DESARROLLO DE AGROMUTUA.WEB CON LINQ

Alumno: **Alejandro Medina Baeza**

Director: **Vicente Pelechano Ferragud**

TESINA DE MÁSTER DE INGENIERÍA DEL SW, MÉTODOS FORMALES Y SISTEMAS DE INFORMACIÓN
Universidad Politécnica de Valencia

ÍNDICE

AGRADECIMIENTOS

Quiero dar las gracias a Prodevelop SA por brindarme la posibilidad de desarrollar la tesina en la empresa y ampliar mis conocimientos profesionales. Lugar donde he encontrado a compañeros que me han realizado la estancia muy cómoda y agradable desde el primer día y que nunca han dudado en ayudarme en la medida de lo posible.

Quiero agradecer también la indispensable ayuda del tutor de la tesina, Vicente Pelechano, él ha sido el que me ha guiado y el que ha administrado mis esfuerzos para llegar a conseguir escribirla y alcanzar los objetivos fijados.

Y sobretodo quiero dar las gracias a mi familia, mis padres Manolo y Belén y a mi novia Sandra por estar siempre cerca de mí, apoyándome, ayudándome en todo y dándome la oportunidad de ser lo que hoy soy.

Gracias a todos.

1. PRESENTACIÓN DE LA TESINA

Durante el desarrollo de la tesina "Desarrollo de Agromutua.Web con LINQ" del Máster de Ingeniería del Software, Métodos Formales y Sistemas de Información de la UPV se va a explicar lo que ha supuesto para los desarrolladores de lenguajes de Microsoft (VB, C#, ASP.Net, etc.) la aparición del nuevo Framework 3.5 y en concreto la nueva característica de tratamiento de datos: LINQ.

Los datos, ya sean tablas en una base de datos, objetos en memoria o en formato XML siempre han sido una de las tareas más complejas y costosas para los programadores, incluyendo los desarrolladores de .NET que han tenido que lidiar mucho tiempo con costosas interfaces ADO para la interacción entre las aplicaciones y las bases de datos. Con LINQ se descubrirá una forma de acceder y interaccionar con los datos sencilla y muy similar a la sintaxis conocida SQL, alcanzando una gran integración entre lenguaje y datos. Aquí es donde encaja LINQ to Objects, LINQ to XML y LINQ to SQL, entre otros tipos LINQ.

LINQ, abreviatura de *Language Integrated Query* se incluye ya por defecto en VS2008 (viene con el .NET Framework 3.5) y proporciona una interfaz de interacción entre la programación y los datos, de esta forma va a permitir interaccionar con la base de datos y otros almacenes de datos de una forma más sencilla y menos laboriosa que antes, incluyendo consultas directamente en el código y nuevas características. Consultas formadas por expresiones parecidas al SQL estándar (Select, From, etc.) y incluyendo nuevas palabras reservadas que siguiendo un conjunto de reglas permitirán realizar las consultas en el propio código de la aplicación.

Con LINQ se deja de lado las tediosas y difíciles tareas de mantenimiento que tardaban eternidades en resolverse y que solo contribuían a generar problemas en el futuro con cualquier cambio en la base de datos. Así LINQ nos trae nuevas características del lenguaje VB.NET y C# añadiendo funcionalidad y reduciendo líneas de código (LINQ to Objects), pudiéndose complementar y orientar a diferentes tipos de LINQ como en el que se centra más la tesina que es que tiene que ver con la interacción con la base de datos (LINQ to SQL). Mapeando de manera sencilla la base de datos se puede realizar consultas integradas en el código y manipular las colecciones de datos.

El objetivo que se ha perseguido con la elaboración de este documento será analizar, comparar y enseñar el funcionamiento de LINQ, sus ventajas, sus puntos débiles y las características que nos depara en su evolución en el futuro basándonos en la experiencia que se ha brindado al poder trabajar en un proyecto empresarial como es el desarrollo de una aplicación desarrollada por la empresa Prodevelop S.A. y de la cual sirve tanto de ejemplo, como de aprendizaje, Agromutua.Web. Una aplicación web desarrollada en ASP.NET para la entidad aseguradora Agromutua S.A. y para la cual se gestionan los seguros de las pólizas, documentos adjuntos y los datos derivados de todos ellos.

El capítulo 2 servirá de especificación de requisitos de la aplicación que se desarrolla. De esta forma se mostrarán descripciones, requisitos y una visión global de lo que es Agromutua.Web.

El siguiente capítulo se analizará todos los aspectos tecnológicos empleados, la arquitectura vieja y la nueva arquitectura con la aparición de LINQ así como todas las tecnologías, lenguajes y herramientas que se emplean en la implementación del programa.

A partir del capítulo 4 se entra de lleno en LINQ, el capítulo 4 concretamente nos presenta LINQ. La definición, el proceso que se ha llevado a cabo para llegar a LINQ y sus características, la evolución en los lenguajes VB.NET y C#, los métodos de extensión, los operadores y secuencias más comunes. Acompañados de ejemplos de la base de datos de Agromutua.

Dentro del propio capítulo 4, los puntos 4.4 y 4.5 presentan LINQ to Objects que servirá de base y LINQ to SQL que es el tipo de LINQ más importante dentro de esta tesina y hacia el que más énfasis se ha puesto en explicar y hacer ejemplos. Es tan importante como para dedicarle el capítulo siguiente, el capitulo 5 para comentar el debate existente por la aparición de LINQ to SQL y su efecto en las arquitecturas de los programas. La fácil utilización de LINQ to SQL, puede llevar al programador a olvidar la existencia de la arquitectura multicapa y en concreto la existencia de una capa persistencia, en este capítulo se expondrán las diferentes opciones que se pueden adoptar.

El capítulo 6, es un capítulo de investigación y muestra las novedades que aparecerán o que son muy recientes y que todavía no se han incorporado a Agromutua. Se explicará las características básicas de PLINQ que nos proporcionará sacarle mayor rendimiento a las CPU con más de un núcleo y lanzar consultas en paralelo. Y también se presentará el ADO.NET Entity Framework que soluciona algunos de los problemas de LINQ to SQL, todo ello ya disponible en lo que será en nuevo entorno Visual Studio 2010.

El último capítulo, es el capítulo de las conclusiones y se detalla la experiencia propia con LINQ y el trabajo desarrollado dentro del proyecto.

2. ESPECIFICACIÓN DE AGROMUTUA.WEB

El capítulo 2 va enfocado a la especificación de requisitos del proyecto. Para la realización de la especificación de requisitos de software (ERS) se ha seguido el estándar IEEE Std. 830-1998 con el que se persigue definir cuáles son los requerimientos y funcionalidades que debe tener Agromutua.Web.

2.1. INTRODUCCIÓN

A continuación se describirán los requerimientos y las funcionalidades de Agromutua.Web.

2.1.1. PROPÓSITO

El propósito de esta ERS es definir cuáles son los requerimientos que debe tener Agromutua.Web para que realice su correcto funcionamiento para contratar pólizas y redactar otros tipos de documentos de seguros agrícolas.

La aplicación ha sido encargada por Agromutua S.A. para llevar a cabo todo el control de las pólizas, pagos, cálculos, etc. a la empresa de desarrollo de aplicaciones software Prodevelop S.A.

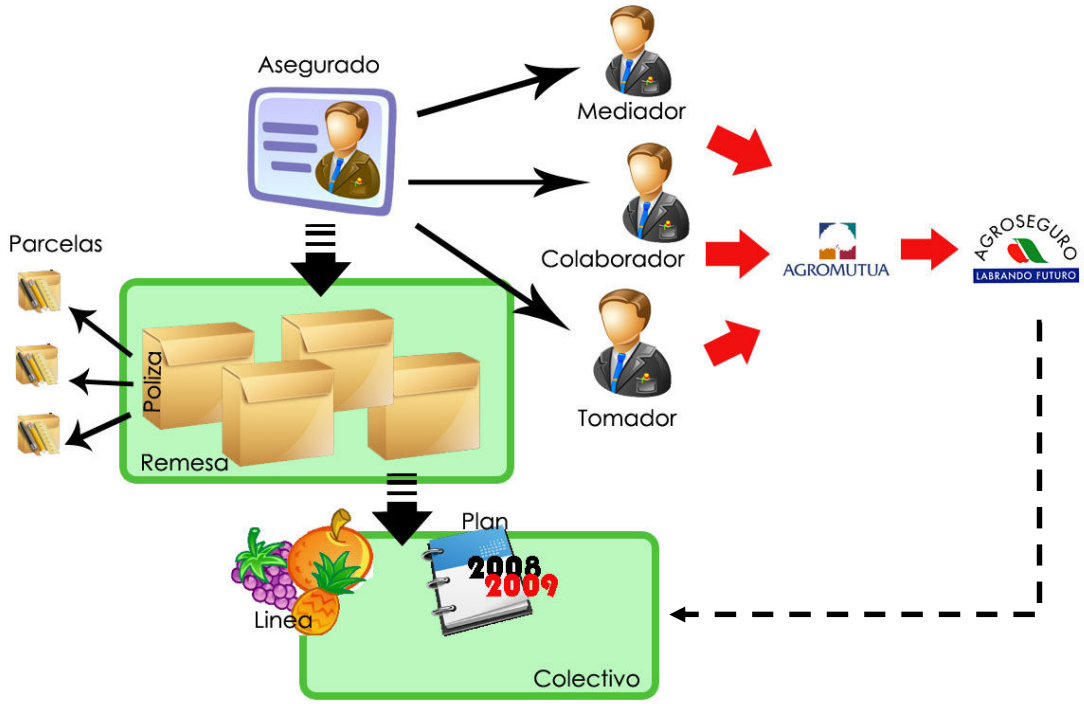
2.1.2. ÁMBITO

La aplicación se conoce por el nombre de Agromutua.Web ya que es una versión posterior de otra aplicación con la que se trabajaba antes denominada WinCamp y a diferencia de esta la nueva versión trabaja en un entorno web mientras que la antigua era de escritorio y programada en Visual Basic 6. La razón de que la nueva aplicación sea web es dada la necesitad de centralizar los datos y trabajar con una base de datos común entre usuarios y aseguradora, como se verá más adelante en la ERS existen

personas intermedias entre la entidad aseguradora y el asegurado que tienen que disponer de los datos actualizados en cualquier momento y en cualquier lugar para calcular y redactar correctamente los seguros con sus correctas variedades, cultivos, precios, subvenciones, etc.

Agromutua.Web es una aplicación web de gestión de seguros agrarios que permite a los mediadores de Agromutua contratar los seguros de los agricultores y realizar otros documentos adjuntos. Para realizar el cálculo se necesitan datos de cultivos y años anteriores que los proporciona la misma entidad que se encargar de validar una vez hechas las pólizas, Agroseguro.

En el gráfico de a continuación se muestra el funcionamiento básico de la gestión de una póliza. Los encargados de hacerlas, los mediadores, colaboradores y tomadores remiten las pólizas de los asegurados, los agricultores que aseguran sus parcelas, a Agromutua que se encarga de realizar los cálculos y validaciones y así poderse remitir a Agroseguro para que la valide definitivamente y pueda abonarla el agricultor. El ámbito que controla el programa incluye hasta el envío de las pólizas de Agroseguro.



2.1.3. DEFINICIONES, ACRÓNIMOS Y ABREVIATURAS

- Agroseguro: entidad encargada de validar las pólizas
- Agente. Entidad a través de la cual se realizan pólizas. Las cooperativas hacen siempre pólizas a través de un agente, nunca directamente con la compañía de seguros. Es normal que una entidad (Uteco, p.e.) tenga a distintos usuarios realizando pólizas con el mismo identificador de agente.
- Asegurado. Persona física o jurídica titular de la producción y que en defecto del tomador, asume los derechos y obligaciones del contrato que firme (normalmente, la póliza).
- Asegurador. Persona jurídica que asume el riesgo pactado con el asegurado.
- Colectivo (documento de seguro colectivo). Ver Pólizas asociadas a colectivo.

- Condicionado. Conjunto de valores que se utilizan para realizar el cálculo del valor de las pólizas. Agroseguro lo envía anualmente, aunque existen ciertos casos puntuales en que llegan una vez está listo y se incorpora al condicionado del año actual.
- Línea de cultivo. Agrupación lógica de cultivos con el objetivo de establecer condiciones en las pólizas que se creen. Normalmente Agroseguro entrega anualmente (por plan) un conjunto de documentos de normativa, el condicionado y formularios tipo para cada línea de cultivo. Ejemplos: línea 4, la uva; línea 43, pimiento; línea 11, tomate.
- Plan. Año natural sobre el que se trabaja (2006, 2007, etc.).
- Póliza. Documento donde se formaliza el contrato entre el tomador del seguro y el asegurador. En ella se establecen todas las condiciones que regulan el contrato. Una póliza es válida si se ha firmado por ambas partes y se ha abonado el coste del seguro. Agromutua tiene capacidad de hacer pólizas agrícolas y de ganado.
- Las pólizas a su vez se subdividen en:
    - Pólizas colectivas. Es una póliza a la que se subscriben un conjunto de asegurados y es administrada por uno o varios tomadores del seguro. Las pólizas colectivas tienen un conjunto de condiciones generales y condiciones excepcionales. Ambos están ligados a la línea de cultivo.
    - Pólizas asociadas a colectivo. Conjunto de pólizas individuales que están ligadas a una póliza colectiva y que por tanto se acogen a los términos definidos en la póliza colectiva.
    - Pólizas Individuales. Son pólizas donde se establece una relación contractual entre el tomador y un solo asegurado. Agromutua nunca realiza pólizas individuales, siempre realiza asociadas al colectivo.

- Remesa. Documento de pago de una o varias pólizas a Agroseguro. El pago siempre se realiza mediante una remesa, incluso aunque sea para pagar una única póliza. Una póliza se da como pagada si existe su remesa correspondiente.

- Tomador. Persona física o jurídica que administra una póliza. Tiene responsabilidad legal sobre la misma. La función del tomador es proteger al usuario. Suelen ser entidades con un representante legal, que también se especifica.
- Usuario. Persona física que utiliza el programa de Agromutua para poder operar con entidad.

2.1.4. REFERENCIAS

Los siguientes documentos propiedad de Prodevelop S.A. se toman como referencias del presente proyecto:

- Especificación funcional Agromutua.web. “Definición de las funcionalidades incluidas por Agromutua” basado en las funcionalidades vigentes de WINCAMP.
- Manual de usuario de funcionalidades aplicación WINCAMP.
- Oferta 2006-12020, “Desarrollo de sistema Agromutua.web”.
- Documento de glosario de términos y abreviaturas a utilizar en el proyecto.
- Impresiones de Pantallas y Listados cedidos por Agromutua a Prodevelop.

2.1.5. VISIÓN GLOBAL

La misión del proyecto es definir un sistema de información que gestione la totalidad de la problemática en la contratación de Seguros Agrarios de Agromutua. La solución pasa por un desarrollo en ASP .NET Framework 3.5 en el cual los usuarios finales podrán acceder remotamente desde sus oficinas y trabajar normalmente igual que lo están haciendo hasta ahora.

Actualmente Agromutua dispone de un sistema cliente-servidor “Wincamp” que incorpora toda la funcionalidad necesaria para el correcto funcionamiento del negocio de Agromutua desarrollado en Visual Basic 6.0, donde la aplicación reside en cada una de las oficinas que trabaja con Agromutua, existiendo programas de sincronización de datos entre las oficinas y Agromutua. El nuevo sistema “Agromutua.Web”, viene a sustituir a Wimcamp, realizando la mayoría de la funciones de Wincamp como quedará especificado en el punto de descripción general del producto.

2.2. DESCRIPCIÓN GENERAL

A continuación se dará a conocer una visión de las principales funcionalidades de Agromutua.Web

2.2.1. PERSPECTIVA DEL PRODUCTO

Agromutua.Web es una herramienta para realizar las gestiones necesarias para contratar pólizas agrícolas, se trata de un programa de gestión único para Agromutua. Existen otras entidades aseguradoras que disponen de programas que realizan la misma funcionalidad pero implementados por otras empresas y con diferentes tecnologías.

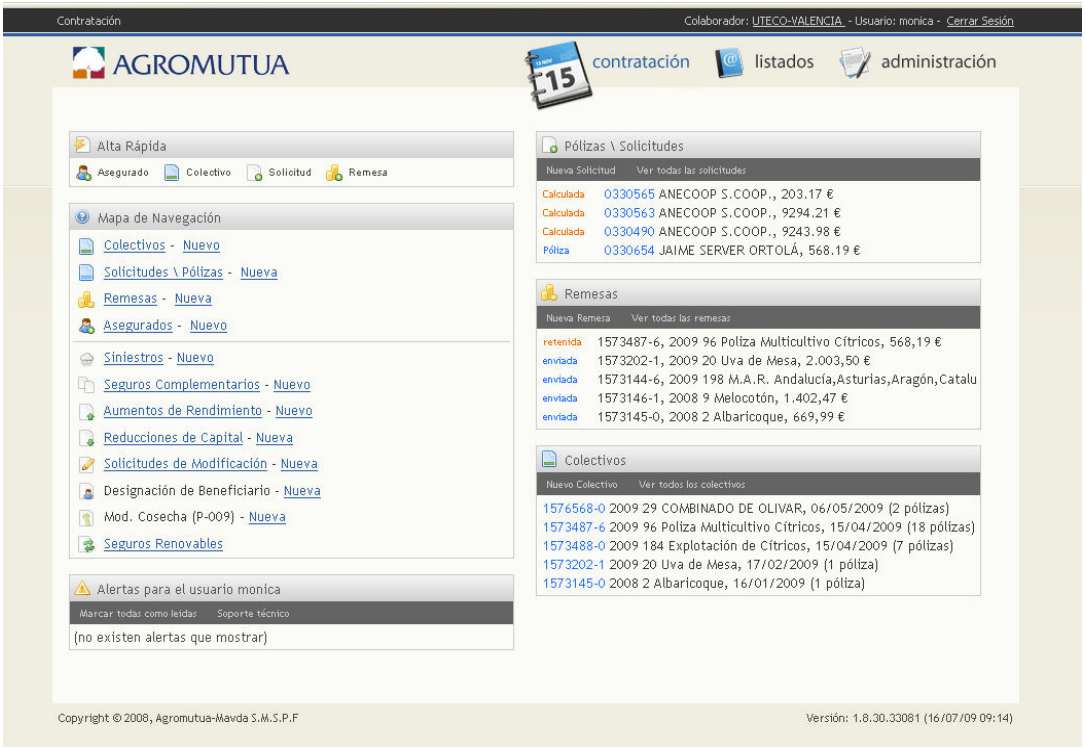
2.2.2. FUNCIONES DEL PRODUCTO

A continuación se van a detallar a modo de esquema los aspectos de la gestión más relevantes. En el apartado 2.3.2 se verán sus funciones y los datos necesarios.

La distribución de este punto de la especificación será de acuerdo al menú superior que nos lleva a las tres grandes pantallas de la aplicación: Contratación, Listados, Administración. Y se proporciona un prototipo de pantalla para ayudarse en la explicación y observar la distribución de los contenidos.

Contratación

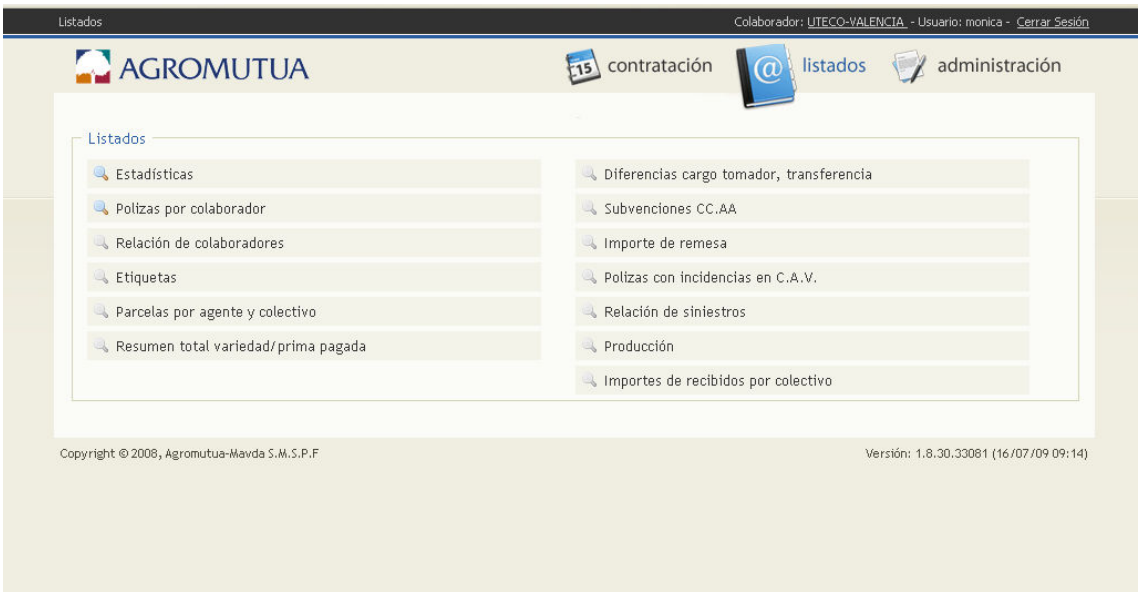
La contratación es el inicio en Agromutua.Web para los usuarios de la aplicación que contratan los seguros. Se caracteriza por servir de acceso directo a las principales funcionalidades del programa. En las dos columnas que se observan se distribuyen los accesos directos, en la parte izquierda los accesos directos a altas nuevas de documentos y los listados de estos, y en la parte derecha el acceso directo a los últimos de los documentos realizados por el propio usuario.



- Colectivo
    - Listado
    - Añadir/Actualizar/Borrar
- Solicitudes/Pólizas
    - Cálculo
    - Añadir/Actualizar/Borrar
    - Parcelas
        - Añadir/Actualizar/Borrar parcelas
    - Realizar pago
    - Añadir/Actualizar/Borrar documentos adjuntos
        - Siniestro
        - Seguro Complementario
        - Reducción de capital
        - Solicitud de modificación
        - Designación de beneficiario
        - Modificación de cosecha
- Remesas
    - Listado
    - Añadir/Actualizar/Borrar
- Asegurado
    - Listado
    - Añadir/Actualizar/Borrar

Listados

Los listados son estadísticas y agrupaciones de datos que proporcionan datos estadísticos de contratación de pólizas, cultivos, variedades, subvenciones, etc. Estos listados son conocidos como "reports" y son documentos impresos en PDF. Cada listado despliega una ventana emergente que configura los datos a mostrar.



- Estadísticas
- Pólizas por colaborador
- Relación de colaboradores
- Etiquetas
- Parcelas por agente y colectivo
- Resumen total variedad/prima pagada
- Diferencias cargo tomador, transferencia
- Subvenciones CC.AA
- Importe de remesa
- Pólizas con incidencias en C.A.V.
- Relación de siniestros
- Producción
- Importes de recibidos por colectivo

Administración

La administración de la aplicación será llevada a cabo por las personas encargadas del mantenimiento en las oficinas de la entidad aseguradora de Agromutua SA. Estos usuarios con permisos especiales tendrán acceso a los datos de configuración y los datos necesarios para realizar cálculos de los seguros, así como otras tareas propias de la administración como la exportación/importación de datos o el acceso a registro de usuarios de Agromutua (Mediadores, Tomadores y Colaboradores).

El administrador de la aplicación es el encargado del envío de las pólizas y los documentos adjuntos a Agroseguro, que lo hará desde una pantalla situada en este menú.

- Actores del seguro
    - Tomador
        - Listado
        - Añadir/Actualizar/Borrar
    - Mediador
        - Listado
        - Añadir/Actualizar/Borrar
    - Colaborador
        - Listado
        - Añadir/Actualizar/Borrar
    - Asegurado
        - Listado
        - Añadir/Actualizar/Borrar
    - Usuario
        - Listado
        - Añadir/Actualizar/Borrar
- Maestros
    - Líneas de seguro
        - Listado
        - Añadir/Actualizar/Borrar
    - Gestión de contadores
        - Listado
        - Añadir/Actualizar/Borrar
    - Cambio de contraseña
    - Envíos a Agroseguro
        - Registro de envío
        - Envío manual de póliza y remesa
        - Envío manual de siniestro
        - Configurador del planificador
    - Exportación de producción
        - Generar fichero de producción
        - Producciones aseguradas
    - Importación de renovable
    - Comisiones

2.2.3. CARACTERÍSTICAS DEL USUARIO

Tendrán acceso a Agromutua.Web solo usuarios autorizados con su login y contraseña, proporcionados por Agromutua S.A.. En la aplicación se identifican dos tipos de usuarios: administradores y agentes intermedios.

Los agentes intermedios son usuarios con permisos de Agromutua S.A. para realizar pólizas y cálculos en la aplicación, teniendo acceso a todas las funciones descritas en el menú principal del punto 2.2.2.

Los administradores dispondrán de control total de la aplicación (contratación y administración del punto 2.2.2) y serán los encargados de mantener los datos y la consistencia de estos.

La utilización de Agromutua.Web no requiere ningún tipo de conocimiento avanzado de informática para los usuarios normales, siendo único requisito estar familiarizado con la navegación por internet y la utilización del navegador web dando por supuesto que cualquier usuario conoce el propio negocio.

2.2.4. RESTRICCIONES GENERALES

La aplicación trabaja sobre dos servidores interconectados, uno sirve de repositorio de aplicaciones y otro de repositorio de datos.

Es importante disponer de un potente servidor de datos debido a que existen pantallas que necesitan de una carga rápida de datos puntuales existentes en tablas con un número enorme de registros en el menor tiempo posible, sobrecargando la red en ocasiones un número elevado de usuarios. Además de un potente servidor se acompaña de tecnologías que hacen ganar tiempo de carga como Javascript y AJAX trabajando los datos en pequeñas peticiones asíncronas desde el cliente.

La aplicación se caracterizará por un gran nivel de seguridad de los datos, y se tendrá muy en cuenta la visualización correcta por parte de los usuarios identificados.

Por último se hace necesario un usuario que tenga acceso a la gestión de todos los contenidos, este superusuario será el administrador de Agromutua.Web y trabajara desde la sede de la mutua resolviendo los problemas surgidos durante la contratación.

En lo que respecta al navegador web, se dará soporte para trabajar con Internet Explorer 7 y Mozilla Firefox 2. El resto de navegadores quedan, en principio, excluidos de la propuesta, pese a que se entiende que la mayoría de ellos podrán ejecutar la aplicación en la medida en que se ciñan a los estándares establecidos en Internet.

Se asume la configuración por defecto de cada uno de ellos como plataforma estándar de desarrollo cliente. Esto implica, niveles de seguridad medios para internet, JavaScript habilitado, cookies habilitadas y restricciones por defecto en cuanto al uso de ventanas emergentes y control de descargas. Se intentará que la aplicación sea, en la medida de lo posible, compatible con XHTML 1.0 Transitional y CSS de nivel 2, según los estándares del W3C (http://www.w3.org/TR/xhtml1/, http://www.w3.org/TR/REC-CSS2/).

En vistas a la escalabilidad de la aplicación, la reducción de tráfico de datos y la mejora de la experiencia del usuario, el cliente deberá tener Javascript habilitado y este seguir los estándares del ECMA referentes a ECMAScript. Los estándares propuestos se intentarán seguir siempre y cuando no resulten en una pérdida de funcionalidad de la aplicación.

Se recomienda una resolución mínima de 1024x768 para las pantallas cliente, entendiendo que es el mínimo para que muchos de los formularios definidos en el proceso de análisis quepan en la pantalla para facilitar el trabajo. En la medida de lo posible se intentará que la aplicación trabaje con resoluciones superiores a 1280x1024. Se establece esta última resolución como límite para las pruebas funcionales que se realizarán. Dado que casi la totalidad de tarjetas gráficas del mercado lo permite, se diseñarán los diferentes formularios con una profundidad de color de 32bits.

2.2.5. SUPUESTOS Y DEPENDENCIAS

Existen las dependencias tecnológicas que supone utilizar ASP.NET, requisito que el cliente pidió durante la elaboración de la aplicación, por eso al tratarse de una tecnología Windows va ligada con un servidor Windows Server 2003 y una base de datos SQL Server 2005.

Por otra parte, Agromutua.Web se trata de una migración de tecnología de escritorio a web con lo que esto supone un cambio a la hora de trabajar. La web tiene sus beneficios pero también inconvenientes, como la cache, los navegadores, las sesiones y otras características que supone que tanto el usuario como el administrador de Agromutua.Web deben tener en cuenta y adaptarse.

Al ser web los datos están centralizados y se recuperarán explícitamente para cada usuario identificado.

2.2.5.1 SISTEMAS DE LOS QUE DEPENDE AGROMUTUA.WEB

El proyecto Agromutua.Web depende de los siguientes sistemas:

- Débilmente del sistema informático de Agroseguro en la recepción de varios documentos.
- Del sistema WinCamp en lo que se refiere a la integración de datos:
- WinCamp transmitirá información a Agromutua.Web

2.2.5.2 SISTEMAS QUE DEPENDEN DE AGROMUTUA.WEB

Los siguientes sistemas o proyectos dependen de Wincamp:

- Agromutua.Web deberá generar información en Access para los usuarios finales que deseen integrar sus sistemas informáticos propios con la información centralizada en Agromutua.

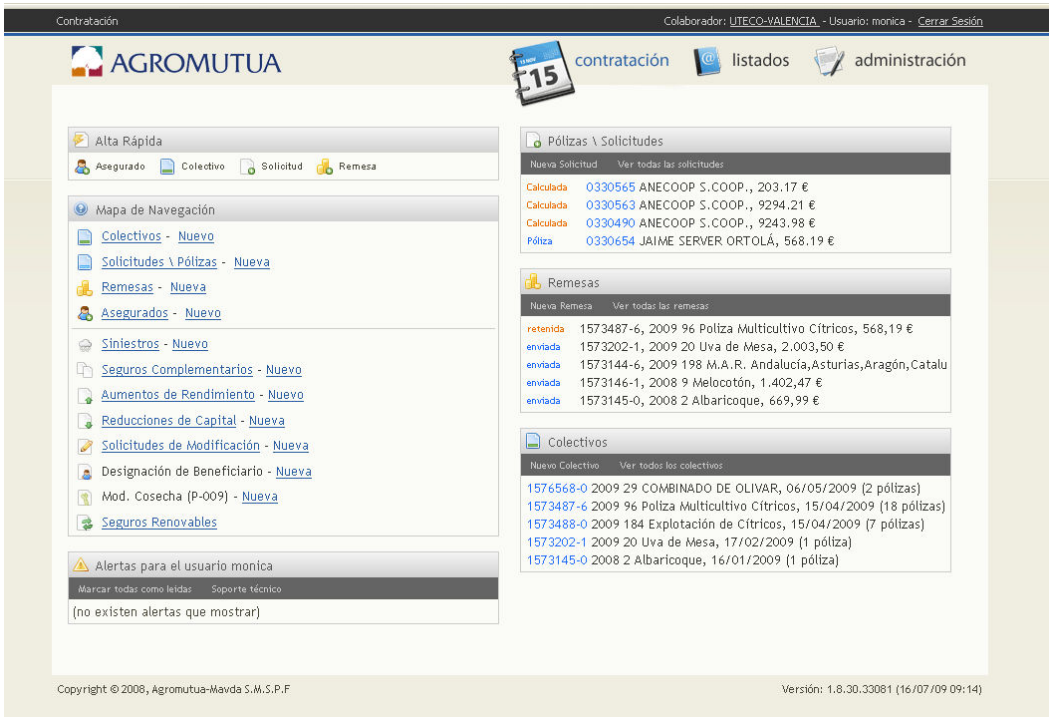
2.3. REQUISITOS ESPECÍFICOS

A continuación se describirán detalladamente los requisitos de la aplicación Agromutua.Web, basándonos en el estándar 830-1998, se ha elegido hacer una descripción de la organización de este punto por modo de funcionamiento, en el que se describe el modo de ejecución u operación del sistema.

2.3.1. REQUISITOS DE INTERFACES EXTERNOS

2.3.1.1. INTERFACES DE USUARIO

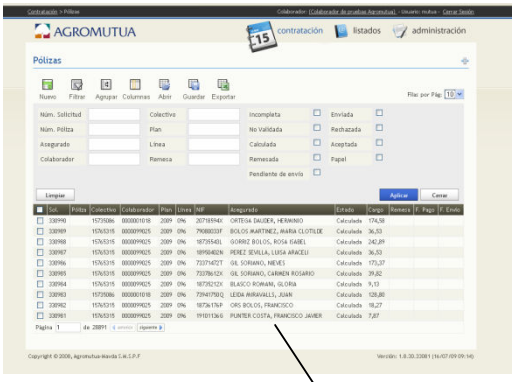
La interfaz de la aplicación estará basada en el modelo búsqueda-listado-detalle e irá acompañada de dos componentes propios de Prodevelop SA. De tal forma se que se dispondrá de un menú principal de acceso a los contenidos generales:



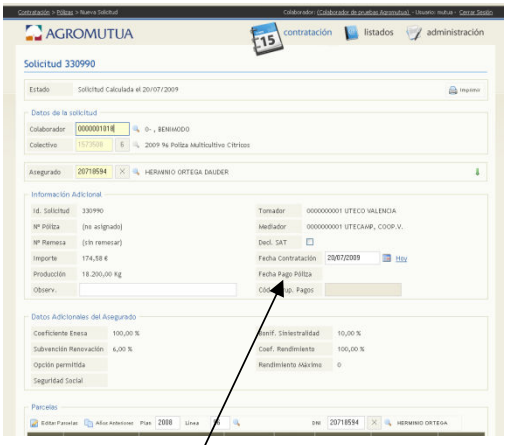
En la parte derecha de la pantalla principal se observan los enlaces directos a las últimas acciones que ha realizado el usuario y en la parte izquierda se muestra el menú de navegación dentro de la aplicación, accediendo se muestra el listado que permite el filtrado y su posterior acceso al detalle del documento.

También se tiene la opción directa de crear un nuevo documento.

