión de la aplicación entre plataformas, es decir, que se escribe la aplicación en un lenguaje para múltiples plataformas. El código es compilado en código p para una maquina virtual. Existe una maquina virtual para el sistema operativo palm y otra para Windows CE[3]. El desarrollo de aplicaciones es simple como si fuera usar directamente lenguaje C o C++.

[3] Windows CE es una versión del sistema operativo Windows para dispositivos portátiles

CASL corre solamente bajo Windows y este, a diferencia de los anteriores, no es gratuito aunque existe una versión demo disponible en la dirección http://www.caslsoft.com.

5.3.4 Desarrollo de formas de alto nivel

Los dispositivos Palm son tan numerosos y las aplicaciones son tan populares que existen ambientes de desarrollo especializados en crear aplicaciones Palm basadas en formas.

5.3.4.1 Pendragon forms

Esta aplicación Windows provee una forma muy sencilla de crear formas simples multipágina, que contienen campos de texto, casillas de verificación, botones de opción, etc. Pendragon Forms también tiene un conducto que automáticamente transfiere los datos de la Palm hacia la PC en archivos con datos separados por coma, estos archivos pueden ser fácilmente importados a hojas de calculo o programas manejadores de bases de datos.

Pendragon Forms tampoco es gratuito y se puede encontrar información acerca de este ambiente en la dirección http://www.pendragon-software.com

5.3.4.2 Satellite Forms

Satellite Forms que es de la empresa Puma Technology, es un ambiente de desarrollo donde se crean aplicaciones muy sofisticadas para el sistema operativo Palm. En Satellite Forms la aplicación consiste de un número de tablas y formas. Cada forma esta ligada a una tabla especifica y despliega los elementos de esa tabla.

En vez de usar código C/C++ se controlan las acciones de la aplicación en una de las dos formas siguientes:

- Se especifica un número de acciones predefinidas que ocurren cuando el usuario realiza un tap sobre un control. Cuando un botón es presionado podría ser la petición de que una nueva forma sea abierta o que se regrese a la forma previa.

- Se especifica algún código que se quiera ejecutar, este código es creado usando un lenguaje script que es muy similar a Visual Basic.

La aplicación tiene un número de controles predefinidos así como una librería de rutinas. Satellite forms también tiene una extensión que es un mecanismo que permite escribir código en C para nuevos controles y nuevas librerías, por ejemplo, una librería que contenga rutinas financieras o un nuevo control definido por el usuario.

Satellite Forms tiene un control ActiveX[4] que es conectado a un conducto HotSync. Se puede usar el control ActiveX durante la sincronización HotSync para copiar cualquier tabla desde o hacia el dispositivo Palm. Las tablas son guardadas en la computadora de escritorio como archivos .DBX que pueden ser fácilmente integrados con cualquier base de datos.

A principios de 1999, el precio de Satellite Forms era 369 dlls mas aparte la licencia para el motor de tiempo de ejecución. Se tienen también un par de requerimientos, y el primero es que solo corre en Windows y las aplicaciones que se crean requieren librerías de tiempo de ejecución en el dispositivo Palm. Hay un cargo a la licencia de la librería de tiempo de ejecución, lo que puede hacer que la distribución de aplicaciones construidas con Satellite Forms sea demasiado cara. Existe una versión

[4] Son controles previamente desarrollados que cuentan con cierta funcionalidad y que están basados principalmente en controles comunes y elaborados por varios de éstos.

demo la cual limita el numero y tamaño de tablas y esta disponible en el sitio http://www.pumatech.com.

Existen algunas cosas que no se pueden hacer con Satellite Forms, por ejemplo, no se tiene un control directo de los eventos, no se pueden especificar los elementos del menú y los campos de texto tienen una longitud máxima, también, puede ser un poco difícil crear una interfaz de usuario muy especializada.

Existen productos comerciales de gran calidad que han sido construidos con Satellite Forms como por ejemplo:

- Punch List desarrollado por la empresa Strata Systems, que es un proyecto para el manejo de software para la industria de la construcción.

- Real Estate Companion desarrollado por la empresa Mobile Generation Software, que es un cliente y maneja la información apropiada para profesionales.

- Helpdesk on the Go desarrollado por la empresa Kerem Krikpinar, que es un servicio para soporte técnico.

5.4 CODEWARRIOR PARA EL SISTEMA OPERATIVO PALM

El ambiente de desarrollo oficial para el sistema operativo Palm y el elegido para el proyecto de esta tesis es Metrowerk's CodeWarrior para Palm OS. Este ambiente de desarrollo comercial permite crear programas en ANSI C o en C++, ya sea en Windows 95/98/NT o en Macintosh, además, actualmente incluye un kit de software para el desarrollo de conductos Palm, y la documentación de Palm asume que se esta usando este paquete. CodeWarrior para Palm OS esta disponible en una suscripción básica por un año

y actualizaciones gratis. Aquí se presenta una descripción de las herramientas del CodeWarrior ofrece para el desarrollo en Palm OS.

El Constructor de Metrowerk's es un editor gráfico que se puede usar para crear los elementos de la interfaz de usuario de la aplicación.

5.4.1 Ambiente de Desarrollo CodeWarrior

Este es una herramienta que incluye:

- Un compilador motorola 68000 C/C++
- Un enlazador (linker)
- PalmRez (formalmente llamado PilotRez), el cual crea los archivos .PRC del código compilado 68000 y convierte los recursos que están en formato del constructor a formato PRC.

5.4.2 CodeWarrior Debugger

Por este programa trabaja nivel de código fuente y es utilizado para realizar un debug[5] a las aplicaciones Palm OS. Puede realizar un debug a una aplicación que se esta ejecutando en el dispositivo Palm que esta conectado a la PC por medio del cable serial, también realiza un debug a aplicaciones que corren en el emulador (POSE).

[5] Esta operación consiste en una ejecución del programa que se esté desarrollando pero revisando cada una de las instrucciones que lo forman y examinando los valores que va tomando en dicha ejecución.

5.4.3 Kit de Desarrollo de Software Palm

Incluye archivos de cabecera, documentación, un tutorial e invaluable código de ejemplo. Los ejemplos incluyen el código fuente de aplicaciones preconstruidas como lo son el directorio, las tareas, etc.

5.4.4 Kit de Desarrollo de Software para Conductos

Este kit es usado para crear conductos y esta disponible separado de CodeWarrior, pero normalmente viene incluido como una cortesía. Esta herramienta requiere Microsoft Visual C++ para Windows para crear conductos de Windows, y para crear conductos de Macintosh se necesita CodeWarrior para Mac OS.

CodeWarrior fue originalmente un ambiente de desarrollo para Macintosh que ha sido adecuado a Windows. Muchos usuarios de Windows dicen que CodeWarrior no tiene una apariencia de Windows mas bien su apariencia es de un producto de Macintosh y algunos métodos abreviados no funcionan como se espera.

5.5 HERRAMIENTAS DE PALM COMPUTING

Palm ofrece también un gran conjunto de herramientas para el desarrollo de aplicaciones Palm, como son:

POSE. Esta aplicación es similar a lo que es Copilot, ya que sirve como un reemplazo de un dispositivo Palm OS mientras se realiza el desarrollo de una aplicación. Puede cargar una imagen ROM del disco y emula diferentes versiones del sistema operativo Palm. Mediante esta aplicación (Emulador) se puede probar la aplicación que se está desarrollando, nada mas que hay que tener en cuenta, que la prueba final de la aplicación, debe hacerse en un dispositivo Palm antes de que se distribuya y no solamente con el emulador.

Debug ROMs. Existen las imagenes 2.0 y 3.0 OS ROM que se pueden usar con POSE. Estas no son las versiones de ROM usadas en los dispositivos en producción, pero que han agregado, entre otras cosas, un código que facilita la acción de debugueo y sirve para verificar principalmente parámetros.

Documentación Palm OS. Toda la documentación para Palm OS puede ser encontrada en el sitio en Internet de Palm, donde existe un gran número de preguntas frecuentes, notas, las llamadas páginas blancas entre otros, y toda esta documentación esta siendo actualizada muy frecuentemente.

Tutorial de Palm. Esto es una herramienta que muestra la construccion de una aplicación desde el principio hasta que queda completada. El Tutorial[6] asume que se estará desarrollando la aplicación en CodeWarrior para Palm OS. Existen versiones del Tutorial para Windows y para Macintosh y pueden ser bajados de internet gratuitamente del sitio de Palm.

Kit para el desarrollo de Conductos. Este es el Kit para desarrollar conductos para el sistema operativo Mac o Windows utilizando lenguaje C o C++. Este Kit comúnmente viene incluido con CodeWarrior para Palm OS. La versión para Windows requiere Visual C++ y la versión para Macintosh requiere CodeWarrior para el sistema

[6] El tutorial es un programa que sirve para enseñar a utilizar cierta herramienta, mostrando las caracteristicas de ésta.

operativo Mac. También existe una herramienta que utiliza el lenguaje Java para el desarrollo de conductos el cual es "SDK de conductos Edición Java".

5.6 LA OPCION ADECUADA

Ahora ya se tiene una mejor apreciación de las opciones para crear aplicaciones para dispositivos Palm, y se puede apreciar que CodeWarrior es la mejor opción. Los programadores de Windows tienen mas flexibilidad a comparación de Macintosh y Unix.

Si se desea programar en ensamblador, entonces la mejor opción es utilizar Assambler SDK. O si lo que se quiere es programar en C/C++, entonces la mejor opción es CodeWarrior o GCC. Si el programador es ocasional o es programa de hobbie entonces GCC es la mejor elección dado su precio atractivo. Pero tiene la desventaja que mientras es mas flexible, es mas difícil de usar.

Si el precio es un factor importante, entonces, Pendragon Forms es la opción de mas bajo costo para crear formas simples para la captura de datos. Si se necesita crear una aplicación con un presupuesto muy reducido Pendragon Forms es la elección.

Ahora que si lo que se quiere es tener una gran facilidad en el manejo y un precio que no sea muy alto (aunque tampoco es muy bajo), entonces, se debe utilizar CodeWarrior. La inclusión del Kit para el desarrollo de software de conductos para Palm como parte del paquete, la documentación, el código fuente que proporciona y el soporte de Palm, hacen que este ambiente de desarrollo sea la mejor opción para desarrollar aplicaciones Palm.

5.7 TERMINOLOGIA

Así como en cualquier otro sistema operativo, Palm OS tiene su propia terminología. Mucha puede ser familiar, ya que se parece a cualquier otra. A continuación se presenta:

Forma. Es una ventana de una aplicación, lo que se podría conocer como una vista, y que usualmente cubre la pantalla completa. Una forma opcionalmente contiene controles, áreas de texto y menús. En el sistema operativo Palm, existe una sola forma activa en un momento.

Ventana. Es un área rectangular en la cual objetos como diálogos, formas y menús son dibujados en una aplicación. El manejador de ventanas se asegura que las ventanas son desplegadas apropiadamente con respecto a otras ventanas, por ejemplo, tiene la habilidad de restaurar el contenido de la ventana anterior cuando una ventana es cerrada. Todas las formas son ventanas, aun cuando no todas las ventanas son formas.

Base de datos. Es una colección de pedazos de memoria. Las hay de dos tipos: Base de recursos y Base de datos de registros.

Recurso. Es un conjunto de datos almacenado en una base de datos de recursos. Cada recurso es identificado por un tipo de recurso y un numero. Una aplicación Palm es una colección de recursos.

Registro. Es una estructura de datos identificado por un único identificador de registro. Las aplicaciones almacenan los datos en una base de datos de registros.

Evento. Es una estructura de datos que describe cosas que suceden en una aplicación. Los eventos pueden ser de bajo nivel, los llamados eventos de hardware como el toque con el stylus o presionar un botón. También pueden ser de alto nivel como

cuando un carácter es escrito, un elemento del menú es seleccionado o un botón de software es presionado.

El sistema operativo Palm es un sistema manejado por eventos. Solo una aplicación está abierta en un momento y cuando se está ejecutando, entonces, corre en un ciclo de eventos que captura los eventos y continúa manejándolos hasta que el usuario empieza otra aplicación.

El principal ciclo de eventos. Es el principal ciclo que se ejecuta en una aplicación, la cual repetidamente captura los eventos y entonces actúa sobre ellos.

Código invocado. Es un parámetro pasado a una aplicación que especifica lo que la aplicación debe hacer cuando ese código particular es ejecutado. Una aplicación normalmente maneja mas de un código invocado, éste es el método de comunicación utilizado entre el sistema operativo y una aplicación y entre aplicaciones.

Menú. Los menús son almacenados en recursos agrupados juntos en menubars y son desplegados cuando el usuario hace un tap[7] en el área de menú.

Menubar. Es una colección de menús almacenados en un recurso. Cada forma puede tener un menubar asociado.

Diálogos. Es una ventana que contiene controles, y requiere que el usuario haga una decisión. En otras palabras, el dialogo debe ser cerrado (usualmente dando un tap en uno de sus botones) antes de que la aplicación pueda continuar.

Alerta. Un mensaje de precaución o un diálogo de información que necesita ser cerrado por el usuario.

[7] Un tap es un toque en la pantalla de una Palm

5.8 ELEMENTOS DE LA INTERFAZ DE USUSARIO EN UNA APLICACION

El sistema operativo Palm tiene una gran cantidad de elementos para la interfaz de usuario. A continuación se muestran algunos de estos elementos y una breve descripción de estos:

Alertas

Es un simple dialogo modal que despliega un titulo, un mensaje, un icono y uno o mas botones. El usuario es el responsable de establecer el texto del titulo, el mensaje y los botones, así como también, el tipo de alerta. Los tipos de alerta pueden ser cualquiera de los siguientes:

- **Información.** Este tiene un icono "i", y contiene información para el usuario, por ejemplo, que cierta acción no puede ser completada y no hay pérdida de datos.

- **Confirmación.** Este tiene un icono "?", y realiza una pregunta al usuario, preguntándole por la confirmación de una acción o para elegir de entre varias posibilidades.

- **Precaución.** Este tiene un icono "!", y lo que hace es preguntarle al usuario si la acción es realmente intencional. La pérdida de datos podría ocurrir si la acción es completada, por ejemplo, la aplicación de notas (Memo) usa una alerta de confirmación en el momento de borrar alguna nota. También existe la opción en la que el usuario marca una casilla para salvar en el disco de la PC y así los datos no son perdidos. Pero por ejemplo, el sistema usa una alerta de precaución cuando el usuario elige borrar alguna aplicación, y entonces, después del borrado la aplicación esta completamente perdida.

- **Error.** Este tiene un icono de "Alto", y le dice al usuario que un error ha ocurrido como resultado de la última acción.

Formas

Una forma es un contenedor de propósito general para uno o mas de los otros elementos de la interfaz de usuario. Una forma puede contener botones, listas, tablas, controles e iconos, y también puede tener un menubar asociado. Las formas pueden ser desde dialogos modales hasta contenedores de listas o tablas de datos, pequeñas o que llenan la pantalla completa del dispositivo Palm.

La apariencia de una forma, incluyendo la apropiada distribución de los botones, está cubierta en la documentación de Palm OS.

Menús, Elementos de Menú y Barras de Menú

Menús, elementos de Menú y las barras de Menú están relacionados entre ellos. Una barra de menú contiene uno o mas menús. Un menú contiene uno o mas elementos de menú. Y los elementos de menú normalmente tienen accesos en Graffiti asociados con cada uno.

Tablas y listas

Las tablas y las listas son usadas para propósitos similares. Una tabla se usa cuando se quiere desplegar múltiples columnas de datos; y una lista cuando se quiere desplegar únicamente una sola columna. Las tablas pueden soportar diferentes tipos de datos.

Otros elementos de la interfaz de usuario

Existen otros elementos para la interfaz con el usuario, como son botones, casillas de verificación, bitmaps, campos, artefactos, etiquetas e indicadores de Graffiti, listas desplegables, botones de opción, de repetición y barras. A continuación se presenta una tabla con los elementos mas comunes para la interfaz con el usuario:

Elemento para la interfaz de usuario	Descripción
Botón	Un botón es un objeto que se le puede hacer un tap y que contiene una etiqueta, además, una acción ocurre cuando se realiza un tap sobre el.
Casilla de verificación	Una casilla de verificación representa un estado de encendido / apagado.
Campo	Este es usado para que el usuario introduzca datos, puede tener una o varias líneas de texto editable y puede ser no editable, también.
Bitmap	Es un objeto de imagen que usualmente es blanca y negra con escala de grises.
Artefacto	Esto es un objeto para la interfaz de usuario personalizado, el cual es limitado, únicamente, por la imaginación del desarrollador y puede ser para uso complicado.
Indicador de Graffiti	Este muestra el estado actual de graffiti e indica si esta en modo de introducir puntuación, símbolos, mayúscula o minúsculas. Este indicador debe estar en cualquier pantalla que permita la entrada de texto y debe de colocarse en la esquina inferior derecha de la pantalla.
Etiqueta	Este es un objeto de texto no editable.
Lista desplegable	Se realiza un tap sobre este objeto para desplegar una lista. Este objeto muestra el elemento seleccionado de la lista.
Botón de opción	Representan un estado de encendido o apagado y estan agrupados de tal manera que unicamente uno puede estar seleccionado del grupo
Botón de repetición	Este trabaja como un boton pero causa una acción de repetición mientras el botón se mantiene oprimido.
Barra de desplazamiento	Este objeto es comúnmente usado para moverse a través de texto o tablas. Permite el desplazamiento de una línea, una página y la navegación directa a un punto en particular.

