ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Conjunto de aplicaciones avanzadas para gestión del servicio de comidas de un restaurante

AUTOR: Antonio T. Marsilla Fernández
DIRECTOR: Francisco Monzó Sánchez

09/2006

Autor	ANTONIO T. MARSILLA FERNÁNDEZ
E-mail del autor	marsillailla@hotmail.com
Director	FRANCISCO MONZÓ SÁNCHEZ
e-mail del Director	Francisco.Monzo@upct.es
Título del PFC	Conjunto de aplicaciones avanzadas para gestión del servicio de comidas de un restaurante.
Resumen	Conjunto de aplicaciones multimedia y telemáticas, para gestionar el servicio de comidas de un restaurante, mediante aplicación de marcas temporales, además de gestionar las comunicaciones entre usuarios. Se compone de 3 aplicaciones principales (camarero, barra y cocina) ejecutándose en diferentes máquinas; más un conjunto de elementos complementarios.
Titulación	Ingeniero Técnico de Telecomunicación, especialidad Telemática
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Septiembre 2006

Índice General

Capítulo 1

Introducción

1.1 Estructura del contenido

Capítulo 1 *Introducción:* En este primer capítulo se fijan los objetivos de la realización del proyecto, y se muestran cuales son las motivaciones y la justificación de la realización del mismo. Además se hace un breve estudio sobre los antecedentes existentes en el mercado.

Capítulo 2 *Ámbito de aplicación* y *rango de soluciones del proyecto:* En este capítulo se realiza un estudio de viabilidad del proyecto a realizar, así como las distintas posibilidades que existen en entornos de programación y que soluciones son las más acertadas para llevar a cabo el proyecto.

Capítulo 3 *Análisis de entorno y requisitos del sistema:* Se estudian cuales son las posibilidades de ejecución de la aplicación, se describen a los usuarios a los que va destinada, los posibles escenarios de uso y los requisitos de red, hardware y software que son necesarios para su correcta ejecución.

Capítulo 4 *Bases de datos en Delphi:* Se da un brevísimo repaso a los fundamentos de datos, y una visión muy general sobre qué herramientas nos proporciona Delphi para su manejo y, en especial, se ven las distintas posibilidades que Delphi ha implementado. Con toda esta información podremos justificar la elección de los componentes y de los tipos de tablas más adecuados para nuestra aplicación.

Capítulo 5 *Aplicación desarrollada:* En este capítulo se describe cómo se ha desarrollado la plataforma software de comunicaciones, describiendo la implementación y el funcionamiento de cada uno de sus servicios.

Capítulo 6 *Elementos complementarios:* Las tres principales aplicaciones se sirven de una base de datos compuesta por diferentes tablas, y con un programa para modificarlas. Aquí se explica la estructura y el formato de las tablas y el programa implementado para editarlas y actualizarlas.

Capítulo 7 *Protocolo UDP* y *TCP:* Se realiza una breve descripción de los protocolos de red UDP y TCP, que han sido usados para el desarrollo de la aplicación y, también, asentar las bases teóricas para así poder entender de manera más fácil y práctica el protocolo de la aplicación.

Capítulo 8 *Protocolo de la aplicación:* La finalidad de este capítulo es describir el funcionamiento y estructura de los mensajes utilizados por la plataforma de comunicaciones desarrollada y dar también una visión general del comportamiento del sistema.

Capítulo 9 *Componentes:* Para la realización del proyecto, se precisaban soluciones pequeñas y específicas, muchas de ellas no existentes aún en el mercado, y otras sí, pero que han precisado pequeños retoques o cambios. Por todo ello se han

desarrollado una serie de componentes para Delphi, y en este capítulo se explica su implementación y funcionamiento.

Capítulo 10 *Conclusiones* y *retos futuros:* Finalmente, en este capítulo, se hace una evaluación de los resultados conseguidos con el proyecto, se describen las conclusiones finales obtenidas, realizando un breve resumen del proceso seguido en la realización del mismo y, por último, también se habla de los retos o nuevos cambios que se deben acometer en el futuro para seguir el desarrollo de la aplicación y perfeccionarla.

1.2. Objetivos

El principal objetivo planteado en el proyecto es el siguiente:
Diseñar y desarrollar una plataforma software para gestión de un restaurante, bar, centro de comidas rápidas, o establecimiento/s similar/es, capaz de organizar los pedidos al cociner@ o persona/s encargada/s de la preparación, mediante marcas temporales que le indiquen a dichos Sres/as el tiempo del que disponen para cada plato, el orden en el que deben realizarlos y/o servirlos, etc.

Además la aplicación debe ser también capaz de ayudar a camareros en la recogida de pedidos, y en las comunicaciones entre ellos y con la cocina. Todo ello sirviéndose de la utilización de una red wireless para una perfecta movilidad de los usuarios. Además son cuanto menos deseables algunas de estas características:

- Gestionar una base de datos con suficiente información sobre todos los productos que el restaurante dispone y ofrece a los clientes.

-Sistema muy visual y fácil de usar. El esfuerzo que hacen restauradores para integran los equipos electrónicos en su trabajo, no debe verse agravado por dificultades como programas demasiado complejos o que requieran de un largo aprendizaje.

- Debe ser totalmente práctico. Se necesita que las labores cotidianas que desempeñan camareros y cocineros sean recogidas también en el software. Por ejemplo estamos hablando de que los clientes suelen hacer variaciones en los platos, que deben poder gestionar también la aplicación. Inútil sería si después de realizar un pedido, se debiera acercar a cocina para aclararle que lo que pone en pantalla no es realmente lo que ha pedido.

- Sistema totalmente automatizado en cuanto a variaciones en la carta de productos, ya que no se les puede pedir a los usuarios de la aplicación que modifiquen la programación, sino que el programa automáticamente cambie a la vez que lo hace la carta de productos.

- La gestión de las mesas, la evolución de los pedidos, y el estado actual de las mismas es otro punto muy interesante, que pueda verse reflejado en pantalla, para que los camareros puedan controlar y distribuirse el trabajo por las mesas. Eso facilitaría mucho el trabajo y su planificación.

1.3. Motivaciones y justificación.

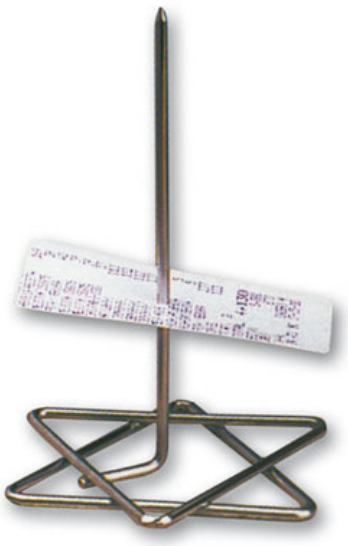
Las motivaciones que llevaron a la realización de este proyecto fueron pensadas desde un principio para facilitar la tarea de camarer@s y cociner@s de bares y restaurantes en su organización con los platos mediante equipos informáticos y redes.

La experiencia de visitar bares y restaurantes nos dice cómo se distribuyen el trabajo los operarios. Muchas veces hemos observado cómo se acumulan las notas de papel en las cocinas, en algunos casos clavadas en un pincho, donde algunas se pierden o desordenan y, en otros, los cociner@s disponen sólo de su memoria que, como la de todo ser humano, puede fallar y está limitada en cuanto a su capacidad. Por eso, con bastante frecuencia, este sistema falla, y las culpas recaen en los cociner@s, ya de por si demasiado agobiad@s con la preparación de los diferentes menús.

A veces uno se pierde leyendo el título de algunos proyectos, y no es capaz de entender siquiera el enunciado si no le afecta el problema. Sólo a los que sí les afecta o lo han estudiado en profundidad para darle solución, como en este caso, lo comprenden y es algo fácilmente observable y entendible por todos los que acudimos, con más o menos frecuencia, alguna vez a "Restaurantes, bares, o tiendas de comida rápida".

Se trataba de un proyecto multidisciplinar que ofrecía muchos puntos de interés, en el que intervenían programación y comunicaciones, para dar solución o respuesta a un problema muy común y cotidiano, nada abstracto.

Por un lado está la programación. Un problema tan práctico requería una solución real y cotidiana, lo cual es la mejor forma de aprender a programar de la forma más parecida a como se programa en aplicaciones realizadas por empresas reales.



Aplicaciones tan completas requieren emplearse a fondo en los mecanismos de programación. El hecho de crear algo práctico, que pudiera enseñarnos todas y cada una de las fases de la programación de un software real, es algo muy atractivo para cualquiera. Y si a eso le sumamos que tenemos herramientas de programación versátiles y poderosas, con muchos componentes ya implementados, y junto con equipos electrónicos muy completos, estamos hablando de que disponemos de los principios básicos para poder entender y aportar código a la mayoría de herramientas software que conocemos en el mercado.

Por otro lado la inclusión de redes nos acerca al estudio de protocolos de varias capas OSI ya que se configuran equipos de redes inalámbricas. Y además desde cero porque se necesita un protocolo específico para nuestra herramienta.

1.4. Antecedentes: Descripción de Software para restaurantes analizados.

La cantidad de programas parecidos que nos ofrece el mercado para la gestión de Restaurantes y bares es enorme. Aquí se presenta una pequeña selección de las aplicaciones software más características y versátiles, y a la vez, las que presentan mayores diferencias entre sí.

Se han estudiado las características comunes y se han escogido los puntos fuertes que presentaban cada una para aplicarlos al programa que se presenta en este proyecto. Cada característica que se repite en los programas da una pista muy favorable: es un punto que una aplicación que se precie también debe incluir. En cuanto a las diferencias entre ellos, nos hemos centrado en las más novedosas, y en los puntos flacos para intentar dar una solución, y de cada uno, junto con algunas nuevas conceptos, se ha realizado un compendio de buenas ideas para aplicar a nuestro programa.

1.4.1. AM Táctil (www.amsystem.es).

Figura 1.1 Pantalla principal del programa AM Táctil

Los nuevos sistemas de pantalla táctil aportan soluciones más eficaces al Terminal Punto de Venta en General y al de Hostelería en particular. La comunicación entre el usuario y el ordenador se hace mediante toques de pantalla. Esta característica hace innecesario el teclado y el ratón, por lo que usuarios inexpertos pueden desde el primer momento manejar fácilmente la aplicación implementada. Puede optar por dos opciones: Terminal compacto o PC con pantalla táctil.

Características principales:
» Facilidad de manejo
» MultiGestión

» Paneles de mesas configurables por grupos: Restaurante, Terraza, Barra, etc.
» Gestión de Camareros
» Emisión de comanda a impresora de cocina
» Gestión de Formas de Pago, Tarjetas de Crédito, Efectivo y Cargos a habitaciones de Hotel
» Control de Artículos y Tipos de I.V.A. aplicables
» Diferentes tarifas de precios para Barra y Mesas
» Apertura, Arqueo y Cierre de Caja
» Paneles de 42 botones, totalmente configurables por el usuario
» Informes de Ventas por: Artículos, Empleados y Familias
» Gestión de usuarios, con posibilidad de asignar o quitar permisos a procesos
» Albaranes de proveedores para entrada de mercancías
» Generación y Restauración de Copias de Seguridad
» Comprobación, reparación y compactación del estado de las bases de datos
» Requerimientos Físicos:
Memoria Mínima: 256 MB de RAM
Microprocesador: Mínimo Pentium II o superior.
Espacio Ocupado en Disco Duro: 100 MB.
Sistema operativo: Windows 98, XP, 2000 .

1.4.2. RestBar (www.restbar.com)

El programa RestBar integra las diferentes areas de control **para** su negocio, la facturación de mesas, ventas rápidas y servicio express (delivery), recetas y costos, la caja (ingresos y egresos de dinero), los inventarios de bebidas, insumos y otros, control de entradas y salidas de empleados, las cuentas por cobrar a clientes, las cuentas por pagar a proveedores, las reservaciones de clientes, la planeación de eventos (calculo de requerimientos de insumos), estadísticas mensuales varias y utilitarios. Además, de una rápida y sencilla rutina **para** dar entrada a las comandas, compatible con pantallas tactiles o "touchscreen".

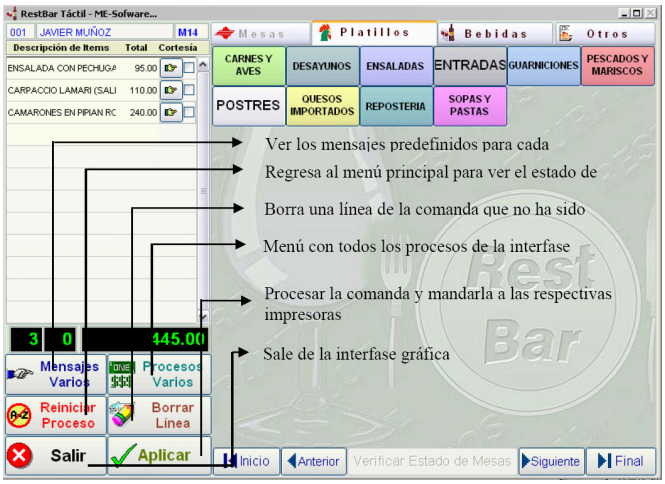


Figura 1.2 Pantalla principal del programa RestBar

Requisitos:

- Computador con procesador Celeron, Pentium IV o similar (recomendable de 1.8 Ghz o superior).
- Resolución de pantalla mínima de 800 x 600 píxeles. (*)
- Memoria RAM de 256 Mb mínimo.
- Disco duro de 20 Gb o superior @7200 RPM
- Sistema operativo Windows XP, 2000 o superior.
- Impresora POS para Tickets (similar a: Epson TMU-220D)
- Impresora POS para Comandas en Cocina/Bar (similar a: Epson TM-U300)
- Impresora de Matriz de Puntos ó Inyección de tinta. (Reportes)

1.4.3. BDP Software *(www.bdpcenter.com)*

Bdp Software es un conjunto de programas de gestión creados y desarrollados para optimizar y maximizar los recursos de la tecnología táctil, no olvidando en ningún momento la posible gestión efectuada desde teclado.

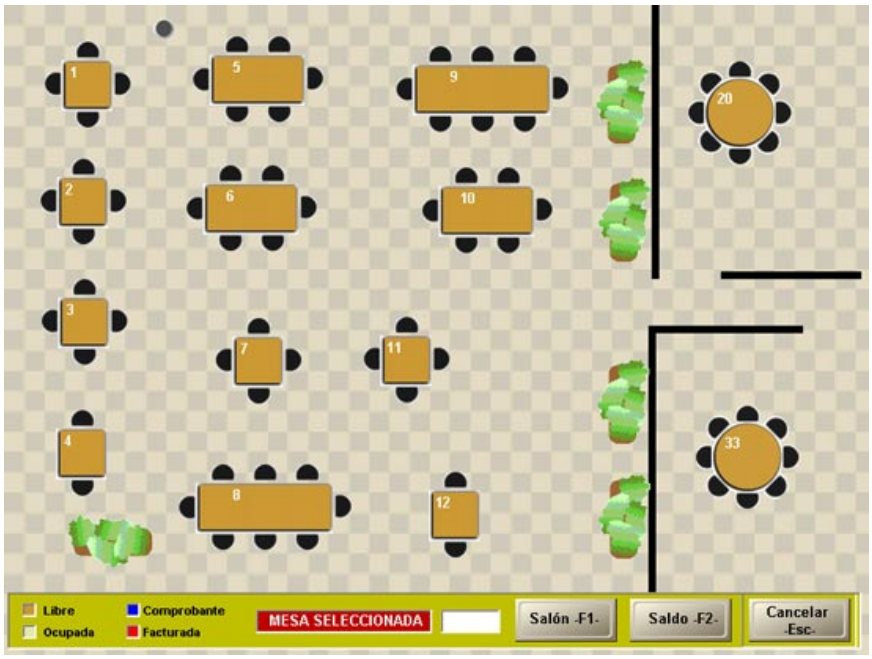
La modulación de nuestros programas, se ha efectuado precisamente para que todos los sectores dispongan de una aplicación informática que les pueda cubrir de una forma potente, rápida, eficaz y segura, todas aquellas necesidades que las empresas necesitan.

Figura 1.1 Pantalla principal del programa AM Táctil

Estas son algunas de sus principales características:

- Preparado para pantallas táctiles, teclado y ratón.
- 3 pantallas de trabajo con plus de diferentes departamentos para cada terminal
- 10 formas de pago por Terminal
- Conexión / Desconexión de la impresión de tickets
- Control de caja por fechas y turnos
- RESUMEN DE INFORMES
- Los informes se pueden obtener por locales, terminales, turnos y fechas
- Propinas
- ENLACES Y CONECTIVIDAD
- El programa permite la conexión a los siguientes elementos periféricos:
- Terminales de radio-comandas (Handy Terminal Orderman)
- Escáneres de código de barras
- Balanzas

- Tarjetas magnéticas para control de clientes
- Capturadores de datos con escáner para inventarios
- Teclados programables
- Diseño y visualización gráfica de los salones con mesas libres y ocupadas
- Hasta 6 grupos de platos por menú (entrantes, primeros, etc.)
- Control de comensales (configurable como obligatorio o voluntario)
- Entrada de comentarios por cada plato del desglose de menús
- Emisión de la factura o ticket con diferentes agrupaciones de cargos
- Servicio a Domicilio
- Pubs y Discotecas.



Capítulo 2
Ámbito de aplicación y rango de soluciones del proyecto

2.1 Elección del entorno de desarrollo

El entorno de desarrollo es otra de las decisiones más importantes a tomar antes del comienzo real del diseño de la aplicación. Para escoger el entorno de desarrollo final se tomaron en cuenta varios sistemas diferentes:

• **Visual Basic [8]:**

Se trata de un entorno de desarrollo implementado por Microsoft que destaca por su fácil manejo, y sencillez del lenguaje de programación (Basado en BASIC). Se trata de un lenguaje interpretado, que requiere una librería que contenga el motor de interpretación del lenguaje. Este es un lenguaje que genera ejecutables lentos y grandes, aunque su facilidad, hace que la programación en él sea muy rápida. Su sistema de empaquetado y distribución resulta muy fácil de usar y es muy cómodo, aunque resulta muy poco eficiente.

Visual Basic no soporta realmente programación orientada a objetos, proporcionando un substitutivo agregando a la clase hija, una instancia del padre. Es perfecto para usar bases de datos Access.

• **Jbuilder [9]:**

Este entorno de desarrollo esta implementado por Borland y destaca por su apuesta por el lenguaje de programación JAVA. Java, inicialmente ideado por SUN y que está orientado a objetos, se distingue por su potencia en aplicaciones distribuidas en Internet. Se trata de un lenguaje interpretado que usa un compilador Just-in-time alojado en una maquina virtual que ha de estar instalada en el ordenador cliente.

La filosofía Java se basa en la realización de aplicaciones cuyo código fuente se haya en la maquina servidora, mientras que la aplicación se ejecuta realmente en la maquina cliente. Esta filosofía hace fácil la distribución de la aplicación y su multilocalidad, ya que se podría ver desde cualquier máquina conectada a Internet porque además es multiplataforma. El problema de Java es su dificultad para acceder a bajo nivel, ya que no esta pensado para ello, lo que dificulta la toma de datos vitales. El entorno también ofrece mucha facilidad para documentación

• **Visual C++ [10]:**

Se trata de un lenguaje de programación orientado a objetos, efectivo y eficiente que usa toda la potencia de C++ para la programación de aplicaciones visuales en Windows. Su principal problema es que su entorno de desarrollo poco intuitivo y nada amigable ha hecho

que muchos amantes de la programación en C hallan pasado a programar en Borland builder aun siendo más eficiente el compilador de Microsoft. Otro de los grandes problemas de este lenguaje es que no ofrece su entorno en otros sistemas operativos diferentes a Windows, por su vinculación a Microsoft.

• **Borland Delphi [11]:**

Es un entorno de desarrollo sencillo y orientado a objetos basado en Pascal (Object Pascal), lo que hace que la programación en él sea también muy sencilla. La experiencia de Borland en el desarrollo de compiladores de Pascal, hace a este entorno perfectamente comparable al compilador de visual C++ en eficiencia y eficacia. Su entorno de desarrollo es amigable y fácil de usar.

Una ventaja de Delphi es que debido a su no vinculación a Microsoft tiene un entorno de desarrollo similar para linux llamado Kylix, con lo que gran parte del código creado, se podría compilar para linux, con todas las ventajas que ello conlleva.

Entorno elegido: Como decisión final se decidió por Borland Delphi, por delante de Visual C, ya que se considera bueno por lo intuitivo de su entorno, la sencillez del lenguaje y su no vinculación a Microsoft. Inicialmente se descartó Visual Basic, por su falta de eficiencia, y Jbuilder, por sus carencias en la programación a bajo nivel.

2.2 Posibles soluciones del proyecto.

Una vez se ha elegido el entorno de desarrollo de la aplicación, hay que pensar de que forma se va a diseñar la estructura y la funcionalidad de la misma, para así tener más o menos claro desde un principio que se quiere realizar y cual es la mejor forma de hacerlo.

Es en este punto donde se tienen que sopesar las distintas posibilidades que pueden existir para llegar a los objetivos de la aplicación. Hay que estudiar las opciones más interesantes y coherentes para llegar a los fines propuestos, ya que una elección errónea puede ocasionar que no se vean cumplidos algunos de los objetivos en su totalidad, e incluso, que no se pueda llegar a realizar la aplicación.

El primer punto a tratar es determinar el sistema temporal por el cual se van a organizar los pedidos en la cocina. Para ello se recopiló información acerca de qué tiempos necesitan los cocineros en tener los platos listos desde que un cliente se sienta en la mesa, hasta que se le sirve el plato. En contra de lo que se pudiera pensar, los cocineros no preparan un plato enteramente en el momento que se le pide. Es humanamente imposible tener listo un plato que necesita una hora de cocción en 5 minutos, por muchos cocineros que se pongan a trabajar.

El sistema que suelen utilizar es tener el 80% del trabajo realizado antes de que lleguen los clientes. Para ello precocinan un plato, y lo congelan o conservan en la nevera casi totalmente realizado, por lo que un plato de pasta se guardaría en la nevera semicrudo, para terminar de cocinarlo en 5 minutos. Así que en media, cualquier plato que normalmente necesita 30 minutos de preparación, realmente sólo tardarán unos 6 o 7 minutos entre que el cliente lo pide,

y ellos se lo sirven en la mesa. En el mejor de los casos ya habrán cocinado otros muchos platos que ya están preparados antes de que el cliente los pida, por lo que en menos de un minuto se le puede servir.

Por todo esto, la experiencia nos lleva a recoger toda esta información de tiempos medios de preparación de los cocineros en una base de datos, pero de forma que cada cocinero la pueda modificar y optimizar a su velocidad. Ya que no se necesita mucha precisión en los tiempos, pues es un tiempo de ayuda a la organización, no un límite rígido que haya que cumplir, se opta por escoger tiempos de medio minuto de precisión.

Otra pregunta que surge en relación con este tema es cómo evitar que el programa nos obligue a tener preparados todos los platos a la vez. Pobre del restaurante que de un golpe pusiera en la mesa de su cliente los entrantes, el primer plato, el segundo y el postre. Cuando empezara a tomar el segundo plato es seguro que se habría enfriado. Así que todo buen restaurante cuenta con un margen entre platos. Este otro tiempo también se añade en la base de datos, y estrictamente consiste en el tiempo que transcurre entre la hora en la que el cliente ha pedido y la hora en la que debiera servírsele ese plato.

Se ha introducido el término base de datos. Este es otro punto a discutir porque la información de todos los productos que el restaurante ofrece, es bastante amplia, y a la vez cambiante, y además deben poder acceder todos los usuarios a esta información. Dada la naturaleza de la aplicación, se trata de un sistema que se comunica a través de la red, y va a ser usado por múltiples usuarios, la primera elección sería usar una estructura cliente/servidor, o un sistema totalmente distribuido, en el que no se haga uso de ningún tipo de servidor central.

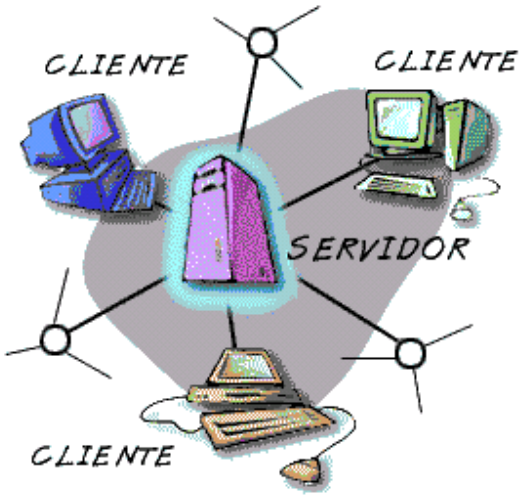


Figura 2.1 Estructura Cliente-Servidor

Las ventajas de usar una estructura cliente/servidor, en la que existiera una maquina servidor y múltiples maquinas clientes que se conectasen a ella para obtener los servicios, son que permite un mayor control sobre los usuarios y una mayor seguridad en las comunicaciones, además de ser una estructura mucho más sencilla de diseñar e implementar.

Los inconvenientes de esta estructura son que al ser un sistema centralizado, si la maquina servidor cae o deja de funcionar, la aplicación deja de funcionar también, dado que ninguno de los clientes puede comunicarse con los demás. Además, al realizar la maquina servidor todo el control de los servicios, todas las comunicaciones fluyen a través de ella, puede llegar un momento en el que al usar el sistema muchos usuarios, la maquina servidor se colapse y no pueda procesar tanto trabajo.

En cambio, implementar un sistema totalmente distribuido, puede solucionar en gran medida los inconvenientes de una estructura cliente/servidor, pero puede requerir más esfuerzo y dificultad a la hora de diseñar e implementar un sistema de estas características.

Las ventajas de un sistema totalmente distribuido son que no se hace necesario tener ninguna maquina servidor que realice el control de las comunicaciones de la aplicación, y de esta forma tener la seguridad de que los usuarios pueden comunicarse entre ellos siempre y cuando el medio físico de transporte funcione correctamente.

Los inconvenientes de una estructura de este tipo son la mayor dificultad que supone el control de los usuarios y de los servicios en la aplicación, dado que ahora el control se debe de hacer de forma distribuida entre todos los usuarios de la red.

Presentadas las ventajas e inconvenientes de las dos opciones, se decide una solución más cercana al sistema cliente servidor. Por un lado porque es más fácil, y porque los equipos móviles no tienen un medio físico de transporte lo suficientemente fiable como para confiar en un sistema totalmente distribuido. Un servidor central que contendrá las bases de datos más actuales, y los demás usuarios se conectarán a este equipo para actualizar su base de datos. No se verán entre ellos, porque para cualquier tipo de comunicación se verán las caras con el ordenador central.

Pero por otra parte cada equipo contendrá una copia de esas bases de datos, y de todos los archivos que necesite en su equipo localmente. Ya que son unidades móviles de no muy elevada capacidad de procesamiento, lo mejor es que contengan copias de las tablas de la base de datos localmente, y se conecten a la unidad central sólo para actualizarse esa base de datos y recoger información del estado de otras mesas. Haber tenido que conectarse a la base de datos del ordenador central para cada consulta, sería un gran error, aún más cuando se producen tantas consultas a la base de datos de manera constante, y sobre todo porque la base de datos de productos, se modifica muy pocas veces en el transcurso del día.

Una vez elegido el mejor diseño de comunicaciones para la aplicación, hay que estudiar cuál es la mejor forma para implementar ese diseño, sopesando las posibilidades que ofrecen los distintos protocolos de red existentes.

Dado que la aplicación va a ser usada en una red interna del tipo Fast Ethernet, lo más lógico sería usar la pila de protocolos TCP/IP, dado que son un estándar internacional y esta sobradamente demostrada su capacidad y eficacia, y por ello hay que estudiar las posibilidades que ofrece TCP/IP para realizar tales funcionalidades.

Como se van a utilizar en una red inalámbrica donde los equipos son móviles, y no siempre la comunicación estará garantizada, se debe usar TCP/IP para asegurar que los ficheros que se intercambien con el servidor central llegan a su destinatario. Pero en contra de esto, se tiene otra característica que va a ser cubierta por el protocolo UDP, y es el envío de mensajes broadcast. En ciertas ocasiones, como las notificaciones de actualización, o de otro tipo, el mensaje debe ser enviado a todas las estaciones de la red mediante broadcast. Como se ha dicho anteriormente, la red inalámbrica, y la morfología propia de un restaurante, donde no siempre está garantizada la conectividad inalámbrica, nos piden que además, utilicemos algún tipo de protocolo superior que nos asegure que estas notificaciones han llegado a todos.

Capítulo 3.

Análisis del entorno y requisitos del sistema.

3.1 Descripción de los usuarios

La aplicación está destinada a cualquier tipo de usuario, sin importar demasiado sus conocimientos en entornos informáticos o nivel cultural, dado que trabaja sobre el sistema operativo Windows, y este entorno operativo se caracteriza por su fácil interfaz con el usuario y por no necesitarse para su uso grandes conocimientos en informática. Acorde con este hecho, la aplicación ha sido diseñada para controlarla mediante una interfaz gráfica, de uso sencillo y vistoso para el usuario, por lo que la mayoría de usuarios con nivel de conocimientos bajo, medio o alto, puede usar este software sin ninguna dificultad.

3.2 Escenarios de uso

Esta aplicación ha sido diseñada principalmente para restaurantes, bares, cáterings y centros de comida rápida. Aunque, como ha pasado en muchas empresas, la aplicación podría también servir a otro tipo de comercios que no tienen por qué disponer de una cocina (alimenticios, de ocio, etc.), con sólo unas pequeñas modificaciones. Por ejemplo los bares de copas utilizan programas muy parecidos, pero bastante más simples ya que no necesitan comunicación entre equipos, y su personal está compuesto sólo por camareros, así que la aplicación camarero cumpliría sus expectativas a la perfección. Las aplicaciones de cocina y de barra en este caso no se tendrían que utilizar, y ni si quiera tienen que disponer de una red de comunicaciones. Por tanto, no sólo se restringe a los centros de comida que al principio de este párrafo se mencionaron, sino que abarca más centros de ventas como heladerías, cafeterías, terrazas, pubs…

Diariamente vemos como la tecnología nos crea unas necesidades que hace pocos años ni teníamos, ni imaginábamos. Un caso claro es el teléfono móvil sin el que ahora parece imposible la vida, pero que hace muy pocos años nadie tenía e increíblemente todos ¡sobrevivían! Pues por esto se puede ver que también es una buena oportunidad de negocio crearles a los clientes una nueva necesidad, y ofrecer un servicio novedoso del que, dentro de no mucho tiempo, quizás también sean dependientes. Todo esto aprovechando el tirón que la tecnología está teniendo en nuestra vida cotidiana.

Se está hablando aquí de dos puntos fuertes de marketing con el que debería contar este proyecto y que amplía el número de empresas que pueden utilizar este proyecto. Por un lado una futura versión debería crear una página Web para que los clientes pudieran reservar mesa a una hora determinada, pedir el menú, y servírselo a domicilio, etc. Se amplía así entonces el rango de aplicación a empresas de comida con servicio a domicilio.

La otra oportunidad, que ya tiene éxito en Japón, es que los clientes dispongan de una Tablet PC (o PDAs o equipos integrados directamente en las mesas), para que ellos

mismos puedan ver la carta, las fotos de los platos, e incluso los ingredientes. ¿No pasa a menudo que cuando acudes a un restaurante te apetece escoger la comida más que por su nombre, por sus ingredientes? ¿O nunca os habéis llevado una decepción cuando pediste un plato y finalmente no era lo que te imaginabas?

Figura 3.1 Cliente de un restaurante pidiendo con una Tablet PC

Por tanto, se puede concluir que tanto restaurantes, bares, cáterings, centros de comida rápida, etc. pueden sacarle mucho partido a esta tecnología, como un medio de ayuda a sus propios trabajadores; pero también, como un nuevo frente de negocio, que seguro atraerá mucha más clientela.

3.3. Topología de la Red

La red utilizada debe contener al menos tecnología inalámbrica que soporte los protocolos TCP/IP, para la parte móvil de la red (camareros). Dos buenas posibles soluciones son la tecnología wi-fi y bluetooth, ya que están muy extendidas en PDAs, móviles, y equipos informáticos. La tecnología para la comunicación entre cocina y la barra puede ser una red LAN, formada por un cable cruzado de ethernet de equipo a equipo. Esta segunda tecnología es rápida, fiable y simple, pero podría utilizarse otra cualquiera que soporte los protocolos TCP/IP.

