1.- Marco teórico

1.1.- Plataforma .Net

1.1.1.- Introducción

En junio de 2000 Microsoft anunció la iniciativa de .Net, una gran nueva visión para estrechar el Internet y la Web en el desarrollo, la ingeniería y el uso del software. Un aspecto crucial en la estrategia de .Net es su independencia de un lenguaje específico o plataforma. En vez de forzar a los desarrolladores a utilizar un solo lenguaje de programación, los desarrolladores pueden crear una aplicación .NET en cualquier lenguaje compatible. Los programadores pueden contribuir al mismo proyecto de software, escribiendo código en los lenguajes de .NET (Como C#, Visual C++ .NET, Visual Basic .NET y muchos más) que mejor dominen. Una parte de la iniciativa incluye la tecnología .NET Active Server Pages (ASP) de Microsoft, la cual permite a los programadores crear aplicaciones Web.

La plataforma .NET puede existir en múltiples plataformas, ampliando la portabilidad de los programas en .NET. Además la estrategia de .NET incluye un nuevo proceso de desarrollo de programa el cual podría cambiar la manera en que los programas son escritos y compilados, llevando a incrementar la productividad.

Un componente principal de la arquitectura de .NET son los servicios Web (Web Services), que son aplicaciones que pueden ser utilizadas en Internet. Clientes y otras aplicaciones pueden utilizar estos servicios Web como bloques de construcción reutilizable. La estrategia de .NET amplia el concepto de la reutilización de código en Internet, permitiendo a los programadores concentrarse en sus especialidades sin tener que implementar cada componente en cada aplicación.

1.1.2.- .NET Framework

.NET podría considerarse como una respuesta de Microsoft al creciente mercado de los negocios en entornos Web, como competencia a la plataforma Java de Oracle y a los diversos Framework de desarrollo Web basados en PHP. Su propuesta es ofrecer una manera rápida y económica, a la vez que segura y robusta, para desarrollar aplicaciones o como la misma plataforma las denomina, soluciones que permiten una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

NET Framework es un componente integral de Windows que admite la compilación y la ejecución de la siguiente generación de aplicaciones y servicios Web XML. El diseño de .NET Framework está enfocado a cumplir los objetivos siguientes:

- Proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se pueda almacenar y ejecutar de forma local, ejecutar de forma local pero distribuida en Internet o ejecutar de forma remota.
- Proporcionar un entorno de ejecución de código que reduzca al máximo posible la implementación de software y los conflictos de versiones.
- Ofrecer un entorno de ejecución de código que promueva la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza.
- Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan scripts o intérpretes de comandos.
- Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes, como las basadas en Windows o en el Web.
- Basar toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se puede integrar con otros tipos de código.

.NET Framework contiene dos componentes principales: Common Language Runtime y la biblioteca de clases de .NET Framework. Common Language Runtime (Motor en tiempo de ejecución) es el fundamento de .NET Framework. CLR administra la memoria, ejecución de subprocesos, ejecución de código, comprobación de la seguridad del código, compilación y demás servicios del sistema. Estas características son intrínsecas del código administrado que se ejecuta en Common Language Runtime. La biblioteca de clases, el otro componente principal de .NET Framework, es una completa colección orientada a objetos de tipos reutilizables que se pueden emplear para desarrollar aplicaciones que abarcan desde las tradicionales herramientas de interfaz gráfica de usuario (GUI) o de línea de comandos hasta las aplicaciones basadas en las innovaciones más recientes proporcionadas por ASP.NET, como los formularios Web Forms y los servicios Web XML. En la siguiente imagen (figura 1.1) se muestra la arquitectura del .Net Framework.

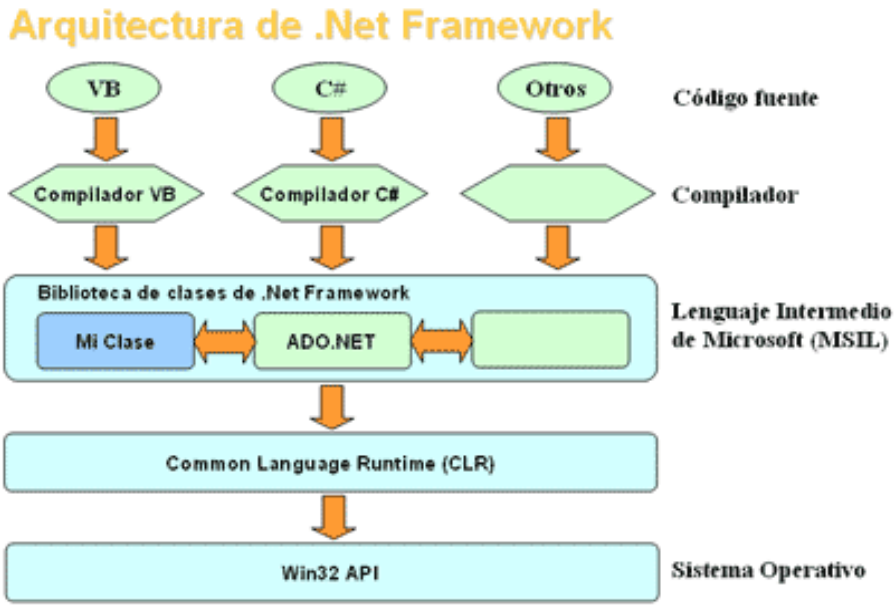

Figura 1.1
Arquitectura de .Net Framework.

