UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

"DESARROLLO DE UNA APLICACION EN POWER BUILDER"

T E S I S

QUE PARA OBTENER EL TITULO DE:

M A T E M A T I C O

P R E S E N T A:

CARLOS GARCIA CURIEL

DIRECTORA DE TESIS:
M. EN C. MARIA GUADALUPE ELENA IBARGUENGOITIA GONZALEZ

2005

VNIVERSIDAD NACIONAL
AVTNOMA DE
MEXICO

ACT. MAURICIO AGUILAR GONZÁLEZ
Jefe de la División de Estudios Profesionales de la
Facultad de Ciencias
Presente

Comunicamos a usted que hemos revisado el trabajo escrito:

"Desarrollo de una aplicación en power builder"

realizado por Carlos García Curiel

con número de cuenta 08235165-8 , quien cubrió los créditos de la carrera de: Matemáticas

Dicho trabajo cuenta con nuestro voto aprobatorio.

A t e n t a m e n t e

Director de Tesis M. en C. María Guadalupe Elena Ibarguengoitia González
Propietario

Propietario Dra. Amparo López Gaona

Propietario Mat. Facundo Ruiz Doncel

Suplente M. en C. Gustavo Arellano Sandoval

Suplente M. en C. Gustavo Arturo Márquez Flores

Consejo Departamental de Matemáticas

M. en C. Alejandro Bravo Mojica

DEDICATORIAS

A la memoria de mi padre ***Rosalino***, por su ejemplo de respeto y lucha por la vida. A mi madre ***Luz*** por su amor y cariño de siempre.

A mi amada esposa ***Claudia*** y a mis queridos hijos ***Karla e Iván***. Porque su amor incondicional me fortalece en todos los sentidos.

A mis hermanos: ***Arturo, Ismael, Gloria, Hortencia, Rosa María, Raúl, Jorge y Francisco***. Por la confianza y respaldo que me han dado todo el tiempo.

AGRADECIMIENTOS

A la Maestra en Ciencias ***María Guadalupe Elena Ibargüengoitia González*** por compartir sus conocimientos, su tiempo y dedicación que me brindo para la realización de este trabajo.

Al Matemático ***Facundo Ruiz Doncel***, a la ***Doctora Amparo López Gaona***, al Maestro en Ciencias ***Gustavo Arellano Sandoval*** y al Maestro en Ciencias ***Gustavo Arturo Márquez Flores*** por sus comentarios y sugerencias que enriquecieron este trabajo.

INDICE

INTRODUCCIÓN

La tecnología de la orientación a objetos ha influenciado a una amplia gama de elementos para el desarrollo de software incluyendo lenguajes, interfaces de usuario, bases de datos y sistemas operativos. Aunque la orientación a objetos no es la panacea, ya ha demostrado que puede ayudar a gestionar la creciente complejidad y costos del desarrollo de software.

El propósito de esta tesis es mostrar las ventajas que tiene esta tecnología en el desarrollo de sistemas a través de un **"Marco de Referencia"** (que es una colección de clases que proporciona un conjunto de servicios para un dominio particular) el cual permite el desarrollo de sistemas como la generación rápida de aplicaciones, la integración de un equipo de programadores y una programación consistente y estandarizada.

Para comprender lo anterior, en el primer capítulo se dará una introducción de los conceptos generales de la Tecnología de la Orientación a Objetos y de las Bases de Datos Relacionales; en el segundo capítulo, se explica el entorno de programación de Power Builder y la plataforma .NET. En el tercer capítulo se define qué es un marco de referencia y sus ventajas, así como la implementación de un marco de referencia para Power Builder llamado infraestructura; en el cuarto capítulo, se explica cómo crear una aplicación con la Infraestructura de Power Builder y finalmente, en el quinto capítulo se desarrolla un sistema específico implementando la Infraestructura de Power Builder.

Posteriormente se incluyen las conclusiones de este trabajo.

I. CONCEPTOS GENERALES

I.1 Conceptos de la Programación Orientada a Objetos

"La programación orientada a objetos es un método de implementación, en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales, representa una instancia de alguna(s) clase(s), y cuyas clases son, todas ellas, miembros de una jerarquía de clase unidas mediante relaciones de herencia" (Booch, 1994).

"Hay tres partes importantes en esta definición, la programación orientada a objetos (1) utiliza objetos, no algoritmos, como sus bloques lógicos de construcción fundamentales; (2) cada objeto es una instancia de alguna clase; y (3) las clases están relacionadas con otras clases por medio de relaciones de herencia" (Booch, 1994).

I.1.1 Modelo de Objetos

En la programación orientada a objetos existe un marco de referencia conceptual llamado modelo de objetos, este modelo describe la estructura de los objetos en un sistema y consta de cuatro elementos fundamentales:

- Abstracción
- Encapsulamiento
- Modularidad
- Jerarquía

I.1.1.1 Abstracción

"Una abstracción denota las características esenciales de un objeto que lo distingue de todos los demás tipos de objeto y proporciona así fronteras conceptuales definidas respecto a la perspectiva del observado" (Booch, 1994).

"Una abstracción se centra en la visión externa de un objeto, y por lo tanto, sirve para separar el comportamiento esencial de un objeto de su implementación" (Booch, 1994).

I.1.1.2 Encapsulamiento

"El encapsulamiento es el proceso de almacenar en un mismo compartimiento los elementos de una abstracción que constituyen su estructura y su comportamiento, sirve para separar la interfaz contractual de una abstracción y su implementación" (Booch, 1994).

El encapsulamiento se puede definir como la forma de ocultar la información, es una técnica en la cual los datos son almacenados con sus correspondientes métodos. Esto significa que los datos internos de un objeto sólo pueden ser accedidos a través de los mensajes de interfaz de tal objeto.

I.1.1.3 Modularidad

La modularidad es la propiedad que tiene un sistema que ha sido descompuesto en varios módulos independientes pero que tienen conexiones con otros módulos, con la ventaja de que cada módulo es menos complejo que la totalidad del sistema.

I.1.1.4 Jerarquía

"La jerarquía es una clasificación u ordenación de abstracciones" (Booch, 1994).

I.1.2 Herencia

La herencia es la jerarquía de clases "es un elemento esencial en los sistemas orientados a objetos. Básicamente define una relación entre clases, en la que una clase comparte la estructura de comportamiento definida en una o más clases (lo que se denomina herencia simple o herencia múltiple). La herencia representa así una jerarquía de abstracciones, en la que una subclase hereda una o más subclases. Típicamente una subclase aumenta o redefine la estructura o comportamiento de su superclase" (Booch, 1994).

La herencia es un mecanismo de compartir código. Esto permite reciclar el comportamiento de una clase para la definición de nuevas clases.

I.1.3 Objetos

Un objeto es una entidad lógica que contiene datos (atributos) y un código especial (métodos) que indica como manipular los datos.

"Un objeto tiene estado, comportamiento e identidad, la estructura y comportamiento de objetos similares están definidos en su clase común, los términos de instancia y objeto son intercambiables" (Booch, 1994).

"En todos los sistemas orientados a objetos, los objetos son diferenciados y reconocidos gracias a su identificador (nombre único). Este hecho permite su uso

como valores de atributos de otros objetos, ofreciendo gran riqueza en la construcción de estructura de datos de gran complejidad" (Tkach, 1994).

I.1.4 Clases

"Una clase es un conjunto de objetos que comparten una estructura común y un comportamiento común" (Booch, 1994).

La diferencia entre un objeto y clase es que un objeto "es una instancia de una clase que desempeña algún papel en el sistema global y la clase captura la estructura y comportamiento común de todos los objetos relacionados" (Booch, 1994).

I.1.5 Polimorfismo

La característica de polimorfismo de la programación orientada a objetos permite que las instancias de diferentes clases reaccionen de una forma particular al llamado de una función. Por ejemplo, en una jerarquía de formas gráficas (punto, línea, cuadro, rectángulo, etc.), cada forma tiene una función llamada *"Draw"* responsable de contestar adecuadamente a peticiones del trazo de esa forma.

El polimorfismo permite que las instancias de diferentes clases respondan a la misma función en forma adecuada para cada clase.

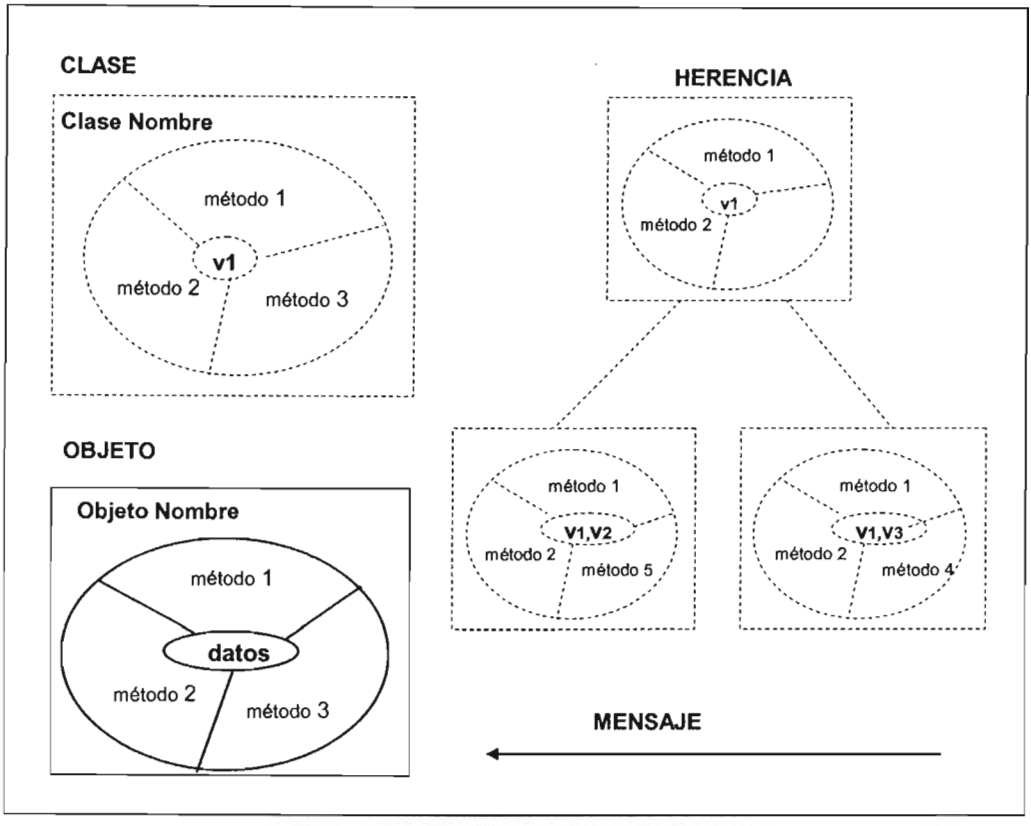


Figura 1.1 Representa los conceptos de la orientación a objetos; las clases se indican con líneas punteadas con sus respectivos métodos y atributos encapsulados; el objeto está representado con líneas continuas y contiene sus valores referenciados.

I.1.6 Beneficios de la Tecnología Orientada a Objetos

La Tecnología Orientada a Objetos (TOO) proporciona los siguientes beneficios, algunos son importantes para el usuario final y otros para el desarrollador:

- Un objeto es una representación de un pensamiento en el mundo real, y por lo tanto, puede ser diseñado para ser manipulado en una forma similar a la manipulación de objetos en el mundo real, y por ende, las aplicaciones se hacen amigables y fáciles de usar. La TOO permite a los usuarios

comunicar sus requerimientos fácilmente y a los programadores estructurar sus programas en una forma más natural (Tkach, 1994).

- La TOO tiene un potencial significativo en la reducción del mantenimiento de las aplicaciones, porque los módulos en una aplicación pueden ser construidos de módulos ya existentes a través de la herencia, y por lo tanto, es menor el código requerido. Además se reducen los errores de programación, al encapsular los datos que son modulares y facilitan el modelado y la reutilización del código (Tkach, 1994).
- Conceptualmente la TOO puede ofrecer uniformidad a través de todo el desarrollo del ciclo de vida de la aplicación. Además de contribuir significativamente en la explotación de la información dentro de una empresa (Tkach, 1994).

I.1.7 Ambientes de Desarrollo de la Orientación a Objetos

La Orientación a Objetos (OO) está influenciando a una amplia gama de componentes de software incluyendo lenguajes, interfaz de usuario, bases de datos y servidores de aplicación. La OO ha demostrado que puede ayudar a gestionar la creciente complejidad y costos del desarrollo de software.

La OO cambia la forma de trabajar de los programadores y aumenta la velocidad de producción de la próxima generación de aplicaciones. También puede aumentar la capacidad de programación del usuario final, de acuerdo a lo señalado anteriormente, porque se construyen los módulos existentes a través de la herencia, reduciendo los errores de programación, facilitando el modelado y reutilización del código; además incrementa la funcionalidad que puede incorporarse a las aplicaciones.

Los estándares de la OO se han cristalizado en ampliaciones de objetos en los lenguajes de programación populares, como por ejemplo, Java, C++, entre otros.

Las herramientas de desarrollo OO facilitan la codificación, depuración y el diseño. También se encuentran en uso en entornos de desarrollo para la construcción de interfaz de usuario y conexiones en bases de datos, por ejemplo, Power Builder y la plataforma .NET. Estos entornos de desarrollo incorporan una interfaz gráfica permitiendo la creación más rápida de aplicaciones interactivas a partir de bibliotecas de objetos ya existentes.

El éxito de la OO depende de la convergencia e integración de los sistemas de software, lenguajes, herramientas de bases de datos y bibliotecas de objetos prefabricadas (bibliotecas de clases). Para apoyar la creación y realización de sistemas completos con arquitectura de objetos.

I.2 Base de Datos Relacional

Una base de datos relacional consiste en una colección de tablas que contienen varios tipos de información, incluyendo datos de usuarios e información del mantenimiento del sistema, necesarios para administrar y manipular los datos.

Los elementos de una base de datos son definidos en la base de datos y son propios de ella.

I.2.1 Elementos de una Base de Datos Relacional

Los elementos básicos de una base de datos son:

- **Tabla**: define una colección de renglones con columnas asociadas.

- **Tipos de datos**: define los valores de datos permitidos en una columna o variable.
- **Restricciones**: define las reglas para validar los valores permitidos en las columnas, es un mecanismo estándar para forzar la integridad de los datos.
- **Omisiones**: define los valores que por omisión se asignan a una columna cuando no son proporcionados.
- **Regla**: contiene la información que define los valores validos y que son almacenados en una columna o tipo de dato.
- **Índice**: es una estructura de almacenamiento que proporciona accesos rápidos para recuperar datos y puede forzar la integridad de los datos.
- **Vistas**: proporciona una estructura para poder englobar datos de una o más tablas, o vistas, en una base de datos.
- **Procedimientos almacenados**: es una colección de sentencias de SQL precompiladas dentro de la base de datos.
- **Lanzadores**: es una forma especial de un procedimiento almacenado, se ejecuta automáticamente cuando un usuario modifica un dato en una tabla o vista.