Selector Disparable	Cuando un usuario hace un tap sobre este objeto un cuadro de dialogo aparece para permitir al usuario editar el valor.

TABLA 5.1

5.9 ESTRUCTURA DE UNA APLICACIÓN EN CODEWARRIOR

5.9.1 Los archivos #include

Pilot.h es un archivo que incluye la mayoría de los archivos include para el sistema operativo Palm estándar, es decir, que la mayoría de las funciones están definidas en éste archivo. Este archivo es utilizado si se está usando CodeWarrior.

Otro archivo include como es Callback.h define algunas macros necesarias si se esta utilizando GCC y que son necesarias para manejar llamadas desde el sistema operativo Palm hacia el código.

Otro archivo include, es uno que es creado a partir de los recursos de la aplicación (normalmente lleva el nombre de la aplicación con terminación Rsc y extensión h) y define constantes para todos los recursos de la aplicación. Por ejemplo, cuando se está desarrollando en CodeWarrior, es necesario crear la parte visible de la aplicación con el "Constructor" que es una herramienta de CodeWarrior y la cual crea el archivo *Rsc.h automáticamente; pero, si se esta utilizando GNU PalmPilot SDK, entonces, es necesario crear dicho archivo por separado.

5.9.2 La rutina principal: PilotMain

Esta función es el punto de entrada a la aplicación, y tiene el mismo propósito que la función Main[8] en lenguaje C o que la función WinMain para programadores de

[8] Es la función principal, la cual llama a todas las demás funciones en lenguaje C/C++

Windows. Pero a diferencia de éstas, las aplicaciones Palm utilizan un diferente conjunto de argumentos que los programas tradicionales en C/C++. El prototipo de la función PilotMain es el siguiente:

Dword PilotMain (Word cmd, Ptr cmdPBP, Word launchFlags)

El significado de los parámetros es el siguiente:

cmd es el código de invocación o el código de acción que identifica como la aplicación fue invocada y que es lo que se esta pidiendo que haga. Si la aplicación es abierta con normalidad, éste parámetro contiene el valor sysAppLaunchCmdNormalLaunch.

cmdPBP es un apuntador a un bloque de parámetro, cuyo contenido depende del código de invocación.

launchFlags es un conjunto de banderas bit que proporciona información adicional del contexto de la aplicación.

El valor de retorno de la función PilotMain es el valor del status de error, un valor de cero significa que la aplicación corre sin errores. Cuando existen errores, entonces, regresa un valor diferente de cero, que usualmente son códigos de error del sistema.

Si el primer parámetro de la función PilotMain contiene el valor sysAppLaunchCmdNormalLaunch, se debe hacer la inicialización en la función StartApplication y después la aplicación es manejada por el ciclo de eventos de la función EventLoop hasta que el usuario hace algo para cerrar la aplicación.

5.9.3 La rutina de inicio: StartApplication

En esta función se realiza la inicialización de la aplicación, se abren las bases de datos y se lee la información de las preferencias del usuario.

Ejemplo:

```
Static Err StartApplication(void)
{
   FrmGotoForm(NombreForm);
   Return 0;
}
```

5.9.4 La rutina de cierre: StopApplication

Normalmente en esta rutina se cierran todas las operaciones que pudieran estar abiertas, se cierra la base de datos y se salva el actual estado en las preferencias.

5.9.5 La rutina EventLoop

Es un ciclo donde continuamente se procesan eventos. Primero, la función EvtGetEvent obtiene el evento y lo pasa a uno de los cuatro manejadores anidados, donde cada uno de estos tiene una oportunidad para manejar el evento. Si un de estos manejadores regresa un valor verdadero, es que ya manejó el evento y ya no se procesa mas. Entonces, la función EvtGetEvent obtiene el siguiente evento de los que están formados para ser atendidos, y el ciclo repite el proceso. El ciclo continua de esta forma hasta que se dispara el evento appStopEvent. Después se pasa el control a la función StopApplication donde todo se limpia.

El manejo de eventos con EvtGetEvent. El propósito de esta rutina de manejo de eventos es obtener el siguiente evento de los que están formados. Esta función tiene dos parámetros, el primero es un apuntador al evento que se va a manejar, el segundo es el valor de tiempo en el que busca el siguiente evento (centésimas de segundo). EvtGetEvent regresa un valor, cuando un evento a ocurrido regresa verdadero o cuando por tiempo se dispara y regresa un valor verdadero.

5.9.6 La cola de eventos y el ciclo de eventos de la aplicación

Una forma de ver a una aplicación Palm es como un manejador de eventos, ya que toma todos los eventos ordenados, les da un manejo adecuado por varios manejadores. Los manejadores pueden convertir a cada evento en otro evento y regresarlo a la cola de eventos para que pueda ser manejado con otro manejador.

Si por ejemplo un usuario tiene una aplicación abierta y hace un tap sobre la pantalla en el botón de Menú, se genera un evento. Primero, éste evento pasa a la cola de eventos y es manejado por la rutina SysHandleEvent, la cual, lo interpreta y crea un nuevo evento que es regresado a la cola de eventos.

Ahora la rutina MenuHandleEvent toma el manejo del evento y lo reconoce como una petición de Menú. Entonces, la rutina MenuHandleEvent muestra la barra de menú y despliega uno de los menús. Si el usuario hace un tap fuera del menú, entonces, éste desaparece.

Si un elemento es seleccionado, un nuevo evento es generado y enviado a la cola de eventos. Este nuevo evento es atrapado por el ciclo de eventos, donde es pasado a SysHandleEvent y nuevamente a MenuHandleEvent.

Como se puede apreciar existen diferentes manejadores y cada uno de los cuales se encarga de diferentes tipos de eventos.

5.9.7 La rutina SysHandleEvent

La primera rutina en el ciclo de eventos es SysHandleEvent. Esta rutina proporciona la funcionalidad común a todas las aplicaciones Palm. Maneja los eventos de los botones de las aplicaciones que se desarrollan. Lo hace disparando el evento appStopEvent que cierra la aplicación actual, después, el sistema puede llamar a la aplicación deseada.

Maneja los toques con la pluma que son llamados taps, que se llevan a cabo en el área de Graffiti o en los botones que se encuentran en la misma área. Por ejemplo, si un usuario da un tap en el botón "Find" (Buscar), la rutina SysHandleEvent maneja completamente la búsqueda, regresando hasta que la búsqueda sea hecha.

Estos son algunos de los eventos mas importantes que maneja:

KeyEvent. Ocurre, entre otras veces, cuando uno de los botones construidos son presionados. Tiene un código que indica que botón en particular fue presionado.

SysHandleEvent maneja los eventos generados por el stylus en el área de Graffiti. Cuando un carácter es ingresado, SysHandleEvent genera un evento keyEvent con el carácter reconocido.

PenDownEvent. Ocurre cuando el usuario presiona con el stylus la pantalla.

PenMoveEvent. Ocurre cuando el usuario mueve el stylus sobre la pantalla.

5.9.8 MenuHandleEvent

Es la segunda rutina en el ciclo de eventos, y como se puede intuir, es la rutina que maneja todo lo referente a Menús. Los eventos ocurren cuando el usuario:

Hace un tap sobre el botón de Menú del área de Graffiti. La función encuentra el objeto menubar de la forma actual y despliega el Menú.

Hace un tap en algún lugar que no sea el Menú cuando este esta desplegado. La función cierra el menú cuando el usuario hace un tap fuera de el.

MenuHandleEvent también sirve como un interruptor si el usuario hace un tap sobre el menubar. Cierra el Menú y el Menubar si el usuario hace un tap sobre algún elemento del Menú. Genera un evento de Menú que después será atrapado por una llamada a EvtGetEvent.

5.9.9 ApplicationHandleEvent

La tercera rutina en el ciclo, y es la responsable de cargar formas y asociar un manejador de eventos con la forma.

5.9.10 FrmDispatchEvent

Esta es la cuarta y última rutina en el ciclo. Es la que proporciona directamente un control específico sobre la forma, maneja la funcionalidad estándar de la forma, por ejemplo, el evento pen-down sobre un botón lo resalta y un evento pen-up genera un evento ctlSelectEvent a la cola de eventos. Cortar, copiar y pegar en cuadros de texto son otros ejemplos de funcionalidad manejada por FrmDispatchEvent. Cuando esta rutina obtiene un evento, hace una llamada a la rutina NombreFormaHandleEvent. Donde NombreForma es el nombre de la forma sobre la que se esta desarrollando, y en esta rutina es donde el programador escribe el manejo que se le quiera dar a la forma o a la aplicación.

La estructura de una aplicación desarrollada en CodeWarrior es la siguiente:

- Un conjunto de archivos include que son necesarios para la aplicación
- Una rutina de inicio llamada StartApplication, la cual maneja toda la configuración inicial de la aplicación
- Una rutina PilotMain, que es el punto de entrada al programa, la cual hace una llamada a la rutina StartApplication e inicia un ciclo de eventos para manejar los eventos que le pasa el sistema
- Un ciclo de eventos, el cual continuamente maneja los eventos en una serie de cuatro rutinas de manejo las cuales son, SysHandleEvent, MenuHandleEvent, ApplicationHandleEvent y FrmDispatchEvent.
- Un conjunto de rutinas para manejar la funcionalidad específica de la forma, la cual es NombreFormaHandleEvent
- Una rutina de cierre llamada StopApplication, la cual maneja el cierre apropiado de la aplicación.

CONCLUSIONES

Con el desarrollo de la tecnología en los últimos tiempos se ha tratado de satisfacer la necesidad de transportar o llevar consigo algún tipo de información, y se ha visto como con el tiempo se han desarrollado gran cantidad de dispositivos que cumplan con esta función. Y con ello también se ha desarrollado software diverso que permita la comunicación y proporcione la funcionalidad de tan diversos dispositivos.

En la actualidad han tenido un gran auge los dispositivos portátiles sobre todo los que cuentan con el sistema operativo Palm OS. Los dispositivos portátiles de la marca Palm (que por supuesto cuentan con el sistema operativo Palm OS) son los mas utilizados en estos días, ya que tienen una gran variedad de características con las que se pueden llevar a cabo muchas funciones, y esto se debe al sistema operativo con el que cuentan. Además, este sistema operativo presenta la característica de que se puede desarrollar software que agregue mas funcionalidad a estos dispositivos.

Es por esto que los dispositivos portátiles que cuentan con el sistema operativo Palm OS son los mas apropiados para el desarrollo de aplicaciones personalizadas.

El SPT1700 es un dispositivo portátil que cuenta con el sistema operativo Palm OS, y que permite desarrollar aplicaciones como la propuesta en esta tesis. Además, este dispositivo tiene la característica de poder leer códigos de barras permitiendo que éstos sean manipulados por dicha aplicación. El SPT1700 a diferencia del SPT1500 (que también tiene las características anteriores), tiene un diseño que permite un uso mas rudo, de tal manera que pueda utilizarse en un ambiente del tipo industrial como es el caso de una bóveda en una institución bancaria.

En el presente proyecto de tesis se utiliza la codificación en códigos de barras como medio de identificación de las unidades de empaque de los billetes y es por esto que se eligió el SPT1700 ya que cuenta con las características deseables para implantar un sistema que requiere de dispositivos portátiles que puedan ser introducidos en una bóveda y que pueda manejar los códigos de barras.

Este sistema es propuesto para cualquier institución bancaria que quiera cambiar de la forma tradicional con que son manejados los billetes, a una forma mucho mas moderna, que proporciona un gran control sobre los billetes y proporciona una forma de centralizar toda la información relacionada con estos; y que también facilita la operación de quienes laboran en una bóveda y que realizan operaciones con los billetes como son: el llenado o vaciado de algún tipo de contenedor, el transportar de una bóveda a otra grandes cantidades de dinero.

Por otro lado, la tarifa de referencia que se aplicó para este proyecto establece que un técnico programador, por desarrollar un sistema con las características ya mencionadas, cobra $250.00 la hora mientras que un ingeniero cobra $350.00 la hora. Entonces tenemos que el sistema propuesto tiene un costo aproximado de $336,000.00. Considerando que el sistema debió haber sido desarrollado en un tiempo razonable para que el avance de la tecnología no propiciara que al término de su desarrollo, ya tuviera algunas características que fueran obsoletas. El sistema fue desarrollado en un tiempo de 3 meses por dos ingenieros

El sistema benefició en gran medida a la institución bancaria ya que se tiene ahora un gran control sobre el manejo de los billetes, abarcando todas las sucursales de dicha institución incluyendo la oficina central de la misma.

Con este sistema se eliminó la forma anterior de controlar la cantidad de lo que se almacena en los contenedores de los billetes, pasando a una forma mucho mas práctica y sencilla. También se tiene ahora un control mucho mas efectivo de los traspasos que se hacen de los contenedores de los billetes, sabiendo en el momento de la lectura del código de barras del contenedor, la bóveda de la cual partió.

Se consiguió también la centralización del control, ya que cuando un dispositivo portátil es sincronizado con una computadora de escritorio la información es transferida a ésta última y a su vez la computadora de escritorio transfiere la información recibida a un

servidor donde se concentra toda la información de todas las bóvedas con que cuente la institución bancaria.

Con el sistema propuesto ahora se tiene un mayor control al momento de identificar el estado físico de los billetes, ya que se tiene que diferenciar entre los billetes nuevos, los billetes que tiene media vida y los que ya están deteriorados y tienen que salir de circulación.