Actualmente, el Framework de .Net es una plataforma no incluida en los diferentes sistemas operativos distribuidos por Microsoft, por lo que es necesaria su instalación previa a la ejecución de programas creados mediante .Net. El Framework se puede descargar gratuitamente desde la web oficial de Microsoft.

.Net Framework soporta múltiples lenguajes de programación y aunque cada lenguaje tiene sus características propias, es posible desarrollar cualquier tipo de aplicación con cualquiera de estos lenguajes. Existen varios lenguajes adaptados a .Net, desde los más conocidos como C# (C Sharp), Visual Basic o C++ hasta otros lenguajes menos conocidos como Perl o Cobol.

1.1.3.- Common Language Runtime

El motor en tiempo de ejecución se puede considerar como un agente que administra el código en tiempo de ejecución y proporciona servicios centrales, como la administración de memoria, la administración de subprocesos y la comunicación remota, al tiempo que aplica una seguridad estricta a los tipos y otras formas de especificación del código que promueven su seguridad y solidez. De hecho, el concepto de administración de código es un principio básico del motor en tiempo de ejecución. El código destinado al motor en tiempo de ejecución se denomina código administrado, a diferencia del resto de código, que se conoce como código no administrado.

Los programas son compilados dentro de instrucciones específicas de máquina en dos pasos. Primeramente el programa es compilado dentro del Microsoft Intermediate lenguaje (MSIL), el cuál define instrucciones para el CLR. El código de otros lenguajes y fuentes convertidos dentro del MSIL pueden ser mezclados por la CLR. Entonces otro compilador en el CLR compila el MSIL en código máquina, creando una aplicación simple.

¿Por qué molestarse en un paso extra convirtiendo de C# a MSIL, en vez de compilar directamente a lenguaje maquina? Las principales razones son portabilidad entre sistemas operativos, interoperabilidad entre lenguajes y características de administración de ejecución como la gestión de la memoria y seguridad.

Además, el motor en tiempo de ejecución impone la solidez del código mediante la implementación de una infraestructura estricta de comprobación de tipos y código denominado CTS (Common Type System, Sistema de tipos común). CTS garantiza que todo el código administrado es auto descriptivo. Los diversos compiladores de lenguaje de Microsoft y de otros fabricantes generan código administrado que se atiene al CTS. Esto significa que el código administrado puede consumir otros tipos e instancias administrados, al tiempo que se exige fidelidad de tipos y seguridad de tipos estrictamente.

Además, el entorno administrado del motor en tiempo de ejecución elimina muchos problemas de software comunes, por ejemplo controla automáticamente la disposición de los objetos, administra las referencias a éstos y los libera cuando ya no se utilizan. Esta administración automática de la memoria soluciona los dos errores más comunes de las aplicaciones: la pérdida de memoria y las referencias no válidas a la memoria.

También, CLR aumenta la productividad del programador. Por ejemplo, los desarrolladores pueden crear aplicaciones en el lenguaje que prefieran y seguir sacando todo el provecho del CLR, la biblioteca de clases y los componentes escritos en otros lenguajes por otros colegas. El proveedor de un compilador puede elegir destinarlo al motor en tiempo de ejecución. Los compiladores de lenguajes que se destinan a .NET

Framework hacen que las características de .NET Framework estén disponibles para el código existente escrito en dicho lenguaje, lo que facilita enormemente el proceso de migración de las aplicaciones existentes.

El CLR está diseñado para mejorar el rendimiento. Aunque Common Language Runtime proporciona muchos servicios estándar de motor en tiempo de ejecución, el código administrado nunca se interpreta. Una característica denominada compilación JIT (Just-In-Time) permite ejecutar todo el código administrado en el lenguaje máquina nativo del sistema en el que se ejecuta. Mientras tanto, el administrador de memoria evita que la memoria se pueda fragmentar y aumenta la zona de referencia de la memoria para mejorar aún más el rendimiento.

Por último, el motor en tiempo de ejecución se puede hospedar en aplicaciones de servidor de gran rendimiento, como Microsoft® SQL Server™ e Internet Information Services (IIS). Esta infraestructura permite utilizar código administrado para escribir lógica empresarial, al tiempo que se disfruta del superior rendimiento de los mejores servidores empresariales del sector que pueda hospedar el CLR.

1.1.4.- Lenguaje intermedio de Microsoft (MSIL).

El código generado por el compilador de C# está escrito en un lenguaje llamado Lenguaje intermedio de Microsoft o MSIL. MSIL se compone de un conjunto específico de instrucciones que especifican como debe ser ejecutado el código.
Contiene instrucciones para operaciones como la inicialización de variables, los métodos de llamada a objetos y el control de errores, por citar unos pocos.
MSIL no es un conjunto de instrucciones específicas para una CPU física, MSIL no sabe nada de la CPU de su equipo y su equipo nada de MSIL. Entonces ¿cómo se ejecuta el código .NET, si su equipo no la puede interpretar? La respuesta es que el código MSIL se convierte en código específico de CPU cuando se ejecuta por primera vez. Este proceso se llama compilación “Justo a tiempo” o JIT. El trabajo de un compilador JIT es convertir el código genérico MSIL en código que su equipo pueda ejecutar en su CPU. Pero ¿por qué generar MSIL cuando un compilador podría generar directamente código específico para la CPU?, la respuesta es porque MSIL permite que el código compilado se transfiera fácilmente a hardware diferente.
Al instalar .NET Framework en su ordenador personal se incluye un compilador JIT que convierte el MSIL en código específico para la CPU de su ordenador personal. Al instalar .NET Framework en un nuevo ordenador se incluye un compilador JIT que convertiría el mismo MSIL, instalado en el ordenador anterior, en código específico para la CPU de este nuevo ordenador. Ahora tiene un solo código base MSIL que puede ejecutarse en cualquier equipo que tenga un compilador JIT .NET.

