2. Análisis Previo.

2.1. Decisión del entorno de desarrollo a usar.

2.2. Elección de la base de datos.

2. Análisis Previo.

En este capítulo vamos a ver los motivos por los que se han usado una herramienta de desarrollo y un formato de bases de datos determinados. Se harán comparativas y se mostrarán datos lógicos que justifiquen las decisiones.

2.1. Decisión del entorno de desarrollo a usar.

La primera decisión que se toma a la hora de realizar este proyecto, es usar un entorno RAD (desarrollo rápido de aplicaciones). Esto se debe a que son los entornos más usados hoy en día para generar aplicaciones para Windows, debido a su comodidad y a que el diseño se lleva a cabo con mayor celeridad, pudiendo el desarrollador centrarse en otras cuestiones. Esta elección hace que la oferta de entornos de desarrollo se limite de forma notable, y nos queden tres opciones que son de las más importantes actualmente en el mercado:

- Microsoft Visual Basic, se programa en un lenguaje que es una evolución de Visual Basic.
- Borland Delphi, que tiene Object Pascal como lenguaje de programación.
- Borland C++ Builder, como indica su nombre, usa C++.

Es importante reseñar que Visual C++ queda completamente descartado (a pesar de ser muy usado y una gran herramienta) por no ser un entorno de tipo RAD. Pero hay otros motivos:

- A diferencia de los tres entornos mencionados antes, no facilita el acceso a bases de datos con un juego de componentes o asistentes.
- Con Visual C++, y para desarrollar una aplicación destinada a bases de datos, se pierde demasiado tiempo en tareas como creación de interfaces, gestión de mensajes del sistema, etc… Con las herramientas RAD se puede dedicar más tiempo al análisis, planificación y diseño.

- Visual C++ es famoso por ser un entorno más duro para programar que cualquier herramienta de tipo RAD.

Una vez descartado Visual C++, vamos a proceder a comentar porqué no se usó Visual Basic:

- En Visual Basic se usa un lenguaje evolución del clásico Basic. El autor estaba familiarizado desde antes con C++, gracias a la asignatura "Computadores" de cuarto curso de carrera.
- Con Visual Basic es más difícil hacer cosas de más bajo nivel, ya que simplemente no está preparado para ello. Es una herramienta que facilita mucho las cosas, pero no permite hacer cosas muy complicadas. Para ello hay que recurrir a Visual C++.
- Visual Basic usa un lenguaje orientado a objetos. Pero sus objetos no soportan todas las características de los objetos reales. De hecho, en la última versión de Visual Basic (.NET), al estar éste integrado plenamente en la plataforma .NET, que sí soporta objetos reales, han surgido problemas para migrar una aplicación de un Visual Basic anterior hasta el .NET. Incluso Microsoft ha publicado un manual para llevar a cabo dicha migración. Esta recientísima versión de Visual Basic también ha sido descartada, en este caso por la falta de documentación abundante. La guía de Microsoft para migrar una aplicación de Visual Basic 6 a Visual Basic .NET se puede observar en la siguiente dirección de Internet: http://msdn.microsoft.com/vbasic/techinfo/articles/upgrade/vbupgrade.asp.

Finalmente queda por hacer la elección entre Delphi y C++ Builder. Se ha buscado mucho por Internet información para deshacer esta duda. Pero se ha llegado a la conclusión de que es cuestión de gustos. Diferentes autores dan distintas razones para usar uno u otro. En algunos foros se observan interminables (y violentas) charlas sobre estos asuntos, sin llegar a ninguna conclusión clara. Parece que C++ Builder proporciona algo más de flexibilidad que Delphi, pero Delphi hace las cosas de una forma más fácil. Por tanto, para tomar la decisión en el presente proyecto de fin de carrera se tuvieron en cuenta motivos no muy técnicos:

- El autor ya estaba familiarizado con C++, por ello la curva de aprendizaje con C++ Builder sería menor que con Delphi y podría dedicar más tiempo al desarrollo efectivo de la aplicación.
- La cantidad de documentación sobre C++ Builder en el entorno del autor era abrumadora, sin embargo la información sobre Delphi brillaba por su ausencia.

Por estos dos motivos, y por las consultas realizadas por el autor a algunos profesionales de la programación, se decidió adoptar C++ Builder como herramienta de desarrollo para el presente proyecto de fin de carrera.