APENDICE A

```
/*****************************************************************************************
* Archivo: c:\bmx\sifca2000\ulsifca2000.sqc
*       Fecha          :05 Julio 2000
* Autor          : Oscar G. / Juan P. / Jorge A.
* Proposito: Definicion de los metodos para el manejo de datos de
*                                                 de la base de datos en la PALM.
*
*****************************************************************************************/
// Declaro el objeto que realiza las transacciones
EXEC SQL INCLUDE SQLCA;

#include <Pilot.h>
#include "platform.h"
#include "ulsifca2000.h"
#include "sqlca.h"
#include <DLServer.h>
#include "MainFormRsc.h"

#if 0
    // Estas se requieren para SQLPP, pero son #if 0 porque las variables locales
    //  se vuelven miembros de la base de datos o variables locales
        EXEC SQL BEGIN DECLARE SECTION;
                        DECL_DATETIME    fechaConsulta;
                        char                        m_CodigoBarras[18];
                        char                        m_EmpID[7];
                        char        m_LectoraID[6];
  EXEC SQL END DECLARE SECTION;

        EXEC SQL BEGIN DECLARE SECTION;

                long                                        sql_institucion_empaque;
                char                                        sql_consecutivo[7];
                unsigned short  sql_especimen;
                char                                        sql_tipo_envase[3];
                char                                        sql_digito_verificador[2];
                long                                        sql_institucion_empaque_padre;
                char                                        sql_consecutivo_padre[7];
                short                                sql_status_envase;
                unsigned short  sql_estado_fisico;

                unsigned short          sql_count_e_f;
                unsigned short          sql_count_e;
                unsigned short          sql_count;
                unsigned short          sql_count_1=0;
                unsigned short          sql_count_2=0;
                unsigned short          sql_count_3=0;
                unsigned short          sql_count_4=0;


                unsigned short  sql_tipo_operacion;
                unsigned short  sql_folio_operacion;
                DECL_DATETIME           sql_fecha_operacion;
                unsigned short  sql_estado_fisico_boveda;
                char                                        sql_clave_empleado[7];

                unsigned short  sql_offset;

                unsigned short  sql_clv_edo_fis;
                char                                        sql_EdoFisico[16];

                unsigned short    sql_clv_boveda;
                char                        sql_Boveda[21];
                unsigned short    sql_clv_especimen;
                char                                        sql_Especimen[16];

                char                sql_restriccion[2];

                unsigned short          sql_edo_fis_origen;
                unsigned short          sql_edo_fis_destino;

                char                                        sql_constante[26];
                char                                sql_valor_constante[6];
        EXEC SQL END DECLARE SECTION;
#endif

/*****************************************************************************
* Funcion: ulsifca
* Fecha          : 06JULIO2000
* Autor          : Jorge A.
* Parametro      :ninguno
* Regresa        : nada
*****************************************************************************/
ulsifca::ulsifca()
{
    m_CodigoBarras[0] = '\0';

    m_LectoraID[0] = '\0';
    DlkGetSyncInfo(NULL, NULL, NULL, &m_LectoraID[0], NULL, NULL);

    // Establecer los parametros de sincronizacion
```

```
    m_SynchMethod = SYNCH_SERIAL;
    m_SynchParms[0] = '\0';
}

/*****************************************************************************
* Funcion: ~ulsifca
* Fecha          :06JULIO2000
* Autor          : Jorge A.
*****************************************************************************/
ulsifca::~ulsifca()
{
}

/*****************************************************************************
* Funcion: Rollback
*       Fecha            :06JULIO2000
*       Autor            : Oscar G.
* Descripcion: Ejecuta la sentecia ROLLBACK
* Parametro              :ninguno
* Regresa                :nada
*****************************************************************************/
void ulsifca::Rollback( void )
{
    EXEC SQL ROLLBACK;
}

/*****************************************************************************
El manejo de errores lo establezco aqui porque utilizo la funcion Rollback
que defino justo antes
*****************************************************************************/
EXEC SQL WHENEVER SQLERROR { Rollback();};

/*****************************************************************************
* Funcion:      Inicio
* Fecha         : 05Julio2000
*       Autor           : Oscar G.
* Parametros    : ninguno
* Regresa                       : true si la base de datos fue abierta e inicalizada correctamente
*                                                               false en caso contrario
* Descripcion: Inicializar la base de datos y establecer la conexion
*****************************************************************************/
SByte ulsifca::Inicio( void )
{
  SByte ret = 0;

  if( db_init( &sqlca ) ) {
                        EXEC SQL CONNECT "dba" IDENTIFIED BY "sql";

                        ret = 1;
  }
  return( ret );
}

/*****************************************************************************
* Funcion:      Fin
* Fecha         : 06Julio2000
* Autor         : Jorge A.
* Parametros    : ninguno
* Regresa                       : true si la base de datos fue cerrada correctamente
*                                                               false en caso contrario
* Descripcion: Cerrar la base de datos
*****************************************************************************/
Int ulsifca::Fin( void )
{
    EXEC SQL DISCONNECT;
    db_fini( &sqlca );
    return( 1 );
}

/*****************************************************************************
* Funcion: SetSynchParams
* Fecha : 06JULIO2000
* Autor : Oscar g.
* Descripcion   : Establecer los parametros para la sincornizacion de los datos
* Parametrsos: metodo.- Metodo que usare para sincronizar          parms.- cadena con los parametros de la
sincronizacion
* Regresa                       :nada
*****************************************************************************/
void ulsifca::SetSynchParms( SynchMethod metodo, char *parms )
{
    switch( metodo ) {
        case SYNCH_SERIAL:
        case SYNCH_TCPIP:
        case SYNCH_HOTSYNC:
            m_SynchMethod = metodo;
            break;
        default:
            break;
    }
```

```
    if( parms != NULL ) {
    StrCopy(m_SynchParms, parms);
    }
}

/*****************************************************************************************
* Funcion: Synchronize
* Fecha  : 15AGO2000
* Autor  : Jorge A.
* Descripcion   :Llevar a cabo la sincronizacion de la palm a la bd consolidada
* Parametros: Ninguno
* Regresa       :nada
*****************************************************************************************/
Int ulsifca::Synchronize( void )
{
    Int ok;

    switch( m_SynchMethod ) {
        case SYNCH_SERIAL:

            ULSynchronize( &sqlca, m_LectoraID, ULSerialStream(), m_SynchParms );
            break;
        case SYNCH_TCPIP:
          ULSynchronize( &sqlca, m_LectoraID, ULSocketStream(), m_SynchParms );
            break;
        default:
            return( false );
    }

    ok = ( SQLCODE != SQLE_COMMUNICATIONS_ERROR );
    return( ok );
}

/*****************************************************************************************
* Funcion:
* Fecha :
* Autor :
* Descripcion   :
* Parametros: Ninguno
* Regresa       : 0 - sin error, <>0 otro caso.
*****************************************************************************************/
Int ulsifca::db_D_ALL_TABLES()
{
        Int li_resultado = 0;

        EXEC SQL DELETE UL_D_MONTO;
        EXEC SQL DELETE UL_D_ARMADO;
        EXEC SQL DELETE UL_D_OPERACION;
        EXEC SQL DELETE UL_OPERACION;
        EXEC SQL DELETE UL_ENVASE;

        EXEC SQL DELETE UL_ESPECIMEN_DENOM_GRAL;
        EXEC SQL DELETE VALIDACION_OPERACION;
        EXEC SQL DELETE ESPECIMEN;
        EXEC SQL DELETE BOVEDA;
        EXEC SQL DELETE ESTADO_FISICO;
        EXEC SQL DELETE TIPO_ENVASE_ESPECIMEN_HIJOS;

        EXEC SQL COMMIT;

        return li_resultado;
}

/************************        FUNCIONES  GENERALES        ************************/
/************************              UL_ENVASE             ************************/

/*****************************************************************************************
* Funcion       : db_IUL_ENVASE
* Fecha         : 08.03.2001
* Autor         : Jorge A. Contreras
* Descripcion   : Inserto un registro en la tabla UL_ENVASE
* Paramteros    : a_institucion_empaque, a_consecutivo, a_especimen, a_tipo_envase, a_digito_verificador,
a_institucion_empaque_padre, a_consecutivo_padre, a_status_envase, a_estado_fisico
* Regresa       : 0 si no hay error o -1 si lo hay
*****************************************************************************************/
SByte ulsifca::db_IUL_ENVASE(Long  a_institucion_empaque,
                                                char    a_consecutivo[7],
                                                UInt    a_especimen,
                                                char  a_tipo_envase[3],
                                                char    a_digito_verificador[2],
                                                Long    a_institucion_empaque_padre,
                                                char    a_consecutivo_padre[7],
                                                Int     a_status_envase,
                                                UInt    a_estado_fisico)

{
        SByte  resultado = 0;

  EXEC SQL BEGIN DECLARE SECTION;
                long    sql_institucion_empaque;
```

```
                char    sql_consecutivo[7];
                long    sql_institucion_empaque_padre;
                char    sql_consecutivo_padre[7];
                char    sql_tipo_envase[3];
                char    sql_digito_verificador[2];
                short sql_status_envase;
        unsigned short sql_especimen;
                unsigned short sql_estado_fisico;
                char    sql_Lectora_ID[6];
                unsigned short   sql_count_e_f;
                unsigned short   sql_count_e;
        EXEC SQL END DECLARE SECTION;

        // Las pongo en variables locales

        sql_institucion_empaque                 = a_institucion_empaque;
        sql_especimen                                   = a_especimen;
        sql_institucion_empaque_padre = a_institucion_empaque_padre;
        sql_estado_fisico                               = a_estado_fisico;
        sql_status_envase                                       = a_status_envase;

        StrCopy(sql_consecutivo,                        a_consecutivo);
        StrCopy(sql_tipo_envase,                        a_tipo_envase);
        StrCopy(sql_digito_verificador,         a_digito_verificador);
        StrCopy(sql_consecutivo_padre,          a_consecutivo_padre);
        StrCopy(sql_Lectora_ID,                         m_LectoraID);

 // Se verifica que el ESTADO_FISICO exista..

        EXEC SQL SELECT count(*)
                                INTO :sql_count_e_f
                                FROM ESTADO_FISICO
                        WHERE estado_fisico = :sql_estado_fisico;

 // y que NO EXISTA ese ENVASE previamente en la tabla UL_ENVASE
        EXEC SQL SELECT count(*)
                                INTO :sql_count_e
                                FROM UL_ENVASE
                        WHERE institucion_empaque  = :sql_institucion_empaque
                                AND consecutivo            = :sql_consecutivo;

        if (sql_count_e_f == 0 && a_estado_fisico != 0)
           resultado = -3;

        if (sql_count_e != 0 )
                resultado = -1;

 // Hago el insert en la tabla
        if (resultado == 0) {
                EXEC SQL INSERT INTO UL_ENVASE(institucion_empaque,
                                        consecutivo,
                                        especimen,
                                        tipo_envase,
                                        digito_verificador,
                                        institucion_empaque_padre,
                                        consecutivo_padre,
                                        status_envase,
                                        estado_fisico,
                                        lectora_id)
                                VALUES(:sql_institucion_empaque,
                                         :sql_consecutivo,
                                         :sql_especimen,
                                         :sql_tipo_envase,
                                        :sql_digito_verificador,
                                        :sql_institucion_empaque_padre,
                                        :sql_consecutivo_padre,
                                        :sql_status_envase,
                                        :sql_estado_fisico,
                                        :sql_Lectora_ID);
                EXEC SQL COMMIT;
        } else
                        Rollback();
        return resultado;
}

SByte ulsifca::db_UUL_ENVASE(Long    ai_institucion_empaque,
                                                char    ac_consecutivo[7],
                                                Boolean ab_padre,
                                                Long    ai_institucion_empaque_padre,
                                                char ac_consecutivo_padre[7],
                                                Boolean ab_status,
                                                Int     ai_status_envase,
                                                Boolean ab_estado,
                                                UInt    ai_estado_fisico)
{
        SByte resultado = 0;

        EXEC SQL BEGIN DECLARE SECTION;
                long    sql_institucion_empaque;
                char    sql_consecutivo[7];
```

```
                long    sql_institucion_empaque_padre;
                char    sql_consecutivo_padre[7];
                short sql_status_envase;
                unsigned short sql_estado_fisico;
                unsigned short   sql_count_e_f;
                unsigned short   sql_count_e;
        EXEC SQL END DECLARE SECTION;

        // Las pongo en variables locales
        sql_institucion_empaque                 = ai_institucion_empaque;

        sql_estado_fisico                               = ai_estado_fisico;
        sql_status_envase                                       = ai_status_envase;

        StrCopy(sql_consecutivo,                        ac_consecutivo);

// Se verifica que el ESTADO_FISICO exista..
        EXEC SQL SELECT count(*)
                                INTO :sql_count_e_f
                                FROM ESTADO_FISICO
                        WHERE estado_fisico = :sql_estado_fisico;

// y que EXISTA ese ENVASE en la tabla UL_ENVASE
        EXEC SQL SELECT count(*)
                                INTO :sql_count_e
                                FROM UL_ENVASE
                        WHERE institucion_empaque  = :sql_institucion_empaque
                                AND consecutivo             = :sql_consecutivo;

        if (sql_count_e_f == 0 && ai_estado_fisico != 0)
           resultado = -3;

        if (sql_count_e == 0 )
                resultado = -5;

// Hago el update en la tabla
        if (resultado == 0) {

                if (ab_padre) {   // Si debo actualizar los datos del padre....
                        sql_institucion_empaque_padre = ai_institucion_empaque_padre;
                        StrCopy(sql_consecutivo_padre,  ac_consecutivo_padre);

                        EXEC SQL UPDATE UL_ENVASE
                                                SET institucion_empaque_padre =
:sql_institucion_empaque_padre,
                                                        consecutivo_padre    = :sql_consecutivo_padre
                                          WHERE institucion_empaque = :sql_institucion_empaque
                                        AND consecutivo = :sql_consecutivo;
                }

                if (ab_estado) {   // y /o los del estado_fisico...
                        sql_estado_fisico = ai_estado_fisico;

                        EXEC SQL UPDATE UL_ENVASE
                                                SET estado_fisico = :sql_estado_fisico
                                          WHERE institucion_empaque = :sql_institucion_empaque
                                        AND consecutivo = :sql_consecutivo;
                }

                if (ab_status) {  // y por ultimo el status...
                        sql_status_envase = ai_status_envase;

                        EXEC SQL UPDATE UL_ENVASE
                                                SET status_envase = :sql_status_envase
                                          WHERE institucion_empaque = :sql_institucion_empaque
                                        AND consecutivo = :sql_consecutivo;
                }
         }
        if (resultado == 0)
                EXEC SQL COMMIT;
        else
                Rollback();

        return resultado;
}

SByte ulsifca::db_DUL_ENVASE(Long ai_institucion_empaque, char ac_consecutivo[7])
{
        long resultado = 0;

        EXEC SQL BEGIN DECLARE SECTION;
                long    sql_institucion_empaque;
                char    sql_consecutivo[7];
                unsigned short   sql_count;
        EXEC SQL END DECLARE SECTION;

        // Las pongo en variables locales

        sql_institucion_empaque = ai_institucion_empaque;
```

```
        StrCopy(sql_consecutivo,ac_consecutivo);

 // verifica que EXISTA ese ENVASE previamente en la tabla UL_ENVASE
        EXEC SQL SELECT count(*)
                                INTO :sql_count
                                FROM UL_ENVASE
                        WHERE institucion_empaque  = :sql_institucion_empaque
                                AND consecutivo          = :sql_consecutivo;

 // Hago el delete en la tabla
        if (sql_count != 0) {

                EXEC SQL DELETE UL_ENVASE
                                        WHERE institucion_empaque = :sql_institucion_empaque
                                        AND       consecutivo = :sql_consecutivo;

                resultado = 0;
         } else
                resultado = -5;

//      resultado = sqlca.sqlcode;

        if (resultado == 0)
                EXEC SQL COMMIT;
        else
                Rollback();

        return resultado;
}

SByte ulsifca::db_getInfo_UL_ENVASE(Long  ai_institucion_empaque,
                                        char    ac_consecutivo[7],
                                        Int*    ai_status_envase,
                                        UInt*  ai_estado_fisico)
{
        SByte  resultado = 0;

        EXEC SQL BEGIN DECLARE SECTION;
                long      sql_institucion_empaque;
                char    sql_consecutivo[7];
                short   sql_status_envase;
                unsigned short   sql_estado_fisico;
                unsigned short sql_count;
        EXEC SQL END DECLARE SECTION;

        // Las pongo en variables locales

        sql_institucion_empaque = ai_institucion_empaque;
        StrCopy(sql_consecutivo,  ac_consecutivo);

 // Se verifica que el ENVASE exista..

        EXEC SQL SELECT count(*)
                                INTO :sql_count
                                FROM UL_ENVASE
                        WHERE institucion_empaque = :sql_institucion_empaque
                          AND consecutivo         = :sql_consecutivo;

 // Hago el insert en la tabla
        if (sql_count == 1) {
                EXEC SQL SELECT //institucion_empaque_padre,
                                                        //consecutivo_padre,
                                                        status_envase,
                                                        estado_fisico
                                        INTO      //:sql_institucion_empaque_padre,
                                                        //:sql_consecutivo_padre,
                                                        :sql_status_envase,
                                                        :sql_estado_fisico
                                        FROM UL_ENVASE
                                WHERE institucion_empaque = :sql_institucion_empaque
                              AND consecutivo         = :sql_consecutivo;

                EXEC SQL COMMIT;
             *ai_status_envase = sql_status_envase;
             *ai_estado_fisico = sql_estado_fisico;

        } else {
                resultado = -5;
                Rollback();
        }

        return resultado;
}

Boolean ulsifca::db_ES_DENOM_GRAL(UInt ai_especimen)
{
```

```
        EXEC SQL BEGIN DECLARE SECTION;
                unsigned short sql_especimen;
                unsigned short sql_count;
        EXEC SQL END DECLARE SECTION;

        // Las pongo en variables locales
        sql_especimen = ai_especimen;

 // Se verifica que el ESPECIMEN sea de DEOMINACION_GENERAL ..
        EXEC SQL SELECT count(*)
                                INTO :sql_count
                                FROM UL_ESPECIMEN_DENOM_GRAL
                        WHERE especimen    = :sql_especimen;

 // Si tuvo exito el query, es un especimen de DENOMINACION_GENERAL,
        if (sql_count != 0)
                return true;
        else    /// sino COMA ... no
                return false;
}

Boolean ulsifca::db_VALIDA_HIJO_EN_PADRE(char ac_tipo_envase_padre[3], UInt ai_especimen, char
ac_tipo_envase_hijo[3])
{

        EXEC SQL BEGIN DECLARE SECTION;
                char      sql_tipo_envase_padre[3];
                unsigned short      sql_especimen;
                char      sql_tipo_envase_hijo[3];
                unsigned short sql_count;
        EXEC SQL END DECLARE SECTION;

        // Las pongo en variables locales

        sql_especimen = ai_especimen;
        StrCopy(sql_tipo_envase_padre,  ac_tipo_envase_padre);
        StrCopy(sql_tipo_envase_hijo,  ac_tipo_envase_hijo);

 // Se verifica que el LA COMBINACION EXISTA ..

        EXEC SQL SELECT count(*)
                                INTO :sql_count
                                FROM TIPO_ENVASE_ESPECIMEN_HIJOS
                        WHERE tipo_envase = :sql_tipo_envase_padre
                          AND especimen    = :sql_especimen
                          AND tipo_envase_hijo = :sql_tipo_envase_hijo;

 // Si tuvo exito el query, es valida la combinacion,
        if (sql_count != 0)
                return true;
        else    /// sino COMA ... no
                return false;

}

SByte ulsifca::db_getCodigoEnvase(Long  ai_institucion_empaque,
                                        char  ac_consecutivo[7],
                                        UInt*  ai_especimen,
                                        char ac_tipo_envase[3],
                                        char ac_digito_verificador[2])
 {
        SByte resultado = 0;

        EXEC SQL BEGIN DECLARE SECTION;
                long    sql_institucion_empaque;
                char  sql_consecutivo[7];
                unsigned short sql_especimen;
                char  sql_tipo_envase[3];
                char  sql_digito_verificador[2];
                unsigned short    sql_count = 0;
        EXEC SQL END DECLARE SECTION;

// Las pongo en variables locales

        sql_institucion_empaque = ai_institucion_empaque;
        StrCopy(sql_consecutivo,  ac_consecutivo);

 // Se verifica que el ENVASE exista..

        EXEC SQL SELECT count(*)
                                INTO :sql_count
                                FROM UL_ENVASE
                        WHERE institucion_empaque = :sql_institucion_empaque
                          AND consecutivo         = :sql_consecutivo;

 // Hago el insert en la tabla
        if (sql_count == 1) {

                EXEC SQL SELECT especimen,
                                                        tipo_envase,
```

```
                                                        digito_verificador
                                        INTO    :sql_especimen,
                                                :sql_tipo_envase,
                                                :sql_digito_verificador
                                        FROM UL_ENVASE
                                WHERE institucion_empaque = :sql_institucion_empaque
                                AND consecutivo           = :sql_consecutivo;

                EXEC SQL COMMIT;

                *ai_especimen = sql_especimen;
        StrCopy(ac_digito_verificador,sql_digito_verificador);
        StrCopy(ac_tipo_envase, sql_tipo_envase);

        } else {
                resultado = -5;
                Rollback();
        }
        return resultado;
 }

SByte ulsifca::db_Actualiza_Status_Hijos(Long  ai_institucion_empaque_padre,
                                                        char   ac_consecutivo_padre[7],
                                                        Int    ai_nuevo_status_envase)
{
        EXEC SQL BEGIN DECLARE SECTION;
                unsigned short sql_count;
                short sql_status_envase;
                long   sql_institucion_empaque_padre;
                char   sql_consecutivo_padre[7];
        EXEC SQL END DECLARE SECTION;

// Las pongo en variables locales

        sql_institucion_empaque_padre = ai_institucion_empaque_padre;
        sql_status_envase = ai_nuevo_status_envase;
        StrCopy(sql_consecutivo_padre,  ac_consecutivo_padre);

        EXEC SQL SELECT count(*)
                  INTO  :sql_count
                  FROM  UL_ENVASE
                 WHERE  institucion_empaque_padre = :sql_institucion_empaque_padre
                   AND          consecutivo_padre = :sql_consecutivo_padre
                   AND                    status_envase = 0;

        if (sql_count > 0) {
                EXEC SQL UPDATE UL_ENVASE
                                        SET UL_ENVASE.status_envase = :sql_status_envase,
                                                UL_ENVASE.consecutivo_padre = UL_ENVASE.consecutivo,
                                                UL_ENVASE.institucion_empaque_padre =
UL_ENVASE.institucion_empaque
                                                WHERE UL_ENVASE.institucion_empaque_padre =
:sql_institucion_empaque_padre
                                                AND UL_ENVASE.consecutivo_padre = :sql_consecutivo_padre
                                                AND UL_ENVASE.status_envase = 0;

           EXEC SQL COMMIT;

                return 0;
        }
        else
                return -1;
}

SByte ulsifca::db_Actualiza_Status_Padre(Long  ai_institucion_empaque,
                                                        char   ac_consecutivo[7],
                                                        Int    ai_nuevo_status_envase)
{
        EXEC SQL BEGIN DECLARE SECTION;
                unsigned short sql_count;
                short sql_status_envase;
                long   sql_institucion_empaque;
                char   sql_consecutivo[7];
        EXEC SQL END DECLARE SECTION;

// Las pongo en variables locales
        sql_institucion_empaque = ai_institucion_empaque;
        sql_status_envase = ai_nuevo_status_envase;
        StrCopy(sql_consecutivo,  ac_consecutivo);

        EXEC SQL SELECT count(*)
                  INTO  :sql_count
                  FROM  UL_ENVASE
                 WHERE  consecutivo = :sql_consecutivo
                   AND  institucion_empaque = :sql_institucion_empaque;

        if (sql_count > 0) {
                EXEC SQL UPDATE UL_ENVASE                       SET        status_envase = :sql_status_envase
                                WHERE  consecutivo_padre = :sql_consecutivo
```

```
                                        AND institucion_empaque = :sql_institucion_empaque;

                return 0;
        }
        else
                return -1;
}


#if defined( __MWERKS__ )
/*****************************************************************************
* Funcion PalmLaunch
* Fecha         :06JULIO2000
* Autor         : Oscar g.
* Descripcion: Funcion especial para dispsitivos PALM donde se inicia una sesion
*                                                       can la base  de datos UltraLite
* Paramteros    : ninguno
* regresa                       : true.- para ndicar que aslio bien
*               false.- para indicar que algo salio mal.
*****************************************************************************/
Int ulsifca::PalmLaunch( void )
{
    if( m_SynchMethod == SYNCH_HOTSYNC )
    {
                        if( ULPalmLaunch( &sqlca, ULPalmDBStream() ) )
                        {
                        // Esta es la primera vez que la aplicacion ha sido abierta.
                        return( 1 );
                        }

    }
    else
    {
                        if( ULPalmLaunch( &sqlca, NULL ) )
                        {
            // Esta es la primera vez que la aplicacion ha sido abierta.
            return( 1 );
                        }
    }
    return( 0 );
}

/*****************************************************************************
* Funcion PalmExit
* Fecha         :06JULIO2000
* Autor         : Oscar g.
* Descripcion: Funcion especial para dispsitivos PALM donde se cierra una sesion
*                                                       can la base  de datos UltraLite
* Paramteros    : Ninguno
* regresa                       : Nada
*****************************************************************************/
void ulsifca::PalmExit( void )

{
        if( m_SynchMethod == SYNCH_HOTSYNC )
        {
        ULPalmExit( &sqlca, m_LectoraID, ULPalmDBStream() );
  }
  else
  {
                ULPalmExit( &sqlca, NULL, NULL );
  }
}
#endif
```