1.1.5.- Biblioteca de clases de .NET Framework

La biblioteca de clases de .NET Framework es una colección de tipos reutilizables que se integran estrechamente con la Common Language Runtime. La biblioteca de clases está orientada a objetos, lo que proporciona tipos de los que su propio código administrado puede derivar funciones. Esto ocasiona que los tipos de .NET Framework sean sencillos de utilizar y reduce el tiempo asociado con el aprendizaje de las nuevas características de .NET Framework. Además, los componentes de terceros se pueden integrar sin dificultades con las clases de .NET Framework.

Por ejemplo, las clases de colección de .NET Framework implementan un conjunto de interfaces que se puede usar para desarrollar sus propias clases de colección. Estas se combinarán fácilmente con las clases de .NET Framework.

Como en cualquier biblioteca de clases orientada a objetos, los tipos de .NET Framework permiten realizar diversas tareas de programación comunes, como son la administración de cadenas, recolección de datos, conectividad de bases de datos y acceso a archivos. Además de estas tareas habituales, la biblioteca de clases incluye tipos adecuados para diversos escenarios de desarrollo especializados. Por ejemplo, puede utilizar .NET Framework para desarrollar los siguientes tipos de aplicaciones y servicios:

- Aplicaciones de consola.
- Aplicaciones GUI de Windows (Windows Forms).
- Aplicaciones de Windows Presentation Foundation (WPF).
- Aplicaciones de ASP.NET.
- Servicios Web.
- Servicios de Windows.
- Aplicaciones orientadas a servicios utilizando Windows Communication Foundation (WCF).

Por ejemplo, las clases de Windows Forms son un conjunto completo de tipos reutilizables que simplifican enormemente el desarrollo de interfaces GUI para Windows. Si escribe una aplicación Web Form de ASP.NET, puede utilizar las clases de formularios Web Forms.

1.2.- UML

1.2.1.- Introducción

Durante varios años la Ingeniería de software no contó con un estándar que le permitiera la construcción de planos de software que pudieran ser interpretados por cualquier ingeniero del área. Se realizaron varios intentos, uno de ellos fue la publicación en 1976 por parte del CCITT, el organismo internacional para la estandarización en el área de las telecomunicaciones, del Lenguaje de Especificación y Descripción (Specification and Description Language, SDL) para el comportamiento funcional de los sistemas de telecomunicación. Dicho lenguaje es un estándar de modelado de objetos especializado, desarrollado cuando el paradigma de la Orientación a Objetos no había madurado, por lo tanto no contó con la acogida que se esperaba.

Fue sólo hasta el 14 de noviembre de 1997 cuando el Grupo Administrador de Objetos (Object Management Group, OMG) publicó como estándar la versión 1.1 del Lenguaje Unificado de Modelado (Unified Modeling Language, UML).
En el caso concreto de UML se dice que es un lenguaje estándar para escribir planos de software, que se utiliza para visualizar, especificar, construir y documentar los artefactos de un mismo sistema que involucran una gran cantidad de software. Su alfabeto está constituido por elementos y relaciones, los cuales al combinarse cobran sentido al armar una colección de palabras formando diferentes tipos de diagramas. Está pensado para usarse con todos los métodos de desarrollo. Etapas de ciclo de vida, dominios de aplicación y medios. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código así como generadores de informes. La especificación de UML no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos.

1.2.2.- Vista estática.

La vista estática modela los conceptos del dominio de la aplicación, así como los conceptos internos creados como parte de la implementación de la aplicación. Esta visión es estática porque no describe el comportamiento del sistema dependiente del tiempo, que se describe en otras vistas. Los componentes principales de la vista estática son las clases y sus relaciones: asociación, generalización, y varias clases de dependencia, tales como realización y uso. Una clase es la descripción de un concepto del dominio de la aplicación o de la solución de la aplicación. Las clases se dibujan como rectángulos, los cuales van a contener atributos y métodos. Los atributos nos describen las características que tiene nuestra clase y los métodos nos dicen cómo se va a comportar los objetos de la clase.

La figura 1.2 muestra un ejemplo de la notación del UML que captura los atributos y acciones de una lavadora. Un rectángulo es el símbolo que representa a la clase, y se divide en tres áreas. El Área superior contiene el nombre, el área central contiene los atributos, y el área inferior las acciones. Un diagrama de clases está formado por varios rectángulos de este tipo conectados por líneas que muestran la manera en que las clases se relacionan entre sí.

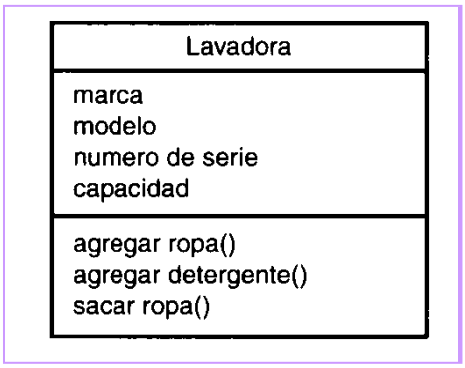

Figura 1.2 El símbolo UML de una clase

1.2.3.- Casos de uso.