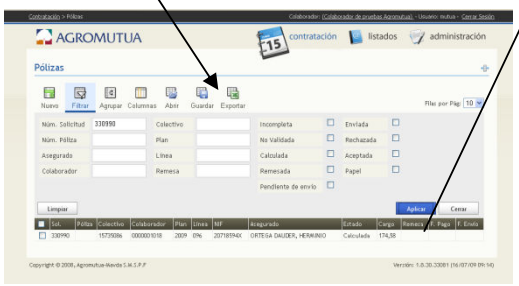
Modelo de Búsqueda-Listado-Detalle



1. LISTADO



3. DETALLE

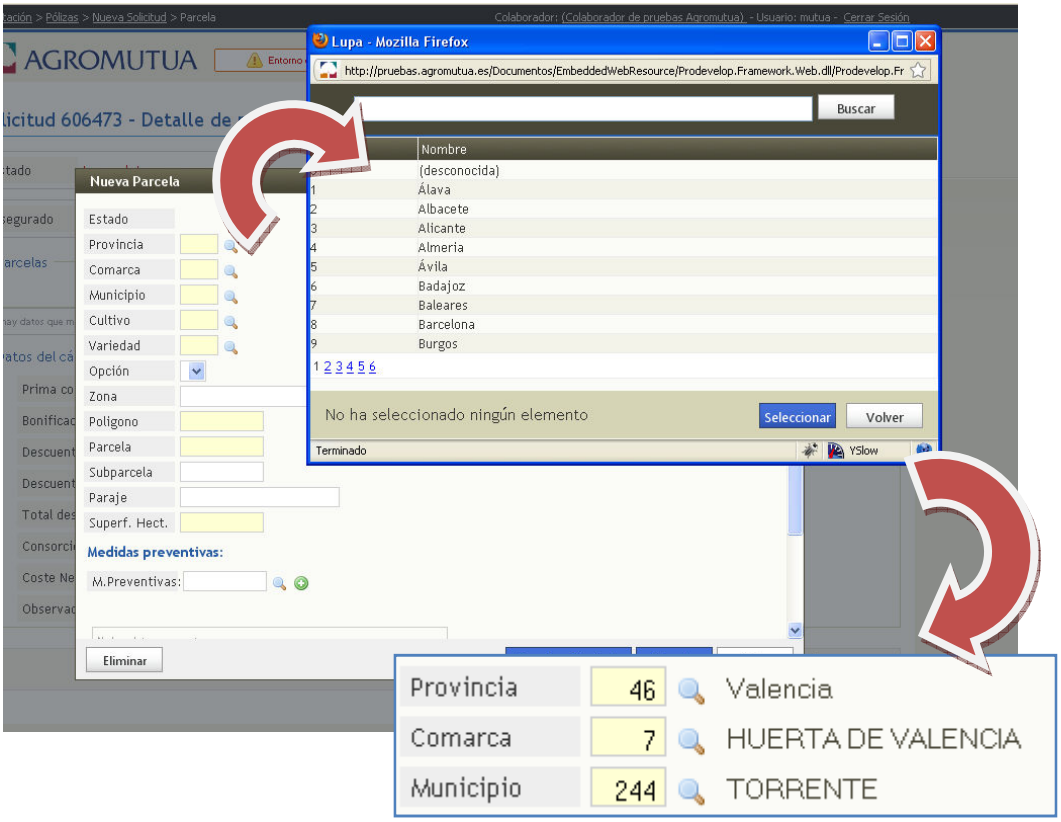


2. BÚSQUEDA

Otros aspectos importantes de la interfaz son dos de los componentes desarrollados íntegramente para Agromutua.Web y que el usuario deberá familiarizarse con ellos. Estos componentes son las “Lupas” y los “Trabajar Con”.

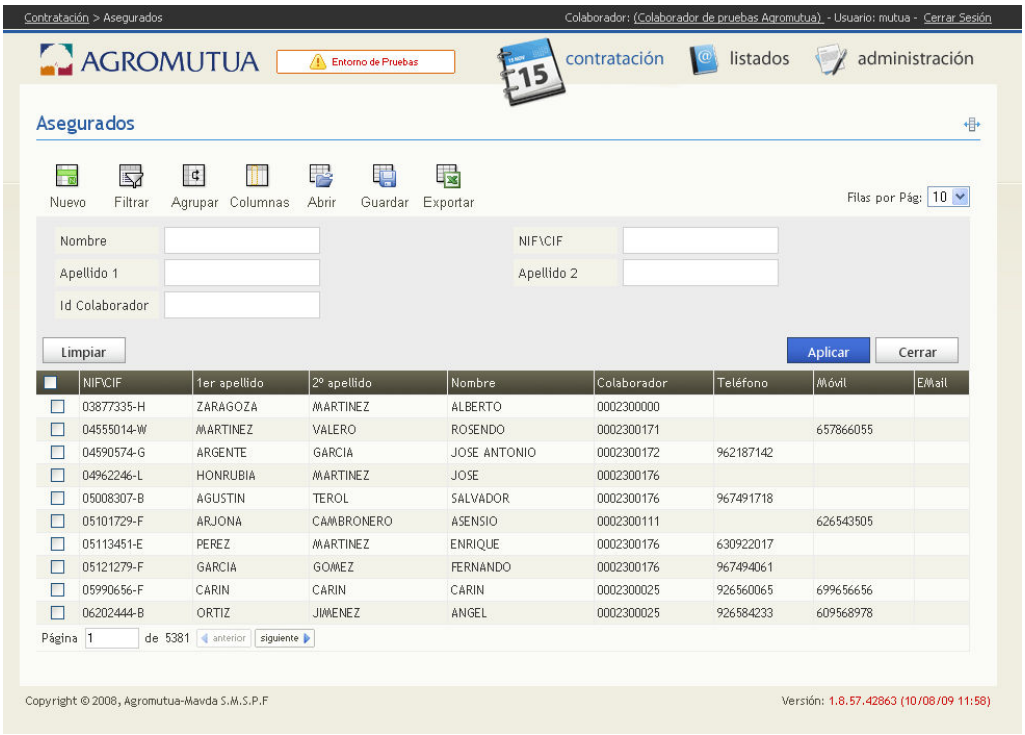
Lupas

Las Lupas son unos iconos situados al lado de cajas de textos que despliegan una ventana emergente para obtener la información que se busque en esta ventana. Junto con el valor que se incluirá en la caja de texto viene la descripción que se situara en la otra parte de la Lupa. Uno de los aspectos más interesantes para este usuario es que se pueden vincular varias Lupas, muy útil en determinados casos como en este ejemplo donde están vinculados Provincia, Comarca y Municipio.



Trabajar Con

Los Trabajar Con son otro componente presente en las búsquedas de los listados, que en su conjunto permiten filtrar, ordenar y exportar los datos a partir de una vista previamente preparada en la base de datos.



2.3.1.2. INTERFACES HARDWARE

Esta arquitectura estará formada por dos servidores: una donde se ejecutarán conjuntamente el Servidor Web y el Servidor de Aplicaciones, conforme a las recomendaciones de Microsoft y otra que ejecutará el Servidor de Base de Datos.

Nombre	Requisitos	Procesador	Memoria	Conexión	Velocidad	Almacenamiento
Servidor de Base de datos	Mínimo	Dos procesadores de núcleo doble Intel Xeon 5120 a 1.86 GHz con FSB a 1066 Mhz.	4 GB con FSB a 533 MHz, ampliable a 32 GB.	Dos adaptadores de puerto Intel PRO 1GB.	4Mb	4 discos SAS de 146 GB a 10.000 rpm en configuración RAID-5, expandible a 6 discos en total
Servidor de aplicaciones	Mínimo	Intel Xeon 5130 a 2.0 GHz con FSB a 1333 Mhz.	4 GB con FSB a 667 MHz, ampliable a 32 GB.	Dos adaptadores de puerto Intel PRO 1 GB	4Mb	3 discos SAS de 73 GB a 15.000 rpm en configuración RAID-5, expandible hasta 6 discos en total

Esta segunda aproximación implementa una arquitectura óptima en cuanto a disponibilidad orientada al servicio, alto rendimiento y suficientemente escalable en cuanto a capacidad de proceso, mediante el uso de un clúster de servidores (Server Clúster).

Esta arquitectura estará formada por cinco servidores: dos para ejecutar el Servidor Web y el Servidor de Aplicaciones en un clúster de dos nodos, dos para ejecutar el Servidor de Aplicaciones, también en un clúster de dos nodos y uno para mantener el disco de Quorum para los dos clúster de servidor.

Las dos máquinas de ambas configuraciones deben ser exactamente iguales para evitar cualquier problema de compatibilidad, máxime implementando esta arquitectura para obtener alta disponibilidad.

Nombre	Requisitos	Procesador	Memoria	Conexión	Velocidad	Almacenamiento
Servidor de Base de datos	Optima	Dos procesadores de núcleo cuádruple Intel Xeon E5320 a 1.86 GHz con FSB a 1066 MHz	4 GB a 533 MHz ampliable a 32 GB.	Cuatro adaptadores de puerto Intel PRO 1 GB.	12Mb	4 discos SAS de 146 GB a 10.000 rpm en configuración RAID-5 expandible a 6 discos en total
Servidor de aplicaciones	Optima	Dos procesadores de núcleo cuádruple Intel Xeon E5335 a 2.0 GHz con FSB a 1333 Mhz.	4 GB con FSB a 667 MHz, ampliable a 32 GB en total	Cuatro adaptadores de puerto Intel PRO 1 GB.	12Mb	3 discos SAS de 73 GB a 15.000 rpm en configuración RAI-5, expandible hasta 6 discos en total

2.3.1.3. INTERFACES SOFTWARE

S.O.	Windows 2003 Server R2 x32 Enterprise
Servidor Web	IIS 6.0
Framework de trabajo	Microsoft .Net Framework 2.0 o superior

2.3.1.4. INTERFACES DE COMUNICACIONES

Agromutua.Web es una aplicación web de gestión que se ejecuta sobre el protocolo HTTP.

HTTP define la sintaxis y la semántica que utilizan los elementos software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. A la información transmitida se la llama recurso y se la identifica mediante un URL. Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores.

2.3.2. REQUISITOS FUNCIONALES

A continuación se describirá los requisitos funcionales más importantes de Agromutua.Web, se dividirán en dos modos: usuario y administrador.

El usuario será la persona encargada de utilizar la aplicación en las distintas oficinas de contratación y su trabajo será el de contratar seguros y documentos adjuntos. Los administradores se encargan de mantener los datos correctos de la aplicación para realizar las pólizas y de programar envíos y recepciones de datos con Agroseguro, así como ayudar y resolver problemas con la contratación de seguros que realizan los usuarios.

Para acompañar las descripciones de las funcionalidades del producto se incorpora a las descripciones de los requisitos una captura del prototipo de la aplicación que permite la visualización de la entrada de los datos.

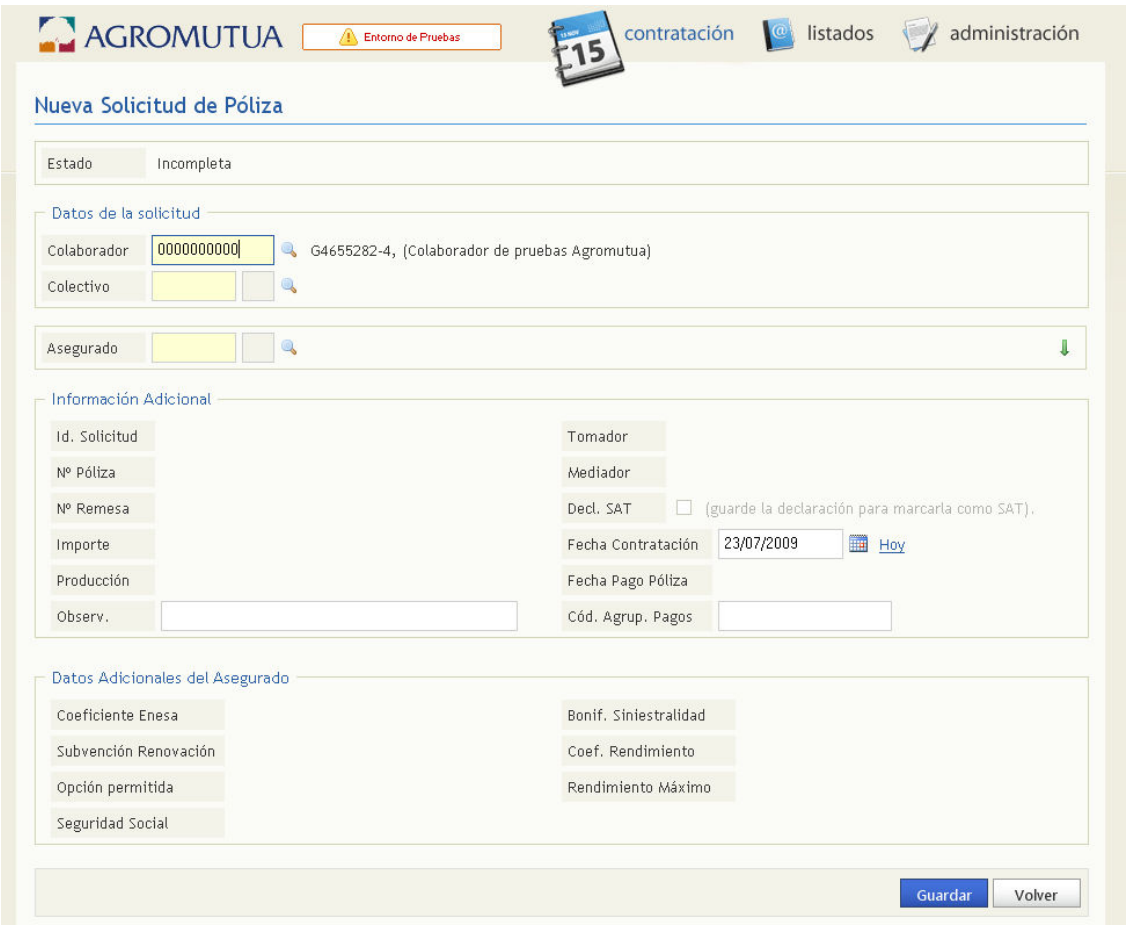
En principio el Estándar IEEE-830 no incluye capturas de pantalla ni prototipos de la aplicación, la decisión de la incorporación de esta característica extra dentro de la especificación de Agromutua.Web se ha decido proporcionar para facilitar la comprensión y el uso de la aplicación, es importante destacar que en estas capturas se hace uso de los componentes descritos en el punto 2.3.1.1.

MODO USUARIO

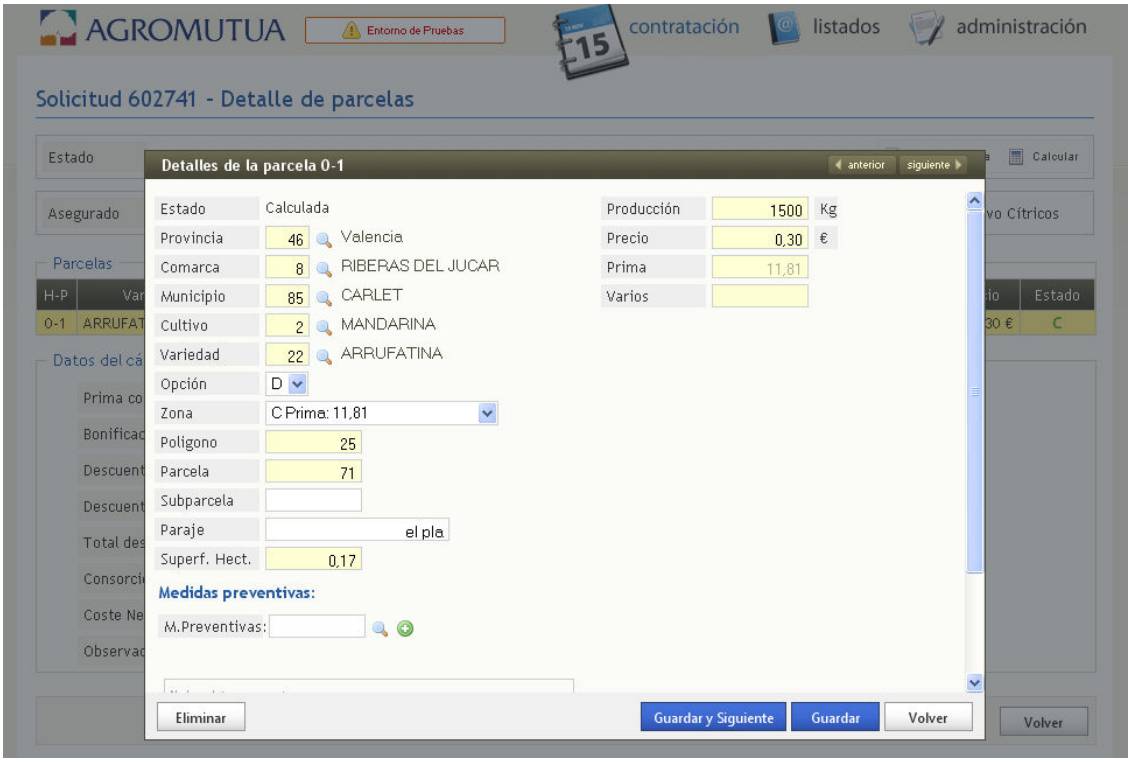
En el modo usuario se proporcionan las funcionalidades que todo usuario tendrá su disposición una vez autentificado. Para poder disponer de un usuario con contraseña previamente ha debido ser dado de alta por los administradores de Agromutua.

Veamos las funcionalidades más importantes de este modo junto con las pantallas necesarias.

Añadir póliza	
Entrada	Colaborador, colectivo, asegurado, fecha contratación, código de agrupación de pagos, observaciones.
Proceso	Comprobar: • que ningún campo sea vacio. • que la fecha tenga formato correcto. • que colectivo, asegurado y colaborador están relacionados. Obtener modulación y subvenciones adicionales de asegurado. Recuperar datos de cultivos y variedades. Inicializar el estado de la póliza a “Incompleta” Crear poliza
Salidas	Confirmación que la operación se ha realizado con éxito. Mostrar el alta de parcelas.

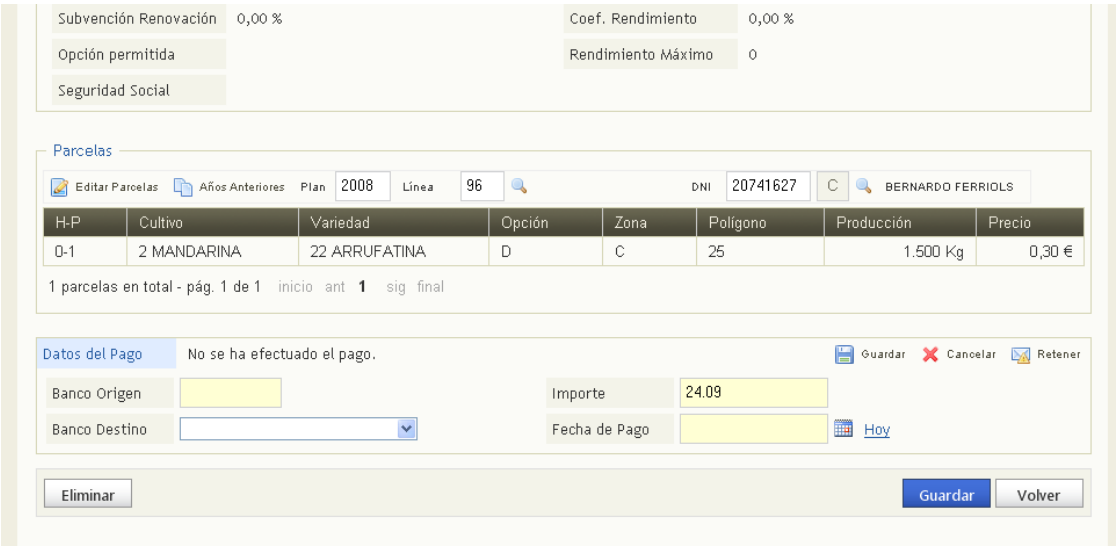


Añadir parcelas a la póliza	
Entrada	Provincia, Comarca, Municipio, Cultivo, Variedad, Opción, Zona, Polígono, Parcela, Subparcela, Paraje, Superficie, Producción, Precio, Prima, Varios
Proceso	Los campos Provincia, Comarca, Municipio, Cultivo, Variedad, Opción, Zona, Polígono, Parcela, Superficie, Producción, Precio y Prima son obligatorios. Comprobar: • el municipio sea de la comarca y la provincia seleccionada. • que el municipio pertenezca a la comarca. • que la variedad pertenezca al cultivo. • que la opción sea válida. • que el precio este en su intervalo máximo y mínimo. Recuperar de la póliza la línea y el plan. Obtener posibles variedades y cultivos. Añadir la parcela a la póliza seleccionada por su campo de identidad Crear parcela
Salidas	Confirmación que la operación se ha realizado con éxito.

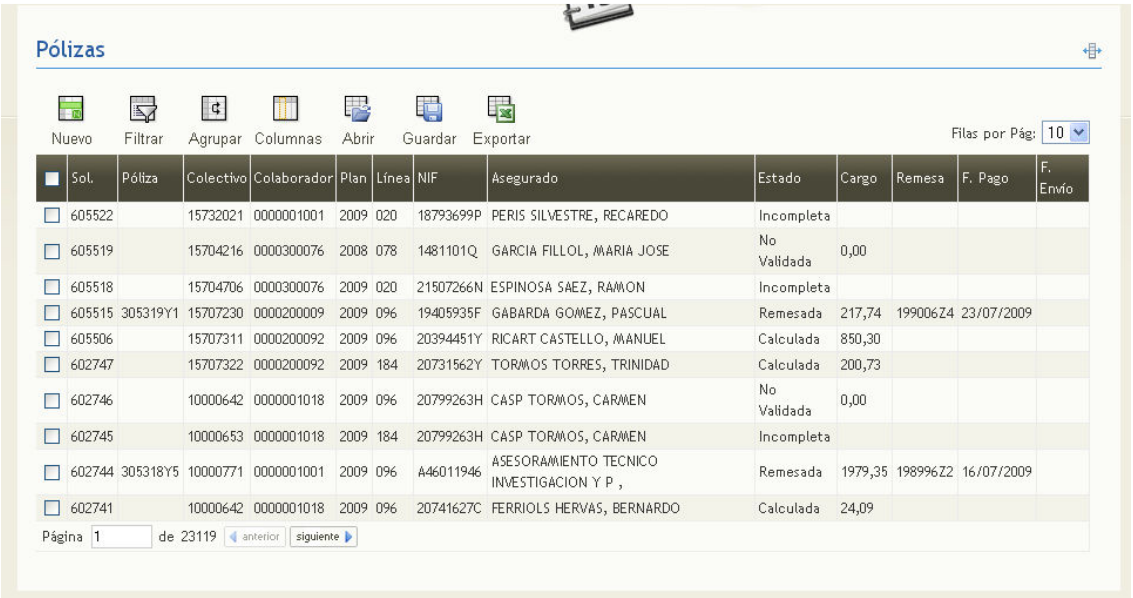


Cálculo de la póliza	
Entrada	Lista de parcelas
Proceso	Comprobar: • que las parcelas estén todas en estado Aceptado • que la póliza este en el periodo de contratación • si existen impuestos, descuentos, subvenciones para la póliza y aplicarlos. Obtener del objeto póliza los datos del asegurado: modulación, subvenciones adicionales Obtener de la póliza la línea y el plan. Obtener de cada parcela de la póliza la variedad y el cultivo. Realizar el cálculo primero del la prima comercial, de los impuestos y las subvenciones. Calcular el Coste del seguro: Coste Total = Coste Neto – (Subvención Enesa + Subvención CCAA) Cambiar el estado de la póliza a “Calculada” Guardar los datos del cálculo individualmente en cada parcela y los totales en la póliza.
Salidas	Confirmación que la operación se ha realizado con éxito. Mostrar los datos totales del cálculo de la póliza. Mostrar el nuevo estado.

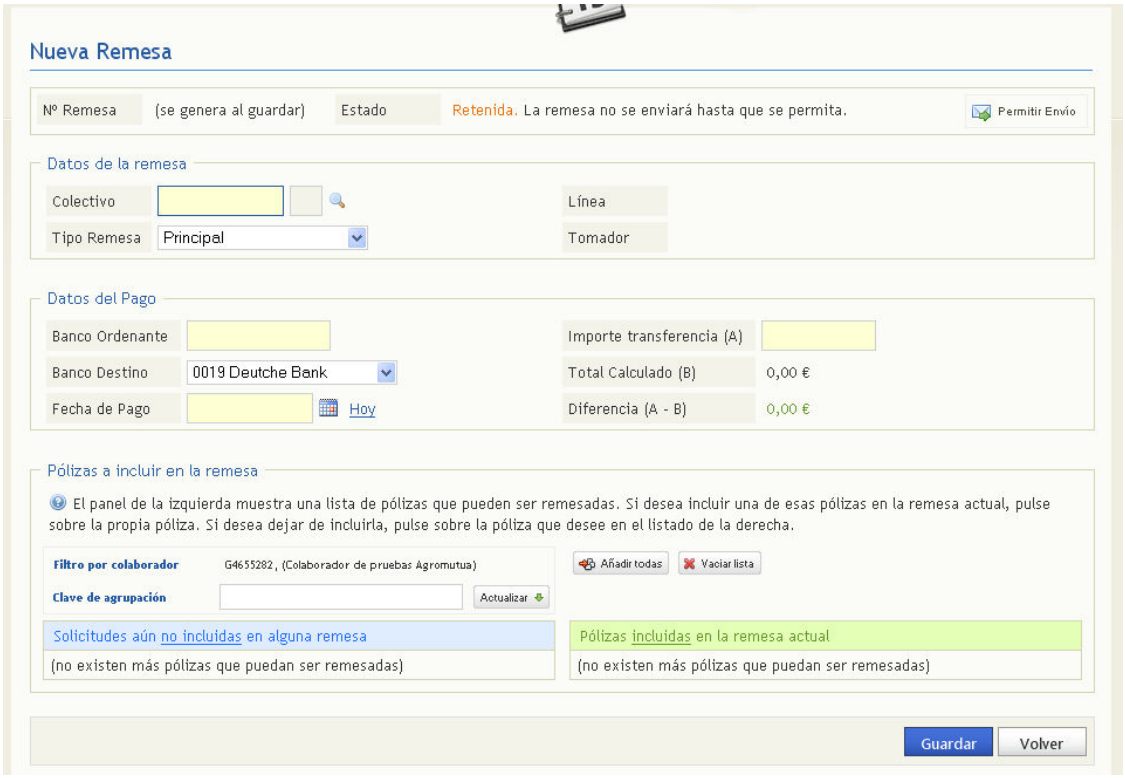
Realizar pago de la póliza	
Entrada	Banco origen, Banco destino, Importe, Fecha pago.
Proceso	Todos los campos son obligatorios. Actualizar la póliza con el importe pagado. Cambiar el estado de la póliza a “Enviable”. Registrar el pago.
Salidas	Confirmación que la operación se ha realizado con éxito. Mostrar el nuevo estado.



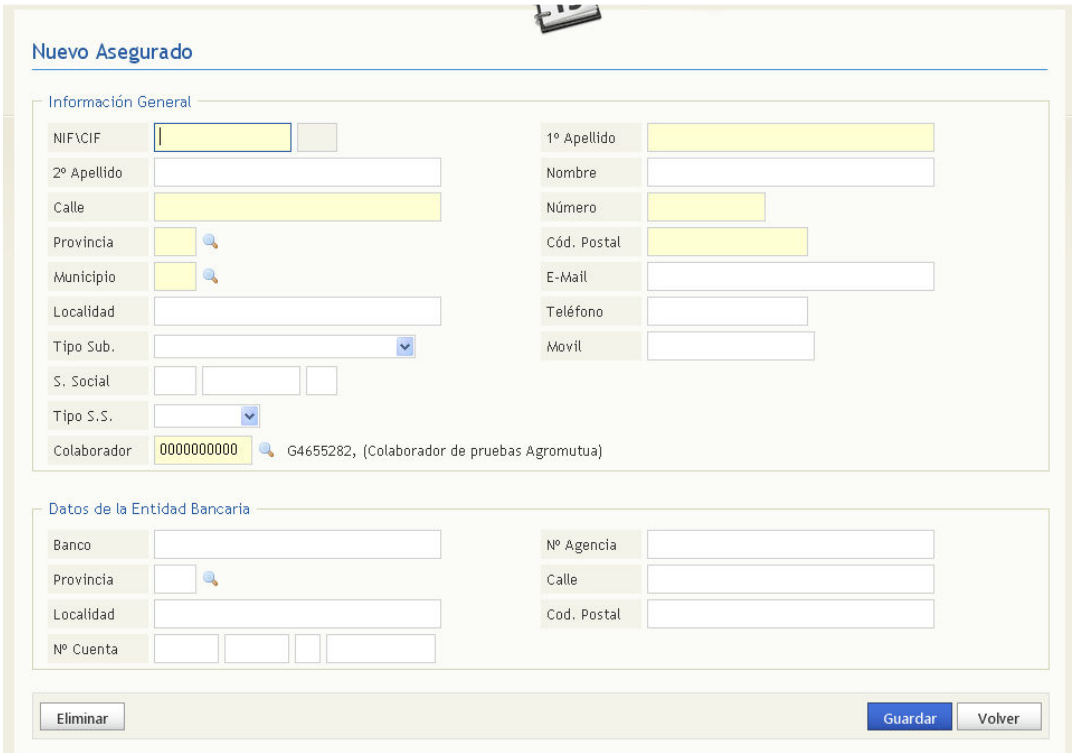
Listados	
Entrada	Tipo de listado
Proceso	Determinar el listado mediante un filtro (campos opcionales) y una vista.
Salidas	Listado



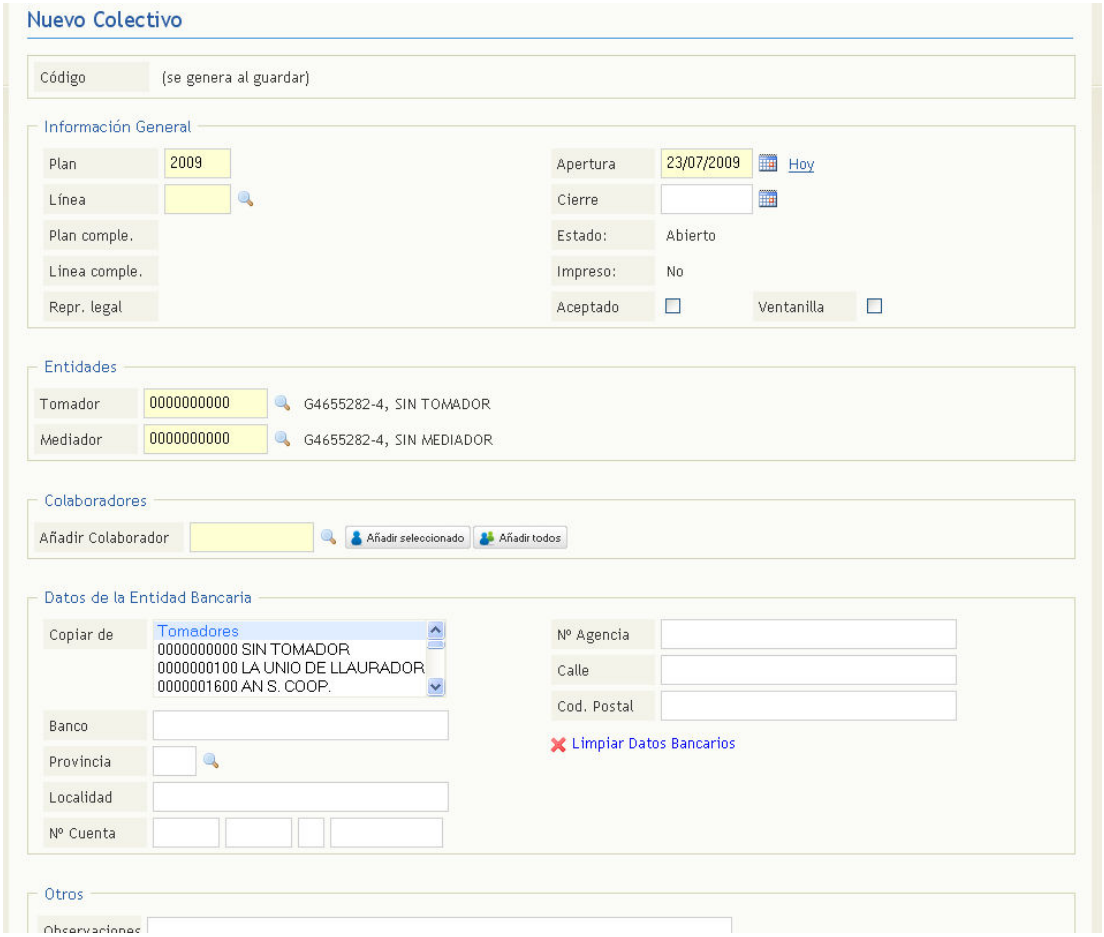
Crear remesa	
Entrada	Colectivo, Colaborador, Banco Ordenante, Banco Destino, Fecha pago, Importe de transferencia, listado de Pólizas a remesar, clave de pago.
Proceso	Comprobar que se ha realizado correctamente el pago Obtener la diferencia, si existe. Comprobar que todas las pólizas pertenecen al mismo colaborador y colectivo. Agrupar por clave de pago. Cambiar estado de póliza a “Remesada”. Crear el nuevo registro. Actualizar la póliza y hacer referencia en esta a la remesa que pertenece.
Salidas	Confirmación que la operación se ha realizado con éxito. Mostrar el nuevo estado. Mostrar la diferencia, si existe.



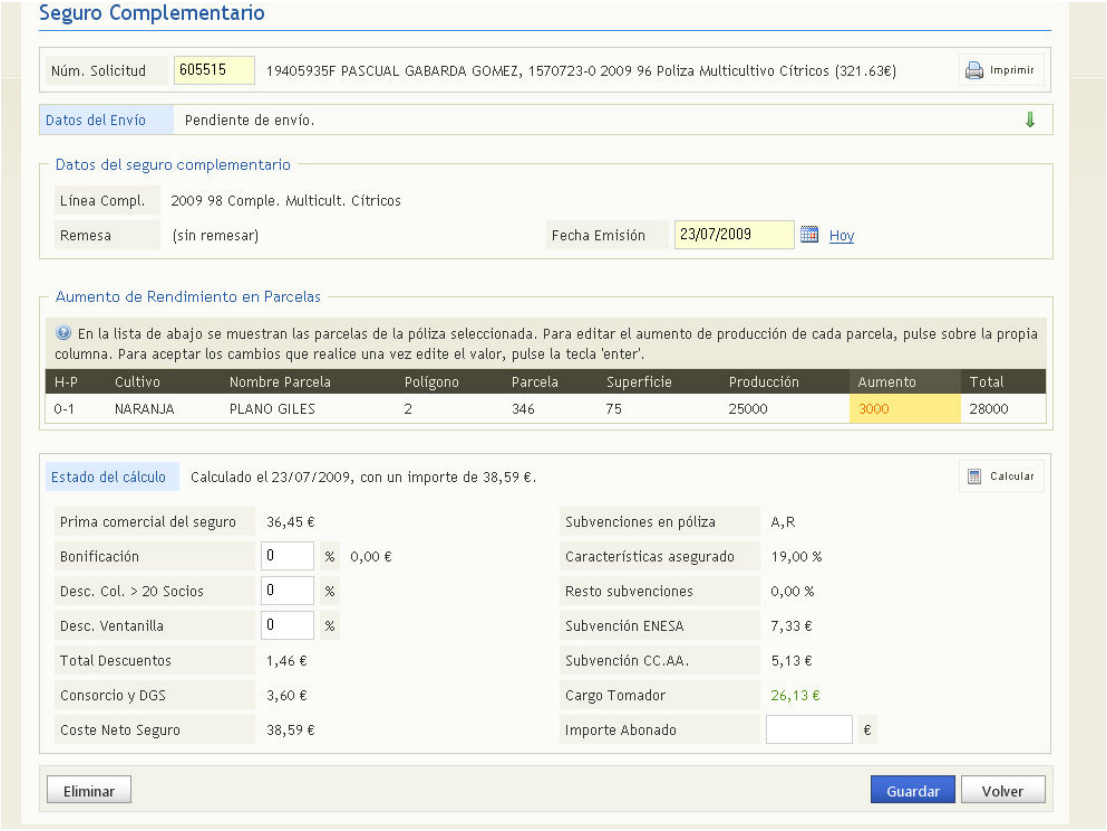
Añadir asegurado	
Entrada	NIF, 1º Apellido, 2º Apellido, Nombre, Calle, Numero, Provincia, Municipio, Localidad, Teléfono, E-mail, Móvil, Seguridad Social, Tipo SS, Colaborador. Opcionalmente: Banco, Número de agencia, Provincia del Banco, Localidad del Banco, Calle y CP del Banco, Número de cuenta.
Proceso	Comprobar: • si el municipio pertenece a la provincia. • campos obligatorios Obtener identificador de Colaborador del que pertenece. Enviar un mail (si tiene) al asegurado. Crear asegurado
Salidas	Confirmación que la operación se ha realizado con éxito.