APENDICE B

```
/************************************ APERTURA ***************************************/
#include <Pilot.h>
#include <SysEvtMgr.h>
#include "MainFormRsc.h"
#include "utils.h"
#include "sqlca.h"
#include "ul_sifcadb.h"

#ifdef __cplusplus                      //Como estoy trabajanco con C++ tengo que añadir esto para el
manejo
extern "C" {                            //del lector de codigo de barras.
#endif

#include "ScanMgrDef.h"                         // Scan Manager constantes
#include "ScanMgrStruct.h"                      // Scan Manager structura
#include "ScanMgr.h"                                    // Scan Manager API functiones

#ifdef  __cplusplus
}
#endif

/************************************ GLOBALES ***************************************/
static char g_code128[18];
static char g_etiqueta[18];

static UInt   gi_total_contenedores = 0;
static UInt   gi_Folio_Local;
static UInt   gi_Tipo_Oper_Local;

static Boolean gb_cursorAbiertoCodigos = false;
static DateTimeType     gd_date;

#define maxCBlen -6
/*************************************************************************
 *
 * FUNCION:     AperturaFormInit
 *
 * DESCRIPCION: Inicializa la forma de Cambio de Estado Fisico
 *
 * PARAMETROS:          Apuntador a la Forma.
 *
 * REGRESA:             nada.
 *
 *************************************************************************/
static void AperturaFormInit(FormPtr frmP);
static void AperturaFormInit(FormPtr frmP)
{
   char lc_folio[6];

   gi_Tipo_Oper_Local = getTipoOperacion();
   gb_cursorAbiertoCodigos = false;

        // Inicializo el lector de codigo de barras
        AbreLector();

        gi_Folio_Local =  getUltimoFolio();

        StrIToA(lc_folio,gi_Folio_Local);
        SetFieldText( AperturaFolioField, lc_folio, 6, true);

        if (I_UL_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local, 0, getFecha(), getemp_ID()) != 0 )
                FrmCustomAlert(ErroretiquetaAlert," no se inserto en UL_OPERACION", NULL, NULL);
}

/*************************************************************************
 *
 * FUNCION:     HandleReadCodeApertura
 *
 * DESCRIPCION: La llamo cuando se realiza la lectura de un codigo de barras. LLevo
 *                              a cabo las operaciones de:
 *
 *                                      - Decodificar el codigo de barras
 *                                      - Abrir el cursor de etiquetas leidas
 *                                      - Mostrar la lista nueva
 * PARAMETROS:          Ninguno.
 *
 * REGRESA:             cero.
 *
 *************************************************************************/
void HandleReadCodeApertura();
void HandleReadCodeApertura()
{
        SByte  li_resul_EnvaseInfo;
        Int li_status;
        UInt li_estado_fisico;

        // Decodifico el dato
        if ( DecodificaDato(AperturaUEField, g_code128)) {
                li_resul_EnvaseInfo = of_getEnvaseInfo(g_code128, &li_status,  &li_estado_fisico);
                // Inserto como libre en UL_ENVASE
```

```
                if (li_resul_EnvaseInfo == 0) {

                        switch (li_status) {
                        case ABIERTO:
                        case MISMO_STATUS_PADRE:
                        case EN_TRANSITO:
                                FrmCustomAlert(ErroretiquetaAlert," no puede participar en una APERTURA",
NULL, NULL);
                                break;
                        case DISPONIBLE_EN_CAJA:
                                if (I_UL_D_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local, g_code128) != 0
)
                                        FrmCustomAlert(ErroretiquetaAlert," no se inserto en
UL_D_OPERACION", NULL, NULL);
                                break;
                        }
                }

                if (li_resul_EnvaseInfo != 0) {
                        if (getContingencia()) {
                                if (I_UL_ENVASE(g_code128,g_code128, ABIERTO, 0) != 0 )
                                        FrmCustomAlert(ErroretiquetaAlert," no se inserto en UL_ENVASE",
NULL, NULL);
                                else
                                        if (I_UL_D_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local,
g_code128) != 0 ) {
                                                FrmCustomAlert(ErroretiquetaAlert," no se inserto en
UL_D_OPERACION", NULL, NULL);
                                                if (D_UL_ENVASE(g_code128) != 0)
                                                        FrmCustomAlert(ErroretiquetaAlert," no se inserto
en UL_D_OPERACION y no se pudo eliminar de UL_ENVASE", NULL, NULL);
                                                }
                                } else
                                        FrmCustomAlert(ErroretiquetaAlert," no ha participado en una preparacion
de desarmado", NULL, NULL);
                        }
        }
}

/*****************************************************************************
 *
 * FUNCION:     Actualiza_Status_Apertura
 *
 * DESCRIPCION:
 * PARAMETROS:
 *
 * REGRESA:
 *
 *****************************************************************************/
 void Actualiza_Status_Apertura(void);
 void Actualiza_Status_Apertura(void)
 {

        SByte li_total_x_oper;
        UInt li_codigo;
        char  lc_codigo[18];

        li_total_x_oper = of_get_Etq_Detalle_una_ope(gi_Tipo_Oper_Local, gi_Folio_Local);

        if(!gb_cursorAbiertoCodigos) {
                AbrirCursorCodigos_una_Ope(gi_Tipo_Oper_Local, gi_Folio_Local);
                gb_cursorAbiertoCodigos = true;
        }

        for (li_codigo = 1; li_codigo <= li_total_x_oper; li_codigo++) {
                GetNextCodigo_una_Ope(li_codigo, lc_codigo);
                if (U_UL_ENVASE_Status(lc_codigo, ABIERTO ) != 0)
                        FrmCustomAlert(ERRORAlert,"No fue posible actualizar el STATUS de ", lc_codigo,
"");
                else {
                        if (of_Actualiza_Status_Hijos(lc_codigo, DISPONIBLE_EN_CAJA) != 0)
                                FrmCustomAlert(ERRORAlert,"No fue posible actualizar el STATUS de los
hijos de ", lc_codigo, "");
                        }
        }

        if(gb_cursorAbiertoCodigos) {
                CerrarCursorCodigos_una_Ope();
                gb_cursorAbiertoCodigos = false;
        }
 }

/*****************************************************************************
 *
 * FUNCTION:    AperturaFormDoButton
 *
 * DESCRIPTION: Esta rutina lleva a cabo las acciones del boton correspondiente
 *
 * PARAMETERS:  buttonId  - id del boton
 *
```

```
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 ***********************************************************************/
static Boolean AperturaFormDoButton(Word buttonId);
static Boolean AperturaFormDoButton(Word buttonId)
{
        UInt16 li_boton;
   char lc_codigo[18];

        Boolean handled = false;
        switch (buttonId)
                {
                        case AperturaHechoButton:
                                if(gb_cursorAbiertoCodigos)
                                        CerrarCursorCodigos_una_Ope();
                                CierraLector();
                                if (ValidaInsersion_UL_OPERACION(gi_Tipo_Oper_Local,gi_Folio_Local))
                                        Actualiza_Status_Apertura();
                                FrmGotoForm(MainForm);
                                handled =true;
                                break;

                        case  AperturaConsultaButton:
                                ConsultarCodigos(AperturaUEList, gi_Tipo_Oper_Local, gi_Folio_Local,
&gb_cursorAbiertoCodigos);
                        handled = true;
                                break;

                        case AperturaEliminarButton:
                                li_boton = DisplayEliminarContenedor(EliminaContenedorForm, lc_codigo);
                                if (li_boton == EliminaContenedorOKButton) {
                                        EliminarContenedor(gi_Tipo_Oper_Local, gi_Folio_Local, lc_codigo,
0, false);
                                        DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local,
AperturaTotalField);
                                }
                                handled =true;
                                break;
                }
        return handled;
}

/***********************************************************************
 *
 * FUNCTION:    AperturaFormHandleEvent
 *
 * DESCRIPTION: This routine is the event handler for the
 *              "CambioEdoFisForm" of this application.
 *
 *
 * PARAMETERS:  eventP  - a pointer to an EventType structure
 *
 * RETURNED:    true if the event has handle and should not be passed
 *              to a higher level handler.
 *
 * REVISION HISTORY:
 *
 ***********************************************************************/
Boolean AperturaFormHandleEvent(EventPtr eventP);
Boolean AperturaFormHandleEvent(EventPtr eventP)
{
    Boolean handled = false;
    FormPtr frmP;

        switch (eventP->eType)
                {
                case ctlSelectEvent:
                        handled = AperturaFormDoButton(eventP->data.ctlSelect.controlID);
                        break;

                case frmOpenEvent:
                        frmP = FrmGetActiveForm();
                        AperturaFormInit( frmP);
                FrmDrawForm ( frmP);
                DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local, AperturaTotalField);
                        handled = true;
                        break;

                case scanDecodeEvent:
                        HandleReadCodeApertura();
                DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local, AperturaTotalField);
                        handled = true;
                        break;

                default:
                        break;
                }
```

```
        return handled;
}

/********************************** ARMADO ***********************************/

#include <Pilot.h>
#include <SysEvtMgr.h>
#include "MainFormRsc.h"
#include "utils.h"
#include "sqlca.h"
#include "ul_sifcadb.h"

#ifdef __cplusplus                      //Como estoy trabajanco con C++ tengo que añadir esto para el
manejo
extern "C" {                            //del lector de codigo de barras.
#endif

#include "ScanMgrDef.h"                         // Scan Manager constantes
#include "ScanMgrStruct.h"                      // Scan Manager structura
#include "ScanMgr.h"                                    // Scan Manager API functiones

#ifdef  __cplusplus
}
#endif

/************************************ GLOBALES ***********************************/

static char g_code128[18];
static char g_code_padre[18];

static UInt   gi_Folio_Local;
static UInt   gi_Tipo_Oper_Local;

static Boolean gb_cursorAbiertoCodigos;
static Boolean gb_cursorAbiertoEdoFis;
static Boolean gb_cursorAbiertoEspecimen;

static Boolean gb_d_armado ;                                    // FLAG: ya he leido el D_ARMADO;
static UInt gi_clave_estado_fisico;
char    gc_tipo_envase_padre[3];
UInt    gi_especimen_padre;
UInt  gi_estado_fisico_padre;
static UInt gi_clv_especimen;
char  gc_monto[20];

/*****************************************************************************
 *
 * FUNCTION:    EstadoFisicoDrawFunc
 *
 * DESCRIPTION: Funcion CALLBACK para dibujar la lista dinamicamente
 *
 * PARAMETERS:  UInt itemNum- numero de campos
 *                              RectanglePtr bounds- limites del rectangulo
 *                              CharPtr *data- No se
 *
 * RETURNED:    nada
 *
 * REVISION HISTORY:
 *
 *****************************************************************************/
void EstadoFisicoDrawFunc(UInt itemNum, RectanglePtr bounds, CharPtr *data);
void EstadoFisicoDrawFunc(UInt itemNum, RectanglePtr bounds, CharPtr *data)
{
        itemNum ++;

        UInt li_c;
        // hacer el llamado a la funcion de la base de datos para obtener la sig. etiq.
        GetNextEstadoFisico(itemNum, &li_c, g_code128);
        DrawCharToFitWidth(g_code128, bounds);
 }

/*****************************************************************************
 *
 * FUNCTION:    ConsultarEstadoFisico
 *
 * DESCRIPTION: Dibuja la lista de etiquetas libres que crea dinamicamente
 *
 * PARAMETERS:  nada
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *****************************************************************************/
static void ConsultarEstadoFisico(void)
{
        UInt li_totalEstados = 0;

        if(gb_cursorAbiertoEdoFis) {
```

```
                CerrarCursorEstadoFisico();
                gb_cursorAbiertoEdoFis  =false;
        }

    // Obtengo el numero de etiquetas en la tabla
        li_totalEstados = of_getEstadoFisicoCount();

        if(li_totalEstados > 0){
                // Inicializo la lista
                iniciaLista(ArmadoListaEdoFisicoList, li_totalEstados, (ListDrawDataFuncPtr)
EstadoFisicoDrawFunc, true);

                // Abro el cursor
                AbrirCursorEstadoFisico();
                gb_cursorAbiertoEdoFis  =true;

                // Dibujo la lista
                LstDrawList((ListPtr) GetObjectPtr(ArmadoListaEdoFisicoList));
        }
        else
                FrmCustomAlert(NoCatalogoAlert, "ESTADO_FISICO.", NULL, NULL);
}
/*****************************************************************************
 *
 * FUNCTION:    SwapPushButtons
 *
 * DESCRIPTION: Intercambia el estado de los Push buttons de
 *                                      Unidad de Empaque / Contenido
 *
 * PARAMETERS:
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *****************************************************************************/
static void SwapPushButtons(void) {
   FormPtr frmP = FrmGetActiveForm();
   Byte li_UE;

   li_UE = CtlGetValue((ControlPtr )FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP,ArmadoUEPushButton)));

        if (li_UE == 1) { /* 0 = OFF, 1 = ON */
           CtlSetValue((ControlPtr )FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP,ArmadoUEPushButton))
,0);
        CtlSetValue((ControlPtr )FrmGetObjectPtr(frmP,
FrmGetObjectIndex(frmP,ArmadoContenidoPushButton)),1);
        } else {
           CtlSetValue((ControlPtr )FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP,ArmadoUEPushButton))
,1);
        CtlSetValue((ControlPtr )FrmGetObjectPtr(frmP,
FrmGetObjectIndex(frmP,ArmadoContenidoPushButton)),0);
        }
}

/*****************************************************************************
 *
 * FUNCION:     ArmadoFormInit
 *
 * DESCRIPCION: Inicializa la forma de Armado
 *
 * PARAMETROS:          Apuntador a la Forma.
 *
 * REGRESA:             nada.
 *
 *****************************************************************************/
static void ArmadoFormInit(FormPtr frmP)
{
   char lc_folio[6];
   gi_Tipo_Oper_Local = getTipoOperacion();
        gb_cursorAbiertoCodigos = false;
        gb_d_armado = false;
        gi_clave_estado_fisico = 0;
        StrCopy(g_code_padre, "");
        gb_cursorAbiertoEspecimen = false;

        // Inicializo el lector de codigo de barras
        AbreLector();

        CtlSetValue((ControlPtr )FrmGetObjectPtr(frmP, FrmGetObjectIndex(frmP,ArmadoUEPushButton)) ,0);
/* 0 = OFF, 1 = ON */
        SwapPushButtons();

        // Despliego la etiqueta de la lista del catalogo de Edo Fis.
        SetFieldText( ArmadoSeleccionField,  "Estado Fisico", 15, true);

        // Asigno un numero de folio, lo despliego  y actualizo el global y
        gi_Folio_Local = getUltimoFolio();
        StrItoA(lc_folio,gi_Folio_Local);
        SetFieldText( ArmadoFolioField, lc_folio, 6, true);
```

```
        if (I_UL_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local, gi_clave_estado_fisico, getFecha(),
getemp_ID()) != 0 )
                FrmCustomAlert(ERRORAlert,"NO se registro en UL_OPERACION", "","");

        if (getContingencia())
                SetFieldText( ArmadoContingenciaField, "CONTINGENCIA", 25, true);
}

/*****************************************************************
 *
 * FUNCION:      HandleReadArmado
 *
 * DESCRIPCION: La llamo cuando se realiza la lectura de un codigo de barras. LLevo
 *                              a cabo las operaciones de:
 *
 *                                      - Decodificar el codigo de barras
 *                                      - Abrir el cursor de etiquetas leidas
 *                                      - Mostrar la lista nueva
 * PARAMETROS:          Ninguno.
 *
 * REGRESA:             cero.
 *
 *****************************************************************/
void HandleReadArmado();
void HandleReadArmado()
{
  FormPtr frmP = FrmGetActiveForm();
        SByte  li_resultado = 0;
        SByte  li_resul_EnvaseInfo = 0;
        Boolean lb_inserta_ENVASE = false;
        Int li_status;
        UInt li_estado_fisico;
        UInt li_estado_fisico_hijo = 0;
        char lc_tipo_envase_hijo[3];
        UInt li_especimen_hijo;

                if (CtlGetValue((ControlPtr )FrmGetObjectPtr(frmP,
FrmGetObjectIndex(frmP,ArmadoUEPushButton))))
                {
//***** MANEJO DEL PADRE.
                        if (DecodificaDato(ArmadoUEField, g_code128)) {  // El que contiene (U)
                                li_resul_EnvaseInfo = of_getEnvaseInfo(g_code128, &li_status,
&li_estado_fisico);
                                if (li_resul_EnvaseInfo != 0 ) {        //SI NO EXISTE EN UL_ENVASE
                                        if (I_UL_ENVASE(g_code128, g_code128, DISPONIBLE_EN_CAJA,
gi_clave_estado_fisico) != 0 )
                                                FrmCustomAlert(ADVERTENCIAAlert,"La Unidad de
Empaque ",
                                "NO se inserto en UL_ENVASE", "");
                                        else {
                                                if (!gb_d_armado) {     // Si no se ha leido el D_ARMADO
                                                        if ( I_UL_D_ARMADO(gi_Tipo_Oper_Local,
gi_Folio_Local, g_code128) != 0 ) {