I.2.2 Características Principales de un Sistema Manejador de Base de Datos (SMBD)

- Persistencia.
- Administración del Disco.
- Integridad de Datos.
- Concurrencia en el Acceso.
- Lenguaje de Consulta.

I.2.2.1 Persistencia

En un lenguaje de programación, todos los datos manipulados desaparecen al término de la ejecución del programa que los creó y/o manipuló. Por el contrario, en un SMBD todos los datos son creados para persistir y poder ser consultados y/o modificados posteriormente por las aplicaciones. La persistencia es entonces la primera característica que ofrece un SMBD.

I.2.2.2 Administración del Disco

Los avances tecnológicos alcanzados en los dispositivos de almacenamiento permiten almacenar cada vez mayores volúmenes de datos a menor costo. Los SMBD son capaces de asegurar eficiencia garantizando rapidez y flexibilidad en el acceso de los datos almacenados.

I.2.2.3 Integridad de Datos

La integridad se define como la propiedad que asegura la calidad del dato, respetando las propiedades o reglas especificadas en su definición. La integridad global de las bases de datos incluye aspectos como: integridad semántica, seguridad en el acceso (confidencialidad y protección de los derechos del usuario) y seguridad en el funcionamiento (recuperación en el caso de caída).

I.2.2.4 Concurrencia en el Acceso

Una de las principales ventajas que ofrece la administración de datos por un SMBD es la posibilidad de poner a disposición de varios usuarios, de manera

simultánea, la misma fuente de información, lo que requiere del establecimiento de protocolos de acceso.

I.2.2.5 Lenguaje de Consulta

La principal operación solicitada por los usuarios y sus aplicaciones es la consulta de la información previamente almacenada. Tomando en consideración el gran éxito logrado por lenguajes de consulta como SQL.

I.2.3 Lenguaje de Consulta Estructurado (SQL)

El SQL es un lenguaje declarativo que se utiliza para definir y manipular los elementos dentro de la base de datos.

I.2.4 Lenguaje de Definición de Datos (LDD)

El LDD permite crear, alterar y eliminar elementos de una base de datos, tales como tablas; controla la estructura de la base de datos y de las cuentas de los usuarios, algunas sentencias que incluye son:

CREATE: Crea nuevos elementos.
ALTER: Modifica elementos.
DROP: Remueve elementos.

I.2.5 Lenguaje de Control de Datos (LCD)

El propósito del LCD es crear un sistema de seguridad para el acceso y manipulación de datos dentro de la base de datos. Las sentencias que se detallan a continuación, son usadas para cambiar los permisos asociados con los roles o usuarios de las bases de datos:

GRANT: Crea un sistema de seguridad en el sistema permitiendo a los usuarios trabajar con datos y ejecutar ciertas sentencias.

REVOQUE: Remueve los permisos programados por el comando GRANT.

I.2.6 Lenguaje de Manipulación de Datos (LMD)

La habilidad para cambiar los valores en una base de datos, agregar nuevos registros y borrar registros es proporcionado por los comandos del LMD, éstos son: INSERT, UPDATE y DELETE.

Las sentencias del LMD regresan un número que indica cuántos registros fueron afectados por la manipulación de la sentencia o un código de error.

INSERT: Almacena nuevos registros en las tablas, los valores insertados deben ser del mismo tipo que fue definido en los campos de las tablas, la sintaxis es:

```
INSERT INTO Nombre de la Tabla
VALUES (col1, col2, clo3... )
```

Sí existe un error en el tipo de dato de la columna debe regresar un mensaje de error. Otra sintaxis es:

INSERT INTO Nombre de la Tabla (nombre columna 1, nombre columna 2...)
VALUES (col1, col2, ...)

DELETE: Elimina registros de una tabla. La cláusula WHERE se utiliza para especificar la condición de borrado, tal como se muestra a continuación:

DELETE FROM Nombre de la Tabla
WHERE Condición

Sí la condición no es proporcionada todos los registros de la tabla son eliminados.

UPDATE: Cambia los valores dentro de un registro. La cláusula SET especifica qué columnas van a cambiar de valor y, opcionalmente, la cláusula WHERE puede especificar cuáles registros son los afectados.

UPDATE Nombre de la Tabla
SET Nombre de la Columna = Nuevo Valor
WHERE Condición

II. ENTORNOS DE DESARROLLO

II.1 El Entorno de Desarrollo de Power Builder

Power Builder (PB) es una herramienta de desarrollo para la construcción de aplicaciones cliente/servidor, se pueden crear potentes aplicaciones gráficas que accesan al servidor de bases de datos. Una de las principales características de este entorno es la integración de todo lo necesario para poder realizar el desarrollo de aplicaciones complejas en un único archivo ejecutable. Sus diferentes facetas de desarrollo son: el desarrollo de clases, la administración de bibliotecas de clases, el control de versiones, el acceso y la administración de las bases de datos.

Power Builder proporciona un soporte completo de la tecnología orientada a objetos como es la herencia, encapsulación y polimorfismo.

La herencia en Power Builder permite crear un objeto nuevo que será miembro de una determinada clase. Se puede utilizar un *painter* para crear una nueva clase, la cual hereda todos los atributos de la clase padre. Se pueden incluir en la clase métodos y atributos nuevos y utilizar los procedimientos tal como se han heredado o ampliándolos o rescribiéndolos.

La encapsulación en Power Builder independiza y protege los datos. Un objeto encapsula sus datos y sus métodos. Por ejemplo, el tamaño de una ventana es un atributo encapsulado de una ventana. Para conocer el tamaño de la ventana se debe enviar un mensaje a la ventana que pida información sobre su tamaño, y solamente el objeto puede modificar sus datos encapsulados. Otros objetos pueden enviar mensajes a un determinado objeto solicitándole cambios, pero sólo el propio objeto puede realizarlos.

El polimorfismo en Power Builder significa que objetos pertenecientes a distintas clases pueden aceptar el mismo mensaje. Se puede enviar el mensaje *"print"* a diferentes objetos. El mismo mensaje provoca que cada uno de los objetos imprima algo.

II.1.1 Herramientas de Power Builder

En Power Builder (PB) cada desarrollo de un elemento diferente se hace en un espacio apropiado para ello llamado *painter*. Cada *painter* es una herramienta con un propósito general.

Al abrir un *painter* se abre una nueva ventana. La barra de título de la ventana indica el tipo de *painter* que esta abriendo. Se puede abrir más de un *painter* al mismo tiempo o tener una copia de este.

La siguiente tabla muestra los diferentes *painters* por defecto en el entorno de PB.

PAINTER	DESCRIPCIÓN
Aplicación	Generación de clases aplicación.
Proyecto	Generación de ejecutables de una aplicación en PB.
Ventana	Clases interfaz de una aplicación.
Objeto de Usuario	Desarrollo de componentes nativos.
Menú	Menús de una aplicación.
Estructura	Elementos de un tipo de estructura usados en una aplicación.
Función Global	Desarrollo de funciones de acceso global en una aplicación.
DataWindow	Clases de acceso y manipulación de datos dentro de una aplicación.
Clase Query	Clases de almacenamiento de preguntas preferidas a la base de datos.
Tuberías de Datos	Clases de reproducción, definición y datos de una base de datos a otra.
Configuración de Fuentes ODBC	Definición de la infraestructura necesaria para acceso a las bases de datos ODBC.

PAINTER	DESCRIPCIÓN
Perfiles de BD	Definición de la información necesaria para el acceso a diferentes tipos de fuentes de datos ODBC o nativos.
Bases de Datos	Entorno de acceso y modificación de la base de datos a la que se conecta el entorno.
Visualizador de clases	Elemento auxiliar que permite la visualización de las clases del sistema y de usuario con todos sus atributos, eventos y métodos.
Administrador de Bibliotecas	Entorno de administración del almacenamiento de las clases desarrolladas e interfaz para el acceso de herramientas externas de control de versiones.
Edición	Editor de texto incorporado.

II.1.2 Aplicaciones en Power Builder

Una aplicación en Power Builder es un conjunto de objetos tales como ventanas, Data Window, Menús y Controles. Cuando los objetos se están utilizando en una aplicación en ejecución, tienen una funcionalidad, la cual se les llama "eventos" a los que responden, como pueden ser la captura de pedido o el control de inventario. Los objetos de una aplicación se encuentran en una o varias bibliotecas de Power Builder.

La base de cualquier aplicación es el objeto de aplicación. El *painter* "aplicación" permite crear y modificar objetos aplicación. El objeto aplicación contienen el punto de comienzo de la aplicación. Cuando la aplicación se ejecuta, el objeto "aplicación" se carga en memoria y empieza a ejecutarse desde el punto de inicio.

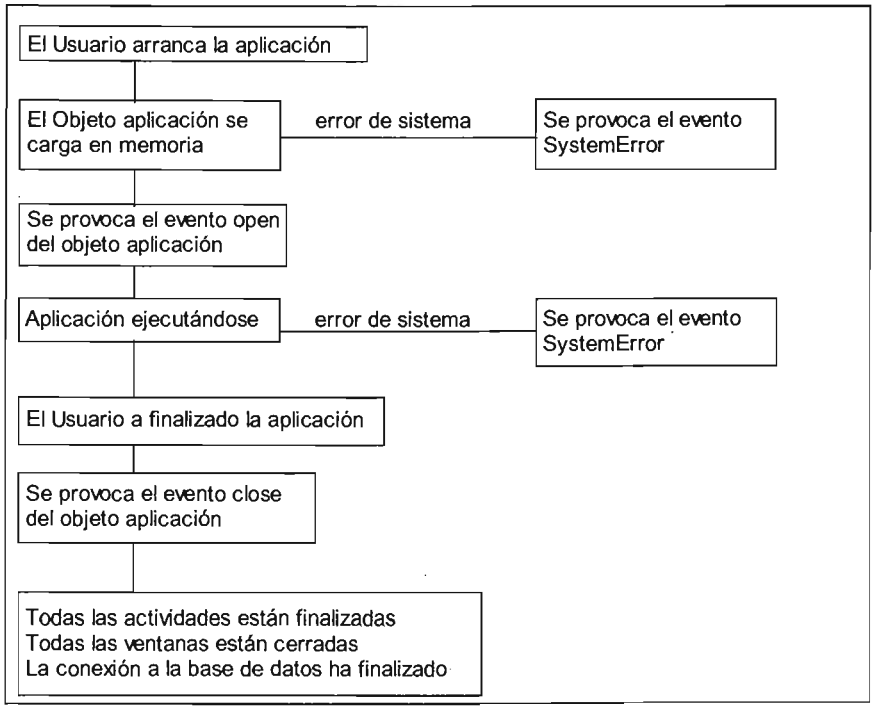


Figura 2.1 Este diagrama muestra la secuencia de eventos que tiene lugar cuando se ejecuta una aplicación.

Cada objeto aplicación tiene los siguientes atributos asociados:

- El nombre
- El icono
- La ruta de acceso a la aplicación
- El tipo de letra por defecto
- Variables globales
- Funciones externas globales

Cada objeto aplicación tiene valores asociados con cada uno de esos atributos. El *painter* "aplicación" permite modificar dichos valores.

También existen ciertas funcionalidades asociadas con un objeto aplicación. Estas son:

EVENTOS	DESCRIPCIÓN
Open	Sucede en el momento de apertura de la aplicación.
Close	Momento de cierre de la aplicación.
Idle	Evento provocado por la función idle que monitoriza la actividad de la aplicación en ejecución.
SystemError	El evento que se provoca cada vez que sucede un error de ejecución del aplicación.

El evento Open se provoca cuando el usuario arranca la aplicación. El procedimiento asociado con el evento Open inicia toda la actividad que tendrá lugar a continuación en la aplicación.

La aplicación finaliza cuando el evento Close tiene lugar. El evento SystemError se provoca si se produce un error del sistema al arrancar la aplicación o durante la ejecución. Una llamada a la función Idle fija el tiempo que la aplicación espera hasta que se provoca el evento Idle.

Cuando se crea un objeto aplicación y este se ha salvado en una biblioteca, se pueden crear otros objetos. Con el *Window Painter* se podría crear un objeto data window y con él se seleccionan los datos y se establece un formato en el cual se quiere representar dichos datos. Sin embargo, no se puede utilizar el data window de manera independientemente, sino que debe estar ligado a un control de la ventana.

Además con el *Window Painter* se puede añadir otros controles a la ventana de la aplicación. Estos controles también son objetos (botones, etiquetas, etc.)

Al archivar la nueva ventana se incluye un nuevo objeto en la biblioteca tal como se muestra en la siguiente figura:

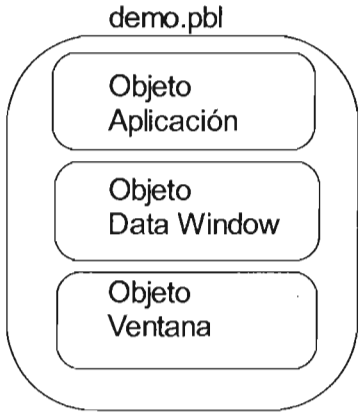


Figura 2.2 Construcción de una aplicación en Power Builder.

Así una aplicación sencilla se construye creando (1) una aplicación, (2) una ventana principal, (3) controles que administran la aplicación y (4) un control Data Window que presenta información.

II.2 Programación Orientada a Objetos (POO)

La POO tiene dos elementos fundamentales: *CLASES* y *OBJETOS* e incorpora elementos que facilita la realización de interfaz gráfica de usuario. Power Builder tiene herramientas gráficas que facilitan la realización de programas orientados a objetos, estas herramientas son los *painters* mencionadas anteriormente.

Con los lenguajes de programación tradicionales se escriben programas formados por módulos que mandan a llamar en una determinada secuencia, establecida

normalmente por el programa, los datos van de acá para allá entre rutinas del programa. El modelo de programación difiere en un entorno orientado a objetos, en el que se crean objetos que responden a eventos y se comunican entre sí por medio de mensajes.

II.3 Modelo Cliente Servidor

Un modelo cliente servidor tiene tres componentes: un servidor, una red y clientes.

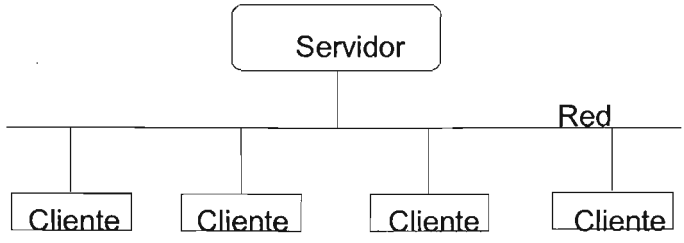


Figura 2.3 Los tres componentes de un entorno Cliente-Servidor

Un servidor es una computadora que almacena y administra datos, es responsable de suministrar cualquier dato requerido por el usuario. El servidor obtiene los datos en función de la petición realizada y los envía al cliente, es responsable del control de la concurrencia, del mantenimiento de la integridad de la base de datos y de la administración de las transacciones, además se responsabiliza de la seguridad de los datos y el control de acceso. Finalmente, también suministra el espacio en los dispositivos de almacenamiento para los datos.