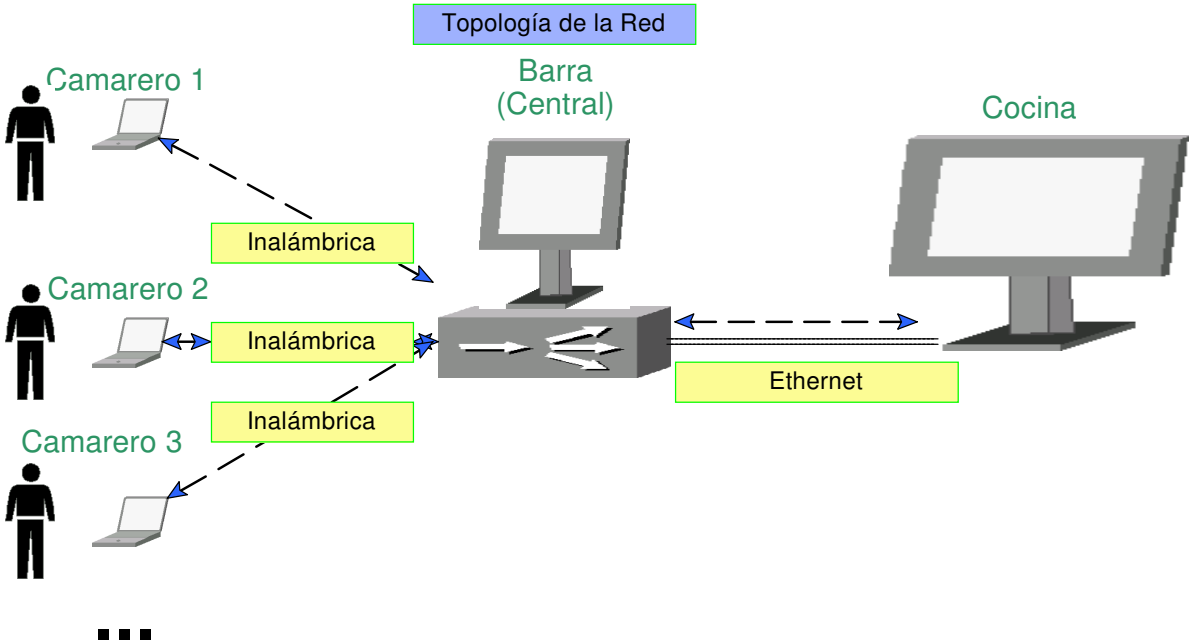


Figura 3.2 Topología de la Red

3.4. Análisis de Requisitos

3.4.1. Requisitos Hardware.

Son necesarios los siguientes componentes hardware y equipos informáticos:

- 1 PDA, tablet PC, o equipo informático de pequeño tamaño (para muy buena movilidad), con tecnología de comunicaciones inalámbrica, por cada camarero.
- 2 PCs para Barra y Cocina con tarjeta de Ethernet
- 2 Pantallas táctiles para Barra y Cocina.
- 1 Tarjeta inalámbrica para el equipo de la barra, con la misma tecnología inalámbrica que la utilizada en las PDAs.

3.4.2 Requisitos Software

La aplicación necesita usarse bajo un sistema operativo Windows, cualquiera de ellos. Ya existen compiladores para sistemas Unix, así que es más que probable que en próximas versiones también corra en cualquiera de estos sistemas operativos.

Para una correcta visualización será necesaria una configuración de más de 256 colores a una resolución como mínimo de 800 x 600 en el caso de los equipos de la barra y de la cocina. Los equipos de los camareros, debido a la multitud de tecnologías que podemos escoger (móviles, PDAs, pocket PCs), variarán su resolución mínima en función del equipo utilizado.

Capítulo 4.

Fundamentos de Bases de Datos en Delphi

4.1. Introducción

Este es el momento apropiado para presentar a uno de los protagonistas de este proyecto: las bases de datos y sus sistemas de gestión, con los que se ha intentado trabajar. Cuando el proyecto estaba en la fase de requerimientos, en la cual las ideas y los requisitos mínimos que se le iban a pedir al programa estaban claros, me pregunté: ¿qué lenguaje debo utilizar para almacenar tanta información? Mas bien la pregunta correcta debió ser: ¿qué sistema de bases de datos es el más apropiado para mis necesidades?

En este capítulo se recordarán los principios básicos de los sistemas de bases de datos relacionales, y se realizará una rápida introducción a algunos sistemas concretos con los que Delphi puede trabajar de modo directo. Y en especial, se realizará un recorrido por los componentes de bases de datos que Delphi ha implementado. Los componentes de Delphi, una vez que se conocen, le facilitan mucho la labor al programador, ya que entonces, es más un trabajo de configuración que de programación.

4.2. BDE

Borland Database Engine (BDE) es un software de 32 bits basado en Windows que es el corazón del manejo de Base de Datos y el encargado de la conexión a la misma detrás de los productos Borland. El BDE ofrece un conjunto de características robustas para asistir los desarrolladores de aplicaciones Cliente-Servidor.

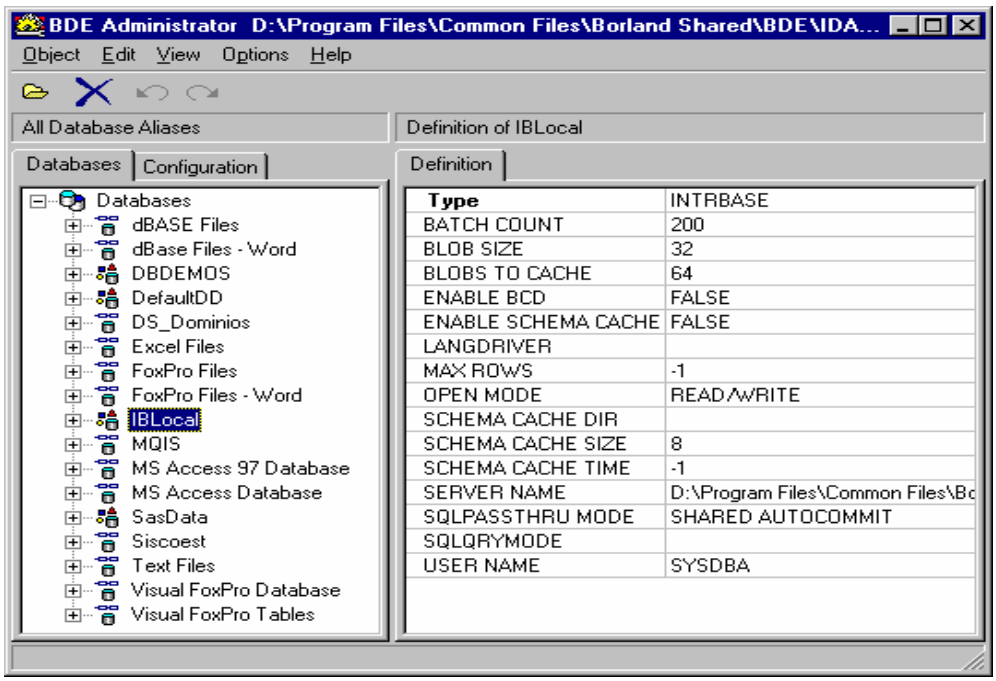


Figura 4.1 Administrador de Bases de Datos de Delphi

El manejo de las bases de datos en Delphi es responsabilidad de una parte del Delphi, diseñada exprofesamente para esa función, en concreto del Motor de Base de Datos de Borland (BDE)

Lo que se consigue con esto es la independencia casi absoluta del código fuente con el tipo de base de datos que se use. Así da igual que un programa gestione una base de datos en formato Dbase o que sea Paradox. La única preocupación del programador es teclear la orden correspondiente, que BDE se encargará de pelearse con el controlador del tipo de base de datos que use el programador.

Por defecto BDE gestiona tres tipos de bases de datos, la tres de Borland, pero eso no significa que con los controladores adecuados no pueda gestionar otros tipos, incluidos ODBC. Y si esto no es suficiente, hay controles desarrollados por terceros controlan otros tipos de bases de datos. Los tipos de bases de datos que controla BDE por defecto son: dBase, Paradox, Interbase.

Y todo esto en lo que se refiere al aspecto de aplicaciones mono puesto, pero las versiones más altas de Delphi están preparadas para comunicarse a servidores de datos, como SQL Server.

4.3. Acerca de Base de Datos y Tablas en Delphi

El término base de datos es algo muy común en nuestros días, y todos tenemos un idea básica de lo que es una base de datos, por lo menos a nivel abstracto. En un ordenador, una base de datos, no es más que un conjunto de ficheros con unas características propias que los hacen especiales. Entre estas características se puede destacar su facilidad para almacenar datos de diversos formatos en un mismo fichero, y su ordenación (si se desea).

En general, podemos utilizar el término *base de datos* como una colección de tablas. ¿Pero que es exactamente una tabla? Una tabla en una base de datos puede ser comparada con un archivo de registros en pascal. Una tabla tiene registros o filas, y un número determinado de columnas, una para cada campo del registro.

Una vez se tenga un tabla de base de datos, usted puede realizar un numero determinado de acciones, como editar valores, insertar registros, y borrar registros existentes.

Alias. Un alias no es más que un seudónimo para un directorio. En Delphi se utiliza para indicar el camino donde está la tabla con la ruta completa. Por ejemplo podemos utilizar este alias: Contabd06, para ahorrarnos el escribir una larga ruta de directorios cada vez que necesitemos escribirla:

C:\MisBases\Contabilidad\Restaurante\2006\PimerSemestre.

La utilización de alias nos proporciona una nueva posibilidad: poder cambiar la ruta a la que apunta el alias (por ejemplo cada semestre de la contabilidad), y no tener que modificar el código. Para añadir, modificar, o borrar un alias, en principio, hay que recurrir al administrador de BDE, el cual encontrarás en la carpeta donde esta instalado el Delphi, o en el caso del Delphi 3.0 en el panel de control y que lo veras con la denominación Database Desktop.

4.4. Componentes

Delphi incluye un numero de componentes relativos a *Base de Datos* .La paleta de componentes Data Access contiene componentes usados para interactuar con las base de datos. La mayoría de ellos son componentes no visuales, ya que ellos encapsulan conexiones de base de datos, tablas, queries, y elementos similares. Delphi también provee un numero predefinido de componentes usted puede utilizar para visualizar y editar los datos. En la paleta de componentes Data Controls, están los componentes visibles usados para visualizar y editar los datos en un Form. Estos controles son llamados controles *data-aware.*

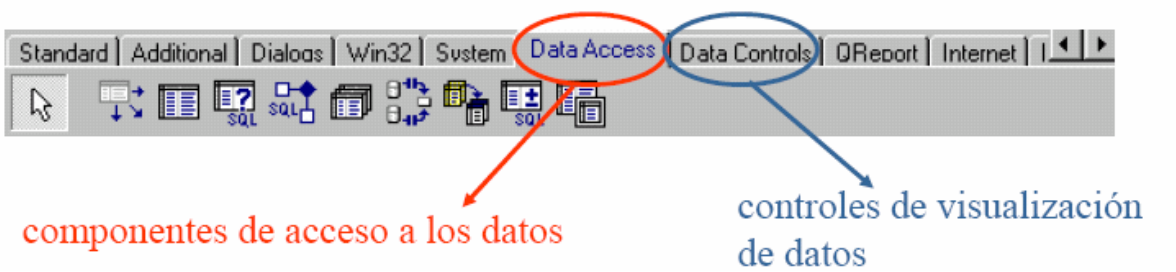
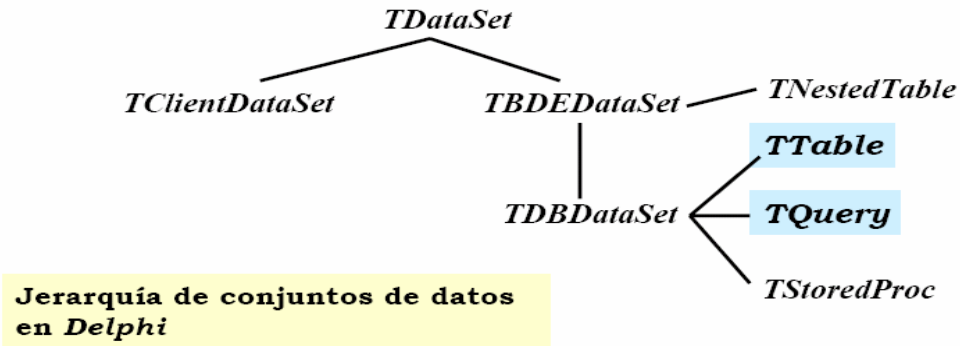


Figura 4.2 Barra de componentes de Delphi relacionados con las BDs.

4.4.1. Componentes Data Access

Los conjuntos de datos que manejan las aplicaciones de BD descienden de la clase TDataSet.

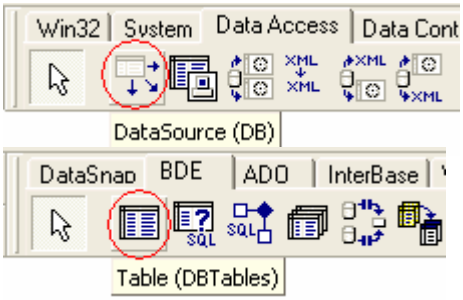


Jerarquía de conjuntos de datos en *Delphi*

Los componentes en la página de Data Access de la paleta de Componentes permiten conectarse a base de datos usando el Borland Database Engine (BDE):

	*DataSource*	Actúa como medio entre un componente Dataset como TTable y los componentes de visualización de datos como TDBGrid.
	*Table*	Toma los datos de una tabla físicamente en una base de vía el BDE y suple a uno o más de los componentes de visualización de datos a través de un componente DataSource. Convenientemente, envía data recibida del componente a la base de datos físicamente vía el BDE.
	*Query*	Utiliza comandos SQL para tomar datos de una tabla física en una base de datos y suple uno o mas componentes de visualización a través del componente TdataSource. Convenientemente, usa comando de SQL para enviar datos desde el componente a la base de datos físicamente vía el BDE.
	*Database*	Configura una conexión persistente a una base de datos, especialmente a una base de datos remota requiriendo login y password de usuario.

Los más importantes se llaman DataSource y Table respectivamente. Estos controles son del tipo no visuales, como los cuadros de dialogo, y se van a encargar de gestionar la base de datos para nosotros.



Estos dos componentes trabajan en estrecha relación. Así en la propiedad DataSet del componente DataSource debe relacionarse con un componente Table, que hay el formulario.

TDataSource. Actúa como una interfaz enlazando un componente de acceso a datos (TTable o TQuery) con un control de visualización de datos. También se utiliza para vincular tablas o consultas en formularios maestro/detalle. Sus propiedades más importante son:

-DataSet: fuente de datos que proporciona los datos de la BD (generalmente un TTable o TQuery)
- Enabled: indica si la conexión está realmente activa o no.
- AutoEdit.

El componente **Table**, tiene un montón de eventos. Esos eventos se producen antes (before), después (After) y, algunos, durante (On) las operaciones que se pueden hacer con una base de datos. Por cierto que Delphi las llama Tablas (Table), y de ahora en adelante me voy a referir así a las bases de datos.

En el aspecto de las propiedades hay tres que destacan, son: Active, DataBaseName, y TableName. Active, nos permite abrir y cerrar la tabla. Por otra parte las dos restantes están relacionadas entre sí.

DataBaseName, almacena el alias que vamos a usar, y TableName, el nombre de la tabla. Una vez que has puesto un alias o una ruta en DatabaseName, en TableName aparecerán todos los nombres de las tablas de ese directorio y abra que seleccionar una.

Cuando se trabaja con una base de datos, el principal responsable de prácticamente todo es el componente Table. Este componente es el encargado de leer, escribir, actualizar, etc. la base de datos a la cual esta conectado, y forma un "tándem" muy útil con el componente Datasourcer, el cual es el encargado de conectar al componente table con los controles usados para visualizar los datos.

Así el componente Table tiene una larga lista de eventos, que se pueden dividir en tres grupos, los eventos que se producen antes de una operación (apertura, cierre, lectura, etc) los cuales empiezan por la palabra Before (en inglés significa antes de); luego están los eventos que se producen en el momento cuyo nombre empieza por On (recuerda que el evento que se producia cuando se pulsaba un botón se llama Onclick); y por ultimo los eventos que se producen después, cuyo nombre empieza por After (del inglés: después).

Antes de poner una tabla con los principales eventos os voy a comentar las principales operaciones de una tabla, estas son:

Open	*Se abre la tabla*
Close	*Se cierra la tabla*
Edit	*Para editar el campo actual*
Insert	*Añade datos a la tabla*
Post	*Graba los datos*
Cancel	*Cancela la operación actual*

Así pues los principales eventos son los siguientes:

AfterDelete	*Se produce despues de el borrado de un registro*
AfterEdit	*Después de editar un registro*
AfterInsert	*Después de Insertar*
After Post	*Después de grabar los datos*
BeforeCancel	*Antes de cancelar la operación*
BeforeClose	*Antes de que se cierra la tabla*
BeforeDelete	*Antes de borrar*
BeforeEdit	*Antes de editar el registro actual*
BeforeInsert	*Antes de insertar un registro*
BeforeOpen	*Antes de abrir la tabla*
BeforePost	*Antes de guardar los datos*
OnCalcFields	*Se produce cuando se va a calcular el valor de un campo*
OnDeleteError	*Cuando se produce un error al borrar un registro*
OnEditError	*Cuando hay un error al editar un registro*
OnFilterRecord	*Cuando se activa el filtrado*
OnNewRecord	*Cuando se añade un registro*
OnPostError	*Cuando hay un error durante el grabado de datos*

Otro componente de acceso a datos en Delphi es el componente **Query**. Un query es usualmente más complejo que un *Table*, porque requiere un comando del lenguaje SQL. El componente Query tiene la propiedad *DatabaseName* como el componente *Table*, pero no tiene una propiedad *TableName*. La tabla es indicada en el comando SQL, guardado en la propiedad *SQL.*

Como ejemplo puede escribir un comando simple de SQL como sigue:
Select * from *Customers* donde *.customers* es el nombre de la tabla, y el asterisco (*) indica que cuando se quiere usar todos los campos de la tabla. Si posee buenos conocimientos de SQL, usted podría usar un componente *query* más a menudo, pero la eficiencia de la tabla o el *query* depende de la base de datos que se utilice. El componente *Table* tiende a ser más rápido en base de datos locales, mientras que el componente *Query* tiende a ser más rápido en los servidores SQL, además esto no es siempre el caso.

El tercer componente de acceso a datos es el ***StoredProc***, que se refiere a procedimientos almacenados de los servidores de base de datos SQL. Puede correr estos procedimientos y obtener resultados en un form de una tabla. Los procedimientos almacenados pueden solamente ser usados con servidores SQL.

Consideraciones importantes. Los componentes TTable y TQuery contienen un grupo de componentes Tfield que se corresponden con los campos (columnas) de la tabla. Estos TField se crean:

- automáticamente al activarse el TTable o Tquery
- durante el diseño (Fields Editor …) con el nombre la tabla o query concatenado con el nombre del campo.

4.4.2. Componentes Data Controls de visualización de datos

Los componentes en la página Data Controls de la paleta de Componentes contienen elementos especializados de controles de base de datos disponibles para las aplicaciones. Incluyen controles tradicionales para visualizar datos asociados a BDs. Son como controles estándar, pero su contenido se recupera directamente de algún campo de una tabal o incluso de una tabla completa. Todos tienen la propiedad DATASource indicando el enlace entre el control y la fuente de datos. Pero el uso de estos no garantiza la integridad de los datos.

Los componentes son:

DBGrid	*Tablero que permite ver y editar los datos tabularmente similar a una hoja electrónica.*
DBNavigator	*Botones de navegación que permite mover el puntero del registro actual hacia delante y hacia atrás. El navegador puede poner la tabla en estados de Insert, Edit o Browse, guardar o modificar registros y tomar datos actualizados para actualizar el despliegue.*
DBText	*Etiqueta que despliega el valor de un campo en el registro actual.*
DBEdit	*Caja de edición que despliega o edita un campo en el registro actual.*
DBMemo	*Caja memo para desplegar o editar textos en el registro actual.*
DBImage	*Caja de imagen despliega, corta o pega imágenes bitmaps al y desde el registro actual seleccionado.*
DBCheckBox	*Caja de marcar que despliega o edita el valor de un campo boleano desde el registro actual.*
DBRadioGroup	*Grupo de botones radio que despliega o guarda valores de columna.*

Los componentes interesantes de este grupo, para manejar una tabla son la barra de navegación (dbNavigator), el TDBGrid, y el TDBControl grid.

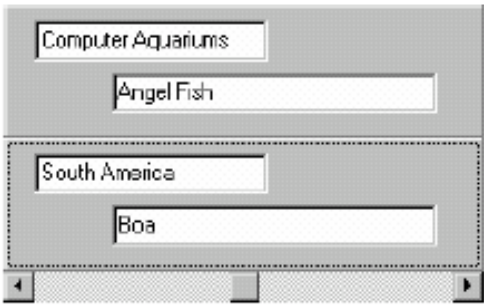
El **TDBGrid** visualiza una tabla completa de una BD que se puede modificar (todo ello en forma de tabla o rejilla).

	SIZE	AREA	WEIGHT
▶	2	Computer Aquariums	2
	10	South America	8
	30	Screen Savers	20
	10	New Orleans	5
	40	Africa and Asia	35

Sus propiedades más importantes son:

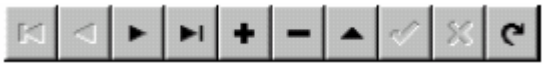
- Options, que personaliza la forma en que se aparece y se comporta el gris (editable, bordes, multiselección…)
- Con el editor de columnas (Columns Editor) se puede limitar las columnas a visualizar, modificar el orden, etc.

El **TDBControlGrid** visualiza campos de una tabla en una rejilla de controles definida por el usuario.



Sus propiedades son:
- AllowInsert, AllowDelete: si se permite insertar y/o eliminar registros.
- Orientation: si los registros aparecerán en bandas horizontales o verticales.
- RowCount y ColCount : nº de campos que se muestran simultáneamente.

 El TDBNavigator permite desplazarse por los datos de una tabla mediante el uso de botones, además de realizar operaciones de inserción, modificación, borrado,…



De sus propiedades específicas se pueden resaltar algunas. Una es la propiedad VisibleButtons la cual contiene la relación de los botones que son visibles. Por defecto los botones visibles son todos, y como quizás hayas descubierto cada uno tiene una función, de derecha a izquierda los botones que se muestran son:

nbFirst	*Desplaza la tabla al primer registro*
nbPrior	*Se desplaza al registro anterior*
nbNext	*Avanza un registro*
nbLast	*Se coloca en el último registro*
nbInsert	*Inserta un registro en la tabla*
ndDelete	*Borra el registro actual*
nbEdit	*Pone la tabla en modo edición*
nbPost	*Guarda los datos (post)*
ndCancel	*Cancela la operación actual*
nbRefresh	*Refresca los datos*

Otra propiedad es ShowHint, la cual si su valor es true muestra una breve descripción de cada botón, pero inglés. Aunque hay otra propiedad llamada Hints, del tipo TString donde podemos escribir las descripciones que queramos para cada botón. Teniendo en cuenta que la primera línea contendrá la descripción del primer botón. Y ojo porque si un botón o más no están visibles debes colocar, igualmente, la descripción del mismo. Así que un pequeño truco que yo uso es, que lo primero que hago es colocar las descripciones de todos los botones, y luego desactivo los botones que me no interesan.

Quizás a alguien le interese crearse una barra de herramientas en su aplicación y usar botones convencionales en sustitución de la barra de navegación, o simplemente ha decido poner un menú en su aplicación que haga las misma funciones que la barra de navegación. Pues la solución es colocar en el evento Onclick de los menús la orden correspondiente a la acción que queremos hacer, pero esta vez hay que indicárselo a la tabla, y no a la barra de navegación. Los métodos que hay que llamar son:

First	*Para llevar la tabla al principio*
Prior	*Para retroceder un registro*
Next	*Para avanzar un registro*
Last	*Para ir al último registro*
Edit	*Para editar el registro actual*
Insert	*Para inserta un registro*
Delete	*Borra el registro actual*
Edit	*Coloca la tabla en modo edición*
Post	*Guarda los datos*
Cancel	*Cancela la operación (inserción)*

En la pestaña DataControls de la barra de herramientas de Delphi, están los controles para manejar los datos de la tabla; son iguales que sus "homólogos" comunes, con la particularidad que contiene las dos conocidas propiedades que permiten acceder a una tabla, que son: DataSource donde indicaremos el componente TDataSource el cual está conectado a la tabla que queremos acceder, recordar que el componente TDataSource hace de "interprete" o "puente" en una tabla y los componentes; DataField, es la propiedad en la cual se indica el campo de la tabla que queremos mostrar.

4.5. Los Módulos de Datos

Los módulos de datos le permiten tener todas las reglas de base de datos y relaciones de forma centralizada para ser compartida por los diferentes proyectos, grupos y entidades de la corporación. Los módulos de datos están encapsulados en el componente TDataModule del VCL. Piense en un TDataModule como un Form invisible donde puede colocar los componentes “Data Access” que se utilizarán en el proyecto. Para crear un instancia del módulo de datos es bastante simple: Seleccione *File* | *New* del menú prinicipal y luego seleccione “Data Module” del repositorio de objetos.

La justificación simple para usar el TDataModule de poner sólo los componentes “Data Access” en un Form es que es más sencillo compartir los mismos datos a múltiples Ventanas y unidades en su proyecto. En una situación más compleja, se tendrá una gran cantidad de componentes TTable, Tquery y/o TstoredProc. Se tendrán relaciones definidas entre los componentes y además reglas a nivel de campos, como valores mínimo / máximo o formatos de despliegues.

Además este arreglo de componentes “Data Access” modela las reglas del negocio de la corporación. Después de tomar tanto tiempo para configurar esto, no se desearía tener que hacer esto nuevamente en otra aplicación. En estos casos, se podría guardar su módulo de datos en el repositorio de objetos para usarlo luego. Si trabaja en un ambiente de equipo, debería guardarse el objeto de repositorio compartido en la red para ser utilizado por los desarrolladores del equipo.

4.6. Bases de datos locales y de Cliente/Servidor

La primera gran división se hace entre los sistemas de bases de datos locales, o de escritorio, y las bases de datos SQL, o cliente/servidor.

A los sistemas de bases de datos locales se les llama de este modo porque comenzaron su existencia como soluciones baratas para un solo usuario, ejecutándose en un solo ordenador. Sin embargo, no es un nombre muy apropiado, porque más adelante estos sistemas crecieron para permitir su explotación en red. Tampoco es adecuado clasificarlas como "lo que queda después de quitar las bases de datos SQL". Es cierto que en sus inicios ninguna de estas bases de datos soportaba un lenguaje de consultas decente, pero esta situación también ha cambiado.

En definitiva, ¿Cuál es la esencia de las bases de datos, de escritorio? Pues el hecho de que la programación usual con las mismas se realiza en una sola capa. Todos estos sistemas utilizan como interfaz de aplicaciones un motor de datos que, en la era de la supremacía de Windows, se implementa como una DLL. En la época de MS-DOS, en cambio, eran funciones que se enlazaban estáticamente dentro del ejecutable. Observe el siguiente diagrama, que representa el uso en red de una base de datos "de escritorio":

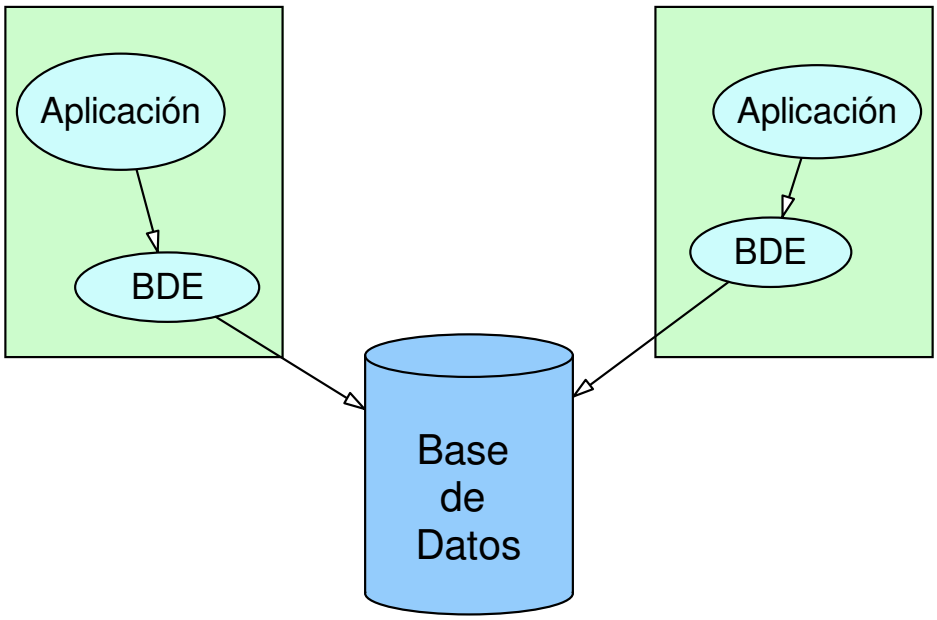


Figura 4.3 Base de Datos local

Aunque la segunda burbuja dice "BDE", el Motor de Datos de Borland, sustituya estas siglas por DAO, y podrá aplicar el diagrama a una tabla de Acces. He representado la aplicación y el motor de datos en dos burbujas separadas, pero en realidad, constituyen un mismo ejecutable, lo más importante, sin embargo, es que no existe un software central que sirva de árbitro para el acceso a la base de datos "física". Es como si las dos aplicaciones deambularan a ciegas en un cuarto oscuro, tratando de sentarse en algún sitio libre. Por supuesto, la única forma que tienen de saberlo es intentar hacerlo y confiar en que no haya nadie debajo.

Debido a esta forma primitiva de resolver las inevitables colisiones, la implementación de la concurrencia, las transacciones y, en último término, la recuperación después de fallos, ha sido tradicionalmente el punto débil de las bases de datos de escritorio. Si está pensando en decenas o cientos de usuarios atacando simultáneamente a sus datos, y en ejércitos de extremidades inferiores listas para tropezar con cables olvídese de Paradox, dBase y Acces, pues necesitará una base de datos cliente/servidor.

Las bases de datos cliente/servidor, o bases de datos SQL, se caracterizan por utilizar al menos dos capas de software, como se aprecia en el siguiente diagrama:

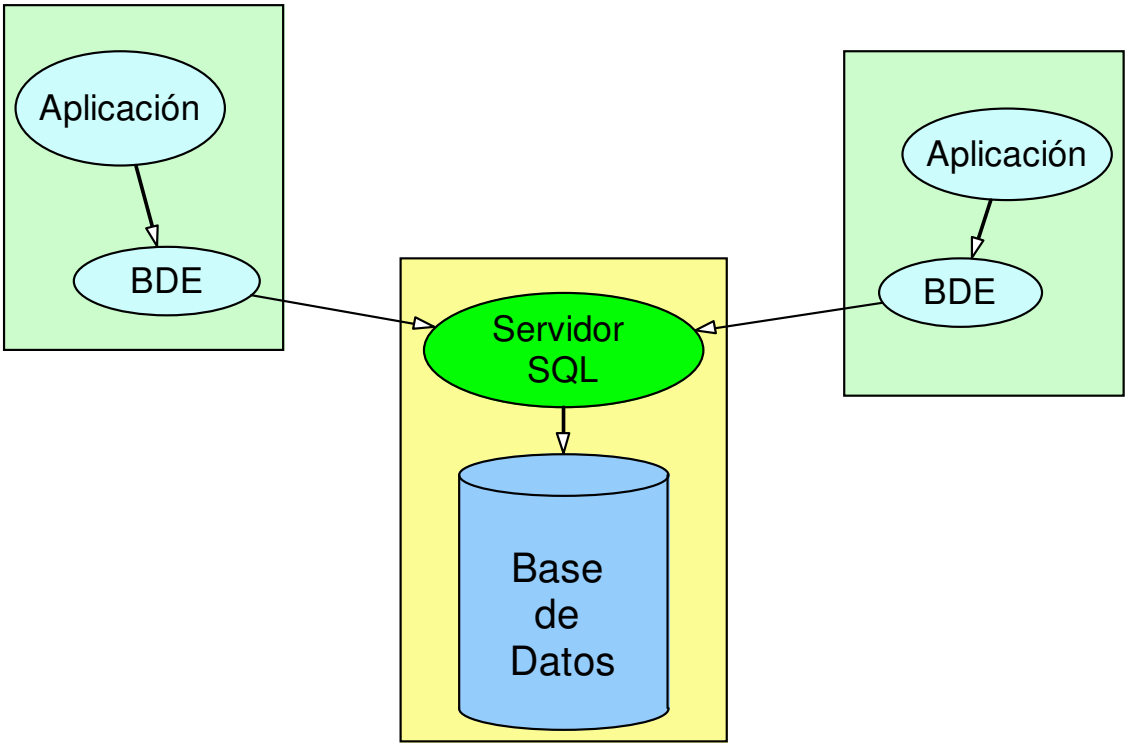


Figura 4.4 Base de Datos cliente servidor

El par aplicación + motor local de datos ya no tiene acceso directo a los ficheros de la base de datos, pues hay un nuevo actor en el drama: el servidor SQL. He cambiado el rótulo de la burbuja del motor de datos, ya ahora dice “cliente SQL”. Esta es una denominación genérica. Para las aplicaciones desarrolladas con Delphi y el Motor de Datos de Borland, este cliente consiste en la combinación del BDE propiamente dicho más alguna biblioteca dinámica o DLL suministrada por el fabricante de la base de datos. En cualquier caso, todas estas bibliotecas se funden junto a la aplicación dentro de una misma capa de software, compartiendo el mismo espacio de memoria y procesamiento.

La división entre bases de datos de escritorio y las bases de datos SQL no es una clasificación tajante, pues se basa en la combinación de una serie de características. Puede que uno de estos días aparezca un sistema que mezcle de otra forma estos rasgos definitorios.