                                                                FrmCustomAlert(INFORMACIONAlert,"La
Unidad de Empaque ",
                                "NO se inserto en UL_D_ARMADO", "");
                                                                D_UL_ENVASE(g_code128);
                                                        } else {
                                                                gb_d_armado = true;
                                                                StrCopy(g_code_padre, g_code128);
                                                                of_getEnvaseTipoEnvase(g_code_padre,
gc_tipo_envase_padre);
                                                                gi_especimen_padre =
of_getEnvaseEspecimen(g_code_padre);
                                                                gi_estado_fisico_padre =
gi_clave_estado_fisico;
                                                                SwapPushButtons();
                                                        }
                                                        // end if/else I_UL_D_ARMADO
                                                } else {                        // Si ya se habia
leido el D_ARMADO
                                                        if
(of_get_Etq_Detalle_una_ope(gi_Tipo_Oper_Local, gi_Folio_Local) == 0) {
                                                                if ( U_UL_D_ARMADO(gi_Tipo_Oper_Local,
gi_Folio_Local, g_code128) != 0) {
                                                                FrmCustomAlert(INFORMACIONAlert,"La Unidad de Empaque ",
                                "NO se actualizo en UL_D_ARMADO", "");
                                                                D_UL_ENVASE(g_code128);
                                                        }       else {
                                                                StrCopy(g_code_padre, g_code128);
                                                                of_getEnvaseTipoEnvase(g_code_padre, gc_tipo_envase_padre);
                                                                gi_especimen_padre =
of_getEnvaseEspecimen(g_code_padre);
```

```
gi_clave_estado_fisico;                                                    gi_estado_fisico_padre =
                                                                           SwapPushButtons();
                                                                       }
                                                                  } else
                                                                       FrmCustomAlert(ERRORAlert,"NO se puede
modificar ",
                        "la Unidad de Empaque PADRE ",
                        "cuando ya tiene CONTENIDO");
                                                  } // end if/ else gb_d_armao
                                             } // end ELSE
                                        } // end if no existe
                                        if (    (li_resul_EnvaseInfo == 0)                    // SI EXISTE EN
UL_ENVASE
                                                && (    (li_estado_fisico == 0)                    // Y ya
sea que tiene Edo. Fis. Cero
                                                    || (    (li_estado_fisico == gi_clave_estado_fisico)
                                                            && (li_estado_fisico != 0)))) // O bien
tiene el nuevo que se asignara...
                                        {
                                                switch (li_status) {
                                                        case MISMO_STATUS_PADRE:
                                                        case EN_TRANSITO:
                                                                FrmCustomAlert(ERRORAlert, " NO es posible
utilizar esa Unidad de Empaque",
                        " para un ARMADO","");
                                                                break;
                                                        case ABIERTO:
                                                        case DISPONIBLE_EN_CAJA:
                                                                if (!gb_d_armado) {
                                                                        if ( I_UL_D_ARMADO(gi_Tipo_Oper_Local,
gi_Folio_Local, g_code128) != 0 )
                                                                                FrmCustomAlert(INFORMACIONAlert,"
La Unidad de Empaque ",
                                                                "NO se inserto en UL_D_ARMADO", "");
                                                                        else {
                                                                                gb_d_armado = true;
                                                                                strCopy(g_code_padre, g_code128);
of_getEnvaseTipoEnvase(g_code_padre, gc_tipo_envase_padre);
                                                                                gi_especimen_padre =
of_getEnvaseEspecimen(g_code_padre);
                                                                                SwapPushButtons();
                                                                                gi_estado_fisico_padre =
li_estado_fisico;
                                                                                FrmCustomAlert(RASTREOAlert,"La
Unidad de Empaque ",
                                "se inserto en UL_D_ARMADO", "");
                                                                        }
                                                                } else {
/// Para validar que no se haya leido ya el detalle....
                                                                        if
(of_get_Etq_Detalle_una_ope(gi_Tipo_Oper_Local, gi_Folio_Local) == 0) {
                                                                                if (
U_UL_D_ARMADO(gi_Tipo_Oper_Local, gi_Folio_Local, g_code128) != 0)
FrmCustomAlert(INFORMACIONAlert,"La Unidad de Empaque ",
                                                                "NO se actualizo en UL_D_ARMADO", "");
                                                                                else {
                                                                                        strCopy(g_code_padre,
g_code128);
of_getEnvaseTipoEnvase(g_code_padre, gc_tipo_envase_padre);
                                                                                        gi_especimen_padre =
of_getEnvaseEspecimen(g_code_padre);
                                                                                        SwapPushButtons();
                                                                                        gi_estado_fisico_padre =
li_estado_fisico;
FrmCustomAlert(RASTREOAlert," La Unidad de Empaque ",
                                                                "se actualizo  en UL_D_ARMADO", "");
                                                                                }
                                                                        } else
                                                                                FrmCustomAlert(ERRORAlert," NO se
puede modificar ",
                                "la Unidad de Empaque PADRE ",
                                "cuando existe  CONTENIDO");
```

```
                                                    }
                                                    break;
                                            default:
                                                    FrmCustomAlert(ERRORAlert, "STATUS NO valido para
",g_code128,"");
                                                    break;
                                    } // end case
                            } // end if SI existe

///********* MANEJO DE LOS HIJOS ***************
                            } else {
                            if (DecodificaDato(ArmadoContenidoField, g_code128)) { // El contenido (C)
                                    if (gb_d_armado) {
                                    li_resul_EnvaseInfo = of_getEnvaseInfo(g_code128, &li_status,
&li_estado_fisico_hijo);

                                    of_getEnvaseTipoEnvase(g_code128, lc_tipo_envase_hijo);
                                    li_especimen_hijo = of_getEnvaseEspecimen(g_code128);


                                    if (li_estado_fisico_hijo == 0 ) li_estado_fisico_hijo =
gi_clave_estado_fisico;

                                    if ((gi_clave_estado_fisico != 0) &&
                                            (of_Valida_Hijo_en_Padre(gc_tipo_envase_padre,
lc_tipo_envase_hijo,
gi_especimen_padre, li_especimen_hijo,
gi_clave_estado_fisico, li_estado_fisico_hijo,
gi_Folio_Local, g_code128)))
                                    {
                                    if (li_resul_EnvaseInfo != 0) { //SI NO EXISTE EN UL_ENVASE
                                            if (getContingencia()){ /// No existe pero estamos en
contingencia
                                                    if (I_UL_ENVASE(g_code128, g_code128,
DISPONIBLE_EN_CAJA, gi_clave_estado_fisico) != 0 )
                                                            FrmCustomAlert(ADVERTENCIAAlert,"La
Unidad de Empaque ",
                                    "NO se inserto en UL_ENVASE", "");
                                                    else {
                                                            if ( I_UL_D_OPERACION(gi_Tipo_Oper_Local,
gi_Folio_Local, g_code128) != 0 ) {
                                                                    FrmCustomAlert(ADVERTENCIAAlert,
"La Unidad de Empaque ",
                                            "NO se inserto en UL_D_OPERACION", "");
                                                                    D_UL_ENVASE(g_code128);
                                                            } // End if I_UL_D_OPERACION

                                                    } // end ELSE
                                    } // end IF gd_contingencia
                                    else                                            /// No
existe y es operacion normal.. NO LO ACEPTA.
                                                    FrmCustomAlert(ADVERTENCIAAlert,"Unidad de
Empaque no valida. ",
                                    "NO registrada para ",
                                    "ARMADO / DESARMADO");
                                    } // end if no existe
                                    if (li_resul_EnvaseInfo == 0)   // SI EXISTE EN UL_ENVASE
                                            switch (li_status) {
                                                    case ABIERTO:

                                                    case MISMO_STATUS_PADRE:

                                                    case EN_TRANSITO:
                                                            FrmCustomAlert(ERRORAlert, " Unidad de
Empaque NO valida ",
                                    " como contenido ","en un ARMADO");
                                                            break;
                                                    case DISPONIBLE_EN_CAJA:
                                                            if ( I_UL_D_OPERACION(gi_Tipo_Oper_Local,
gi_Folio_Local, g_code128) != 0 )
                                                                    FrmCustomAlert(ADVERTENCIAAlert,"
La Unidad de Empaque ",

                                            "NO se inserto en UL_D_OPERACION", "");
                                                            break;
                                                    default:
                                                            FrmCustomAlert(ERRORAlert, "STATUS NO
valido para ",g_code128,"");
                                                            break;
```

```
                                        } // end case
                                    // end if si existe
                                    } else
                                        FrmCustomAlert(ERRORAlert,"Unidad de Empaque con: ",
        "Edo. Fis. / Especimen / Tipo de Envase ",
        "NO valido");                   // if paso validaciones
                    } else
                                        FrmCustomAlert(ERRORAlert,"Unidad de Empaque NO
procesada. ",
        "Falta leer Edo. Fis.", "");
                    } // end else if Decodifica (C)
                // end if CtlGetValue
}
/*****************************************************************
 *
 * FUNCTION:    EspecimenDrawFunc
 *
 * DESCRIPTION: Funcion CALLBACK para dibujar la lista dinamicamente
 *
 * PARAMETERS:  UInt itemNum- numero de campos
 *                          RectanglePtr bounds- limites del rectangulo
 *                          CharPtr *data- No se
 *
 * RETURNED:    nada
 *
 * REVISION HISTORY:
 *
 *****************************************************************/
 void EspecimenDrawFunc(UInt itemNum, RectanglePtr bounds, CharPtr *data);
 void EspecimenDrawFunc(UInt itemNum, RectanglePtr bounds, CharPtr *data)
 {
       char lc_especimen[16];
       UInt li_especimen;
       itemNum ++;

       // hacer el llamado a la funcion de la base de datos para obtener la sig. etiq.
       GetNextEspecimen(itemNum, &li_especimen, lc_especimen);
       DrawCharToFitWidth(lc_especimen, bounds);
 }
/*****************************************************************
 *
 * FUNCTION:    ConsultarEspecimen
 *
 * DESCRIPTION: Dibuja la lista de especimenes que crea dinamicamente
 *
 * PARAMETERS:  nada
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *****************************************************************/
static void ConsultarEspecimen(void)
{
   UInt  li_totalEspecimen=0;

  // Obtengo el numero de etiquetas en la tabla
       li_totalEspecimen = of_getEspecimenCount();

       if(gb_cursorAbiertoEspecimen) {
             cerrarCursorEspecimen();
             gb_cursorAbiertoEspecimen       =false;
       }

       if(li_totalEspecimen > 0){

             // Inicializo la lista
             iniciaLista(DetalleMontoSeleccEspecimenList, li_totalEspecimen, (ListDrawDataFuncPtr)
EspecimenDrawFunc, true);

             // Abro el cursor
             AbrirCursorEspecimen();
             gb_cursorAbiertoEspecimen       =true;


             // Dibujo la lista
             LstDrawList((ListPtr) GetObjectPtr(DetalleMontoSeleccEspecimenList));
       }
       else
             FrmCustomAlert(NoCatalogoAlert, " ESPECIMEN.", NULL, NULL);
}
```

```
/*****************************************************************************
 *
 * FUNCTION:    DisplayDetalleMonto
 *
 * DESCRIPTION: Despliega la forma modal para leer el codigo que se va a eliminar
 *
 * PARAMETERS:  nada
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *****************************************************************************/
UInt DisplayDetalleMonto();
UInt DisplayDetalleMonto()
{
        FormPtr PreviousForm = FrmGetActiveForm();
        FormPtr frm = FrmInitForm(DetalleMontoForm);
        UInt hitButton = 0;

        gi_clv_especimen = 0;
        FrmSetActiveForm(frm);
        char lc_clv_especimen[4];
        char lc_old_monto[21];

        char* Ptr_DetalleMonto;
        char* Ptr_Especimen;

        SetFieldText( DetalleMontoSeleccEspecimenField,  "Especimen      ", 25, false);
        FrmSetEventHandler(frm, DetalleMontoFormHandleEvent);
        ConsultarEspecimen();

        hitButton = FrmDoDialog(frm);

        switch (hitButton) {
                case DetalleMontoHechoButton:
                                Ptr_DetalleMonto = FldGetTextPtr((FieldPtr )FrmGetObjectPtr(frm, (UInt16
)FrmGetObjectIndex(frm, DetalleMontoMontoField)));
                                Ptr_Especimen = FldGetTextPtr((FieldPtr )FrmGetObjectPtr(frm, (UInt16
)FrmGetObjectIndex(frm, DetalleMontoClvEspecimenField)));

                                if (Ptr_DetalleMonto != NULL) StrCopy(gc_monto , Ptr_DetalleMonto);
                                if (Ptr_Especimen != NULL) {
                                        StrCopy(lc_clv_especimen , Ptr_Especimen);
                                        gi_clv_especimen = StrAToi(lc_clv_especimen);

                                }

                                if (gi_clv_especimen != 0)
                                        {
                                        if (of_get_UL_D_MONTO(gi_Tipo_Oper_Local, gi_Folio_Local,
gi_clv_especimen, lc_old_monto) == 0) {
                                                if (U_UL_D_MONTO(gi_Tipo_Oper_Local, gi_Folio_Local,
gi_clv_especimen, gc_monto) != 0 )
                                                        FrmCustomAlert(ERRORAlert,"NO se actualizo en
UL_D_MONTO", "","");
                                        } else {
                                                if (I_UL_D_MONTO(gi_Tipo_Oper_Local, gi_Folio_Local,
gi_clv_especimen, gc_monto) != 0 )
                                                        FrmCustomAlert(ERRORAlert,"NO se registro en
UL_D_MONTO", "","");
                                                }
                                        }

                        break;
                case DetalleMontoCancelarButton:
                        break;
        }

        if (PreviousForm)
                FrmSetActiveForm(PreviousForm);

        FrmDeleteForm(frm);

        return hitButton;
}
/*****************************************************************************
 *
 * FUNCION:     Actualiza_Edo_Fis_y_Status
 *
 * DESCRIPCION:
 * PARAMETROS:
 *
 * REGRESA:
 *
 *****************************************************************************/
void Actualiza_Edo_Fis_y_Status(void);
 void Actualiza_Edo_Fis_y_Status(void)
{
```

```
        SByte li_total_x_oper;
        UInt    li_codigo;
        char  lc_codigo[18];
        char    lc_edo_fis[4];
        char    lc_status[4];

        li_total_x_oper = of_get_Etq_Detalle_una_ope(gi_Tipo_Oper_Local, gi_Folio_Local);

        if(!gb_cursorAbiertoCodigos) {
                AbrirCursorCodigos_una_Ope(gi_Tipo_Oper_Local, gi_Folio_Local);