Una computadora cliente solicita datos del servidor y ejecuta un programa localmente, preparado para solicitar adecuadamente los datos al servidor.

Asimismo, es responsable de la administración de la presentación de datos, de la interacción con el usuario y de la operativa de recuperación de datos. Además valida los datos introducidos por el usuario y genera las solicitudes necesarias al servidor.

PB tiene las herramientas necesarias para construir aplicaciones cliente/servidor que se ejecutan en una computadora cliente comunicándose entre sí utilizando protocolos de red estándar. Las aplicaciones pueden acceder a datos locales o remotos provenientes de una serie de fuentes.

Los servidores pueden ser computadoras personales como: Novell y Microsoft SQL; estaciones de trabajo o mini-computadoras con Oracle, Informix o Sybase; o Mainframes con DB y se pueden ejecutar la mayor parte de los sistemas de bases de datos importantes.

Un programa en PB que se ejecuta en una máquina cliente puede presentar un determinado formato de pantalla. El usuario puede darlos por medio de ese formato y el programa los utiliza para hacer una solicitud de datos al servidor.

II.4 Desarrollo en Web

La década de los 80's fue marcada por el surgimiento de la PC y de la interfaz gráfica. En la década de los 90's Internet permitió conectar computadoras en una escala global. En un principio la conexión fue entre PCs y servidores por medio del navegador de Internet.

A comienzos de este siglo es clara la necesidad de permitir a las computadoras conectadas a Internet comunicarse entre ellas. Esto ha generado la forma de un nuevo modelo de computación distribuida llamado Servicios Web basados en XML. El objetivo es permitir comunicarse entre sí a sistemas heterogéneos dentro

y fuera de la empresa. Esta comunicación es independiente del sistema operativo, lenguaje o modelo de programación.

Para conseguir esto se desarrollaron estándares. El consorcio de Internet http://www.w3c.org es el encargado de crear y mantener estos estándares.

Estos son algunos de los estándares que permiten hacer uso de los Servicios Web basados en XML:

- **XML (Lenguaje de Marcado eXtensible)**: Es un formato universal para representar los datos.
- **SOAP (Protocolo Simple de Acceso a Objetos)**: Es un protocolo que permite mover los datos entre aplicaciones y sistemas. Es el mecanismo por medio del cual los Servicios Web son invocados e interactúan.
- **UDDI (Descubrimiento, Descripción e Integración Universal)**: Lenguaje que permite publicar, encontrar y usar los Servicios Web basados en XML. Es la 'Página Amarilla' de los Servicios Web, es decir, un directorio para poder encontrarlos. Puede ser accedido con un explorador en http://www.uddi.org o programáticamente ya que UDDI es también un Servicio Web.
- **WSDL (Lenguaje de Descripción de Servicios Web)**: Lenguaje por medio del cual un Servicio Web describe, entre otras cosas, qué hace o qué funcionalidad implementa.

Los Servicios Web son similares a los componentes que están representados como cajas negras, donde los programadores pueden utilizarlos para agregar

nuevas funcionalidades. Estos servicios están diseñados para interactuar directamente con otras aplicaciones basadas en Internet proporcionado una interfaz estándar.

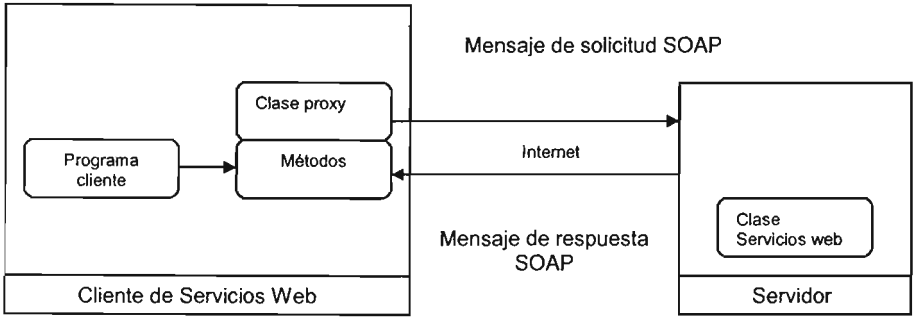


Figura 2.4 Comunicación de un servicio web

II.4.1 Entorno de Desarrollo de la Plataforma .NET

La plataforma .NET de Microsoft provee los cimientos para la nueva generación de software. Utiliza los Servicios Web como un medio para poder interoperar las diferentes tecnologías. Permite conectar distintos sistemas operativos, dispositivos físicos, información y usuarios. Proporciona a los desarrolladores las herramientas y tecnologías para hacer rápidamente soluciones de negocios que involucran distintas aplicaciones, dispositivos físicos y organizaciones.

II.4.1.1 El .NET Framework

Es un conjunto de servicios de programación diseñados para simplificar el desarrollo de aplicaciones en un entorno altamente distribuido de Internet.

Los componentes del .NET Framework proveen los "ladrillos" necesarios para construir las Aplicaciones Web, los Servicios Web y cualquier otra aplicación.

II.4.1.2 Componentes del .NET Framework

El Common Language Runtime (CLR) es un motor que ejecuta todos los lenguajes .NET, proporciona las bibliotecas utilizadas por el lenguaje y provee lo que se llama código administrado, es decir, un entorno que proporciona servicios automáticos al código que se ejecuta. Los servicios son variados:

- **Cargador de Clases**: Permite cargar en memoria las clases.
- **Compilador MSIL a Nativo**: Transforma código intermedio de alto nivel independiente del hardware que lo ejecuta a código de máquina propio del dispositivo.
- **Administrador de Código**: Coordina toda la operación de los distintos subsistemas del Common Language Runtime.
- **Recolector de Basura**: Elimina de memoria objetos no utilizados.
- **Motor de Seguridad**: Administra la seguridad del código que se ejecuta.
- **Motor de Depuración**: Permite hacer un seguimiento de la ejecución del código aún cuando se utilicen lenguajes distintos.
- **Verificador de Tipos**: Controla que las variables de la aplicación que se usan en el área de memoria estén asignadas.
- **Administrador de Excepciones**: Maneja los errores que se producen durante la ejecución del código.
- **Soporte de Multiproceso (threads)**: Permite ejecutar código en forma paralela.

- **Empaquetador de COM**: Coordina la comunicación con los componentes COM para que puedan ser usados por el .NET Framework.
- **Soporte de la Biblioteca de Clases Base**: Interfaz con las clases base del .NET Framework.

La biblioteca de clases base son las clases sobre las cuales se construyen todas las demás clases que utilizan los programas .NET. La clase madre de todas es System. A partir de ella, por un mecanismo llamado herencia de clases, se construyen las demás clases.

Debido a que en la biblioteca de clases base hay muchas clases, para identificarlas se utiliza un mecanismo llamado espacio de nombres (namespace). La parte del nombre de la clase que se encuentra a la derecha del último punto se llama tipo de la clase. Todo lo que resta se llama espacio de nombres. Por ejemplo, en la clase llamada System.Runtime.InteropServices, InteropServices es el tipo de la clase y System.Runtime es el espacio de nombre. El espacio de nombre es una manera de organizar en grupos las distintas clases. Esto las hace más manejables y fáciles de usar.

La biblioteca de clases base es independiente del lenguaje. Permite el uso y la depuración de otros lenguajes, es extensible, ya que por el mecanismo de herencia, el usuario puede crear nuevas clases que usan las clases base como "ladrillos", se pueden incorporar en bibliotecas para su utilización posterior y tiene distintos mecanismos de seguridad para permitir o restringir su uso.

III. MARCO DE REFERENCIA

Un beneficio importante de los lenguajes de programación orientada a objetos es el grado de reutilización que puede conseguirse en los sistemas bien diseñados. Un alto grado de reutilización significa que hay que escribir menos código para cada nueva aplicación; consecuentemente, hay menor el código que se debe mantener y probar (Booch, 1994).

La reutilización de software puede adoptar muchas formas: se pueden reutilizar líneas individuales de código, clases específicas o conjuntos de clases relacionadas lógicamente.

Se puede conseguir un mayor aprovechamiento reusando conjuntos enteros de clases organizadas en un marco de referencia.

Un marco de referencia es una colección de clases que proporciona un conjunto de servicios para un dominio particular; exporta así una serie de mecanismos y clases individuales que los clientes pueden usar o adoptar (Booch, 1994).

La definición de marco de referencia es el diseño reutilizable de todas las partes del sistema que esta representado por un conjunto de clases abstractas y de la forma en que sus instancias interactuan.

Otra definición común es que un marco de referencia es un esqueleto de una aplicación que puede ser personalizada por un desarrollador de aplicaciones. Estas definiciones no son contradictorias, la primera describe la estructura de un marco de referencia, y la segunda, el propósito (Booch, 1994).

Los marcos de referencia pueden, en realidad, ser neutrales respecto al dominio, lo que significa que pueden emplearse a una amplia gama de aplicaciones, tales

como las bibliotecas básicas generales, bibliotecas matemáticas y las bibliotecas para interfaz gráfica de usuarios entran en esta categoría.

La función de un marco de referencia es presentar una interfaz simple para un desarrollador, conteniendo un servicio simple o un conjunto de servicios, esto lo hace de manera fiable y eficiente. El desarrollador puede invocar los servicios enviando mensajes a sus clases, también de manera alterna, el desarrollador puede definir sus propias clases como subclases del marco de referencia, y por lo tanto, heredar sus conductas y de nuevo accesar los servicios.

La técnica ideal de reutilización es proporcionar componentes que son fáciles de conectar para crear un nuevo sistema. Los desarrolladores de sotfware no tienen que saber como estan implementados estos componentes, sin embargo, estos componentes deben ser fáciles de usar para el desarrolldor.

Los marcos mejoran significativamente la productividad, mantienen a los desarrolladores enfocados en los resultados del proyecto en lugar de la programación de este. Además prevé problemas de código común de cada programador. Logrando así un alto grado de consistencia dentro de cualquier proyecto.

Algunos ejemplos de marcos de referencia son las herramientas gráficas y calendarios. Las herramientas gráficas son usadas para expresar información breve y rápida; debe tener clases capaces de soportar varias formas de despliegue, selección y manipulación de la información fundamental. Similarmente, un calendario tiene la función de recobrar y desplegar información relacionada con periodos y días.

III.1 Requisitos de la Biblioteca de Clases Básicas en un Marco de Referencia

"La biblioteca básica de clases debe proporcionar una colección de estructuras de datos independientes del dominio y suficientes para cubrir las necesidades de la mayoría de las aplicaciones con calidad de producción. Además, la biblioteca debe ser:

- **Completa:** la biblioteca debe proporcionar una familia de clases unidas por una interfaz compartida, pero empleando una representación diferente, de forma que los desarrolladores puedan seleccionar la semántica de tiempo y espacio más apropiada para la aplicación de que se esté tratando.
- **Adaptable:** todos los aspectos específicos de la plataforma deben estar claramente identificados y aislados, de tal forma que puedan realizarse sustituciones locales. En particular, los desarrolladores deben tener control sobre políticas de administración del almacenamiento y de la semántica de sincronización de procesos.
- **Eficiente:** los componentes deben ser fáciles de ensamblar (eficientes en términos de compilación), imponer la menor sobrecarga en memoria y tiempo de ejecución (eficiente en términos de ejecución) y ser más fiable en los mecanismos creados particularmente (eficiente en términos del desarrollador).
- **Segura:** cada abstracción debe ser segura respecto al tipo, deben usarse excepciones para identificar condiciones bajo la cual se viola la semántica dinámica de una clase.
- **Simple:** la biblioteca debe usar una organización clara y consistente que facilite la identificación y selección apropiada de clases concretas.
- **Extensibles:** los desarrolladores deben ser capaces de añadir nuevas clases independientes, conservándose al mismo tiempo la integridad arquitectónica del marco de referencia" (Booch, 1994).

III.2 Ventajas del Desarrollo de Marcos de Referencia

- Explota la potencia expresiva de todos los lenguajes de programación orientados a objetos.
- Alimenta la reutilización de componentes de software.
- Lleva a sistemas más flexibles al cambio.
- Reduce el riesgo de desarrollo.
- Resulta atractivo al funcionamiento de la mente humana.

III.3 El Marco de Referencia llamado Infraestructura para Power Builder

En Power Builder se pueden construir Marcos de Referencia, el que aquí se presenta, esta desarrollado para la version 4.0, fue migrado para la versión 9.0 y consta de lo siguente:

De un conjunto de clases base que hacen más fácil la programación en este ambiente desarrollador. Incluye espacio de trabajo (*workspace*), ventanas, objetos de usuario (*user objects)*, funciones globales, estructuras, ventanas de datos (d*ata windows)* y otros objetos, con sus respectivos eventos, tanto nativos como definidos por el usuario (*user events*).

La infraestructura proporciona una funcionalidad básica para el manejo de catálogos, reporteadores, ventanas de respuesta (*response)* y menús, entre otras cosas; de esta manera, desarrollar el prototipo de un sistema es bastante fácil y rápido.

III.3.1 Niveles de la Infraestructura

La infraestructura consta de tres niveles: base, aplicativo y de transacción. En cada uno de estos niveles, se definen una serie de objetos con eventos (métodos) y programación genérica; los objetos de un nivel son heredados del nivel inmediato anterior; de esta manera, en el nuevo nivel se tiene la funcionalidad del primero más los eventos y código propio de éste.

En los apartados siguientes se explican cada uno de los niveles, las bibliotecas y los objetos que le pertenecen.

III.3.2 Nivel base

En este nivel se definen todos los objetos básicos genéricos para una aplicación que se encuantra en una biblioteca; el nombre de la biblioteca se forma con el prefijo *infr* y cuatro letras descriptivas de dicha biblioteca, las cuales estan conformadas de la siguiente manera:

- infraplb.pbl
- infrbase.pbl
- infrhost.pbl
- infrsegs.pbl

La descripción específica de cada biblioteca y sus objetos se detalla a continuación:

III.3.2.1 Infraplb.pbl

Esta biblioteca está conformada por objetos genéricos para cualquier aplicación. Se tomó originalmente de la biblioteca applib.pbl, proporcionada por Power Builder, y se adaptó a la infraestructura (se tradujeron al español las ventanas y los mensajes, se eliminaron algunos objetos, etc.). Los objetos que la conforman son:

OBJETO	DESCRIPCIÓN
f_block_text	Función para formatear una cadena dentro de un ancho especificado.
f_boolean_to_string	Función para definir la cadena 'Falso' o 'Verdadero', dependiendo del valor booleano.
f_db_error	Función para abrir la ventana para los errores de la base de datos.
f_error_box	Función para abrir la ventana w_error_box.
f_login	Función para abrir la ventana w_login.
Str_frame	Estructura para pasar información al *frame* de la aplicación.
Str_parms	Estructura con arreglos para todos los tipos de variables.
w_db_error	Ventana para desplegar los mensajes de error de la base de datos.
w_entrada	Ventana con el BMP de entrada al sistema.
w_error_box	Ventana no-modal para mensajes.
w_login	Ventana para la firma al sistema.
w_set_sqlca	Ventana para cambiar la conexión a la base de datos.
w_set_toolbars	Ventana para definir la barra de herramientas.