4.6.1. Paradox

Comenzaremos nuestra aventura explicando algunas características de los sistemas de bases de datos de escritorio que pueden utilizarse directamente con Delphi. Paradox no reconoce directamente el concepto de base de datos, sino que administra tablas en ficheros independientes. Cada tabla se almacena en un conjunto de ficheros, todos con el mismo nombre, pero con las extensiones que explicamos a continuación:

Extensión	*Explicación*
.db	*Definición de la tabla y campos de longitud máxima fija.*
.mb	*Campos de longitud variable, como los memos y gráficos*
.px	*El índice de la clave primaria*
.Xnn, .Ynn	*Índices secundarios*
.val	*Validaciones e integridad referencial*

En el fichero .db se almacena una cabecera con la descripción de la tabla, además de los datos de los registros que corresponden a campos de longitud fija. Este fichero está estructurado en bloques de idéntico tamaño; se pueden definir bloques de 1,2, 4, 8, 16 Y 32KB. El tamaño de bloque se determina durante la creación de la tabla mediante el parámetro BLOCK SIZE del controlador de Paradox del BDE (por omisión, 2048 bytes); en el próximo capítulo aprenderemos a gestionar éste y muchos otros parámetros. El tamaño de bloque influye en la extensión máxima de la tabla, pues Paradox solamente permite 216 bloques por fichero. Si se utiliza el tamaño de bloque por omisión tendríamos el siguiente tamaño máximo por fichero de datos:

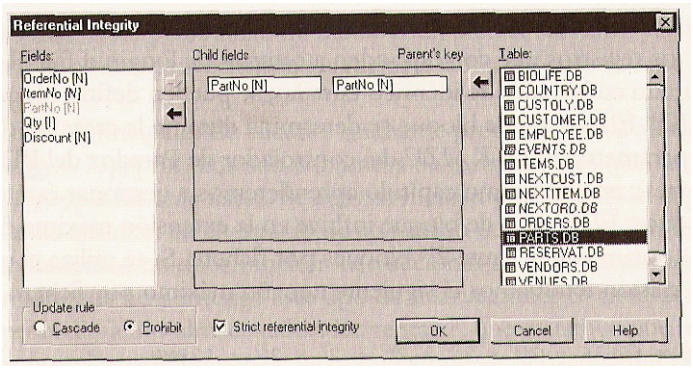
$2048 \times 65536 = 2^{11} \times 2^{16} = 2^{27} = 2'^{7}X\ 2^{20} = 128MB$

Un registro debe caber completamente dentro de un bloque, y si la tabla tiene una clave primaria (¡recomendable!) deben existir al menos tres registros por bloque; esto se refiere a los datos de longitud fija del registro, excluyendo campos de texto largos, de imágenes y binarios. Dentro de cada bloque, los registros se ordenan según su clave primaria, para agilizar la búsqueda una vez que el bloque ha sido leído en la caché. Un registro, además, siempre ocupa el mismo espacio dentro del fichero .db, incluso si contiene campos alfanuméricos que, a diferencia de lo que sucede en dBase, no se rellenan con blancos hasta ocupar el ancho total del campo.

Como explicamos anteriormente, el índice primario de la tabla se almacena en el fichero de extensión px. No es necesario que una tabla tenga índice primario, pero es altamente ventajoso. Al estar ordenados los registros dentro los bloques de datos, sólo se necesita almacenar en el índice la clave del primer registro de cada bloque. Esto disminuye considerablemente los requerimientos de espacio del índice, y acelera las búsquedas.

Es interesante conocer cómo Paradox implementa los índices secundarios. En realidad, cada índice secundario es internamente una tabla, con su propio fichero de datos, de extensión .Xnn, y su índice asociado. Ynn. La numeración de los índices sigue un esquema sencillo: los dos últimos caracteres de la extensión indican el número del campo en hexadecimal, si es un índice sobre un solo campo. Si es un índice compuesto, se utiliza una pseudo numeración hexadecimal, pero comenzando desde el "valor" eo y progresando en forma secuencial. Desde el punto de vista del usuario, los índices secundarios tienen un nombre asociado. El índice primario, en cambio, no tiene nombre.

Paradox permite la definición de relaciones de integridad referencial. La siguiente imagen muestra el diálogo de Database Desktop que lo hace posible:



El motor de datos crea automáticamente un índice secundario sobre las columnas de la tabla dependiente que participan en una restricción de integridad. Si la restricción está basada en una sola columna, el índice recibe el nombre de la misma. La tabla que incluyo a continuación describe la implementación en Paradox del comportamiento de la integridad referencial respecto a actualizaciones en la tabla maestra:

	BORRADOS	*MODIFICACIONES*
Prohibir la operación	*Sí*	*Sí*
Propagar en cascada	*No*	*Sí*
Asignar valor por omisión	*No*	-

No se permiten los borrados en cascada. No obstante, este no es un impedimento significativo, pues dichos borrados pueden implementarse fácilmente en las aplicaciones clientes. En cambio, trate el lector de implementar una propagación en cascada de un cambio de clave cuando existe una integridad referencial...

No fue hasta la aparición de la versión 3.0 del BDE, que acompañó a Delphi 2, que Paradox y dBase contaron con algún tipo de soporte para transacciones. No obstante, este soporte es actualmente muy elemental. Utiliza un undo log, es decir, un fichero en el que se van grabando las operaciones que deben deshacerse. Si se desconecta la máquina durante una de estas transacciones, los cambios aplicados a medias quedan grabados, y actualmente no hay forma de deshacerlos. La independencia entre transacciones lanzadas por diferentes procesos es la mínima posible, pues un proceso puede ver los cambios efectuados por otro proceso aunque éste no haya confirmado aún toda la transacción.

Finalmente, estas transacciones tienen una limitación importante. La contención entre procesos está implementada mediante bloqueos. Actualmente Paradox permite hasta 255 bloqueos por tabla, y dBase solamente 100. De modo que una transacción en Paradox puede modificar directamente un máximo de 255 registros.

Paradox usa un sistema de Semáforos para informar a otras aplicaciones o instancias de programas que ciertos registros están siendo usados por otro usuario, en contra de otros sistemas como el dBase que marcaba físicamente los registros que estaban en uso, Paradox crea unos ficheros en los que se registran los usuarios en uso, así como los registros bloqueados. En este capítulo nos centraremos en el protocolo de bloqueos de la versión 7 y especialmente en el BDE 4.0 que acompaña a Delphi 3.

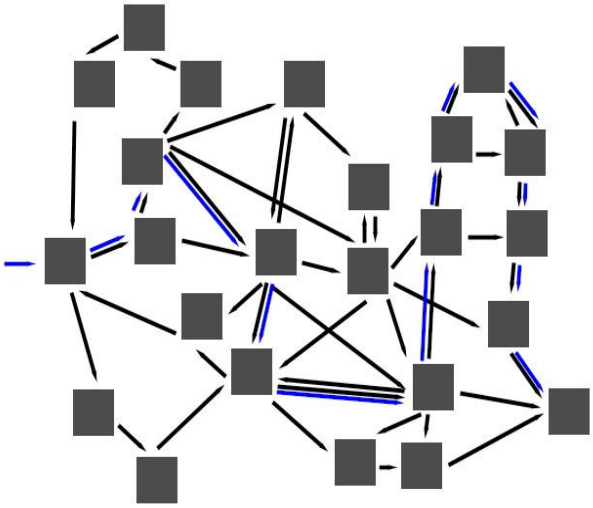
Paradox crea el fichero PDOXUSRS.LCK en cada directorio donde una aplicación este accediendo a tablas, este fichero regula el acceso concurrente a las tablas, este

fichero apunta y se registra en el fichero PDOXUSRS.NET que contiene la información de todos los usuarios -aplicaciones que acceden a un directorio. Tambien, para evitar el acceso a las tablas por versiones anteriores del BDE se crea un fichero exclusivo PARADOX.LCK. El BDE necesita un directorio para la creación de tablas temporales y otros usos, para ello en el llamado "Directorio Privado", crea sendos ficheros PDOXUSRS.LCK y PARADOX.LCK evitando que otros programas accedan a ese directorio.

4.6.2. Protocolo de Bloqueos

Paradox anota los registros bloqueados de las tablas en el fichero de bloqueos PDOXUSRS.LCK, en este fichero se controlan los accesos a diferentes tablas y por distintos usuarios, cuando un usuario intenta bloquear un registro mira en el fichero y comprueba que otro usuario no lo tenga bloqueado ya, en caso de que este libre escribe su bloqueo, impidiendo que otros lo hagan, cuando el registro es liberado, se actualiza su estado en PDOXUSRS.LCK.

Cuando varios usuarios intentan leer y escribir en una misma tabla dentro de un entorno de red, todos deben realizar sus bloqueos por medio del fichero PDOXUSRS.NET que a su vez mantiene una relación con los fichero *.LCK. Un error común al realizar aplicaciones multiusuario es que las aplicaciones no apuntan al mismo fichero *.NET, creando un desconcierto en el BDE y como resultado que los bloqueos no se realizan y las actualizaciones son mas bien aleatorias. Dentro del fichero NET, también se guarda otro tipo de información, como número y nombre de usuarios conectados, y otra información menos relevante.



Cuando se intenta acceder a una Tabla Paradox el BDE busca el fichero PDOXUSRS.NET, si no lo encuentra crea uno, continuando con el proceso de arranque. Si BDE encuentra un fichero NET pero esta apuntando a otro directorio, nos muestra el mensaje de "Multiple net files in use", generando una excepción impidiendo la carga del BDE. Si no hay errores crea los fichero *.LCK en el directorio privado, si encuentra ficheros, genera la excepción y no se carga el BDE.

En resumen podríamos decir que el fichero NET gobierna los directorios, y los ficheros *.LCK los registros.

Correcta Inicialización. Para evitar errores en la inicialización de nuestra aplicaciones en RED deberemos haber comprendido el mecanismo de bloqueos de paradox y obrar en consecuencia. n primer lugar en la maquina que usemos como servidor crearemos una estructura de este tipo:

....\Datos

....\Datos\NET

\Datos será el directorio en el que crearemos las Tablas y \NET lo usaremos para albergar el fichero PDOXUSRS.NET. Una vez creados los directorios crearemos un recurso compartido de Windows en el directorio \Datos para que el resto de usuarios tengan acceso, por ejemplo , una vez hecho, nos referiremos siempre a este directorio con las normas UNC, así \Datos pasara ha ser \\MiServer\MisDatos y el directorio para el fichero de control de Usuarios será \\MiServer\Misdatos\Net.

Si usamos ALIAS de Delphi deberemos actualizar estos datos en cada maquina, estableciendo, por ejemplo, como directorio privado el directorio temporal de Windows, estableciendo el resto de datos como se indica arriba.

Si realmente estamos haciendo aplicaciones potentes y profesionales, deberemos por un lado, o bien crear ALIAS, en tiempo de ejecución, o mejor aun, usar componentes Tsession y redirigir los directorios sobrescribiendo los valores del BDE por defecto. Siguiendo con el ejemplo anterior en nuestro DataModule deberemos tener un componente Tsesion y otro TdataBase, cuando se crea el DataModule asignamos los valores, bien manteniéndolos prefijados por programa o leyéndolos de un fichero a nuestro gusto (¿un archivo con extensión .INI ?).

```
}

MiDataModule.OnCreate;

Begin

With MiSesion do begin
  NetFileDir := '\\MiServer\MisDatos\NET';
  PrivateDir := 'c:\Temp';
End;

With MiDataBase do begin
  Params.Clear;
  Params.Add('PATH='+'\\MiServer\MisDatos');
  Params.Add('DEFAULT DRIVER=PARADOX');
  Params.Add('ENABLE BCD=FALSE');
  Connected := true;
End;

End;
{
```

Lógicamente todas las tablas del datamodule deberán, hacer referencia a la session y la Database modificada.

Con estas sencillas reglas y formas de trabajo estaremos seguros que nuestros programas en red usando Paradox no fallen, y realicen los bloqueos correctamente.

Más cosas. Como no todo es perfecto o bien nuestro programa puede colgarse en un momento dado, o un usuario con prisas puede apagar el ordenador sin más, a veces quedan ficheros *.LCK en el directorio de datos sin que este ningún usuario conectado, en este caso cuando nuestro programa intenta inicializar el BDE, este nos da un error que las tablas están "busy", siendo falso. El remedio es verificar que hay usuarios conectados y antes de abrir las tablas comprobar la existencia de tales ficheros, si no hay usuarios y los

ficheros están el directorio nuestra aplicación deberá ser lo suficientemente inteligente para borrarlos y olvidares del tema. Este proceso lo haremos una vez activa la session pero sin tablas abiertas.

Limites del BDE. Estos son los limites máximos del BDE y Paradox.

Clientes en el sistema:	48	Campos por tabla	255
Sesiones por clientes:	256	Registros por tabla	2 billones
Tablas abiertas:	2048	Bytes por tabla DB	2 billones
Drivers cargados:	32	Usos concurrentes de una tabla	255
Sesiones por sistema:	12288	Numero de campos en índices	16
Cursores por session:	4000	Numero de índices secundarios por tabla	127

Máximos errores consecutivos (Stack): 16

Tablas abiertas por el sistema: 127

Registros bloqueados: 255

Registros en una transacción: 255

Ficheros físicos abiertos (DB,PX,MB,X?,Y?,VAL,TV) 512

Usuarios en un mismo PDOXUSRS.NET 300

Capítulo 5.

Aplicación desarrollada

5.1. Descripción

Se desarrolla un sistema completo de aplicaciones software para restaurantes, con comunicaciones en red entre los diferentes puestos de trabajo, y con la innovación de marcas temporales que ayuden a cocineros y clientes a saber los tiempos de preparación y espera de los pedidos.

Figura 5.1 Camarero anotando pedido con PDA

El sistema se compone de varias aplicaciones corriendo en diferentes máquinas, con una constante en todas ellas: una aplicación principal y una aplicación de red (servidor y/o cliente) corriendo en todas las máquinas. Con esto se consigue que las distintas máquinas puedan compartir información entre ellas, siendo totalmente independientes, pero pudiendo cooperar entre ellas.

Los servicios desarrollados por la aplicación son variados, pero se centran sobre todo en aportar sencillez, velocidad y eficacia a los trabajos propios de camareros, cocineros y en especial a la comunicación.

El cocinero obtiene así una organización temporal de los pedidos, teniendo en cuenta su experiencia en cuanto a tiempos de preparación de los platos, y olvidándose de tener que comunicarse con los camareros para recoger pedidos y notificar los platos que ya han salido de cocina. Los camareros tienen en su mano una aplicación totalmente portátil e inalámbrica con toda la información de los productos del restaurante, que permite anotar los pedidos y comunicárselos al cocinero automáticamente.

Existe otra tercera aplicación, que controla las comunicaciones y los pedidos, y que se sitúa en la barra. El personal del restaurante que está en la barra necesita recoger los platos cocinados y conocer a qué mesa/s corresponde/n, además de comunicarse con la cocina y los demás camareros para un eficaz servicio.

Pues bien, todas estas comunicaciones, el control de las mesas y de los pedidos, de su evolución, y en general del estado de cada mesa es lo que implementa esta tercera aplicación.

Además, otra serie de herramientas de gestión, simplifican el uso de las anteriores, y añaden algunas funcionalidades más, bastante prácticas por cierto, como son una sencilla herramienta para realizar y modificar una base de datos con productos e información adicional acerca de los platos, y de las preferencias de los clientes.

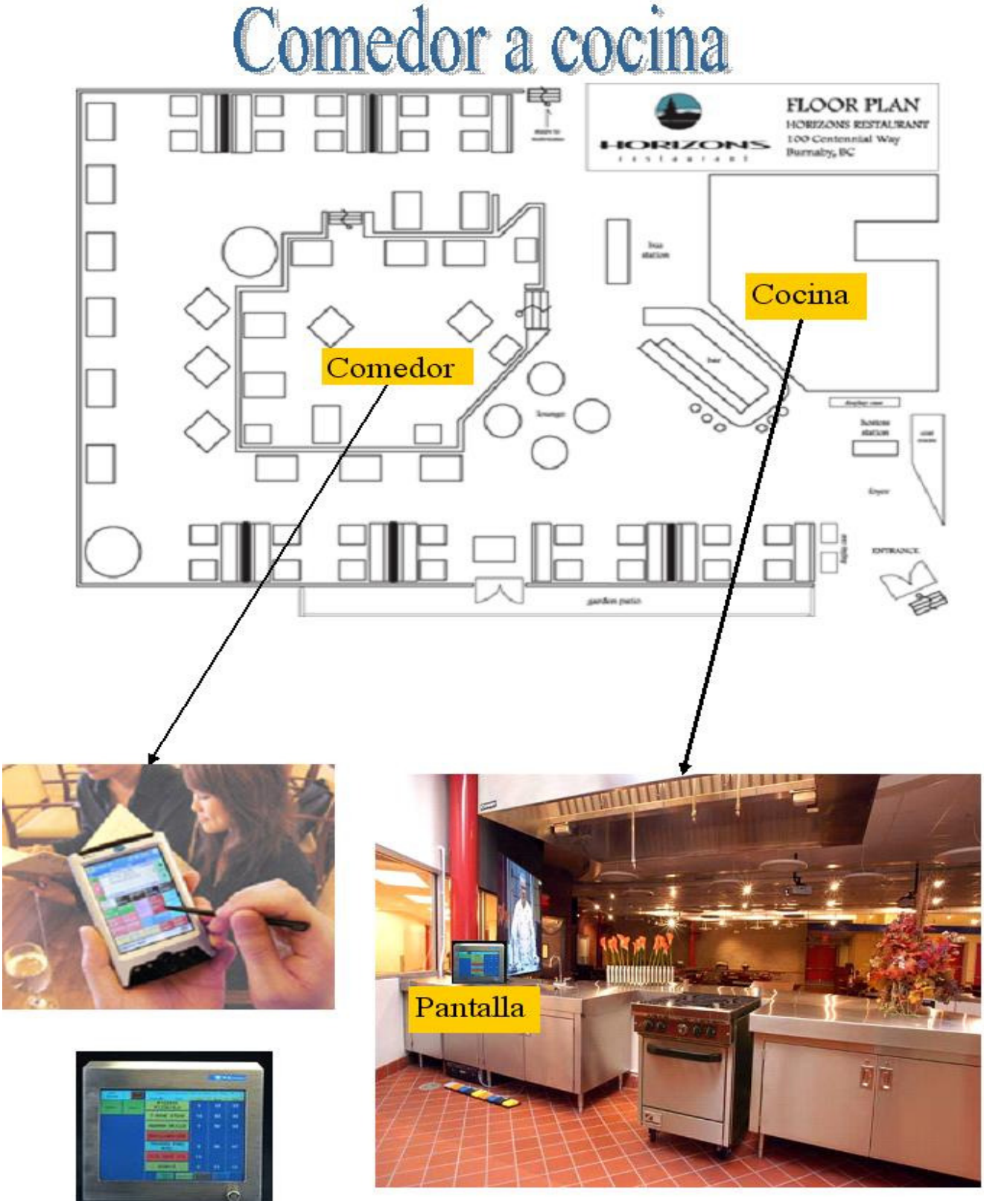


Figura 5.2 Diagrama de situación de los equipos en un restaurante

En el diagrama siguiente se observa un funcionamiento normal del conjunto de todas las aplicaciones. En el comedor un usuario realiza un pedido que es recogido por el camarero mediante su PDA (o un ordenador de pequeño tamaño). La PDA que porta el camarero envía automáticamente e inalámbricamente el pedido a la barra. Cuando llega a la barra el pedido, ésta reenvía los platos del pedido que han de ser preparados a la cocina, pero algunos otros, como son las bebidas y otros productos que no necesitan preparación, pueden ser servidos directamente desde la barra. Así ésta notifica a los camareros que ya pueden servir dichos productos.

Cuando el cocinero realiza un plato, la aplicación automatiza el envío de un mensaje a la barra para avisar de que la comida ya está preparada para servirse, así que en la barra recogen toda la información del transcurso y evolución de los pedidos. Finalmente la cuenta también la emite la barra.

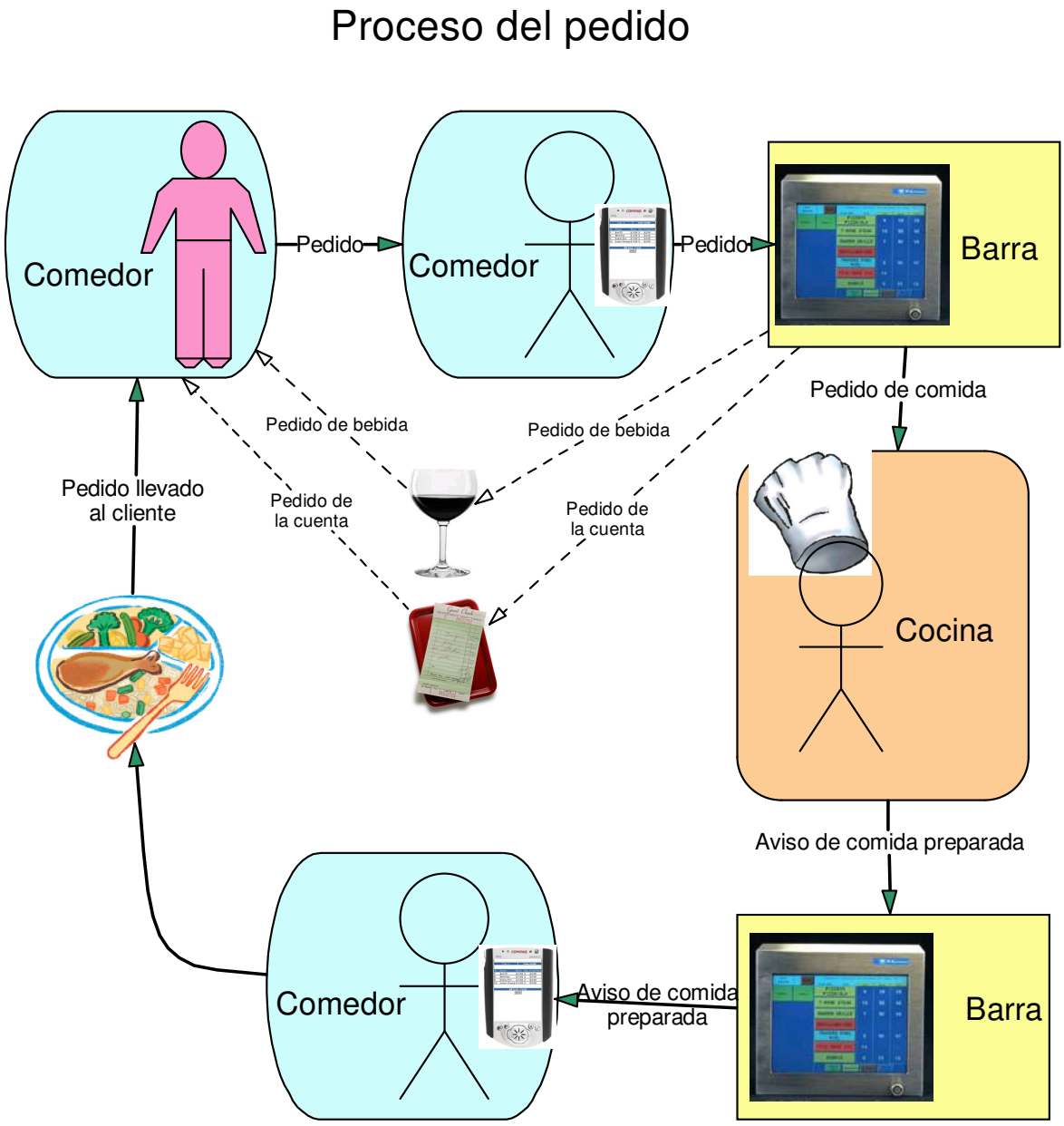


Figura 5.3 Proceso general de un pedido

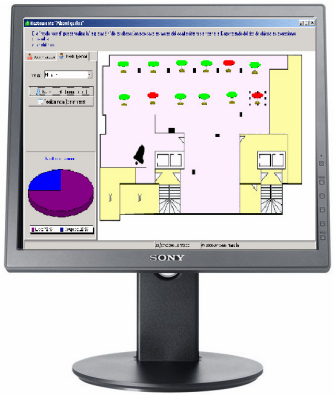
5.2. Tecnología utilizada

Por las razones que ya se han analizado en este documento, la aplicación se desarrolla con el entorno de programación Delphi 7 de la compañía Borland.

Para la comunicación por red se usa la arquitectura TCP/IP, y más concretamente el protocolo UDP y TCP. Se usa el protocolo UDP porque permite realizar broadcast, y esto es esencial para una aplicación que intente notificar a todas las máquinas inalámbricas de su radio de acción que deben actualizar la información que contienen porque se ha modificado.

El protocolo TCP se usa para servicios que no necesitan ser broadcast y, además, requieren que sea un servicio fiable entre dos usuarios, como por ejemplo "la transferencia de ficheros", ya que si se implementara con UDP, éste requeriría realizar el control de errores y reordenación de paquetes en la propia aplicación, pero TCP ya proporciona este servicio.

Todo lo anterior está referido al Software, pero en cuanto al hardware, el sistema está ideado para que convivan dos tipos de tecnologías de red: inalámbrica y cableada; y dos tipos de máquinas: de pequeño tamaño como PDAs (agendas personales portátiles) y PCs embebidos (pequeños ordenadores), y PCs normales o de mayor tamaño. Todos ellos de pantalla táctil, que aunque tienen un mismo funcionamiento que los que no la poseen, los hacen mucho más rápidos y sencillos de utilizar.



Esta tecnología es perfectamente asumible desde el punto de vista económico por cualquier restaurador ya que no supone grandes inversiones de dinero, sobre todo si tenemos en cuenta que el otro equipamiento de un restaurante suele superar en decenas de veces el precio de esta tecnología. Además es interesante comprobar cómo las empresas que han apostado por introducir esta tecnología en su labor cotidiana han despegado económicamente, y ahora les es imprescindible.

5.3. Arquitectura de la aplicación

La estructura de nuestro sistema se compone principalmente de tres (o más) equipos informáticos situados en tres puntos diferentes: uno en la cocina, otro cerca de ella (posiblemente en la barra), y el otro (u otros) es un dispositivo móvil de pequeño tamaño que porta cada camarero en todo momento.

El equipo situado en la barra es el núcleo central de las comunicaciones; por allí pasan todos los mensajes de la red, así que controla las comunicaciones y, además, reúne toda la información del restaurante y la comunica oportunamente a los demás módulos para que siempre estén actualizados. Los equipos de la cocina y de los camareros sólo se ven con la aplicación de la barra, así que su comunicación es punto a punto, pero obligatoriamente inalámbrica en el equipo de los camarer@s.

Cada equipo utiliza una diferente aplicación (excepto los de los camareros que son iguales entre sí), específica para su determinado uso, pero todas perfectamente comunicadas al utilizar un mismo protocolo.

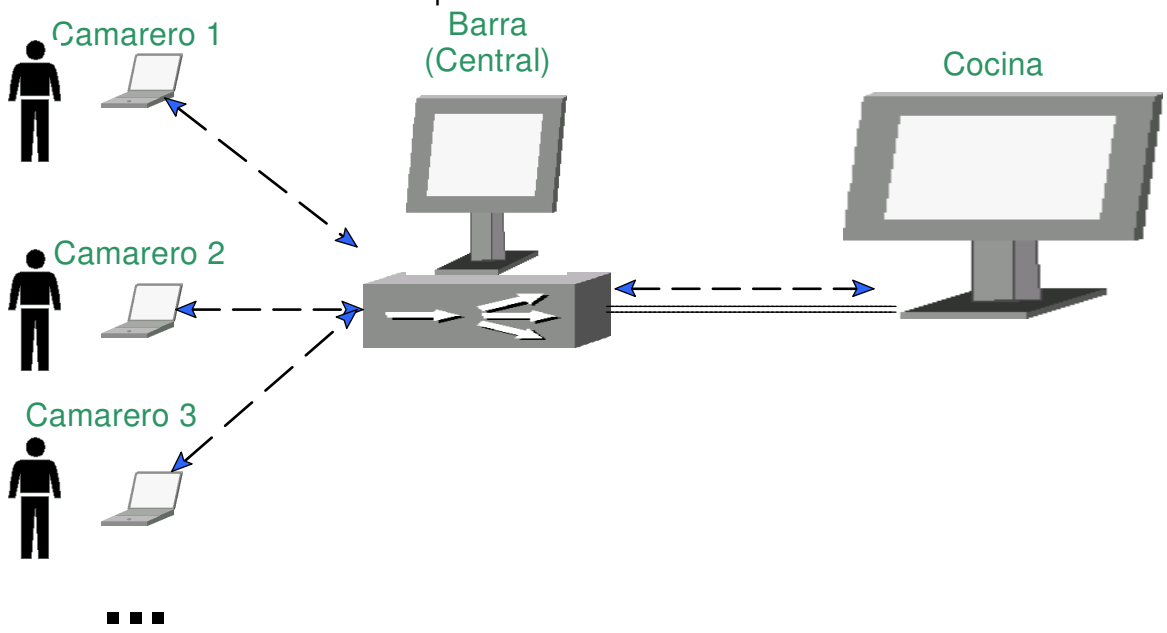


Figura 5.4 Estructura de todo el sistema

5.4. Camarero

5.4.1. Descripción.

Figura 5.5 Toma de pedidos mediante PDA

Esta aplicación informática simula la típica libreta del camarero cuando toma la nota del pedido de los clientes, pero con más y mejores opciones, propias de una aplicación electrónica en red.

Esa es una posibilidad pero la otra es darle la libertad al cliente de pedir lo que quiera en el momento que lo desee, directamente desde su mesa utilizando una pantalla táctil instalada en su propia mesa, que le permitiría poder ver los platos, con su foto, precio y tiempo medio de preparación.

Figura 5.6 Pocket PC con el programa Camarero para que sea el cliente el que directamente pide.

Figura 5.7
PDA del camarero

Por una parte el programa muestra los productos, junto con su precio y, también, si se desea, una foto del producto, para un más rápido y fácil manejo. Se está obligado a que no sea necesario utilizar un teclado, ya que la aplicación correrá fundamentalmente en PDAs (agendas personales portátiles), PCs embebidos (pequeños ordenadores), y PCs normales, pero de pantalla táctil todos ellos. Esto aporta de nuevo facilidad de uso, al no tener que transportar grandes máquinas (en el primer caso) por no llevar teclado y, además, porque no hay que escribir nada, sólo pulsar.

5.4.2. Estructura.

La aplicación Camarero se compone de varias unidades, pero tiene sólo dos ventanas (o forms) la inicial llamada “Nuevo pedido” y la principal, también llamada carta.

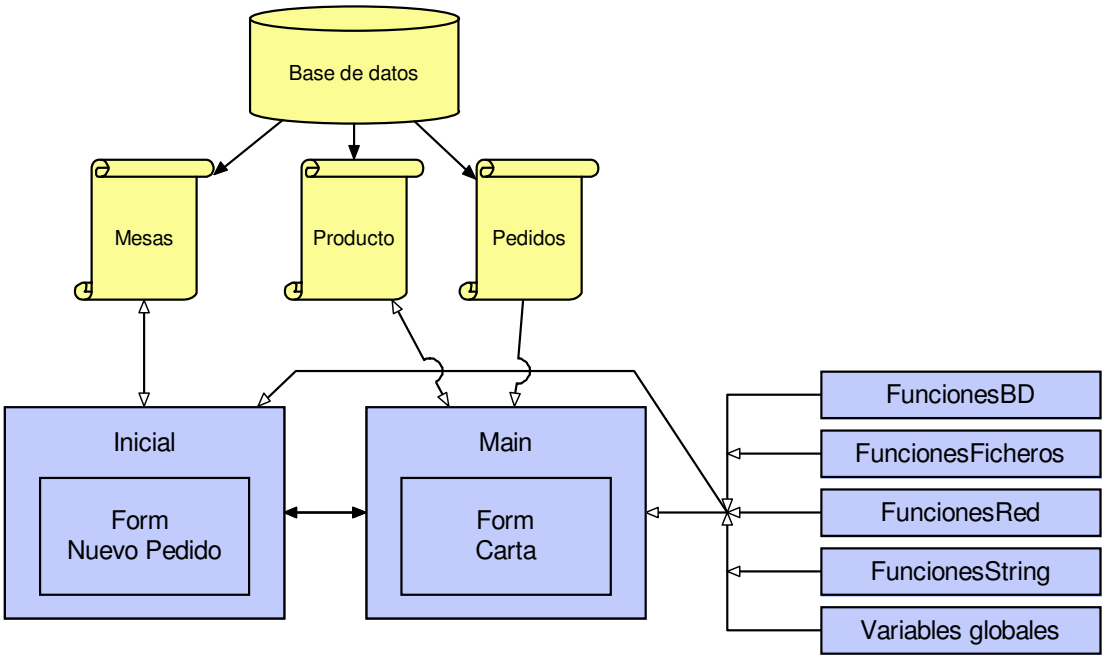


Figura 5.8 Estructura del programa *Camarero*

Las ventanas “Nuevo Pedido” y main se valen de las otras unidades para liberarse un poco de código, ya que utilizan las funciones que se han declarado e implementado en las unidades “Funciones…”. Con esto se consigue dar mayor claridad al código, ya que contienen mucho menos código, y además están las funciones ordenadas en varias unidades, según en el campo en el que trabajan: strings, bases de datos, red… Las ventanas principales declaran en su cabecera que van a usar estas funciones, con la partícula “uses”, y sólo se limitan a repartir las funciones entre varias unidades y después llamar a estas dentro de las ventanas principales.