                gb_cursorAbiertoCodigos = true;
        }

        for (li_codigo = 1; li_codigo <= li_total_x_oper; li_codigo++) {
                GetNextCodigo_una_Ope(li_codigo, lc_codigo);
                if (U_UL_ENVASE_Edo_Fis(lc_codigo,  gi_Clave_estado_fisico ) != 0)
                        FrmCustomAlert(ERRORAlert,"No fue posible actualizar el estado fisico de ",
lc_codigo, lc_edo_fis);
                if (U_UL_ENVASE_Status(lc_codigo,  MISMO_STATUS_PADRE ) != 0)
                        FrmCustomAlert(ERRORAlert,"No fue posible actualizar el status de ", lc_codigo,
lc_status);
                if (U_UL_ENVASE_Padre(lc_codigo,  g_code_padre ) != 0)
                        FrmCustomAlert(ERRORAlert,"No fue posible actualizar el padre de ", lc_codigo,
g_code_padre);
        }

        if(gb_cursorAbiertoCodigos) {
                CerrarCursorCodigos_una_Ope();
                gb_cursorAbiertoCodigos = false;
        }
}
/*****************************************************************************
 *
 * FUNCTION:    ArmadoFormDoButton
 *
 * DESCRIPTION: Esta rutina lleva a cabo las acciones del boton correspondiente
 *
 * PARAMETERS:  buttonId  - id del boton
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *****************************************************************************/
static Boolean ArmadoFormDoButton(UInt buttonId)
{
        Boolean handled = false;
        char lc_codigo[18];
        UInt li_boton;
        Int li_status;
        UInt li_estado_fisico;

        switch (buttonId)
                {
                        case ArmadoHechoButton:
                                if (of_get_Etq_Detalle_una_ope(gi_Tipo_Oper_Local, gi_Folio_Local) != 0)
{
                                   if (gi_clave_estado_fisico == 0 ){
                                        FrmCustomAlert(INFORMACIONAlert,"No es posible registrar
la operacion sin haber seleccionado",
                " un Estado Fisico","");
                                        return true;
                                   }
                                   if (of_getEnvaseInfo(g_code_padre , &li_status,
&li_estado_fisico) != 0) {
                                        FrmCustomAlert(INFORMACIONAlert,"No es posible registrar
la operacion sin haber seleccionado",
                  " una Unidad de Empaque ","'PADRE'");
                                        return true;
                                }
                                if (gb_cursorAbiertoEdoFis)
                                        CerrarCursorEstadoFisico();

                                if (gb_cursorAbiertoEspecimen)
                                        CerrarCursorEspecimen();

                                CierraLector();

                                if(gb_cursorAbiertoCodigos) {
                                        CerrarCursorCodigos_una_Ope();
                                        gb_cursorAbiertoCodigos = false;
                                }

                                if (validaInsersion_UL_OPERACION(gi_Tipo_Oper_Local,gi_Folio_Local))
                                        Actualiza_Edo_Fis_y_Status();
```

```
                        // Regreso a la pantalla principal
                                FrmGotoForm(MainForm);
                                handled =true;
                                break;

                        case ArmadoEliminarButton:
                                li_boton = DisplayEliminarContenedor(EliminaContenedorForm, lc_codigo);
                                if (li_boton == EliminaContenedorOKButton) {
                                        EliminarContenedor(gi_Tipo_Oper_Local, gi_Folio_Local, lc_codigo,
DISPONIBLE_EN_CAJA, true);
                                        DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local,
ArmadoTotalField);
                                }
                                handled =true;
                                break;

                        case ArmadoMontoButton:
                                DisplayDetalleMonto();
                                handled = true;
                                break;

                case ArmadoConsultaButton:
                        ConsultarCodigos(ArmadoContenidoList, gi_Tipo_Oper_Local, gi_Folio_Local,
false);
                                handled = true;
                                break;
                }
        return handled;
}

/*****************************************************************************
 *
 * FUNCTION:    ArmadoFormHandleEvent
 *
 * DESCRIPTION: This routine is the event handler for the
 *              "ArmadoForm" of this application.
 *
 * PARAMETERS:  eventP  - a pointer to an EventType structure
 *
 * RETURNED:    true if the event has handle and should not be passed
 *              to a higher level handler.
 *
 * REVISION HISTORY:
 *
 *****************************************************************************/
Boolean ArmadoFormHandleEvent(EventPtr eventP);
Boolean ArmadoFormHandleEvent(EventPtr eventP)
{
    Boolean handled = false;
    FormPtr frmP;
        char lc_edofisico[16];

        switch (eventP->eType)
                {

                case ctlSelectEvent:
                        handled =ArmadoFormDoButton(eventP->data.ctlSelect.controlID);
                        break;

                case frmOpenEvent:
                        frmP = FrmGetActiveForm();
                        ArmadoFormInit( frmP);
                   FrmDrawForm ( frmP);
                   ConsultarEstadoFisico();
                   DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local, ArmadoTotalField);

                        handled = true;
                        break;

                case scanDecodeEvent:
                        HandleReadArmado();
                   DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local, ArmadoTotalField);

                        handled = true;
                        break;

                case popSelectEvent:
                        if (of_get_Etq_Detalle_una_ope(gi_Tipo_Oper_Local, gi_Folio_Local) == 0) {
                                gi_clave_estado_fisico = 0;
                                GetNextEstadoFisico(eventP->data.popSelect.selection + 1,
&gi_clave_estado_fisico, lc_edofisico);
                                SetFieldText( ArmadoSeleccionField, lc_edofisico, 25, true);
                                if (U_UL_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local,
gi_clave_estado_fisico) != 0 )
                                        FrmCustomAlert(ERRORAlert,"No se actualizo "," el Estado Fisico
", "en UL_OPERACION");
                                else if (gb_d_armado && (U_UL_ENVASE_Edo_Fis(g_code_padre,
gi_clave_estado_fisico )  != 0))
                                        FrmCustomAlert(ERRORAlert,"No se actualizo "," el Estado Fisico
", "de la Unidad de Empaque en UL_ENVASE");
```

```
                        } else
                                        FrmCustomAlert(INFORMACIONAlert,"No es posible cambiar el ",
          "Estado Fisico. ",
          "Ya existe Contenido en la U. de Empaque.");
                                handled = true;
                                break;

                default:
                                break;
                }
        return handled;
}

/*************************************** CAMBIO DE ESTADO FISICO
****************************************/


#include <Pilot.h>
#include <SysEvtMgr.h>
#include "MainFormRsc.h"
#include "utils.h"
#include "sqlca.h"
#include "ul_sifcadb.h"


#ifdef __cplusplus                      //Como estoy trabajanco con C++ tengo que añadir esto para el
manejo
extern "C" {                            //del lector de codigo de barras.
#endif

#include "ScanMgrDef.h"                         // Scan Manager constantes
#include "ScanMgrStruct.h"                      // Scan Manager structura
#include "ScanMgr.h"                                    // Scan Manager API functiones

#ifdef  __cplusplus
}
#endif

/************************************** GLOBALES ***************************************/

static char g_code128[18];

static UInt     gi_Folio_Local;
static UInt     gi_Tipo_Oper_Local = 0;
static Boolean gb_cursorAbiertoEdoFis;
Boolean gb_cursorAbiertoCodigos;
static Boolean gb_HayCatalogo = true;
static UInt gi_clave_estado_fisico;


/*****************************************************************************
 *
 * FUNCTION:    EstadoFisicoDrawFunc : Prototipo
 *
 * DESCRIPTION: Funcion CALLBACK para dibujar la lista dinamicamente
 *              DEFINIDA EN ARMADO*******
 *****************************************************************************/
 void EstadoFisicoDrawFunc(UInt itemNum, RectanglePtr bounds, CharPtr *data);


/*****************************************************************************
 *
 * FUNCTION:    ConsultarEstadoFisico
 *
 * DESCRIPTION: Dibuja la lista de etiquetas libres que crea dinamicamente
 *
 * PARAMETERS:  nada
 *
 * RETURNED:    nothing
 *
 * REVISION HISTORY:
 *
 *
 *****************************************************************************/
static void ConsultarEstadoFisico(void)
{
        UInt li_totalEstados;

        if(gb_cursorAbiertoEdoFis) {
                CerrarCursorEstadoFisico();
                gb_cursorAbiertoEdoFis  =false;
        }

        // Obtengo el numero elementos en el catalogo
        li_totalEstados = of_getEstadoFisicoCount();

        if(li_totalEstados > 0){
```

```
                gb_HayCatalogo = true;

                // Abro el cursor
                AbrirCursorEstadoFisico();
                gb_cursorAbiertoEdoFis  =true;

                // Inicializo la lista
                IniciaLista(CambioEdoFisSeleccEdoFisList, li_totalEstados, (ListDrawDataFuncPtr)
EstadoFisicoDrawFunc, true);

                // Dibujo la lista
                LstDrawList((ListPtr) GetObjectPtr(CambioEdoFisSeleccEdoFisList));
        }
        else {
                FrmCustomAlert(NoCatalogoAlert, "ESTADO_FISICO.", NULL, NULL);
                gb_HayCatalogo = false;
        }
}

/*****************************************************************************
 *
 * FUNCION:     CambioEdoFisFormInit
 *
 * DESCRIPCION: Inicializa la forma de Cambio de Estado Fisico
 *
 * PARAMETROS:          Apuntador a la Forma.
 *
 * REGRESA:             nada.
 *
 *****************************************************************************/

static void CambioEdoFisFormInit()
{
    char lc_folio[6];

   gi_Tipo_Oper_Local = getTipoOperacion();
   gb_cursorAbiertoCodigos = false;

        // Inicializo el lector de codigo de barras
        AbreLector();

        // Muestra la etiqueta de la lista de catalogo de Estados Fisicos
        SetFieldText( CambioEdoFisSeleccEdoFisField, "Estado Fisico", 15, true);


        // Asigno una clave de Estado Fisico, un numero de folio y lo despliego.
        gi_clave_estado_fisico = 0;
        gi_Folio_Local = getUltimoFolio();
        StrIToA(lc_folio,gi_Folio_Local);

        SetFieldText( CambioEdoFisFolioField, lc_folio, 6, true);
        if (I_UL_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local, gi_clave_estado_fisico, getFecha(),
getemp_ID()) != 0 )
                FrmCustomAlert(ERRORAlert,"NO SE REGISTRO EN UL_OPERACION","","");
}

/*****************************************************************************
 *
 * FUNCION:     HandleReadCambioEdoFis
 *
 * DESCRIPCION: La llamo cuando se realiza la lectura de un codigo de barras. LLevo
 *                      a cabo las operaciones de:
 *
 *                                      - Decodificar el codigo de barras
 *                                      - Abrir el cursor de etiquetas leidas
 *                                      - Mostrar la lista nueva
 * PARAMETROS:          Ninguno.
 *
 * REGRESA:             cero.
 *
 *****************************************************************************/
void HandleReadCambioEdoFis();
void HandleReadCambioEdoFis()
{
        Int li_status;
        UInt li_estado_fisico;
        SByte li_resul_EnvaseInfo;

        // Decodifico el dato
        if ( DecodificaDato(CambioEdoFisUEField, g_code128)) {
                li_resul_EnvaseInfo = of_getEnvaseInfo(g_code128, &li_status,  &li_estado_fisico);

                if (li_resul_EnvaseInfo == 0)           //SI YA EXISTE EN UL_ENVASE
                        switch (li_status) {
                                case ABIERTO:
                                case MISMO_STATUS_PADRE:
                                case EN_TRANSITO:
                                        FrmCustomAlert(ERRORAlert, "No es posible cambiarle el Estado
Fisico a esa Unidad de Empaque","","");
                                        break;
```

```
                        case DISPONIBLE_EN_CAJA:
                                if (I_UL_D_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local,
g_code128) != 0 )
                                        FrmCustomAlert(INFORMACIONAlert, "La Unidad de Empaque
","ya se estaba registrada en","esta operacion.");

                                break;
                        default:
                                FrmCustomAlert(ERRORAlert, "STATUS No valido para
",g_code128,"");
                                break;
                        }
                if (li_resul_EnvaseInfo == -5) {        //SI NO EXISTE EN UL_ENVASE
// Se inserta como disponible en caja y por el momento su Edo. Fis.
// es CERO.
                        if (I_UL_ENVASE(g_code128, g_code128, DISPONIBLE_EN_CAJA, 0) != 0 )
                                FrmCustomAlert(ErroretiquetaAlert," no se inserto en UL_ENVASE", NULL,
NULL);
                        else    if (I_UL_D_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local, g_code128) != 0
)
                                FrmCustomAlert(ErroretiquetaAlert," no se inserto en
UL_D_OPERACION", NULL, NULL);
                        }
        }
}

 /*********************************************************************
  *
  * FUNCION:      Actualiza_Edo_Fis
  *
  * DESCRIPCION:
  * PARAMETROS:
  *
  * REGRESA:
  *
  *********************************************************************/

 void Actualiza_Edo_Fis(void);
 void Actualiza_Edo_Fis(void)
 {
        SByte li_total_x_oper;
        UInt li_codigo;
        char  lc_codigo[18];

        li_total_x_oper = of_get_Etq_Detalle_una_ope(gi_Tipo_Oper_Local, gi_Folio_Local);

        if(!gb_cursorAbiertoCodigos) {
                AbrirCursorCodigos_una_Ope(gi_Tipo_Oper_Local, gi_Folio_Local);
                gb_cursorAbiertoCodigos = true;
        }

        for (li_codigo = 1; li_codigo <= li_total_x_oper; li_codigo++) {
                GetNextCodigo_una_Ope(li_codigo, lc_codigo);
                if (U_UL_ENVASE_Edo_Fis(lc_codigo,  gi_Clave_estado_fisico ) != 0)
                        FrmCustomAlert(ERRORAlert,"No fue posible actualizar el estado fisico de ",
lc_codigo,"" );
        }

        if(gb_cursorAbiertoCodigos) {
                CerrarCursorCodigos_una_Ope();
                gb_cursorAbiertoCodigos = false;
        }
 }

/*********************************************************************
  *
  * FUNCTION:     CambioEdoFisFormDoButton
  *
  * DESCRIPTION: Esta rutina lleva a cabo las acciones del boton correspondiente
  *
  * PARAMETERS:  buttonId  - id del boton
  *
  * RETURNED:     nothing
  *
  * REVISION HISTORY:
  *
  *********************************************************************/
static Boolean CambioEdoFisFormDoButton(UInt buttonId)
{
        Boolean handled = false;
        char lc_codigo[18];
        UInt li_boton;

        switch (buttonId)
                {
```

```
                    case CambioEdoFisHechoButton:
                        if ((gi_clave_estado_fisico == 0 ) &&
                                (of_get_Etq_Detalle_una_Ope(gi_Tipo_Oper_Local, gi_Folio_Local)
!= 0)) {
                                FrmCustomAlert(INFORMACIONAlert,"No es posible registrar la
operacion sin haber seleccionado",
        " un Estado Fisico","");
                                return true;
                        }
                        if (gb_cursorAbiertoEdoFis)
                                CerrarCursorEstadoFisico();
                        CierraLector();
                        if(gb_cursorAbiertoCodigos) {
                                CerrarCursorCodigos_una_Ope();
                                gb_cursorAbiertoCodigos = false;
                        }
                        if (ValidaInsersion_UL_OPERACION(gi_Tipo_Oper_Local,gi_Folio_Local))
                                Actualiza_Edo_Fis();
                    // Regreso a la pantalla principal
                        FrmGotoForm(MainForm);
                        handled =true;
                        break;

                    case CambioEdoFisEliminarButton:
                        li_boton = DisplayEliminarContenedor(EliminaContenedorForm, lc_codigo);
                        if (li_boton == EliminaContenedorOKButton) {
                                EliminarContenedor(gi_Tipo_Oper_Local, gi_Folio_Local, lc_codigo,
DISPONIBLE_EN_CAJA, true);
                                DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local,
CambioEdoFisTotalUEField);
                        }
                        handled =true;
                        break;

                case CambioEdoFisConsultaButton:
                        ConsultarCodigos(CambioEdoFisUnidadesEmpaqueList, gi_Tipo_Oper_Local,
gi_Folio_Local, &gb_cursorAbiertoCodigos);
                        handled = true;
                        break;
                }
        return handled;
}

/*****************************************************************************
 *
 * FUNCTION:    CambioEdoFisFormHandleEvent
 *
 * DESCRIPTION: This routine is the event handler for the
 *              "CambioEdoFisForm" of this application.
 *
 * PARAMETERS:  eventP  - a pointer to an EventType structure
 *
 * RETURNED:    true if the event has handle and should not be passed
 *              to a higher level handler.