III.3.2.2 Infrbase.pbl

En esta biblioteca se encuentran los objetos para el desarrollo de sistemas, contienen eventos y código que proporciona funcionalidad básica para las transacciones, por ejemplo, para desarrollar un catálogo.

OBJETO	DESCRIPCIÓN
d_custommicrohelp	*Datawindow* para la barra de ayuda del *frame*.
d_windowstatus	*Datawindow* para la barra de estatus de las hojas.
f_descripcion	Función para obtener la descripción de un *dropdown datawindow*.
f_mensaje_error	Función que manda un mensaje de error.
f_mensaje_info	Función que manda un mensaje de información.
f_system_error	Función que controla los errores de ejecución.
m_base_menu	Menú base del sistema.
u_base_cb	*Command Button* base.
u_base_dw	*Control DataWindow* base.
u_base_em	*Edit Mask* base.
u_base_mle	*Multiline Edit* base.
u_base_pb	*Picture Button* base.
u_base_pic	*Picture* base.
u_base_sle	*Single Line Edit* base.
u_base_st	*Static Text* base.
u_base_st_ayuda	*Static Text* base para ayuda de botones.
u_base_transaction	*Transaction* base.
u_windowstatus	Barra de Status para las hojas.
w_base_frame	Ventana *MDI Frame* base.
w_base_mdetail	Ventana de hoja base para maestro-detalle, heredada de w_base_sheet.
w_base_mresponse	Ventana Base de Tipo Response con botones de Mantenimiento.

OBJETO	DESCRIPCIÓN
w_base_popup	Ventana *Popup* base.
w_base_response	Ventana de *Response* base.
w_base_sheet	Ventanas de hoja base.
w_base_rpreview	Ventana de hoja base para presentación preeliminar de reportes.
w_custommicrohelp	Barra de ayuda del *frame*.

III.3.2.3 Infrhost.pbl

La biblioteca del "host" contiene los objetos dedicados al lanzamiento de reportes en el servidor. Más adelante se profundizará en esta funcionalidad.

OBJETO	DESCRIPCIÓN
d_host	*Datawindow* para el mantenimiento al host.
d_monitoreo	*Datawindow* para el monitoreo de procesos.
f_host	Función para la ejecución de comandos de lanzadores.
f_lanza	Funcion que envia a ejecución los lanzadores.
f_validar_negativo	Función que valida que no se ingresen valores negativos en los campos numéricos de un *datawindow*.
sg_impresion	Estructura con el nombre de la impresora y el número de copias.
w_host	Ventana de mantenimiento a Procesos del Host.
w_impresora	Ventana para solicitar la impresora y copias para el lanzador.
w_intervalo	Ventana para solicitar el intervalo en segundos.
w_lanza	Ventana lanzadora, heredada de w_base_sheet.
w_monitoreo	Ventana para monitorear los procesos del usuario.

III.3.2.4 Infrsegs.pbl

En esta biblioteca se almacenan los objetos cuya función es controlar la seguridad del sistema, es decir, opciones del menú, acceso a transacciones, alta de usuarios, etc.

OBJETO	DESCRIPCIÓN
d_accesos	*Datawindow* de mantenimiento de accesos para procesos del sistema.
d_ddcia	*Dropdown Datawindow* de compañías.
d_ddciausuario	*Datawindow* de mantenimiento a niveles de acceso de compañías.
d_ddmodulo	*Dropdown Datawindow* de módulos.
d_ddproceso	*Dropdown Datawindow* de procesos.
d_ddusuario	*Dropdown Datawindow* de usuarios.
d_dlusuario	*Dropdown Datawindow* lista de usuarios.
d_empresa	*Datawindow* para la conexión a compañías.
d_mensaje	*Datawindow* para el catálogo de mensajes.
d_navegamod	*Datawindow* del navegador de módulos.
d_navegapro	*Datawindow* del navegador de procesos.
d_procesos	*Datawindow* de mantenimiento a procesos del sistema.
d_toolbar	*Datawindow* de mantenimiento a la barra de ayuda.
d_usuario	*Datawindow* de mantenimiento a usuarios del sistema.
f_acceso_proceso	Función que obtiene los accesos de determinado proceso.
f_nombre_cia	Función para obtener el nombre de la compañía.
f_orden_toolbar	Función que regresa el *handler* del *toolbar* según el orden.
f_preferencias	Función que construye el menú de preferencias.
w_accesos	Ventana de mantenimiento para los accesos de los procesos del sistema.
w_empresa	Ventana de conexión a empresas.
w_mensaje	Ventana de mantenimiento al catálogo de mensajes.

OBJETO	DESCRIPCIÓN
w_navegador	Ventana del navegador.
w_procesos	Ventana de mantenimiento a procesos.
w_toolbar	Ventana de mantenimiento a los procesos preferidos por el usuario.
w_usuario	Ventana de mantenimiento a usuarios del sistema.

III.3.3 Nivel Aplicativo

Este nivel está conformado por los objetos específicos para cada aplicación. El nombre de las bibliotecas depende del sistema pero siempre deben existir las siguientes:

- <sistema>apexe.pbl
- <sistema>ap<sistema>.pbl
- <sistema>dddw1.pbl
- <sistema>func1.pbl

Donde <sistema> son las 3 letras que describen al sistema.

III.3.3.1 <sistema>apexe.pbl

Los únicos dos objetos que se deben encontrar en esta biblioteca son el nombre de la aplicación y un proyecto para la generación del archivo ejecutable.

OBJETO	DESCRIPCIÓN
<sistema>	Aplicación para el sistema.
<sistem>proy	Proyecto para la generación del archivo ejecutable.

III.3.3.2 <sistema>ap<sistema>.pbl

La biblioteca de la aplicación esta conformada por objetos genéricos a una aplicación específica.

OBJETO	DESCRIPCIÓN
m_<sistema>_menu	Menú aplicativo del sistema, heredado de m_base_menu.
u_<sistema>_cb	*Command Button* aplicativo, heredado de m_base_cb.
u_<sistema>_dw	*Control DataWindow* aplicativo, heredado de m_base_dw.
u_<sistema>_em	*Edit Mask* aplicativo, heredado de m_base_em.
u_<sistema>_mle	*Multiline Edit* aplicativo, heredado de m_base_mle.
u_<sistema>_pb	*Picture Button* aplicativo, heredado de m_base_pb.
u_<sistema>_pic	*Picture* aplicativo, heredado de m_base_pic.
u_<sistema>_sle	*Single Line Edit* aplicativo, heredado de m_base_sle.
u_<sistema>_st	*Static Text* aplicativo, heredado de m_base_st.
u_<sistema>_st_ayuda	*Static Text* aplicativo para ayuda de botones, heredado de m_base_st_ayuda.
u_<sistema>_transaction	*Transaction* aplicativo, heredado de m_base_transaction.
w_<sistema>_frame	Ventana *MDI Frame* aplicativa, heredado de m_base_frame.
w_<sistema>_mdetail	Ventana de hoja aplicativa para maestro-detalle, heredada de w_base_mdetail.
w_<sistema>_mresponse	Ventana Aplicativa de Tipo Response con botones de Mantenimiento, heredada de m_base_mresponse.
w_<sistema>_popup	Ventana *Popup* aplicativa, heredada de m_base_popup.
w_<sistema>_response	Ventana de *Response* aplicativa, heredada de m_base_response.

OBJETO	DESCRIPCIÓN
w_<sistema>_sheet	Ventanas de hoja aplicativa, heredada de m_base_sheet.
w_<sistema>_rpreview	Ventana de hoja aplicativa para presentación preeliminar de reportes, heredada de m_base_rpreview.

III.3.3.3 <sistema>dddw1.pbl

En esta biblioteca se almacenan los *dropdown datawindows* del sistema. El número de ellos y sus nombres dependen del sistema. La tabla que se presenta a continuación es un ejemplo del sistema de nómina.

OBJETO	DESCRIPCIÓN
Nomddarea	*Dropdown datawindow* para las áreas.
Nomddause	*Dropdown datawindow* para los ausentismos.
Nomddbanc	*Dropdown datawindow* para los bancos.

III.3.3.4 <sistema>func1.pbl

Esta biblioteca almacena las funciones globales del sistema. El número y nombre de las funciones también depende del sistema.

OBJETO	DESCRIPCIÓN
f_cve_categoria	Función para obtener la clave de categoría.
f_edo_empleado	Función para obtener el estado de un empleado.
f_ins_integrado	Función para insertar un integrado.

III.3.4 Nivel de Transacción

En este nivel se definen los objetos para las transacciones del sistema, por ejemplo, ventanas de catálogos, alta de empleados, etc.

Pueden existir tantas bibliotecas como se necesiten, el nombre se forma como sigue:

<sistema><descripción>.pbl

III.3.5 Conexión a la Base de Datos

La conexión a la base de datos se efectúa mediante el objeto *SQLCA,* ésta es una variable global que conoce Power Builder y el manejador de la base de datos, los campos de esta variable se llenan con un archivo de inicialización llamado *<sistema>.ini*, el nombre del usuario y *password* que son capturados en la ventana de login. También es posible cambiar estos valores en tiempo de corrida mediante la opción *Conectar a Base de Datos*, del submenú *Archivo*.

III.3.6 Mensajes de Error de la Base de Datos

Para dejar genéricos los mensajes de error de la base de datos se cuenta con la tabla SEG_MENSAJE y la transacción *w_mensaje*. En esta tabla se dan de alta los códigos de error y el mensaje correspondiente de la base de datos. Por ejemplo, una llave duplicada en Oracle tiene el código -1, mientras que en Sybase es 2601. Esto facilita la codificación y traducción de errores porque para cada manejador se da de alta la tabla correspondiente.

Solamente existirán "mensajes de error" para aquellos que se encuentran en la base de datos, no para los mensajes de las validaciones particulares.

III.3.7 Control de la Seguridad

Para controlar la seguridad de la aplicación se cuenta con un conjunto de tablas, transacciones y variables de instancia. Existen primordialmente tres tipos de seguridad: seguridad para accesar la aplicacion, permisos para acceso a la información de las tablas y permisos específicos por aplicación.

III.3.8 Acceso a la Aplicación

Para tener acceso a la aplicación se debe contar con un usuario de la base de datos; este usuario se crea con las herramientas que proporciona el proveedor. Una vez dado de alta al usuario en el servidor, éste deberá darse de alta en la tabla *SEG_USUARIO* para que pueda utilizar el sistema. La transacción para hacerlo es *w_usuario*, la cual se encuentra en *infrsegs.pbl*; existe una particularidad, ya que la primera vez es necesario insertar por medio de SQL al primer usuario de la tabla.

En la tabla *SEG_USUARIO* se define el usuario y password de la aplicación, junto con su nivel de acceso. Este nivel es el que permite hacer la unión con las tablas *SEG_PROCESO* y *SEG_ACCESO* para que el usuario tenga derecho a un menú específico del navegador.

La tabla *SEG_PROCESO* contiene todas las opciones del sistema. Para llenar esta tabla existe la ventana *w_procesos* en la biblioteca *infrsegs.pbl*. Asimismo, la tabla *SEG_ACCESO* es la que define las opciones que se tomarán de

SEG_PROCESO para construir el navegador por usuario firmado; la ventana para esta tabla es *w_accesos* y se encuentra en la misma biblioteca. El acceso se construye a partir de un rango de niveles que tienen derecho a la opción; de esta manera, una opción puede estar englobada en varios rangos de niveles y estar disponible para usuarios con diferente nivel.

III.4 Marcos de Referencia para Internet

Las aplicaciones distribuidas son una opción para crear "Marcos de Referencia para Internet", pues estas aplicaciones se crean con el fin de aprovechar la capacidad de procesamiento que ofrecen los servidores, generando software escalable y flexible.

Una aplicación distribuida suele combinar elementos del software tradicional, como una base de datos, con las tecnologías basadas en Internet, como un navegador.

Los componentes son los pilares de las aplicaciones distribuidas, es un módulo de software escrito para controlar un tipo concreto de información o un proceso en particular. Cada uno se diseña de tal manera que pueda modificarse fácilmente para ajustarse a los requisitos determinados. De esta manera, en vez de escribir el mismo tipo de código una y otra vez, los programadores pueden utilizar código de software prefabricado y sólo necesitan ajustar los elementos que son diferentes en la aplicación. Los programadores crean aplicaciones mediante la integración de componentes.

El software de componentes sirve para mucho más, aparte de reducir la cantidad de código que los programadores tienen que escribir desde el principio, puesto que las aplicaciones creadas con dichos componentes pueden vincularse a través

de redes y también supone una manera eficaz de crear aplicaciones distribuidas, que dividen el proceso entre los equipos cliente y servidor.

III.4.1 Arquitectura de n capas

A finales de los ochenta, las aplicaciones se escribían, por lo general, para que se ejecutaran por completo en un equipo único. A principios de los noventa, apareció el modelo cliente-servidor de dos capas que permitió a los programadores descargar en los equipos cliente parte del trabajo con mayor procesamiento de datos y trasladarlo a servidores de fondo con más capacidad. En este modelo de dos capas, el software de presentación (la interfase de usuario) permaneció en el equipo personal, mientras que la mayor parte del trabajo de procesamiento se traslado al servidor. El modelo de tres capas agrega un elemento adicional de separación entre los niveles de datos y de presentación, ya que permite controlar la lógica de procesamiento en un servidor independiente de las funciones de base de datos y de presentación.

La arquitectura de n capas separa una aplicación en tres componentes distintos:

- **Presentación:** Es la parte de la aplicación con la que interactúa un usuario.
- **Lógica de la aplicación:** Este componente contiene todas las reglas y la lógica empresarial asociadas con la aplicación.
- **Datos:** Se trata del mecanismo que almacena y administra los datos asociadas con la aplicación.

La división de aplicaciones en las secciones de presentación, lógica de la aplicación y datos, permite un modelo de programación simplificado que es la manera estándar de crear aplicaciones que aprovechan las ventajas de las comunicaciones de Internet y de Intranet.

IV. GENERACIÓN DE UNA APLICACIÓN UTILIZANDO LA INFRAESTRUCTURA

Los pasos a seguir para generar una aplicación nueva son los siguientes:

IV.1 Creación de Bibliotecas Base

- Copiar las bibliotecas de infraestructura de alguna otra aplicación, *infrbase.pbl, infrapli.pbl, infrhost.pbl* e *infrsegs.pbl.*
- Crear la librería <sistema>apexe.pbl, con el objeto *Application* correspondiente.
- Para el objeto *Application* sustituir la variable global SQLCA por el *user object u_<sistema>_transaction.*
- Crear la librería <sistema>ap<sistema>.pbl con los objetos heredados de *infrbase.pbl.*
- Crear la librería <sistema>dddw1.pbl para los *dropdown datawindows.*
- Crear la librería <sistema>func1.pbl para las funciones globales.
- Crear la librería <sistema>query.pbl para las ventanas *Quero.*

IV.2 Creación de Bibliotecas Específicas

Las bibliotecas específicas dependen de cada aplicación, en general, se busca que no excedan de 800 Kbytes o de 40 objetos para que el archivo ejecutable se genere correctamente. Además, se deben agrupar dentro de una misma librería los objetos que se ligan entre ellos, por ejemplo, tanto la ventana como su *datawindow* deben estar juntos.