La unidad llamada “Variables globales” tiene otro sentido. Como su propio nombre indica, contiene variables que tanto la form “inicial” como la “main” van a usar, y los valores que contienen son globales a la aplicación. Por tanto estarán usando la misma variable ambas unidades. De otra forma, no se hubiera podido trabajar con una variable en una form, y pretender que su valor cambien también en la otra.

Cuando una form se muestra, el título de ésta es lo que el usuario ve y no el nombre de la unidad, por lo que no tiene obligación de corresponderse el nombre de la unidad con el título de la form, ya que tienen distintos propósitos. La unidad puede contener varias ventanas, una o ninguna. Así que, ya que el nombre de la unidad va destinada sólo al programador, su nombre es irrelevante para el usuario. El nombre de la form si es para el usuario, así que debe tener un título más cercano al usuario que al programador. Así se ha elegido esta correspondencia entre nombres de unidades y forms:

- Unidad Inicial que contiene la form “Nuevo Pedido”.
- Unidad Main que contiene la form “Carta”.

Las dos unidades necesitan conectarse continuamente a la base de datos para leer y escribir en ella, excepto en el caso de la tabla de productos, que sólo es modificable por la aplicación “gestor de productos”.

5.4.3. Funcionamiento

El programa al ejecutarse comienza mostrando la ventana “Nuevo Pedido” y, desde ésta, pasa a la ventana “Carta” después de pulsar el botón ocupar mesa. La ventana carta vuelve a la anterior ventana cuando se pulsa uno de los botones: “Enviar pedido”, “Borrar pedido”, y se acepta su ventana emergente. Nunca se ven las dos ventanas a la vez, así que cuando una se muestra, la otra desaparece.

NUEVO PEDIDO. Contiene una lista de mesas disponibles, cuando se selecciona una de entre ellas, el programa se comunica con la base de datos para marcarla como ocupada.

CARTA. La ventana principal de la aplicación camarero está diseñada siguiendo la distribución de la figura 5.9. Para realizar este diseño se estudiaron otros programas de bares y restaurantes y, con todos ellos, se ha realizado una especie de collage, observando sobre todo cuáles eran las necesidades de los usuarios de este tipo de software. En todos ellos siempre hay elementos comunes: la rapidez, elementos visuales de gran tamaño, y fáciles de reconocer y entender por alguien que se encuentre por primera vez con este programa. Los botones suelen ser siempre de gran tamaño ya que todos estos programas se ven en pantallas táctiles, por eso el camarero va a necesitar una gran área en los botones, para no fallar en la pulsación. También las letras serán de gran tamaño para que faciliten la lectura.

Figura 5.9. Ventana principal del programa “Camarero”

Pasemos a comentar más en profundidad la ventana principal. Se reconocen cuatro partes principales, que a continuación explicamos por zonas:

1. En la zona superior central aparece la lista con los productos que se solicitan. Cada línea es un nuevo producto, y está formada por las siguientes columnas:
   - Producto, con el nombre del plato elegido.
   - Cantidad, indica el número de pedidos.
   - Precio unitario de cada producto.
   - Notas con observaciones que suelen añadir los clientes al pedido de un plato, por ejemplo: “con poca sal”, “muy hecho”, “sin patatas”, etc.

2. Arriba, en ambos lados de la pantalla, tenemos distribuidas las botoneras, con las opciones y necesidades más importantes del camarero, como son:
   - Borrar y realizar un nuevo pedido, ya que el último realizado no vale.
   - Enviar el pedido.
   - Borrar una línea del mismo.
   - Aumentar o disminuir la cantidad solicitada de un producto
   - Añadirle una nota a un producto.
   - Cambiar el precio de un producto
   - Realizar un descuento sobre el producto.

Figura 5.10 Ventana principal realizando un pedido

3. En la parte inferior izquierda, se encuentra la botonera con las "13 secciones" o familias de productos en los que se han clasificado los pedidos. Cada botón corresponde a una familia diferente. Al pulsarlo, se despliegan en la botonera de la parte inferior derecha los productos de esa familia, y los demás quedan ocultos.

Figura 5.11 Botonera de la familia de productos.

4. Botonera inferior derecha. Esta es finalmente la parte más importante del programa. Aquí se despliegan toda la serie de productos o platos que componen la carta de un restaurante o bar. Está compuesta por una lista de pestañas con paneles para cada diferente familia de productos.

En cada panel, hay un botón y una etiqueta con el nombre de cada producto. Además cada botón puede llevar una foto del plato, lo que nos va a facilitar mucho el encontrar el deseado entre tanta variedad; y además, su uso se hace mucho más intuitivo. Al presionar cada pestaña también se activa el panel correspondiente y se ocultan los demás. Es lo mismo que hacíamos con la anterior parte, aunque en esta ocasión su menor tamaño lo hace menos para una pantalla táctil. Cuando se presiona cada botón, automáticamente el producto se añade al final de la lista de pedidos.

Figura 5.12 Botones de productos del programa principal.

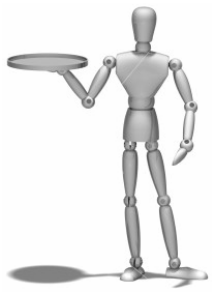
La funcionalidad más importante (y más difícil de implementar) ha sido posibilitar que el usuario pueda cambiar los botones, el número, su apariencia, etc. de una manera sencilla y rápida, sin necesidad de que estos profesionales sepan programar, ni modificar el código. Esto se consigue dando dos capacidades al programa:

- Poder crear botones en tiempo de ejecución, apareciendo en la carta tan sólo los botones de los productos que aparecen en la base de datos.

- Conectarse a una base de datos con toda la información de la carta de productos, y conocer en tiempo de ejecución todos sus datos (precio, si se ha agotado, etc.). Esa base de datos contiene además de las tablas con los productos, una carpeta con todas las imágenes de los iconos en formato .BMP de los archivos. Cada imagen tiene como nombre el código usado en la tabla de productos para ese producto en concreto. Esto se puede realizar manualmente, pegando la imagen con el número de código como nombre (esto se puede consultar en la base de datos en la tabla de productos). Pero esto tiene limitaciones, ya que habría que ajustarse a un tamaño concreto, una forma cuadrada, y un formato .BMP. Pues para liberar de esto a los usuarios, se le añade una funcionalidad a la herramienta "gestor de productos" que nos pide automáticamente una imagen para el producto, y la convierte al formato y al tamaño adecuado. Si no se la facilitamos, el programa adjudica una imagen estándar por defecto.

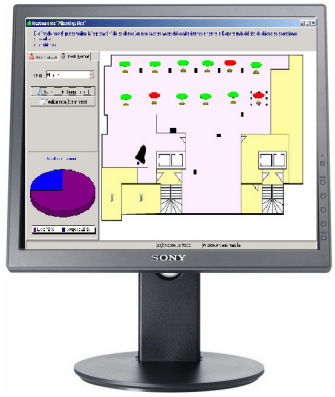
5.5. Barra

5.5.1. Descripción.



Se desarrolla una aplicación informática capaz de controlar las comunicaciones en un restaurante, además de dar información acerca de qué mesas están ocupadas o qué mesas tienen todos los primeros platos listos para servir o bien cuáles de ellas tienen algo pendiente para llevarles, etc.

Se hace uso de tablas de bases de datos y de una aplicación servidora que corre junto con la aplicación principal. La a. principal contiene un mapa del restaurante y la distribución de las mesas, permitiendo cambiar su situación en el local, al igual que guardar y cargar diferentes distribuciones de mesas para celebrar distintos eventos.



A esta aplicación se conectan tanto las aplicaciones "camarero" como la aplicación "cocina". La primera para enviarle sus pedidos, haciendo de intermediario entre la cocina y los camareros, como también para recoger las últimas bases de datos actualizadas, o saber el estado de las mesas.

Con la segunda se comunica para reenviar los pedidos, y recibir la lista de los platos cocinados, junto con algunas opciones más como son la actualización de bases de datos, o reenvío de mensajes de productos agotados.

5.5.2. Estructura.

La aplicación "barra" está compuesta por dos aplicaciones muy importantes como son "FMain" y la aplicación "servidor", ambas claramente destacables sobre las otras.

La unidad principal de toda la estructura es la unidad "FMain". Ésta es la que implementa la form principal, la que se conecta con las bases de datos y la que llama al servidor. Al iniciarse ésta, llama a la aplicación servidor para que ambas corran separadas durante toda la ejecución del programa principal; y cuando ésta termina, el programa servidor también muere. La aplicación FMain necesita conectarse a la base de datos para conocer información, por ejemplo, referente a las mesas; o para enviar a los demás puntos de la red las actualizaciones de la base de datos, a través del servidor.

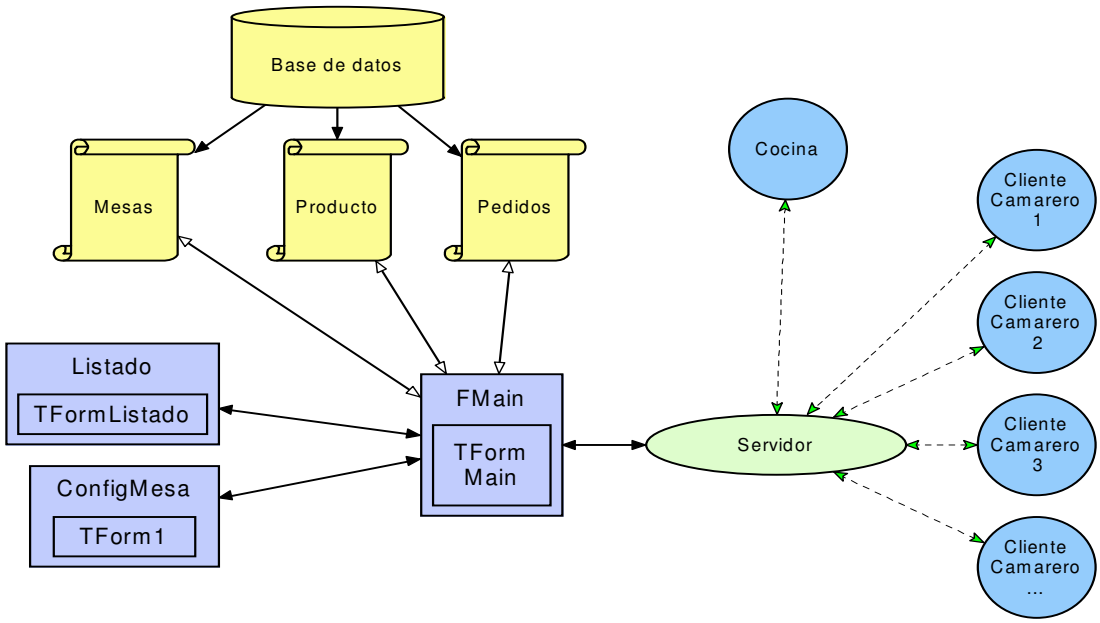


Figura 5.13. Estructura del programa "Barra"

El servidor es el punto central de las comunicaciones en el restaurante. Nos controlará las comunicaciones entre camareros y cocina, además de reenviar y actualizar ficheros en los anteriormente mencionados elementos de red.

Las otras dos unidades, "listado" y "configMesa", contienen ventanas adicionales que el programa principal llama para distintos usos, como mostrar la lista de platos pendientes por servir (para el caso de "Listado").

5.5.3. Funcionamiento.

Modo Normal.

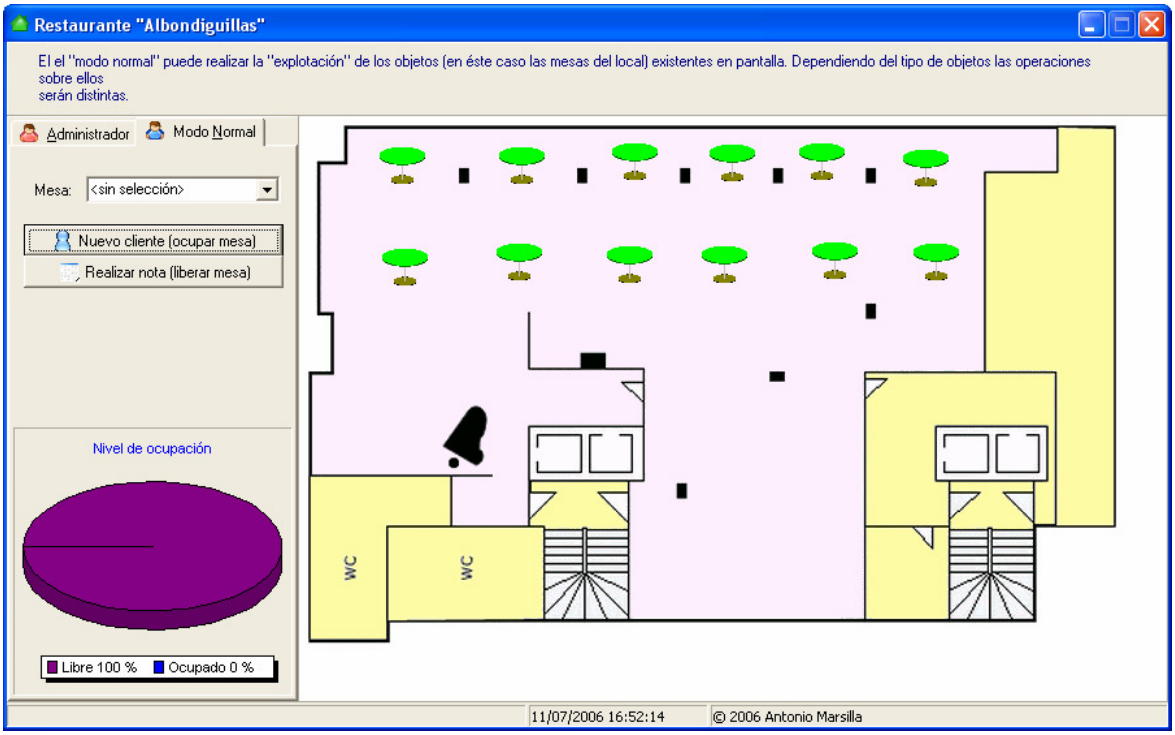


Figura 5.14 Ventana principal de la aplicación "Barra"

El programa nada más abrirse presenta una apariencia como la de la figura de la página anterior, que corresponde al modo normal. Se explica a continuación el programa por partes:

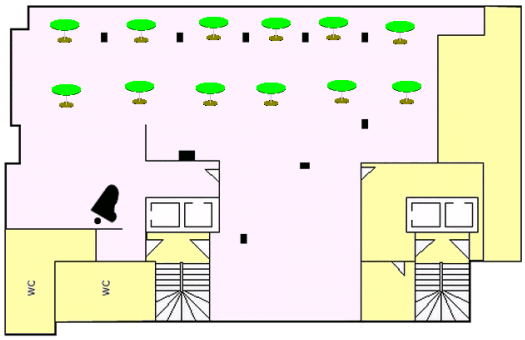


Figura 5.15 Mapa del restaurante con las mesas y sus estados

Croquis. Es la parte más visible y llamativa del programa. Es un pequeño mapa en color del restaurante en planta, junto con la distribución de mesas escogidas. Si se hace un clic sobre una de las mesas, esta se activa con unos pequeños cuadraditos negros alrededor de ésta que nos indican qué mesa está seleccionada. Esto ocurre tanto en el modo normal como en el modo administrador (más tarde hablaremos de este modo). También aparece el número de mesa que se ha seleccionado en la parte de la izquierda.

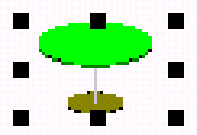


Figura 5.16 Mesa libre

Si en vez de un clic simple, lo que se realiza es un doble clic, se desplegará la ventana listado (la figura 5.17) que nos muestra información muy completa acerca del estado de la mesa: los platos que hay esperando ser cocinados, los que ya están preparados para servir y los que ya se han servido.

Figura 5.17 Ventana desplegable con el estado de las mesas

Además, su botonera inferior es muy simple, aunque no por eso menos práctica. Veamos a continuación la función que cumple cada botón:

- "Servir Platos" elimina de la lista "Platos preparados" (aunque también pueden ser bebidas u otro tipo de productos sin necesidad de preparación) y los envía a la lista Platos servidos. Por supuesto que a la vez el icono de la mesa cambia a su color correspondiente.

- "Cargar Lista". Este botón cargará en la lista de platos en espera todos los platos que había solicitado esa mesa. Sólo se ha de pulsar este botón cuando por alguna razón hayamos perdido algún pedido (error del camarer@ o de la máquina) y queramos recuperar la lista completa de los pedidos. Es importante borrar previamente los pedidos de esa mesa si no queremos que se nos mezclen unos con otros. Eso se hace con el siguiente botón.

- “Borrar todos los platos”. Podemos en ocasiones querer eliminar el pedido al completo, ya sea para volver a hacer nuevos cambios, o porque simplemente ya se ha anulado ese pedido. Esto lo haremos con este botón, que eliminará todos los platos de esa mesa de las 3 listas.

- “Cerrar”, cierra esta ventana y vuelve a la anterior. No hace nada diferente que no haga el botón típico marcado con una “X” que hay en la esquina superior derecha de todas las ventanas de Windows. Sólo que es mucho más grande para que en una máquina con pantalla táctil sea más fácil de pulsar.

Como se dijo anteriormente, en la parte de la izquierda se indica el número de la mesa seleccionada. Si se pincha en la flecha de su derecha, se despliegan todas las mesas existentes. También contiene otros dos botones que sirven para realizar la nota (y por tanto liberar la mesa), y otro para marcar la mesa como ocupada. Cuando realizamos la nota, una ventana como la de la figura 5.20 nos aparece para indicarnos el precio total de la cuenta.

Es muy importante señalar que cuando se libera o se ocupa una mesa, el color de ésta cambia, mostrando diferentes colores para diferentes estados. Esto se explica con más detalle en el punto próximo.

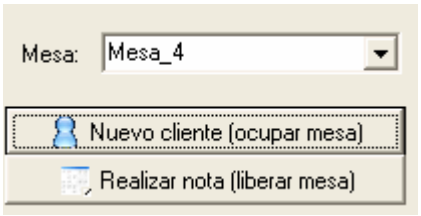


Figura 5.18 Mesa seleccionada.

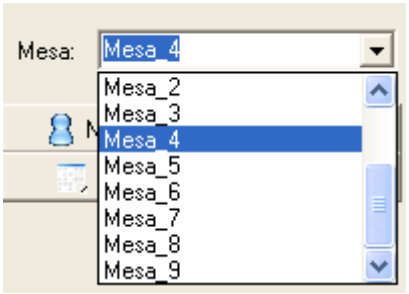


Figura 5.19 Listado de mesas desplegado

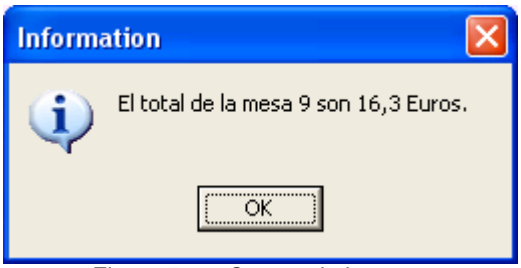


Figura 5.20 Cuenta de la mesa.

En la esquina inferior izquierda, un gráfico nos muestra el nivel de ocupación del restaurante. En versiones futuras, se puede distinguir entre los varios estados posibles que tiene cada mesa, en vez de utilizar sólo dos, aunque mucho más simples.

Figura 5.21 Nivel de ocupación del restaurante

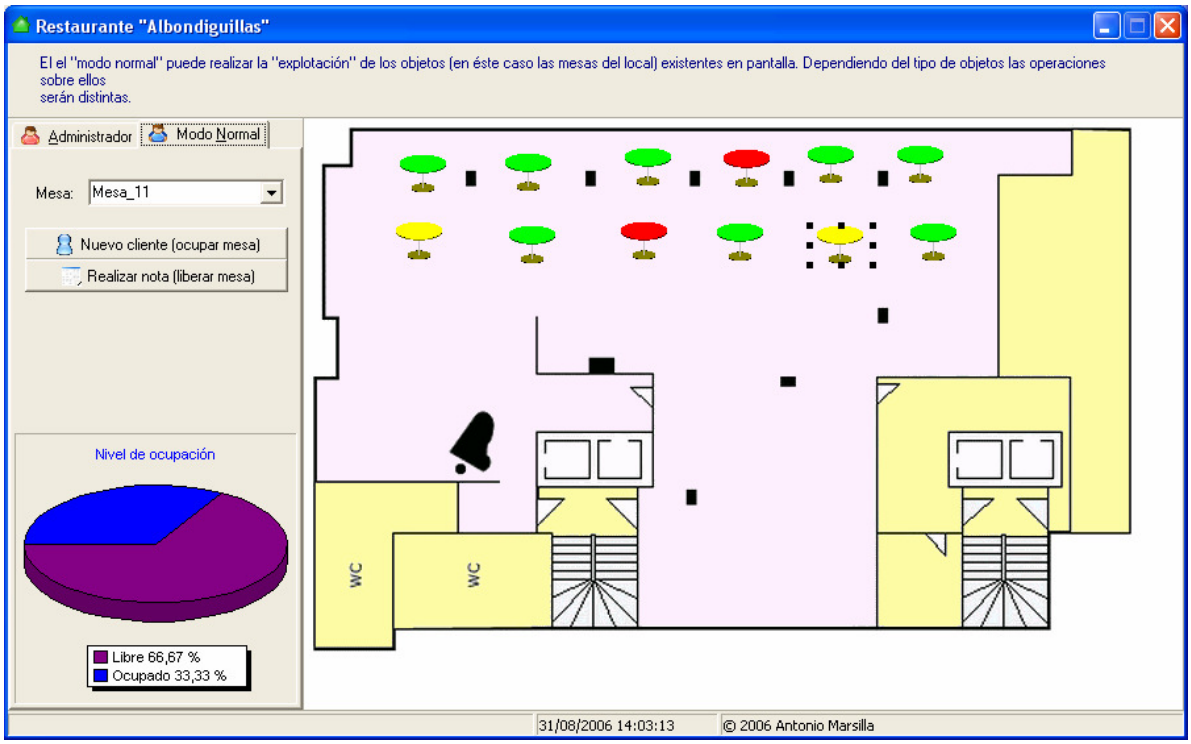


Figura 5.22 Programa Barra con mesas ocupadas

Modo Administrador. El otro modo de uso que tiene el programa barra es el modo Administrador. Es fácil ver que podremos cambiar de uno a otro sólo con pulsar en las pestañas que hay en la esquina superior izquierda del programa. La apariencia del modo administrador es estructuralmente muy parecida a la del modo normal. Veamos por partes las diferencias principales:

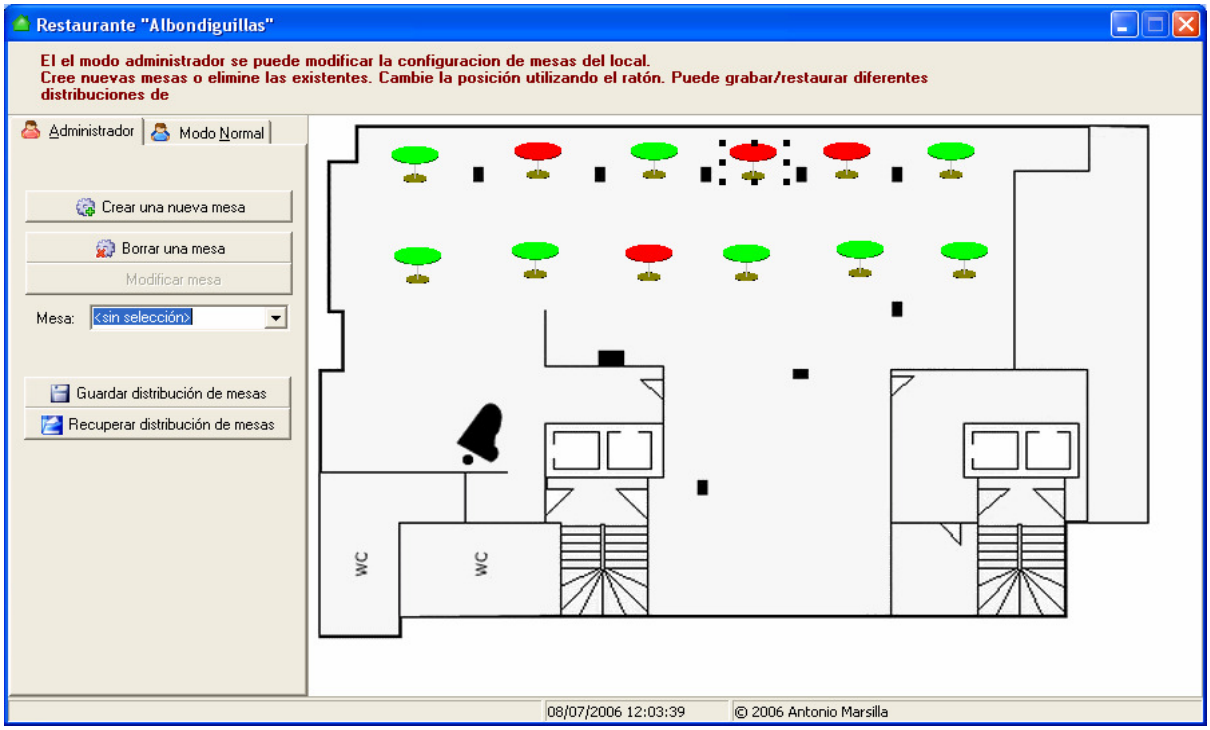


Figura 5.23 Programa Barra en el modo Administrador

El croquis ahora está en blanco y negro, principalmente para dar apariencia de un modo provisional (algo así como lo que utiliza Windows en su modo a prueba de fallos). Pero la gran diferencia es que ahora las mesas se pueden desplazar por todo el mapa, para permitir cambiar la distribución. Estos cambios realizados en la distribución permanecen cuando el programa cambia de modo, e incluso cuando el programa se vuelve a abrir después de haberse apagado.

Figura 5.24 Componente TSaveComps

Aquí se debe aclarar como se ha podido hacer esto, y es gracias a la creación de un componente llamado TSaveComps. Este componente permite guardar la posición y tamaño de todos los elementos que se encuentran en la ventana (o form). Basta con colocar el componente en el formulario y activarlo. Cuando la aplicación se cierra, graba la posición y el tamaño en un archivo .ini y al volver a ejecutarla los recupera de forma automática. El único requisito que se pide para poder utilizar este componente es que los elementos que necesitamos guardar su posición deben contener la propiedad Name.

```
…[Mesa_2]          [Mesa_4]          [Mesa_5]
Left=457           Left=379          Left=309
Top=20             Top=20            Top=21
Width=48           Width=48          Width=48
Height=30          Height=30         Height=30
ClassName=TImage   ClassName=TImage  ClassName=TImage
Parent=pnlPlano    Parent=pnlPlano   Parent=pnlPlano …
```

Figura 5.25 Porción de archivo .ini con la posición de algunos componentes.

En cuanto a la botonera de la izquierda, nuevos botones han aparecido, y desapareció el diagrama de ocupación del restaurante. Es lógico que lo hayamos quitado, ya que este modo está fundamentalmente ideado para cambiar la distribución de las mesas, algo que normalmente se hace cuando no hay clientes. Pero de todas formas, permanecen los estados de las mesas que había anteriormente, y cuando volvamos de nuevo al modo normal, el diagrama de ocupación volverá a aparecer mostrando la ocupación actual.

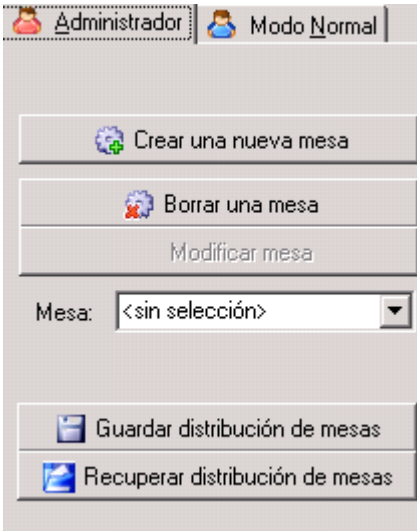


Figura 5.26 Botonera del modo Administrador

Los nuevos botones son:

- Crear una nueva mesa. Al pulsarlo aparece automáticamente una nueva mesa en el mapa, con toda la misma funcionalidad que tienen sus hermanas.
- Borrar una mesa. Desaparece la mesa seleccionada. Si no se ha seleccionado ninguna, un mensaje de alerta nos avisará de que tenemos que hacerlo.

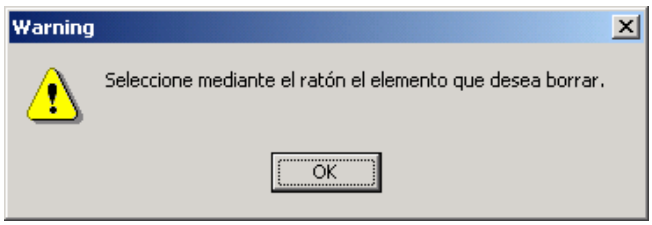


Figura 5.27. Warning para una mesa sin seleccionar.

- Guardar distribución de mesas. Después de modificar la distribución a nuestro antojo, podemos guardarla en un archivo, para poder recuperarla posteriormente. Se nos abrirá una nueva ventan para elegir el nombre y la ubicación del archivo.

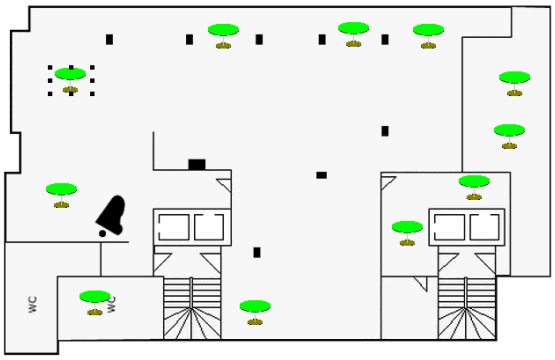


Figura 5.28. Distribución de mesas modificada

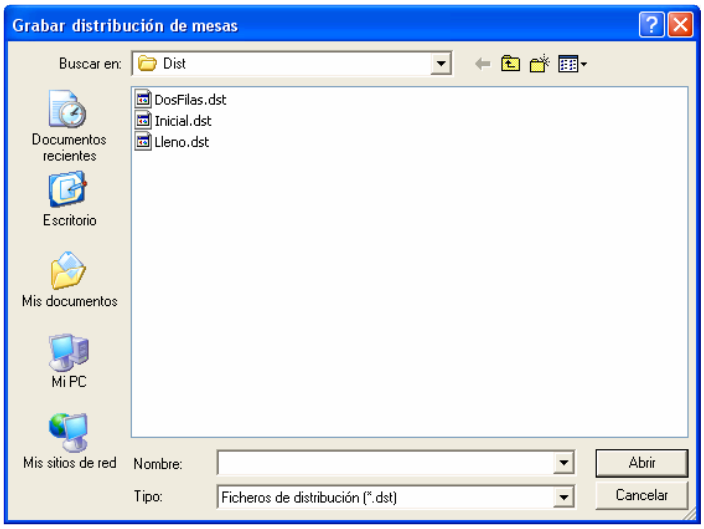


Figura 5.29 Guardando la distribución de mesas

- Recuperar distribución de mesas. El nombre del botón es claro, y volverá a aparecernos la anterior ventana (figura 5.29) para que escojamos el archivo con la distribución de mesas deseada.

5.5.4. Estados de las mesas.

Cada mesa por separado puede pasar por varios estados, en diferentes momentos. A continuación se explican cada uno de ellos, y el ciclo que siguen:

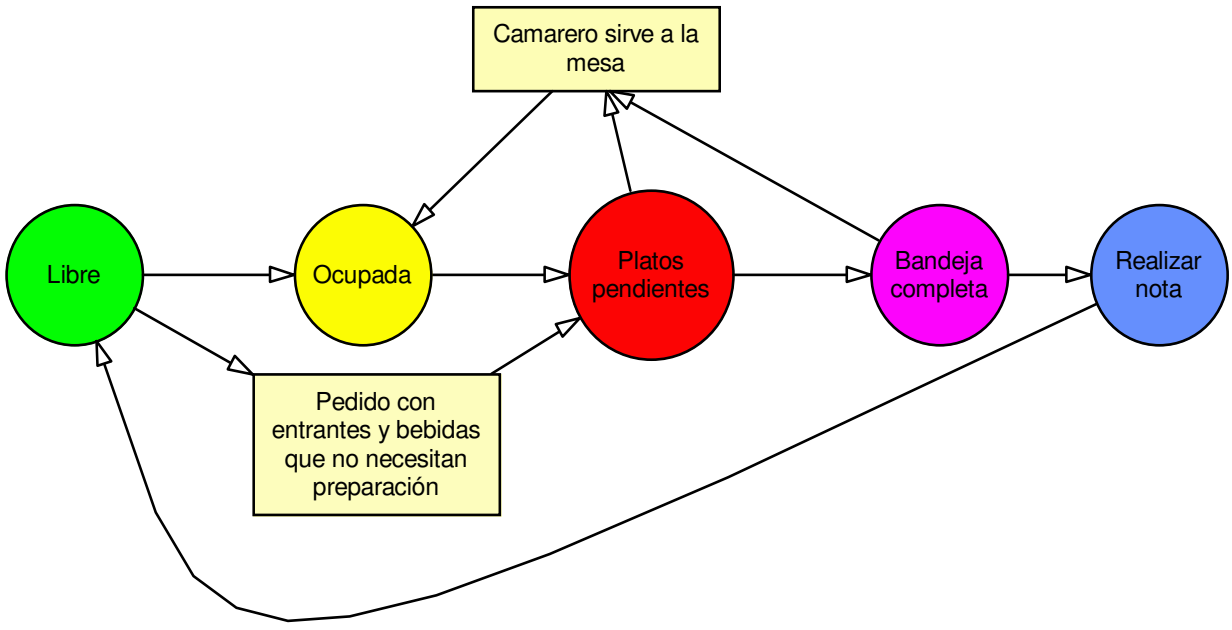


Figura 5.30 Diagrama de estados de las mesas

Libre: antes de haber ocupado la mesa. Es el estado inicial.