Los casos de uso modelan la funcionalidad del sistema según como lo perciben los usuarios externos, llamados actores. Un caso de uso es una unidad coherente de funcionalidad, expresada como transacción entre los actores y el sistema. El propósito de los diagramas de casos de uso es enumerar a los actores y los casos de uso, y demostrar qué actores participan en cada caso de uso. Se emplean para visualizar el comportamiento del sistema, una parte de él o de una sola clase. El diagrama de uso es muy útil para

definir como debería ser el comportamiento de una parte del sistema, ya que solo especifica cómo deben comportarse y no como están implementadas las partes que define.

Un caso de uso puede participar en varias relaciones, además de poderse asociar con actores (Ver Figura 1.3).
Un caso de uso se dibuja como una elipse con su nombre dentro o debajo de ella. Se conecta por líneas con trazo continuo con los actores que se comunican con ella.

Relación	*Función*	*Notación*
asociación	La línea de comunicación entre un actor y un caso de uso en el que participa	———
extensión	La inserción de comportamiento adicional en un caso de uso base que no tiene conocimiento sobre él	«**extend**» - - - >
generalización de casos de uso	Una relación entre un caso de uso general y un caso de uso más específico, que hereda y añade propiedades a aquél	——▷
inclusión	Inserción de comportamiento adicional en un caso de uso base, que describe explícitamente la inserción	«**include**» - - - >

Figura 1.3 Tipos de relaciones de Casos de Uso

Utilizamos una lavadora, obviamente, para lavar su ropa. La figura 1.4 le muestra cómo representaría esto en un diagrama de casos de uso UML.

Figura 1.4 Diagrama de casos de uso UML

1.2.4.- Diagrama de Secuencia

Diagrama de secuencia representa una interacción como un gráfico bidimensional. La dimensión vertical es el eje de tiempo, que avanza hacia abajo de la página. La dimensión horizontal muestra los roles de clasificador que representan objetos individuales en la colaboración. Cada rol de clasificador se representa mediante una columna vertical la cual es la línea de vida. Durante el tiempo que no existe un objeto, el rol se muestra como una línea discontinua. Durante el tiempo que dura una activación de un procedimiento en el objeto, la línea de vida se dibuja como una línea doble.

Un mensaje se muestra como una flecha desde la línea de vida de un objeto a la de otro objeto. Las flechas se organizan en el diagrama en orden cronológico hacia abajo.

Un uso de un diagrama de secuencia es mostrar la secuencia del comportamiento de un caso de uso. Cuando está implementado el comportamiento, cada mensaje en un diagrama de secuencia corresponde a una operación en una clase, a un evento disparador.

Continuando con el ejemplo de la lavadora, entre los componentes de la lavadora se encuentran: una manguera, un tambor (donde se coloca la ropa) y un sistema de drenaje. Por supuesto, estos también son objetos (como verá, un objeto puede estar conformado por otros objetos).
Si damos por hecho que la lavadora completó las operaciones "agregar ropa", "agregar detergente" y "activar", la secuencia seria más o menos así:

1. El agua empezará a llenar el tambor mediante una manguera.

2. El tambor permanecerá inactivo durante cinco minutos.

3. La manguera dejará de abastecer agua.

4. El tambor girará de un lado a otro durante quince minutos.

5. El agua jabonosa saldrá por el drenaje.

6. Comenzará nuevamente el abastecimiento de agua.

7. El tambor continuará girando.

8. El abastecimiento de agua se detendrá.

9. El agua del enjuague saldrá por el drenaje.

10. El tambor girará en una sola dirección y se incrementará su velocidad por cinco minutos.

11. El tambor dejará de girar y el proceso de lavado habrá finalizado.

La figura 1.5 presenta un diagrama de secuencias que captura las interacciones que se realizan a través del tiempo entre el abastecimiento de agua, el tambor y el drenaje (representados como rectángulos en la parte superior del diagrama). En este diagrama el tiempo se da de arriba hacia abajo.