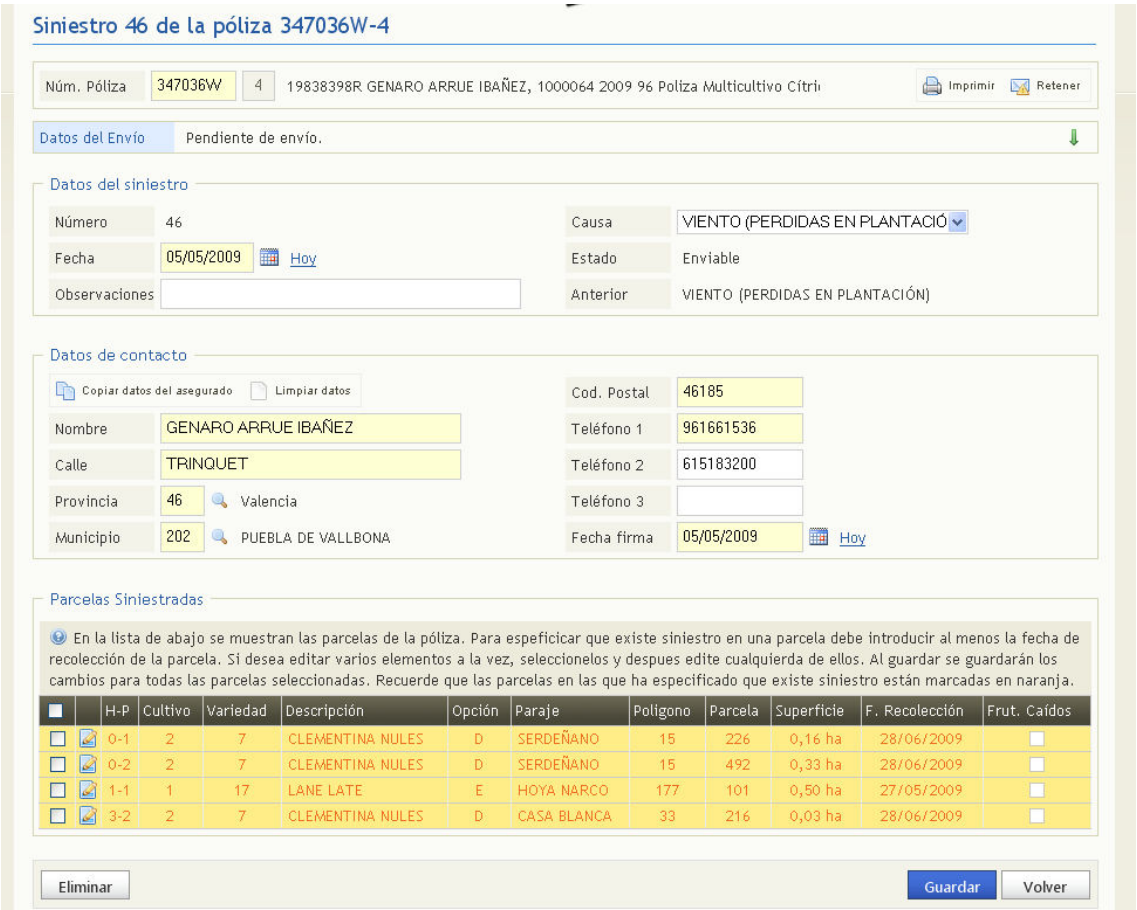
Añadir colectivo	
Entrada	Plan, Línea, Fecha apertura, Fecha de cierre, Aceptado, Ventanilla, Tomador, Mediador, Lista de colaboradores, Datos bancarios (banco, provincia, número de cuenta…)
Proceso	Validar que la Línea está dada de alta y es correcta. Comprobar Fecha Apertura < Fecha Cierre Comprobar que Lista de Colaboradores > 0 Obtener identificadores de Tomador y Mediador. Obtener identificadores de cada Colaborador. Crear colectivo
Salidas	Confirmación que la operación se ha realizado con éxito.



Crear seguro complementario de una póliza	
Entrada	Fecha emisión, Listado de parcelas con aumentos de producción
Proceso	Comprobar: • que la línea tiene complementario. • que las parcelas tienen aumento. Producción < Producción + Aumento • que existen tarifas, impuestos y subvenciones para la línea complementaria. Realizar el cálculo de las parcelas con aumento (Aumento > 0). Crear seguro complementario
Salida	Confirmación que la operación se ha realizado con éxito.



Crear siniestro de una póliza	
Entrada	Numero de póliza, Fecha de siniestro, Causa, Observaciones, Datos de contacto: Nombre, Calle, Provincia, Municipio, CP, Teléfono, Fecha firma. Lista de parcelas siniestradas con la fecha de recolección.
Proceso	Comprobar: • que existen parcelas con siniestro. • que la fecha de recolección es posterior al documento de siniestro. • los datos de contacto son obligatorios. • Se ha seleccionado una causa y una fecha • Fecha Recolección < Fecha Firma Crear siniestro y parcelas siniestradas
Salida	Confirmación que la operación se ha realizado con éxito.

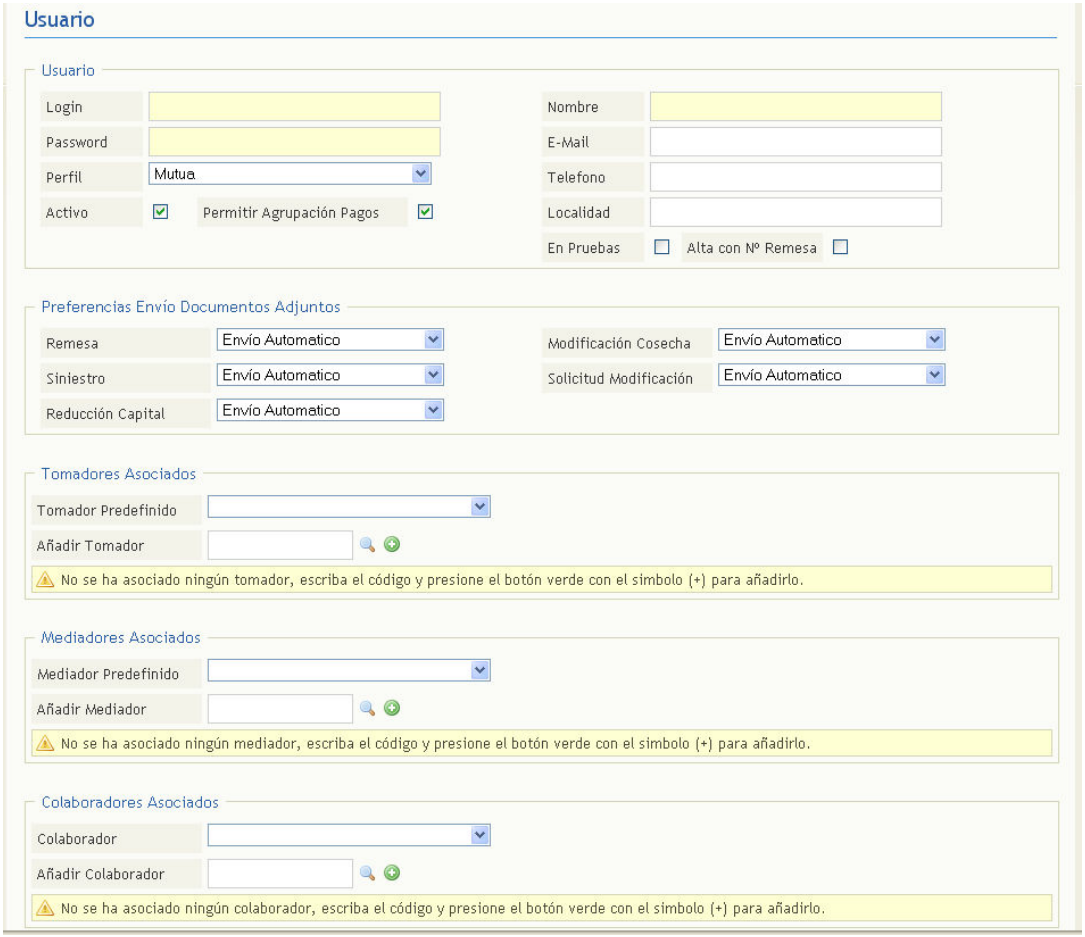


MODO ADMINISTRADOR

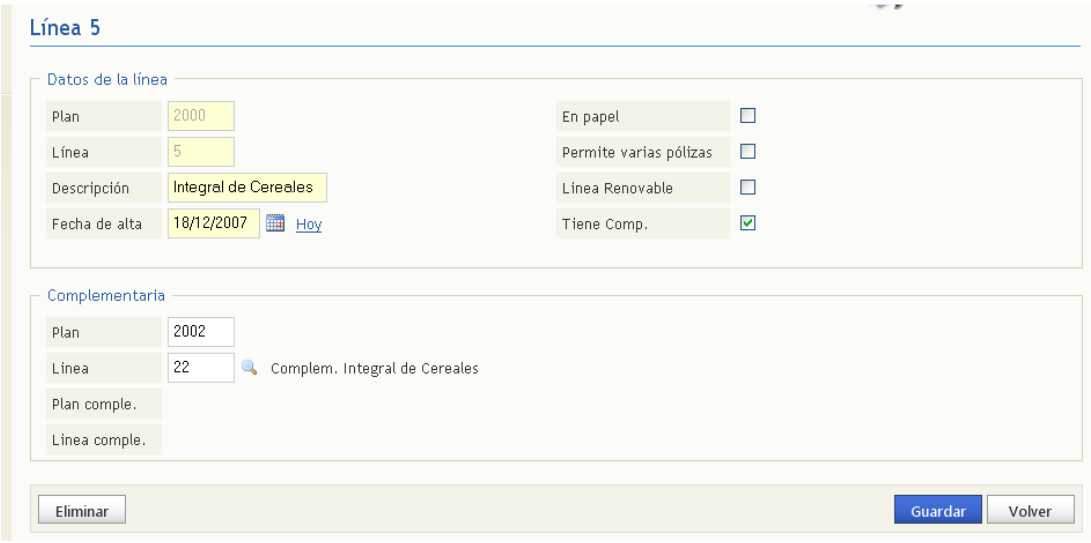
El administrador es el superusuario de la aplicación y tendrá acceso tanto a la gestión de todo lo relacionado con datos y configuraciones como a la modificación y rectificación de datos introducidos por todos los usuarios normales.

Estas son las funcionalidades que puede realizar:

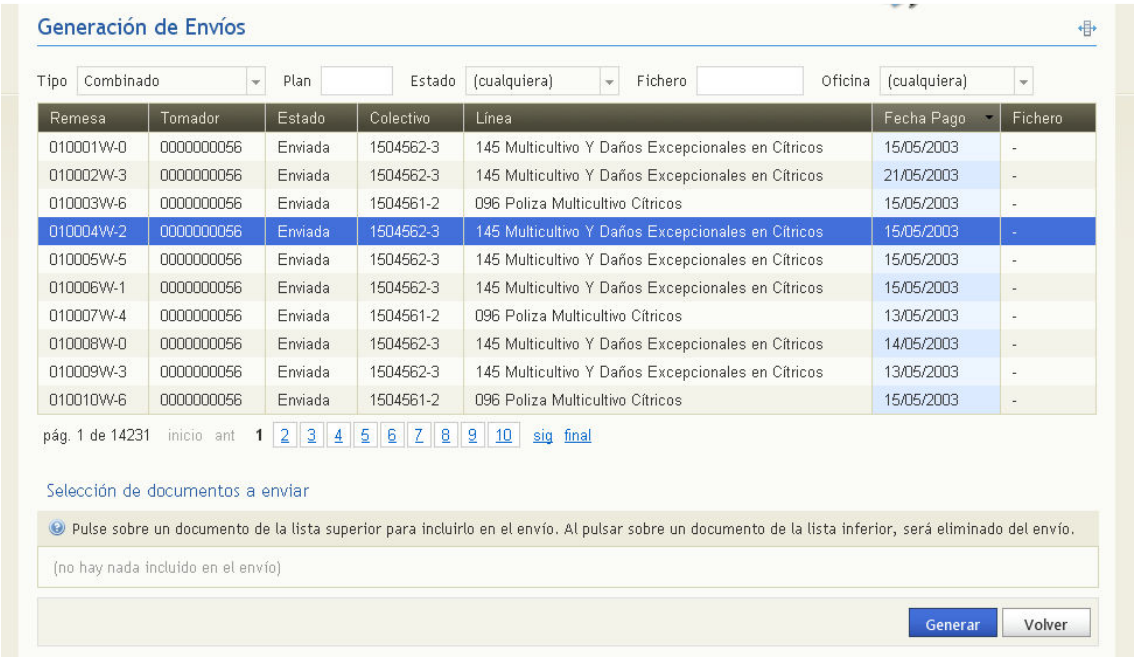
Crear usuario	
Entrada	Login, Password, Nombre, E-mail, Telefono, Localidad, Perfil, Activo, Permitir agrupación de pagos, Preferencias de: remesa, siniestro, reducción de capital, modificación de cosecha y solicitud de modificación. Lista de Tomadores, listas de Mediadores, lista de Colaboradores
Proceso	Comprobar: • Debe haber al menos 1 Colaborador • Debe haber al menos 1 Mediador • Debe haber al menos 1 Tomador Obtener identificadores de la lista de Tomadores, Mediadores y Tomadores. Enviar mail, si existe. Crear usuario
Salidas	Confirmación que la operación se ha realizado con éxito.



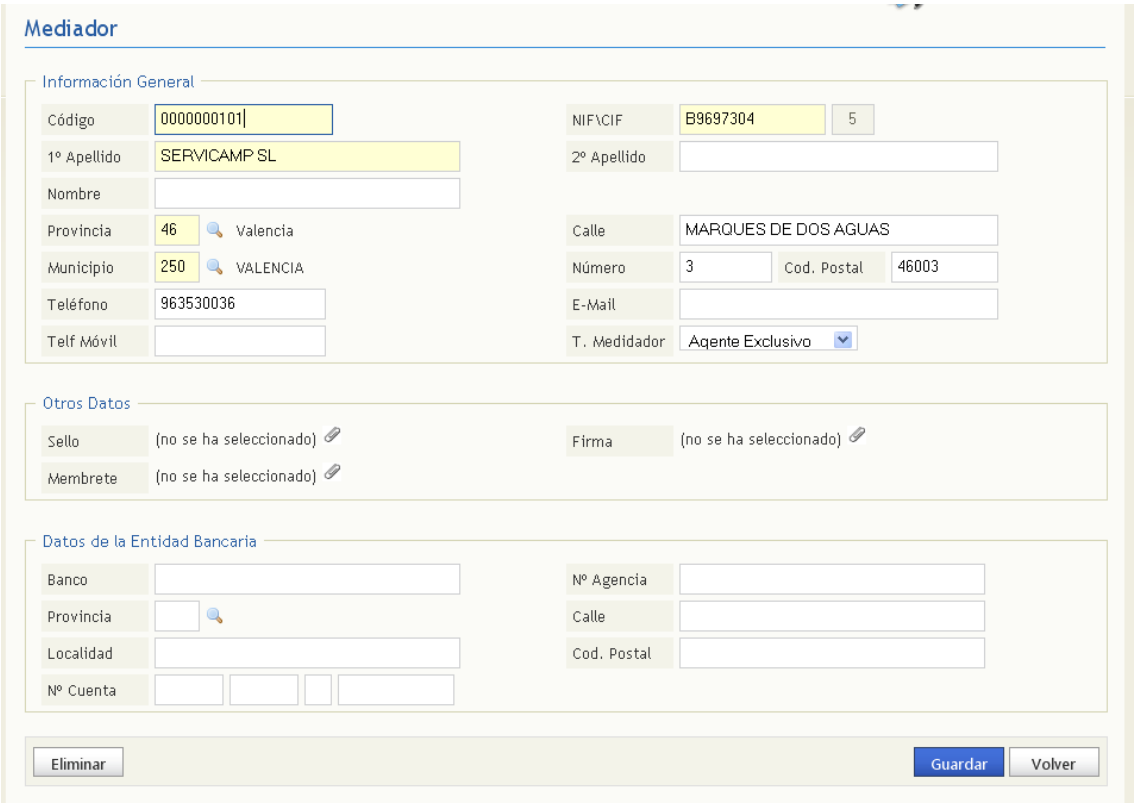
Crear línea de seguro	
Entrada	Plan, Línea, Descripción, Fecha alta, Papel, Varias pólizas, línea renovable, Complementario (si existe): Plan, Línea.
Proceso	Comprobar: • Validar campos obligatorios: Línea, Plan • Si se ha marcado complementaria, el plan y la línea complementaria tienen que estar completados. Crear línea de seguro
Salidas	Confirmación que la operación se ha realizado con éxito.



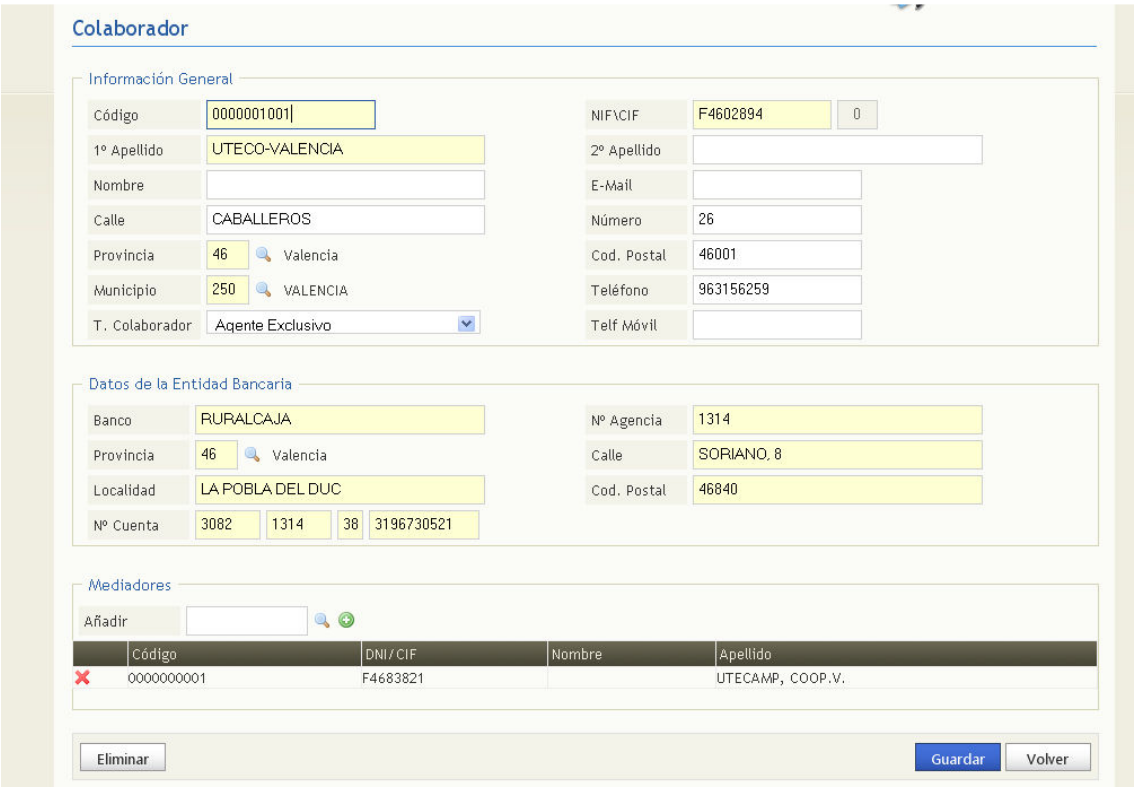
Envío manual de póliza y remesa	
Entrada	Lista de remesas seleccionadas.
Proceso	Comprobar: • que las líneas y plan de la remesas son las mismas. • que tienen el mismo tipo. Preparar la conexión con el FTP de Agroseguro. Generar archivos txt de acuerdo a la norma. Empaquetar los archivos Transmitir los archivos. Cambiar el estado de la remesa a Enviada Cambiar estado remesa como “enviada”.
Salidas	Confirmación que la operación se ha realizado con éxito. Mostrar el nuevo estado de la remesa.



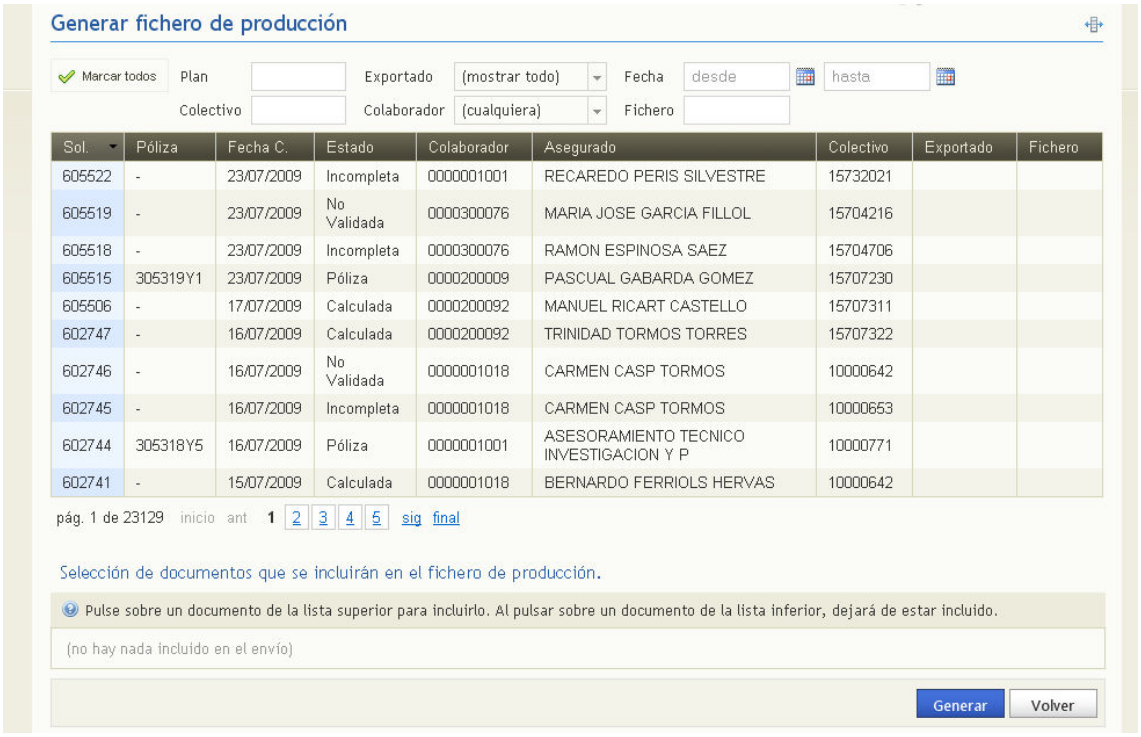
Añadir mediador	
Entrada	Código, 1º apellido, 2º apellido, nombre, NIF, Provincia, Municipio, Teléfono, Calle, numero, CP, email, tipo mediador, sello, firma, membrete, datos bancarios
Proceso	Validar campos obligatorios: NIF, apellidos, código, provincia y municipio Si tiene sello, membrete o firma, se debe comprobar que existe un archivo válido. Si se ha incluido un número de cuenta, debe ser correcto. Validar que no existe ya el código. Crear mediador
Salidas	Confirmación que la operación se ha realizado con éxito.



Añadir colaborador	
Entrada	Código, 1º apellido, 2º apellido, nombre, NIF, Provincia, Municipio, Teléfono, Calle, numero, CP, email, tipo mediador, sello, firma, membrete, datos bancarios, lista de mediadores
Proceso	Validar campos obligatorios: NIF, apellidos, código, provincia y municipio Debe tener al menos un mediador en la lista. Mediadores > 0 Si se ha incluido un número de cuenta, debe ser correcto. Validar que no existe el código en base de datos. Crear colaborador
Salidas	Confirmación que la operación se ha realizado con éxito.



Generar fichero de producción	
Entrada	Lista de solicitudes
Proceso	Validar que los registros sean correctos. Generar fichero .mdb y incluirlo en un directorio temporal. Recorrer los registros e ir añadiéndolos al archivo .mdb.
Salidas	Fichero .mbd



2.3.3. REQUISITOS DE EFICIENCIA

Uno de los aspectos a tener en cuenta en la aplicación es la eficiencia. Los usuarios deben trabajar fluidamente desde sus estaciones de trabajo. Para ello se recomienda a todos los usuarios que vayan a utilizar Agromutua.Web una conexión de al menos 1 Mb de subida y de 500Kb de bajada, o lo que es lo mismo cualquier conexión convencional de ADSL del mercado.

Las estadísticas indican que la eficiencia con las interfaces hardware disponibles es optima, aún teniendo contratada la opción mínima recomendada en esta ERS (punto 2.3.1.2).

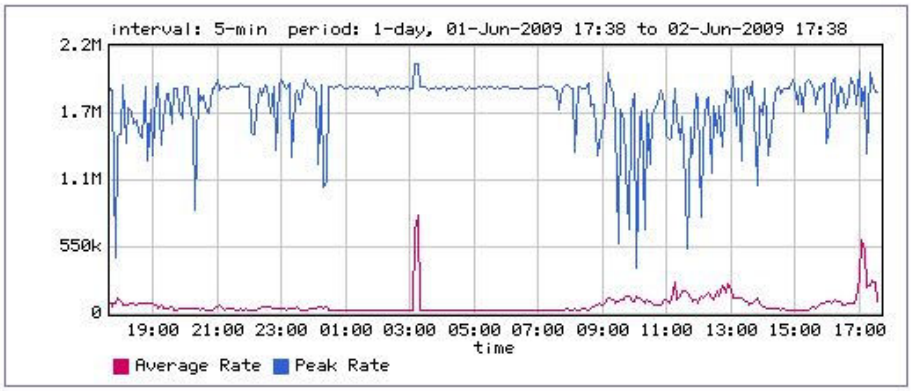
Los datos que se muestran en la gráfica de a continuación muestran las visitas durante un periodo que va de 1 de Junio del 2009 al 30 de Junio del 2009. Las conclusiones que se pueden obtener son que el nivel de contratación es constante y que oscila entre las 100 y 200 visitas diarias. Durante este mes la mutua indica que la contratación realizada en web es del 30%.

Obsérvese en el dato que apunta el cursor el día 2 de Junio se recibieron 118 visitas.

Los datos proporcionados por el proveedor de la conexión referentes a ese día muestran claramente que es eficiente el ancho de banda contratado y que está preparado para disponer de un carga de usuarios superior 100% de la contratación estimada por la mutua.

Bandwidth Utilization with Peaks Report
for Class /Inbound/Hosting_Dedicado/HD_AgroMutua

Class Utilization with Peaks



2.3.4. RESTRICCIONES DE DISEÑO

En lo que respecta al navegador web, se dará soporte para trabajar con Internet Explorer 7 y Mozilla Firefox 2. El resto de navegadores quedan, en principio, excluidos de la propuesta, pese a que se entiende que la mayoría de ellos podrán ejecutar la aplicación en la medida en que se ciñan a los estándares establecidos en Internet.

Se asume la configuración por defecto de cada uno de ellos como plataforma estándar de desarrollo cliente. Esto implica, niveles de seguridad medios para internet, JavaScript habilitado, cookies habilitadas y restricciones por defecto en cuanto al uso de ventanas emergentes y control de descargas. Se intentará que la aplicación sea, en la medida de lo posible, compatible con XHTML 1.0 Transitional y CSS de nivel 2, según los estándares del W3C (http://www.w3.org/TR/xhtml1/, http://www.w3.org/TR/REC-CSS2/).

2.3.5. ATRIBUTOS

El proyecto AGROMUTUA.WEB, además de los anteriores requerimientos enunciados, deberá cumplir los siguientes requerimientos de carácter no funcional y considerados como atributos de la aplicación.

Se considerará atributo todo aquello que sea una característica del proyecto y no que no se pueda plasmar en código y quede como norma para los desarrolladores.

1. Las nuevas tablas de datos que se necesiten se crearán sobre SQL SERVER 2005, la mayoría de estas serán heredadas de la base de datos antigua de WINCAMP pero se pueden crear nuevas para apoyo o para mejorar las anteriores. Igual sucede con las relaciones entre ellas.
2. La arquitectura de aplicaciones .NET a utilizar en el proyecto será una arquitectura en tres capas.
3. Se restringirá la utilización del sistema y de acceso a los datos e información mediante mecanismos que permitan la identificación, la autenticación, la gestión de derechos de acceso y, en su caso, la gestión de privilegios. Para ello se crearán los debidos campos en la base de datos al realizar cálculos o operaciones que lleven un riesgo para la mutua. De la misma forma se recogerá un log en el servidor con los procesos que se realizan.
4. El sistema deberá mantener un registro de los inicios y cierre de sesiones de cada usuario, así como las actualizaciones de datos que se han realizado.
5. Se agilizará la entrada de datos con valores por defecto para cada atributo según se establezca en las reuniones con los usuarios finales.
6. El sistema permitirá, mediante diálogos o prefiltros, fijar criterios de filtrado y ordenación de la información, para su explotación a formatos Microsoft Excel. Se llamarán “Trabajar Con” o “TC” y están explicados en el punto anterior.
7. En lo que respecta al navegador web, se dará soporte para trabajar con Internet Explorer 7 y Mozilla Firefox 2. El resto de navegadores quedan, en principio, excluidos de la propuesta, pese a que se entiende que la mayoría de ellos podrán ejecutar la aplicación en la medida en que se ciñan a los estándares establecidos en Internet.
8. Se asume la configuración por defecto de cada uno de ellos como plataforma estándar de desarrollo cliente. Esto implica, niveles de seguridad medios para internet, javascript habilitado, cookies habilitadas y restricciones por defecto en cuanto al uso de ventanas emergentes y control de descargas. Se intentará que la aplicación sea, en la medida de lo posible, compatible con XHTML 1.0 Transitional y CSS de nivel 2, según los estándares del W3C (http://www.w3.org/TR/xhtml1/, http://www.w3.org/TR/REC-CSS2/).
9. En vistas a la escalabilidad de la aplicación, la reducción de tráfico de datos y la mejora de la experiencia del usuario, el cliente deberá tener Javascript habilitado y este seguir los estándares del ECMA referentes a ECMAScript (ECMA-262). Los estándares propuestos se intentarán seguir siempre y cuando no resulten en una pérdida de funcionalidad de la aplicación.
10. Sistemas Operativos Cliente: Se acepta cualquier sistema operativo cliente que pueda ejecutar alguno de los navegadores propuestos (IE7, Firefox2) y que cumpla con los requisitos de fuentes, visibilidad, funcionalidad, etc. especificados por la aplicación. Las pruebas se realizarán sobre Windows XP y Windows Vista.
11. Soporte Hardware Cliente: Se establecen los mínimos equivalentes a la ejecución de cualquiera de los dos navegadores propuestos.
12. Se recomienda una resolución mínima de 1024x768 para las pantallas cliente, entendiendo que es el mínimo para que muchos de los formularios definidos en el proceso de análisis quepan en la pantalla para facilitar el trabajo. En la medida de lo posible se intentará que la aplicación trabaje con resoluciones superiores a 1280x1024. Se establece esta última resolución como límite para las pruebas funcionales que se realizarán. Dado que casi la totalidad de tarjetas gráficas del mercado lo permite, se diseñarán los diferentes formularios con una profundidad de color de 32bits.

3. ARQUITECTURA Y TECNOLOGÍAS

En el capitulo anterior se ha presentado el negocio en la aplicación Agromutua.Web, en el actual se va a presentar la arquitectura utilizada y las tecnologías empleadas para construirla y poder poner en práctica las especificaciones del negocio. Como en todos los proyectos a largo plazo es lógico que durante la fase de desarrollo aparezcan nuevas tecnologías o evolucione la actual que lejos de dejarlas de lado se han ido incorporando para poder beneficiarse de sus ventajas, como es el caso de LINQ.

Este capítulo servirá de recorrido por las capas de Agromutua.Web, analizando cada una y viendo como se ha construido la arquitectura y las tecnologías empleadas para implementarla, desde la interfaz utilizando JavaScript para realizar peticiones AJAX hasta la utilización de C# y VB.NET para controlar la lógica dentro del servidor.

3.1. ARQUITECTURA DEL PROYECTO

Agromutua.Web es un proyecto con más de dos años de vida, durante este periodo las tecnologías han evolucionado y se han tenido que ir incluyendo tanto en él proyecto a nivel de interfaz como en su arquitectura, el cambio más importante viene dado sobre todo por la aparición del Framework 3.5 de .NET y la aparición de LINQ como una de las características más destacadas.

La aplicación usa una arquitectura multicapa convencional dividida en tres niveles bien diferenciados como interfaz, negocio y persistencia. Este modelo de arquitectura consigue el propósito de la separación de contenidos y crear una dependencia entre las capas sin mezclar funcionalidad.

Con la aparición de LINQ la capa de persistencia se “solapó” con la de negocio apareciendo un mapeado completo de la base de datos y dejando de lado la tediosa programación que suponía la interacción con objetos ADO.NET. A nivel de arquitectura se podría debatir si la inclusión de LINQ sería o no otra capa, en el capítulo 5 de esta tesina se profundizará en este aspecto, de momento en la explicación de la arquitectura se presentará como un elemento intermedio entre la capa de persistencia y la de negocio.