Se deben utilizar comentarios explicativos para cada una de las bibliotecas y objetos.

IV.3 Inicialización de la Base de Datos

Para una aplicación nueva se necesita dar de alta las claves de usuario de la base de datos, crear sus tablas e insertar los datos iniciales. La creación de las claves de usuario y sus tablas se hace por medio de las herramientas del proveedor de la base de datos (*SQLPLUS* o *ISQL* para Oracle y Sybase, respectivamente), los pasos para la inserción son:

- Insertar las opciones en *SEG_ACCESO* y *SEG_PROCESO*. Usualmente se toman los datos de algún otro sistema funcionando y se eliminan las que no correspondan.
- Insertar el primer usuario en *SEG_USUARIO*.
- Insertar el permiso del usuario para utilizar la compañía en *SEG_CIA_USUARIO*.

IV.4 Creación del Archivo ini

Crear el archivo <sistema>.ini tal como se especificó en la sección de Conexión a Base de Datos.

IV.5 Generación del Archivo Ejecutable

Para generar el archivo ejecutable es necesario tomar en cuenta los factores de hardware, software y la organización de las bibliotecas. Esto se explica a continuación.

IV.6 Requerimientos de las Bibliotecas

Son tres puntos los que se deben afinar para el archivo ejecutable:

- **Tamaño de las bibliotecas**: No debe exceder 800 Kbytes o 40 objetos
- **Orden de la lista de bibliotecas**: Deben estar definidas en el orden en el cual se heredan y ocupan los objetos. El orden típico es el siguiente:
  - <sistema>apexe.pbl
  - infrapli.pbl
  - infrbase.pbl
  - infrhost.pbl
  - infrsegs.pbl
  - <sistema>ap<sistema>.pbl
  - <sistema>dddw1.pbl
  - <sistema>func1.pbl
  - <sistem>query.pbl
  - Las demás bibliotecas.
- **Distribución de objetos en las bibliotecas:** Los objetos que estén referenciados entre sí deben estar en la misma biblioteca. En la biblioteca <sistema>apexe.pbl solamente deben estar los objetos *Application* y *Project*.

IV.7 Creación del Proyecto

Para generar el archivo ejecutable debe crearse primero un proyecto en el *Project Painter* y no utilizar la opción *Create Executable* del submenú *File*, en el *Application Painter*. Esta última opción es para aplicaciones pequeñas.

Una vez creado el proyecto se deben marcar los *checkboxes* para que se genere un archivo *pbd* para cada biblioteca, excepto la biblioteca <sistema>apexe.pbl

El *checkbox* permite regenerar todos los objetos; existe la posibilidad de marcarse únicamente pero es más recomendable regenerar manualmente cada biblioteca. Esto es porque algunas veces se bloquea la máquina, dependiendo del tamaño de la memoria y de los *drivers* cargados.

IV.8 Programación del Sistema

En esta sección se presentan los pasos a seguir para programar una transacción dentro del sistema, se utilizan los eventos y variables de instancia descritos en las secciones anteriores y se describe el uso de algunas funciones globales genéricas.

IV.8.1 Programación de Ventanas

En esta sección se describen los pasos a seguir para programar una ventana con la infraestructura.

IV.8.2 Creación de una Ventana Nueva

Para la programación de ventanas se deben realizar los siguientes pasos:

- Heredar la hoja de w_<sistema>_sheet.
- Agregar las *datawindows* heredándolos de u_<sistema>_dw.
- Agregar en el evento *ue_postopen* de la ventana, las instrucciones para definir cual *datawindow* es maestro, cuales no y cual es el actual. Por ejemplo:

  - dw_toolbar.ib_dwMaestro = True

```
- dw_usuario.ib_dwMaestro = False
- This.idw_Maestro = dw_toolbar
- This.idw_Actual = dw_toolbar
```

- Agregar en el evento *constructor* de la datawindow, si se registra la datawindow para que se le envíen todos los mensajes del menú o no; los mensajes son usualmente ue_salvar, ue_insertar, ue_borrar, ue_criterio, ue_consultar, etc., por ejemplo:

```
- This.ib_Registrar_dw = False
```

- También se debe agregar en el evento *constructor* de la *datawindow* los campos de auditoría. Por ejemplo:

```
- This.is_User_Creo = "acc_user_creo"
- This.is_Fec_Creo  = "acc_fec_creo"
- This.is_User_Mod  = "acc_user_mod"
- This.is_Fec_Mod   = "acc_fec_mod"
```

  Estos campos de auditoría son los nombres de los campos en la *datawindow*, que en la mayoría de los casos son los mismos nombres que en las columnas de la tabla.

- Efectuar las validaciones propias de cada transacción, por ejemplo, validar los valores de los campos, rangos de fechas, etc. Por omisión, la infraestructura salva, inserta, borra, permite introducir y ejecutar consultas. Si se desea suprimir estas acciones se debe sobrescribir el *script* y llamarlos posteriormente, si así se requiere.

IV.8.3 Eventos Típicos para Validaciones

Los eventos típicos para validaciones en la *datawindow* son:

- **ItemChanged**: Este evento sirve para validar los cambios que se hacen por campo, para lo cual se utiliza la instrucción *Choose Case*; dentro de cada columna se efectúan las validaciones correspondientes, por ejemplo:

```
String ls_Columna, ls_Valor
```

```
Long ll_Registro, ll_Nulo
Double ldb_Saldo

If This.ib_Criterio Then
  Return
End If

ls_Columna  = This.GetColumnName()
ls_Valor    = This.GetText()
ll_Registro = This.GetRow()

If ll_Registro < 1 Then
  Return
End If

CHOOSE CASE ls_Columna

    CASE 'mba_cve_mov'
    f_Descripcion(This, ls_Columna, 'wk_nom_movto', ls_Valor, &
            'tmo_cve_mov', 'tmo_desc', ll_Registro)

    CASE 'mba_importe_original'
    If Not fw_Calcular_Importe(ls_Columna, ls_Valor) Then
      This.SetActionCode(1)
      Return
    End If

    If Not fw_Obtener_Saldo(ldb_Saldo) Then
      This.SetActionCode(1)
      Return
    End If

END CHOOSE
```

IV.8.4 Paso de Parámetros entre Ventanas

Para pasar parámetros entre ventanas se utiliza una estructura global llamada *str_parms*, la cual contiene arreglos de todos los tipos de datos de Power Builder. La ventana invocadora llena la estructura y abre a la ventana invocada pasándole esta estructura; la ventana invocada debe tomar la estructura en la primera instrucción ejecutable del evento *Open*. La ventana invocadora puede abrir la otra ventana en el evento *DoubleClicked* de la *datawindow* o en el *Clicked* de un botón. Es importante señalar que la ventana invocada es de tipo *u_<sistema>_mresponse* o *u_<sistema>_response*.

IV.8.5 Protección de Campos

Para que un campo de la *datawindow* se proteja automáticamente dependiendo de los valores de otros campos o de otras condiciones, se utiliza el atributo *Protect* junto con funciones predefinidas de la *datawindow*. Por ejemplo, para proteger la llave de un registro consultado de la base de datos se programa en el *Protect* del campo lo siguiente:

```
Protect = If(IsRowNew(), 0, 1)
```

De esta manera, si el registro es nuevo se tiene acceso al campo (*Protect=0*), si es consultado de la base de datos está protegido (*Protect=1*).

Otro ejemplo, para el campo vacm_fecha se define:

```
Protect = If(vacm_dias > 0, 0, 1)
```

Donde vacm_dias es otra columna de la *datawindow*.

IV.8.6 Ventanas Maestro Detalle

Una ventana maestro-detalle contiene una *datawindow* maestro y una de detalle. Se debe coordinar manualmente el registro maestro con sus detalles correspondientes; para esto, los detalles se definen con *retrieval arguments*; cada vez que cambia el registro en el maestro se efectúa la consulta en los detalles. Sí el maestro ha tenido cambios se despliega el mensaje para preguntar al usuario si desea salvar los cambios. Estas características ya están programadas en el objeto *u_base_mdetail* y su correspondiente *u_<sistema>_mdetail*; si se desea una ventana maestro-detalle se hereda de este objeto y no de *u_<sistema>_sheet*; por lo que respecta a la programación, es muy similar a la de una hoja normal.

Si se necesita una ventana con un maestro y varios detalles se deben programar en forma particular porque no existe ningún objeto base para ello.

IV.8.7 Diseño Óptimo de la Base de Datos

Siempre es recomendable que el cliente efectúe el mayor número posible de operaciones para disminuir el acceso a la base de datos, y por lo tanto, el tráfico en la red. Por ejemplo, se pueden agrupar y ordenar las columnas desde la definición de la *datawindow* pero es más conveniente hacerlo del lado del cliente con las opciones que éste proporciona; es decir, no hay que hacerlo en el *Select*, sino con las opciones de la *Datawindow Painter*.

Otras recomendaciones son:

- Para evitar problemas de portabilidad con la base de datos, las sentencias SQL siempre deben codificarse en mayúsculas
- Las sentencias *Insert* deben programarse explícitamente con los campos de la tabla. Por ejemplo:

  Técnica errónea:

```
INSERT INTO CG_SEQNOS
VALUES(1, 1, 'E');
```

  Técnica óptima:

```
INSERT INTO CG_SEQNOS(SEQ_CVE_CIA, SEQ_NUM_POL, SEQ_TPO_POL)
VALUES(1, 1, 'E');
```

- Siempre que sea posible, se deben utilizar pocas uniones de tablas para agilizar el acceso a la base de datos, sobretodo cuando el *join* se hace para obtener descripciones. Por ejemplo:

  Técnica errónea:

```
// Se obtiene la clave del movimiento bancario y su descripción
String ls_Cve_Movto, ls_Desc_Movto
        SELECT MBA_CVE_MOVTO, TMO_DESC
        INTO :LS_CVE_MOVTO, :LS_DESC_MOVTO
```

```
FROM BAN_MOV_BAN, BAN_TPO_MOVTOS
WHERE MBA_CVE_CIA = :SQLCA.IL_CLAVE_CIA
   AND MBA_CVE_BAN = :IS_CVE_BANCO
   AND MBA_CTA_CHEQ = :IS_CVE_CHEQUERA
   AND TMO_CVE_CIA = :SQLCA.IL_CLAVE_CIA
   AND TMO_CVE_MOV = MBA_CVE_MOVTO;
```

Técnica óptima:

```
// Se obtiene la clave del movimiento bancario y su descripción

String ls_Cve_Movto, ls_Desc_Movto

// Se obtiene la clave del movimiento

        SELECT MBA_CVE_MOVTO
        INTO :LS_CVE_MOVTO
        FROM BAN_MOV_BAN
        WHERE MBA_CVE_CIA = :SQLCA.IL_CLAVE_CIA
         AND MBA_CVE_BAN = :IS_CVE_BANCO
         AND MBA_CTA_CHEQ = :IS_CVE_CHEQUERA;
// Se obtiene la descripción del movimiento.

        SELECT TMO_DESC
        INTO LS_DESC_MOVTO
        FROM BAN_TPO_MOVTOS
        WHERE TMO_CVE_CIA = :SQLCA.IL_CLAVE_CIA
         AND TMO_CVE_MOV = MBA_CVE_MOVTO;
```

- Respecto al punto anterior, siempre hay que programar los *joins* en el mismo orden que están definidos los índices.

- Después de cada sentencia SQL debe verificarse el código de retorno, utilizando la función *SQLCA.uof_VerificarRetorno()*; esta función retorna *FALSE* si se efectuó un error en la base de datos, y *TRUE* en caso contrario; para el error típico *NotFound*, la función también retorna *TRUE* y se debe preguntar explícitamente por el error 100 (predefinido en Power Builder). Por ejemplo:

```
// Se obtiene la clave del movimiento bancario y su descripción

String ls_Cve_Movto, ls_Desc_Movto

// Se obtiene la clave del movimiento

        SELECT MBA_CVE_MOVTO
        INTO :LS_CVE_MOVTO
        FROM BAN_MOV_BAN
        WHERE MBA_CVE_CIA = :SQLCA.IL_CLAVE_CIA
         AND MBA_CVE_BAN = :IS_CVE_BANCO
         AND MBA_CTA_CHEQ = :IS_CVE_CHEQUERA;

        If Not SQLCA.uof_VerificarRetorno() Then
          f_Mensaje_Error('Error al obtener el movimiento bancario')
```

```
        Return False
      End If

      If SQLCA.sqlcode = 100 Then
        f_Mensaje_Error('No existe la clave del movimiento')
        Return False
      End If
// Hasta aquí no hubo errores.
```

- Como punto final, siempre debe programarse el *COMMIT* o *ROLLBACK*, dependiendo de si hubo error o no, para que se hagan permanentes las actualizaciones en la base de datos o se deshagan todos los cambios. En el lugar donde se programa el *commit* o *rollback* depende de la transacción, aunque por omisión, el evento *ue_salvar* de la ventana lo efectúa.

IV.8.8 Recomendaciones Generales

La mayoría de las validaciones se efectúan en funciones de ventana, no dentro del evento *ItemChanged*; esto hace al código más legible y fácil de seguir.

Al utilizar la función *datawindow.GetRow()* siempre se debe verificar si es un registro válido; si no se valida, se puede presentar el error *Invalid Column/Row Name* al ejecutar la aplicación.

Los mensajes de error se despliegan con la función *f_Mensaje_Error*.

Los mensajes de información se despliegan con la función *f_Mensaje_Info*.

Verificar siempre si la *datawindow* está en modo consulta utilizando la variable *datawindow.ib_Criterio*, en cuyo caso no se valida nada.

- **ue_insertar:** Este evento inserta un registro en el *datawindow*, pero por lo general, está sobrescrito para que además inserte valores por omisión de

algunos registros, como por ejemplo, las llaves capturadas en otra *datawindow*.