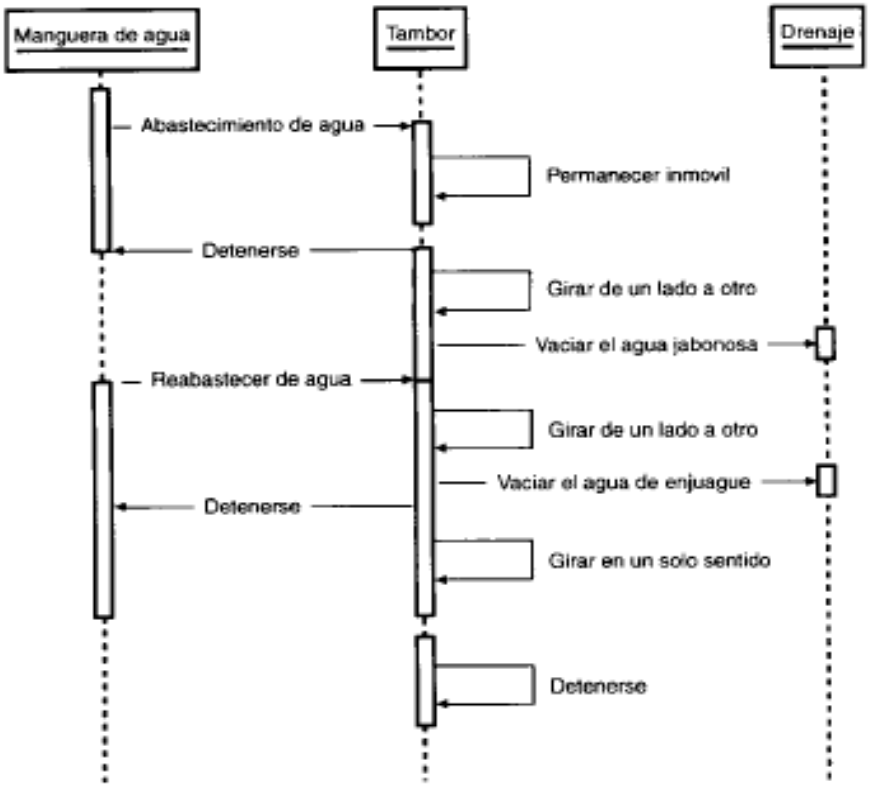

Figura 1.5 Diagrama de secuencia

Podríamos caracterizar los pasos 1 y 2 como el estado de remojo, 3 y 4 como el estado de lavado, 5 a 7 como el estado de enjuague y del 8 al 10 como el estado de centrifugado.

1.2.5.-Diagrama de colaboración

Una colaboración modela los objetos y los enlaces significativos dentro de una interacción. Los objetos y enlaces son significativos solamente en el contexto proporcionado por la interacción. Los mensajes se muestran como flechas ligadas a las líneas de la relación, que preceden a las descripciones del mensaje.

Un uso de un diagrama de colaboración es mostrar la implementación de una operación. La colaboración muestra los parámetros y las variables locales de la operación.

Los diagramas de colaboración y los diagramas de secuencia muestran interacciones, pero acentúan aspectos diferentes. Los diagramas de secuencia muestran secuencias de tiempo claramente pero no muestran explícitamente relaciones de objetos. Los diagramas de colaboración muestran las relaciones entre objetos claramente, pero las secuencias de tiempo se deben obtener a partir de números de secuencia. Los diagramas de secuencia son a menudo más útiles para mostrar escenarios; los diagramas de colaboración son a menudo más útiles para mostrar el diseño detallado de procedimientos.

Este ejemplo agrega un cronómetro interno al conjunto de clases que constituyen a una lavadora. Luego de cierto tiempo, el cronómetro detendrá el flujo de agua y el tambor comenzará a girar de un lado a otro. Ver figura 1.6

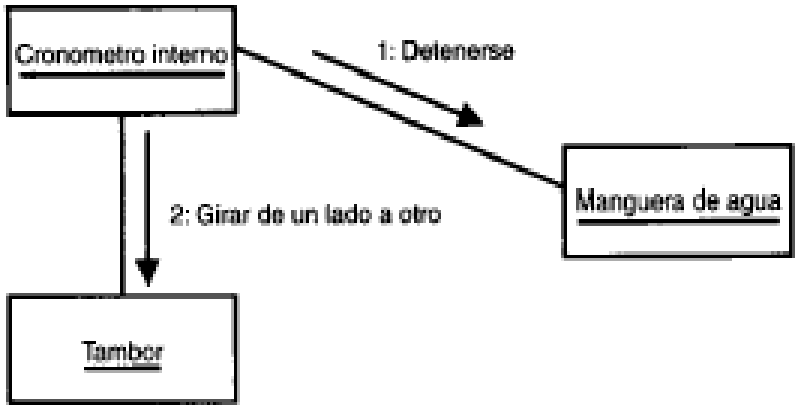

Figura 1.6 Diagrama de Colaboraciones UML.

1.2.6.- Diagrama de distribución

El diagrama de distribución UML muestra la arquitectura física de un sistema informático. Puede representar los equipos y dispositivos, mostrar sus interconexiones y el software que se encontrará en cada máquina. Cada computadora está representada por un cubo y las interacciones entre las computadoras están representadas por líneas que conectan a los cubos. La figura 1.7 presenta un ejemplo.

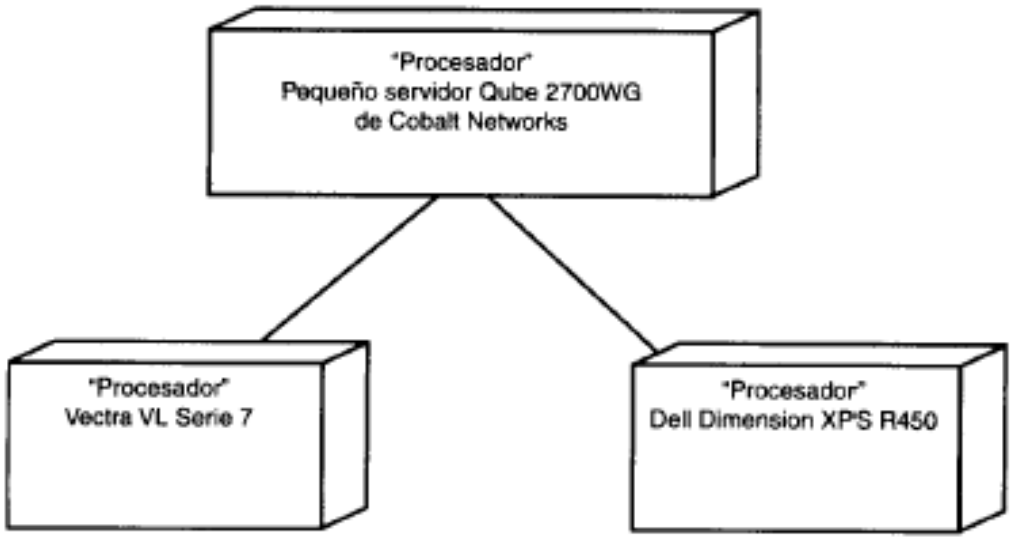

Figura 1.7
Diagrama de Distribución

1.3.- Bases de datos.