La migración al Framework 3.5 y la aparición de LINQ supusieron un cambio en la arquitectura y en la forma de pensar del programador, permitiendo acceder rápidamente a la base de datos para obtener los datos, pero por otro lado la convivencia de dos partes de acceso a datos muy diferentes. Mientras LINQ se disemina por todo el proyecto (aún teniendo la aparición de una capa de persistencia paralela) el acceso a datos con ADO.NET se concentra claramente en la capa de persistencia. Dicho de otra forma, el acceso sencillo a datos que nos proporciona LINQ permite abusar de él en cualquier parte, ya que en consultas individuales y no comunes a otras es mucho más rápido incluir un fragmento LINQ que no realizar todo el recorrido por todas las capas, escribiendo código que muchas veces se utiliza apenas.

De manera gráfica se muestran unas imágenes de la arquitectura con la que se empezó y en el punto en el que está actualmente con la aparición de LINQ:

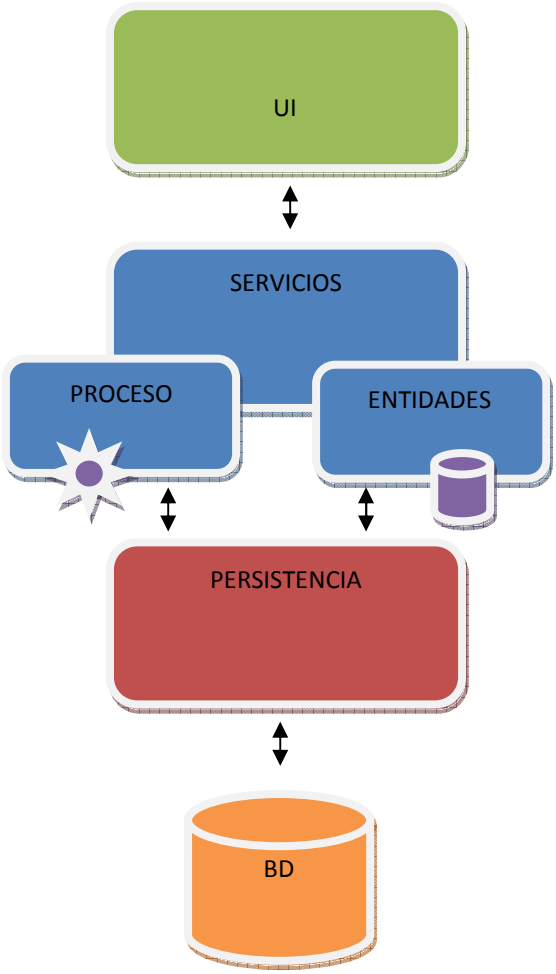


Figura 2, Arquitectura antes

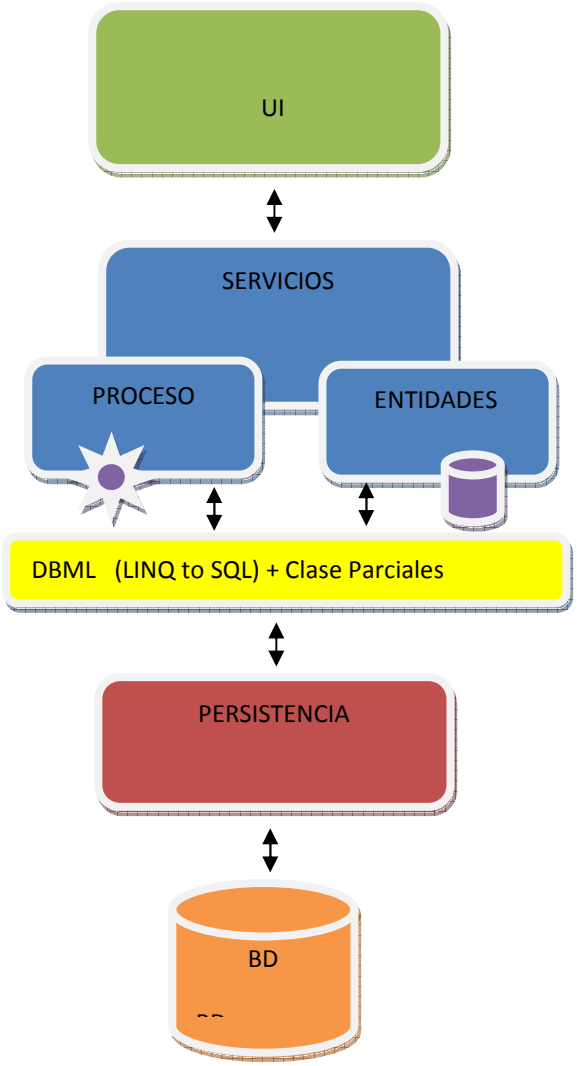


Figura 3, Arquitectura actual

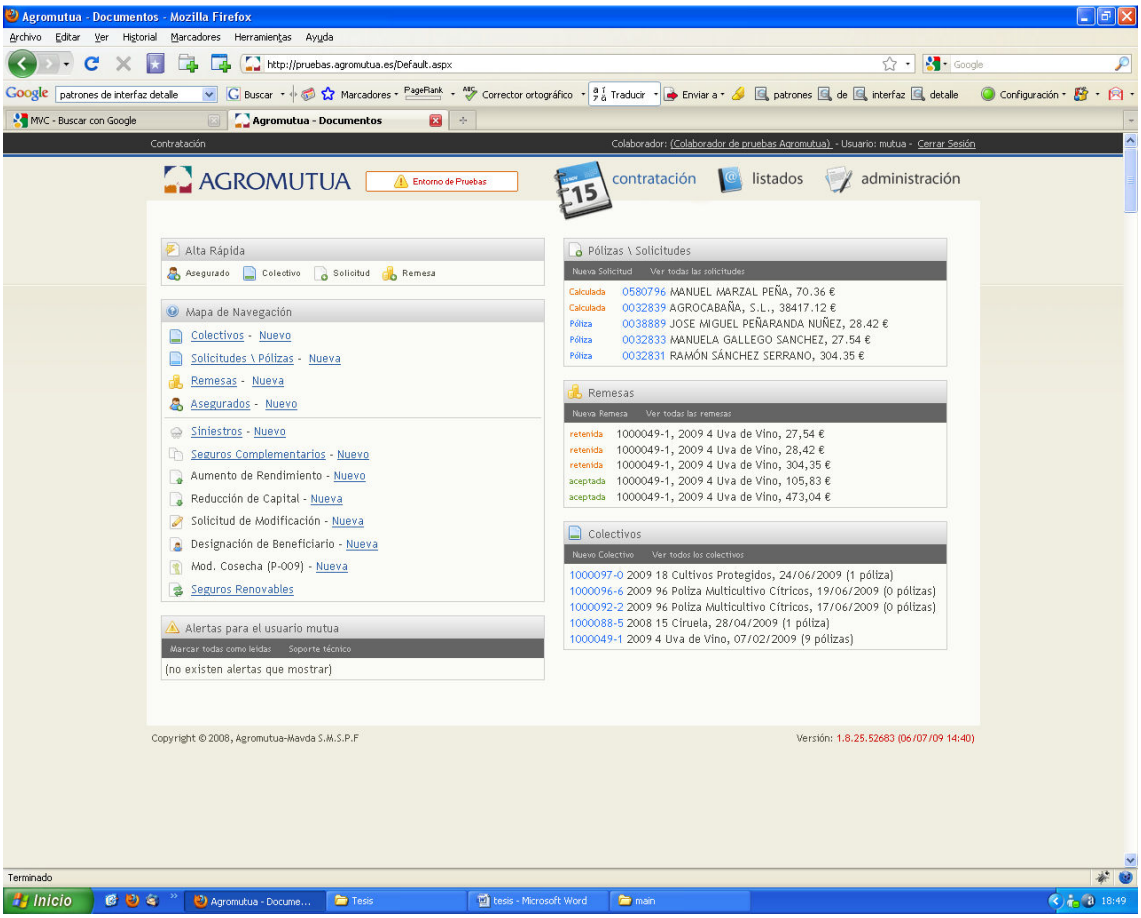
La *figura 2* muestra la primera arquitectura, mientras que la *figura 3* muestra la nueva arquitectura. Aunque realmente es la misma, la arquitectura de 3 capas (Interfaz, Servicios y Persistencia) la nueva arquitectura ha pasado a tener una capa extra intermedia entre persistencia y servicios que incluye los DBML generados por el mapeo de la base de datos para la utilización de LINQ to SQL, estos DBML que veremos más adelante no son otra cosa que las tablas de la base de datos debidamente mapeadas, y a las cuales les añadimos clases parciales para hacer más rica esta capa, estas clases parciales nos proporcionan validaciones y funcionalidad extra.

El mapeo de la base de datos permite ahorrarse además de tiempo, muchas comprobaciones a nivel de tipos de datos y escritura de consultas. Como se verá en el capítulo 4, LINQ to SQL ayuda sobre todo a realizar tareas de programación e interacción con los datos de una forma más rápida y sencilla.

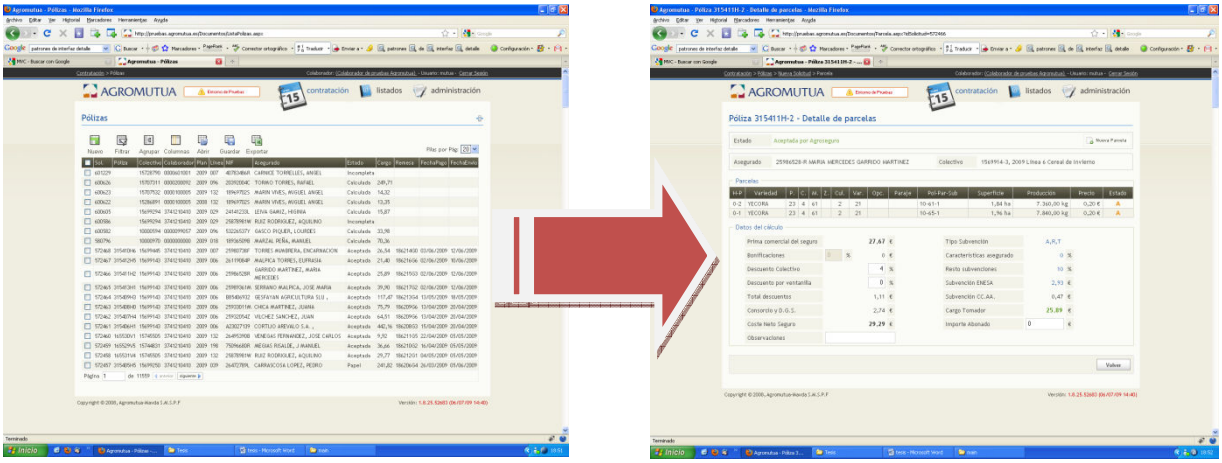
3.1.1. LA INTERFAZ

La interfaz de Agromutua.Web se encarga de mostrar al usuario la información comunicándole la información y capturando la información introducida por el usuario mediante formularios.

Basada en el patrón de listado-detalle, la usabilidad que se percibe al trabajar con la aplicación es alta debido al nivel de detalle y la distribución de los contenidos. Véase la pantalla de inicio:



A continuación un ejemplo de listado-detalle:



La interfaz es la capa situada en la parte superior de la arquitectura, teniendo por debajo la capa de lógica que contiene las entidades y el negocio de la aplicación y que va a ser la encargada de suministrar la información y tratar la información introducida en formularios o producida por el usuario al interactuar con la aplicación.

3.1.2. La capa de Servicios o Lógica de negocio

La capa de servicios es la encargada de toda la lógica del proyecto y de ella depende en gran medida el correcto funcionamiento de la aplicación ya que el negocio depende de ella.

Si se observa el dibujo de la arquitectura se ve que esta capa está representada en dos subcapas, que en realidad no es otra cosa que la capa de Entidades que aglutina todas las clases con constructores, destructores, etc. y la subcapa de Proceso que incluye todos los métodos y funcionalidades.

Por tanto los objetos que navegan por el proyecto entre capas van a depender de la estructura definida en la capa de Entidades. Mientras que las peticiones de usuario y los datos agregados que dependan de reglas de validación se procesarán en la parte de Proceso.

Dicha capa intermedia va a permitir la transferencia de datos entre interfaz y persistencia, y es aquí donde LINQ y más concretamente LINQ to SQL va a ser clave en la nueva arquitectura. Para hacer funcionar LINQ to SQL se mapean las tablas de la base de datos en objetos (véase punto 4.5.1), dentro de lo que es el DBML, al mapear se consigue que se puedan crear objetos y colecciones que representen los datos de la base de datos integrados dentro del código funcional. Una de las características de LINQ to SQL y del mapeo que realiza es que cuando genera las clases, estas clases están definidas de forma parcial, lo que permite añadir más fragmentos de la misma clase en diversos archivos, por ejemplo para realizar validaciones que no realiza el propio LINQ y que sean del negocio.

Esta características va a permitir ahorrar muchas horas al programador, cuando se hablaba de la arquitectura original de Agromutua.Web las subcapas de Entidades y Proceso había que programarlas y más complicado aún, mantenerlas. Ahora con LINQ to SQL tanto el mantenimiento como coste en horas de trabajo disminuye significativamente.

El único inconveniente que surge con la aparición de LINQ to SQL en el proyecto es la tentación de hacer consultas a la base de datos en la capa de interfaz, que de hecho se realiza en este proyecto apareciendo consultas diseminadas por archivos de la interfaz. Pero esto aunque parezca que deja de lado la arquitectura, cumplirlo a rajatabla sería además de laborioso, en ocasiones inútil, ya que en ocasiones las consultas son únicas o muy concretas haciéndose factible sobre todo para el programador saltarse el concepto de la arquitectura.

3.1.3 PERSISTENCIA

La capa de nivel inferior y la capa encargada de transmitir y estar en continuo contacto con la base de datos. La capa de Persistencia se va a encargar de guardar, obtener y actualizar los registros de la base de datos. Cuando se trabaja con LINQ to SQL, la capa va a ser autoimplementada y posiblemente lo más interesante será que no se va a tener que hacer un esfuerzo en la comprobación del tipo de datos que viene y va hacia la base de datos ya que LINQ to SQL a nivel de persistencia lo controlará, una tarea bastante complicada en la arquitectura anterior con ADO.NET.

En la nueva arquitectura el mantenimiento de esta capa se reducirá a la actualización del mapeo de la base de datos, y la generación de un DBML actualizado.

3.2 TECNOLOGÍAS

En Agromutua.Web el uso de una tecnología u otra no fue una decisión de la empresa sino del cliente. Se trabaja con un entorno de Visual Studio 2008 y el Microsoft Framework 3.5 (que incluye LINQ), por tanto al ser un proyecto web utiliza ASP.NET, C# y VB.NET. Al ser un aplicación muy cargada de datos y cabe la posibilidad de que en determinadas ocasiones sean muchos los usuarios conectados se vio necesario la utilización de mucho JavaScript y AJAX (el sp1 de Visual Studio proporciona un gran soporte para AJAX) para no sobrecargar el servidor y realizar algunas tareas en el cliente (navegador). La base de datos a la que se conecta es SQL Server 2005 y como se verá más adelante se utilizan diferentes versiones dependiendo del puesto de trabajo.

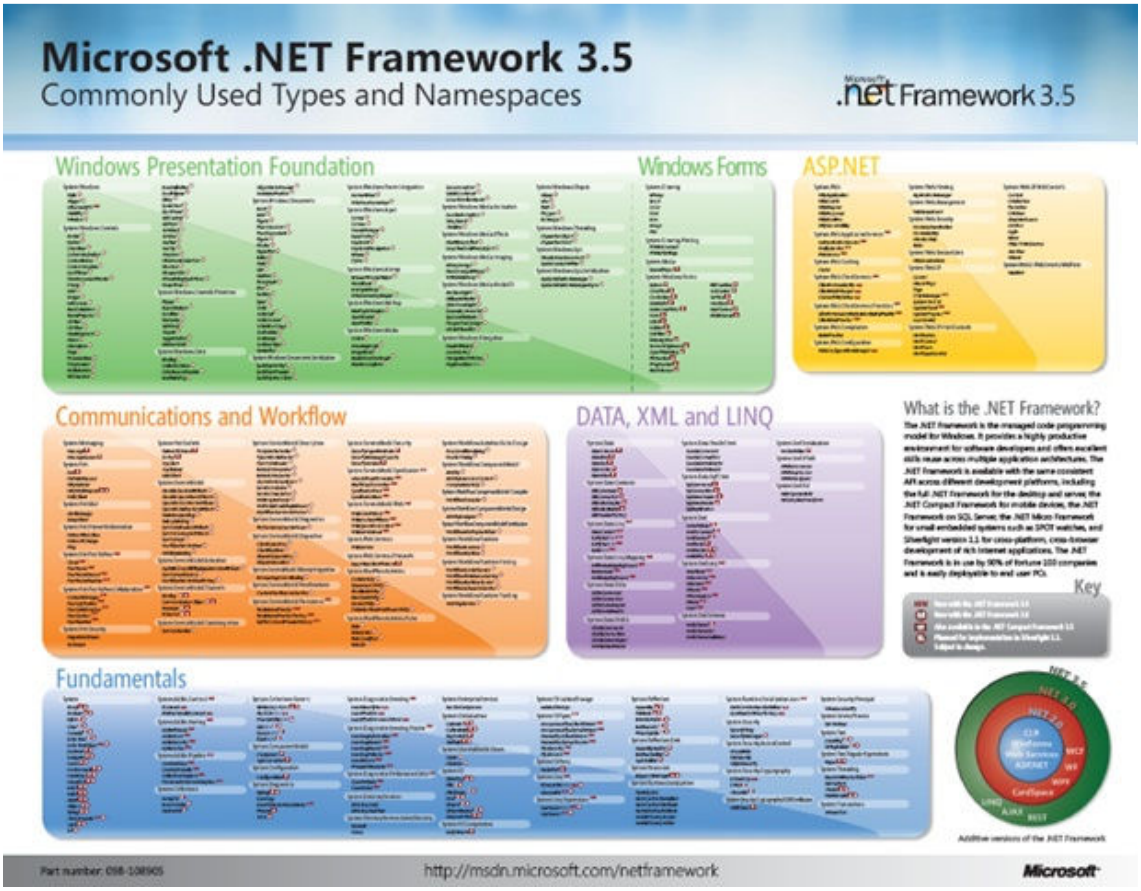
3.2.1 ENTORNO DE DESARROLLO: VISUAL STUDIO 2008

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Este potente IDE permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET. Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

Visual Studio 2008 ofrece herramientas de desarrollo avanzadas, funciones de depuración de código, funciones para bases de datos y funciones innovadoras que permiten crear rápidamente aplicaciones para distintas plataformas. En la versión 2008 incluye mejoras en los diseñadores visuales para un

desarrollo más rápido con .NET Framework 3.5, mejoras sustanciales en las herramientas de desarrollo Web y mejoras de programación que aceleran el desarrollo a partir de todo tipo de datos.



Visual Studio 2008 ofrece todo el soporte requerido para frameworks y herramientas, necesario para crear aplicaciones AJAX para Web, completas y expresivas. Frameworks que permiten construir fácilmente aplicaciones Web concentradas en los clientes, que se integran con cualquier proveedor de datos de back-end, que se ejecutan dentro de cualquier explorador moderno, y que tienen acceso total a los servicios de aplicaciones ASP.NET y de la plataforma Microsoft.

Para que los desarrolladores puedan crear rápidamente software moderno, Visual Studio 2008 ofrece funciones de programación y de datos mejoradas, como LINQ *(Language Integrated Query)*, que facilita el armado de soluciones capaces de analizar información y de actuar en consecuencia. Visual Studio 2008 también brinda la posibilidad de trabajar con distintas versiones de .NET Framework desde el mismo entorno de desarrollo. Por lo tanto, se podrán construir aplicaciones que apunten a .NET Framework 2.0, 3.0 o 3.5, y así podrán admitir una amplia variedad de proyectos en un mismo entorno.

Las nuevas herramientas aceleran la creación de aplicaciones conectadas con las últimas plataformas, incluidas la Web, Windows Vista, Office 2007, SQL Server 2008 y Windows Server 2008. Para la Web, ASP.NET AJAX y otras nuevas tecnologías permiten que los desarrolladores creen rápidamente una nueva generación de experiencias más eficientes, interactivas y personalizadas.

Proporciona herramientas mejoradas que ayudan a mejorar la colaboración entre equipos de desarrollo, incluyendo herramientas que colaboran con la integración entre profesionales especializados en bases de datos y diseñadores gráficos.

.NET Framework permite la construcción rápida de aplicaciones ofreciendo componentes (software pre-fabricado) que resuelve las tareas de programación más frecuentes. Juntos, Visual Studio y .NET Framework reducen la necesidad de código en común, disminuyendo los tiempos de desarrollo.

Con la llegada del .NET Framework 3.5 las mejoras fueron aplicadas a áreas fundamentales como la biblioteca básica de clases, Windows Workflow Foundation, Windows Communication Foundation, Windows Presentation Foundation y Windows CardSpace. [10]

3.2.2. SERVIDOR WEB: INTERNET INFORMATION SERVER (IIS)

IIS proporciona una serie de servicios para los servidores que funcionan con Windows. Originalmente era parte del *Option Pack* para Windows NT. Luego fue integrado en otros sistemas operativos de Microsoft destinados a ofrecer servicios, como Windows 2000 o Windows Server 2003.

Los servicios que ofrece son: FTP, SMTP, NNTP y HTTP/HTTPS.

Los servicios de Internet Information Services (IIS) proporcionan las herramientas y funciones necesarias para administrar de forma sencilla un servidor Web seguro, gestionando las tareas para la administración de sites y el acceso a máquinas.

El servidor web se basa en varios módulos que le dan capacidad para procesar distintos tipos de páginas, por ejemplo Microsoft incluye los de Active Server Pages (ASP) y ASP.NET. También pueden ser incluidos los de otros fabricantes, como PHP o Perl.

3.2.3. SISTEMA GESTOR DE BASE DE DATOS: SQL SERVER 2005

Para la gestión de la base de datos se ha utilizado Microsoft SQL Server 2005, un sistema de gestión de bases de datos relacionales (SGBD) basado en el lenguaje Transact-SQL.

SQLServer conocido inicialmente como proyecto *Yukon* no es una actualización de su predecesor 2003, sino que Microsoft ha mejorado todos sus aspectos más importantes siendo un producto muy superior y distinto a versiones anteriores. Por tanto hablamos que la versión 2005 es mucho más integrada, más flexible y ampliable que otros productos de bases de datos relacionales.

Existen cinco versiones de SQLServer 2005:

- Microsoft SQL Server 2005 Enterprise Edition
- Microsoft SQL Server 2005 Standard Edition
- Microsoft SQL Server 2005 Workgroup Edition
- Microsoft SQL Server 2005 Developer Edition
- Microsoft SQL Server 2005 Express Edition

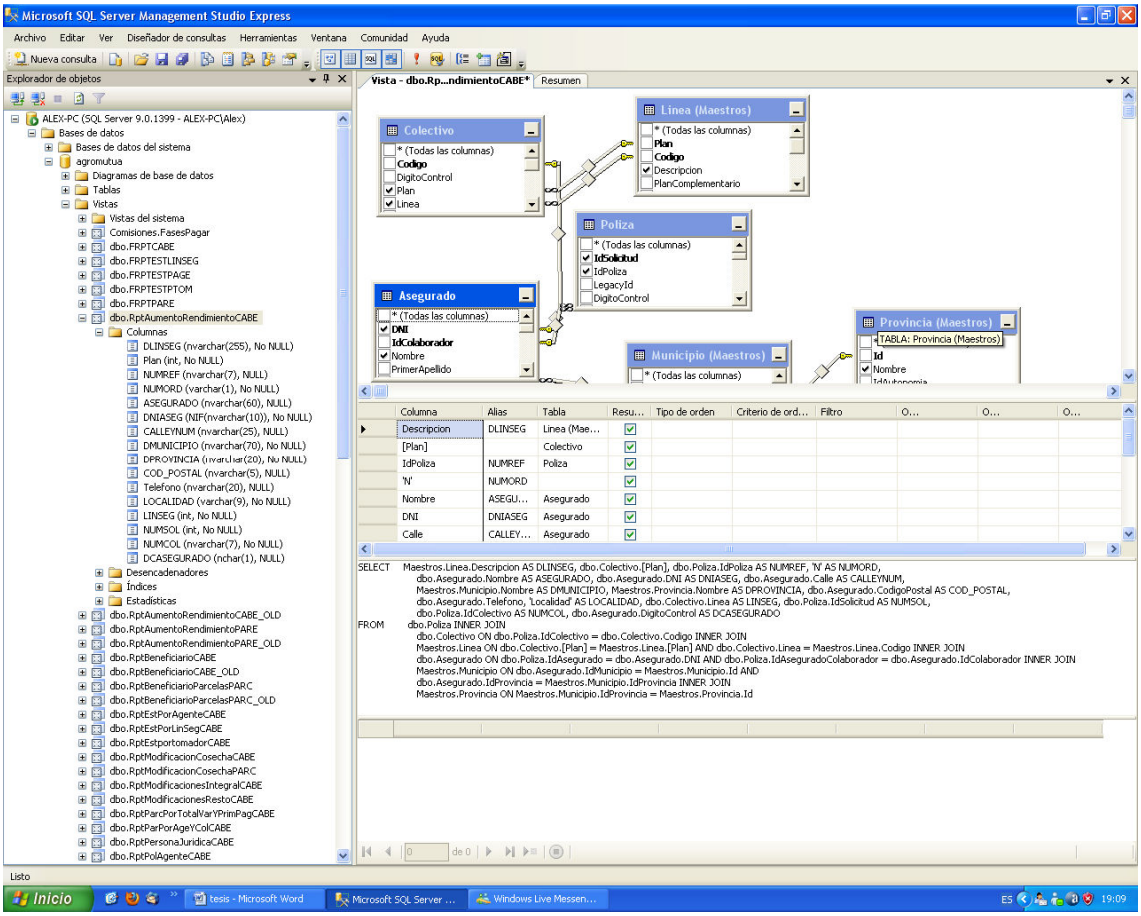
En el desarrollo de Agromutua.Web se han utilizado dos versiones, Microsoft SQL Server 2005 Enterprise Edition para gestionar la base de datos del servidor de la aplicación y la edición reducida de Microsoft SQL Server 2005 Express Edition disponible para su descarga en la web de SQL Server para el desarrollo de la aplicación en las maquinas de los programadores.

Microsoft SQL Server 2005 Enterprise Edition se presenta en las variedades de 32 y 64 bits, utilizando Agromutua.Web la primera. Se trata de la edición ideal cuando se necesita que el SGBD se pueda ampliar a un tamaño ilimitado al tiempo que ofrece procesamiento transaccional en línea (OLTP, *Online Transaccion Processing*) de nivel empresarial, análisis de datos muy complejos, sistemas de almacenamiento de datos y sitios Web.

Esta versión viene con todos los accesorios y está perfectamente equipado para proporcionar capacidades exhaustivas analíticas y de inteligencia empresarial. Incluye funcionalidades de alta disponibilidad como creación de reflejos de las bases de datos y de clústeres por conmutación de errores.

Microsoft SQL Server 2005 Express Edition es una base de datos gratuita disponible solo en 32 bits y se caracteriza por su sencillez a la hora de manejarla. Prescinde de muchas de las funciones de las demás ediciones, como Management Studio, Notification Services, Analysis Service, Integration Services y el Generador de informes, por nombrar unas cuantas. SQL Server Express puede funcionar como base de datos cliente o como base de datos de servidor básica. Se trata de una buena elección cuando no se realizan tareas avanzadas y solo se requiere para disponer locamente de la base de datos.

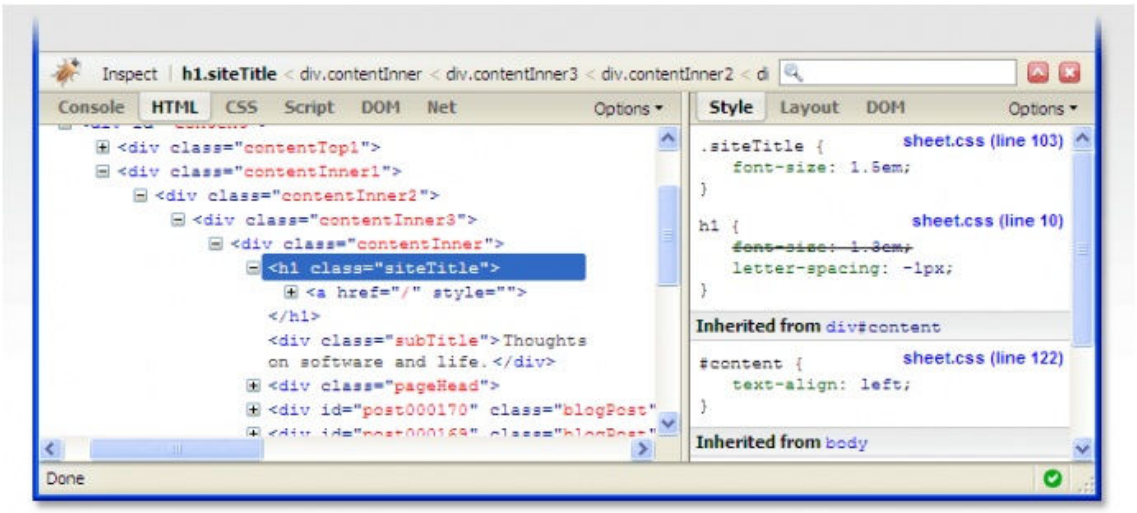
Uno de los aspectos clave incorporados a SQL Server 2005 es el alto grado de integración entre los aspectos de la programación y de la administración. Ejemplo de esto es la total remodelación de SQL Server Management Studio [3]



Actualmente existe una versión superior en el mercado, SQL Server 2008 con nuevas características muy interesantes como la utilización del *Intellisense* de Microsoft en las consultas.

3.2.4. HERRAMIENTAS PARA AUMENTAR EL RENDIMIENTO EN EL DESARROLLO

Firebug es una extensión de Firefox creada y diseñada especialmente para desarrolladores y programadores web. Es un paquete de utilidades con el que se puede analizar (revisar velocidad de carga, estructura DOM), editar, monitorizar y depurar el código fuente, CSS, HTML y JavaScript de una página web de manera instantánea e *inline*.



Firebug no es un simple inspector como DOM Inspector, además edita y permite guardar los cambios. Su atractiva e intuitiva interfaz, con solapas específicas para el análisis de cada tipo de elemento (consola, HTML, CSS, Script, DOM y red), permite al usuario un manejo fácil y rápido. Esta herramienta está encapsulada en forma de plug-in o complemento de Mozilla, es Open Source y de distribución gratuita.

3.2.5. LENGUAJES DE PROGRAMACIÓN

Como se ha mencionado en la introducción de este punto, se han utilizado varios lenguajes de programación diferentes. La utilización de estos lenguajes va normalmente ligada a la capa donde se utiliza. De esta forma la interfaz requiere ASP.NET para utilizar sus controles principalmente y JavaScript conjuntamente con AJAX para no sobrecargar el servidor de peticiones. La capa de negocio por su parte utiliza lenguajes más robustos como C# y VB.NET conjuntamente para implementar la lógica y la persistencia de la aplicación.

Cuando se presentan estos lenguajes, se presentan como los lenguajes utilizados para los programadores de la aplicación, internamente Visual Studio y las tecnologías de Microsoft hacen utilizar XML, lenguajes intermedios, etc.

3.2.5.1. JAVASCRIPT

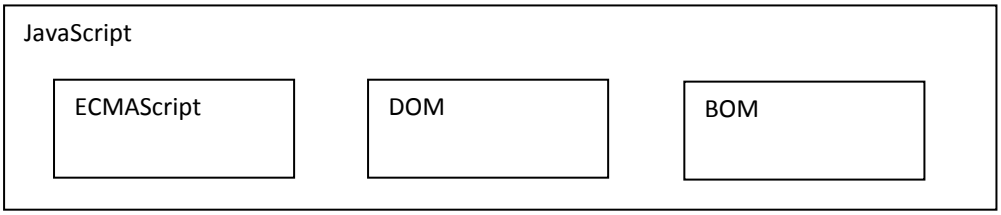
JavaScript es básicamente un lenguaje de programación de creación de secuencias de comandos en lado del cliente que se utiliza en navegadores WEB y con la propia ventana del navegador. JavaScript se basa someramente en Java, un lenguaje de programación orientado a objetos que se utiliza en la Web a través de subprogramas incrustados. Aunque Javascript cuenta con una sintaxis y metodología de programación similares, no es una versión ligera de Java, sino un lenguaje propio, que habita en navegadores Web de todo el mundo y que permite una interacción mejorada del usuario con sitios y aplicaciones Web.

JavaScript nació y creció vinculado a la web pero sus inicios fueron un poco caóticos ya que en pleno auge de las tecnologías web muchas organizaciones y empresas lanzaban sus propias versiones como CEnvi de ScriptEase , JScript de Microsoft y JavaScript de Netscape Navigator.

Fue en 1997 cuando un comité compuesto por las principales empresas interesadas en el futuro de los lenguajes de secuencias de comandos, se reunieron para elaborar el ECMA-262, un estándar que definía un nuevo lenguaje de secuencias denominado ECMAScript. Al año siguiente la ISO/IEC también adopto como estándar ISO/IEC-16262, desde entonces los navegadores han adoptada el ECMAScript como base de implementación del JavaScript.

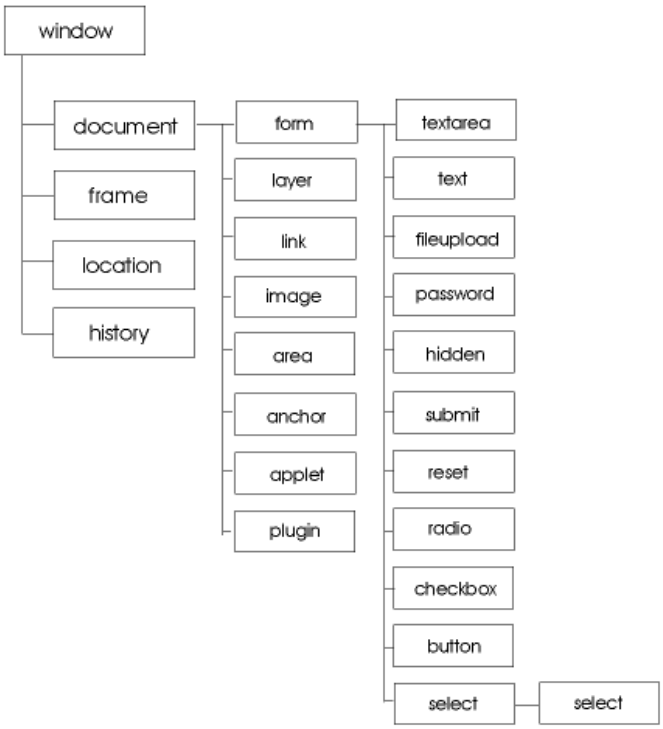
Una implementación completa de JavaScript está compuesta por tres partes:

- El núcleo (ECMASScript)
- El Modelo de objetos de documento (DOM)
- El Modelo de objetos de navegador (BOM)



ECMAS por tanto puede ofrecer funciones de creación de secuencias de comandos para distintos entornos anfitriones y, por tanto, se especifica el lenguaje de secuencias de comandos básico, desvinculado de cualquier entorno anfitrión concreto.