2.2. Elección de la base de datos.

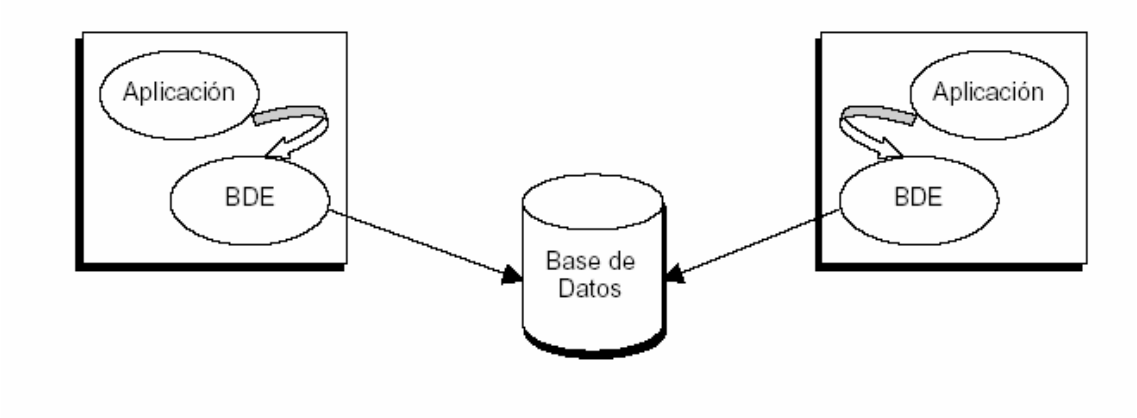
Antes de comenzar a hablar sobre los motivos que llevaron a una elección, realizaremos una breve introducción al mundo de las bases de datos, ya que ello nos proporcionará información importante para tomar una decisión en uno u otro sentido.

La primera gran división entre los sistemas de bases de datos existentes se produce entre los sistemas 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 adecuado, porque más adelante estos sistemas crecieron para permitir su explotación en red. Tampoco es apropiado calificarlas 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 con una sola capa. Todos estos sistemas utilizan como interfaz de aplicaciones un motor de datos que se implementa como una DLL.

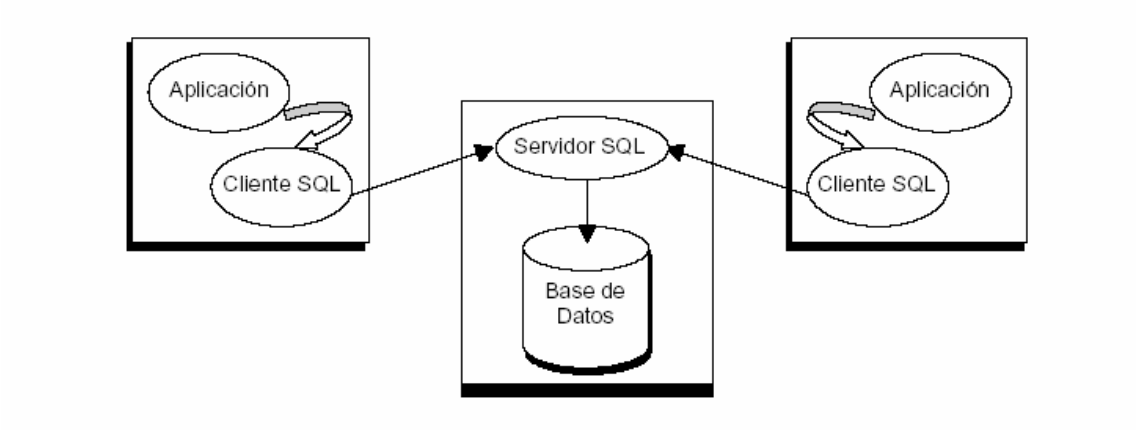
Obsérvese el siguiente diagrama, que representa el uso en red de una base de datos “de escritorio”:



Aunque la segunda burbuja dice “BDE”, el motor de bases de datos de Borland, sustituya estas siglas por DAO, y podrá aplicar el diagrama a Access. Se ha 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 ámbito para el acceso a la base de datos “física”. Es como si las dos aplicaciones deambularan a ciegas en cuarto oscuro, tratando de sentarse en algún sitio libre. Por supuesto, la única forma que tienen de saberlo es intentar hacerlo y confiar 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 estamos pensando en decenas de usuarios atacando simultáneamente a sus datos, y en ejércitos de extremidades inferiores listas para tropezar con cables, debemos olvidarnos de Paradox, dBase y Access, pues se necesitaría una base de datos cliente/servidor. En el caso de SIGPYME, el programa está pensado para que sólo un usuario utilice la aplicación a la vez. De hecho incluso se prohibirá que haya más de una instancia de programa corriendo de forma simultánea. Pero sigamos hablando de bases de datos.