1.3.1.- Introducción

El objetivo principal de las bases de datos es el de unificar los datos que se manejan y los programas o aplicaciones que los manejan. Anteriormente los programas se codificaban junto con los datos, es decir, se diseñaban para la aplicación concreta que los iba a manejar, lo que desembocaba en una dependencia de los programas respecto a los datos, ya que la estructura de los ficheros iba incluida dentro del programa, y cualquier cambio en la estructura del fichero provocaba modificar y recompilar programas. Además, cada aplicación utiliza ficheros que pueden ser comunes a otras de la misma organización, por lo que se produce una "redundancia" de la información, que provoca mayor ocupación de memoria, laboriosos programas de actualización (unificar datos recogidos por las aplicaciones de los diferentes departamentos), e inconsistencia de datos (no son correctos), si los datos no fueron bien actualizados en todos los programas. Con las bases de datos, se busca independizar los datos y las aplicaciones, es decir, mantenerlos en espacios diferentes. Los datos residen en memoria y los programas mediante un sistema gestor de bases de datos, manipulan la información. El sistema gestor de bases de datos recibe la petición por parte del programa para manipular los datos y es el encargado de recuperar la información de la base de datos y devolvérsela al programa que la solicitó. Cada programa requerirá de una cierta información de la base de datos, y podrá haber otros que utilicen los mismos datos, pero realmente residirán en el mismo espacio de almacenamiento y los programas no duplicarán esos datos, si no que trabajarán directamente sobre ellos concurrentemente. Aunque la estructura de la base de datos cambiara, si los datos modificados no afectan a un programa específico, éste no tendrá por qué ser alterado. Estas técnicas de base de datos se pretenden conseguir a través del Sistema Gestor de Bases de Datos (SGBD):

• INDEPENDENCIA de los Datos: Cambios en la estructura de la Base de Datos no modifican las aplicaciones.

• INTEGRIDAD de los Datos: Los datos han de ser siempre correctos. Se establecen una *serie de restricciones (reglas de validación) sobre los datos.*

• SEGURIDAD de los Datos: Control de acceso a los datos para evitar manipulaciones de estos no deseadas.

1.3.2.- Definición

Una base de datos es un “almacén” que nos permite guardar grandes cantidades de información de forma organizada para que luego podamos encontrar y utilizar fácilmente.

Desde el punto de vista informático, la base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos.

Lo que debemos tener claro es la diferencia entre Base de Datos y SGBD. La base de datos es el almacenamiento donde residen los datos. El SGBD es el encargado de manipular la información contenida en ese almacenamiento mediante operaciones de lectura/escritura sobre la misma. Además las bases de datos no sólo contendrán las tablas (ficheros) de datos, sino que también almacenará formularios (interfaces para edición de datos), consultas sobre los datos, e informes. El SGBD se encargará de manipular esos datos, controlar la integridad y seguridad de los datos, reconstruir y reestructurar la base de datos cuando sea necesario.
Cada base de datos se compone de una o más tablas que guarda un conjunto de datos. Cada tabla tiene una o más columnas y filas. Las columnas guardan una parte de la información sobre cada elemento que queramos guardar en la tabla, cada fila de la tabla conforma un registro.

1.3.3.- Características

Entre las principales características de los sistemas de base de datos podemos mencionar:

- Independencia lógica y física de los datos.
- Redundancia mínima.
- Acceso concurrente por parte de múltiples usuarios.
- Integridad de los datos.
- Consultas complejas optimizadas.
- Seguridad de acceso y auditoría.
- Respaldo y recuperación.
- Acceso a través de lenguajes de programación estándar.

1.3.3.1.-Ventajas de los SGBD

Mejora en la productividad:

El SGBD proporciona muchas de las funciones estándar que el programador necesita escribir en un sistema de ficheros. A nivel básico, el SGBD proporciona todas las rutinas de manejo de ficheros típicas de los programas de aplicación.

El hecho de disponer de estas funciones permite al programador centrarse mejor en la función específica requerida por los usuarios, sin tener que preocuparse de los detalles de implementación de bajo nivel.

Aumento de la concurrencia:

En algunos sistemas de ficheros, si hay varios usuarios que pueden acceder simultáneamente a un mismo fichero, es posible que el acceso interfiera entre ellos de modo que se pierda información o se pierda la integridad. La mayoría de los SGBD gestionan el acceso concurrente a la base de datos y garantizan que no ocurran problemas de este tipo.

1.3.3.2 Desventajas

Complejidad:

Los SGBD son conjuntos de programas que pueden llegar a ser complejos con una gran funcionalidad. Es preciso comprender muy bien esta funcionalidad para poder realizar un buen uso de ellos.

Coste del equipamiento adicional:

Tanto el SGBD, como la propia base de datos, pueden hacer que sea necesario adquirir más espacio de almacenamiento. Además, para alcanzar las prestaciones deseadas, es posible que sea necesario adquirir una máquina más grande o una máquina que se dedique solamente al SGBD. Todo esto hará que la implantación de un sistema de bases de datos sea más cara.

Vulnerable a los fallos:

El hecho de que todo esté centralizado en el SGBD hace que el sistema sea más vulnerable ante los fallos que puedan producirse. Es por ello que deben tenerse copias de seguridad (Backup).