 *
 * REVISION HISTORY:
 *
 *****************************************************************************/
Boolean CambioEdoFisFormHandleEvent(EventPtr eventP);
Boolean CambioEdoFisFormHandleEvent(EventPtr eventP)
{
    Boolean handled = false;
    FormPtr frmP;
        char lc_edofisico[16];

        switch (eventP->eType)
                {

                case ctlSelectEvent:
                        handled =CambioEdoFisFormDoButton(eventP->data.ctlSelect.controlID);
                        break;

                case frmOpenEvent:
                        frmP = FrmGetActiveForm();
                        CambioEdoFisFormInit();
                FrmDrawForm ( frmP);
                ConsultarEstadoFisico();
                DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local, CambioEdoFisTotalUEField);

                        handled = true;
                        break;

                case scanDecodeEvent:
                        if(gb_HayCatalogo) {
                                HandleReadCambioEdoFis();
                        DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local, CambioEdoFisTotalUEField);

                   } else
                                FrmCustomAlert(INFORMACIONAlert,"No es posible registrar la operacion sin
```

```
el catalogo de Estados Fisicos","","");
                        handled = true;
                        break;

                case popSelectEvent:
                        gi_clave_estado_fisico = 0;
                        GetNextEstadoFisico(eventP->data.popSelect.selection + 1,
&gi_clave_estado_fisico, lc_edofisico);
                        SetFieldText( CambioEdoFisSeleccEdoFisField, lc_edofisico, 25, true);
                        if (U_UL_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local, gi_clave_estado_fisico) !=
0 )
                                FrmCustomAlert(ErroretiquetaAlert," no se actualizo en UL_OPERACION",
NULL, NULL);
                        handled = true;
                        break;

                default:
                        break;
                }
        return handled;
}

/************************************* TRASPASOS ***************************************/

#include <Pilot.h>
#include <SysEvtMgr.h>
#include "MainFormRsc.h"
#include "utils.h"
#include "sqlca.h"
#include "ul_sifcadb.h"


#ifdef __cplusplus                      //Como estoy trabajanco con C++ tengo que añadir esto para el
manejo
extern "C" {                            //del lector de codigo de barras.
#endif

#include "ScanMgrDef.h"                         // Scan Manager constantes
#include "ScanMgrStruct.h"                      // Scan Manager structura
#include "ScanMgr.h"                                    // Scan Manager API functiones

#ifdef  __cplusplus
}
#endif

/************************************* GLOBALES ***************************************/

static char g_code128[18];
static char g_boveda[21];

static UInt   gi_Folio_Local;
static Boolean gb_cursorAbiertoCodigos;

static UInt gi_clave_boveda;
static UInt   gi_Tipo_Oper_Local = 0;
static Boolean gb_cursorAbiertoBov = false;
// Boolean gb_cursorAbiertoCodigos = false;


/*****************************************************************************
 *
 * FUNCTION:    EstadoFisicoDrawFunc : Prototipo
 *
 * DESCRIPTION: Funcion CALLBACK para dibujar la lista dinamicamente
 *              DEFINIDA EN ARMADO *******
 *****************************************************************************/
 void BovedaDrawFunc(UInt itemNum, RectanglePtr bounds, CharPtr *data);
 void BovedaDrawFunc(UInt itemNum, RectanglePtr bounds, CharPtr *data)
{
        itemNum ++;

        UInt li_c;

        // hacer el llamado a la funcion de la base de datos para obtener la sig. etiq.
        GetNextBoveda(itemNum, &li_c, g_boveda);
        DrawCharToFitWidth(g_boveda, bounds);
 }

/*****************************************************************************
 *
 * FUNCTION:    ConsultarEstadoFisico
 *
 * DESCRIPTION: Dibuja la lista de etiquetas libres que crea dinamicamente
 *
 * PARAMETERS:  nada
 *
 * RETURNED:    nothing
 *
```

```
 * REVISION HISTORY:
 *
 *
 ************************************************************************/
static void ConsultarBoveda(void)
{
        UInt li_totalBoveda;

        if(gb_cursorAbiertoBov) {
                CerrarCursorBoveda();
                gb_cursorAbiertoBov     =false;
        }

        // Obtengo el numero elementos en el catalogo
        li_totalBoveda = of_getBovedaCount();

        if(li_totalBoveda > 0){

                // Abro el cursor
                AbrirCursorBoveda();
                gb_cursorAbiertoBov     =true;

                // Inicializo la lista
                inicialista(TraspasosSeleccBovedaList, li_totalBoveda, (ListDrawDataFuncPtr)
BovedaDrawFunc, true);

                // Dibujo la lista
                LstDrawList((ListPtr) GetObjectPtr(TraspasosSeleccBovedaList));
        }
        else
                FrmCustomAlert(NoCatalogoAlert, "BOVEDA.", NULL, NULL);

}

/************************************************************************
 *
 * FUNCION:     TraspasoFormInit
 *
 * DESCRIPCION: Inicializa la forma de Cambio de Estado Fisico
 *
 * PARAMETROS:          Apuntador a la Forma.
 *
 * REGRESA:             nada.
 *
 ************************************************************************/

static void TraspasoFormInit()
{
    char lc_folio[6];

   gi_Tipo_Oper_Local = getTipoOperacion();
        gb_cursorAbiertoCodigos = false;

        // Inicializo el lector de codigo de barras
        AbreLector();


        // Muestro en pantalla el tipo de operacion y
        // la etiqueta correspondiente al catalogo de Boveda de que se trate...
    switch(gi_Tipo_Oper_Local) {
        case ALTA_TRASPASO:
                        SetFieldText( TraspasosTipoOperField,    "ALTA", 15, true);
                                SetFieldText( TraspasosSeleccBovedaField,"Bov. Destino          ",  21,
true);
                break;
        case RECEP_TRASPASO:
                        SetFieldText( TraspasosTipoOperField,    "RECEPCION", 15, true);
                                SetFieldText( TraspasosSeleccBovedaField,"Bov. Origen           ",  21,
true);
            break;
        case CANC_TRASPASO:
                        SetFieldText( TraspasosTipoOperField,    "CANCELACION", 15, true);
                                SetFieldText( TraspasosSeleccBovedaField,"Bov. Origen           ",  21,
true);
                break;
                default:
                                SetFieldText( TraspasosTipoOperField,    "NO DEFINIDA", 15, true);
                                SetFieldText( TraspasosSeleccBovedaField,"BOVEDA*****           ",  21,
true);
                        break;
   }
        gi_clave_boveda = 0;
        // Asigno un numero de folio, lo despliego  y actualizo el global y
        gi_Folio_Local = getUltimoFolio();
        StrItoA(lc_folio,gi_Folio_Local);
        SetFieldText( TraspasosFolioField, lc_folio, 6, true);

        if (I_UL_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local, gi_clave_boveda, getFecha(), getemp_ID())
!= 0 )
                FrmCustomAlert(ErroretiquetaAlert," no se inserto en UL_OPERACION", NULL, NULL);
```

```
}

/*************************************************************************
 *
 * FUNCION:      HandleReadTraspaso
 *
 * DESCRIPCION: La llamo cuando se realiza la lectura de un codigo de barras. LLevo
 *                              a cabo las operaciones de:
 *
 *                                      - Decodificar el codigo de barras
 *                                      - Abrir el cursor de etiquetas leidas
 *                                      - Mostrar la lista nueva
 * PARAMETROS:          Ninguno.
 *
 * REGRESA:             cero.
 *
 *************************************************************************/
void HandleReadTraspaso();
void HandleReadTraspaso()
{

        Int li_status;
        UInt li_estado_fisico;
        SByte li_resul_EnvaseInfo;

        // Decodifico el dato

        if ( DecodificaDato(TraspasosUEField, g_code128)) {
                li_resul_EnvaseInfo = of_getEnvaseInfo(g_code128, &li_status,  &li_estado_fisico);
                if (li_resul_EnvaseInfo == 0) {

                        switch (li_status) {
                                case ABIERTO:
                                case MISMO_STATUS_PADRE:
                                        FrmCustomAlert(ErroretiquetaAlert," no puede participar en un
traspaso", NULL, NULL);
                                        break;

                                case EN_TRANSITO:
                                        if (gi_Tipo_Oper_Local == RECEP_TRASPASO || gi_Tipo_Oper_Local ==
CANC_TRASPASO) {
                                                if (I_UL_D_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local,
g_code128) != 0 )
                                                        FrmCustomAlert(ErroretiquetaAlert," no se insertó
en UL_D_OPERACION", NULL, NULL);
                                        }
                                        else
                                                FrmCustomAlert(ErroretiquetaAlert," no se pudo guardar
porque esta en transito", NULL, NULL);
                                        break;

                                case DISPONIBLE_EN_CAJA:
                                        if (gi_Tipo_Oper_Local==RECEP_TRASPASO ||
gi_Tipo_Oper_Local==CANC_TRASPASO)
                                                FrmCustomAlert(ErroretiquetaAlert," no se pudo guardar
porque no esta en transito", NULL, NULL);
                                        else {
                                                if (I_UL_D_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local,
g_code128) != 0 )
                                                        FrmCustomAlert(ErroretiquetaAlert," no se inserto
en UL_D_OPERACION", NULL, NULL);
                                        }
                                        break;
                                }
                }

                if (li_resul_EnvaseInfo == -5) {
                        if (I_UL_ENVASE(g_code128,g_code128, DISPONIBLE_EN_CAJA, 0) != 0 )
                                FrmCustomAlert(ErroretiquetaAlert," no se inserto en UL_ENVASE", NULL,
NULL);
                        else {
                                if (I_UL_D_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local, g_code128) != 0
) {
                                        FrmCustomAlert(ErroretiquetaAlert," no se inserto en
UL_D_OPERACION", NULL, NULL);
                                        //Borra la etiqueta de UL_ENVASE por no insertarse en
UL_D_OPERACION
                                        if (D_UL_ENVASE(g_code128) != 0 )
                                                FrmCustomAlert(ErroretiquetaAlert," no se inserto en
UL_D_OPERACION y no se pudo eliminar de UL_ENVASE", NULL, NULL);
                                }
                        }
                }
        }
}
/*************************************************************************
 *
 * FUNCION:      Actualiza_Status
```

```
 *
 * DESCRIPCION:
 * PARAMETROS:
 *
 * REGRESA:
 *
 ************************************************************************/
 void Actualiza_Status(void);
 void Actualiza_Status(void)
{
        SByte li_total_x_oper;
        UInt li_codigo;
        char  lc_codigo[18];

        li_total_x_oper = of_get_Etq_Detalle_una_Ope(gi_Tipo_Oper_Local, gi_Folio_Local);

        if(!gb_cursorAbiertoCodigos) {
                AbrirCursorCodigos_una_Ope(gi_Tipo_Oper_Local, gi_Folio_Local);
                gb_cursorAbiertoCodigos = true;
        }

        for (li_codigo = 1; li_codigo <= li_total_x_oper; li_codigo++) {
                GetNextCodigo_una_Ope(li_codigo, lc_codigo);
                switch (gi_Tipo_Oper_Local) {
                case ALTA_TRASPASO:
                        if (U_UL_ENVASE_Status(lc_codigo, EN_TRANSITO ) != 0)
                                FrmCustomAlert(ERRORAlert,"No fue posible actualizar el STATUS de ",
lc_codigo, "");
                break;
                case RECEP_TRASPASO:
                case CANC_TRASPASO:
                        if (U_UL_ENVASE_Status(lc_codigo, DISPONIBLE_EN_CAJA ) != 0)
                                FrmCustomAlert(ERRORAlert,"No fue posible actualizar el STATUS de ",
lc_codigo, "");
                break;
                }
        }
        if(gb_cursorAbiertoCodigos) {
                CerrarCursorCodigos_una_Ope();
                gb_cursorAbiertoCodigos = false;
        }
}
/************************************************************************
 *
 * FUNCTION:    LecturaFormDoButton
 *
 * DESCRIPTION: Esta rutina lleva a cabo las acciones del boton correspondiente
 *
 * PARAMETERS:  buttonId  - id del boton
 *
 * RETURNED:    nothing
 *
 ************************************************************************/
static Boolean TraspasoFormDoButton(Word buttonId)
{
        Boolean handled = false;
        char lc_codigo[18];
        UInt li_boton;

        switch (buttonId)
                {
                        case TraspasosHechoButton:
                                if ((gi_clave_boveda == 0)
                                        &&      (of_get_Etq_Detalle_una_ope(gi_Tipo_Oper_Local,
gi_Folio_Local) != 0))  {
                                        FrmCustomAlert(INFORMACIONAlert,"No es posible registrar la
operacion sin haber seleccionado",
          " una Boveda","");
                                        return true;
                                }
                                if (gb_cursorAbiertoBov)
                                        CerrarCursorBoveda();

                                CierraLector();

                                if(gb_cursorAbiertoCodigos) {
                                        CerrarCursorCodigos_una_Ope();
                                        gb_cursorAbiertoCodigos = false;
                                }

                                if (ValidaInsersion_UL_OPERACION(gi_Tipo_Oper_Local,gi_Folio_Local))
                                        Actualiza_Status();

                                // Regreso a la pantalla principal
                                FrmGotoForm(MainForm);
                                handled = true;
                                break;
```

```
                        case TraspasosEliminarButton:
                                li_boton = DisplayEliminarContenedor(EliminaContenedorForm, lc_codigo);
                                if (li_boton == EliminaContenedorOKButton) {
                                        switch (gi_Tipo_Oper_Local) {
                                        case ALTA_TRASPASO:
                                                EliminarContenedor(gi_Tipo_Oper_Local, gi_Folio_Local,
lc_codigo, 0, false);
                                                break;
                                        case RECEP_TRASPASO:
                                        case CANC_TRASPASO:
                                                EliminarContenedor(gi_Tipo_Oper_Local, gi_Folio_Local,
lc_codigo, 0, false);
                                                break;
                                        }
                                        DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local,
TraspasosTotalUEField);
                                }
                                handled = true;
                                break;

                        case TraspasosConsultaButton:
                                ConsultarCodigos(TraspasosUnidadesEmpaqueList, gi_Tipo_Oper_Local,
gi_Folio_Local, &gb_CursorAbiertoCodigos);
                                handled = true;
                                break;
                        }
        return handled;
}

/*****************************************************************************
 *
 * FUNCTION:     TraspasoFormHandleEvent
 *
 * DESCRIPTION: This routine is the event handler for the
 *               "CambioEdoFisForm" of this application.
 *
 * PARAMETERS:   eventP  - a pointer to an EventType structure
 *
 * RETURNED:     true if the event has handle and should not be passed
 *               to a higher level handler.
 *
 *****************************************************************************/
Boolean TraspasoFormHandleEvent(EventPtr eventP);
Boolean TraspasoFormHandleEvent(EventPtr eventP)
{
        Boolean handled = false;
    FormPtr frmP;
        char lc_Boveda[21];

        switch (eventP->eType)
                {

                case ctlSelectEvent:
                        handled =TraspasoFormDoButton(eventP->data.ctlSelect.controlID);
                        break;

                case frmOpenEvent:
                        frmP = FrmGetActiveForm();
                        TraspasoFormInit();
                FrmDrawForm ( frmP);
                ConsultarBoveda();
                DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local, TraspasosTotalUEField);

                        handled = true;
                        break;

                case scanDecodeEvent:
                        HandleReadTraspaso();
                        DespliegaTotalxOpe(gi_Tipo_Oper_Local, gi_Folio_Local, TraspasosTotalUEField);

                        handled = true;
                        break;

                case popSelectEvent:
                        gi_clave_boveda = 0;
                        GetNextBoveda(eventP->data.popSelect.selection + 1, &gi_clave_boveda, lc_Boveda);
                        SetFieldText( TraspasosSeleccBovedaField, lc_Boveda, 21, true);
                        if (U_UL_OPERACION(gi_Tipo_Oper_Local, gi_Folio_Local, gi_clave_boveda) != 0 )
                                FrmCustomAlert(ErroretiquetaAlert," no se actualizo en UL_OPERACION",
NULL, NULL);
                        handled = true;
                        break;

                default:
                        break;
                }
        return handled;
}
```