Ocupada: La mesa ha sido ocupada, pero no hay ningún plato, (ni bebida, u otro tipo de pedido) que se pueda servir. Hay que esperar a que en cocina terminen de realizar los platos. Cuando el camarero sirve una bandeja, el estado de la mesa vuelve a ser éste, excepto si está ya todo el pedido servido, con lo cual pasaríamos a "Realizar nota".

Platos pendientes: este es un estado intermedio. Se puede pinchar sobre este icono, y se ve como hay algunas cosas que ya se pueden servir, pero hay pendientes otros productos, hasta completar una bandeja. Cuando se ve este icono, el camarero ve aconsejable buscar otras mesas con una bandeja completa para servir (el siguiente icono azul); pero si no hubiera ninguna bandeja completa, entonces será conveniente que se ocupe de estas mesas, si no quiere que se le acumule el trabajo.

Bandeja completa: Se entiende por bandeja completada, a un conjunto de platos de la misma familia de una misma mesa. Por ejemplo, cuando una familia pide un menú en un restaurante, se suelen servir todos los primeros platos a la vez, así que una bandeja estará completa cuando todos los primeros platos de la misma mesa estén cocinados y listos para servir. Este icono de momento no se usa, es una previsión de futuro.

Realizar nota: Todos los platos han sido servidos, y sólo resta entregarle la nota al cliente y cobrarle. De este estado pasamos al primero "Libre", y vuelta a empezar.

5.6. Servidor de "Barra"

5.6.1. Descripción.

La aplicación Servidor tiene como función principal el control de las comunicaciones entre la cocina y los camareros, ya que corre en el punto central de toda la red. Se comunica con ambas partes (con sus aplicaciones cliente), y hace las funciones propias de una red: actualización y reenvío de ficheros entre ambas partes, y envíos de información como IPs, o estado de los pedidos.

5.6.2. Estructura.

El programa servidor está realizado en un solo archivo, utilizando el componente indy IdTCPServer. La estructura de la red ante la que se enfrenta el servidor es esta:

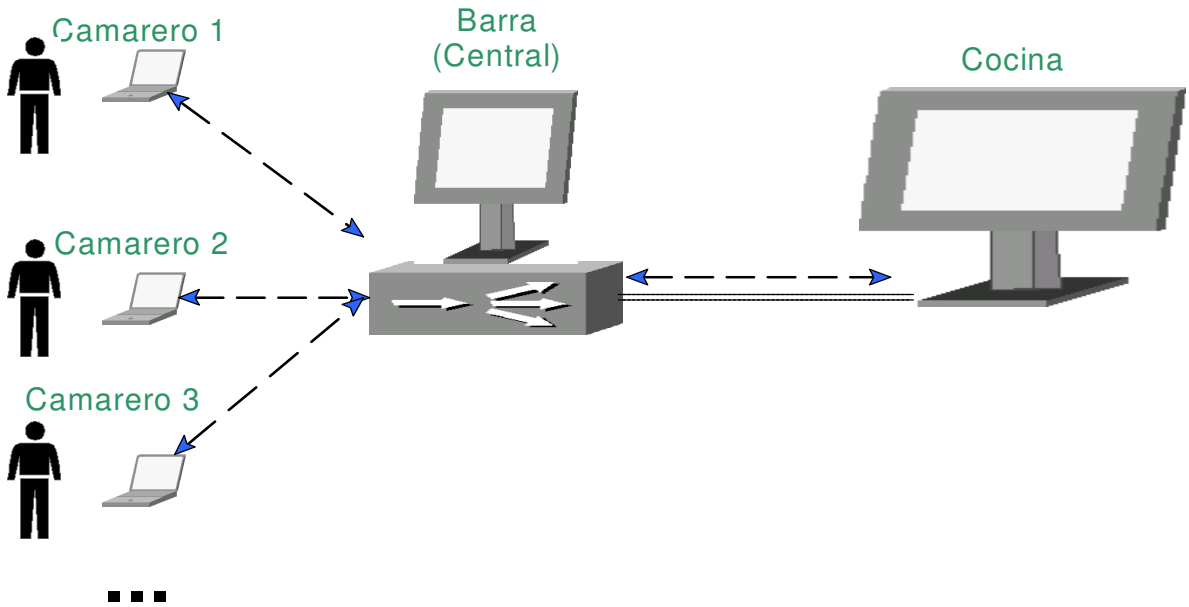


Figura 5.31 Distribución de la red de ordenadores

5.6.3. Funcionamiento.

La aplicación servidor, corre junto con el programa barra, en el ordenador central, que es el que controla las comunicaciones y hace de intermediario en la mayoría de los casos. Cuando el programa barra termina, el servidor también muere, así que ambos corren como si fuera un único programa a los ojos del usuario, ya que el servidor no es visible para el mismo.

Los mensajes y ficheros más importantes que debe servir y reenviar son los siguientes:

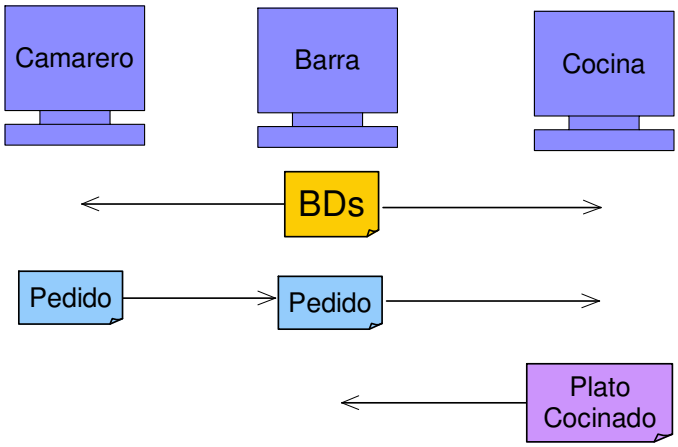


Figura 5.32 Envío de ficheros entre hosts.

- Bases de Datos. Todas las máquinas de la red del restaurante necesitan usar la base de datos, en particular, algunas de sus tablas en diferentes momentos de la ejecución. Pero es importante que la base de datos sea siempre la misma, aunque cada máquina contenga una copia. El servidor se encarga de que todas estas copias siempre estén actualizadas, es decir, que sean siempre iguales en todas las máquinas, cada vez que se produzca un cambio. Cuando esto ocurra, y se le notifique al servidor, éste inmediatamente realiza un broadcast del archivo a todas las máquinas de la red.

- Pedido. Cuando el camarero ha tomado nota de lo que necesitan los clientes, ese archivo con el pedido es enviado a la barra, y de aquí reenviado a la cocina. Este envío no se hace directamente a la cocina porque, por un lado, en la barra necesitan conocer lo que se ha pedido, lo que falta por cocinar, y lo que ya se puede servir; y por otro lado, por la arquitectura típica de los restaurantes. ¿Qué quiero decir con esto? Pues que la cocina suele estar en otra habitación separada de los clientes, y las comunicaciones inalámbricas entre camareros y cocina podrían fallar en ciertas ocasiones debido a la presencia de muros. Es una mejor opción diseñar el host de la barra como el centro de las comunicaciones, y que se conecte con la cocina de forma cableada, y con los camareros de forma inalámbrica para posibilitar la comunicación.

- Platos cocinados. Cada vez que la cocina termina un plato, se conecta la cocina con el servidor de la barra para enviarle un mensaje diciéndole que ha terminado un plato. Por supuesto también le indica cual de ellos, ya que es imposible saber cual es el último plato que ha salido, porque el orden de los platos que imponen las marcas temporales para el cocinero no es una obligación, sólo una ayuda, ¡menuda esclavitud sería si no!

Para que el servidor pueda atender a todos los clientes a la vez, en el caso de que se produzcan varias peticiones de forma concurrente, se ha elegido el componente IndyTCP que posibilita atender a varios clientes con un solo servidor. Cuando un cliente se conecta, este thread (el hilo de ejecución del servidor) transfiere todas las operaciones de comunicación a otro thread (un nuevo hilo de ejecución). Esta arquitectura es muy buena porque la aplicación cliente podrá conectarse en cualquier instante, incluso si hay varios clientes en paralelo intentando conectarse al servidor. Este componente Indy, ya viene con un montón de opciones, y lo importante es saber aprovecharlas y configurarlas de forma adecuada.

Dentro de los servidores multihilo se presentan dos modelos: IdThreadDefault y IdThreadPool.

IdThreadDefault se utiliza es el thread estándar, el más usado en las aplicaciones. Su característica principal es que para cada conexión crea un nuevo thread aparte. En este sistema se ha optado por éste ya que no hay un gran número de clientes (sólo hablamos de unos cuantos camareros y una aplicación en cocina).

IdThreadPool, el segundo modelo de thread, está diseñado para servidores con mucha carga. En entornos con una gran aglomeración de clientes, crear y destruir threads consume muchos recursos, por lo que el servidor se puede ver muy deteriorado en su rendimiento. En cambio, con este modelo, se especifica el número de threads que se han de crear cuando el servidor comienza. Cuando la conexión comienza, un thread de los que están libres se utiliza, pero cuando la conexión termina, el thread no se destruye.

5.7. Cocina

5.7.1. Descripción.

La aplicación de cocina consiste en una lista ordenada de platos y organizada por tiempos, con varios campos más de información acerca de las preferencias del usuario sobre el plato, o el número de platos que hay que realizar. Además incorpora algunas funcionalidades más como son envío de mensajes a los otros equipos notificando por ejemplo que un producto se ha agotado, o que hay demasiada carga en cocina, para que se anule un producto, o para que los próximos pedidos se hagan con mayor margen de tiempo, respectivamente.

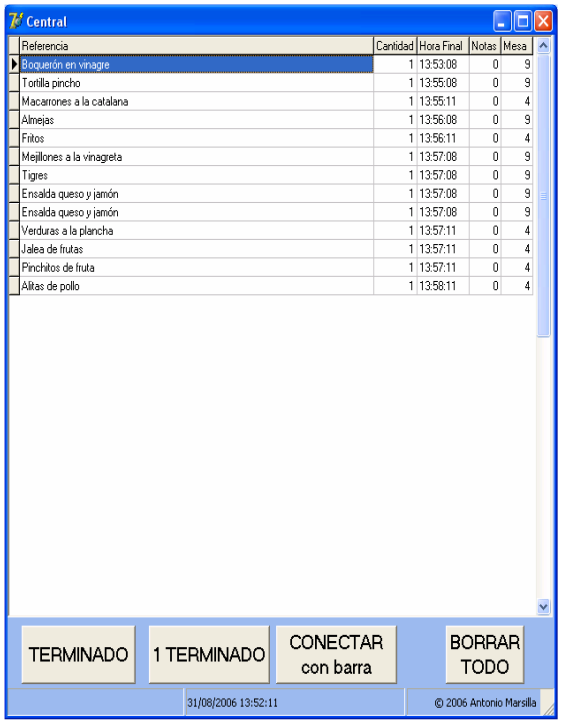


Figura 5.33 Aplicación cocina

Esta aplicación es la más simple de utilizar, quizás la menos vistosa, pero la que ha dado a luz a este proyecto, ya que el fin primero de toda esta idea fue el de ayudar a los cocineros organizándoles temporalmente los pedidos, y la comunicación con sus camareros.

De todas formas, la relativa simplicidad de la cocina se debe a que por la naturaleza de su trabajo, los cocineros no pueden andar manejando aplicaciones en las que tengan realizar un gran de número de clics y desplazamientos; sus manos casi siempre están ocupadas, así que un simple clic es la forma más práctica de ayudarles.

Los cocineros también podrán, con este equipo, pedir un margen de tiempo más amplio para preparar los platos y, también, que los nuevos clientes conozcan la demora o tiempo de espera antes de que se les empiece a servir.

Es cuanto menos indeseable y engañoso, entrar en un restaurante en el que realizas un pedido rápidamente, pero luego compruebas que el servicio es muy lento y los platos no aparecen por ningún lado, hasta pasada media hora. Mucho más honesto y correcto sería comunicarle al cliente que, debido a la gran carga de trabajo de ese momento, su pedido no se puede servir y que deberá esperar al menos unos 20 minutos, para que la cocina esté más descargada. Así el cliente tiene toda la información para decidir el quedarse o irse y no se siente engañado. Pues esto lo hace automáticamente este programa gracias a su temporización.

5.7.2. Estructura.

Esta aplicación se compone de 4 unidades en Delphi, de las cuales la principal es Main.

Existe, como en los anteriores, una unidad llamada Funciones que alberga métodos y funciones que descargan al programa principal de código, para que sea mucho más claras su lectura y comprensión, y su posterior remodelación.

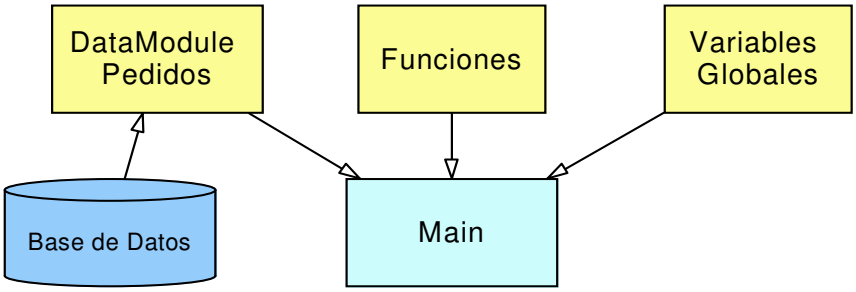


Figura 5.34 Estructura del programa "Cocina"

Sigue en la misma dinámica de las anteriores el hecho de utilizar una unidad llamada Variables globales para que los valores y los cambios que se produzcan en esas variables sean visibles y modificables por todas las unidades de este proyecto.

La Base de Datos en este caso se utiliza por medio de una fórmula que ya se vio en el capítulo cuatro de Bases de datos, que consiste en crear un módulo de datos. Este módulo es una forma aparte, que accede a las bases de datos y que posibilita, a diferencia de los programas anteriores, que cualquiera de sus ventanas pueda acceder a las tablas de la base de datos. Esto es porque ahora es visible por todos, y antes sólo la veía la unidad que la contenía que era la principal.

Es otra forma más de programar el acceso a las bases de datos, pero mucho más útil si se pretende que ahora (o en un futuro) varias de las forms (ventanas) que componen la aplicación puedan acceder a la base de datos. Aunque ahora no contiene nada más que una unidad interesada en la BD, es más que probable que incorpore nuevas unidades de código en una futuras versiones del programa. Todo esto se podría hacer para conseguir mayor vistosidad y mayores funcionalidades, como por ejemplo contener una herramienta para la gestión y modificación de la tabla de productos, o un sistema de avisos más eficaz y visual.

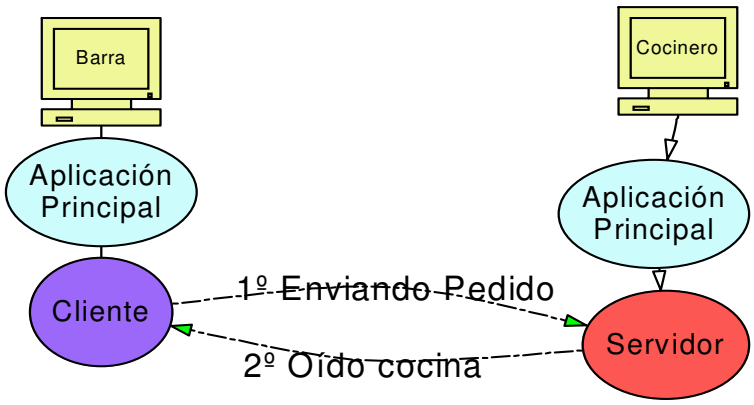


Figura 5.35. Servidor de Cocina recibiendo pedido

Por otro lado hay que decir que, en la parte de las comunicaciones, la estructura es bastante similar a la que utilizaba el servidor de la barra. Una aplicación servidor corre en el mismo ordenador de la mano de la aplicación principal ya que comienza y acaba cuando la primera finaliza. Su función principal del servidor es escuchar la red a la espera de pedidos que le envíe la barra para dárselos a la aplicación principal, y que ésta los cargue.

5.7.3. Funcionamiento.

El programa comienza ejecutando la pantalla principal y la única que tiene. En segundo plano (de forma invisible) se ejecuta el servidor de la cocina. Como se comentó anteriormente, el servidor comienza y acaba su ejecución cuando lo hace la aplicación principal.

La pantalla está diseñada con una lista que la ocupa casi totalmente, más unos cuantos botones en su parte inferior. El funcionamiento es muy simple: se trata de una lista de productos ordenados en filas, de manera que el que aparece más arriba en la lista es el que debería salir antes y el que está abajo el último que debería salir.

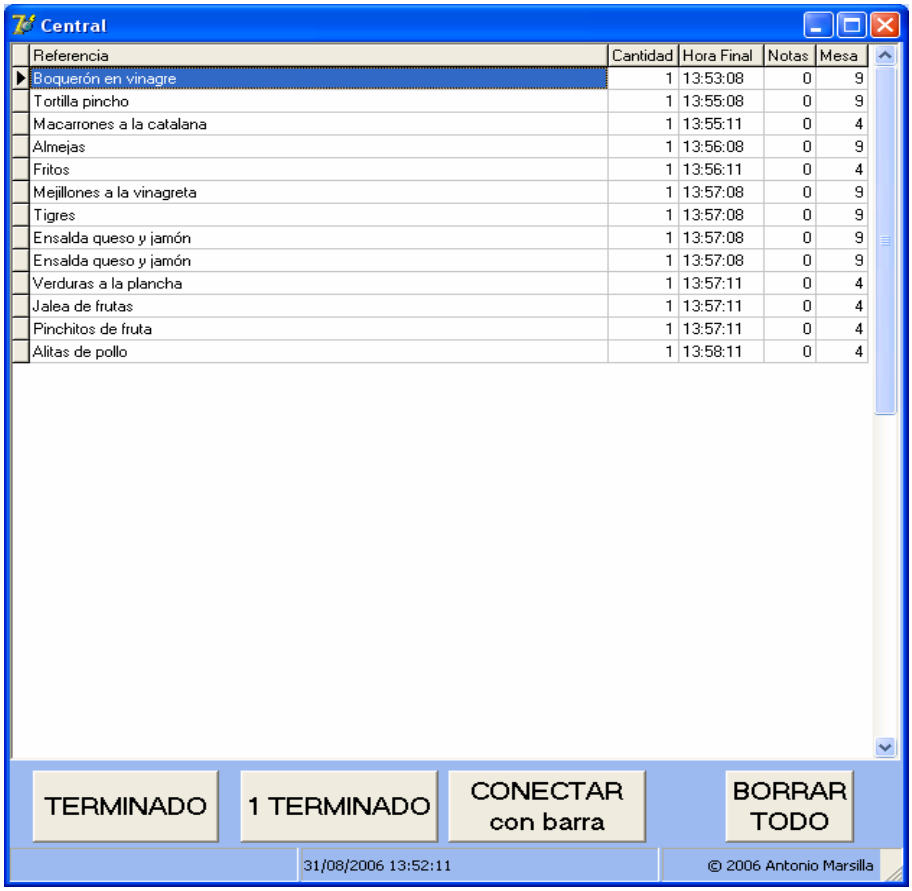


Figura 5.36 Ventana principal del programa "Cocina"

Cuando la lista se llena tanto que no caben todos los productos en pantalla, unas barras de desplazamiento le aparecen a un lado para que podamos desplazarnos a ver los de abajo, aunque normalmente no es necesario hacerlo ya que los que interesan son los de arriba, que son los que corren más prisa. Cuando un nuevo pedido le llega, el cocinero no tiene que tocar ningún botón, la lista con los productos se actualiza automáticamente, y se ordenan por tiempos, como ya se ha explicado.

Las columnas que le siguen son:

- Cantidad. Número de productos que hay que cocinar de ese mismo plato.
- Hora final. Es la hora límite que se le ha indicado al cocinero para que la organización temporal funcione de acuerdo a lo previsto.
- Notas. Aquí va información del plato acerca de variaciones que introducen los clientes, como por ejemplo “sin sal”, “muy hecho”, “con salteado de verduras”, “sin patatas”, etc.
- Mesa. El número nos dice a qué mesa corresponde ese plato.

Toda esta lista de tiempos no es más que una ayuda para el camarero a la hora de organizarse y, a la vez, un punto extra en Marketing ya que los clientes pueden conocer más o menos el tiempo en el que estarán listos sus platos. Pues bien, como se ha dicho, esta organización flexible posibilita que el cocinero termine antes un plato que esté el 5º en la lista que el que esté primero. Para decir que lo ha terminado sólo tiene que pinchar sobre él y pulsar el botón “Terminado”

Llegados a este punto podemos diferenciar entre dos botones, “Terminado” y “1 terminado”. Cuando pinchamos sobre el primero, la fila se elimina de la lista, ya fueran cinco, tres o un sólo plato. En cambio “1 Terminado” decrementa la cantidad de platos en esa fila, y si es sólo un plato, pues lo elimina.

Otros dos botones más se presentan en la parte inferior: “Conectar” y “Borrar todo”. El primero sólo es necesario utilizarlo cuando encendemos la cocina y todavía no se ha encendido la barra. Al encenderse la cocina automáticamente intenta conectarse con la barra, pero ya que no puede por estar desconectada, tendremos que hacerlo manualmente más tarde. “Borrar todo” nos da la posibilidad de eliminar pedidos de días anteriores que se han quedado grabados, y finalmente no se sirvieron. Ya que podría haber un apagón, se necesita que el programa guarde todos los pedidos y los vuelva a recuperar cuando se enciende el programa y; si así lo deseamos, borrar manualmente los pedidos.

Capítulo 6.

Elementos complementarios.

6.1. Tabla de productos

6.1.1. Descripción.

La tabla de productos está creada con el fin de dotar a todos los equipos de la red de una base de datos común, con información modificable y actualizable referente a los platos, como son nombre, precio, producto agotado, etc.; que posibilite adaptarse rápida y automáticamente a los constantes cambios en la carta de productos de un restaurante (o bar). Y por otro lado, guardar los datos de tiempos de preparación y los márgenes entre platos, proveniente de la experiencia de cocineros y restauradores.

Las aplicaciones que se conectan a la tabla necesitan disponer de cierta interactividad con el usuario, sin tener que modificar la programación del software, por lo que la tabla posibilita que haya una cierta interactividad entre el usuario y el programa al adaptarse a todos los cambios de la carta. Con esta, las aplicaciones pueden cambiar el número, la apariencia y la información que contienen sus botones, atendiendo a cada producto.

En esta base de datos se pueden borrar, leer, introducir, cambiar datos por parte del usuario, de forma fácil, valiéndose de la aplicación software desarrollada "gestor de productos", que accede a la base de datos y la controla en su totalidad.

Recoge tanto tiempos, como nombres de productos, como tiempos de preparación, el margen de tiempo entre que se sirven los entrantes, los primeros, y los segundos; así como si el producto está agotado.

6.1.2. Estructura.

El formato de la tabla es Paradox, un formato nativo de Delphi. La tabla de productos está creada con el fin de dotar al sistema de una rápida información común para todos los equipos de la red, que facilite encontrar toda la información referente a los platos y productos del pedido (como el precio, el nombre, o el tiempo de preparación; y además, de interactividad al sistema. ¿Interactividad? Sí, porque la parte novedosa del programa es que una vez añadido un nuevo producto a la base de datos, aparece automáticamente en el programa "Camarero" un nuevo botón correspondiente a esa nueva entrada en la tabla de productos, y no sólo eso, sino que cuando uno se agota, su botón asignado al mismo desaparece.

Se ha ordenado el formato en seis columnas, más la columna implícita que utiliza delphi para los índices (llaman a cada fila con un número). Pasamos a continuación a comentar el significado de cada columna en particular:

Productos	Código	Referencia	Precio	Preparacion	Margen	Agotado
34	5012	Pulpitos	4,00	7,00	0,00	False
35	6001	Ensalada dulce	2,50	5,00	0,00	False
36	6002	Ensalda normal	2,50	5,00	0,00	False
37	6003	Ensalda queso y jamón	2,50	5,00	0,00	False
38	6004	Sopa de cocido	4,00	4,00	0,00	False
39	6005	Sopa de pescado	3,75	4,00	0,00	False
40	6006	Entremeses frios y calientes	3,00	3,00	0,00	False
41	6007	Fritos	5,00	5,00	0,00	False
42	6008	Macarrones a la catalana	5,00	4,00	0,00	False
43	6009	Alubias con chorizo	5,50	8,00	0,00	False
44	6010	Menestra de verduras	5,50	6,00	0,00	False
45	6011	Paella de Marisco	6,00	8,50	0,00	False
46	6012	Arroz campero	6,50	8,50	0,00	False
47	7001	Verduras a la plancha	4,00	6,00	0,00	False
48	7004	Alitas de pollo	4,00	7,00	0,00	False
49	8001	Manzanas maceradas	5,00	6,00	0,00	False
50	8002	Jalea de frutas	5,00	6,00	0,00	False
51	8003	Pinchitos de fruta	4,00	6,00	0,00	False
52	8004	Sandía rellena	5,00	6,00	0,00	False
53	8005	Sandía	2,00	2,00	0,00	False
54	13001	Mousse de limón	3,50	4,00	0,00	False
55	13002	Cassata	3,00	1,50	0,00	False
56	13003	Tarta de Crema	3,00	1,50	0,00	False
57	13004	Tarta de turrón y chocolate	3,00	1,50	0,00	False

Figura 6.1 Tabla de productos.

Código. Es de tipo entero, es el campo clave de la tabla, y se distribuye de la siguiente forma:

Familia	*Subconjunto*	*Producto*
00	*0*	*00*

Figura 6.2 Estructura del campo código.

Para la familia de productos hemos utilizado dos cifras, aunque realmente se pueden utilizar más, ya que es un entero, y podríamos llegar a utilizar varias. Pero es bastante improbable que el número de familias sea mayor de 30 ó 40. La distribución de familias de productos que he realizado es esta:

Menús	*01*
Tapas	*02*
Bollería	*03*
Bocadillos	*04*
Entrantes	*05*
Primeros	*06*
Segundos	*07*
Postres	*08*
Café Te e Infusiones	*09*
Bebidas	*10*
Vinos	*11*
Alcohol	*12*
Helados	*13*

Figura 6.3 Numeración de las familias de productos.

La cifra subconjunto es una previsión de futuro, ya que se ha querido con esta ofrecer dos posibilidades:

- Prever futuras ampliaciones del programa, para dotar de un funcionamiento más práctico y rápido al diferenciar entre subconjuntos de productos. Por ejemplo en

una carta de vinos, podíamos separarlos en rosados, tintos, blancos…, y, gracias a esto, poder separarlos en diferentes zonas dentro de su panel.

- Aumentar la cantidad límite de productos que se le pueden introducir a la base de datos dentro de una tabla, para poder llegar hasta 1000 productos dentro de cada familia. También estamos evitando volver a usar códigos por falta de números. Así podrían seguir estando en lista, antiguos productos que han sido temporalmente eliminados de la carta, sin tener que tirar a la basura el arduo trabajo de introducción de datos en las bases de datos.

Ambas posibilidades no son incompatibles, ya que podemos también usar, por ejemplo las cifras 1,2 y 3 para un subconjunto, la 4, 5, 6 para otro… y así tener 300 productos por subconjunto.

Referencia. Contiene una cadena de caracteres que corresponde al nombre del producto, y que como máximo tiene 55 caracteres.

Precio. Es de tipo number en tablas Paradox, el equivalente al tipo double común a la mayoría de lenguajes. Guarda el precio en Euros de cada producto elaborado. Éste, junto con el descuento que puede realizar el camarero, formará el total a cobrarle al cliente por cada plato.

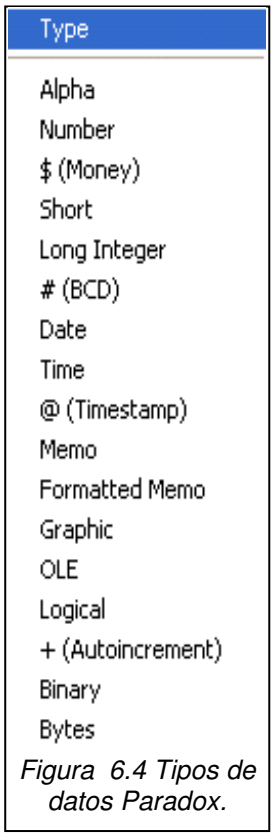


Figura 6.4 Tipos de datos Paradox.

Preparación. También de tipo number, guarda el tiempo que en promedio suele tardar en prepararse un cierto plato. Si su valor es 0, y no hay un margen de tiempo para servir (no es un primero ni un segundo), se sirve inmediatamente. Se ha elegido este tipo numeric frente al tipo "time" porque es mucho más rápido para el usuario a la hora de introducir los valores, ya que el tipo time requería introducir también las horas, los minutos y los segundos. Aquí para introducir sólo 7 minutos y medio debemos poner 7,5 (con el tipo time debiéramos haber introducido 00:07:30). Y ya que no se necesita tanta precisión para usar segundos, ni llegamos a mayor precisión que medios minutos, se optó por este tipo.

Margen. Al igual que los anteriores campos, es de tipo number. Denota el tiempo normal que deben esperar los camareros para servir un plato, desde que se ha realizado un pedido (Por ejemplo, entre el primero y el segundo plato, y entre el segundo y el postre).

Agotado. De tipo booleano, y nos indica si hay actualmente existencias de este producto. Cuando éste se agota, el usuario accede a la tabla de productos mediante el "gestor de productos", y automáticamente, ese artículo desaparece de la carta del programa del camarero. Su valor por defecto es false.

	Field Name	Type	Size	Key
1	Código	S		*
2	Referencia	A	55	
3	Precio	N		
4	Preparacion	N		
5	Margen	N		
6	Agotado	L		

Figura 6.5 Campos de la tabla de productos, con su tipo, tamaño y clave.

6.2. Tabla de pedidos

6.2.1. Definición.

El objetivo de la tabla de pedidos es el de recoger, en una lista, los productos (platos y bebidas) que el cliente le pide al camarero, junto con alguna información extra.

Esta tabla es la unidad de intercambio de información entre aplicaciones respecto al pedido del cliente. Esto quiere decir que contiene toda la información del pedido del cliente y viaja por la red para que el pedido le llegue tanto a la barra como a la cocina.

La tabla la crea la aplicación camarero, y es la lista que se muestra en su ventana principal, más algunos campos que no se muestran. Al igual que en las otras tablas, se puede modificar, borrar, y añadir filas, pero en este caso la modificación corresponde no sólo al camarero, sino a la barra también. La barra la copia, y tras procesar los productos que debe enviar a cocina, manda una nueva lista a cocina sólo con esos productos. Pero se queda con una copia que le servirá para llevar un control de los

6.2.2. Estructura.

La tabla de pedidos tiene los siete campos que se ven en la siguiente figura. Algunos de ellos (Código, Producto y Precio) son una copia de la tabla de productos. A continuación se comentan los nuevos campos que contienen esta tabla que no tenía la anterior:

Figura 6.6 Tabla de pedidos

Cantidad. Es de tipo entero (long integer en Paradox), y recoge el número de platos (raciones) que corresponden a ese producto. Esto descarga a la tabla de tener un gran número de filas, innecesarias por otra parte, y evita tener que volver a escribir por ejemplo la misma nota sobre el mismo plato que ha pedido otro cliente de la misma mesa.

Nota. De tipo entero, su valor está relacionado con el código de la tabla de notas, donde se recogen las variaciones que hacen los clientes acerca de los platos ("poco hecho", "sin sal"…). Para conocer qué significa ese valor numérico, hay que ir consultarlo en la tabla de notas.

Descuento. De tipo punto flotante (number en Paradox), su valor se le restará al campo de precio, y con estos dos campos construiremos el Total. Tanto este campo, como el campo total, no se muestran en la lista del programa camarero. Se ha incluido este campo como previsión de una futura versión, que amplía algunas opciones como la de realizar descuentos en los pedidos.

Total. Al igual que su predecesor es de tipo punto flotante (number en Paradox), y su valor se obtiene de la siguiente fórmula: Precio x Cantidad – Descuento = Total.