1.3.4.- Tipos de Campos

Cada Sistema de Base de Datos posee tipos de campos que pueden ser similares o diferentes. Entre los más comunes podemos nombrar:

- **Numérico:** entre los diferentes tipos de campos numéricos podemos encontrar enteros “sin decimales” y reales “decimales”.
- **Booleanos:** poseen dos estados, Verdadero y Falso.
- **Fechas:** almacenan fechas facilitando posteriormente su explotación. Almacenar fechas de esta forma posibilita ordenar los registros por fechas o calcular los días entre una fecha y otra.
- **Alfanuméricos:** contienen cifras y letras. Presentan una longitud limitada (255 caracteres).
- **Autoincrementables:** son campos numéricos enteros que incrementan en una unidad su valor para cada registro incorporado. Su utilidad resulta: Servir de identificador ya que resultan exclusivos de un registro.

1.3.5.- Tipos de Base de Datos

Entre los diferentes sistemas gestores de base de datos, podemos encontrar los siguientes:

- **MySql:** es una base de datos con licencia GPL basada en un servidor. Se caracteriza por su rapidez. No es recomendable usar para grandes volúmenes de datos.
- **PostgreSql y Oracle:** Son sistemas de base de datos poderosos. Administra muy bien grandes cantidades de datos, y suelen ser utilizadas en intranets y sistemas de gran calibre.
- **Access:** Es una base de datos desarrollada por Microsoft. Esta base de datos, debe ser creada bajo el programa access, el cual crea un archivo .mdb.
- **Microsoft SQL Server:** es una base de datos más potente que Access desarrollada por Microsoft. Se utiliza para manejar grandes volúmenes de información.

1.3.6.- Modelo entidad-relación

Un diagrama o modelo entidad-relación (a veces denominado por su siglas, *E-R* "Entity relationship") es una herramienta para el modelado de datos de un sistema de información. Estos modelos expresan entidades relevantes para un sistema de información, sus inter-relaciones y propiedades.

Es una técnica para definir las necesidades de información de una organización. Proporciona una buena base para sistemas de alta calidad dirigidos a satisfacer las necesidades de su empresa.

Formalmente, los diagramas E-R son un lenguaje gráfico para describir conceptos como se puede ver en la figura 1.9. Informalmente, son simples dibujos o gráficos que describen la información que trata un sistema de información y el software que lo automatiza.

El objetivo de un modelo entidad relación es el de proporcionar un modelo preciso de las necesidades de información de una organización que actuarán como un marco de trabajo para el desarrollo de sistemas nuevos o mejorados.

El modelo es una técnica de la ingeniería de información que se utiliza para desarrollar un modelo de datos de alta calidad. El modelo de datos ofrece una forma estándar de definir los datos y las relaciones entre estos para todos los sistemas de información. Esto mejora enormemente la calidad del sistema e incrementa la productividad del software.

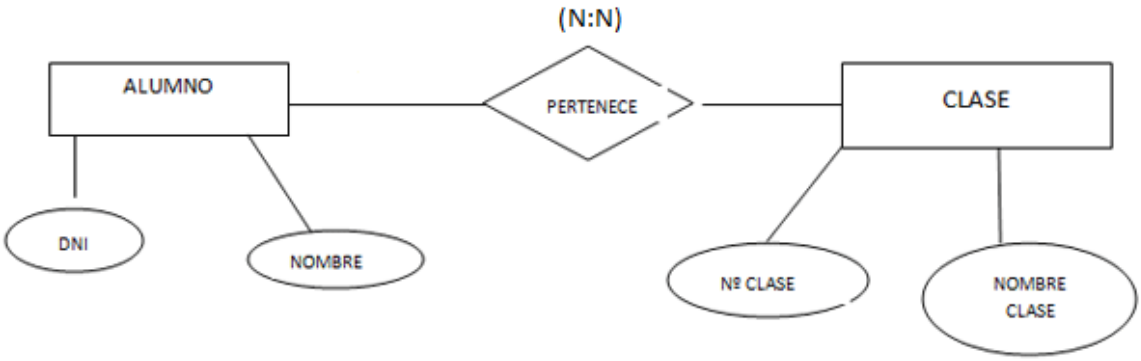

Figura 1.9 Ejemplo de lo que sería un diagrama entidad relación

1.3.6.1.- Entidades

Entidad.- Objeto del mundo real sobre el que queremos almacenar información, por ejemplo una persona.

El término entidad, cuando se le refiere en base de datos, es cualquier objeto sobre el que se tiene información. Una entidad está descrita por sus características. Por ejemplo, la entidad Persona lleva consigo las características de: Nombre, Apellido, Sexo, Estatura, Peso, Fecha de nacimiento, etc. Se representa mediante un rectángulo o "caja" etiquetada en su interior mediante un identificador. Ejemplos de entidades habituales en los sistemas de información son: factura, persona, empleado, etc.

Las entidades están compuestas de *atributos* que son los datos que definen el objeto (para la entidad persona serían ID, nombre, apellidos, dirección, etc). Entre los atributos habrá uno o un conjunto de ellos que no se repite; a este atributo o conjunto de atributos se le llama *clave* de la entidad, (para la entidad persona una clave seria ID). Ya que puede haber varias claves y necesitamos elegir una, lo haremos atendiendo a estas normas:

- Que sea única.
- Que se tenga pleno conocimiento de ella.

1.3.6.2.-Relaciones

Una relación describe cierta dependencia entre entidades. Se representa mediante un rombo etiquetado en su interior con un verbo. Este rombo se debe unir mediante líneas con las entidades (rectángulos) que relaciona.