- **ue_borrar:** En este evento se borra el registro, pero además, se pueden efectuar validaciones extras.
- **ue_salvar:** Este evento es uno de los más importantes porque permite controlar la actualización a la base de datos, efectuar validaciones, etc. Además, se validan automáticamente los campos requeridos por medio de una función encapsulada en la *datawindow* base, dicha función encuentra las columnas de una *datawindow* y llama a la función *FindRequired* para verificar si el campo es obligatorio o no; la función se dispara antes de actualizar la *datawindow* y no se dispara un *ItemError* si se deja un campo en blanco.
- **ue_criterio:** Ponen a la *datawindow* en modo consulta y usualmente se utiliza tal como está programado en la base de datos.
- **ue_consultar:** Este evento es típico que se sobrescriba porque la mayoría de las *datawindows* efectúan la consulta con *retrieval arguments*. Después de consultar los registros se recorre la *datawindow* para llenar las descripciones en los campos de trabajo, para lo cual se utiliza la función *f_descripcion*, explicada anteriormente. Por ejemplo:

```
if not ib_permitir_consultar then
  return
end if

This.SetRedraw(False)
This.Modify("DataWindow.QueryMode=no")
This.Retrieve(SQLCA.il_Clave_Cia, is_Cve_Banco, is_Cve_Chequera)
If This.RowCount() < 1 Then
  This.TriggerEvent("ue_insertar")
  w_CustomMicroHelp.wf_SetMicroMessage("No hay registros que consultar")
End If

// Se llena los campos de trabajo
fw_Campos_Trabajo()

This.SetRedraw(True)
This.SetFocus()

this.ib_criterio=FALSE
```

- **ue_preupdate:** Este evento se dispara antes de actualizar el registro en la base de datos; se utiliza para obtener los valores que se van a modificar y poder ocuparlos en alguna otra tabla. También se pueden validar los valores y detener el procesamiento si así se requiere.
  La función para obtener el número de registro a actualizarse no es *datawindow.GetRow()*, sino *datawindow.GetUpdateStatus()*.
- **ue_preinsert:** Este evento se dispara antes de insertar el registro en la base de datos; se utiliza para obtener los valores que se van a agregar y poder ocuparlos en alguna otra tabla. En este evento también se validan los valores a insertar en la base de datos; idealmente, deben de contener las mismas validaciones hechas en el *ItemChanged*.
  Al igual en *ue_preupdate*, la función para obtener el número de registro a insertarse no es *datawindow.GetRow()*, sino *datawindow.GetUpdateStatus()*.
- **ue_predelete:** Se trabaja igual que *ue_preupdate* y *ue_preinsert*, pero antes de borrar un registro.

Se debe aclarar que para estos tres eventos, no es posible modificar los valores que se van a insertar, actualizar o borrar ya que solamente sirven para validar los valores y para ocuparlos en lo que se requiera.

V. APLICACIÓN

V.1 Especificación del Problema

El sistema de control de adquisiciones tiene por objetivo automatizar el proceso de compras que efectúa el personal de cierta compañía juntando las solicitudes de compras de todos los bienes materiales que solicitan en sus centros de trabajo (Campus).

Esta compañía proporciona dos tipos de servicios: la atención de requisiciones de compra y el surtido mensual de artículos y materiales de uso común por medio de su almacén central.

V.2 Funcionalidad Solicitada

El sistema incluye la captura, validación, autorización e impresión de requisiciones, además de reportes.

El módulo de cotizaciones abarca transacciones, selección de proveedores, generación e impresión de cuadros comparativos.

En cuanto al módulo de pedidos, éste se realiza a partir de las cotizaciones existentes dentro del mismo, generando reportes especiales y una bitácora. Por último, se realiza la recepción y reporte de materiales.

Además, se cuenta con un módulo de catálogos para almacenar, consultar y actualizar la información necesaria, con la finalidad de proporcionar una mayor flexibilidad al sistema, así como evitar duplicidad y redundancia en la información.

V.3 Generación de la Aplicación Utilizando la Infraestructura

Para la versión 9.0 de Power Builder se creó un espacio de trabajo (*workspace*) llamado SP_ADQUISICIONES y, posteriormente, se generó el blanco (*target*) TG_ADQUICICIONES creando las siguientes bibliotecas:

Bibliotecas Base

BIBLIOTECA	DESCRIPCIÓN
INFRBASE.PBL	Objetos Base para Heredar Propiedades
INFRSEGS.PBL	Objetos para la Administración del Sistema
INFRHOST.PBL	Objetos para Ejecutar los Reportes
INFRAPLB.PBL	Objetos para la Firma de Usuario al Sistema

Bibliotecas del Sistema

BIBLIOTECA	DESCRIPCIÓN
ADQAPADQ.PBL	Objetos Heredados de la Clase INFRBASE.PBL
ADQFUNC1.PBL	Funciones Globales
ADQDDDW1.PBL	Datawindows para la Selección de Cajas de Texto
ADQCATAL.PBL	Contiene los Objetos de los Catálogos
ADQCAMPU.PBL	Contiene los Objetos de los Campus
ADQREQUI.PBL	Contiene los Objetos de las Requisiciones
ADQCUCOM.PBL	Contiene los Objetos de los Cuadros Comparativos
ADQPEDID.PBL	Contiene los Objetos de los Pedidos
ADQRECEP.PBL	Contiene los Objetos de las Requisiciones Recibidas
ADQESTAD.PBL	Contiene los Objetos de las Estadísticas

Ventanas de la Biblioteca ADQCATAL.PBL

VENTANA	DESCRIPCION
ADQWCUNI	Catálogo de Unidades de Medida de Artículos
ADQWCMCA	Catálogo de Marcas de Artículos
ADQWTAR	Catálogo de Tipos de Artículos
ADQWCART	Catálogo de Artículos
ADQWCEMP	Catálogo de Empresas
ADQWCPTL	Catálogo de Campus
ADQWCPRO	Catálogo de Proveedores
ADQWCEST	Catálogo de Estatus
ADQWART	Reporte de Catálogo de Artículos
ADQWLCATA	Interfaz de Catálogo de Campus

Ventanas de la Biblioteca ADQCAMPU.PBL

VENTANA	DESCRIPCIÓN
ADQWMRQPTL	Captura de Requisiciones
ADQWLRQESARC	Interfaz de Requisiciones de Campus Empresa
ADQWLRQACTPTL	Carga de Requisiciones en el Campus
ADQWLACATA	Carga de Catálogos en el Campus
ADQWRECPEDPTL	Carga de Pedidos en el Campus
ADQWRCQCONSOLID	Interfaz de Consolidaciones de Requisiciones
ADQWRQCARCONS	Carga de Consolidaciones de Requisiciones

Ventanas de la Biblioteca ADQREQUI.PBL

VENTANA	DESCRIPCIÓN
ADQWLRQCARGA	Carga de Requisiciones en la Compañía
ADQWASICOMREQ	Asignación de Requisiciones a Compradores
ADQWMRQMA	Mantenimiento de Requisiciones
ADQWLREQPTL	Interfaz de Requisiciones de la Compañía al Campus
ADQWLREQM	Impresión de Requisiciones

Ventanas de la Biblioteca ADQESTAD.PBL

VENTANA	DESCRIPCIÓN
ADQWLREQ	Estadística de Requisiciones por Campus
ADQWLPAR	Estadística de Partidas por Campus
ADQWLPARPTL	Reporte de Partidas por Campus
ADQWLCOM	Estadística de Requisiciones por Comprador
ADQWLPARCOM	Estadística de Partidas por Comprador
ADQWLREQCOM	Reporte de Partidas por Comprador

Ventanas de la Biblioteca ADQCUCOM.PBL

VENTANA	DESCRIPCIÓN
ADQWMANCC	Generación de Cuadros Comparativos
ADQWLCOTIZA	Impresión de Cuadros Comparativos

Ventanas de la Biblioteca ADQPEDID.PBL

VENTANA	DESCRIPCIÓN
ADQWLGPED	Generación de Pedidos
ADQWMANPE	Consulta de Pedidos
ADQWLPED	Impresión de Pedidos
ADQWGENPEDPTL	Interfaz de Pedidos de la Compañía al Campus

- ADQWLGPEDCAJA Generación de Pedidos Caja Chica
- ADQWMANPECAJA Mantenimiento de Pedidos Caja Chica
- ADQWLPEDCAJA Impresión de Pedidos de Caja

Ventanas de la Biblioteca ADQRECEP.PBL

VENTANA	DESCRIPCIÓN
ADQWLRECMAT	Recepción de Materiales
ADQWLRREC	Reporte de Recepción
ADQWGENPED	Interfaz de Recepción
ADQWRECPED	Carga de Recepción en la Compañía

Modulo de Seguridad del Sistema

VENTANA	DESCRIPCIÓN
W_PROCESOS	Mantenimiento de Procesos
W_ACCESOS	Mantenimiento de Accesos a Procesos
W_USUARIOS	Mantenimiento a Usuarios
W_TOOLBAR	Mantenimiento a Preferencias
W_MENSAJE	Mantenimiento a Mensajes del Sistema

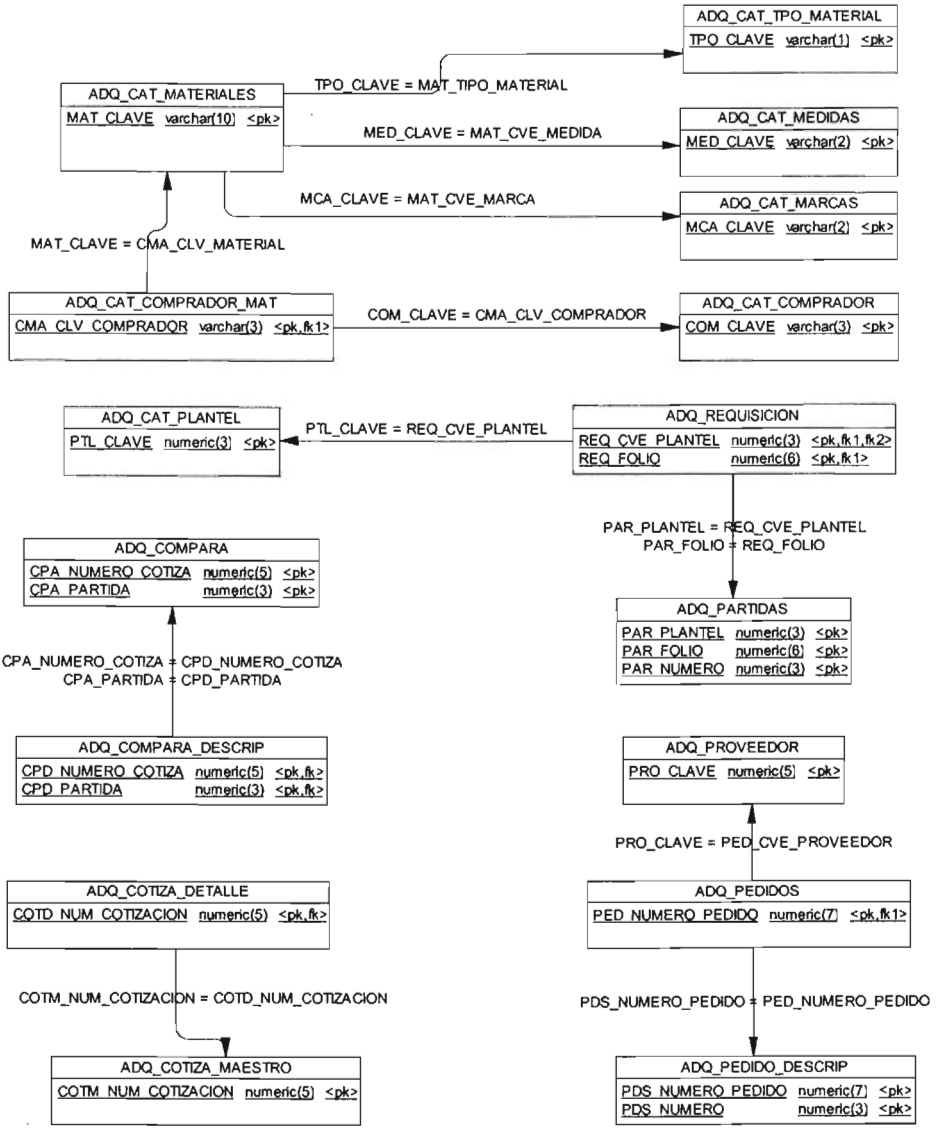
V.4 Inicialización de la Base de Datos

Para el desarrollo de esta aplicación de utilizó la base datos Anywhere versión 9.0 generada con la herramienta Central Sybase versión 4.1 creando las siguientes tablas:

ADQ_ARTICULO
ADQ_EMPRESA
ADQ_ESTATUS
ADQ_MARCAS
ADQ_MEDIDAS
ADQ_PLANTEL
ADQ_PROVEEDOR
ADQ_COTIZA
ADQ_COTIZA_DETALLE
ADQ_PEDIDOS
ADQ_PEDIDOS_DET
ADQ_PARTIDAS
ADQ_REQUISICION
SEG_ACCESO
SEG_HOST

SEG_MENSAJE
SEG_PROCESO
SEG_TOOLBAR
SEG_USURIO

DIAGRAMA BASE DE DATOS DEL SISTEMA DE ADQUISICIONES



La comunicación de la base de datos de realizó a través del administrador Conexión de Base de Datos Abierta (ODBC), generando un nombre de fuente de datos (DNS) llamado *dbadq*.

Se creó un usuario DBA, con los privilegios de selección, inserción, actualización y borrado en las tablas anteriores y se agregó a la tabla *SEG_USUARIO* con el nivel de acceso 999 que accede a todos los módulos del sistemas.

V.5 Creación del Archivo ini

El archivo de inicialización adquisición.ini contiene las configuraciones para conectarse a la base de datos, como son: el tipo de conexión si es nativa o por ODBC, nombre del servidor, nombre de la base de datos, clave del usuario y parámetros de la base de datos, en este caso, el DNS. El archivo se muestra a continuación:

```
[Database]                    sección de base de datos
DBMS = "ODBC"                 indica que la conexión será mediante ODBC
AutoCommit = False            las transacciones auto-salvado son por falsa omisión
DBParm = "ConnectString='DSN=dbadq;UID=dba;PWD='"          para leer el DNS
ServerName=123456789          nombre del servidor
Database=Adquisición          nombre de la base de datos
LogId=DBA                     login de la aplicación
```

V.6 Generación del Ejecutable

Para la generación del ejecutable se creó un proyecto de la aplicación mediante el la herramienta *Proyect* (*Painter*) ye cual de agrego en la biblioteca *ADQAPEXE* con el nombre *adqproy*.

Una vez creado el proyecto se marcan las casillas para generar las bibliotecas dinámicas *pbd*, para cada biblioteca en esta sección se pueden agregar archivos de recursos *pbr* si se requieren (figura 5.1).