Por otra parte el Modelo de objetos de documento (DOM) es una interfaz de programación de aplicaciones (API) para HTML y XML. El DOM confecciona una página completa en forma de documento compuesto por una serie de nodos jerárquicos. Cada parte de una página HTML o XML parte de un nodo.



Mientras el Modelo de objetos del navegador (BOM) describe métodos e interfaces para interactuar con el navegador. Lo que convierte al BOM en algo exclusivo y en ocasiones problemático, es que es la única parte de una implementación JavaScript que carece de estándar.

En Agromutua.Web a la hora de programar en JavaScript se ha intentado estandarizar lo máximo posible el código de manera de hacerlo funcionar en todos los navegadores posibles. Aún así se ha puesto mayor empeño a que funcionara todo correctamente en Mozilla Firefox, el predecesor de Netscape Navigator.

La razón de la utilización en el proyecto de código en JavaScript viene dada para liberar de procesos, sobretodo de comprobaciones del lado del servidor, reduciendo la carga. Así con JavaScript se han implementado muchas de las pantallas y procesos que el usuario puede realizar en la aplicación y que gracias a esta tecnología se pueden realizar desde el navegador sin realizar (en ocasiones) peticiones al servidor. [5]

Para ello, se ha requerido de un potente framework JavaScript como es la Yahoo User Interface (YUI) que proporciona un marco de trabajo muy interesante para los desarrolladores y que comprende muchos componentes personalizables y extremadamente utiles en tareas web.

Yahoo **U**ser **I**nterface(YUI), una serie de bibliotecas escritas en JavaScript, para la construcción de aplicaciones interactivas (RIA). Liberadas bajo licencia BSD por parte de la compañía Yahoo. Dichas bibliotecas son utilizadas para el desarrollo web

específicamente para ser usadas como la programación de aplicaciones de escritorio, con componentes vistosos y personalizables y con una amplia implementación con AJAX.

La biblioteca está completamente documentada en su página web (http://developer.yahoo.com/yui/) y se compone de seis componentes: Núcleo YUI, utilidades, controles UI, componentes CSS, herramientas de desarrollo y de construcción.

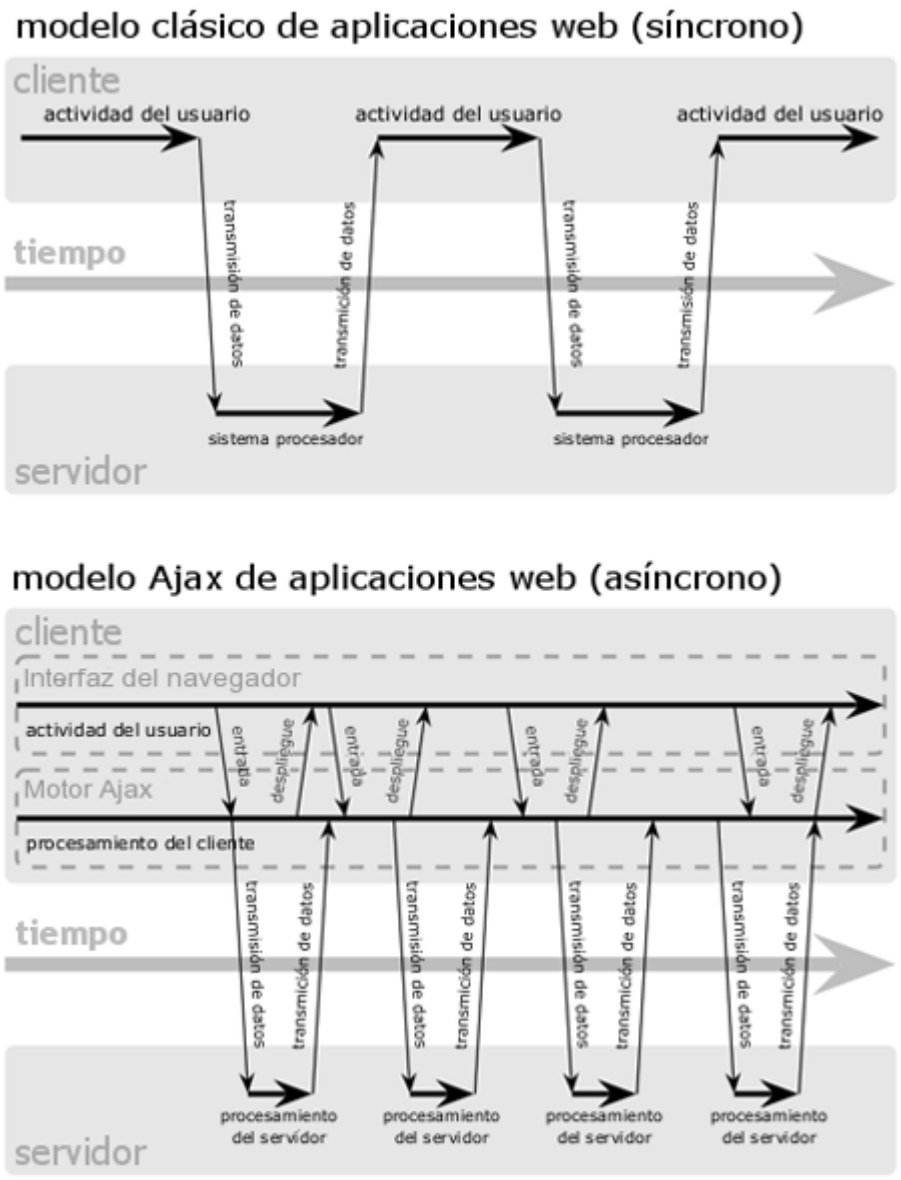
Algunos de estos componentes de YUI están orientados para hacer peticiones AJAX, véase en el siguiente punto como funciona AJAX.

3.2.5.1.1 AJAX

El nombre es un acrónimo (AJAX, Asynchronous JavaScript + XML), por tanto no es un lenguaje de programación, ni siquiera realmente AJAX es una tecnología, sino varias cada una de ellas se ha afianzado en el desarrollo web y que se unen en nuevas y potentes formas. Las tecnologías que AJAX incorpora son:

- Presentación basada en estándares utilizando XHTML y CSS
- Muestra de resultados e interacciones y dinámicas gracias al DOM
- El intercambio y manipulación de datos utilizando XML y XSLT
- Recuperación asíncrona de datos mediante el objeto XMLHttpRequest
- Y JavaScript uniéndolo todo

El modelo clásico de aplicaciones web funciona realizando peticiones web HTTP al servidor web, este las procesas y devuelve otra página web diferente al cliente. Una aplicación con AJAX elimina de la interacción en la Web la situación de arrancar-parar-arrancar-parar introduciendo un intercambio entre el usuario y el servidor [6]



Resumen detallado del gráfico:

1. Se realiza una acción
2. Se invoca el objeto XMLHttpRequest y se inicia el proceso de petición al servidor.
3. Se ejecuta la petición asíncrona mediante el objeto recién creado
4. El servidor procesa los datos recibidos, les da formato y devuelve un XML con toda la información resultante.
5. El objeto XMLHttpRequest interpreta el XML y reparte la información donde corresponde
6. La parte de la página que iba a ser actualizada con la información que el objeto XMLHttpRequest conteía esta lista, ha sido escrita y forma el objeto DOM con los nuevos datos.

Cuando se emplea AJAX en partes de la aplicación se obtienen beneficios mejorando la UI de la web, pareciéndose incluso a la de una aplicación de escritorio. El tráfico de datos con el servidor se reduce considerablemente y de manera dinámica se consigue solo refrescar una parte la web sin necesidad de recargarla entera. También es una tecnología independiente de la plataforma donde se ejecute ya que utiliza las tecnologías anteriormente mencionadas: JavaScript y XML

Lógicamente como se indica anteriormente cuando se realizan las peticiones AJAX no es necesario implementar cada vez toda la petición de inicio a fin, utilizando por ejemplo frameworks de trabajo como se explica en el punto anterior de JavaScript, la utilización de la YUI proporciona unas API muy cómodas de utilizar, tal y como se hace el Agromutua.Web. De esta forma se interactúa con el servidor serializando y deserializando las peticiones en formatos que son cómodos de utilizar como JSON u otros. [8]

3.2.5.2. ASP.NET

ASP.NET es un conjunto de tecnologías de desarrollo Web producidas por Microsoft, que se usa para construir sitios web dinámicos, aplicaciones Web y aplicaciones Web basadas en XML. ASP.NET forma parte del entorno de trabajo .NET y permite a los desarrolladores construir aplicaciones en múltiples lenguajes como VB.NET, C# y JScript. [9] [6]

La evolución del lenguaje ASP.NET desde que apareció hasta la actualidad se ha ido adaptando a los cambios y las necesidades de la web.

Cuando surgió ASP 1.0 y 1.1 la plataforma .NET de Microsoft introdujo una forma completamente nueva de programar. Los desarrolladores de Microsoft se interesan principalmente por los hilos y por la memoria. Este modelo afecta a todas las áreas de desarrollo, incluyendo al desarrollo Web, depositando una carga pesada sobre los programadores.

ASP.NET introdujo servicios de tiempo de ejecución y bibliotecas correctamente diseñadas para mejorar significativamente el desarrollo web. En cierto modo, el ASP clásico era algo “pegado” de la arquitectura IIS/ISAPI sin ningún concepto orgánico de cómo las primeras decisiones diseñadas podrían afectar posteriormente a los desarrolladores.

Entre las funciones que se incluían en ASP.NET 1.0 y 1.1 están:

- Una plataforma orientada a objetos para la definición de aplicaciones
- Separación de las declaraciones de la interfaz del usuario (HTML) y la lógica de la aplicación
- Código compilado para ejecutar la lógica de la aplicación
- Administración configurable del estado de la sesión
- Copia caché de datos integrada
- Copia caché de contenido integrada
- Una arquitectura bien definida de la componentización de la interfaz del usuario
- Componentes de alto nivel para administrar el formato de datos (grids, tablas formateadas, etc.)
- Rastreo y diagnostico del programa integrado
- Validación de la entrada de usuario integrada
- Un mecanismo de autentificación fácil de utilizar
- Integración sólida con ADO.NET (base de datos predecesora de .NET)
- Excelente soporte para Web Services
- Elimina la dependencia de Component Object Model
- Segmentación extensible con muchos lugares en los que se puede interceptar una petición

Con la aparición de ASP.NET 2.0 que se basaba en las versiones 1.0 y 1.1 proporcionando algunas características nuevas además de las ya existentes, algunas de las importantes eran:

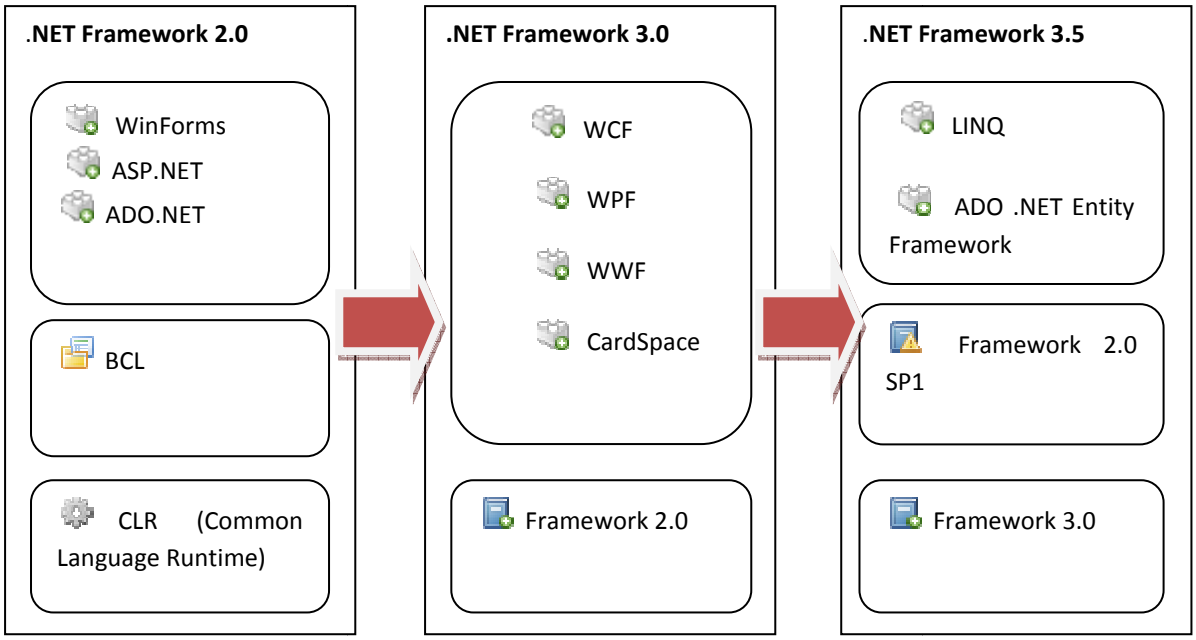
- Plantillas y diseños

- Enlaces de datos actual
- Soporte para navegación y mapa del servidor
- Habilita el patrón de modelo
- Nuevas funciones de caché
- Controles web
- Controles de personalización
- Configuración programable
- Herramientas de administración
- Nuevo modelo de compilación

Con todas estas nuevas características ASP.NET 2.0 se hacia una plataforma para crear sitios Web muy potente.

Con la aparición de ASP.NET 3.5 se introducción soporte para AJAX, soporte para Windows Communication Fundation (WCF) y un gran avance en el tratamiento de las bases de datos relacionales con LINQ (en el capítulo 4 se describen las novedades en el lenguaje) y el ADO.NET Entity Framework, además de un incremento significativo de soporte de ASP.NET dentro de Visual Studio.

En las siguientes imágenes se aprecia de forma grafica las características que se han ido añadiendo en los diferentes .NET Frameworks y que representan mejoras a la vez en ASP.NET:



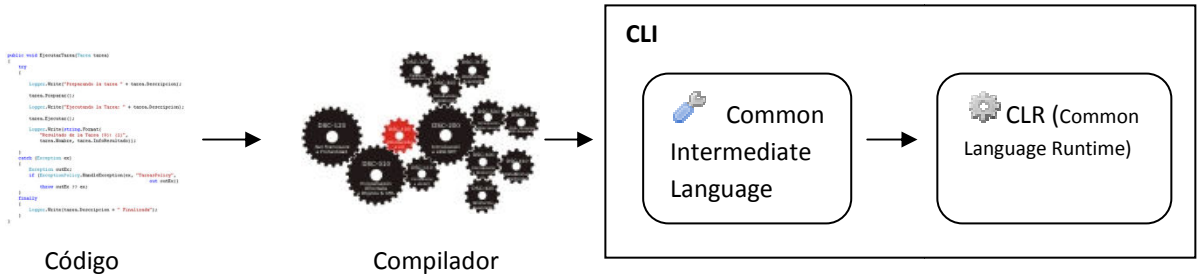
3.2.5.3. C# 3.0 Y VB.NET 9.0

C# es un lenguaje de programación orientado a objetos desarrollado por Microsoft para convertirse en parte clave de su plataforma de desarrollo de software .NET. Al ser orientado a objetos, C# se compone de una colección de clases que pueden interactuar entre sí. [6]

Este lenguaje se basa en el lenguaje C++, pero no hay duda que está influido por otro lenguaje de Microsoft, Visual Basic. Una de las principales ventajas de C# es que su sintaxis es similar a la de otros lenguajes de programación populares, principalmente C++, Visual Basic, Java y Delphi.

VB.NET al igual que C# es otro lenguaje desarrollado por Microsoft, pero con muchos más años de experiencia y con muchas más versiones. [7]

Respecto a cómo funciona Microsoft .NET Framework por dentro, es decir su compilador se explicará brevemente mediante la figura de a continuación:



Si compilamos la aplicación con VB.NET 9.0 lo hara el compilador de VB.NET 9.0 e igual si se hace con C# 3.0, lo novedoso es que el compilador de cada lenguaje compilara el código a un lenguaje intermedio denominado CIL (*Common Intermediate Language*). Por otro lado ese lenguaje intermedio será ejecutado por el CLR, encargado de compilar la última instancia.

Tanto C# como VB.NET están diseñados para aprovechar el CLR (Common Language Runtime) sobre el cual se basan todos los programas de .NET. Todas las aplicaciones escritas en C# y VB.NET requieren CLR para ejecutarse.

Algunas características del CLR:

- Código gestionado: código gestionado por aplicaciones Visual Studio y ejecutado por la plataforma .NET Framework
- Instalación fácil/automática de la aplicación: Esto se puede realizar utilizando la caché de ensamblados global.
- Gestión de memoria: CLR ofrece a los programadores una forma sencilla aunque eficaz de gestionar la memoria.
- Recolección automática de basura: La plataforma .NET libera automáticamente memoria cuando los objetos dejan de ser necesarios.
- Niveles excelentes de seguridad durante la ejecución: La plataforma .NET Framework incluye un modelo integrado de seguridad que proporciona permiso a recursos basándose en la evidencia encontrada en ensamblajes.

Lo que se ha explicado da entender la utilización en Agromutua.Web de proyectos en ambos lenguajes y su convivencia sin ningún tipo de problema.

4. LINQ

Podemos decir que el software es sencillo, ya que se reduce en dos cosas: código y datos. Eso sí, escribir software ya no es tan sencillo y una de las actividades más complicadas y que más veces se repite es escribir código que trate con datos.

Existen gran cantidad de lenguajes de programación y todos ellos incluyen el tratamiento de los datos ya sean ficheros de datos, XMLs o tablas de una base de datos.

Partiendo de la idea del tratamiento de los datos se explicará en este apartado de la tesina que se espera de las novedades del Framework 3.5 de .NET en lo que a datos y en concreto a LINQ se refiere, y se verá cómo se consigue una gran integración entre lenguaje y datos con LINQ to Objects, LINQ to SQL y LINQ to XML.

4.1 ¿QUÉ ES LINQ?

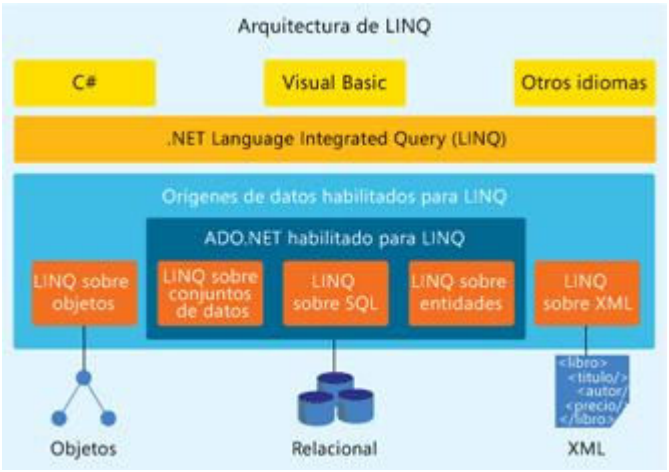
Cuando se escribe una aplicación en .NET llega un momento en que necesitamos hacer persistir objetos a una base de datos, hacer una consulta en la base de datos y cargar los resultados en objetos. El problema en muchos casos es el vacío que existe entre el lenguaje de programación y la base de datos.

Tras muchos intentos de proporcionar bases de datos orientadas a objetos que estarían más cerca de plataformas orientadas a objetos y lenguajes de programación como C# o VB.NET, aún siguen siendo dominantes las bases de datos relacionales y tienen que enfrentarse a accesos a datos y persistencia en todos sus programas.

La motivación de Microsoft a la hora de crear LINQ fue abordar las dificultades conceptuales y técnicas encontradas cuando se utilizan bases de datos como lenguajes de programación .NET y de esta manera LINQ proporciona una solución para el problema del mapeado relacional de objetos así como simplificar la interacción entre objetos y fuentes de datos.

LINQ con el tiempo evoluciono y se convirtió en un conjunto de herramientas de consulta integradas en el lenguaje de aplicación general. Este conjunto de herramientas se pueden utilizar para acceder a datos que están en memoria (LINQ to Objects), bases de datos (LINQ to SQL) , documentos XML (LINQ to XML) , un sistema de archivo o cualquier otra fuente.

Por tanto LINQ unifica el acceso a datos, cualquiera que sea la fuente de datos y permite mezclar diferentes tipos de fuentes y realizar consultas de forma similar a estas fuentes. LINQ integra consultas directamente en lenguajes .NET como C# o VB.NET a través de un conjunto de extensiones a estos lenguajes: LINQ significa *Language INtegrated Query*, Consulta integrada en los lenguajes.



Esto supone una gran ventaja para los desarrolladores ya que antes que existiera LINQ tenían que mezclar diferentes lenguajes como SQL, XML o XPath junto con varias tecnologías y APIs como ADO.NET o System.Xml en cada aplicación.

LINQ tiene unos aspectos claves que como la sintaxis que se utiliza de consulta permite acceder a cualquier tipo de dato y además vamos a trabajar en un mundo altamente tipado. Por tanto esta tecnología es un paso hacia un modelo de programación más declarativo.

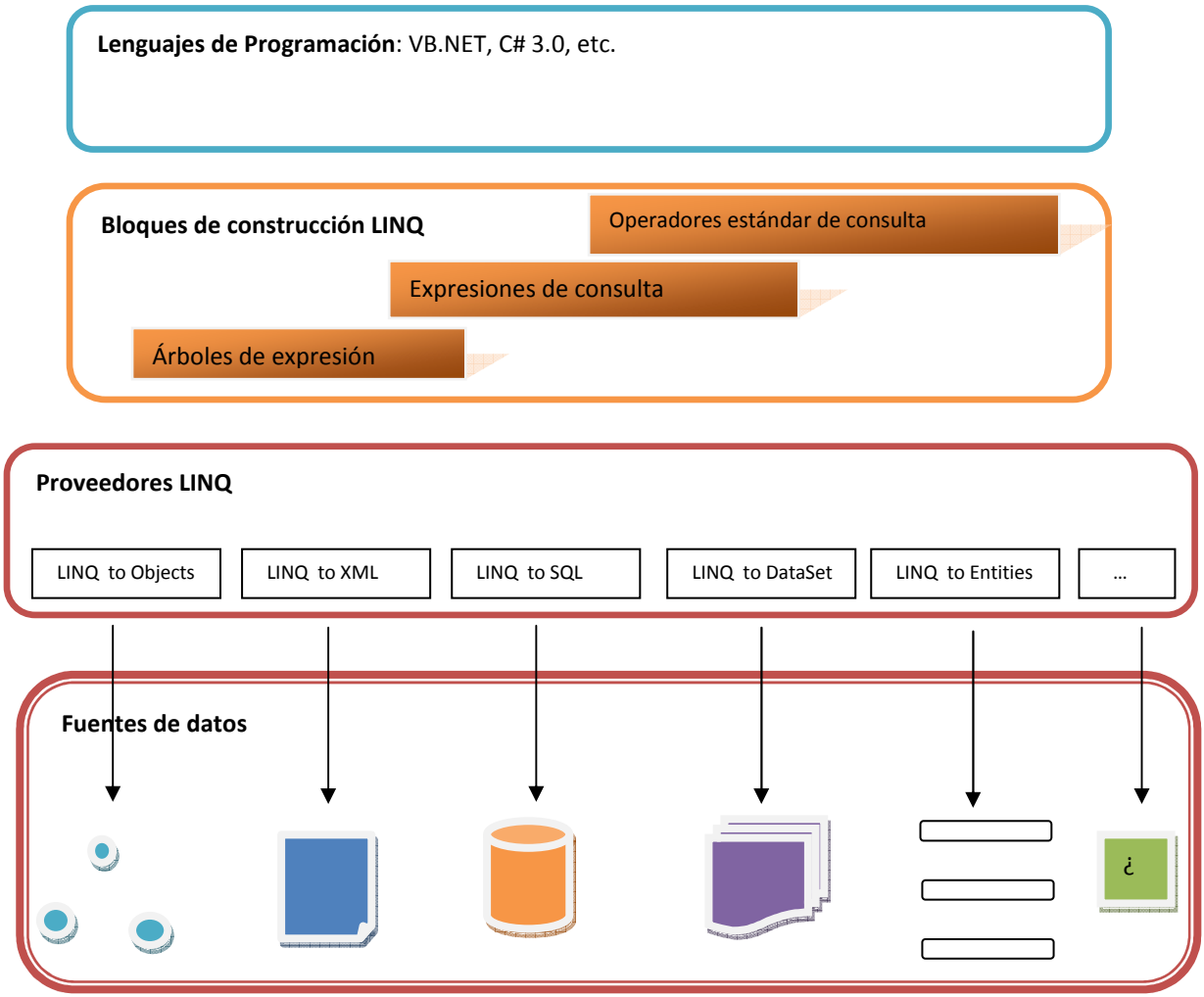
4.1.1 LINQ COMO UN CONJUNTO DE HERRAMIENTAS Y EXTENSIONES DE LENGUAJE

Los tres proveedores principales de datos de LINQ y que explicaremos con más detenimiento más delante de la tesina (LINQ to Objects, LINQ to SQL y LINQ to XML) no son los únicos y LINQ está abierto a nuevas fuentes de datos.

LINQ está formado por una base común de operadores de consulta, expresiones de consulta y árboles de expresión que permiten que el uso de herramientas LINQ sea totalmente extensible.

Es así que se pueden crear otras variantes de LINQ para proporcionar diversas formas de fuentes de datos. Y los vendedores de software lo están aprovechando y están lanzando implementaciones de LINQ (ejemplo LINQ to Amazon). Es el propio Microsoft el que también ha lanzado LINQ to Dataset y LINQ to Entities para dar soporte al nuevo ADO.NET Entity.

La figura muestra cómo podemos representar los bloques de construcción LINQ y el conjunto de herramientas.



Los proveedores de LINQ no son herramientas independientes, esto es posible ya que el marco de trabajo de LINQ se presenta como un conjunto de extensiones de lenguaje.

El conjunto de extensiones de lenguaje que vienen con LINQ permiten formular consultas sobre varios tipos de almacenamiento de datos en los lenguajes de programación, con esto se gana en integración.

La idea principal de LINQ es por tanto, que con C# por ejemplo nos va permitir acceder a datos y no será necesario para el programador conocer otro lenguaje como SQL o otras API para realizar peticiones de datos.

4.1.2 Orígenes y objetivos del proyecto LINQ

Es importante saber las metas que Microsoft tenía cuando se propuso conseguir el proyecto LINQ. A continuación se revisan los objetivos uno por uno y su motivación. También se trataran las relaciones que tiene LINQ con otros proyectos Microsoft como Cω, ObjectSpaces, WinFS o el soporte para XQuery en el marco de trabajo .NET.

4.1.2.1 LOS OBJETIVOS DEL PROYECTO LINQ

La tabla que se muestra a continuación revisa los objetivos que estableció Microsoft para el proyecto LINQ para proporcionar una idea clara de lo que ofrece esta tecnología.

Objetivo	**Motivación**
Integrar objetos, datos relacionales y XML	Sintaxis de consulta unificada entre fuentes de datos para evitar diferentes lenguajes para diferentes fuentes de datos. Modelo sencillo para procesar todos los tipos de datos con independencia de la fuente o de la representación en memoria.
Potencia como la de SQL y XQuery en C# y VB	Integrar posibilidades de consulta en el propio lenguaje de programación.
Modelo extensible para lenguajes	Permitir implementación para otros lenguajes de programación.
Modelo extensible para múltiples fuentes de datos	Ser capaz de acceder a otras fuentes de datos además de las bases de datos relacionales y XML. Acceder a que otros marcos de trabajo permitan el soporte LINQ para sus propias necesidades.
Verificación de tipos seguros	Comprobación del tipo en tiempo de compilación para evitar problemas que se habían descubierto previamente sólo en tiempo de ejecución. El compilador capturará los errores en sus consultas.
Soporte ampliado para IntelliSense	Ayudar a los desarrolladores cuando escriben consultas para mejorar su productividad y ayudarles a trabajar con la nueva sintaxis. El editor le guiará cuando escribe consultas.
Soporte del depurador	Permitir que los desarrolladores depuren consultas LINQ paso a paso y con información de depuración.
Construido sobre la base de C# 1.0 y 2.0, VB .NET 7.0 y 8.0	Reutilizar las características que se han implementado en las versiones anteriores de los lenguajes.
Ejecutar sobre .NET 2.0 CLR	Evitar que se requiera un nuevo tiempo de ejecución y crear innecesarios problemas de desarrollo.
Mantener compatibilidad inversa 100%	Ser capaz de utilizar las colecciones estándar y genéricas, vinculación de datos, controles de formularios Windows, etc.

La característica más importante de LINQ es sin duda la posibilidad de tratar con varios tipos de datos y fuentes, presentada en la tabla. Otra característica esencial de LINQ es que está fuertemente tipado, que significa:

- Comprobación en tiempo de compilación para todas las consultas.
- IntelliSense con Visual Studio cuando escribimos consultas

NOTA

Microsoft IntelliSense es la aplicación de autocompletar, mejor conocido por su utilización en Microsoft Visual Studio entorno de desarrollo integrado. Además de completar el símbolo de los nombres que el programador está escribiendo, IntelliSense sirve como documentación y desambiguación de los nombres de variables, funciones y métodos de utilización de metadatos basados en la reflexión.

4.1.2.2 UN POCO DE HISTORIA

LINQ es el resultado de un largo proceso de investigación por parte de Microsoft. Varios son los proyectos que han evolucionado sus lenguajes de programación y métodos de acceso a datos y que han derivado en LINQ to Objects, LINQ to XML (antes conocido como XLinq) y Linq to SQL (antes conocido como DLinq)

Cω

Cω fue un proyecto del Microsoft Research que ampliaba el lenguaje C# básicamente en:

- Una extensión de flujo de control para concurrencia asíncrona de área ancha (antes conocido como Polyphonic C#)
- Una extensión de tipo de datos para XML y manipulación de base de datos (antes conocido como Xen y como X#)

Cω se concibió para experimentar consultas integradas, mezclando C# y otros lenguajes como SQL o XQuery. La experiencia tras el lanzamiento de Cω y tratamiento de los datos supuso una serie de extensiones de Cω a .NET y el lenguaje C# y sus consultas al estilo SQL y el contenido XML fueron los primeros pasos. Cω incorporo el tipo de dato análogo en el Framework 2.0 `System.Collections.Generic.IEnumerable<T>`, y entre otras cosas definió constructores para registros tipados que son similares a los tipos anónimos que tenemos en C# 3.0 y VB .NET 9.0

OBJECTSPACES

ObjectSpaces fue el primer intento de Microsoft en el mapeado relacional de objeto, se trataba de una API de acceso a datos que permitía que los datos fueran tratados como objetos, independientemente del almacén de datos.

En 2004 ObjectSpaces dependía de WinFS (proyecto para un sistema de archivo relacional para Windows cancelado en 2006) Una vez que WinFS no se convertiría en la primera versión de Windows Vista el mundo se dio cuenta que ObjectSpaces no vería la luz.

XQUERY

De forma similar a ObjectSpaces y más o menos al mismo tiempo Microsoft comenzó a trabajar en el procesador XQuery. Se incluyo una beta en el Framework 2.0 pero al final no se decido pasar al lado del cliente en la versión final, solo SQLServer 2005 incluye en el cliente XQuery y ahora que se ha terminado el estándar XQuery se está considerando incluirlo en .NET.

El problema principal con XQuery es que se trata de un lenguaje adicional que tendríamos que aprender para tratar con XML

Todos estos proyectos fallidos, aparcados o de pruebas tuvieron respuesta en el PDC de 2005 cuando se anuncio en proyecto LINQ. LINQ fue diseñado por Anders Heljsberg y otros de Microsoft para abordar el desajuste de impedancia entre lenguajes de programación como C# y VB .NET. Los proyectos anteriores solo sirvieron para beneficiarse de los trabajos de investigación y así conseguir hacer consultas sobre cualquier cosa.

4.2 EVOLUCIÓN EN LOS LENGUAJES VB.NET Y C#

En este apartado se verá las nuevas extensiones de lenguaje que traen de la mano C# 3.0 y VB.NET 9.0. Estas características son principios básicos de LINQ y es importante que antes de usar LINQ el programador las conozca, pero son parte integral de los lenguajes C# y VB.NET y se pueden utilizar por separado.

Las principales extensiones de lenguaje son:

- Variables locales implícitamente tipadas
- Inicializadores de objeto y colección
- Expresiones lambda
- Métodos de extensión
- Tipos anónimos

Otras extensiones de lenguaje, que son importantes tenerlas en cuenta, pero no necesarias para trabajar con LINQ son por ejemplo las propiedades auto implementadas exclusiva para C#.

VB.NET 9.0 presenta características de lenguaje, pero no están relacionadas con LINQ como el operador `If` como ternario estilo al `? :` de C# y como sustituto del `IIf`. Otras mejoras incluyen delegados relajados e inferencia mejorada de tipos genéricos. Estas y otras no se entrarán a explicar por no estar relacionadas con LINQ.

En los siguientes puntos se explicarán las extensiones principales y el uso que hace LINQ de ellas y que permite que las consultas se mezclen en los lenguajes de programación.

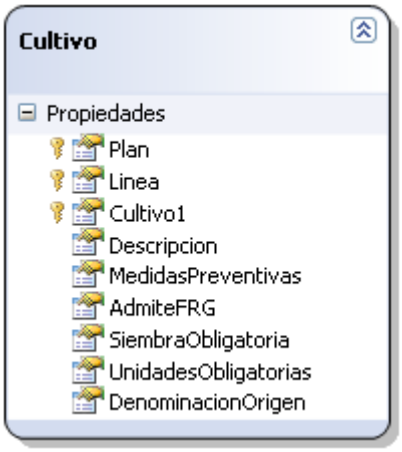
4.2.1 EL EJEMPLO DE GUÍA

Como se ha indicado en el punto anterior las nuevas extensiones del lenguaje van a suponer un cambio a la hora de programar.

Para demostrar y explicar con más claridad las principales extensiones de lenguaje, las cuales serán detalladas en los siguientes puntos, tendremos un ejemplo que servirá para entender y observarlas.

Aprovechando el tema de la tesina y basándonos en que la realización de esta ha servido para explicar LINQ, el ejemplo de guía que se ha decido utilizar tendrá que ver con el negocio de Agromutua (ver punto 1 de la tesina).

Dentro de todas las tablas del proyecto se ha seleccionado la de *Condicionado.Cultivo* por la sencillez de esta para los ejemplos, filtrar, ordenar y ver demostraciones.