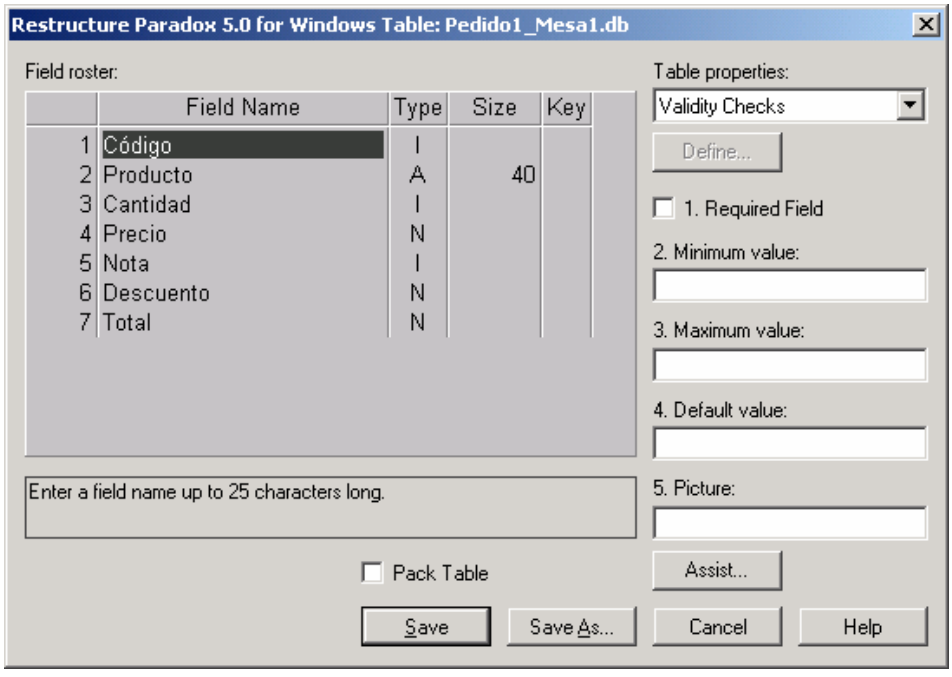


Figura 6.7 Formato de los campos de la tabla "Pedidos"

6.3. Tabla de notas

6.3.1. Descripción.

La tabla de notas está creada con el fin de recoger todas las observaciones que hacen los clientes en cuanto a las preferencias de los platos; de forma ordenada, numerada, y rápida; y de manera que facilite el trabajo a cocineros y camareros, ya que en décimas de segundo tendrá a su alcance toda una gran lista de notas, que de otra manera hubiera tenido que escribir, y luego aclararle al cocinero. Ahora el camarero añade la nota sólo pinchando un botón de la lista desplegable, que ambos tienen, de forma rápida y detallada cada observación que hacen los clientes en el pedido (tales como poco hecho, sin sal, con patatas, etc.).

En esta base de datos, al igual que en la anterior tabla, se pueden borrar, leer, introducir, cambiar datos por parte del usuario, de forma fácil, pero esta vez valiéndose directamente de la aplicación "Camarero", que accede a la base de datos y la controla en su totalidad.

Recoge en la base datos de notas sólo el número (o código) correspondiente a la nota, y la nota en sí, ordenadas según tipos y clases de notas, de la forma más flexible y actualizable.

6.3.2. Estructura.

Esta tabla de notas es más simple y fácil que las anteriores, ya que sus campos son sólo dos: código y referencia.

CÓDIGO. Es un número de tipo entero, que atiende a la siguiente estructura:

Tipo	*Subconjunto*	*Nota*
00	*0*	*00*

Figura 6.7 Estructura del campo código.

Es apreciable como la estructura es idéntica a la anterior, ya que se comprobó un buen resultado y, aunque necesitemos un menor número de notas, la previsión de crecimiento siempre es deseable. Así, en la versión inicial de la aplicación, se puede decir que la cifra subconjunto toma siempre un valor 0, ya que no ha sido necesario utilizar más de 100 notas por subconjunto, y tampoco, se han hecho subconjuntos dentro de los tipos de notas.

Referencia. Contiene una cadena de caracteres que corresponde al nombre de la nota, y que como máximo tiene 55 caracteres.

6.4. Gestor de productos

6.4.1. Definición.

Esta herramienta está creada con el fin de actualizar y mantener la base de datos de productos por parte del usuario, de forma más fácil aún que la que nos ofrece Delphi con su Database Desktop. El usuario normal necesita aplicaciones rápidas, sencillas y específicas a sus necesidades, y esto no lo cumple la herramienta Database Desktop de Delphi.

En cambio, con esta aplicación se pueden crear nuevas entradas en la tabla, modificar las existentes, añadirle una foto al producto, o marcarlo como agotado. Se le pueden añadir fotos o dibujos a cada producto, para que luego sea mucho más fácil y visual utilizar el programa Camarero. Soporta los dos formatos gráficos más extendidos que son JPG y BMP (su explicación se adjunta en los anexos), para que sea mucho más simple encontrar imágenes para los platos y bebidas.

6.4.2. Estructura.

La estructura del programa está representada en la figura siguiente:

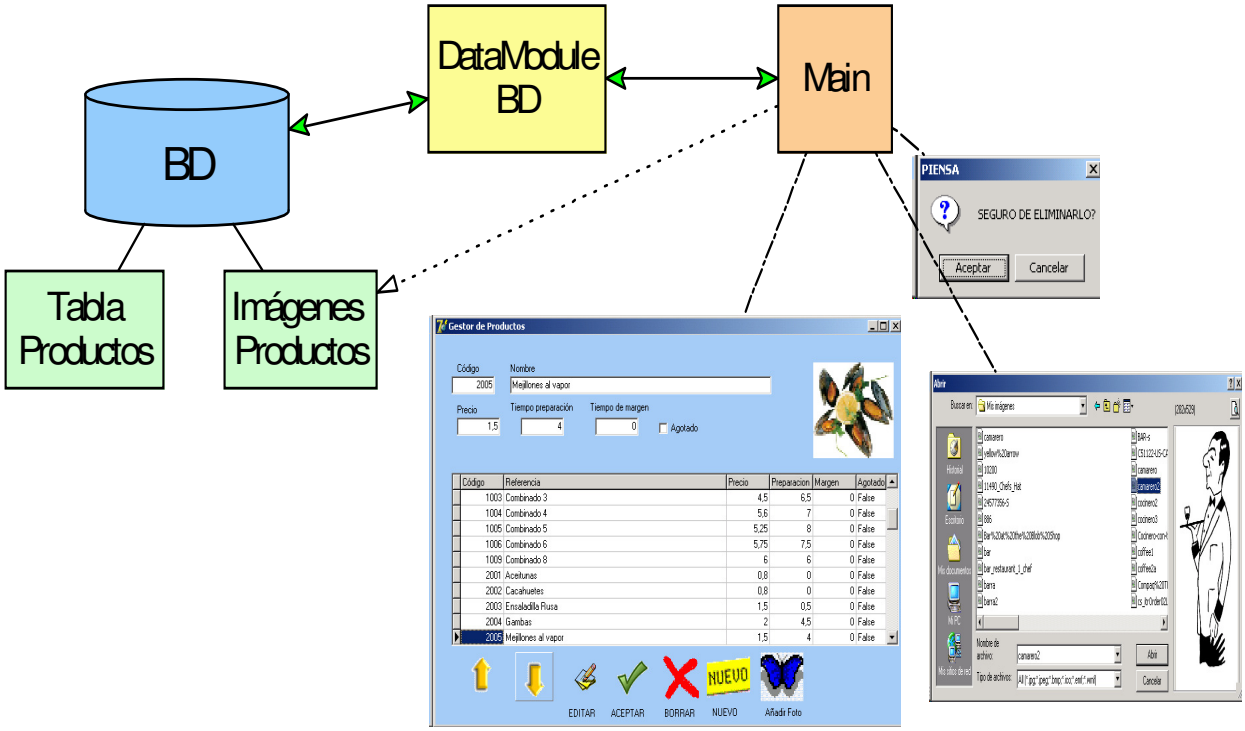


Figura 6.8 Estructura de archivos y ventanas del "gestor de productos"

Dos unidades de código componen la totalidad del programa "Gestor de Productos", de los cuales el único que tiene componentes y ventanas visibles es la unidad Main. El otro fichero, DataModuleBD, está construido para compactar en un archivo, fuera

del principal, el manejo y la conexión con las bases de datos. Si el programa creciera, y se le añadieran nuevas ventanas y unidades, todos podrían acceder a la base de datos a través de este módulo de datos. En cambio, si no se hubiera utilizado este archivo intermedio, cada uno se hubiera visto obligado a manejar las bases de datos internamente. Se hubiera tenido que añadir componentes Table, Date, Query, etc. a cada fichero de código, pudiendo producirse más conflictos por haber un número mayor de componentes intentando acceder a la misma Base de Datos.

En la base de datos, además de las tablas, se encuentra también una carpeta que contiene todas las imágenes que corresponden a los productos. La unidad "Main" podrá acceder a ellas, guardar nuevas, modificar las existentes o borrarlas. Se les ha incluido en el esquema dentro de la base de datos porque aunque realmente no es ninguna tabla, sí que están completamente ligadas a la tabla de productos. Se podría por ejemplo haber incluido un campo gráfico a la base de datos que contuviera estas imágenes, pero se desechó esta idea por ser mucho más transportable y fácilmente modificable una carpeta con las imágenes en formato BMP, que una fichero de Tabla de Base de Datos del cual no se pueden extraer tan rápida y fácilmente las imágenes.

6.4.3. Funcionamiento.

El funcionamiento es más que sencillo, pero no por ello menos práctico, ya que cumple su función de forma rápida. Se trata de que se puedan añadir nuevos productos a la tabla de la base de datos, modificar los existentes, sus datos, y añadirle gráficos y fotos a los productos, e intuitivamente se puede adivinar como funciona.

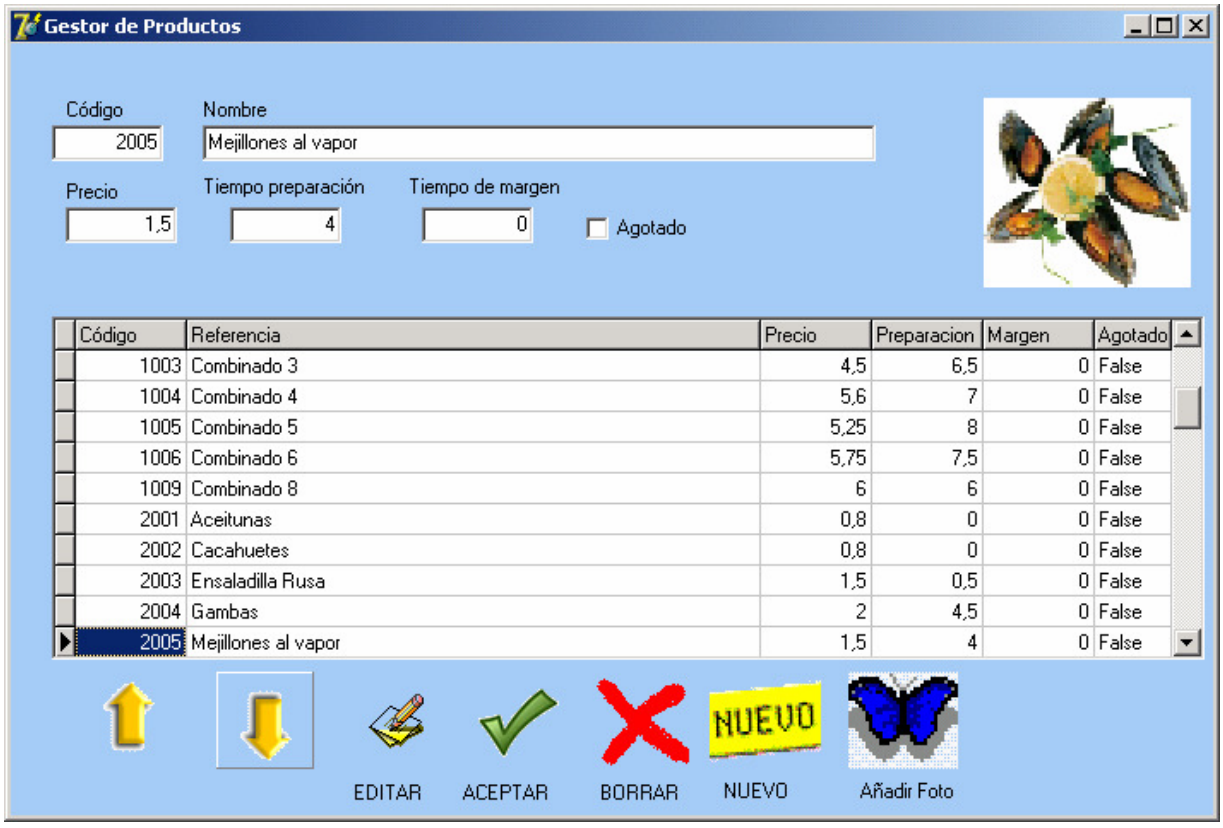


Figura 6.9 Ventana principal del programa "Gestor de Productos"

De arriba abajo la ventana está compuesta por casillas con la información de cada campo de una fila, más la imagen de esa fila (a la derecha), una lista con toda la información en la parte intermedia de la ventana, y una botonera inferior. Con **las flechas** de la botonera inferior podemos desplazarnos por la lista producto a producto. También existe la barra de desplazamiento que hay en la izquierda para moverse mucho más rápido por la lista de los productos. Cada vez que pulsamos las flechas, la lista se desplaza y el gráfico cambia. El gráfico sólo cambia con estas flechas, por eso puede ocurrir que nos desplacemos con la barra de la derecha, pinchemos sobre un producto y el gráfico no cambie. Para que se actualice tenemos que pinchar en las flechas.

Las cajas de arriba dan la información de la fila seleccionada pero no se pueden modificar hasta que pulsemos el botón **Editar**. No se puede editar directamente en la lista, sino escribiendo en las cajas. Cuando todo esté correcto Pulsaremos Aceptar. Aquí hay que aclarar que el programa puede fallar si no somos cuidadosos con el número de código que introduzcamos, ya que es un campo clave de la tabla, y si intentamos introducir otra fila con el mismo código de una ya existente, el programa nos dará un mensaje de error.

Para eliminar un producto completamente sólo hay que pulsar sobre **Borrar**. Una ventana emergente como la de la figura de abajo nos aparecerá preguntándonos si estamos seguros de que queremos borrarla. Si pulsamos "Aceptar" la fila quedará totalmente eliminada, así como la imagen que le correspondía.

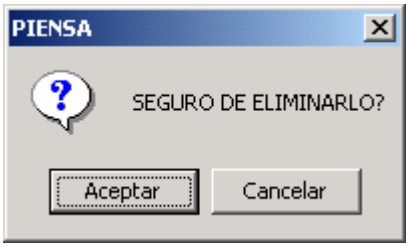


Figura 6.10 Confirmación de borrado de un producto

Llegando al tema de las imágenes, hay que decir que se escogieron los dos formatos más extendidos y estandarizados que hay en Internet (JPG y BMP), para que pudieran usarse en el programa. Pero implícitamente el programa realiza una conversión hacia el formato BMP ya que los botones utilizados en las aplicaciones de Delphi no soportaban de momento el otro formato. Así que se tuvo que implementar una función que convirtiera JPGs a BMPs. Además también cambian su tamaño para adaptarse a los botones que se usan en el programa camarero, y para que no ocupen mucho espacio en memoria (BMP es un formato no comprimido por lo cual ocupa mucho más que JPG).

Para añadir una imagen a un producto o modificar la existente se pulsa el botón "Añadir Foto" (identificado con una mariposa). Inmediatamente se abre una nueva ventana en la que podemos buscar una imagen.

Lo práctico de esta ventana es que es un navegador completo como el que utilizamos normalmente para cualquier otra aplicación en la que guardamos un archivo pero especializado en las imágenes. Sólo se ven los archivos que corresponden a gráficos, dibujos o fotos, y además poder visualizarlas en una vista previa a la derecha de la ventana. Se muestra en la figura siguiente.

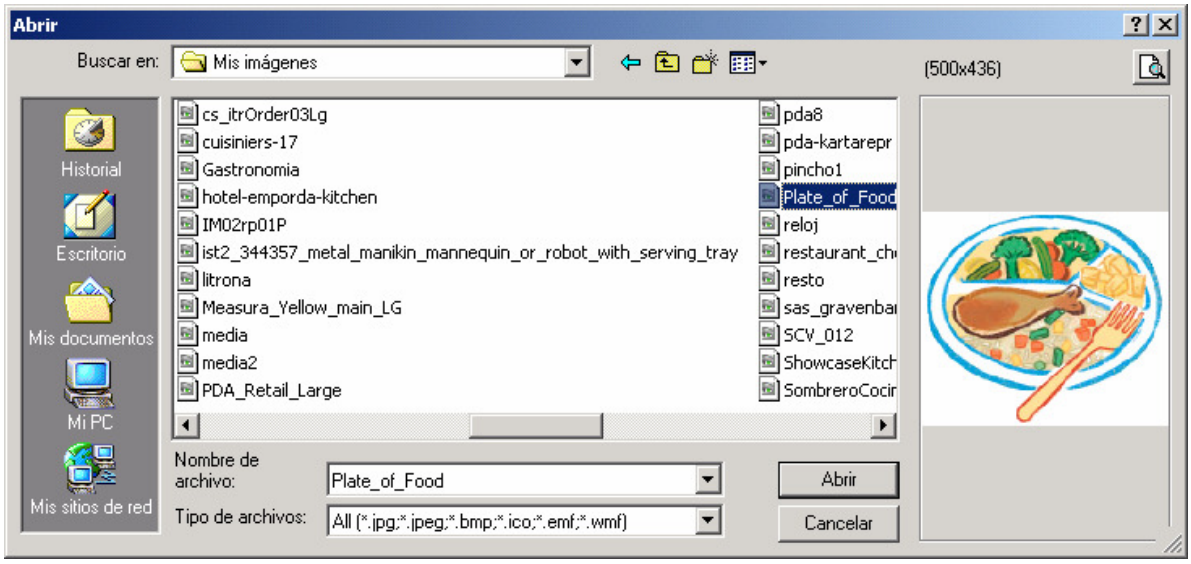


Figura 6.11 Navegador para elegir una imagen

Si la vista previa es tan pequeña que queremos verlo en tamaño natural, podemos pinchar sobre el botoncito que hay en la esquina superior derecha, y la foto se despliega en tamaño real (Figura 6.12).

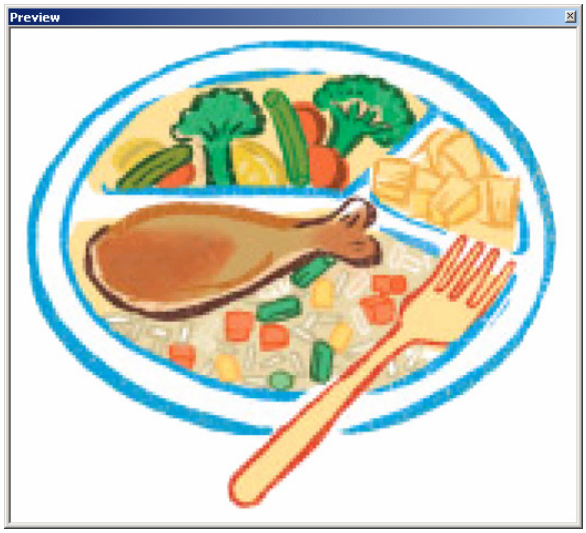


Figura 6.12 Vista previa del gráfico en tamaño real.

Capítulo 7.

Protocolo UDP y TCP

El protocolo TCP tiene la robustez, la confiabilidad y las funcionalidades propias de un protocolo de transporte orientado a conexión, pero a veces resulta demasiado complejo. Por ejemplo, cualquier transmisión de información TCP requiere como mínimo el intercambio de seis mensajes para establecer la comunicación y terminarla, además mientras una conexión existe ocupa una serie de recursos en el host.

7.1. Protocolo UDP

7.1.1. Introducción

Para casos en los que no es necesario tanto control de los datos enviados la Capa de Transporte implementa también el **Protocolo de Datagrama de Usuario (UDP)**, protocolo no confiable y no orientado a conexión para la entrega de mensajes discretos. En este caso los paquetes enviados mediante el protocolo IP reciben el nombre específico de datagramas, y estos se envían y ya está, no se realiza una conexión definida entre los host ni un control de los paquetes enviados y recibidos. Los datagramas se rutean independientemente, por lo que deben llevar la dirección completa de destino.

UDP es un protocolo estándar con número 6 de STD. Este protocolo se describe en el RFC 768 - Protocolo de Datagrama de Usuario. Es simple, eficiente e ideal para aplicaciones como el TFTP y el DNS. Una dirección IP sirve para dirigir el datagrama hacia una máquina en particular, y el número de puerto de destino en la cabecera UDP se utiliza para dirigir el datagrama UDP a un proceso específico en dicha máquina. La cabecera UDP también contiene un número de puerto origen que permite al proceso recibido conocer como responder al datagrama.

Es una alternativa al TCP en aquellos casos en los que no sea necesaria tanta complejidad en el envío, por lo que se usa cuando una entrega rápida es más importante que una entrega garantizada, o en los casos en que se desea enviar tan poca información que cabe en un único datagrama. Así, una de sus utilidades más comunes es el envío de mensajes entre aplicaciones de dos host.

UDP no admite numeración de los datagramas, factor que, sumado a que tampoco utiliza señales de confirmación de entrega, hace que la garantía de que un paquete llegue a su destino sea mucho menor que si se usa TCP. Esto también origina que los datagramas pueden llegar duplicados y/o desordenados a su destino. Por estos motivos el control de envío de datagramas, si existe, debe ser implementado por las aplicaciones que usan UDP como medio de transporte de datos, al igual que el reensamblado de los mensajes entrantes.

Es por ello un protocolo del tipo best-effort (máximo esfuerzo), porque hace lo que puede para transmitir los datagramas hacia la aplicación, pero no puede garantizar que la aplicación los reciba.

Tampoco utiliza mecanismos de detección de errores. Cuando se detecta un error en un datagrama, en lugar de entregarlo a la aplicación destino, se descarta.

Cuando una aplicación envía datos a través de UDP, éstos llegan al otro extremo como una unidad. Por ejemplo, si una aplicación escribe 5 veces en el puerto UDP, la aplicación al otro extremo hará 5 lecturas del puerto UDP. Además, el tamaño de cada escritura será igual que el tamaño de las lecturas.

7.1.2. Paquete UDP

Al igual que TCP, UDP usa al protocolo IP para transportar sus segmentos. El formato del segmento UDP es el siguiente:

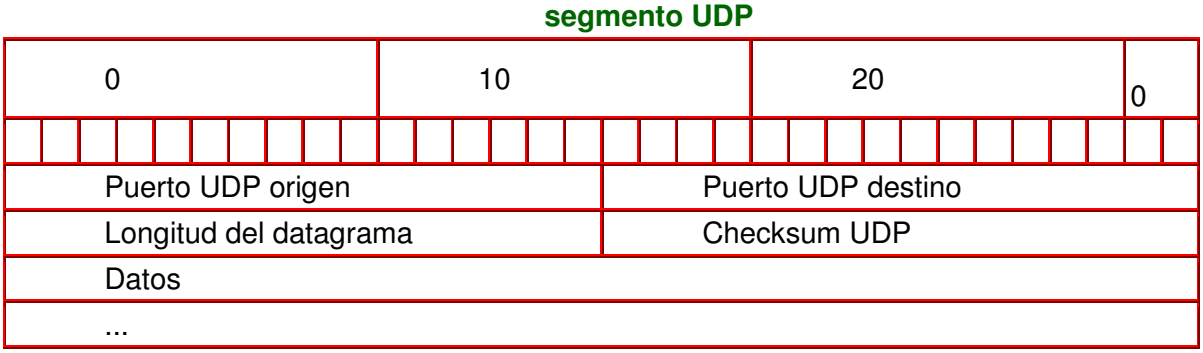


Figura 7.1 Formato Paquete UDP

Cuyos campos principales son:

- Puerto UDP de origen: campo opcional de 16 bits que especifica el puerto del host origen remitente del datagrama.

- Puerto UDP de destino: campo obligatorio de 16 bits que especifica el puerto del host destino al que se envía el datagrama.

- Longitud del datagrama: campo obligatorio de 16 bits que especifica la longitud en bytes del datagrama, incluyendo la cabecera. La longitud mínima es de 8 bytes. En realidad es la longitud del datagrama IP menos el tamaño de la cabecera IP. Como la longitud máxima del datagrama IP es de 65.535 bytes y la cabecera estándar de IP es de 20 bytes, la longitud máxima de un datagrama UDP es de 65.515 bytes.

- Checksum UDP: campo de 16 bits opcional que almacena una suma de comprobación de errores del datagrama, que se calcula a partir de una pseudo-cabecera. Si el checksum es cero, quiere decir que no se calculó el checksum en el origen y que puede ser ignorado. Esto significa que el módulo UDP del computador origen puede o no generar checksums. Si Ethernet es el único tipo de red entre los dos módulos UDP que se comunican, entonces puede que no sea necesario calcular el checksum. Sin embargo, es recomendable habilitar la generación de checksums porque quizás en el futuro un cambio en la tabla de encaminamiento puede enviar los datos a través de un medio menos fiable que Ethernet.

  Si el checksum es válido (o cero), se examina el número de puerto destino y si hay una aplicación escuchando en ese puerto, un mensaje de aplicación queda en la cola de lectura de esa aplicación. Si no hay ninguna aplicación en ese puerto, el datagrama UDP se descarta. Si los datagramas UDP llegan más rápido de lo que

la aplicación puede leerlos, la cola comenzará a llenarse; si se sobrepasa un valor máximo, los datagramas UDP siguientes serán descartados por UDP. UDP seguirá descartando datagramas hasta que haya espacio en la cola.

- Datos: campo obligatorio que contiene los datos que se envían las aplicaciones con el datagrama.

7.1.3. Aplicaciones

Algunos ejemplos de situaciones en las que es más conveniente un servicio no orientado a conexión, en los que es más útil un protocolo del tipo UDP, son:

- **Aplicaciones de tiempo real** como audio o video, donde no se puede tolerar el retardo producido por los ACK.

- Consultas a servidores en que se requiere el envío de uno o dos mensajes únicamente como es el caso del DNS.

- Situaciones en las que se necesita conectar con un ordenador de la propia red, usando para ello el nombre del sistema. Para ello hay que conectar primero con el servidor de red apropiado para que transforme dicha dirección de red en una dirección IP válida, siendo en este caso más apropiado el protocolo UDP.

- Cuando queremos transmitir información en modo **multicast** (a muchos destinos) o en modo **broadcast** (a todos los destinos) pues no tiene sentido esperar la confirmación de todos los destinos para continuar con la transmisión. También es importante tener en cuenta que si en una transmisión de este tipo los destinos enviarán confirmación, fácilmente el emisor se vería colapsado, pues por cada paquete que envía recibiría tantas confirmaciones como destinos hayan recibido el paquete.

Entre los protocolos superiores que usan UDP se incluyen TFTP (Trivial File Transfer Protocol), DNS (Domain Name Server), SNMP (Simple Network Management Protocol), NTP (Network Time Protocol), NFS (Network File System), etc.

7.2. Protocolo TCP

7.2.1. Introducción.

La mayoría de los Sistemas Operativos actuales soportan el multiproceso, mediante el cual varios procesos se pueden estar ejecutando a la vez en una máquina, de los cuales puede haber distintos de ellos comunicándose por red al mismo tiempo con la misma máquina o con máquinas diferentes.

En este entorno, decir que un proceso de la máquina receptora es el destino final de un datagrama concreto enviado por otro proceso de la máquina emisora es una afirmación que presta a confusión, ya que los procesos son dinámicos, se crean y se destruyen constantemente, por lo que un proceso en el ordenador emisor en un momento dado no puede saber qué proceso del equipo receptor es el destinatario de los paquetes que está enviando. Además, para una correcta comunicación necesitamos saber qué función del equipo receptor es la encargada de recibir los paquetes, independientemente del proceso que ha lanzado dicha función.

TCP puede adaptarse dinámicamente a las propiedades de la Internet y manejar fallos de muchas clases. Para obtener los servicios de este protocolo, el transmisor y el receptor deben crear unos puntos terminales de conexión, denominados sockets. Cada socket contiene la dirección IP del host y un número de 16 bits que es de carácter local al host, denominado puerto de protocolo. Los primeros 256 puertos son **puertos bien conocidos**, y se usan para servicios comunes, como HTTP, FTP, etc.

Socket=dirección IP+puerto de protocolo

7.2.2. Conexiones TCP

Una conexión viene definida por dos puntos extremos (sockets), permitiendo TCP que varias conexiones compartan un punto extremo final (conexiones múltiples a la vez), al estar cada una de ellas identificada por una pareja IP-puerto de forma única.

Para obtener un servicio TCP o UDP debe establecerse explícitamente una conexión entre un socket del host transmisor y un socket del host receptor. Para ello se usan determinadas primitivas de llamada a los sockets, entre las que destacan las siguientes:

- **SOCKET**: que crea un nuevo punto terminal de conexión.
- **BIND**: que conecta una dirección local a un socket.
- **LISTEN**: que anuncia la disposición de aceptar conexiones, indicando también el tamaño de cola.
- **ACCEPT**: que bloquea al invocador hasta la llegada de un intento de conexión.
- **CONNECT**: que intenta establecer activamente una conexión.
- **SEND**: que envía datos a través de la conexión.
- **RECEIVE**: que recibe los datos de la conexión.
- **CLOSE**: que cierra la conexión.

Para que se pueda crear una conexión entre dos host es necesario que el extremo servidor haga una **apertura pasiva del puerto**, quedándose a la escucha en el mismo a la espera de peticiones de conexión, mientras que el extremo cliente debe realizar una **apertura activa en el puerto** servidor, abriendo un puerto propio y poniéndose en conexión con el puerto que está escuchando en el servidor. Generalmente los servidores mantienen abiertos una serie de puertos conocidos estándares, en los que un programa concreto (que se suele denominar **demonio**) permanece a la espera de peticiones de conexión.

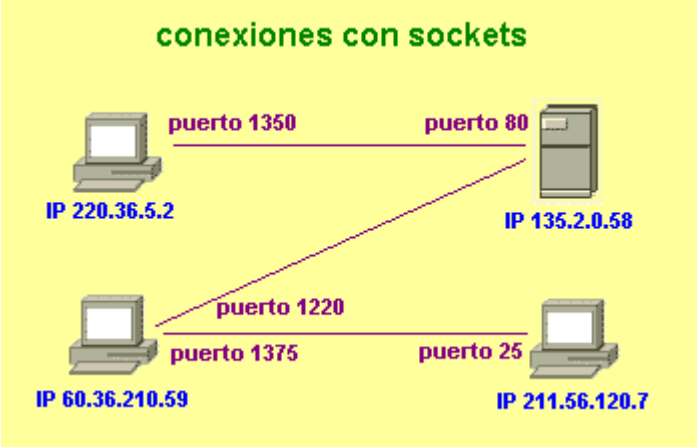


Figura 7.2 Ejemplo de conexión entre sockets

Los puertos van a ser los destinatarios finales de los datagramas en una comunicación entre host de una red. Cada puerto de protocolo podemos imaginarlo como una vía de entrada de información, como un punto de un host que permite la entrada de paquetes a través de él. Si queréis, podéis tener un concepto mental de puerto como una "puerta" de acceso a los niveles de aplicación, una cola de entrada en la que el software de protocolo coloca los datagramas entrantes.

Tanto TCP como UDP usan números de puerto para enviar información a las capas superiores. Los números de puerto se usan para mantener un seguimiento de las distintas conversaciones que atraviesan la red al mismo tiempo.

Para poder establecer una comunicación correcta entre dos máquinas, el emisor necesita conocer tanto la dirección IP como el número de puerto de protocolo del destino dentro de la máquina receptora. Estos puertos los proporciona los protocolos UDP y TCP (capa de transporte), y mediante ellos se puede distinguir entre muchos programas que se ejecutan a la vez dentro de una misma máquina.

EL concepto de puerto en TCP es más complejo que en UDP, ya que TCP utiliza para comunicarse una conexión, no el puerto de protocolo, como su abstracción fundamental: estas conexiones se identifican por un par de **puntos extremos**. Un punto extremo es un par de números host-puerto, en donde host es la dirección IP de un anfitrión y puerto es un puerto TCP en dicho anfitrión.

7.2.3. Puertos

Las conexiones vienen definidas por dos puntos extremos, permitiendo TCP que varias conexiones compartan un mismo punto extremo, siendo los puertos los encargados

de distinguir las distintas conexiones. Las aplicaciones utilizan los puertos para recibir y transmitir mensajes.

Cada puerto de protocolo viene identificado por un número de 16 bits, lo que nos da 65.536 puertos posibles en cada ordenador, que se identifican por su correspondiente número en base decimal. TCP combina la asignación estática de números de puerto con la asignación dinámica, mediante la denominada asignación de puertos bien conocidos para aplicaciones servidoras (asignación estática) y la asignación del resto de los puertos disponibles a las aplicaciones cliente conforme los vayan necesitando (asignación dinámica).

Los números de puerto tienen los siguientes intervalos asignados:

- Los números inferiores a 255 se usan para aplicaciones públicas.
- Los números del 255 al 1023 son asignados a empresas para aplicaciones comercializables.
- Los números superiores a 1023 no están regulados, y se usan generalmente para asignación dinámica.

Cuando una aplicación cliente quiere comunicarse con una aplicación servidora de otro host busca un número de puerto libre y lo utiliza para transmitir los datos, mientras que en el otro host la aplicación servidora permanece a la escucha en su puerto bien conocido para recibir los datos. Por ejemplo, cualquier conversación destinada a la aplicación FTP utiliza el número de puerto estándar 21.