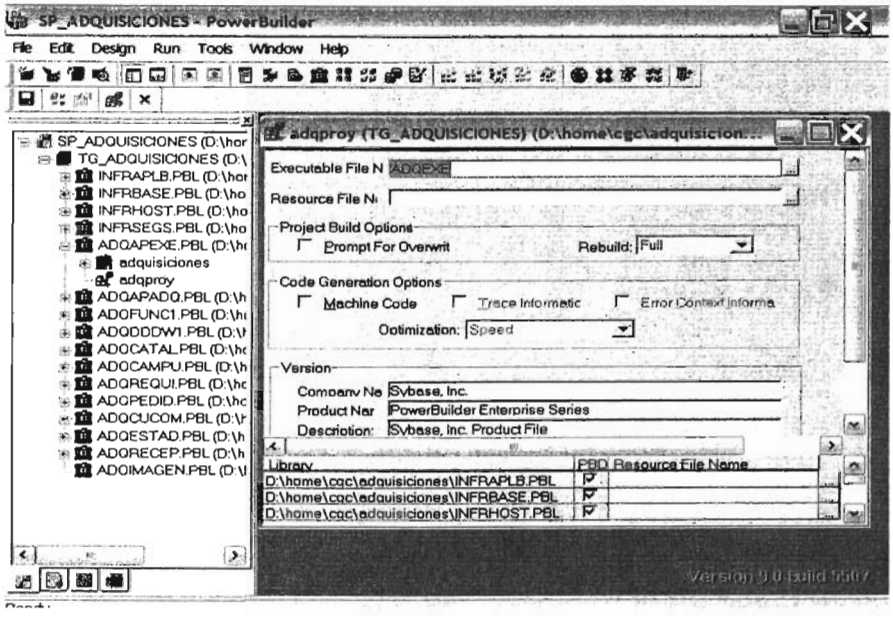


Figura 5.1 Generación del archivo ejecutable

V.7 Elementos de la Aplicación

- Acceso a la Aplicación
- Navegador
- Módulos de la Aplicación
- Hoja base
- Modo de operación
- Mantenimiento y Seguridad de la Aplicación

V.7.1 Acceso a la Aplicación

Para ingresar a la aplicación es necesario capturar la clave del usuario y la contraseña, proporcionado por el administrador del sistema.

Esta clave permite un control de seguridad en la entrada y salida de los usuarios dentro de la aplicación.

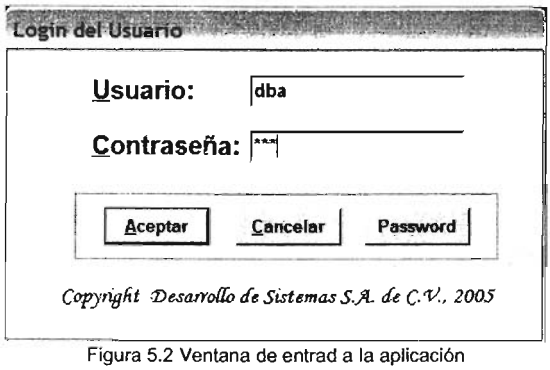


Figura 5.2 Ventana de entrad a la aplicación

Al presionar el botón <Aceptar> se realiza la conexión a la base de datos. Si alguno de los campos es erróneo o no se encuentran registrados en la transacción de usuarios, se indicará el error: "Usuario/Contraseña Erróneos". Si se desea cancelar la conexión se debe presionar el botón <Cancelar>.

En caso de ser correctos los datos del usuario y contraseña, se le dará acceso a la pantalla del "Navegador", de lo contrario, se desplegará un bloque de ERROR con el mensaje "El usuario no tiene acceso a la aplicación".

V.7.2 Navegador

Es el bloque que nos permite acceder los diferentes componentes de la aplicación. Se encuentra dividido en dos partes; el lado izquierdo, contiene los nombres de los módulos que integran la aplicación (el módulo que esta sombreado es el que se

encuentra seleccionado), y el lado derecho, contiene los submenús del módulo seleccionado así como las ventanas de cada submenú.

La aplicación opera a nivel de ventanas, para seleccionar alguna de ellas, se utiliza el ratón, haciendo doble click, para pasar a la ventana seleccionada.

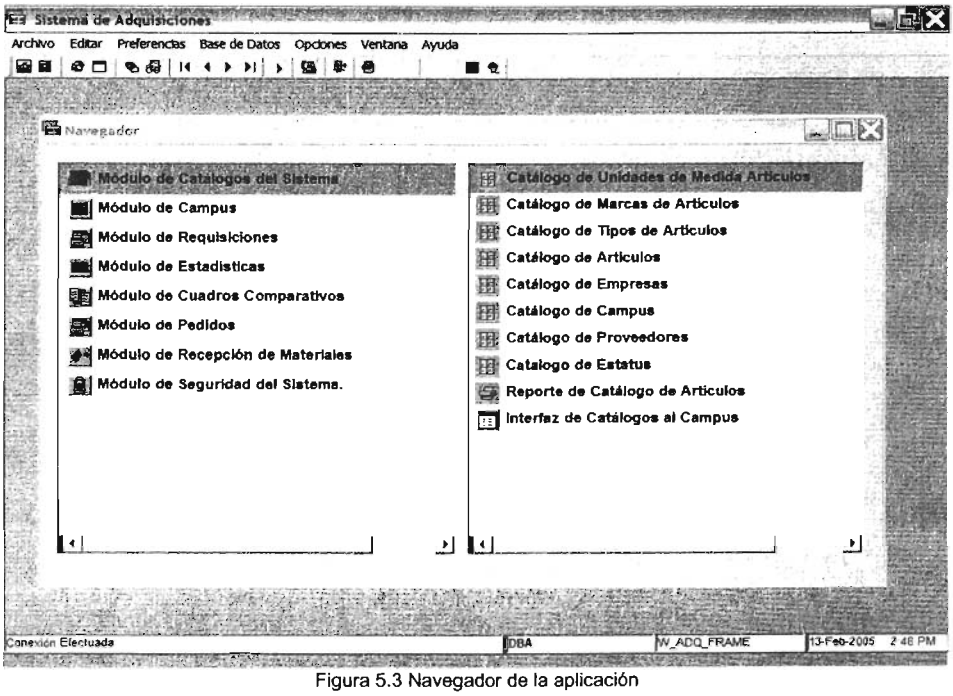


Figura 5.3 Navegador de la aplicación

V.7.3 Módulos de la Aplicación

V.7.3.1 Módulos de Catálogos

A través de este módulo se dará mantenimiento (altas, bajas, modificaciones y consultas) de los datos que se utilizarán en la aplicación durante su manejo para el control de las adquisiciones de acuerdo a las requisiciones solicitadas.

Uno de los objetivos de este módulo es evitar reescribir el mismo dato varias veces evitando la duplicidad de la información.

V.7.3.2 Módulo de Campus

En este módulo se podrán hacer la captura de las requisiciones de los Campus, cargar en el sistema los pedidos o requisiciones, además de tener una interfaz del Campus de la compañía para la transmisión de pedidos.

V.7.3.3 Módulo de Requisiciones

Dentro de este módulo se carga a la aplicación todas las requisiciones que la compañía recibe de los Campus para su adquisición y permite asignarle compradores a cada requisición, también genera reportes de las mismas y permite una interfaz de la compañía con los Campus.

V.7.3.4 Módulo de Estadísticas

A través de este módulo se pueden generar gráficas en las que se muestran las estadísticas de las requisiciones y de las partidas que cada comprador tiene asignadas, así como, por cada Campus deseado de acuerdo al estatus correspondiente a cada una de ellas.

En resumen, este módulo permite al usuario tener un control ilustrado sobre las requisiciones y partidas de cada Campus y por comprador.

V.7.3.5 Módulo de Cuadros Comparativos

La finalidad de este módulo es generar un cuadro donde se puedan comparar los precios de los proveedores de acuerdo al artículo solicitado para poder elegir el más conveniente. De estos cuadros comparativos se puede obtener un reporte impreso.

V.7.3.6 Módulo de Pedidos

A través de este módulo se pueden generar los pedidos, teniendo una opción de consulta y generación de un reporte de los mismos.

Además permite tener una interfaz de la Compañía al Campus, lo cual facilita el proceso de transmisión de los pedidos.

Este módulo cuenta con una opción de generación, mantenimiento (altas, bajas, modificaciones y consultas) e impresión de pedidos de caja chica para controlar los pedidos más simples.

V.7.3.7 Módulo de Recepción de Materiales

A través de este módulo se tiene un control sobre la recepción de materiales de cada Campus o de la compañía; además, permite generar un reporte de esas recepciones, así como tener una interfaz en donde la aplicación indica que un pedido determinado ya fue entregado.

V.7.4 Hoja Base

Esta ventana contiene una descripción general de las partes que conforman las ventanas de manera general y contiene las siguientes partes:

- Menú de la aplicación
- Barra de herramientas
- Línea de ayuda

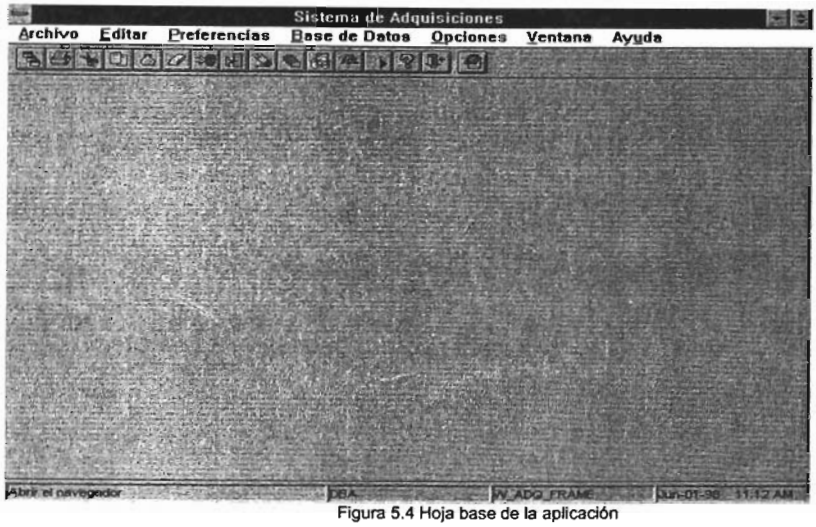


Figura 5.4 Hoja base de la aplicación

V.7.4.1 Menú de la Aplicación

Contiene todas las opciones generales del sistema, algunas de ellas también pueden ser accesadas a través del uso del teclado.

Las opciones que conforman el menú de la aplicación son:

- Archivo
- Editar
- Preferencias
- Base de datos
- Opciones
- Ventana
- Ayuda

V.7.4.1.1 Archivo

- **Navegador:** En esta ventana están registrados los módulos y transacciones de la aplicación, como se mencionó previamente. (Presionar Control N).

- **Cerrar:** Cierra la hoja activa.
- **Conectar a base de datos:** Proporciona la opción de conectarse a otra base de datos definida previamente, de acuerdo al ODBC.
- **Salvar:** Salva la información que contiene la hoja activa o ejecuta la presentación preliminar de un reporte (Presionar Control S).
- **Salvar Como.-** Los registros contenidos en una hoja pueden ser almacenados en diferentes formatos.
- **Imprimir.-** Envía al administrador de impresión, la información contenida en la hoja activa.
- **Configurar Impresora.-** Ejecuta el diálogo de configuración de impresora del manejador instalado. (Presionar control +M).
- **Salir.-** Salir de la aplicación. (Presionar alt + F4).

V.7.4.1.2 Editar

- **Deshacer:** Deshace la última edición hecha en una hoja. (Presionar control + Z).
- **Cortar:** Después de seleccionar algún dato de la hoja, lo corta; es decir, lo borra y almacena para su uso a futuro inmediato (Presionar Control+X).
- **Copiar:** Después de seleccionar algún dato de la hoja, lo almacena para su uso en un futuro inmediato. Se presiona copiar en el lugar que se desee (Presionar Control+C).
- **Pegar:** Después de haber copiado o cortado algún dato, trae la información almacenada al campo. Se presiona pegar en el lugar donde se desea insertar dicha información (Presiona Control+V).
- **Limpiar:** Limpia la información contenida en la ventana activa.

V.7.4.1.3 Preferencias

- **Mantenimiento a requisiciones:** Despliega la ventana de mantenimiento a requisiciones para poder efectuar las altas, bajas, modificaciones o consultas.

V.7.4.1.4 Base de Datos

Reúne las opciones necesarias para interactuar con la base de datos.

- **Insertar:** Inserta un registro nuevo en la hoja activa (Presionar Control+I).
- **Borrar:** Elimina el registro actual de la hoja activa (Presionar Control+B).
- **Duplicar:** Crea una copia del registro actual (Presionar Control+D).
- **Previo:** Lleva el cursor del registro previo al actual.
- **Siguiente:** Coloca el cursor en el registro siguiente al actual.
- **Primero:** Coloca el cursor en el primer registro de la hoja (Presionar Control+P).
- **Último:** Coloca el cursor en el último registro de la hoja (Presionar Control+U).
- **Ordenar:** Ordena la información de acuerdo al criterio impuesto en la caja de diálogo de búsqueda (sort). Para indicar los campos que actuarán en el criterio de orden, basta con arrastrarlos con el ratón hacia el recuadro de columnas e indicar si la búsqueda es en forma descendente o ascendente (figura 5.5).

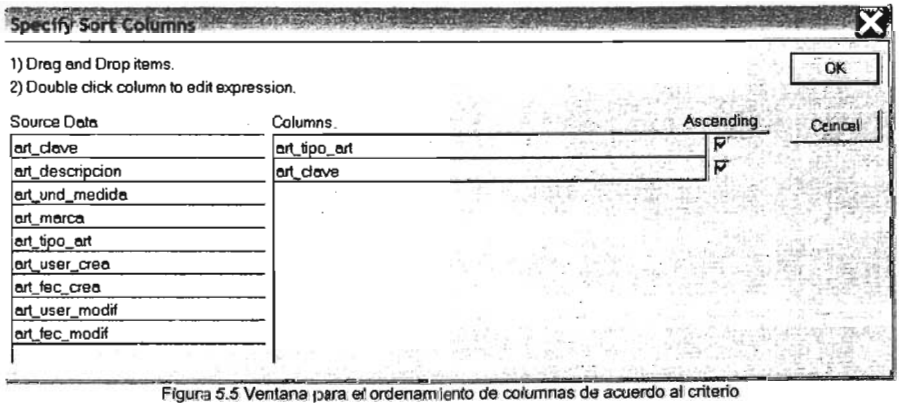


Figura 5.5 Ventana para el ordenamiento de columnas de acuerdo al criterio

- **Filtrar:** Aplica un filtrado de información por medio del criterio indicado en la caja de diálogo de filtro. Para indicar el filtrado, basta con seleccionar las columnas deseadas en combinación con los operadores y funciones disponibles de Power Builder (figura 5.6).

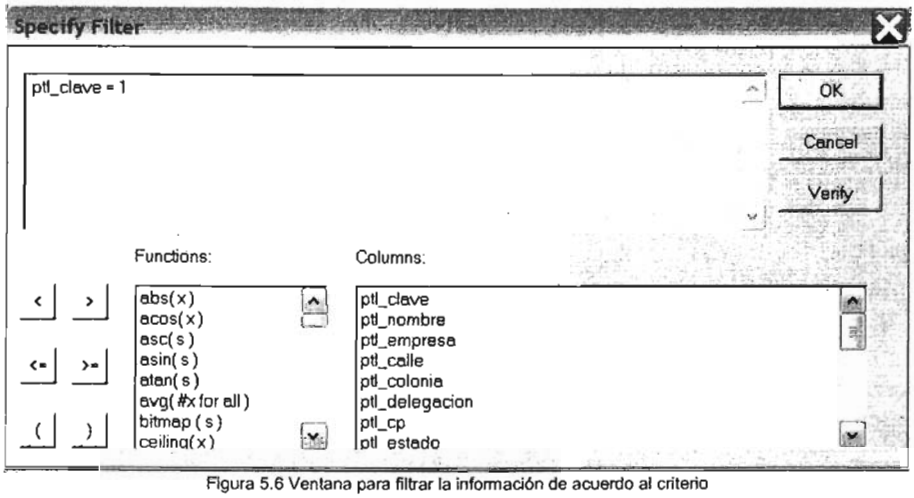


Figura 5.6 Ventana para filtrar la información de acuerdo al criterio

- **Introducir Criterio:** Pone la hoja activa con registros en blanco para introducir criterios con los operadores y funciones de Power Builder
- **Ejecutar Consulta:** Ejecuta la consulta de los registros existentes en la base de datos. Si antes se ha especificado un criterio de consulta, filtra la información en base al mismo.