REFERENCIAS ELECTRONCIAS

http://www.palm.com

http://www.palmos.com

http://www.sybase.com

http://www.ultralite.com

http://www.codewarrior.com

http://www.pumatech.com

http://www.caslsoft.com

BIBLIOGRAFIA

Erdei Guillermo E.
Código de Barras
McGraw Hill
3ª edición

Giguere Eric
Palm Database Programming: The Complete Developer's Guide
Wiley Computer Publishing
1999

Gillenson Mark L.
Introducción a las Bases de Datos
McGraw Hill

Luque Ruiz Irene, Gómez Nieto Miguel Angel
Diseño y Uso de Bases de Datos Relaciónales
Rama
Septiembre de 1997

Palmer Roger C.
The Bar Code Book (Reading and Printing Specification of Bar Code Symbols)
Helmers Publishing
1998

Rhodes Neil y McKeehan Julie
Palm programming, The Developer's Guide
O'Reilly
Primera edición 1999

Sybase manual

UltraLite Developer's Guide
Sybase Inc.
1999

Wiederhold Gio
Diseño de Bases de Datos
McGraw Hill

GLOSARIO

68K Assambler Es una herramienta para el desarrollo del software que permite el desarrollo de aplicaciones en ensamblador Motorola 68K.

ActiveX Son controles reutilizables que están compuestos por varios controles básicos, y tienen una funcionalidad determinada.

API Windows Application Programming Interface, y es la interfaz de programación de aplicaciones para Windows que permite que las aplicaciones Windows se comuniquen entre si, compartiendo funciones y utilizando mensajes entre ellas.

Archivos Include Son archivos de cabecera de un programa en lenguaje C, y son los encargados de proporcionar funcionalidad al programa que se esté desarrollando.

Área Graffiti Es una parte de la pantalla del dispositivo Palm, donde se introducen caracteres, números y caracteres especiales. Es una de los medios de introducir información al dispositivo.

ASCII American Standard Code for Information Interchange. Y significa que se trata de un código americano de normas utilizadas para el intercambio de información entre fabricantes. Esta norma define un valor decimal para cada uno de los diferentes caracteres

Buffer Zona de memoria que almacena datos temporalmente mientras dura la transferencia de estos.

Byte Es un grupo de ocho bits, donde un bit es el valor de cero o uno y es la unidad mínima de información. Con esta cantidad el ordenador puede representar números desde el 0 al 255.

C++ Es el resultado de varios años de experimentación y búsqueda en los laboratorios Bell de AT&T, para crear un sucesor de C incluyendo todas las características de éste y añadiendo nuevas facilidades especialmente importantes, como es la programación orientada a objetos.

Capacitor Sistema de dos conductores o armaduras, separados por un medio aislante, que acumula cargas eléctricas de signos opuestos.

CASL Es un paquete comercial que provee soporte para la plataforma Palm. Es un lenguaje de programación para dispositivos Palm que corre bajo Windows, pero que lo que se desarrolle

	en el puede ser leído en otras plataformas.
Código de Barras	Es la forma de representar información que contiene números u otros caracteres haciendo uso de una secuencia de barras paralelas, claras y obscuras, anchas y estrechas, las cuales son leídas por medio de equipos de lectura óptica.
Compilador	Programa que genera el código máquina a partir de un lenguaje de alto nivel como BASIC o C y que es escrito en algún editor.
Computadora de Escritorio	Sistema de proceso de microcomputadora orientado a un solo usuario y de aplicación general, que puede ejecutar instrucciones de programa para realizar una gran variedad de tareas.
Computadora Portátil	Es una computadora que realiza lo mismo que una computadora de escritorio, nada mas que es mas pequeña, de tal manera que puede transportarse fácilmente.
Conducto (conduit)	Es un programa en la computadora de escritorio que es llamado durante la sincronización para manejar el flujo de la información que va de la base de datos hacia el dispositivo y viceversa.
Conduit Development Kit	Es un conjunto de herramientas que se utiliza para el desarrollo de conductos.
Chip	Dispositivo semiconductor que incorpora un circuito integrado y se une al bus por unas patillas metálicas.
Debugger o Debug	Es una herramienta mediante la cual se puede saber se puede seguir línea por línea un programa.
Disco Duro	Es un dispositivo de almacenamiento permanente en una computadora. Está formado por un conjunto de discos rígidos, fabricado con un soporte de aluminio y recubierto de oxido magnético por ambas caras
DLL	Es una biblioteca dinámica (Dinamic Link Library) permite que las aplicaciones Windows compartan código y recursos. Es un archivo que contiene funciones de Windows que

pueden ser utilizadas por todas las aplicaciones.

Dragon Ball Es el nombre del procesador utilizado por los dispositivos Palm y es una versión de bajo consumo del procesador MC68000.

Emulador Es una herramienta que emula una Palm en una computadora de escritorio, y sirve para ver como se comportaría una aplicación, sin tener que instalarla físicamente en una Palm, ejecutando simplemente un programa en una computadora de escritorio.

Codificación Es una etapa posterior a la programación y consiste en describir, en el lenguaje de programación adecuado, la solución ya encontrada o sugerida, por medio de la programación. Es decir, primero se programa la solución de un problema y después hay que traducirla (codificarla) a la computadora.

Enlazador Es un programa encargado de enlazar los códigos objeto creados por una aplicación al momento de compilarla.

Estándar C Lenguaje de programación de alto nivel que se caracteriza porque también es considerado como lenguaje de bajo nivel. Y al referirse a Estándar C es para diferenciarlo con C++.

GNU Es una herramienta que permite crear aplicaciones para Palm, siendo la primera que permitió desarrollar aplicaciones bajo la plataforma Windows.

GNU PlamPilot SDK Es una colección de herramientas para el desarrollo de software y es lo que permite crear aplicaciones Palm OS en C/C++ para Unix o Windows.

Heap Es utilizado para asignación dinámica de bloques de memoria de tamaño variable. Muchas estructuras de datos. como árboles y listas lo utilizan como sitio de almacenamiento. Esta zona está bajo el control del programador. Muchas veces es conocido como Montón.

HotSync Programa encargado de la sincronización entre un dispositivo portátil y una computadora de escritorio.

Interfaz Es la parte de un sistema que se encarga de la comunicación de dicho sistema ya sea con otro sistema o con el usuario.

Item	Es un término que se utiliza para referirse a un elemento de entre un conjunto de elementos.
Java	Es un lenguaje de programación de alto nivel el cual es utilizado principalmente para desarrollar aplicaciones y applets, aunque en los últimos días están teniendo un gran auge los componentes, las páginas creadas con este lenguaje, las cuales tienen las extensión jsp. Este lenguaje esta basado en la independencia a la plataforma en donde se ejecute, necesitando de una máquina virtual para su compilación.
JDBC	Es la interfaz que permite el acceso externo a bases de datos.
Kernel	Es la parte mas importante de un sistema operativo, se dice que es el corazón de éste.
Kit	Esta palabra se refiere a un conjunto de herramientas.
Librería	Es un archivo que tiene cierta funcionalidad la cual es compartida entre aplicaciones de Windows.
Llave Primaria	Es el campo o campos que sirven para identificar a un registro como único dentro de una tabla.
Maquina Virtual	Es un software que emula a un procesador que puede interpretar instrucciones en lenguaje Java. La máquina virtual de Java es como una computadora con un lenguaje máquina especial que corre programas en Java.
MB	Abreviatura de MegaByte y que es cerca de un millón de bytes. El número exacto es 2^{20}=1048576 bytes.
Memoria Dinámica	Es el tipo de memoria RAM que una PC utiliza y significa que no puede retener datos por mas tiempo que unas milésimas de segundo. necesita recibir constantemente impulsos de reloj para tener "fresca" la memoria.
Memoria RAM	Random Access Memory o memoria de acceso aleatorio, denominada así pues se accede desde cualquier posición pudiéndose leer o escribir datos en ella. Esta memoria solo esta activa cuando esta encendida la computadora y pierde todo lo almacenado en ella cuando se

	apaga.
Memoria ROM	Memoria de solo lectura que contiene los programas de arranque de la computadora que realizan el trabajo de poner en marcha a ésta. También contiene una colección de rutinas en lenguaje máquina que proporcionan los servicios básicos de soporte de la computadora.
Menubar	Es un objeto utilizado por CodeWarrior para desarrollar una aplicación en un dispositivo Palm, y sirve como contenedor de menús.
Motorola MC68328	Es un término utilizado para referirse al procesador modelo MC68328 que fabrica Motorola.
Multipágina	Consiste en dividir un proceso en varios bloques de memoria llamados páginas, los cuales son de igual tamaño y pueden estar separados.
Multi-Thread	Es la opción de manejar varios threads, es decir, a descomponer varios procesos en partes y poder ejecutar cada parte sin tener que esperar a que acabe el proceso completo.
Outlook	Es una herramienta de Microsoft y que forma parte del paquete de Office, la cual sirve para tener un amplio control sobre correos, citas, notas, contactos, tareas, diario, etc.
Palm	Dispositivo portátil que tiene el uso de una agenda electrónica donde se pueden almacenar citas, números telefónicos, lista de tareas, notas, etc. Este dispositivo ha tenido un gran auge en los últimos días gracias a que es muy practico y tiene un tamaño pequeño. Utiliza un sistema operativo Palm OS que permite el desarrollo de aplicaciones personalizadas.
Palm Computing	Término que se refiere a la plataforma Palm.
Palm OS	Palm Operating System. Término utilizado para referirse al sistema Operativo Palm.
PC	Abreviatura del termino en inglés Personal Computer que significa Computadora Personal
Pila de Almacenamiento	Es un área muy importante manejada directamente por el CPU para alojar datos durante la ejecución del programa. Aquí se almacenan las variables locales automáticas y los datos involucrados en el mecanismo de invocación de funciones, de forma que si se utilizan muchas de estas invocaciones de forma anidada o recursiva, la pila crece.

Píxel Abreviatura o contracción de Picture element. Elemento de imagen de una pantalla que se emplea para trabajar con gráficas. Al encender o apagar cada píxel, la computadora puede trazar una imagen gráfica.

Plataforma Es un concepto utilizado para referirse a toda una familia de software. Se basa en un sistema operativo y además incluye muchas otras herramientas que son compatibles entre sí.

Plataforma Win 32 Se refiere a un ambiente donde se utiliza a Windows como sistema operativo, el cual maneja un bus de datos de 32 bits. Además, se utilizan herramientas que son desarrolladas por el mismo fabricante, y son compatibles con Windows.

Procesador El procesador es el encargado de ejecutar las instrucciones de un programa, escritas en lenguaje máquina. Es un dispositivo electrónico diseñado para efectuar cientos de miles o millones de operaciones en un segundo. Es el corazón de la computadora y controla todas las operaciones que ésta realiza.

Puerto Infrarrojo Es el puerto por medio del cual una Palm puede enviar o recibir datos de algún otro dispositivo que cuente con un puerto infrarrojo.

Puerto Serial Conjunto de líneas de entrada / salida que proporciona al microprocesador el acceso a un dispositivo externo (como la Palm) por el que se pueden suministrar o extraer datos, con la principal característica de que los bits de una palabra son transferidos uno a uno mediante una serie de impulsos eléctricos con dos estados, nivel bajo (0) y nivel alto (1).

Refresco (Refreshing) Es un término utilizado para referirse a la acción de dejar de mostrar la información antigua para mostrar la última información.

Reset Se refiere a una operación que se lleva a cabo en algún dispositivo como una computadora personal o una Palm y consiste en inicializar a dicho dispositivo borrando la memoria y dejándolo listo como si se acabara de prender después de adquirirlo. En una Palm existen diferentes tipos de reset y según sea el tipo hay que realizar diferentes pasos para realizarlo.

Scanner Es un dispositivo periférico que sirve para digitalizar una imagen y transferirla a una computadora.

Script	Es un conjunto de sentencias SQL que el servidor Mobilink invoca a la base de datos consolidada para realizar una operación.
Sincronización	Es la operación por medio de la cual se intercambian información un dispositivo portátil como la Palm y una computadora de escritorio.
Sistema Operativo	Es un conjunto ordenado de programas que controlan la operación general de una computadora. Hace posible que el equipo del sistema trabaje con los programas de aplicación del usuario.
Snapshot	Es un objeto que almacena una cantidad de registros que son el resultado de una consulta a una base de datos. Se puede pensar en un snapshot como una tabla. La característica principal de un snapshot es que es de solo lectura, o sea que no se puede hacer ninguna modificación a algún dato por medio de éste objeto.
Stylus	Objeto con el cual se toca la pantalla de una Palm y de esta manera se le puede introducir información.
Tap	Se le llama Tap al toque del Stylus (pluma) de una Palm en la pantalla. Este es el medio por el cual se introduce información al dispositivo Palm
Thread	Se refiere a descomponer la ejecución de un proceso en varias partes llamados threads o hilos.
Tiempo de Ejecución	Es un termino que se refiere a la fase en la que se está ejecutando una aplicación y principalmente se usa para diferenciar con la fase de diseño que es cuando se está desarrollando la aplicación.
Timestamp	Es un tipo de dato que almacena un dato de tiempo.
Trigger	Es un conjunto de sentencias, escritas en algún lenguaje entendible para un manejador de bases de datos, que se ejecutan cuando sucede un evento.
Unix	Sistema operativo desarrollado en los laboratorios Bell, en New Yersey, E.U. Sistema

operativo de tiempo compartido, pequeño, de propósito general y multiusuario. Esta escrito en lenguaje C y cuenta con una gran cantidad de ayudas y utilerías para la programación. Con este sistema operativo es sencillo lograr comunicación y sincronización entre procesos.

Visual Basic Lenguaje de programación de alto nivel, el cual tiene una interfaz gráfica al estilo Windows.

Windows 95/98/NT Es un sistema operativo desarrollado por Microsoft, el cual es el pionero en la utilización de un bus de datos de 32 bits.