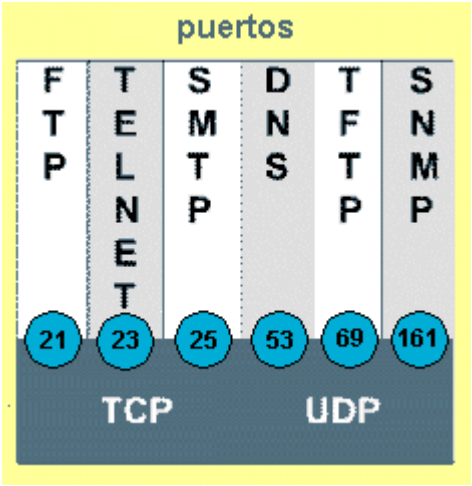


Figura 7.3 Algunos de los puertos más conocidos

Para optimizar su funcionamiento los puertos poseen una memoria intermedia, denominada BUFFER, situadas entre los programas de aplicación y la red. Las aplicaciones transmiten los datos a los puertos, guardándolos éstos en sus buffers hasta que pueden ser enviados por la red. Cuando llegan al host destino, los datos son almacenados de nuevo en los buffers hasta que la aplicación receptora esté preparada para recibirlos.

Capítulo 8.

Protocolo de la Aplicación

8.1 Introducción.

La aplicación en su conjunto consta de varias aplicaciones independientes para usuarios independientes que necesitan comunicarse y compartir información en tiempo real. El protocolo implementado intenta emular la comunicación real que se produce entre los trabajadores de un restaurante. Por ello, se ha realizado un protocolo de mensajes simple y eficaz en el que existe una aplicación central, que es la que contiene la información actualizada, y el que lleva la voz cantante en el restaurante.

8.2. Descripción y Elementos

El protocolo de la aplicación se compone de varios elementos, un servidor central (el de la barra) y unos clientes que se conectan a él para enviarle pedidos, actualizar sus bases de datos, o indicarle que platos ya están listos. Los servidores de los demás puestos de trabajo (camareros y cocina) son, en el primer caso, para recibir notificaciones de actualización y, en el segundo, para recibir los pedidos.

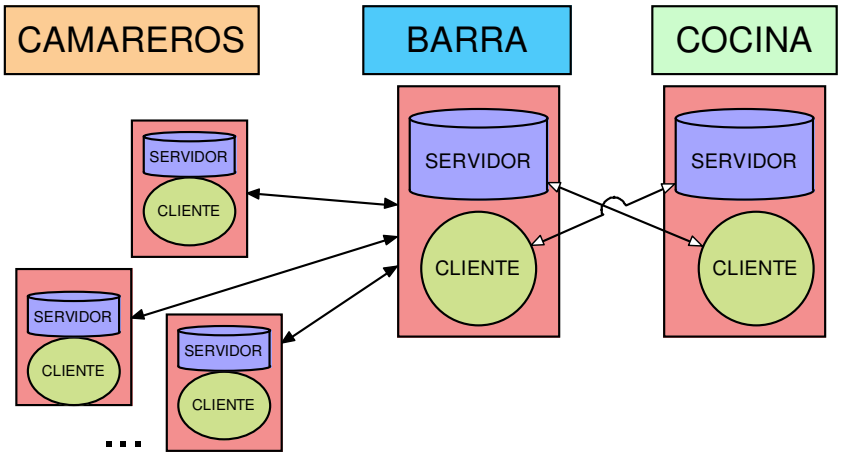


Figura 8.1 Elementos del protocolo de comunicaciones.

8.3. Formato de los mensajes.

Los distintos tipos de mensajes que pueden recibir o enviar se indican en la figura siguiente. Por supuesto que el mensaje no contiene sólo esa palabra, sino que se trata del comando que encabeza el mensaje. Después de estas cinco letras, el mensaje contiene

más información, y además se intercambian más mensajes que en el gráfico no aparecen. Todo esto se explica con más detenimiento en un ejemplo en el punto 7.2.4.

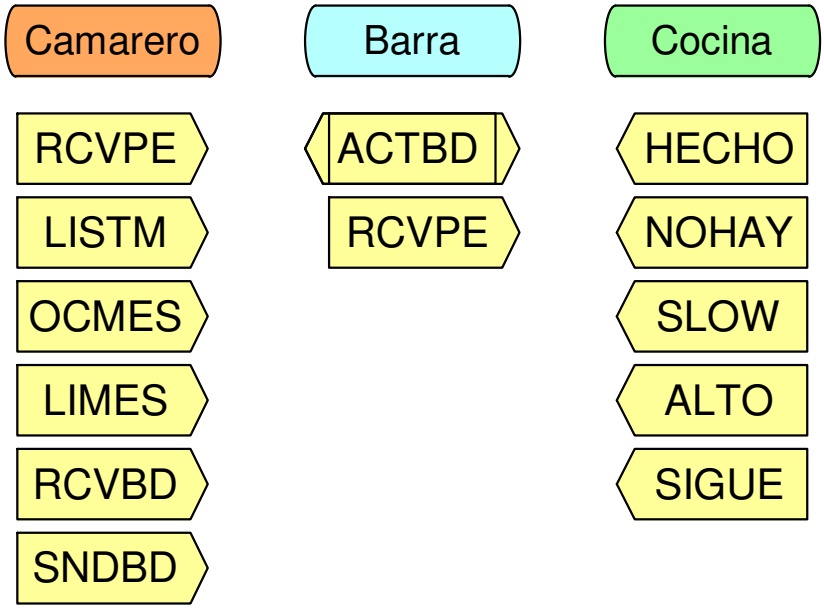


Figura 8.2 Principales mensajes del protocolo.

8.3.1. Mensajes de Camarero a Barra

Nombre del mensaje	Función del mensaje	Formato del mensaje
RCVPE	Recibir pedido (Prepara al servidor de la barra para la recepción del fichero)	RCVPE “nombreDelFichero”
LISTM	Pedir lista de mesas disponibles. Recibirá un array de números.	LISTM
OCMES	Ocupar mesa	OCMES “numeroDeLaMesa”
LIMES	Liberar mesa	LIMES “numeroDeLaMesa”
RCVBD	Recibir base de datos. Prepara a la barra para enviarle el fichero.	RCVBD “nombreFichero”
SNDBD	Enviar base de datos. Le pide a la barra que le envie la BD actual.	SNDBD “nombreFichero”

8.3.2. Mensajes de Barra

Nombre del mensaje	Destino	Función del mensaje	Formato del mensaje
ACTBD	Camareros y Cocina	Actualizar base de datos. Es una notificación, no un envío	ACTBD “Nombre fichero1” “NombreFichero2” …
RCVPE	Cocina	Recibir pedido (Prepara al servidor de la cocina para la recepción del fichero)	RCVPE “nombreFichero”

8.3.3. Mensajes de Cocina

Nombre del mensaje	Función del mensaje	Formato del mensaje
HECHO	Notificar a la barra que los productos ya se han cocinado	HECHO "NumeroDeLaMesa" "NumeroProducto" "Cantidad" "NumeroDeLaMesa" "NumeroProducto" "Cantidad"
NOHAY	Alertar a la barra de que se ha agotado un producto,para que lo actualice en la BD.	NOHAY "Codigo del Pedido"
SLOW	Enlentecer el ritmo de pedidos cuando se empieza a saturar la cocina.	SLOW
ALTO	No se permiten más pedidos de momento por estar al máximo.	ALTO
SIGUE	Seguir con el ritmo normal.	SIGUE
SNDBD	Enviar base de datos. Le pide a la barra que le envie la BD actual.	SNDBD "nombreFichero"

8.4. Códigos de Estado

Los códigos de estado tradicionalmente son un número de 3 dígitos. No hay un estándar que defina un código de estados, pero muchos protocolos siguen un estándar de facto basado en el siguiente:

1xx – Información.
2xx – Éxito.
3xx – Error temporal.
4xx – Error permanente.
5xx – Error interno.

En nuestro protocolo estos son los códigos de respuesta ente los diferentes módulos. Algunos de ellos no están usados en la primera versión del programa, y son sólo una previsión de futuro, para nuevas versiones más completas

Mensaje		
200	Comando Okay	
215	La mesa se ha ocupado.	
220	La mesa ha sido liberada.	
230	Preparado para recibir fichero	
250	Transferencia de Fichero completada	
415	La mesa ya está ocupada, escoja otra	Error
420	La mesas ya está libre, escoja otra	Error
425	No se puede abrir una conexión	Error
426	Conexión cerrada, comando abortado	Error
450	Fichero requerido no encontrado	Error

500	Sintaxis errónea, comando no reconocido	Error
502	Comando no implementado	Error
504	Parámetros erróneos	Error
505	Faltan parámetros	Error
550	Petición de fichero abortada. Fichero no encontrado	Error
553	Acción requerida abortada: El nombre del fichero es erróneo.	Error

8.5. Tabla de puertos

Módulo	Protocolo	Puertos
Servidor Camarero	UDP	7001, 7002, …, 7499
Cliente Camarero	TCP	7501, 7502, …, 7599
Servidor Barra	TCP	5001
Cliente Barra	TCP	5002
Servidores Cocina	TCP y UDP	6001 y 4001
Cliente Cocina	TCP	6002

8.6. Funcionamiento

Se puede ver un ejemplo del uso de uno de estos mensajes en el siguiente gráfico. En él, un camarero después de haber recogido el pedido de sus clientes, lo quiere enviar a la barra. Para ello primero envía el mensaje “RCVPE Pedido4Mesa5”. Con él le está indicando a la barra que le quiere enviar el archivo con nombre Pedido4Mesa5. La barra le contesta con un mensaje que contiene “230 Preparado” para indicarle que ha recibido su mensaje y que está listo para el envío, el siguiente mensaje es el archivo en sí. Por último la barra confirma que ha recibido el archivo. Este mismo protocolo se utiliza para el envío de cualquier archivo entre aplicaciones.

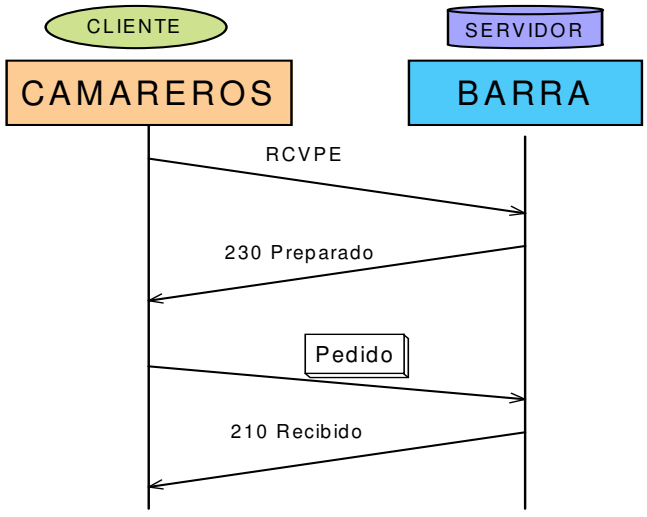


Figura 8.3 Intercambio de mensajes en el envío de un fichero

El único que difiere de esto es el mensaje ACTBD procedente de la barra, ya que es broadcast, así que no se encarga del envío de ningún fichero, sólo de la notificación de actualización. Tras recibir esta notificación los camareros y la cocina vuelven a utilizar el mismo protocolo anteriormente comentado, para enviar ficheros.

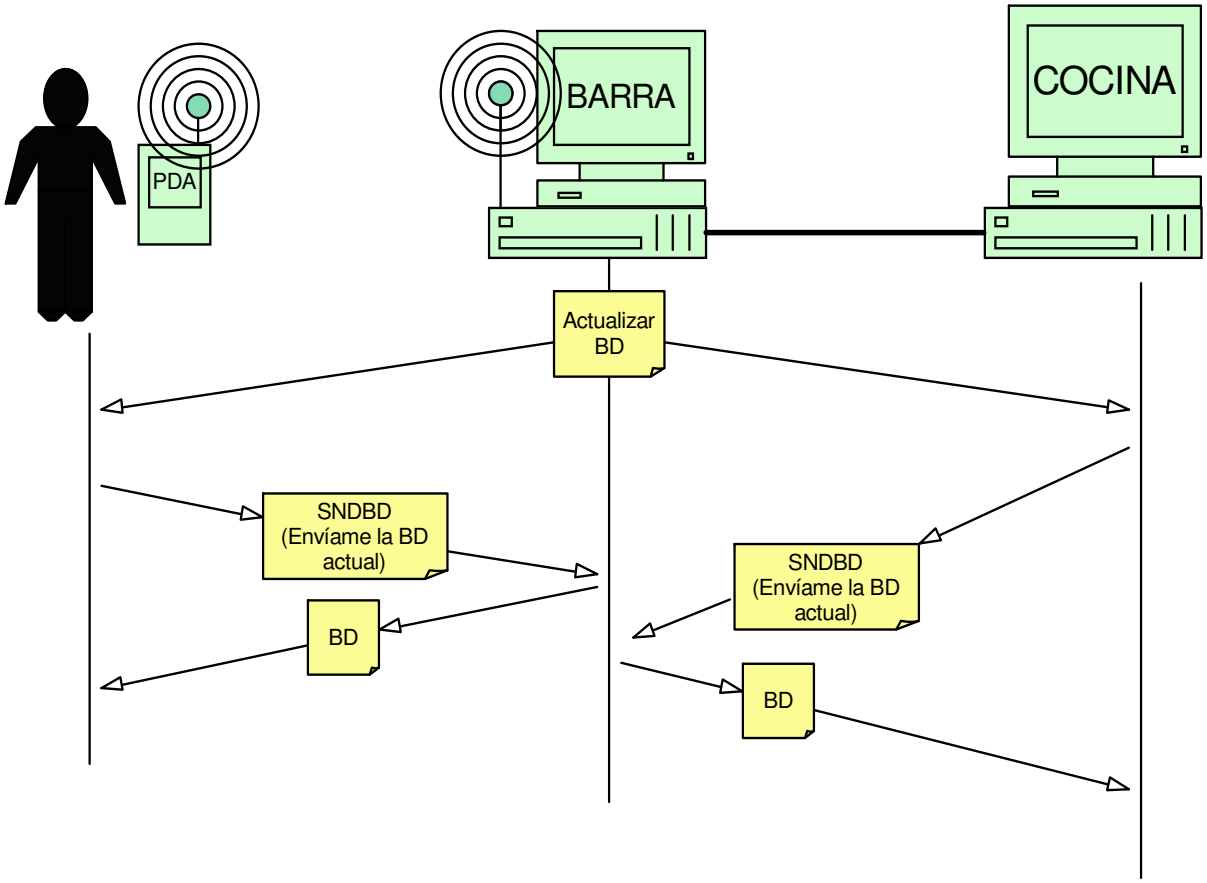


Figura 8.4 Intercambio de mensajes para actualizar la Base de Datos

Al ser el mensaje ACTBD de tipo broadcast, es por tanto también de tipo UDP. Ya que UDP es no fiable, hay que utilizar un protocolo para asegurarse que todos los ordenadores han recibido nuestra notificación de actualización. Para ello se ha optado por la posibilidad de un contador de respuestas. Este contador comienza con un valor igual al número total de equipos que hay en la red (exceptuando al de la barra que es el emisor). Tras enviar el mensaje ACTBD los equipos responderán con un mensaje de petición de la BD, y se decrementará el contador. Si al cabo de un tiempo no se ha llegado a 0, tendrá irremediablemente que volver a comenzar el proceso. Si después de tres intentos fallidos de actualización de la BD no recibimos la respuesta de todos los equipos de la red con la petición de fichero, entonces irremediablemente habrá que desistir en el intento, por no seguir saturando la red. Seguir insistiendo no llevaría a ninguna parte.

En futuras versiones se prevé remodelar este protocolo cambiando el contador por una lista con las IPs de cada ordenador y, si ha fallado uno o más de uno de los equipos, enviarles de nuevo la notificación sólo a aquellos que no contestaron. Gracias a esto se evitaría tener que reenviar archivos que ya se han recibido satisfactoriamente en algunos equipos y que no necesitan tener que actualizarse.

Capítulo 9.

Componentes

9.1. Introducción.

Para la consecución de algunos de los objetivos del proyecto se consultaron páginas Web, foros, libros, y otros documentos buscando soluciones a problemas que en muchos casos no eran nuevos para los programadores. Unos desafortunadamente no se habían implementado todavía, pero otros muchos ya habían sido quebradero de cabeza de programadores en Delphi y, además, habían sido tan amables de ceder sus porciones de código y componentes a los noveles programadores.

Muchas de las páginas recopilan una gran variedad de estos componentes, que a veces son tan completos como complejos. Desafortunadamente la mayoría no eran exactamente lo que se buscaba o no se parecían a lo que se le pedía al programa.

En uno de los casos, el más importante en cuanto a comunicaciones, se ha optado por utilizar los componentes indy, ya que son muy completos y tienen un enorme listado de protocolos y posibilidades que abarcan y que están esperando ser configuradas para empezar a funcionar.

Otro de los componentes encontrados era “SaveComps”, el cual nos posibilita guardar un fichero con la posición, color, y otras características de los diversos componentes que forman una ventana, para que cada modificación que se realice en tiempo de ejecución sobre el programa, perdure en las siguientes ejecuciones.

9.2. Componentes Indy

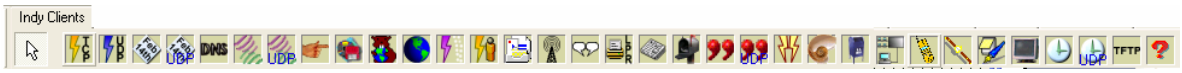
¿Qué son los componentes Indy? Sus autores lo definen como un grupo de componentes de código abierto, escritos en Delphi y basados en “sockets bloqueantes”.

La página oficial de este proyecto puede encontrarse en: http://www.indyproject.org/. En resumen, puede decirse que los componentes Indy nos permiten trabajar con los protocolos más comúnmente utilizados en Internet, sin necesidad de realizar una programación excesivamente complicada, ya que tiene un montón de funcionalidades implementadas, que facilitan la programación. Nos centraremos principalmente en los servidores y los clientes indy, y sobretodo en los dos componentes indy que hemos utilizado: IdTCPClient, IdTCPServer.

Principalmente, cuando se programa una aplicación cliente servidor hay que tener en cuenta que mientras en los clientes las transacciones son secuenciales, en los servidores se emplean varios hilos de ejecución, lo que supone una complicación añadida.

9.2.1. Clientes Indy

Hay una larga colección de protocolos de red implementados en indy. Para cualquier duda acerca del protocolo se puede consultar su rfc (http://www.rfc-editor.org/rfc.html).



TidTCPClient. Este componente es un poco especial dentro de los clientes indy, ya que incluye un cliente TCP (Transmisión Control Protocol) completo. Se puede utilizar como precursor de otras clases que traten un protocolo específico. Muchos componentes indy utilizan esta clase para funcionar. En la mayoría de los casos nunca se utilizará directamente, ya que casi todos los protocolos que nos van a interesar ya están cubiertos con otros componentes específicos.

TidUDPClient. Lo mismo puede decirse en este caso, con la salvedad de que UDP puede usarse para enviar información sin necesidad de establecer conexiones, de forma que es posible que interese usarlo directamente.

Todos los clientes indy son descendientes de TidTCPClient o TidUDPClient. Queda claro entonces, que cada componente de la paleta "Indy Clients", está asociado a un servicio concreto, salvo TidTCPClient, y TidUDPClient, que son genéricos de los respectivos protocolos (TCP-UDP).

9.2.2. Servidores

Indy dispone de una amplia variedad de modelos de servidores, dependiendo de las necesidades del protocolo usado. El gráfico siguiente muestra los iconos que corresponden a algunos de ellos. Comentaremos los dos que hemos utilizado en este proyecto: TIdTCPServer y TIdUDPServer.

TIdTCPServer. TIdTCPServer crea un hilo escuchador secundario independiente del hilo principal. El hilo escuchador espera peticiones de conexión de entrada de los clientes. Por cada cliente al que le responde crea un nuevo thread específico para servir a la conexión con ese cliente. Los eventos son lanzados dentro del contexto de cada hilo.

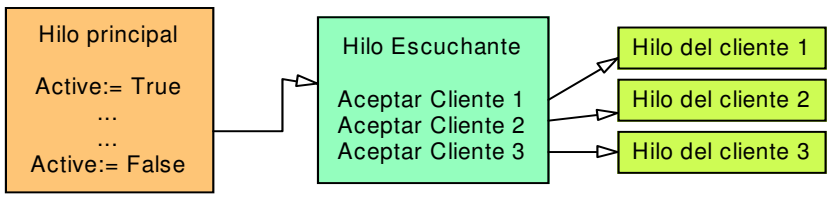


Figura 9.1 Hilos de ejecución de un servidor

TIDUDPServer. Udp es no orientado a conexión. Cuando se activa crea un hilo (thread) para escuchar los paquetes UDP entrantes. Por cada paquete recibido, lanzará un evento OnUDPRead en el hilo principal, o en el hilo que escucha dependiendo del valor de la propiedad ThreadedEvent (cuando es falsa la lanza el principal).

Ya que cada vez que se recibe el evento anterior, se bloquea la recepción de más mensajes, hay que tener en cuenta que el procesado de los eventos OnUDPRead debe ser rápido.

9.2.3. Modelos de servidores TCP

Los servidores TCP Indy soportan dos modelos para construir servidores. Esos métodos son “OnExecute” (en ejecución), y manejadores de comandos.

OnExecute se refiere al evento del TIdTCPServer. Cuando se está implementando un servidor usando este modelo,

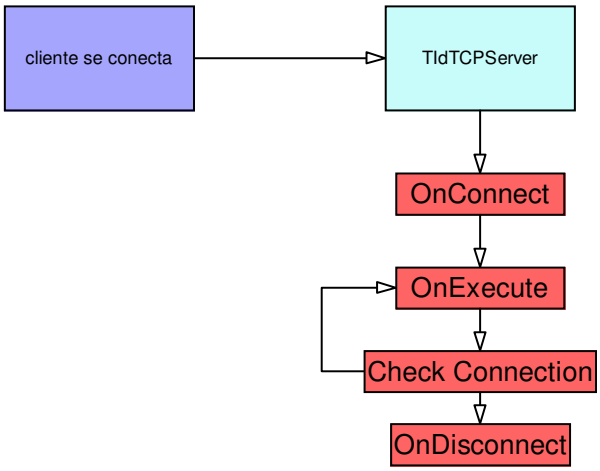


Figura 9.2 Servidor en ejecución

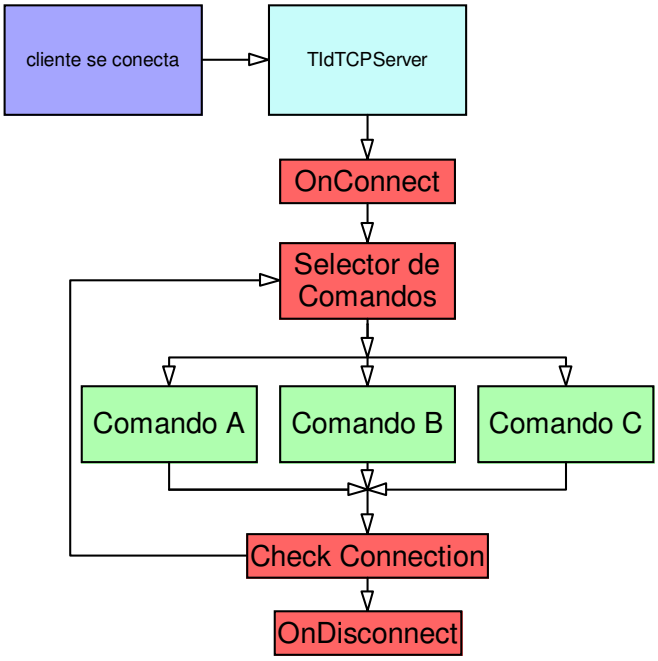


Figura 9.3 Servidor en manejador de comandos

Las diferencias son apreciables a simple vista: El manejador de comandos permite poder ejecutar diferentes códigos y funciones según la orden recibida, mientras que en el primero, todo el código se ejecuta cuando se recibe una conexión. En este primer caso, el programador se descarga de la obligación de tener que averiguar y controlar la porción de código que hay que ejecutar y tener que discernir entre diferentes comandos.

9.2.4 ¿Bloquear o no bloquear?

Los sockets se originaron de la definición de Berkely Unix. Cuando se realiza una operación de lectura sobre un socket, el flujo del programa se detiene hasta que se reciban los datos necesarios. Unix maneja esto mediante una llamada a procedimiento fork, que inicia una copia del programa original y lo ejecuta. De esta manera se evita que el programa original se quede bloqueado. Por otro lado, Windows no dispone de esta funcionalidad, y en la versión 3.x no existían las hebras (Threads). Por ello, cuando se quiso trabajar con sockets síncronos (o bloqueantes) en Windows 3.x existía el problema de que la GUI se quedaba bloqueada y no respondía a las interacciones del usuario. Además, como Windows 3.x era un sistema operativo que utilizaba una multitarea cooperativa (cada proceso cedía el uso del procesador). Cuando una aplicación quisiera operar con sockets, parecía que todo el sistema se quedaba colgado. De ahí surgió la idea de sockets asíncronos (o no-bloqueantes).

Con la aparición de plataformas Win32, se podía utilizar las llamadas bloqueantes ya que existían las hebras. El problema fundamental es que muchas personas se han acostumbrado a utilizar sockets asíncronos, y los síncronos han cogido mala fama. Sin embargo, para trabajar en Linux sigue siendo necesario utilizar sockets bloqueantes, y es recomendable perder el miedo a estos y ver lo sencillo que puede ser su uso de cara a la salida de Delphi para Linux). Las diferencias principales entre otros paquetes de componentes sockets existentes en el mercado e Indy son:

Indy se basa en sockets bloqueantes.
No depende de eventos.
Se ha diseñado para trabajar con hebras.
Se programa utilizando un flujo secuencial.

9.3 Componente SaveComps

Gracias a este componente, podemos desplazar iconos (como las mesas) botones y otros elementos, por toda la ventana, modificarles los tamaños, y poder recuperar todas estas características cuando volvamos a iniciar la aplicación. Y lo sorprendente es que todo esto lo hacemos en tiempo de ejecución. De esta manera los usuarios de las aplicaciones pueden personalizar la apariencia y posición de algunos elementos de la ventana, y recuperarlos automáticamente en una próxima ejecución.

Esto lo hace creando un archivo con extensión .ini al finalizar la aplicación, en el cual guarda la siguiente información:

```
[pnlOcupacion]
Left=0
Top=213
Width=215
Height=206
ClassName=TPanel
Parent=TabSheet2
```

- Nombre
- Dimensiones
- Posición
- Nombre de la clase
- Clase Padre

Cada vez que el programa se inicia, lee este archivo para conocer las últimas personalizaciones y aplicarlas.

Capítulo 10.

Conclusiones y Líneas futuras

10.1 Conclusiones

La elaboración de este proyecto me ha aportado una sensación general de satisfacción al haber logrado los objetivos que se idearon en un principio y, además, por que he adquirido nuevos conocimientos de programación y una metodología de trabajo.

Partiendo de un planteamiento inicial consistente en una serie de necesidades y limitaciones de los restaurantes se dio a luz a unos requerimientos de partida; pero como suele ocurrir con cualquier trabajo de envergadura, la necesidad de dar solución a los problemas que van surgiendo condujo a proyectar nuevas mejoras y funcionalidades. Se consiguió una significativa evolución de las aplicaciones de software con respecto a la fase de diseño; y es ahora, cuando he analizado las distintas fases del proyecto, cuando se puede ver que el producto final es significativamente mejor y más completo de lo que se ideó en un principio.

Asimismo este es un proyecto abierto a nuevos desarrollos e investigaciones, ya que todavía se pueden mejorar mucho las capacidades y funcionalidades de estos programas. Como suele ocurrir con la totalidad del software del mercado, muchos retoques y revisiones tendrán que esperar hasta que lleguen las futuras versiones y actualizaciones, algunas de ellas forman parte de la lista de líneas futuras.

Además he seguido el proceso de desarrollo de todo producto real, y es:
Idea → estudio de mercado → análisis de factibilidad → desarrollo → evolución.

En el estudio de mercado he analizado y estudiado las diferentes soluciones ya existentes, tomando las mejores ideas de cada uno, y mejorando algunas carencias. Es evidente que la aplicación aquí presentada bien puede hacerse un hueco en el mercado aprovechando la oportunidad de negocio que surge con la introducción de las nuevas tecnologías.

Por otro lado, los modestos conocimientos de programación que tenía al comienzo de este trabajo, se han visto ampliados y afianzados. He aprendido a usar un nuevo lenguaje, nuevas herramientas software, y además he trabajado con bases de datos que tan importantes son en el mundo de la programación.

Y para finalizar, sólo queda comentar que para mí ha sido una muy buena toma de contacto con las aplicaciones telemáticas. He estudiado y usado protocolos de red existentes, y he creado un protocolo de comunicaciones propio, utilizando los anteriores como base. Todo esto me hace llegar a la siguiente conclusión: *"he conseguido el objetivo básico de todo proyecto al reforzar y ampliar mi formación académica en un campo en concreto"*.

10.2. Líneas Futuras

Dado el ámbito eminentemente práctico que tiene este proyecto, son muchos los puntos que se podrían ampliar para abarcar mayores funcionalidades y tener así un programa mucho más completo y competitivo en el mercado. A continuación se exponen algunos ejemplos de líneas futuras en las que se podría retomar y mejorar esta aplicación:

- Aplicar sistemas de seguridad y encriptado a las comunicaciones. En cualquier comunicación es, cuanto menos, deseable que no carezca de seguridad. Ante posibles amenazas externas sería bueno que, por ejemplo, se encriptarán las comunicaciones para evitar que alguien ajeno al personal del restaurante pudiera interceptar los mensajes y modificarlos, o simplemente romper esa privacidad.

- Cambiar pedidos. Los clientes a veces cambian su pedido, después de estar realizado, con relativa frecuencia. Para que se pudiesen adaptar esos cambios, habría que modificar el protocolo de comunicaciones entre las aplicaciones, para que el camarero tuviera la posibilidad de indicar a los otros equipos que el pedido ha sufrido variaciones. Esto supondría una buena mejora en el programa.

- Mejorar la interfaz y adaptarla a equipos móviles. Las aplicaciones aquí expuestas han sido sólo probadas en ordenadores convencionales. Una nueva línea de investigación y desarrollo sería adaptar la aplicación camarero a PDAs, y a otros tipos de equipos informáticos, adecuando a su vez la interfaz gráfica a las dimensiones y su capacidad de cómputo. Teniendo en cuenta que cada vez está más extendido el uso de equipos que integran telefonía, Internet, aplicaciones informáticas y redes inalámbricas de pequeño alcance, se podría implementar en móviles de última generación para que los usuarios pudieran descargarse la aplicación desde Internet y poder así usarla en su restaurante, o para que permitiese a los clientes realizar los pedidos en una página web desde su móvil, o reservar una mesa.

- Soportar otros sistemas operativos. Los programas que se han desarrollado corren bajo el sistema operativo Windows, pero en nuevas versiones, se debería utilizar un compilador (que actualmente ya existe y se llama Lazarus), que cree ejecutables válidos para los Sistemas Unix, como Linux.

- Aplicación web. Hoy en día es posible comprar desde Internet casi cualquier cosa, sin moverse de casa. Así que una posible mejora consistiría en crear un espacio web desde el que realizar reservas en el restaurante, hacer un pedido de comida para entrega al domicilio o lugar de trabajo, pagar la última cuenta que dejamos en el restaurante, consultar el menú del día, etc.

- Las notificaciones que reciben los camareros para saber que un plato ya está cocinado y listo para servir, sólo se pueden ver en el equipo de la barra. Esto es así ya que es a donde llegan los productos de la cocina, de donde tienen que recogerlos; pero sería una mejora sustancial que el camarero recibiese un aviso en su PDA indicándole que unos platos en la barra le esperan para ser servidos.

- Algunos de los programas analizados también cuentan con un sistema de contabilidad que diferencia entre camareros y que muestra al final del día las ventas de cada uno. Es una forma bastante extendida de primar a los que más trabajan, gracias a que se sabe todo lo que ha servido cada uno de ellos al cabo de la jornada.

- Poder pasar de un modo rápido a uno lento en función de la ocupación.

- Conseguir una tabla de productos que tenga constancia de la cantidad que queda de cada uno de ellos para que, automáticamente, cuando esté agotado un producto, realice una notificación a la barra y, también si se quiere, envíe por Internet el pedido al proveedor. Para ello se puede aprovechar la información de todos los pedidos, realizando una contabilidad de los productos consumidos y facturados al cabo del día.

- Incluir más tipos de subproductos dentro de las 13 familias de productos y platos.