V.7.4.1.5 Opciones

- **Barra de herramientas:** Muestra la caja de diálogo para definir la configuración de la barra de herramientas. Es posible moverla, esconderla o cambiar el tamaño de los botones (figura 5.7).

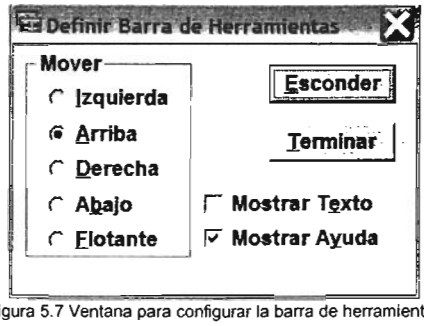


Figura 5.7 Ventana para configurar la barra de herramientas

- **Procesos preferentes:** Ejecuta la transacción de alta de preferencias, descrita en el módulo de Seguridad del Sistema.

V.7.4.1.6 Ventana

Este submenú cambia dinámicamente conforme se abren o cierran ventanas para mostrar las que existen abiertas actualmente y marcar la activa en ese momento.

- **Mosaico vertical:** Muestra un mosaico vertical con las hojas abiertas para interactuar con ellas al mismo tiempo.
- **Mosaico Horizontal:** Muestra un mosaico horizontal con las hojas abiertas para interactuar con ellas al mismo tiempo.
- **Cascada:** Muestra las hojas abiertas en forma de cascada. Es útil cuando se desea ver todas las hojas con las que se está trabajando.
- **Original:** Muestra las hojas abiertas en su tamaño original.
- **Organizar Iconos:** Cuando las hojas abiertas están minimizadas y en desorden, ésta opción organiza los iconos de las hojas en la parte inferior del área de trabajo.

V.7.4.1.7 Ayuda

- **Índice:** Ejecuta el índice de la ayuda de la aplicación.
- **Buscar:** Ejecuta la búsqueda de la ayuda de la aplicación para buscar por temas.

- **Acerca:** Ejecuta la ventana que contiene los datos del autor de la aplicación.

V.7.4.2 Barra de Herramientas

- **La barra de herramientas** es una serie de botones que ejecutan diferentes acciones propias de la aplicación. Cada una de estas acciones tiene su equivalencia en las opciones del menú descritas anteriormente.

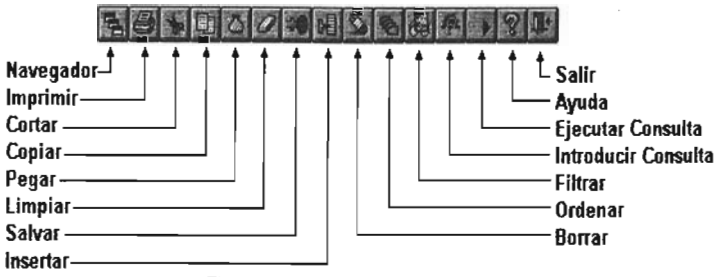


Figura 5.8 La barra de herramientas

V.7.4.3 Línea de Ayuda

- **Línea de mensajes:** Contiene una ayuda sencilla acerca del menú, de la ventana, así como mensajes de ayuda para la carga de datos en las ventanas.
- **Usuario:** Indica el nombre del usuario que se encuentra registrado en ese momento.
- **Nombre de la ventana:** Contiene el nombre de la ventana que se encuentra activa en ese momento.
- **Fecha:** Indica el día, mes y año, así como la hora en la que se está utilizando esa ventana.

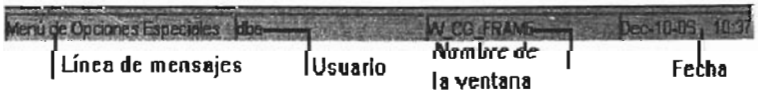


Figura 5.9 Línea de ayuda

V.7.5 Modo de Operación

- **Altas:** Presionar el botón <Insertar>  , el cual mostrará una ventana limpia en la que se podrá introducir los datos necesarios para dar la alta dentro de la aplicación.
- **Bajas:** Presionar el botón de <Borrar>  , el cual eliminará el registro existente en la base de datos.
- **Modificaciones:** Presionar el botón de <Consultas>  y al aparecer la ventana, se podrá realizar los cambios. Presionar el botón de <Salvar>  , para guardar los cambios realizados.
- **Consultas:** Presionar el botón de <Consultas>  y aparecerá lo que se haya dado de alta previamente.
- **Salvar:** Presionar el botón de <Salvar>  , para guardar los cambios realizados.
- **Búsquedas:** Para la búsqueda de un(os) registro(s) en especial se debe presionar el botón de <Criterio>  en el cual se podrá dar la condición que se desea, en los campos que sea permitido y después presionar el botón de <Consulta>  el cual recuperará la información que cumpla con las condiciones especificadas.

V.7.6 Mantenimiento y Seguridad de la Aplicación

Este módulo permite gestionar el mantenimiento de la aplicación desde la adición de nuevos módulos, la construcción del navegador (orden de aparición de módulos y ventanas con sus iconos asociados), implementación de los niveles de de acceso a los módulos y ventanas de la aplicación, hasta la creación de cuentas de usuario con sus respectivos niveles de acceso. Este módulo esta conformado por:

- Procesos
- Accesos
- Usuarios

Procesos: Su propósito es agregar nuevas ventanas a la aplicación a través de la interfaz de mantenimiento de procesos (figura 5.10) donde se captura el número, la descripción y el tipo de proceso (módulo, submenú o ventana), el nombre de la ventana asociada, el número del módulo al que pertenece, el número secuencial para el orden de aparición en el navegador y el icono (imagen que aparece en el navegador).

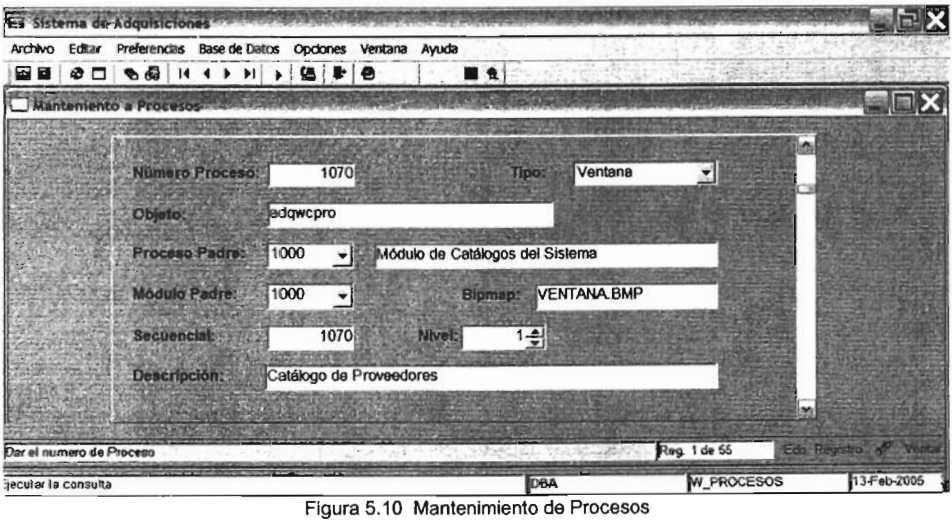


Figura 5.10 Mantenimiento de Procesos

Acceso. El objetivo es generar los niveles de acceso de la aplicación (previamente definidos por el administrador del sistema) con los permisos correspondientes: consulta de datos, actualización de datos, alta de la información y borrado de registros (figura 5.11).

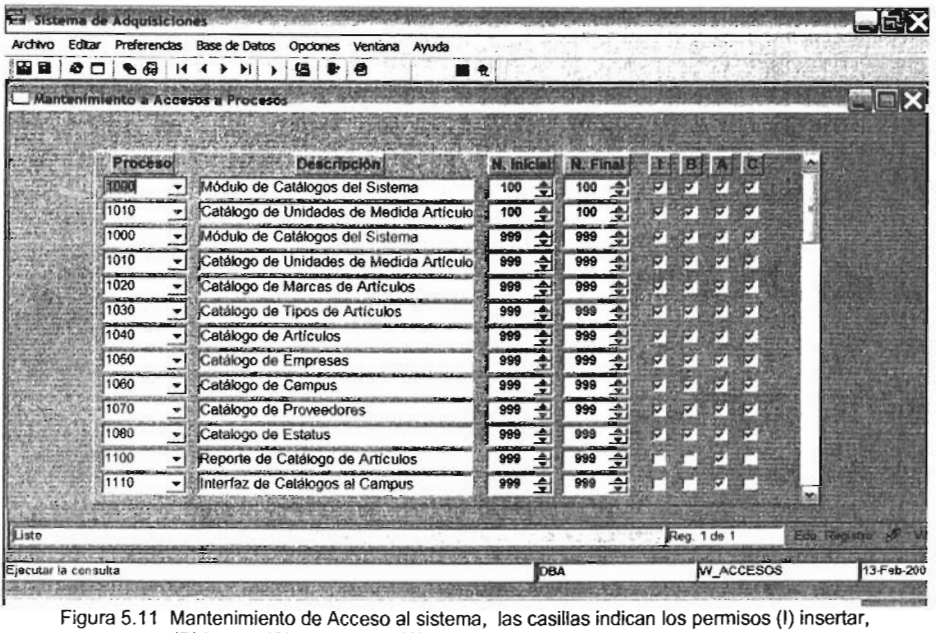


Figura 5.11 Mantenimiento de Acceso al sistema, las casillas indican los permisos (I) insertar, (B) borrar, (A) actualizar y (C) consultar

Usuarios. Sirve para dar de alta las cuentas de los usuarios en la aplicación, identificándolos con la clave de usuario, contraseña, nombre completo del usuario, fecha de expiración de la cuenta, el nivel de acceso que tiene los usuarios y los permisos para ejecutar operaciones (autorizar requisiciones, solicitudes y asignar compradores) (figura 5.12).

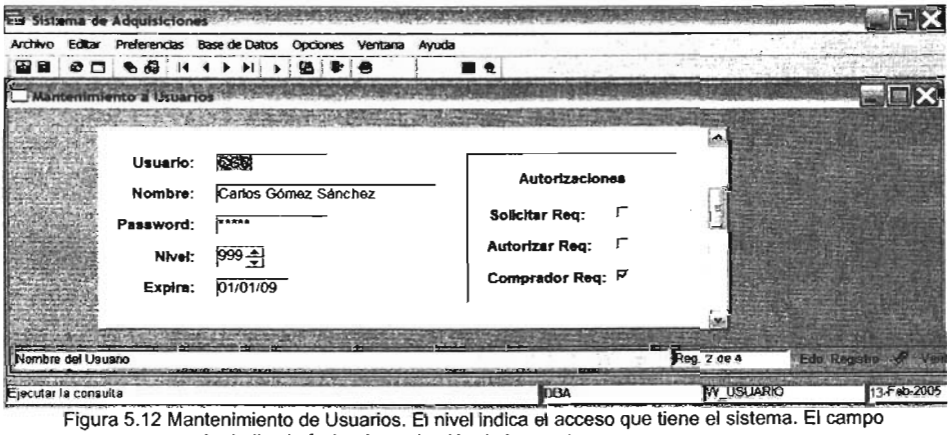


Figura 5.12 Mantenimiento de Usuarios. El nivel indica el acceso que tiene el sistema. El campo expira indica la fecha de expiración de la cuenta.

CONCLUSIONES

La tecnología de la orientación a objetos resulta de gran ayuda para resolver los problemas de modelar el mundo real a través de sus conceptos, y como un paradigma, cambia la forma de pensar de los programadores en el sentido de resolver un problema con un pensamiento más cercano al problema y no con un lenguaje basado en una programación particular.

La aplicación orientada a objetos **"Marco de Referencia"** es una opción para las empresas y los desarrollares, ya que por un lado, el cliente exige que sus aplicaciones sean creadas lo más rápido posible, que requieran un mínimo de mantenimiento, y que sean flexibles a futuros cambios. Por otro lado, proporciona muchas ventajas para el desarrollador:

- Un código reutilizable.
- Programación consistente y estandarizada.
- Lleva a sistemas más flexibles al cambio.
- Reduce el riesgo de desarrollo.
- Resulta atractivo al funcionamiento de la mente humana.

El desarrollo de la infraestructura fue, hasta cierto punto, fácil de implementar en el entorno de desarrollo de Power Builder, ya que este provee de los elementos necesarios, tales como: los objetos visuales (ventanas, botones, objetos de usuario), objetos transaccionales (que se utilizan para el manejo de base de datos) y el mecanismo de herencia; permitiendo la programación de una aplicación potente, sin la necesidad de escribir mucho código, y por lo tanto, la generación de aplicaciones reutilizables pues permite realizar cambios significativos o ampliar más funcionalidades sin afectar el rendimiento de la aplicación.

En consecuencia las aplicaciones finales se generan con una mayor rapidez dado

que muchos de los procesos se heredan, o bien, tienen pocos cambios permitiendo a los programadores generar código consistente y estándar sin tener que conocer como fue implementada la infraestructura.

Una desventaja de Power Builder es que genera archivos ejecutables pesados, porque tiene que copiar todos los archivos con extensión pbd en el directorio donde se encuentran el archivo exe. Power Builder esta diseñado para genera aplicaciones de dos capas, multicapas y aplicaciones para Internet, por lo tanto queda limitado a desarrollar aplicaciones, como por ejemplo, el manejo de sistemas operativos o gráficos avanzados, tal como lo hace como C++ o Java.

BIBLIOGRAFIA

Object-Oriented Modeling and Design
James Rumbaugh
Michal Blaha
Willim Premerlani
Prentice hall
Primera Edición, 1999

Analisis y diseño orientado a objetos Con aplicaciones
Grady Booch
Addison-wesley
Segunda Edición,1994

Object Technology in Application Development
Daniel Tkach
Richard Puttick
The Benjamin/Cummings Publishing Company, Inc.
First Edition, 1994

Power Builder
Desarrollo de aplicaciones cliente/servidor
Paul Mahler
Prentice hall
Primera Edición, 1998

Power Builder UNLEASHED
Gallagher & Herbert
Sams publishing
Second Edition, 1996

The C++ Programming Language
Bjarne Struoustrup
Addison-wesley
Second Edition, 1992

Manual de Referencia de ASP.NET
Matthew MacDonald
MacGrawHill
Primera Edición, 2002