Una relación tiene sentido al expresar las entidades que relaciona. Por ejemplo: una persona (entidad) trabaja para (relación) un departamento (entidad).

Una relación es la asociación entre entidades, sin existencia propia en el mundo real que estamos modelando, pero necesaria para reflejar las interacciones existentes entre entidades. Las relaciones pueden ser de tres tipos, que comúnmente se les llama cardinalidad, a continuación se describen:

- Relaciones 1-1.- Las entidades que intervienen en la relación se asocian una a una, ejemplo: la entidad HOMBRE, la entidad MUJER y entre ellos la relación MATRIMONIO.
- Relaciones 1-N.- Una ocurrencia de una entidad está asociada con muchas (N) de otra, ejemplo: la entidad EMPERSA, la entidad TRABAJADOR y entre ellos la relación TRABAJAR-EN.
- Relaciones N-N.-Cada ocurrencia, en cualquiera de las dos entidades de la relación, puede estar asociada con muchas (N) de la otra y viceversa, ejemplo: la entidad ALUMNO, la entidad MATERIA y entre ellos la relación MATRÍCULA.

1.3.6.3.- Atributos

Los atributos son propiedades relevantes propias de una entidad y/o relación. Se representan mediante un círculo o elipse etiquetado mediante un nombre en su interior. Cuando un atributo es identificativo de la entidad se suele subrayar dicha etiqueta.

Por motivos de legibilidad, los atributos no suelen representarse en un diagrama entidad-relación, sino que se describen textualmente en otros documentos adjuntos.

Los atributos describen información útil sobre las entidades. En particular, los atributos identificativos son aquellos que permiten diferenciar a una instancia de la entidad de otra distinta. Por ejemplo, el atributo identificativo que distingue a un empleado de otro es su número de la Seguridad Social

1.3.6.4.- Representación gráfica de Entidades y Relaciones

Para asimilar fácilmente un diseño de datos cuando se emplea el modelo E/R se utilizan los siguientes elementos gráficos:

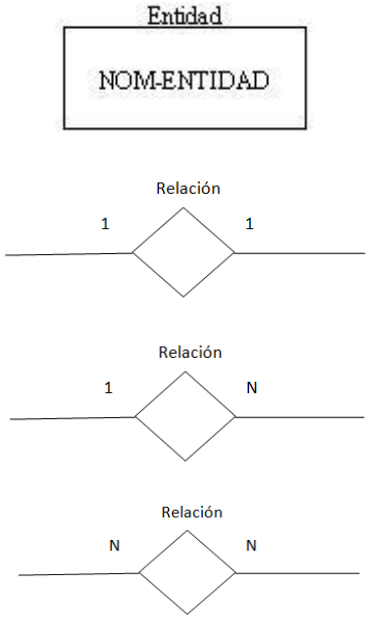

Figura 1.10 Elementos gráficos E/R

La utilización de estos elementos dará como resultado lo que se denomina el esquema entidad-relación de la base de datos. Los ejemplos que se incluyen en el apartado anterior, gráficamente quedarían como sigue, ver figura 1.11

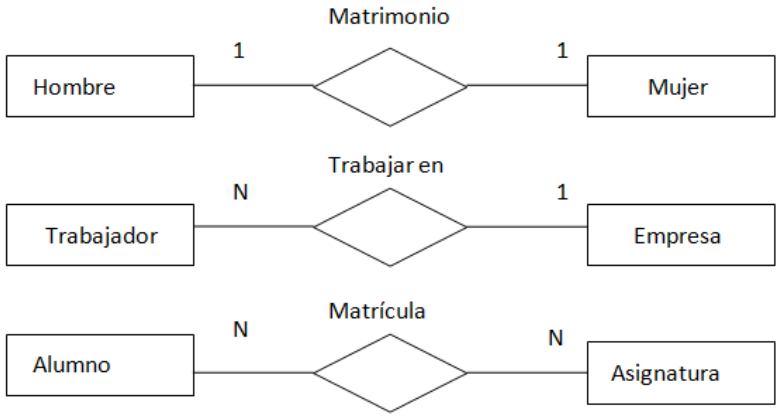

Figura 1.11 Modelo Entidad-Relación

1.3.7.- El lenguaje SQL

SQL es un lenguaje universal en los sistemas de base de datos. Este lenguaje nos permite realizar consultas a nuestras bases de datos para mostrar, insertar, actualizar y borrar datos.

A continuación veremos un ejemplo de ellos:

- **Mostrar:** para mostrar los registros se utiliza la instrucción “Select”.
  Select * From NombreTabla.
- **Insertar:** los registros pueden ser introducidos a partir de sentencias que emplean la instrucción “Insert”. Insert Into NombreTabla (titulo, texto, fecha) Values ('NombreTitulo', 'como esta', '22-10-2007')
- **Borrar:** Para borrar un registro se utiliza la instrucción Delete. En este caso debemos especificar cuál o cuáles son los registros que queremos borrar. Es por ello necesario establecer una selección que se llevará a cabo mediante la cláusula Where.
  Delete From NombreTabla Where id='1'.
- **Actualizar:** para actualizar los registros se utiliza la instrucción Update. Igual que en el caso de Delete, necesitamos especificar por medio de Where cuáles son los registros en los que queremos hacer efectivas nuestras modificaciones. Además, tendremos que especificar cuáles son los nuevos valores de los campos que deseamos actualizar. Update NombreTabla Set titulo='Mi Primer Comentario' Where id='1'.