Como se explica en el apartado 1, todas las pólizas que realiza Agromutua referentes a agricultura tienen un cultivo. Cuando incluimos dicho cultivo vendrá marcado por 3 campos claves, que traducidos a la base de datos serán la clave primaria de la tabla.

Estos campos serán:

- *Plan*: indica el año
- *Línea*: engloban a una serie de cultivos (ej. Línea 11 es de Tomate, y puede ser Tomate Fresco, Tomate Pelado o Tomate Concentrado)
- *Cultivo*: el identificador del cultivo (Tomate Fresco es 1, Tomate Pelado es 2 y Tomate Concentrado es 3)

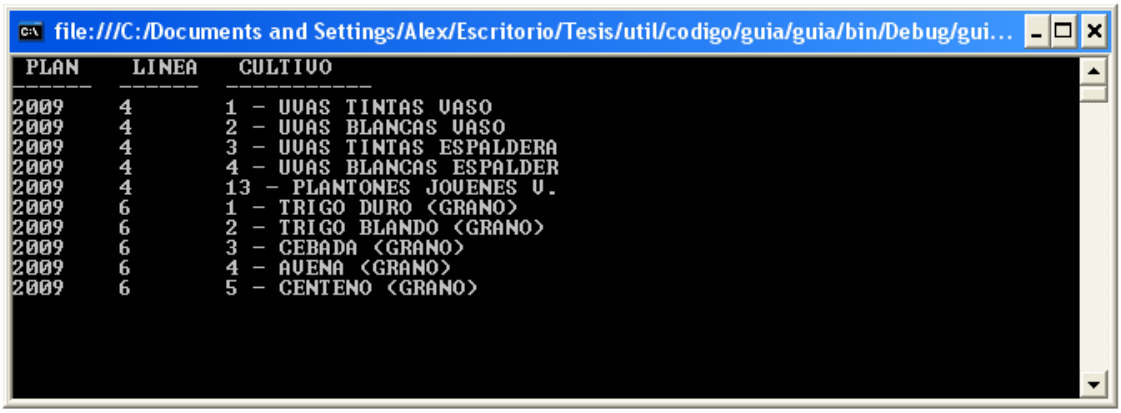
El campo *Descripción* es el que pone nombre al Cultivo y nos servirá como referente a los nombres de los cultivos. Con el resto de campos no trabajaremos en los ejemplos por lo que podemos obviarlos.

Veamos un ejemplo:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace guia
{
    class Program
    {
        static void Main(string[] args)
        {
            DataClassesDataContext bd = new DataClassesDataContext();

            var cultivos = from x in bd.Cultivo
                    where x.Plan == 2009
                    select x;


            Console.WriteLine(" PLAN \t LINEA \t CULTIVO");
            Console.WriteLine("------\t------\t----------");
            foreach (var item in cultivos.Take(10))
            {
                Console.WriteLine( item.Plan + "\t" + item.Linea + "\t" +item.Cultivo1 + " - " + item.Descripcion);
            }
            Console.Read();
        }
    }
```

En el siguiente ejemplo estamos utilizando LINQ to SQL. Se ha mapeado la tabla y se ha creado el *DataContext* (ver punto 2.5, Linq to SQL) y después hemos realizado una consulta LINQ para obtener los resultados cuyo *plan* sea igual al 2009. Como la tabla tiene más de 150 registros en el *foreach* se ha incluido un *take(10)* para obtener los primeros 10 resultados de la consulta.

El resultado es el siguiente:



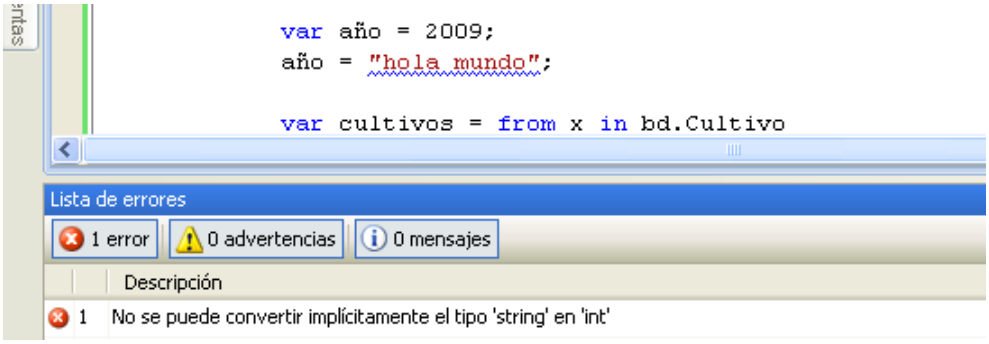
Como hemos visto ya tenemos un ejemplo de guía para nuevas demostraciones en las extensiones del lenguaje.

4.2.2 VARIABLES LOCALES IMPLÍCITAMENTE TIPADAS

C# 3.0 ofrece una característica que permite declarar una variable local sin tener que especificar su tipo, para ello se utiliza la palabra clave *var* (*Dim* en VB.NET).

```
var año = 2009;

var cultivos = from x in bd.Cultivo
        where x.Plan == año
        select x;
```

Fijemos en el ejemplo. Si queremos que el año, que tiene que ser *Integer* sea 2009, lo podemos declarar así, de tal forma que implícitamente se tipa la variable *año*, esto sucede ya que el compilador infiere el tipo de esta variable desde la expresión utilizada para inicializarla.



En la parte superior se demuestra que el compilador se va a quejar cuando una variable implícitamente tipada a *Integer* la pasamos a *String*.

Podemos observar que la sintaxis que utilizamos para declarar las variables locales implícitamente tipadas es la misma que cuando declaramos una variable tipada de la forma tradicional.

Vemos un fragmento de código equivalente:

```
var año = 2009;
var descripción = "Tomate Pelado";

int año = 2009;
string descripción = "Tomate Pelado";
```

En el ejemplo que utilizábamos al principio (punto 2.2.1) vemos como utilizamos las variables implícitamente tipadas en la descripción del año, pero también para realizar la consulta LINQ, que implícitamente se tipa a `IQueryable` (veremos esta colección más adelante) así como en el bucle *foreach* donde es importante tenerlo en cuenta ya que los elemento que recorre no sabe de qué tipo de datos se trata.

Por tanto la posibilidad de trabajar con las palabras clave `var` y `Dim` pueden ayudarnos a escribir fragmentos de código más cortos y poder reutilizarlo, en muchas ocasiones requieren que las acompañemos de otras características de LINQ.

4.2.3 INICIALIZADORES DE OBJETO Y COLECCIÓN

Estas características nos servirán de gran utilidad sobre todo cuando empezamos a escribir expresiones de consulta.

INICIALIZADORES DE OBJETO

Los inicializadores de objeto nos permiten especificar valores para uno o más campos o propiedades de un objeto en una sentencia. Permiten inicializaciones declarativas para todo los tipos de objetos.

Hasta ahora, hemos podido inicializar objetos de primitivas como sigue:

```
int i = 2009;
string str = "abcd";
string[] names = new string[] {"tesina","Alex","Medina"};
```

Pero el problema se daba cuando teníamos que inicializar objetos, donde teníamos que utilizar código parecido a este:

```
Cultivo nuevo = new Cultivo();
nuevo.Linea = 50;
nuevo.Plan = 2009;
nuevo.Cultivo1 = 1;
nuevo.Descripcion = "Avellana";
```

Con C# 3.0 y VB.NET podemos inicializar todos los objetos utilizando un enfoque inicializador (ejemplo en C#):

```
var nuevo = new Cultivo { Linea = 50, Plan = 2009, Cultivo1 = 1, Descripcion = "Avellana" };
```

Las piezas de código con y sin inicializadores de objeto generan el mismo código IL. Por tanto los inicializadores de objeto simplemente ofrecen un método abreviado.

INICIALIZADORES DE COLECCIÓN

Otros inicializadores que se han añadido son los inicializadores de colección. Esta nueva sintaxis nos permite inicializar diferentes tipos de colecciones, con tal que implementen `System.Collections.IEnumerable` y proporcionan métodos adecuados *Add*.

Un ejemplo:

```
var digits = new List<int> { 0, 1, 2, 3, 4 };
```

Estos inicializadores son particularmente útiles cuando se utilizan en la misma pieza de código. Veamos unos ejemplos, el primero con inicializadores:

```
var listaCultivo = new List<Cultivo> {
    new Cultivo{Linea = 50, Plan = 2009, Cultivo1 = 1, Descripcion = "Avellana" },
    new Cultivo{Linea = 49, Plan = 2009, Cultivo1 = 1, Descripcion = "Zanahoria" }
};
```

El segundo, mucho más largo y equivalente pero sin inicializadores:

```
var listaCultivo = new List<Cultivo>();
Cultivo tmp = new Cultivo();
tmp.Linea = 50;
tmp.Plan = 2009;
tmp.Cultivo1 = 1;
tmp.Descripcion = "Avellana";
listaCultivo.Add(tmp);
tmp = new Cultivo();
tmp.Linea = 49;
tmp.Plan = 2009;
tmp.Cultivo1 = 1;
tmp.Descripcion = "Zanahoria";
listaCultivo.Add(tmp);
```

Por tanto nos van a permitir ahorrarnos mucho código. Hay que tener en cuenta que podemos inicializar colecciones de clases que tengan implementada el método *Add* de la clase `IEnumerable`.

Resumiendo, la utilización de inicializadores de objeto y colección nos permiten obtener diversas ventajas:

- Podemos inicializar un objeto dentro de una sola instrucción
- No necesitamos proporcionar un constructor para poder inicializar objetos sencillos
- No necesitamos varios constructores para inicializar diferentes propiedades de objetos

4.2.4 EXPRESIONES LAMBDA

Las expresiones lambda son unas de las nuevas características del lenguaje más útiles que están disponibles para LINQ. Basándose como su nombre indica en el cálculo lambda, pueden verse como un paso hacia los lenguajes funcionales futuros como F#, aunque ya algunos lenguajes funcionales como Lisp utilizan notaciones lambda.

Nota

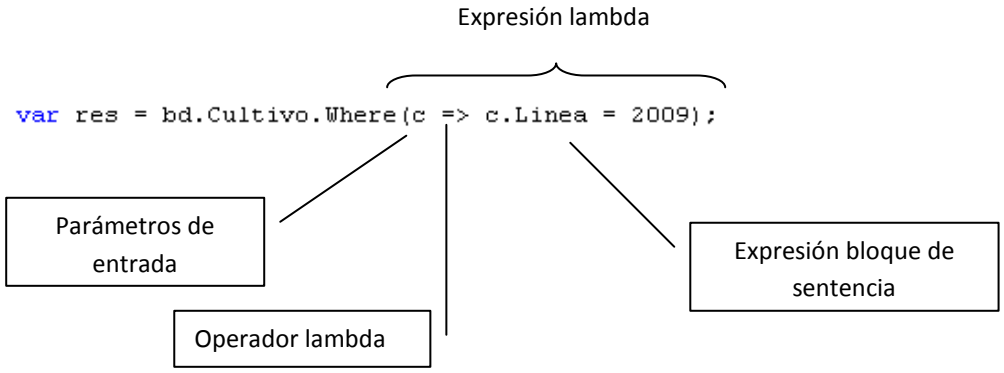
El cálculo lambda es un sistema formal diseñado para investigar la definición de función, la noción de aplicación de funciones y la recursión. Fue introducido por Alonzo Church y Stephen Kleene en la década de 1930; Church usó el cálculo lambda en 1936 para resolver el Entscheidungsproblem. Puede ser usado para definir de manera limpia y precisa qué es una "función computable". El cálculo lambda tiene una gran influencia sobre los lenguajes funcionales, como Lisp, ML y Haskell.

En LINQ utilizaremos las expresiones lambda principalmente para realizar acciones de filtrado sobre colecciones.

La utilización de delegados nos permitía pasar un método como un parámetro a otro y conseguir un filtro, esta características ya estaba disponible en C# 1.0, pero se mejoro en la C# 2.0 para permitir trabajar con delegados mediante métodos anónimos que nos permiten utilizar código más corto y evitar la necesidad de escribir explícitamente los métodos nombrados.

SINTAXIS DE LAS EXPRESIONES LAMBDA

En C#, una expresión lambda se escribe como una lista de parámetros, seguida por un =>, seguida por una expresión o un bloque de sentencia, tal y como se muestra en el ejemplo:



El operador lambda se puede leer como “va a”. El lado izquierdo del operador especifica los parámetros de entrada y la parte derecha contiene la expresión o bloque de sentencia a evaluar.

A modo de apunte, en el ejemplo anterior utilizamos una expresión lambda para incluirlo dentro de un *Where* de LINQ to SQL para obtener los cultivos con línea igual a 2009. Si nos fijamos bien el bloque es equivalente al que incluíamos en los ejemplos anteriores:

```
var cultivos = from x in bd.Cultivo
               where x.Plan == 2009
               select x;
```

Existen dos tipos de expresión lambda:

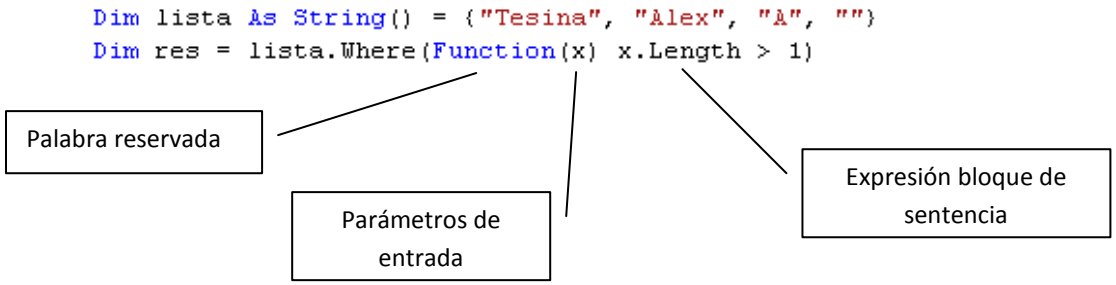
- Lambda expresión: expresión al lado derecho
- Lambda sentencia: igual que la lambda expresión pero en su parte derecha costa de cualquier número de sentencias englobadas en llaves.

`x => x + 1`	Cuerpo de expresión, implícitamente tipado
`x => { return x + 1; }`	Cuerpo de sentencia, implícitamente tipado
`(int  x) => x + 1`	Cuerpo de expresión, explícitamente tipado
`(int x) => { return x + 1; }`	Cuerpo de sentencia, explícitamente tipado
`( x, y ) => x * y`	Múltiples parámetros
`() => 1`	Cuerpo de expresión, no parámetros
`() => Console.WriteLine()`	Cuerpo de sentencia, no parámetros

Nota

Los parámetros de una expresión lambda pueden encontrarse explícitamente o implícitamente tipados.

En VB.NET la sintaxis es diferente que en C#, el principal cambio es que comienzan por la palabra clave *Function*:



En VB.NET 9.0 no soporta sentencias lambdas.

4.2.5 MÉTODOS DE EXTENSIÓN

Siguiendo nuestro recorrido por las nuevas características de lenguaje, vamos a ver los métodos de extensión. Los métodos de extensión no van a permitir añadir métodos a un tipo después de haberlo definido.

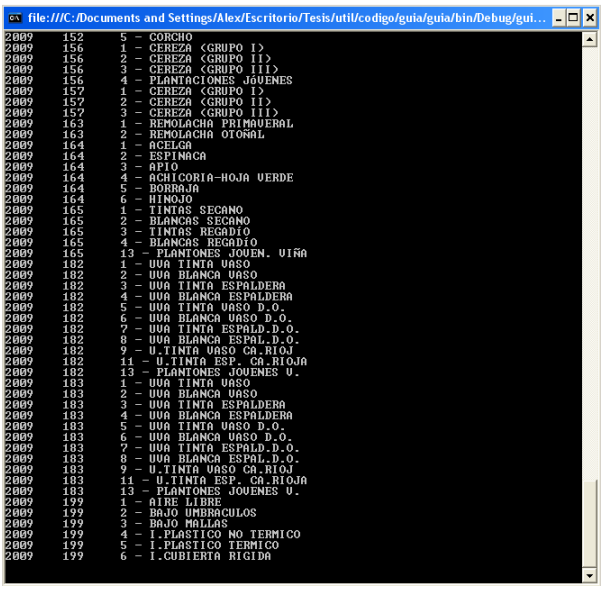
4.2.5.1 CREAR UN MÉTODO DE EXTENSIÓN DE EJEMPLO

Para entender un poco mejor cómo funcionan los métodos de extensión se creará un ejemplo y después veremos algunos de los métodos de extensión ya definidos.

Imaginemos que en nuestro ejemplo queremos tener un método estático que nos imprima los valores del objeto Cultivo que viene en una colección después de hacer una consulta en LINQ to SQL.

```
class Program
{
    static void Main(string[] args)
    {
        DataClassesDataContext bd = new DataClassesDataContext();

        var cultivos = bd.Cultivo.Where(c => c.Plan == 2009);

        Console.WriteLine(" PLAN \t LINEA \t CULTIVO");
        Console.WriteLine("------\t------\t-----------");
        Console.WriteLine(imprimeCultivoStatic( cultivos));

        Console.Read();
    }


    static string imprimeCultivoStatic(IEnumerable<Cultivo> cultivos)
    {
        string res = "";
        foreach (Cultivo item in cultivos)
        {
            res += item.Plan + "\t" + item.Linea + "\t" + item.Cultivo1 + " - " + item.Descripcion + "\n";
        }
        return res;
    }

}
```

Con el código que vemos arriba se obtendrían los valores de la consulta y el método estático se encargaría de realizar de hacer el bucle para devolver el valor y imprimirse en pantalla:



Pero el código del ejemplo se podría mejorar transformado el método estático en un método de extensión. Con esta nueva característica de lenguaje podemos tratar con tipos existentes como si se hubieran ampliado con métodos adicionales.

DECLARAR MÉTODOS DE EXTENSIÓN EN C#

Para transformar un método estático en un método de extensión en C#, todo lo que tenemos que hacer es incluir la palabra reservada *this* en el primer parámetro del método.

```
class Program
{
    static void Main(string[] args)
    {
        DataClassesDataContext bd = new DataClassesDataContext();

        var cultivos = bd.Cultivo.Where(c => c.Plan == 2009);

        Console.WriteLine(" PLAN \t LINEA \t CULTIVO");
        Console.WriteLine("------\t------\t-----------");
        Console.WriteLine(cultivos.imprimeCultivo());

        Console.Read();
    }

}

public static class ClaseEstatica
{
    public static string imprimeCultivo(this IEnumerable<Cultivo> cultivos)
    {
        string res = "";
        foreach (Cultivo item in cultivos)
        {
            res += item.Plan + "\t" + item.Linea + "\t" + item.Cultivo1 + " - " + item.Descripcion + "\n";
        }
        return res;
    }
}
```

Si examinamos la nueva versión del método solo tenemos que tener en cuenta que hemos incluido la palabra *this*. La palabra clave *this* le dice al compilador que trate el método como un método de expresión, y por tanto le indica que este es un método que amplia del tipo `IEnumerable<Cultivo>`.

Resumiendo, para declara los métodos de extensión:

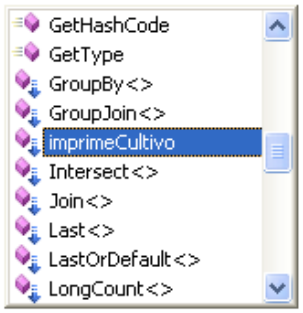
- Se deben declarar en una clase no genérica estática
- El método puede tomar cualquier número de argumentos pero el primer parámetro debe ser del tipo que se amplía
- Debe estar precedido el primer parámetro por la palabra clave *this*

Con todo esto se ha conseguido ejecutar el ejemplo como si fuera un método de la instancia `IEnumerable<Cultivo>`:

```
Console.WriteLine(cultivos.imprimeCultivo());
```

Fijémonos que además nuestro método sale en la lista de métodos del IntelliSense para los tipos soportados por este método, y las extensiones de métodos con un icono específico.

```
Console.WriteLine(cultivos.);
```



Esta característica nos va a resultar clave para escribir consultas en LINQ.

DECLARAR MÉTODOS DE EXTENSIÓN EN VB.NET

En VB.NET los métodos de extensión son métodos compartidos decorados con un atributo personalizado (`System.Runtime.CompilerServices.ExtensionAttribute`) que permite invocarles con sintaxis de método de instancia (un método de extensión puede ser un procedimiento *Sub* o un procedimiento *Function*.) Este atributo lo proporciona el nuevo ensamblador `System.Core.dll`.

Definiremos siempre los métodos de extensión en VB.NET en un módulo aparte como muestra el ejemplo.

```
Module claseExtension

    <System.Runtime.CompilerServices.Extension()> _
    Public Function imprimeCultivo(ByVal cultivos As IEnumerable(Of guia.Cultivo)) As String

        Dim res As String = ""

        For Each item In cultivos
            res += item.Plan & "\t" & item.Linea & "\t" & item.Cultivo1 & " - " & item.Descripcion & "\n"
        Next

        Return res

    End Function

End Module
```

El primer parámetro especifica qué tipo de datos amplía el método. Cuando se ejecuta el método, el primer parámetro se vincula a la instancia del tipo de datos contra la que se aplica el método.

4.2.6 TIPOS ANÓNIMOS

La última de las características que se van a presentar son los tipos anónimos, que utilizaran una sintaxis similar a los inicializadores de objeto. La idea de utilizar los tipos anónimos es la de almacenar agrupaciones de datos en un objeto sin declarar previamente la clase.

```
var res = new {Cultivos = cultivos,Año = "2009");
```

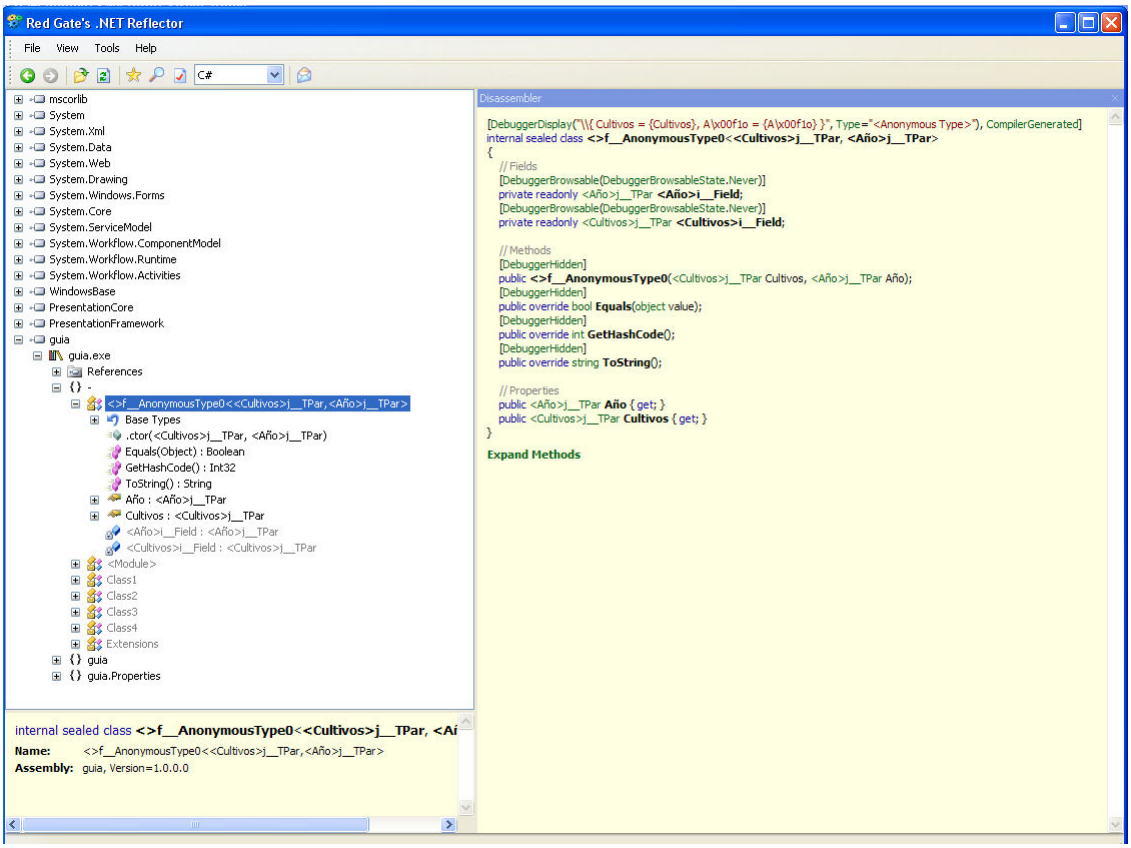
Los objetos declarados que utilizan un tipo anónimo solo puede ser declarados con las palabras reservadas `Dim` o `var`. Esto es porque un tipo anónimo no tiene un nombre que podemos utilizar en nuestro código.

Los tipos anónimos son tipos sin nombres, pero son tipos de cualquier forma. Esto significa que un tipo real se crea por el compilador.

Por tanto si en nuestro código de ejemplo incluimos un tipo anónimo:

```
class Program
{
    static void Main(string[] args)
    {
        DataClassesDataContext bd = new DataClassesDataContext();

        var cultivos = bd.Cultivo.Where(c => c.Plan == 2009);

        Console.WriteLine(" PLAN \t LINEA \t CULTIVO");
        Console.WriteLine("------\t------\t-----------");
        Console.WriteLine(cultivos.imprimeCultivo());
        var res = new {Cultivos = cultivos,Año = "2009");
        Console.Read();
    }
```

En la siguiente imagen podemos observar el ensamblado que se ha creado en nuestro programa al incluir el tipo anónimo producido por el compilador.

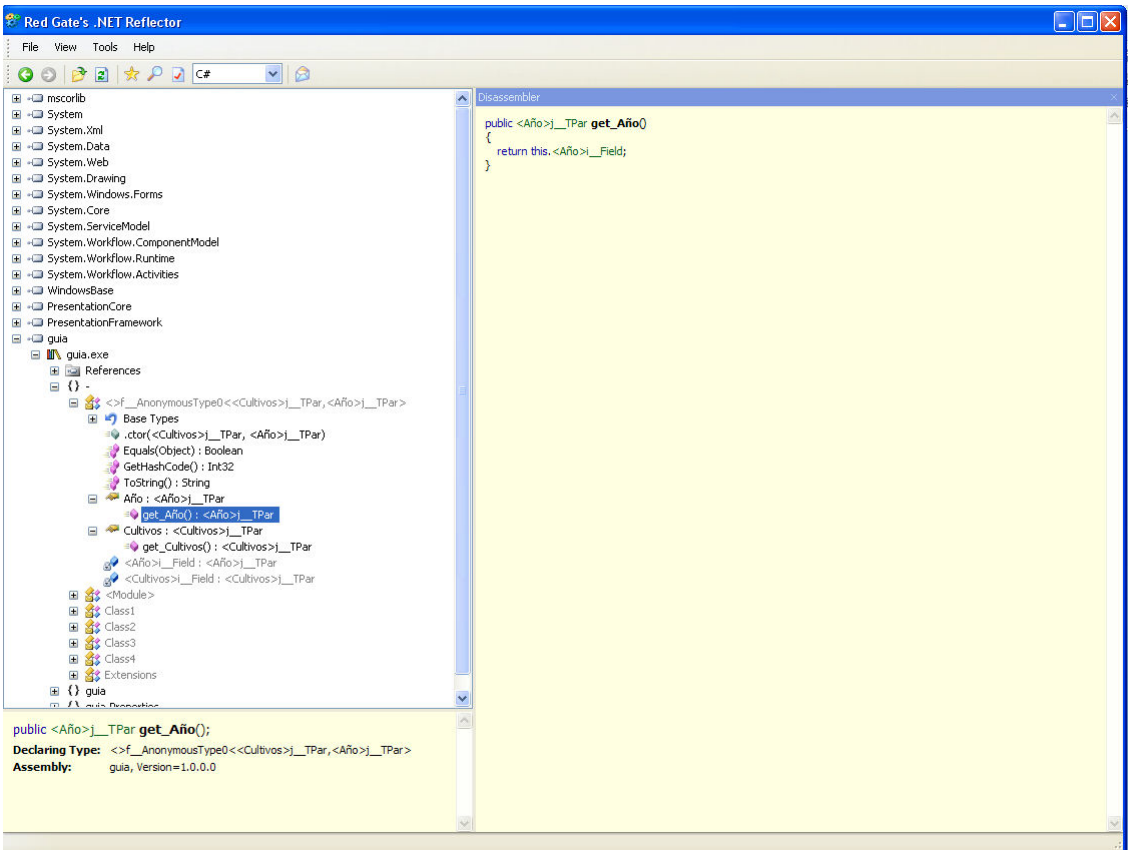


Esta captura esta realizada con el .NET Reflector (http://www.red-gate.com/products/reflector/) un programa libre que nos permite observar el código compilado.

Algunos problemas que podemos encontrar con los tipos anónimos pueden ser que estos objetos realmente su utilidad es temporal, es decir, que se utilizan para un espacio de tiempo limitado. El problema viene cuando queremos trabajar con ellos y deseamos pasarlos a métodos que entonces pueden surgir problemas con el tipo. El tipo del parámetro de entrada del método en teoría deberá ser object, mientras que si el proceso es el contrario y es en un método donde creamos el tipo anónimo y lo deseamos pasar como valor de retorno del método, tendría que ser también object aunque esto no es del todo cierto ya que en ocasiones lo métodos van a forzar una conversión y de lo contrario pueden lanzar una excepción.

Un apunte importante más sobre los tipos anónimos es que en C# las instancias de estos son inmutables, entendiendo como inmutables que los valores de campo y propiedad se fijan para siempre.

Si volvemos a fijarnos en el .NET Reflector y vemos las propiedades que se han creado, observamos que esas propiedades tienen métodos getter y no setter.



La única forma de asignar valores a las propiedades y a sus campos subyacentes es a través del constructor de la clase. Cuando invocamos la sintaxis para inicializar una instancia de un tipo anónimo, el constructor de ese tipo se invoca automáticamente y los valores se establecen para siempre.

Esto se hace para evitar posibles efectos secundarios en los lenguajes funcionales. Los objetos que no cambian de valor permiten acceso concurrente para trabajar mucho mejor. Esto será útil para en el futuro permitir PLINQ (*Parallel LINQ*), un proyecto de Microsoft para incorporar la concurrencia en consultas LINQ.

Por tanto los tipos anónimos inmutables nos lleva a .NET un paso más cerca de un mundo de programación más funcional donde podemos utilizar código libre de efectos secundarios.

Por otra parte, los tipos anónimos inmutables hemos comentado que son para C#, VB.NET tiene un comportamiento diferente y los tipos anónimos son mutables. Esto hace la necesidad de utilizar un modificador `Key` en las propiedades de un tipo.

4.3 BLOQUES DE CONSTRUCCIÓN LINQ

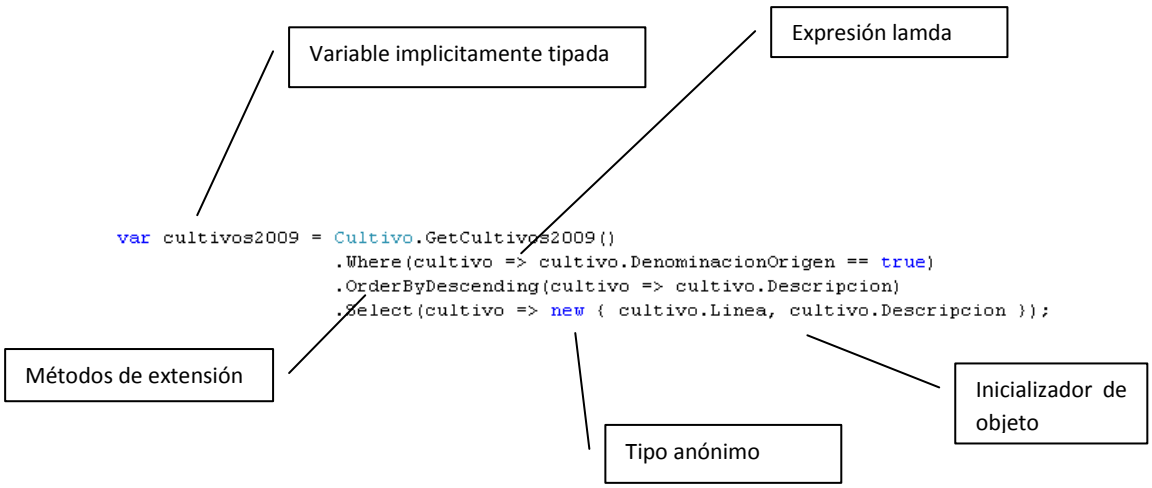
En el punto anterior se revisaron las adiciones del lenguaje hechas para VB.Net y C#, en este apartado se mostrarán nuevos conceptos únicos en LINQ basados en las características del apartado 4.2 de esta tesina.

Se va a detallar las secuencias, los operadores estándar de consulta, expresiones de consulta y árboles de expresión, y por último se verá como amplia LINQ el marco del Framework de .NET.

Para comenzar este apartado de bloques de construcción en LINQ tenemos que conocer con exactitud cuáles son las incorporaciones más significativas en los lenguajes que hemos tratado en el apartado anterior:

- Variables locales implícitamente tipadas
- Inicializadores de objeto
- Expresiones lambda
- Métodos de extensión
- Tipos anónimos

En la figura de a continuación se muestran a modo de ejemplo estas características, donde `GetCultivos2009()` es un método estático de clase `Cultivo` que devuelve una `List<Cultivo>` del plan 2009 :



El recorrido por este punto es muy importante y seguiremos este orden:

- Veremos que son las sentencias y como se utilizan en consultas LINQ
- Utilización de la expresiones y los operadores de consulta
- Veremos la importancia de una ejecución en diferido y la utilización que hace LINQ de los árboles de expresión.

4.3.1 SECUENCIAS EN LINQ

El primer concepto de LINQ que se presentará serán las secuencias por su importancia y porque son las estructuras de datos más utilizadas, por ellos es muy importante comprenderlas.

4.3.1.1 LA INTERFAZ IENUMERABLE<T>

La interfaz `IEnumerable<T>` es una interfaz clave que encontraremos en todas partes cuando se utiliza LINQ. Para explicarla utilizaremos la figura que nos ha servido en la introducción para explicar las extensiones del lenguaje:

```
var cultivos2009 = Cultivo.GetCultivos2009()
                   .Where(cultivo => cultivo.DenominacionOrigen == true)
                   .OrderByDescending(cultivo => cultivo.Descripcion)
                   .Select(cultivo => new { cultivo.Linea, cultivo.Descripcion });
```

Es importante comprender para que utilizamos esta consulta y que nos va a devolver. Nos va a devolver una lista de `Cultivo`, esto no sería interesante si no fuera porque las tablas implementan la interfaz genérica `IEnumerable<T>`. Esta interfaz que nació con .NET 2.0 y hace que se la utilicemos de tal forma que se implemente `IEnumerable<Cultivo>`.

La importancia de esta interfaz reside en que los operadores estándar de consulta en LINQ como `Where`, `OrderByDescending`, `Select` implementan un objeto que espera este tipo.

Ejemplo de la implementación del método `Where` en nuestro ejemplo:

```
public static IEnumerable<TSource> Where<TSource>(this IEnumerable<TSource> source,Func<TSource, Boolean> Predicate)
{
    foreach (TSource element in source)
    {
        if(Predicate(element))
            yield return element;
    }
}
```

El método `Where` es un método de extensión, que se puede deducir por la presencia de la palabra reservada `this` en el primer parámetro, estos métodos de extensión esta proporcionados por la clase `System.Linq.Enumerable`.

Por otra parte la aparición de la sentencia `yield return` y el tipo de retorno en la firma lo convierte en un iterador.

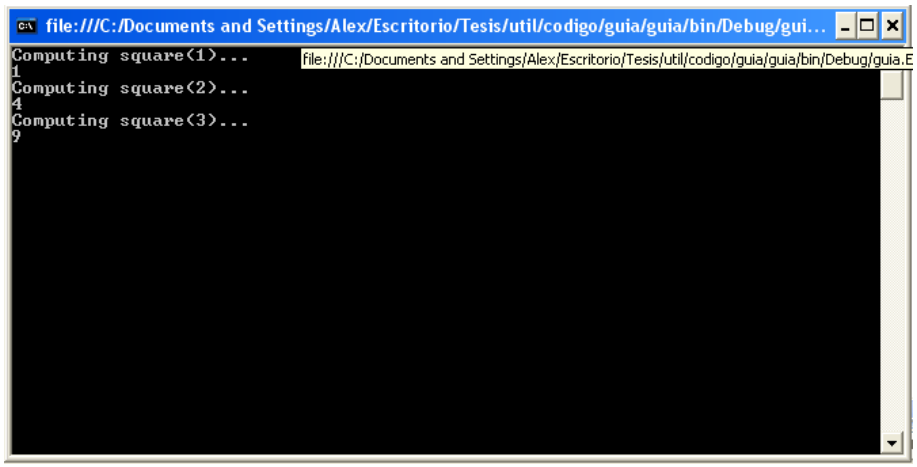
4.3.1.2 EJECUCIÓN DIFERIDA DE CONSULTA

Las consultas LINQ se basan en una evaluación perezosa, en otras palabras una ejecución diferida. Este concepto es muy importante para los desarrolladores que utilizan LINQ ya que una misma consulta la van a poder utilizar distintas veces para obtener distintos resultados entre otras maniobras que se pueden hacer en el código antes de ejecutarla.

Veamos un sencillo ejemplo para comprender lo que se desea explicar:

```
class Program
{
    static void Main(string[] args)
    {
        int[] numbers = {1,2,3};
        var query = from n in numbers
                    select Square(n);

        foreach (var n in query)
            Console.WriteLine(n);

        Console.ReadLine();

    }

    static double Square (double n)
    {
           Console.WriteLine("Computing square(" + n + ")... ");
           return Math.Pow(n,2);
    }
```

Y que da como resultado:



Los resultados muestran claramente que la consulta no se ejecuta inmediatamente. En su lugar, la consulta evalúa a medida que iteramos sobre ella. Por tanto la consulta no se ejecuta cuando es asignada a una variable, sino después paso a paso.

Una de las ventajas principales de la evaluación diferida es que conserva recursos, y por tanto la fuente de datos no se itera hasta que no se desea a diferencia de la programación clásica donde los contenidos de una tabla o lista se cargan en memoria. Esto es porque los resultados se proporcionan como una secuencia (implementado la interfaz `IEnumerable`).

Otra ventaja es que nos permite definir la consulta en un punto y ejecutarla más tarde, en otro punto o varios puntos para obtener resultados diferentes.

Veamos un ejemplo:

```
public List<Cultivo> SearchCultivo(int planDesde, int planHasta, bool denominacionOrigen )
{
    DataClassesDataContext bd = new DataClassesDataContext();

    var cultivosBuscados = from cultivo in bd.Cultivo
                           select cultivo;

    // realizo el filtrado
    if (planDesde != null) cultivosBuscados.Where(cultivo => cultivo.Plan >= planDesde);

    if (planHasta != null) cultivosBuscados.Where(cultivo => cultivo.Plan <= planHasta);

    if (denominacionOrigen != null) cultivosBuscados.Where(cultivo => cultivo.DenominacionOrigen == denominacionOrigen);

    //ejecuto y devuelvo la lista
    return cultivosBuscados.ToList();
}
```

En este ejemplo se ha querido simular un método que nos podría servir por ejemplo para implementar un sistema de filtrado de una búsqueda. Al método le llegan una serie de parámetros y en el propio método se define la consulta, después se detecta si existe el parámetro de búsqueda y filtra por él. De esta forma la consulta no se ejecuta y le vamos dando forma poco a poco.

Fijémonos ahora en lo que se ha querido hacer en el `return` del método. En él la consulta que por defecto tiene un comportamiento de ejecución diferida se fuerza la ejecución explícitamente, de tal forma que lo que devuelve el método es una lista de objetos filtrados y ya tipados. Se ha **forzado** la ejecución.

4.3.2 Operadores de consulta y Expresiones de Consulta

Los métodos de extensión que hasta ahora se están utilizando en esta tesis previenen de la clase `System.Linq.Enumerable` y apartir de ahora los denominaremos **operadores de consulta**.

Los operadores de consulta se pueden combinar para desarrollar operaciones complejas y consultas sobre enumeraciones. Varios operadores de consulta están predefinidos y abarcan un amplio rango de operaciones. Estas operaciones se denominan operadores de consulta.

A continuación se muestra una tabla clasificada acuerdo al tipo de operador [11]:

Operador	Descripción
Agregación	
Aggregate	Realiza un método personalizado sobre una secuencia
Average	Calcula el promedio de una secuencia de valores numéricos
Count	Devuelve el número de elementos en una secuencia como un valor int
LongCount	Devuelve el número de elementos de una secuencia como un valor long
Min	Encuentra el número mínimo de una secuencia de números
Max	Encuentra el número máximo de una secuencia de números
Sum	Suma los números en una secuencia
Concatenación	
Concat	Concatena dos secuencias en una
Conversión	
Cast	Convierte los elementos de una secuencia en un tipo determinado
OfType	Filtra los elementos de una secuencia de un tipo determinado
ToArray	Devuelve una matriz desde una secuencia
ToDictionary	Devuelve un diccionario desde una secuencia
ToList	Devuelve una lista desde una secuencia
ToLookup	Devuelve una búsqueda desde una secuencia
ToSequence	Devuelve una secuencia de IEnumerable
Elemento	
DefaultIfEmpty	Crea un elemento predeterminado para una secuencia vacía
ElementAt	Devuelve el elemento de un índice determinado en una secuencia
ElementAtOrDefault	Devuelve el elemento de un índice determinado en una secuencia o un valor predeterminado si el índice está fuera del intervalo
First	Devuelve el primer elemento de una secuencia
FirstOrDefault	Devuelve el primer elemento de una secuencia o un valor predeterminado si no se

	encuentra ningún elemento
Last	Devuelve el último elemento de una secuencia
LastOrDefault	Devuelve el último elemento de una secuencia o un valor predeterminado si no se encuentra ningún elemento
Single	Devuelve el único elemento de una secuencia
SingleOrDefault	Devuelve el único elemento de una secuencia o un valor predeterminado si no se encuentra ningún elemento
Igualdad	
SequenceEqual	Compara dos secuencias para ver si son equivalentes
Generación	
Empty	Genera una secuencia vacía
Range	Genera una secuencia en función de un intervalo
Repeat	Genera una secuencia mediante la repetición de un elemento un número determinado de veces
Agrupación	
GroupBy	Agrupa elementos en una secuencia en función de una agrupación dada
Unión	
GroupJoin	Realiza una unión agrupada en dos secuencias
Join	Realiza una unión interior en dos secuencias
Ordenación	
OrderBy	Ordena una secuencia en función de los valores en orden ascendente
OrderByDescending	Ordena una secuencia en función de los valores en orden descendente
ThenBy	Ordena una secuencia que ya se ha ordenado en orden ascendente
ThenByDescending	Ordena una secuencia que ya se ha ordenado en orden descendente
Reverse	Invierte el orden de los elementos de una secuencia
Partición	
Skip	Devuelve una secuencia que omite un número determinado de elementos
SkipWhile	Devuelve una secuencia que omite elementos que no cumplen una expresión
Take	Devuelve una secuencia que toma un número determinado de elementos

TakeWhile	Devuelve una secuencia que toma elementos que cumplen una expresión
Proyección	
Select	Crea una proyección de elementos de una secuencia
SelectMany	Crea una proyección uno a varios de elementos de una secuencia
Cuantificadores	
All	Determina si todos los elementos de una secuencia cumplen una condición
Any	Determina si algún elemento de una secuencia cumple una condición
Contains	Determina si una secuencia contiene un elemento dado
Restricción	
Where	Filtra los elementos de una secuencia
Establecimiento	
Distinct	Devuelve una secuencia sin elementos duplicados
Except	Devuelve una secuencia que representa la diferencia entre dos secuencias
Intersect	Devuelve una secuencia que representa la intersección entre dos secuencias
Union	Devuelve una secuencia que representa la unión entre dos secuencias

Otro concepto clave de LINQ es la nueva sintaxis que se propone a la hora de integrarla en el lenguaje que hace que sea mucho más sencilla y resulta más familiarizada para los programadores que conocen SQL. Durante los ejemplos hemos visto llamadas a métodos para realizar las consultas en código LINQ pero veremos que en la mayoría de casos podemos escribirlas como **expresiones de consulta** de tal forma que ganamos en similitud con SQL.

Mejor véase con un ejemplo la nueva sintaxis de las expresiones de consulta. Durante ejemplos anteriores utilizábamos este tipo de consulta:

```
// Ejemplo con Operadores de Consulta Estandar
var cultivosOC = bd.Cultivo
                .Where(cultivo => cultivo.DenominacionOrigen = true)
                .OrderByDescending(cultivo => cultivo.Linea)
                .Select(cultivo => new {cultivo.Linea, cultivo.Descripcion});
```

Pero podemos escribirla de tal forma que parezca una consulta SQL:

```
// Ejemplo con Expresiones de Consulta
var cultivosEC = from cultivo in bd.Cultivo
                 where cultivo.DenominacionOrigen = true
                 orderby cultivo.Linea descending
                 select new {cultivo.Linea, cultivo.Descripcion};
```

Ambas consultas LINQ son semánticamente idénticas, la única diferencia es que la segunda consulta es una abreviatura declarativa conveniente para código que podría escribirse manualmente. Por tanto las expresiones de consulta permiten utilizar la potencia de los operadores de consulta con una sintaxis orientada a consulta SQL (ganando en legibilidad y simplicidad). Veamos la sintaxis en C# y VB.NET

SINTAXIS C#

```
from [ type ] id in source

[ join [ type ] id in source on expr equals expr [ into id ]]

[ from [ type ] id in source | let id = exp | where expression ]

[ orderby ordering, ordering, ...]

select exp | group expr by key

[ into id query ]
```

SINTAXIS VB.NET

```
From id [As type] In source [, id2 [As type] In source2 [...] ]

{
   Aggregate id [As type] In source _
        [, id [As type2] In source2 [...] ]
        [ clause ]
        Into [alias =] aggregateExpresion
        [, [alias =] aggregateExpresion […] ]
   Distinct
   From id [As type] In source [, id2 [As type] In source2 […]]
   Group [ column [, column2 […]]] _
        By keyExpr [, keyExpr2 […] _
        Into groupAlias = Group [, aggregations]
```

```
    Group Join id [As type] In source
         On keyA Equals keyB [And keyA2 Equals keyB2 […]]
         Into expressionList
    Join id In source [joinClause] [groupJoinClause …] _
         On keyA Equals keyB [And keyA2 Equals keyB2 […]]
    Let id = expression [,id2 = expresion2 […]]
    Order By orderExpr [Ascending|Descending] _
         [, orderExpr2 [Ascending|Descending] […]]
    Select [alias =] columnExpr [,[alias2 =] columnExpr2[…]]
    Skip count
    Skip While condition
    Take count
    Take While condition
    Where condition
}
```

Si se compara la sintaxis de VB.NET es más rica en cuanto a operadores de consulta estándar soportados.

El compilador de .NET traduce automáticamente la expresión de consulta en operadores de consulta estándar. Aún así no todos los operadores tienen palabras claves equivalentes en C# y VB.NET por lo que en muchas ocasiones será más sencillo invocar directamente al operador.

4.3.3 ARBOLES DE EXPRESIÓN

Los arboles de expresión permiten a LINQ extensibilidad avanzada y hacen posible en gran medida LINQ to SQL.

Cuando una expresión lambda se utiliza como datos en vez de código se trata de un árbol de expresión. Vemos el ejemplo:

```
Func<int,bool> isOdd = i => (i & 1) == 1;
```

Este código representa un método (un delegado) que toma como parámetro un entero y devuelve un booleano.

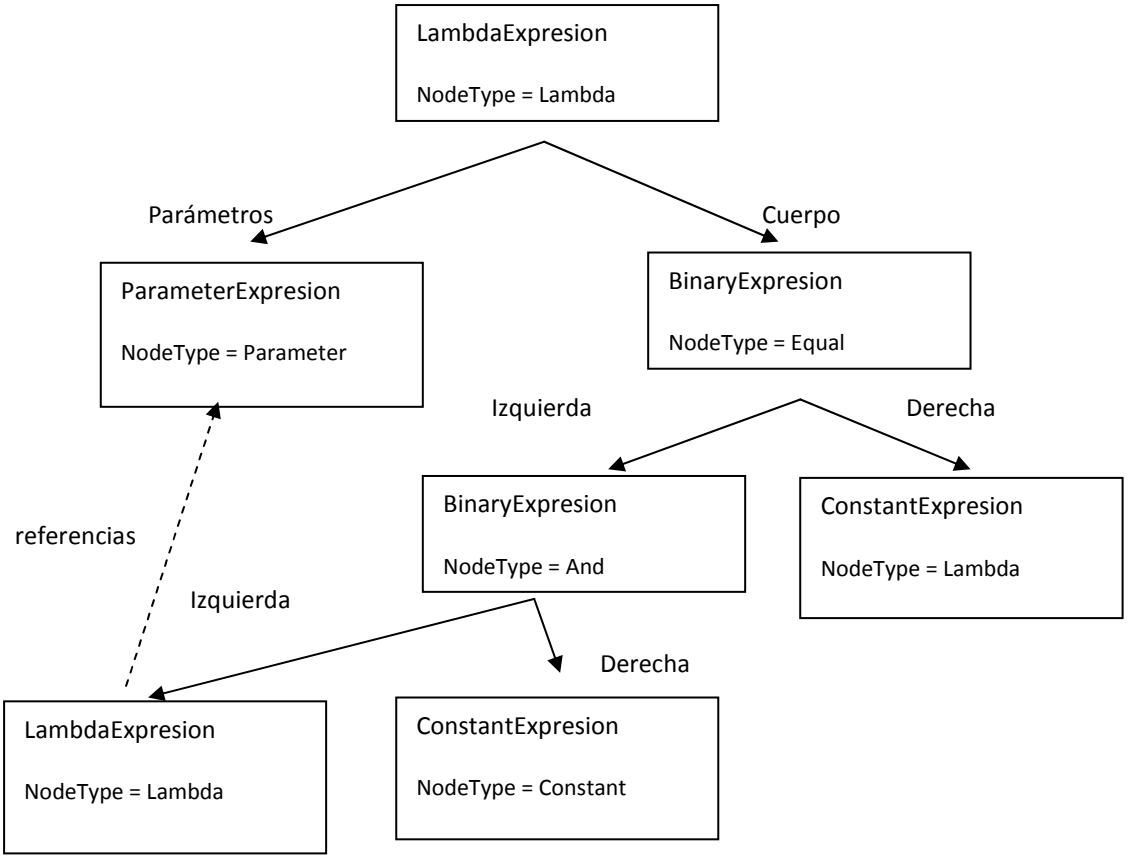
Veámosla como un árbol de expresión:

```
Expression<Func<int,bool>> isOdd = i => (i & 1) == 1;
```

El compilador de .NET reconoce este tipo `Expression<TDelegate>` y hace que se comporte como `Func<T, TResult>` evaluando la expresión y haciendo que se comporte como un árbol de expresión.

Obsérvese como las expresiones lambda con cuerpo de expresión se pueden expresar como arboles de expresión.

Cuando el compilador construye el árbol de expresión no se puede modificar en tiempo de ejecución. Ejemplo, un árbol de expresión generado por el compilador:



A modo resumen podemos decir que las expresiones lambda se pueden representar como código (delegados) o como datos (árboles de expresión). Asignada a un delegado, una expresión lambda emite código IL; asignada a `Expression<TDelegate>`, emite un árbol de expresión, que es una estructura de datos en memoria que representa lambda analizada

4.3.4.1 IQUERYABLE, OTRA FORMA DE REALIZAR LA EJECUCIÓN DIFERIDA

En un punto anterior de la tesina se ha visto una forma de realizar consultas diferidas mediante la interfaz `IEnumerable<T>` y iteradores. Los arboles de expresión son la base de otra forma de realizar las consultas fueras del proceso.

Este tipo de consulta diferida es utilizada sobretodo en LINQ to SQL, escribiendo la consulta y no ejecutándose hasta que no accede al bucle.

```
DataClassesDataContext bd = new DataClassesDataContext();

var queryCultivos = from cultivo in bd.Cultivo
                    where cultivo.DenominacionOrigen = true
                    orderby cultivo.Linea descending
                    select new {cultivo.Linea, cultivo.Descripcion};