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:



El par aplicación + motor local de datos ya no tiene acceso directo a los fichero de la base de datos, pues hay un nuevo actor: el servidor SQL. Se ha cambiado el rótulo de la burbuja del motor de datos, y ahora dice: “cliente SQL”. Ésta es una denominación genérica. Para las aplicaciones desarrolladas con C++ Builder y el motor de datos de Borland, este cliente consiste en la combinación de BDE (Borland Database Engine) 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 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 ser que algún día aparezca un sistema que mezcle de otra forma estor rasgos definitorios.

Tras este breve análisis de las bases de datos, y como ya hemos comentado antes, queda claro que debemos decidirnos por una base de datos de escritorio, ya que SIGPYME no está pensado para ser usado por varios usuarios, ya que no es necesario en el marco en el que se encuadra: una PYME en la que no será necesario tener más de una persona gestionando los clientes o las citas de la empresa de forma simultánea.

En bases de datos de escritorio tenemos tres opciones con las que trabaja C++ Builder: Paradox, dBASE y Access.

Hemos pensado descartar dBASE porque es un modelo que ha quedado obsoleto en la actualidad, de hecho las aplicaciones dBASE han quedado en desuso hoy en día.

Access podría ser un candidato, y de hecho se intentó en su día pero quedó descartado por estos motivos:

- Microsoft no mantiene compatibilidad entre los distintos formatos de Access. Por ejemplo, una base de datos Access 2000 no la puede leer la aplicación Access 97. La solución a esto podría ser usar el formato de Access más antiguo, para asegurar que el usuario la pueda leer con cualquier Access que tenga instalado en su sistema. Esto ya es un inconveniente serio. Todo esto suponiendo que el usuario desea manejar las bases de datos al margen de SIGPYME.
- Lo que podría parecer una ventaja es realmente un inconveniente. Access es un formato muy extendido en la actualidad en los PCs de todo el mundo. Esto hace que el usuario pueda manejar las bases de datos de forma ajena a SIGPYME y pueda introducir algún cambio en las bases de datos que no sea compatible con la estructura esperada por la aplicación.
- Una base de datos Access incluye la posibilidad de tener consultas prediseñadas, informes, etc… Y eso no se necesita en SIGPYME, ya que es una funcionalidad que proporciona la aplicación. Aunque no se usase esa posibilidad en la base de datos Access, sí ocupa algo más de espacio que otros modelos, como dBASE o Paradox.
- La interacción entre C++ Builder y Microsoft Access no es la adecuada. Tal vez esto se deba a la competencia existente entre ambas compañías. De hecho, C++ Builder no puede trabajar simultáneamente con más de una versión de Access. Por ejemplo, si usa bases de datos de Access 97 no puede usar las del 2000. Esto se debe a que hace uso de unas librerías dinámicas DLL que proporciona Microsoft junto con cada versión de Access (o Visual Basic). Estas DLL se especifican en la utilidad "BDE Administrator" que se instala junto con C++ Builder, y, si se quiere cambiar la DLL a usar, obligaríamos al usuario a tener dicho programa instalado. Suponiendo que el programa estuviese instalado, podríamos hacer el cambio desde SIGPYME según el juego de bases de datos que ese esté usando, lo cual puede ser terriblemente tedioso y complicado.

Por todo esto, se ha decidido usar como base de datos en SIGPYME el formato Paradox. Algunos motivos son:

- Es el formato con el que más cómodo se encuentra C++ Builder. Se nota en la perfecta integración con el BDE.
- No es tan común como Access.
- Access puede leer una tabla de Paradox, pero no la puede guardar en ese mismo formato, lo que hace que le usuario no pueda dañar el juego de bases de datos de SIGPYME.
- Grandes programadores lo usan, como Ian Marteens, así lo dice en su libro "La cara oculta de C++ Builder", que se menciona en la bibliografía.
- Una base de datos de Paradox contiene sólo las tablas y los ficheros para mantener los índices. No hay información adicional extra (consultas, informes…).