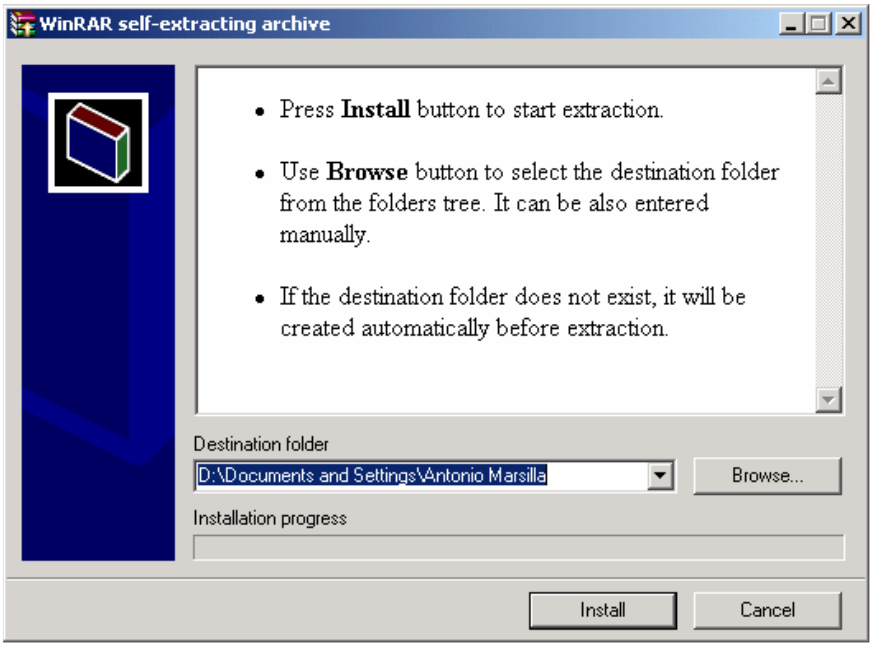
- Incluir menús del día en la carta. Los menús siempre son un método de ahorro con buena calidad, que está muy extendido en la cultura española, ya que se ajustan a un buen precio, con una variedad razonable de platos. Así que deberían poder ordenarse por tiempo sus productos, es decir primeros, 12 minutos después el segundo, y otros 12 para el postre. Esto además, implicaría que al seleccionar un menú, cuando seleccionemos el primer plato, deben desaparecer los platos que no están incluidos dentro del menú, y que el precio se ajuste a lo acordado.

Anexo A.

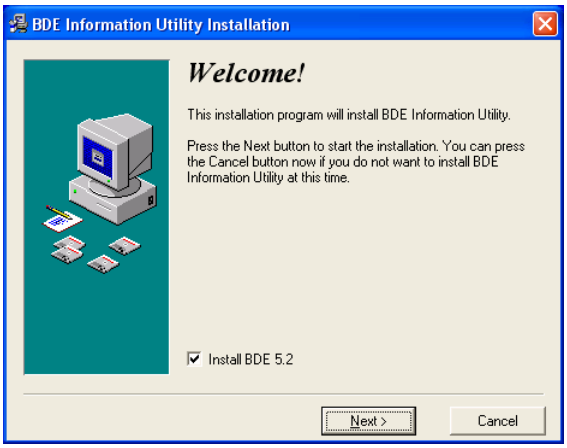
Manual de usuario

A.1. Instalación

Instalación. La aplicación va comprimida en un fichero auto-extraíble. Para extraerlo sólo hay que hacer doble clic sobre ella. Nos aparecerá una ventana que nos pedirá la ruta del directorio donde queremos extraerlo.



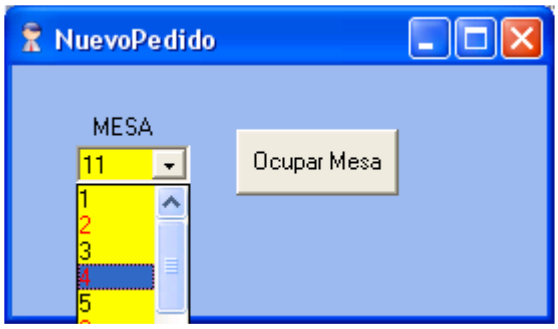
Una vez elegida la ruta completa pulsamos Install. Automáticamente se nos extraen los cuatro programas en sus correspondientes carpetas, más un archivo de instalación del Motor de Base de Datos de Borland (BDE). Este archivo también se ha de instalar para que puedan funcionar todas las aplicaciones que aquí hemos presentado, ya que todas ellas utilizan el BDE para sus bases de datos.



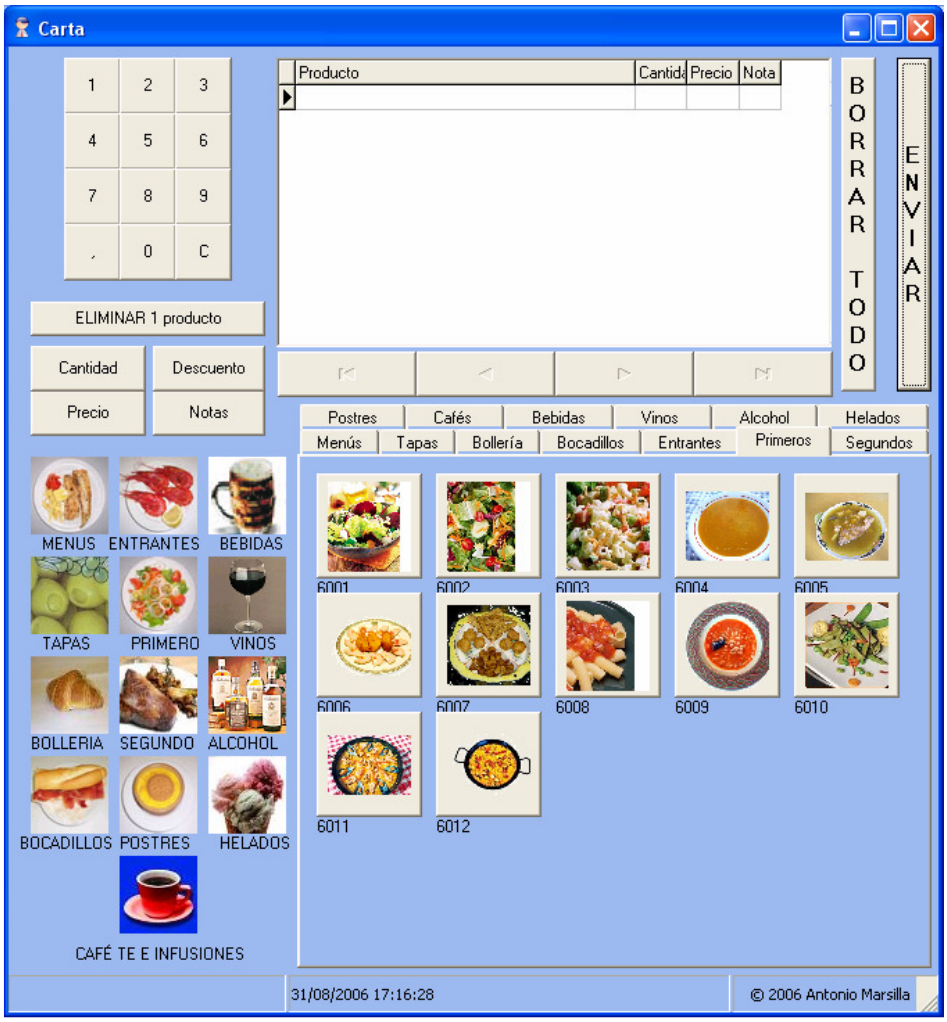
Se puede descargar la última versión desde la página de Borland. La instalación es bastante sencilla y nos bastará con leer el acuerdo de licencia (en inglés), y pulsar 3 veces sobre siguiente.

A.2 Camarero

Nada más iniciar el programa camarero, se nos aparece la primera ventana, llamada NuevoPedido. Aquí tenemos que escoger la mesa, y pulsar el botón. Tras esto cambiamos de ventana:



Para empezar a realizar el pedido lo primero que tenemos que hacer es escoger la familia de productos que nos han pedido. Si es una tapa pinchamos sobre tapas en la botonera de la izquierda, y seguidamente elegimos uno de los productos que nos presenta el panel.



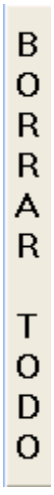
- Si queremos pedir más de ese mismo producto, pulsar sobre el botón Cantidad.

- Si nos hemos equivocado, pulsar sobre el botón "Eliminar" para borrar una fila de la lista de pedidos.

Desplazamientos y modificaciones de pedidos.

Con estos cuatro botones podemos desplazarnos de arriba abajo y viceversa, por la lista de pedidos, para poder aumentar el número de raciones de un producto, disminuirlo, o borrarlo completamente.

Producto	Cantida	Precio	Nota
Bravas	1	1,5	
Tortilla pincho	1	1,2	
Combinado 1	1	5	
Combinado 2	1	5,5	
Combinado 3	1	4,5	
Combinado 4	1	5,6	
Pimientos de piquillo	1	1,5	
Pulpo a la gallega	1	2,5	
Boquerón frito	1	2,5	
Calamares	1	2	



Estos nos desplazan al **principio** o al **final** de toda la lista.

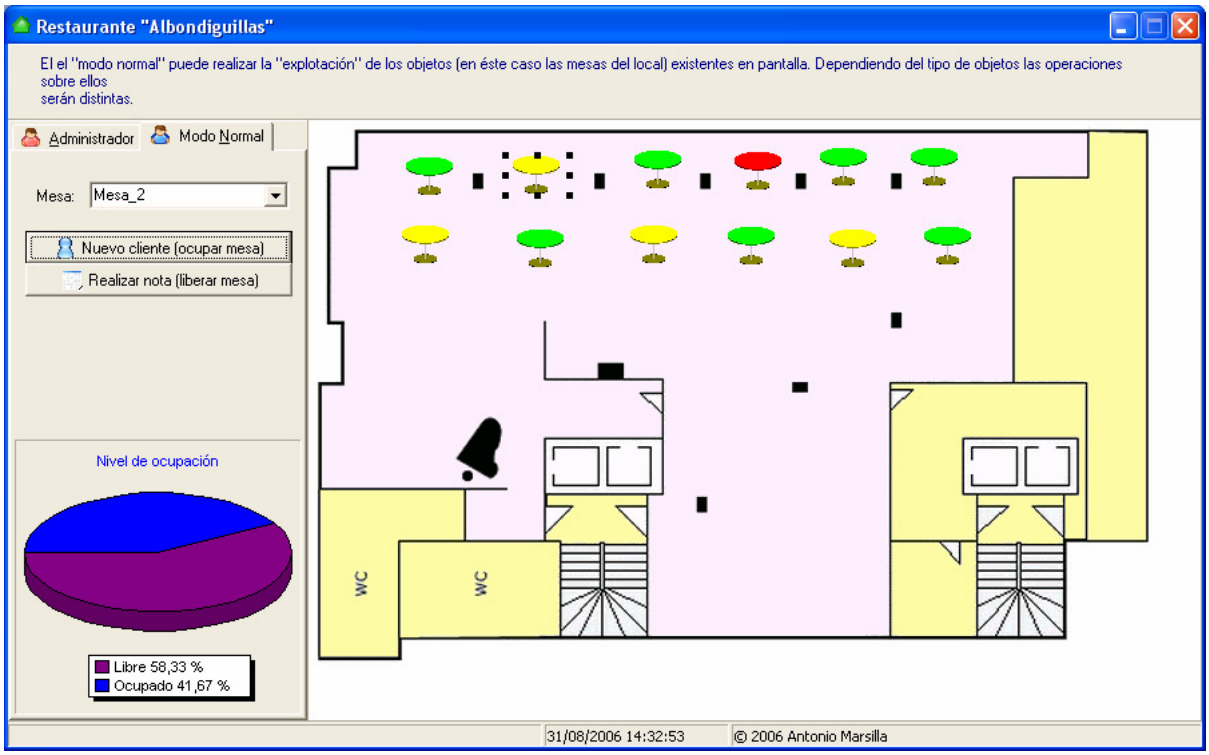
Con estos nos desplazamos **fila a fila** por la lista.

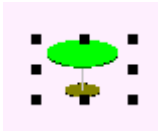
Una vez que se ha terminado el pedido tenemos la opción de **enviarlo** a la barra pulsando sobre "ENVIAR". Si en cambio lo que queremos hacer es **eliminar** el pedido completamente, pulsaremos sobre "BORRAR TODO".

Cualquiera de las dos opciones anteriores nos devolverá a la pantalla inicial, preparados para recoger un nuevo pedido.

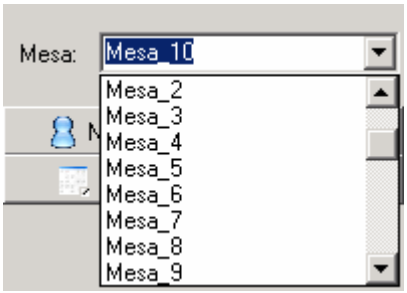
A.3. Barra.

Este programa es bastante intuitivo. La apariencia de la ventana nada más iniciar el programa es esta:





Seleccionar una mesa. Se puede seleccionar pinchando directamente sobre la mesa en el mapa, y aparecen unos cuadros negros alrededor de la mesa. La otra manera es pinchar sobre la flecha que hay al lado del recuadro Mesa, y se nos abre una lista desplegable con todas las mesas disponibles. Pinchando sobre una de estas opciones también se selecciona una mesa.



Ocupar una mesa. Una vez se ha seleccionado previamente la mesa, debemos pinchar sobre el botón "Nuevo cliente", y la mesa cambiará automáticamente de color (al color amarillo).

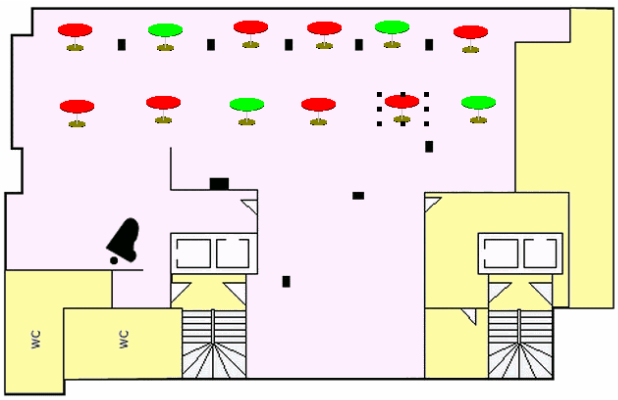
Liberar Mesa y dar la cuenta. Cuando se ha terminado completamente de servir una mesa, y sólo falta pasarle la nota de cobro, pinchando sobre el botón "Realizar nota" la mesa volverá a estar libre, y además nos proporcionará el total de la cuenta.

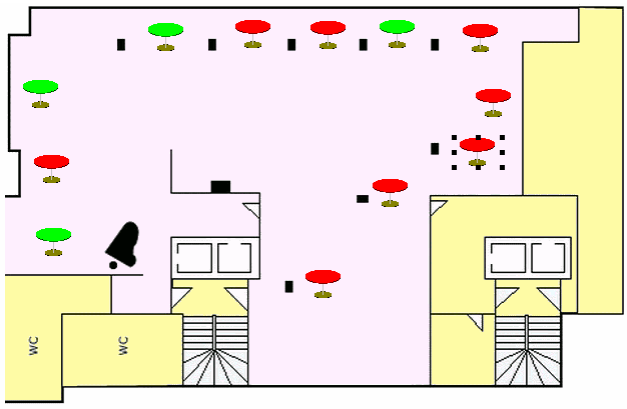
Conocer el estado de una mesa. Su color nos da ya información acerca del estado (ver la figura de abajo). Para obtener una información más detallada hay que realizar un doble clic sobre la mesa elegida, y se desplegará una nueva ventana con información acerca del estado de la mesa, el pedido que se ha realizado, y los platos que ya están cocinados, y los que se han servido ya.

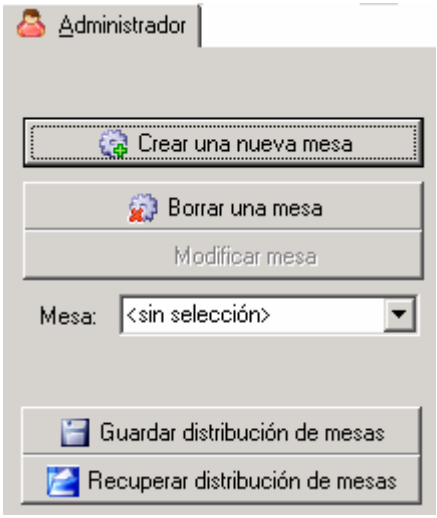
Servir platos. Podemos cambiar el estado de la mesa de "Platos pendientes" a "ocupada" y a la vez cambiar los platos de la lista de preparados a la lista de "Servidos" con sólo pulsar "Servir Platos".

Errores. Si se produjera algún error, como por ejemplo un apagón la lista vuelve a aparecer cuando se inicia el programa, tal yl como la dejamos. Pero para entonces también podría no interesarnos. Así que tanto si queremos recuperar la lista completa de los platos de esa mesa ("Cargar lista"), como si queremos borrarla completamente ("Borrar todos los platos") el programa nos ofrece esas opciones por medio de botones.

Cambiar distribución de mesas. Para ello debemos cambiar del modo normal al modo administrador. En este modo, hay varias posibilidades para desplazar las mesas por todo el local (ver mapa del mismo). Si queremos mover una mesa desde su ubicación original a otra cualquiera, lo único que hay que hacer es pinchar sobre ella y, sin soltarla, arrastrarla a la posición deseada y, una vez allí, soltar.



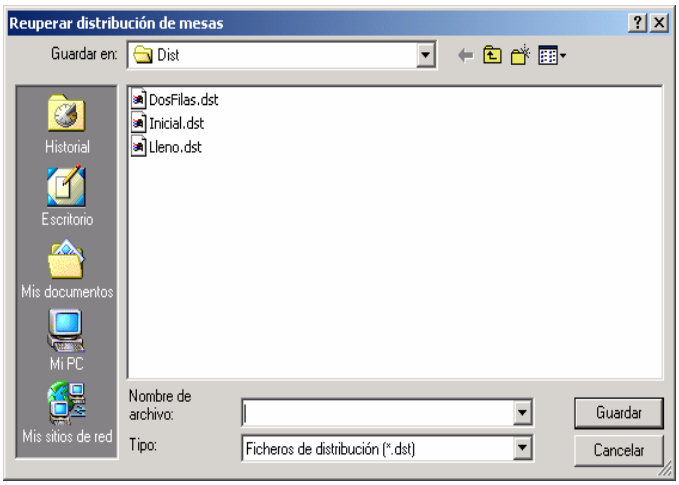




Crear y borrar mesas. Aparece una nueva mesa cada vez que se pulsa el primer botón. Si se selecciona una mesa se puede borrar con el segundo botón.

Guardar distribución de mesas. Una vez elegida la distribución de mesas que mejor se adapta a nuestro restaurante, podemos guardarla en un archivo para que si se produce algún cambio fortuito o intencionado, poder recuperarla. Pinchando sobre el botón "Guardar distribución de mesas" se desplegará una ventana como aparece en el siguiente dibujo::

En esta nueva ventana, debemos dar el nombre del archivo que guardará la distribución de mesas, y la ruta de carpetas (es aconsejable dejar la ubicación por defecto). Con estos dos datos ya elegidos, pulsar sobre "Guardar" (esquina inferior derecha).



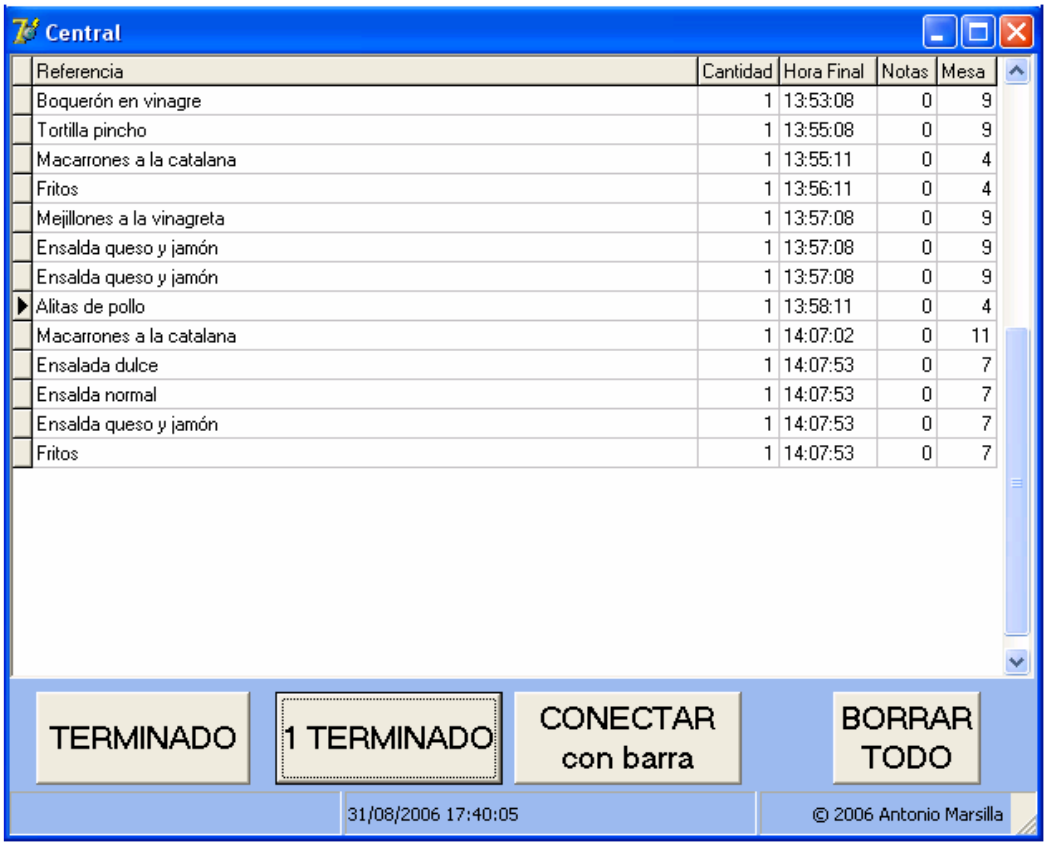
Recuperar una distribución de mesas. Pulsando sobre ese botón nos aparecerá la misma ventana que en el caso anterior, pero esta vez no es necesario escribir el nombre del archivo, sólo pinchar sobre él y pulsar "Abrir".

La barra inferior del programa nos da la información de la hora, tanto en el modo normal como en el administrador.

La esquina inferior izquierda del modo “Normal” muestra un gráfico que da información acerca del nivel de ocupación total del restaurante.

A.4. Cocina.

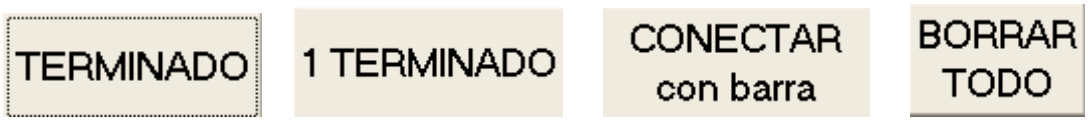
La ventana de cocina tiene este aspecto.



Los pedidos que realizan los clientes, se añaden automáticamente a la lista, si el programa barra también está corriendo. La información que ofrece la lista es:

- Referencia: El nombre del plato.
- Cantidad: Número de raciones de ese mismo plato
- Hora Final: Hora límite para terminar el plato (es sólo una indicación, no hay que verlo como una orden que hay que cumplir a rajatabla).
- Notas: Aclaraciones sobre preferencias de los clientes acerca de los platos ("poco hecho", "sin sal") etc.
- Mesa: El número de la mesa a la que pertenece el plato o pedido.

Los botones que se presentan en la parte inferior nos permiten:



- "TERMINADO". Indicar que un producto ya está preparado para servir. Al pulsarlo, el producto desaparece automáticamente de la lista, apareciendo en la lista de los platos que tiene para servir la barra.

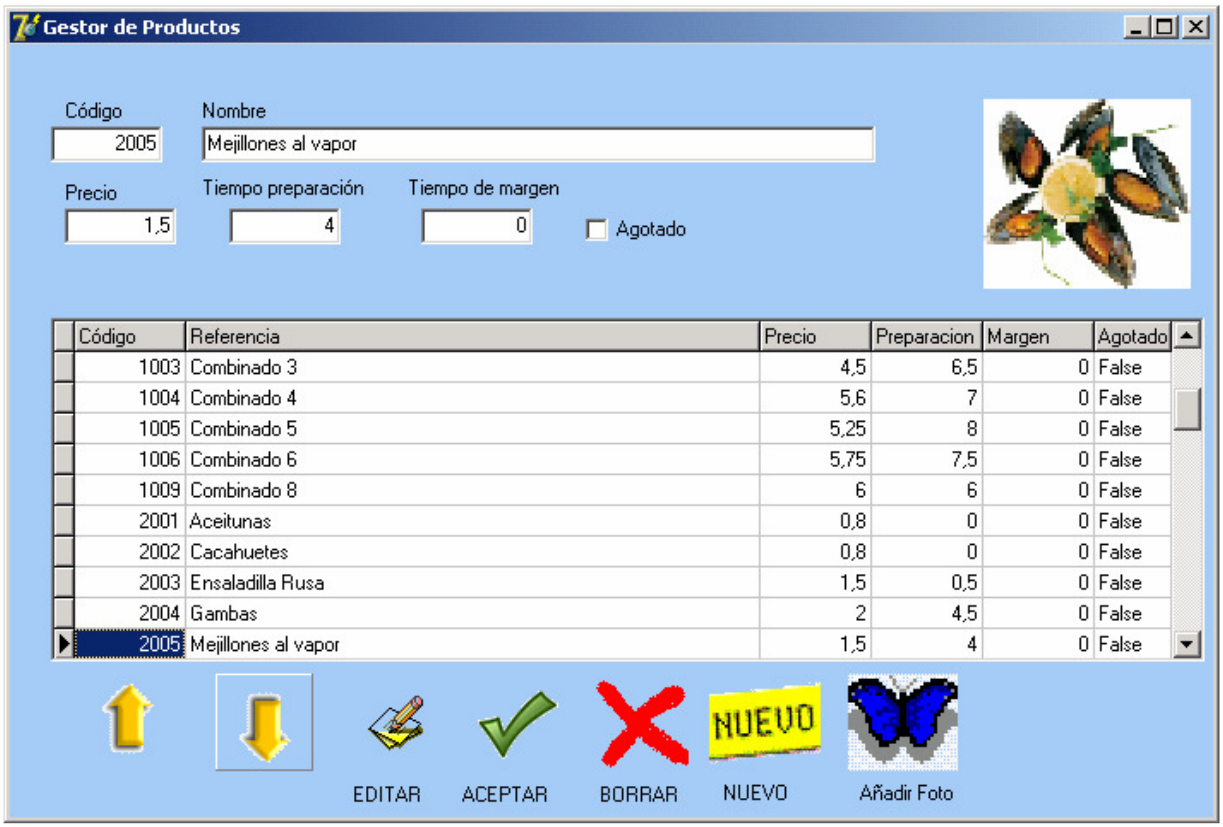
- "1 TERMINADO". Igual que el anterior con la única diferencia de que si hay más de una ración de un plato, decrementa la columna cantidad en vez de eliminarlo totalmente de la lista.

- "CONECTAR con barra". Realizamos una conexión manualmente con la barra para indicarle que la cocina está lista. En el caso de que la barra estuviera encendida cuando este programa se inició, entonces no sería necesario pulsar este botón.

- "BORRAR TODO". Para casos en los que se quiera eliminar el pedido completo este botón nos ayudará.

A.5. Gestor de productos.

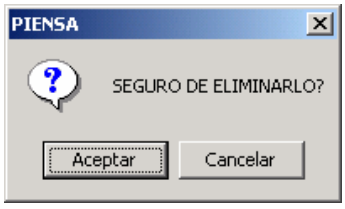
La aplicación es la de la figura siguiente.



Desplazamientos. Nos desplazamos producto a producto utilizando las flechas amarillas de la botonera inferior. Si queremos movernos más rápido, tenemos una barra de desplazamiento a la derecha de la lista de productos.

Ingresar y modificar datos. Nos posicionaremos sobre la fila deseada y pulsaremos "Editar". Automáticamente el cursor se pone sobre las cajas de arriba para que rellenemos o modifiquemos sus valores. Una vez hecho esto hay que pulsar aceptar para que se actualicen los valores.

Borrar un producto. Tan simple como pulsar el botón borrar una vez hemos seleccionado el producto a borrar. Un mensaje de confirmación nos aparecerá para ver si estamos seguros de que lo queremos borrar.



Insertar una nueva foto. Pulsando sobre la mariposa, nos aparecerá un nuevo menú para que elijamos la foto que queremos asignarle al producto, y seguidamente pulsaremos aceptar.

Anexo B.

Formato JPEG

Este formato fue elaborado por el *Joint Photographic Experts Group,* de cuyas siglas deriva su nombre. El formato JPEG (que suele usar nombres de archivo con las extensiones *.JPEG o *.JPG) nació como una respuesta a las limitaciones de otros formatos, entre ellos el GIF, en cuanto a calidad y tamaño de archivos.

JPEG es un formato de compresión con pérdida, esto quiere decir que, al guardar una imagen en este formato, algo de la información que contiene esa imagen se reduce, es decir, esta pierde un poco de calidad, aunque, generalmente, esta pérdida de calidad es imperceptible al ojo humano. Con ello se consigue reducir el tamaño del archivo.

Por otro lado, el formato JPEG permite elegir el nivel de compresión que queremos asignar a un archivo, de modo que podamos decidir qué punto deseamos entre una mayor calidad de imagen (y, por tanto, un mayor tamaño de archivo) o una imagen de baja calidad (con un menor tamaño de archivo).

El sistema 'de compresión que usa JPEG se basa en reducir información promediándola en las zonas de degradado. A grandes rasgos, esto quiere decir que se calcula el valor de color de algunos píxeles en función del color de los píxeles que les rodean. Debido a ello, este formato es muy eficiente a la hora de almacenar imágenes que posean muchos degradados y matices de color, mientras que es casi inútil cuando se enfrenta a dibujos con grandes extensiones de colores planos y uniformes o con bordes muy definidos.

Si a lo anterior unimos que tiene una profundidad de color (número de colores que puede representar) de 24 bits (algo más de 16 millones de colores), veremos que JPEG es ideal a la hora de mostrar fotografías.

JPEG permite también guardar los archivos en modo "progresivo", lo que hará que, a la hora de mostrar la imagen por pantalla, se pueda ver (aunque aún no se haya cargado completamente) con menor calidad. Calidad que irá mejorando a medida que se cargue la imagen hasta obtenerla completamente.

El formato JPEG permite almacenar en el gráfico algo de información en texto (para, por ejemplo, indicar el autor, copyright, etc.) y una copia de tamaño reducido de la imagen, para ser usada por programas de visualización o edición. Pero estas son características que no nos sirven a la hora de hacer páginas web y, además, hacen que el archivo final sea mayor, de modo que las evitaremos en nuestras imágenes

Anexo C.

Formato BMP

El formato BMP (Windows BitMaP) es probablemente el formato de fichero para imágenes más simple que existe. Aunque teóricamente permite compresión, en la práctica nunca se usa, y consiste simplemente en una cabecera y a continuación los valores de cada píxel, comenzando por la línea de más abajo y terminando por la superior, píxel a píxel de izquierda a derecha.

Se ha elegido este formato por carecer de compresión, que aunque en principio podría parecer sólo una desventaja, tiene otras virtudes que son las siguientes, enumeradas de menor a mayor importancia:

- Integración: Este tipo de imagen es el de uso más común y el único soportado por la totalidad de programas Windows, así como por el propio sistema operativo.

- Facilidad de uso: Es soportado directamente como objeto Delphi, con lo que no necesitaremos transformaciones adicionales ni usar componentes externos software, esto es debido al punto anterior, ya que Windows sólo manipula directamente BMP.

- Fidelidad: Al carecer de compresión, al salvar una imagen en formato BMP tenemos la seguridad de que va a ser idéntica al original, cosa que no podemos decir de aquellos formatos como el JPEG, que incluso tiene una cierta componente aleatoria en lo que sería la variación sobre el original. Existen formatos de compresión sin pérdida, pero por ser menos conocidos o estar optimizados para menos profundidad de color (Gif), no se han tenido en cuenta.

Bibliografía.

[1] ***Mastering Delphi 6.*** Marco Cantú. "Ed. Sybex. Julio 2001.

[2] ***La cara oculta de Delphi 4,*** Ian Marteens. . Ed. Danysoft Internacional. 1998.

[3] ***Delphi Developers' Handbook,*** Marco Cantù, Tim Gooch, John F. Lam. Ed. Sybex. 1998.

[4] **Protocolos de Internet, Diseño e implementación en sistemas Unix**. Angel López, Alejandro Novo, Ed. RA-MA, Noviembre 1999

[5] **Página Oficial de Borland**. *http://www.borland.com/delphi*

[6] **The Delphi Super Page**. *http://delphi.icm.edu.pl*

[7] **Torry's Delphi**. *http://www.torry.net/index.htm.*

[8] **The Indy Project**. *http://www.nevrona.com/Indy*

[9] **Boletín Pascal**. *http://www.latiumsoftware.com/es/pascal/index.php*.

[10] **Club Delphi** *http://www.clubdelphi.com*

[11] **Trucomanía** *http://www.q3.nu*

[12] **El rinconcito de Delphi** *http://www.rinconcitodelphi.com*

[13] **Delphi Heaven**. *http://www.delphiheaven.com*

[14] **Delphi-Neftalí**, *http://neftali.clubdelphi.com*. (©)2005, Germán Estévez