foreach (var cultivo in queryCultivos)
{
        Console.WriteLine("Cultivo " + cultivo.Descripcion + " es de la linea " + cultivo.Linea);
}
```

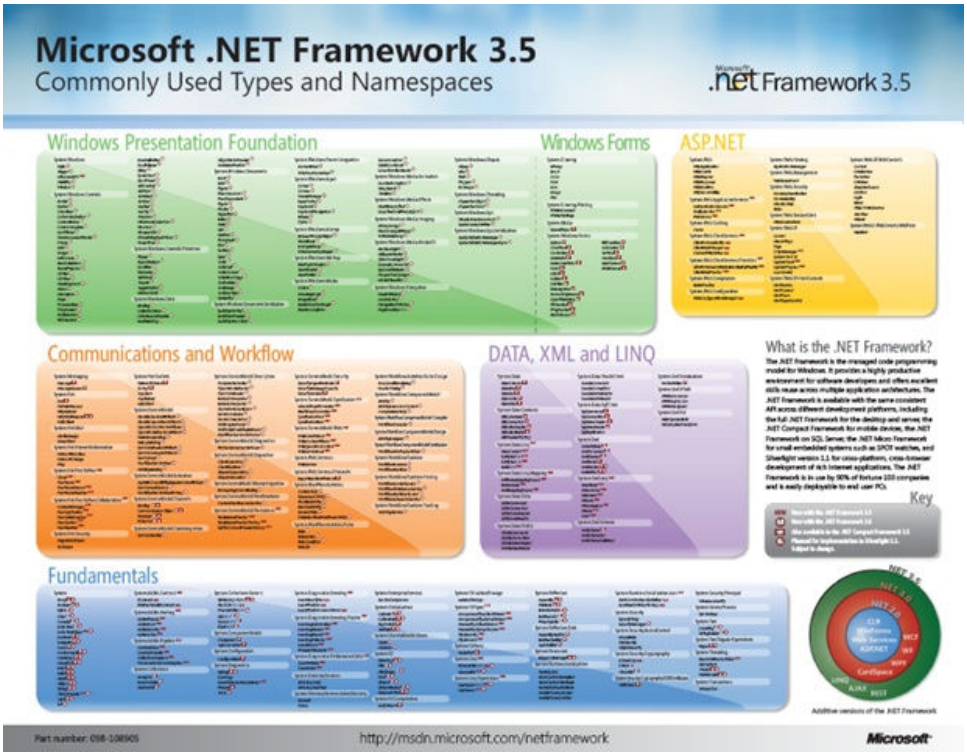
El comportamiento es similar al de `IEnumerable<T>` pero esta vez el tipo `cultivo` no es `IEnumerable<Cultivo>`, sino `IQueryable<Cultivo>`. La forma de procesar este tipo es diferente que como sucede en las secuencias, `IQueryable <T>` recibe un árbol de expresión que puede inspeccionar para decidir que procesado debería realizar.

En el ejemplo en el momento que se comienza el bucle se genera la SQL, se ejecuta y se devuelven los objetos de tipo `Cultivo`.

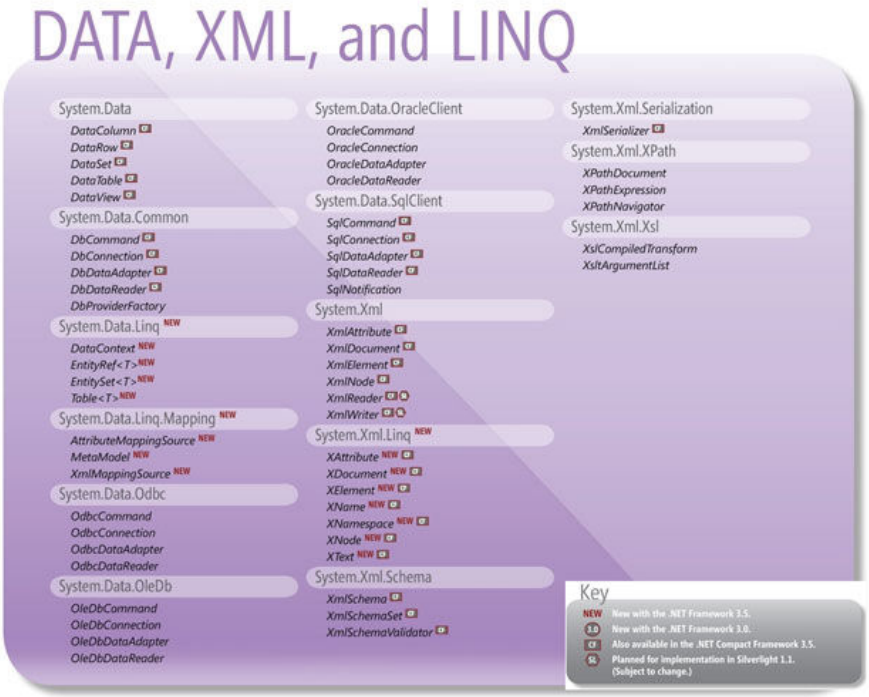
La ejecución de consulta diferida con árboles de expresión permite a LINQ to SQL optimizar una consulta que contiene múltiples consultas anidadas o complejas en el más reducido número de sentencias SQL eficientes posibles, mejorando el patrón de pipeling que utiliza `IEnumerable`.

4.3.5 MARCO DE .NET, LAS DLL Y ESPACIOS DE NOMBRE LINQ

Las clases y ensamblajes (DLL) que utiliza LINQ vienen distribuidas en el .NET Framework 3.5 de Microsoft.



La librería principal que se utilizará será `System.Core.dll` (por defecto se hace referencia a esta librería al crear una nueva solución en Visual Studio 2008). Para escribir consultas LINQ to Objects será necesario utilizar el espacio de nombres `System.Linq`.



4.4. LINQ TO OBJECTS

En los puntos anteriores de este capítulo se han presentado las nuevas características del lenguaje y los conceptos de LINQ. En los tres siguientes puntos veremos ejemplos de los tipos LINQ, en este caso trataremos LINQ to Objects.

LINQ to Objects podría decirse que es el tipo base de LINQ, por la sencilla razón de que la principal característica es tratar colecciones de objetos en memoria y teniendo en cuenta que si se tiene una fuente de datos que carga los datos en memoria para posteriormente operar con ellos podría decirse que es el tipo LINQ que más se utiliza a la hora de desarrollar aplicaciones utilizando LINQ.

4.4.1. LINQ CON COLECCIONES DE MEMORIA

En realidad no todo se puede consultar con LINQ to Objets, pero si la mayoría de colecciones genéricas de .NET.

Todo lo que se requiere para que una colección sea consultable con LINQ es que implemente la interfaz `IEnumerable<T>` (ver punto 4.3.1.1 de la tesina). Normalmente casi todas las colecciones genéricas de .NET implementan dicha interfaz. Algunas de estas colecciones genéricas son:

- **Arrays**: de cualquier tipo de datos. Ej. `Object[] array = {"String", 12 , true, 'a'}`
- **Listas genéricas**:
  - `System.Collections.Generic.List<T>`
  - `System.Collections.Generic.LinkedList<T>`
  - `System.Collections.Generic.Queue<T>`
  - `System.Collections.Generic.Stack<T>`
  - `System.Collections.Generic.HashSet<T>`
  - `System.Collections.ObjectModel.Collection<T>`
  - `System.ComponentModel.BindingList<T>`

- **Diccionarios genéricos**: implementan `IEnumerable<KeyValuePair<TKey,TValue>>` donde la estructura `KeyValuePair` contiene las propiedades `Key` y `Value` tipadas.
  - `System.Collections.Generic.Dictionaty<TKey,TValue>`
  - `System.Collections.Generic.SortedDictionary<TKey,TValue>`
  - `System.Collections.Generic.SortedList<TKey,TValue>`
- **Cadenas**: a primera vista `System.String` no se percibe como una colección pero en realidad implementa `IEnumerable<Char>`

Las colecciones que se proporcionan anteriormente son colecciones proporcionadas por el marco de .NET pero debe incidirse en la idea que cualquier otro tipo de colección que implemente la interfaz `IEnumerable<T>` podrá ser manipulada por LINQ to Objetcs y las operaciones soportadas pueden consultarse en el punto 4.3.2 de la tesina, las operaciones de consulta estándar.

En determinadas ocasiones en programador se puede encontrar con que la colección con la que trabaja no implementa dicha interfaz. De hecho solo las colecciones fuertemente tipadas implementan esta interfaz. Las colecciones no genéricas no implementan `IEnumerable<T>` pero implementan `IEnumerable`, con lo que se soluciona el problema utilizando operadores de consulta `Cast` y `OfType`.

4.5 LINQ TO SQL

En el punto anterior se han abordado el trabajo en LINQ centrado en los objetos en memoria para los cuales era necesario trabajar sobre la interfaz `IEnumerable<T>`. Aunque se hace muy necesario tener esta característica, no es suficiente para abordar la problemática de interactuar con una base de datos relacional como SQL Server.
En este punto se va a presentar LINQ to SQL, un tipo de LINQ que nos permite mapear una base de datos relacional, realizar consultas sobre ella en LINQ y ampliar las clases generadas, lo que permite ahorrarse una tediosa faena desde el punto de vista de cómo se estaba haciendo hasta ahora con ADO.NET.

Comenzaremos por echar un vistazo a las opciones de mapeado existentes y nos centraremos en la que nos proporciona Visual Studio 2008, se explicará la interacción entre las clases generadas y la base de datos abordando los típicos problemas y se terminará explicando cómo se pueden ampliar las clases generadas al realizar el mapeado de la base de datos relacional.

4.5.1. MAPEAR LINQ TO SQL

Para eliminar el código repetitivo de ADO.NET y facilitar el acceso a datos de una base de datos relacional surge LINQ to SQL, y el primer paso que se debe hacer para poder tener dicha interacción es el mapeado de las tablas a clases. Cuando se habla de mapear se entiende que se están generando una serie de clases que emulan las propiedades, relaciones y tipos de la base de datos. Proporcionando clases que nos permitirán traducir nuestras construcciones de consulta declarativas en una sintaxis que pueda reconocer nuestra base de datos.

Con esto conseguimos un marco de trabajo que gestiones el mapeado de las tablas relacionales a nuestras clases de entidad de negocio.

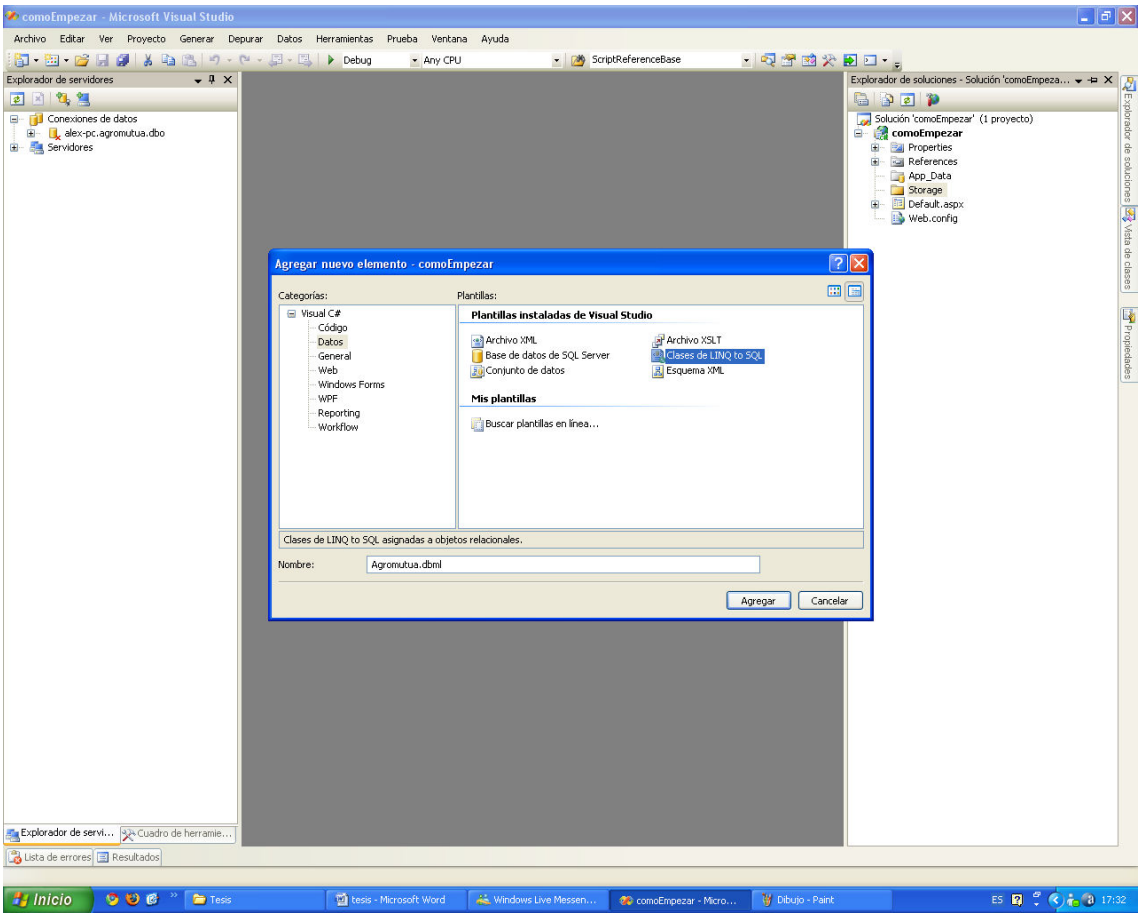
Veamos la lista de opciones que podemos tener a la hora de mapear la base de datos:

- Atributos declarados en línea con sus clases, o dicho de otra forma, hacer el mapeo manualmente.
- Utilizando XML externos
- Herramienta SqlMetal de línea de comando
- Herramienta gráfica LINQ to SQL Designer

Ya que en el proyecto Agromutua.Web que estamos abordando se ha utilizado la herramienta gráfica de Visual Studio 2008 LINQ to SQL Designer, se abordará el mapeo viendo su uso.

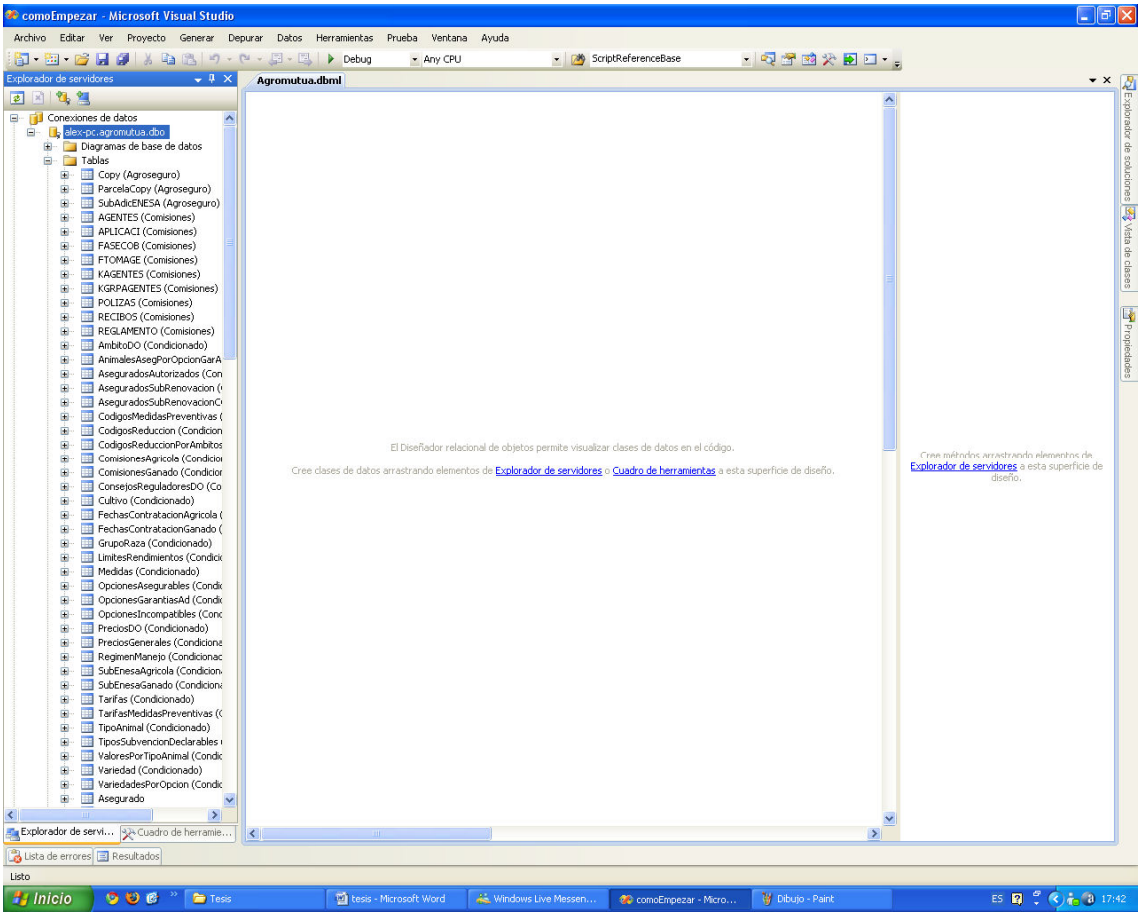
Esta herramienta es muy útil, ya que aparte de proporcionar el mapeo también provee de una visión gráfica de la base de datos que tenemos mapeada y proporciona los enlaces establecidos.

Para empezar el proceso se deberá crear un nuevo archivo de base de datos en el proyecto seleccionando la de la lista de ficheros la opción "Clases de LINQ to SQL"



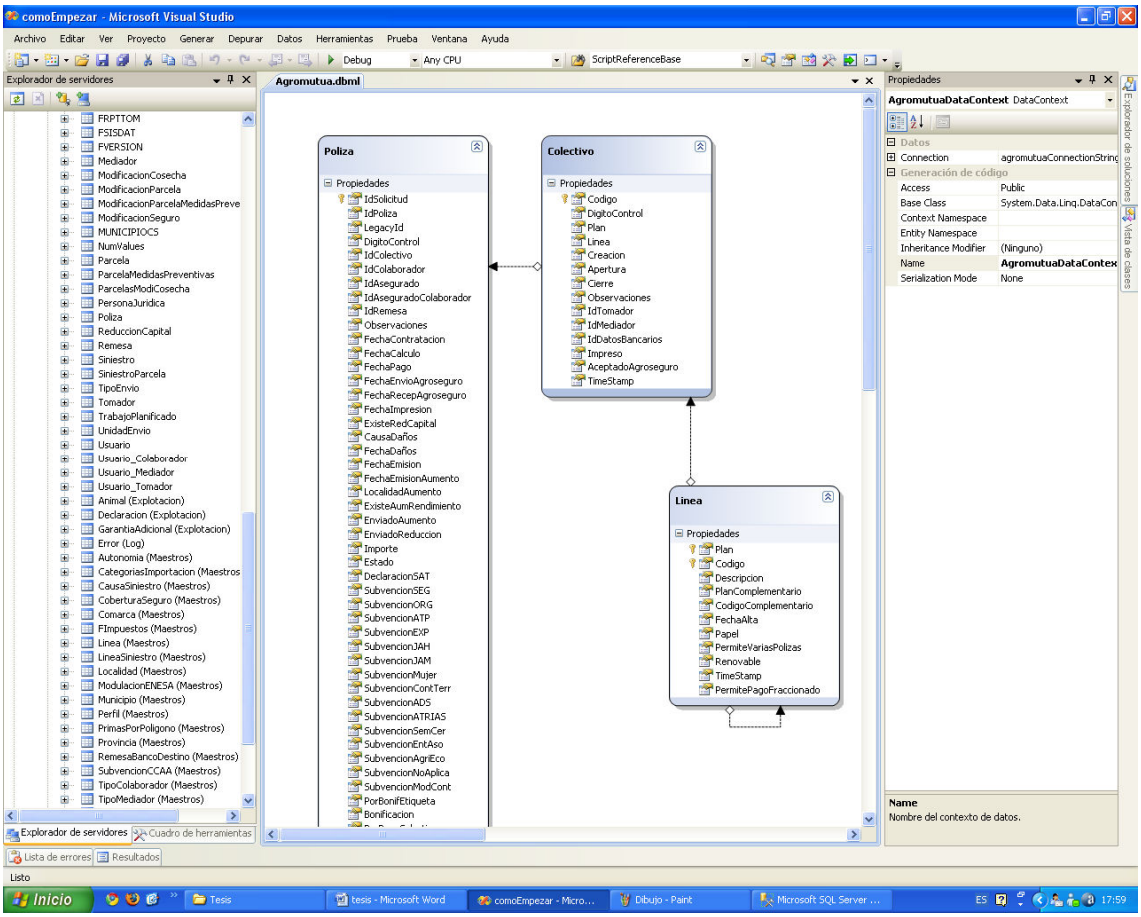
El resultado de añadir este archivo al proyecto donde se esté trabajando es la creación de un archivo con extensión DBML (*Database Markup Language*).

Nada más crearlo se abrirá automáticamente la edición de este archivo con el LINQ to SQL Designer de Visual Studio 2008, pero para poder manipular y añadir las tablas a mapear será necesario disponer de una conexión a un servidor de base de datos. En el ejemplo se conecta con SQL Server 2005



Una vez se enlace el servidor de base de datos con la base de datos aparecen todas las tablas contenidas en la base de datos relacional listas para ser arrastradas al diseñador que es la parte situada en el centro de la imagen.

Y a continuación se pueden arrastrar las tablas que se deseen para que el diseñador mapee automáticamente cada tabla y sus relaciones. En este ejemplo se han arrastrado las tablas de *dbo.Poliza, dbo.Colectivo y Maestros.Linea* y se observa a simple vista todos sus campos, restricciones y asociaciones han sido correctamente creadas.

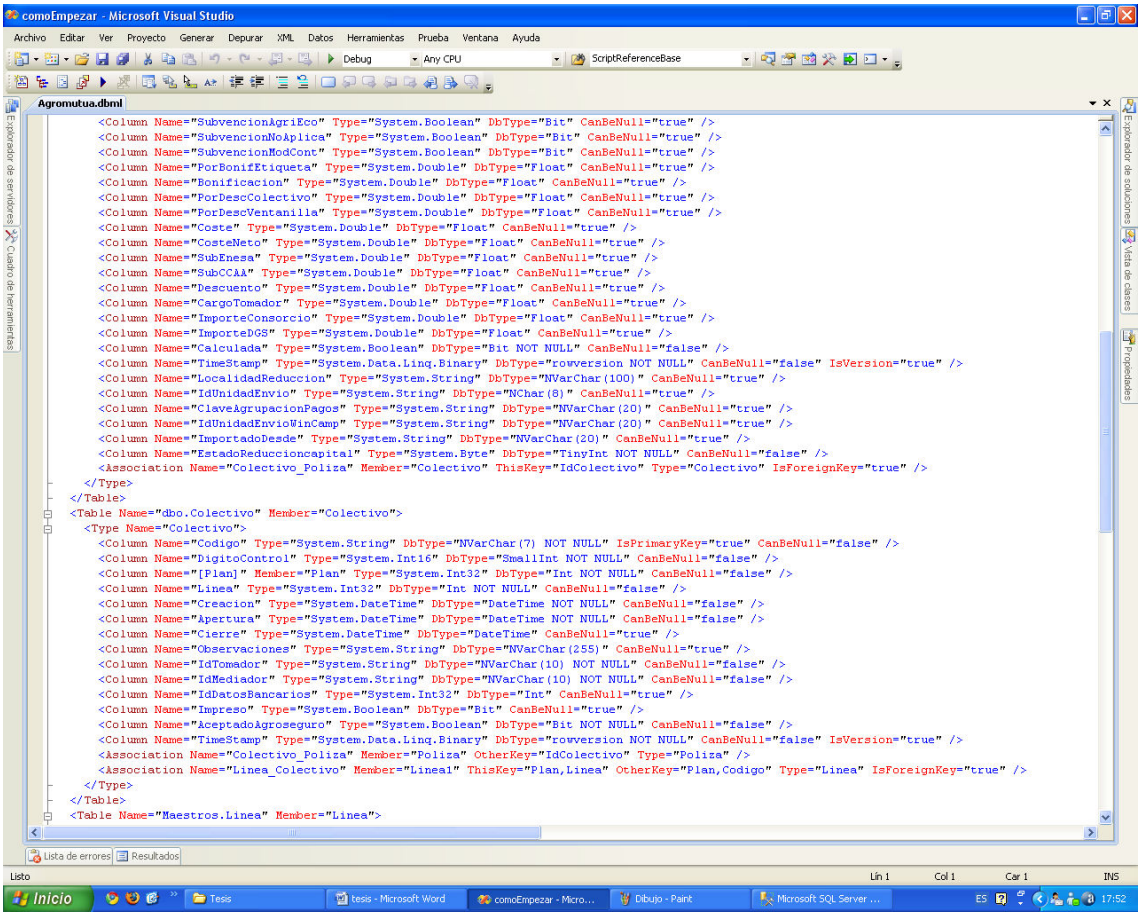


El editor gráfico permite la manipulación y edición de todo lo que se ha arrastrado, por lo que se puede borrar campos, añadir restricciones, etc.

De esta forma cada caja (tabla de la base de datos) que se observa se convertirá en una clase, y cada flecha en una relación.

En la parte de "Propiedades" se pueden configurar las opciones referentes a la cadena de conexión y al nombre del DataContext. Una vez se guarde las tablas que queremos mapear, se generaran un archivo de metadatos basado en XML (el DBML) que especifica cómo se generan las clases. Otro archivo XML que contiene información del diseño visual en la superficie del diseñador (XXX.dbml.layout) y las clases generadas en un archivo (X.designer.cs)

Si se desea ver las relaciones y propiedades que se han creado se puede hacer con el botón derecho sobre el DBML “Abrir con…” -> “Edito de XML” y se apreciaran las conversiones que se han realizado.

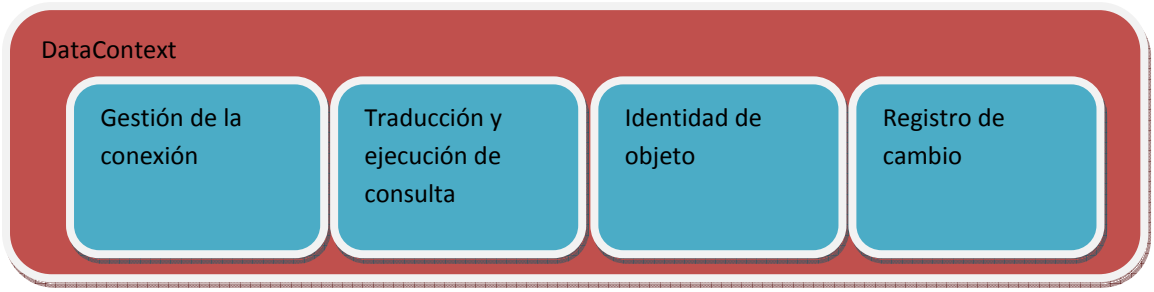


Una vez realizados estos sencillos pasos ya se puede comenzar a trabajar con LINQ to SQL. En los siguientes puntos de este capítulo se verá que tendremos que tener en cuenta y la importancia del DataContext.

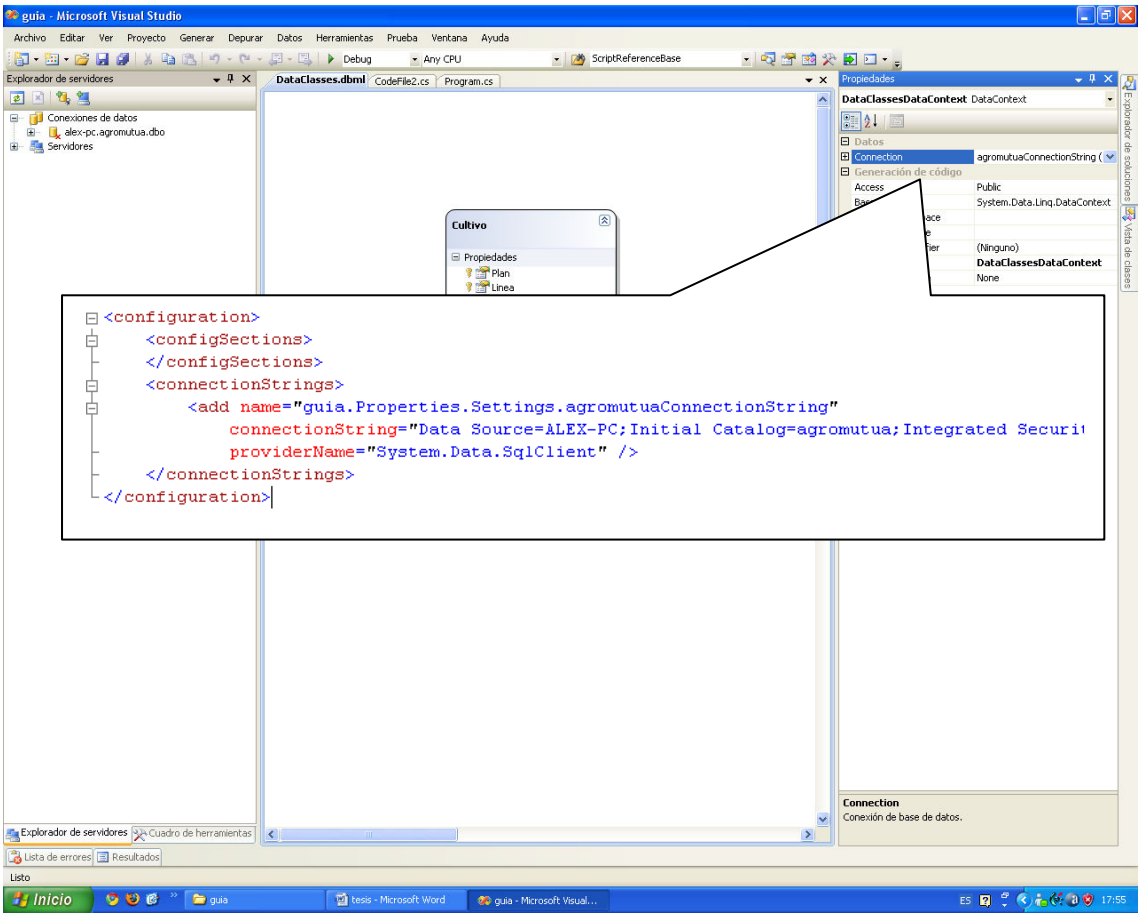
4.5.2. Interacción de LINQ to SQL con la BD

El DataContext, que se muestra en la figura de abajo, se localiza en el corazón de LINQ to SQL y maneja la mayor parte del trabajo. El DataContext gestiona la conexión con la base de datos, indicándole en sus propiedades la cadena de conexión o incluyéndola como parámetro en la creación del objeto. Por tanto con la utilización de este objeto no habrá que preocuparse de la apertura y cierre de la conexión ya que se encargará de manejarla.

Servicios ofrecidos por el DataContext:



Se puede configurar la cadena de conexión en el app.config de la aplicación web de tal forma que después se indique en las propiedades del DataContext con el editor gráfico, este proceso se hace normalmente automático pero el desarrollador podría querer cambiar la conexión en algún momento determinado.



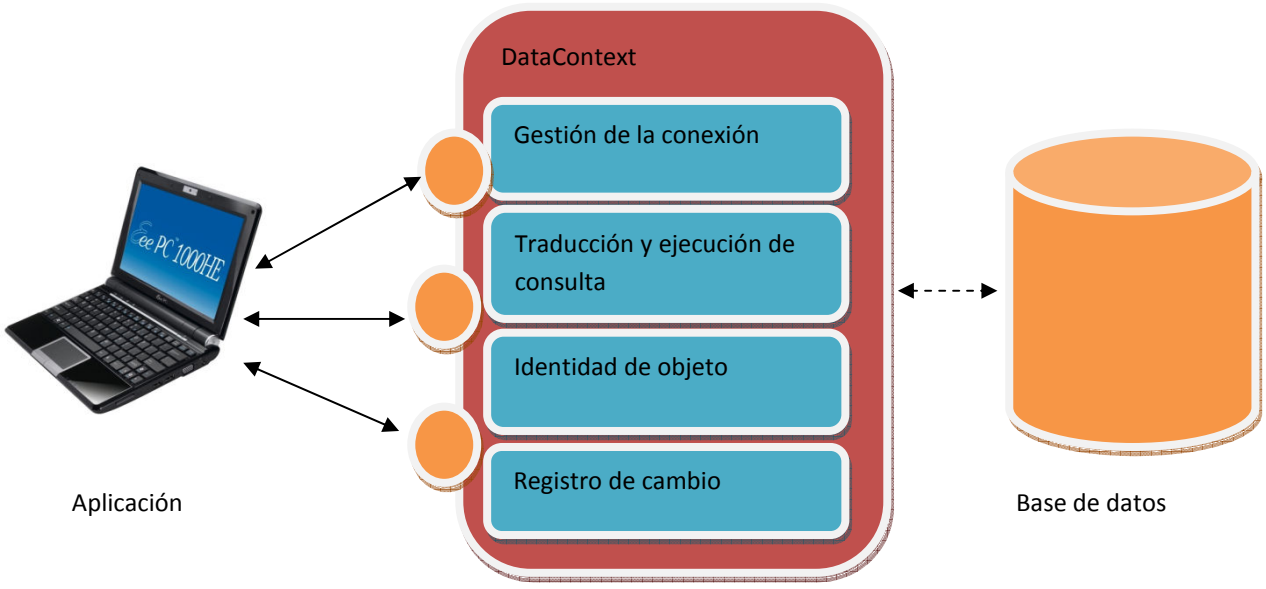
Con esto se consigue tener un enlace que además de gestionar la conexión con la base de datos, realice las pertinentes comprobaciones para su estabilidad entre la base de datos y LINQ to SQL.

Véase un ejemplo completo en el que después de ser creado DataContext es utilizado para realizar una consulta LINQ to SQL y obtener los pertinentes resultados de la base de datos:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace guia
{
    public class CultivoDataAccessObject
    {
        private DataClassesDataContext bd = new DataClassesDataContext();


        public IQueryable<Cultivo> GetCultivoByLinea(string linea)
        {
            var query = from cultivo in bd.Cultivo
                        where cultivo.Linea.ToString() = linea
                        select cultivo;

            return query.ToList();
        }

    }
}
```

Pero además de ver cómo se crean instancias del DataContext es importante comprender el ciclo de vida de la entidad mapeada de la base de datos. El DataContext realiza un papel fundamental al gestionar la conexión, evaluar los mapeados y traducir los árboles de expresión en estructuras consumibles.

Las aplicaciones que se realizan funcionan con los datos obtenidos de las consultas, visualizándose y realizando los cambios oportunos. Para ello el DataContext se encarga en LINQ to SQL de disponer de un mecanismo y detectar los cambios que se hacen en los valores y registrarlos hasta que no se necesiten, manejando el ciclo de vida del objeto.



Contexto personalizado de datos

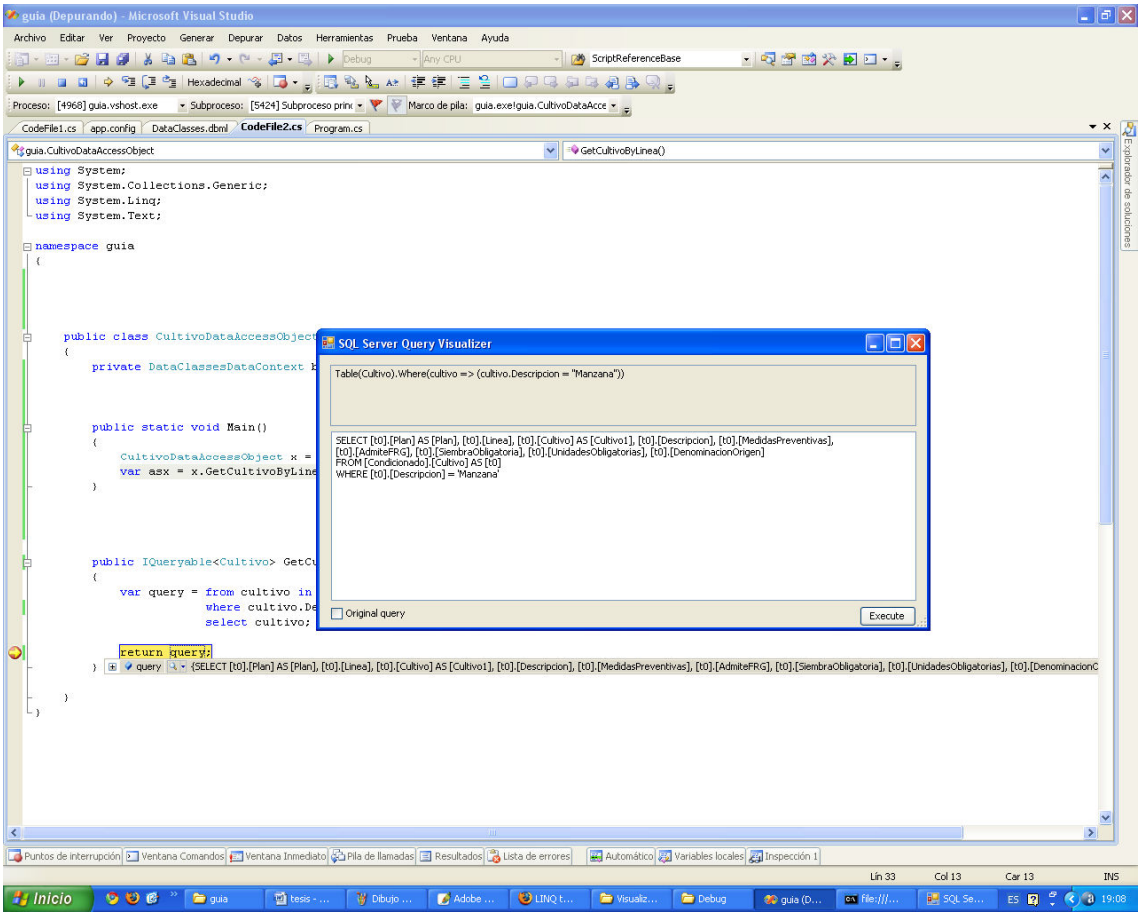
El ciclo de vida comienza cuando leemos por primera vez un valor de la base de datos. Antes de pasar el objeto resultante al código de la aplicación, DataContext mantiene una referencia al objeto. Un servicio de gestión de identidad rastra el objeto en una lista indexada por la identidad designada en el mapeado. Al retener este valor, podemos hacer referencia al objeto según su identidad de objeto.

Cada vez que se consulta valores de la base de datos DataContext comprueba con el servido de gestión de identidad para ver si un objeto con la misma identidad ya se ha proporcionado en una consulta anterior. Si es así, DataContext devolverá el valor almacenado en la cache interna en lugar de volver a mapear la fila de la tabla. Al mantenerse el valor original se puede proporcionar al usuario hacer cambios sin que afecten a los otros usuarios, ya que el dato solo estará en su copia de datos. Por tanto no hay que preocuparse por problemas de concurrencia hasta que los datos se envían.

Cuando se está trabajando con objetos de la base de datos mapeados, en algún momento puede surgir el problema o la necesidad de saber realmente que consulta se está realizando, ya que como se ha indicado más arriba LINQ nos va a servir para interactuar entre la base de datos y la aplicación de una forma sencilla y sin necesidad de aprender un lenguaje de consulta SQL, pero realmente lo que está haciendo es transformado lo que se opera con los objetos mapeados a consultas SQL que serán ejecutadas contra la base de datos.

Así se debe conocer que existen varias formas de conocer que consulta se está ejecutando:

- Herramienta de SQL Server Profiler que viene con SQL Server.
- Propiedad Log de DataContext hacia un flujo de salida, por ejemplo la consola o un archivo.
- Utilizando la herramienta *Query Visualizer* de Microsoft que viene de forma separada de Visual Studio 2008



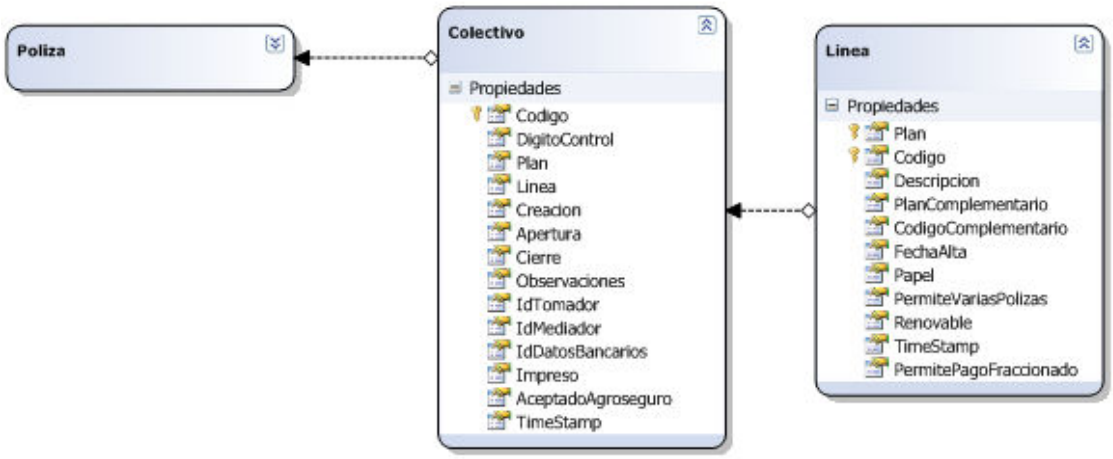
Si se observa las consultas que se lanzan hacia el servidor, en concreto las que contienen filtrado, se detecta que son consultas parametrizadas, lo que implica que LINQ to SQL solventa dos problemas de comunes en las consultas a base de datos. En primer lugar, la gran vulnerabilidad que supone la inyección SQL. La segunda mejora es que LINQ saca partido a la cache, permitiendo rentabilizarla y reducir tiempos en posteriores consultas repetidas.

4.5.3. CARGA DE DATOS EN MEMORIA

Cuando vamos a buscar datos a la base de datos, LINQ to SQL utiliza una técnica denominada ejecución diferida. La ejecución diferida permite que los datos se carguen en memoria cuando se solicite. Esperar a acceder a los valores hasta que se necesitan se denomina carga perezosa.

Cuando mostramos los resultados, la carga perezosa ofrece el beneficio de recuperar solamente los datos cuando se solicitan y devolver solamente los datos solicitados. En muchos casos esto proporciona beneficios de rendimiento, pero en otros casos, puede conducir a un resultado imprevisto.

En este ejemplo se dispone las relaciones entre Póliza, Colectivo y Línea que se observan.



Cuando se realiza una consulta y se desea obtener un objeto póliza concreto, se puede pensar que solo se está realizando una consulta a la base de datos, pero si se observa el fragmento de código de abajo se ve que LINQ to SQL te permite navegar por las relaciones y obtener datos de tablas relacionadas. Por tanto cuando se van a realizar más consultas y se van a traer datos asociados al registro de póliza que se deseaba, es decir, todos los registros hijos de Colectivo y a su vez todos los registros de Línea. En este punto algo que en principio era bueno, la carga perezosa, se convierte en algo malo respecto al rendimiento y a la carga.

```
var query = from p in bd.Poliza
            where p.IdSolicitud = "121212"
            select p;


foreach (var poliza in query.ToList())
{
    poliza.Colectivo.
    Console.Write(
}

Console.ReadLine()
}
```

LINQ to SQL da la posibilidad de cargar los datos inmediatamente, podemos almacenarlos previamente en tablas o listas con los métodos `ToList, ToDictionary, ToLookup o ToArray.`

Al indicar a LINQ que queremos recuperar los resultados, se fuerza a LINQ to SQL a ir a buscar inmediatamente los resultas y completar la nueva lista o tabla. El único inconveniente que se produce es que al convertir los resultado a la estructura de datos se pierden los beneficios de LINQ to SQL. Con estos métodos no se pierden las llamadas de consultas a las tablas dependientes pero si nos da la posibilidad de almacenar los datos para posteriores usos.

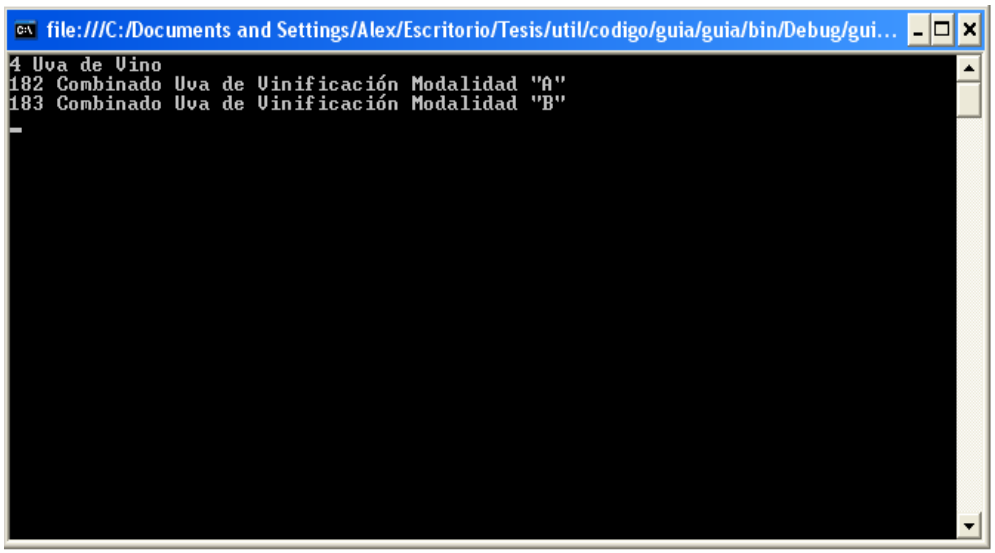
Afortunadamente se tiene la opción `DataLoadOptions` del DataContext que permite especificar los datos que se quieren cargar, eliminando las múltiples subconsultas que eran necesarias con la carga perezosa.

4.5.4. ACTUALIZACIÓN Y BORRADO DE DATOS

LINQ to SQL no solo se reduce a la búsqueda de datos, sería una funcionalidad reducida, por lo que también permite el uso de actualizaciones y borrado de datos. Lo importante es disponer de un DataContext persistente y a partir de ese momento utilizar métodos estándar de LINQ para realizar las operaciones mencionadas.

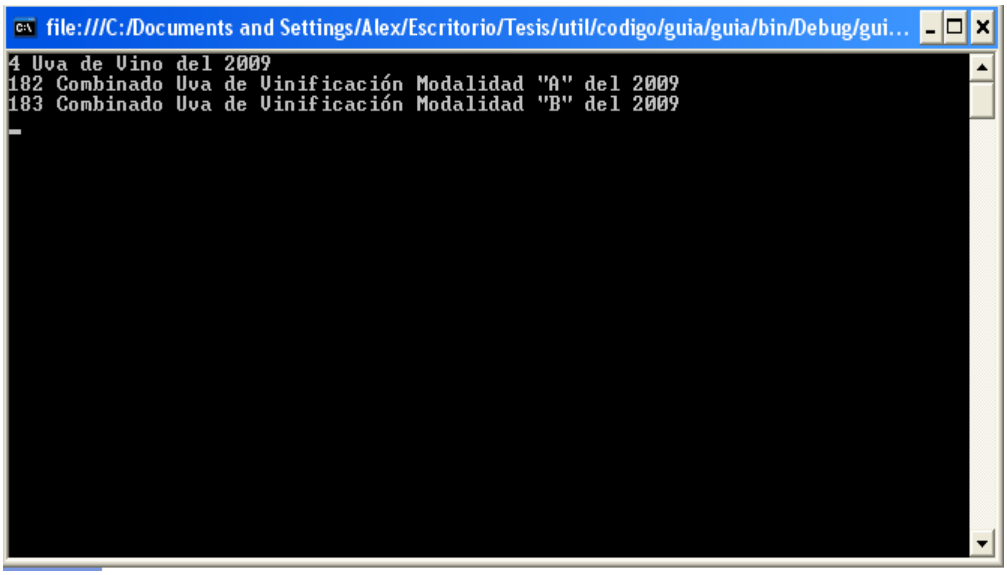
Para actualizar registros se utiliza `SubmitChanges` en el objeto DataContext, véase un ejemplo donde se muestra la *Línea* y su *Descripción*:

```
public static void Main()
{
    DataClassesDataContext bd = new DataClassesDataContext();

    var query = from l in bd.Linea
                where l.Plan == 2009 && l.FechaAlta > new DateTime(2009, 1, 1)
                select l;


    foreach (var linea in query)
    {
        Console.Write(linea.Codigo + " " +linea.Descripcion + "\n");
    }

    Console.ReadLine();

}
```



Con el siguiente código, incluyendo el `SubmitChanges()` se actualiza y ejecutando otra vez el fragmento de código de arriba se observa que se han actualizado los datos de la base de datos.

```
//actualizo
foreach (var linea in query)
{
    linea.Descripcion += " del " + linea.Plan;
}
bd.SubmitChanges();
```



De la misma forma se opera para crear y eliminar registros, utilizando los métodos `InsertOnSubmit()` y `DeleteOnSubmit()` respectivamente. En el ejemplo daríamos de alta y de baja un registro.

```
Linea lin = new Linea();
lin.Plan = 2009;
lin.Descripcion = "Linea de prueba";
lin.Codigo = "111";
lin.Colectivo = "1111111";

bd.Linea.InsertOnSubmit(lin);
bd.SubmitChanges();

bd.Linea.DeleteOnSubmit(lin);
bd.SubmitChanges();
```

Lo que se acaba de presentar anteriormente es realmente útil para trabajar con escenarios desconectados, normalmente trabajar en un entorno conectado con el contexto es totalmente desaconsejable, póngase de ejemplo un página en ASP.NET donde es necesario encapsular normalmente los datos en sesiones o caché.

Así cuando utilizamos los métodos `InsertOnSubmit, DeleteOnSubmit y SubmitChanges` va a ser el DataContext el encargado de gestionar y resolver los posibles conflictos con registros existentes.

Por eso la existencia normalmente de varios usuarios en las aplicaciones hace realmente importante una buena gestión de la concurrencia. Resuelto el problema de la concurrencia pesimista y los bloqueos con la aparición de la API para .NET, ADO.NET se resolvió la problemática del escenario desconectado apareciendo la alternativa de la concurrencia optimista donde se permite hacer cambios a todos los usuarios en su copia de datos para posteriormente comparar y actualizar en la base de datos real. De hecho cuando se invoca `SubmitChanges`, el DataContext implementará automáticamente

concurrencia optimista, generando este la sentencia UPDATE en el servidor y gestionando los conflictos. LINQ to SQL permite gestionar las excepciones de concurrencia sobrescribiendo, no actualizando o utilizando transacciones.

```
try
{

    bd.Connection.Open();
    bd.Transaction = bd.Connection.BeginTransaction();
    bd.SubmitChanges(ConflictMode.ContinueOnConflict);
    bd.Transaction.Commit();
}
catch (ChangeConflictException)
{
    bd.Transaction.Rollback();
}
```

4.5.5. AMPLIAR LA CAPA DE NEGOCIO

Este apartado revisa algunas de las funcionalidades avanzadas que ofrece LINQ to SQL y que se están utilizando en el proyecto Agromutua.WEB. Principalmente la características que se van a presentar se centrar en mejorar el mantenimiento y la reutilización mediante el uso de clases parciales, métodos parciales y herencia de objeto.

Utilizar practicas de programación orientada a objeto añade funcionalidad al dominio de negocio al tomar estructuras base de objeto y ampliarlas para añadir la lógica personalizada de negocio. Anterior a .NET 2.0, la principal forma de ampliar estas estructuras era al heredar de una clase base. Con .NET 2.0, Microsoft incluyo el concepto de clases parciales. Con estas clases parciales, podemos aislar conjuntos específicos de funcionalidad en archivos separados y después permitir que el compilador los fusione en una sola clase.

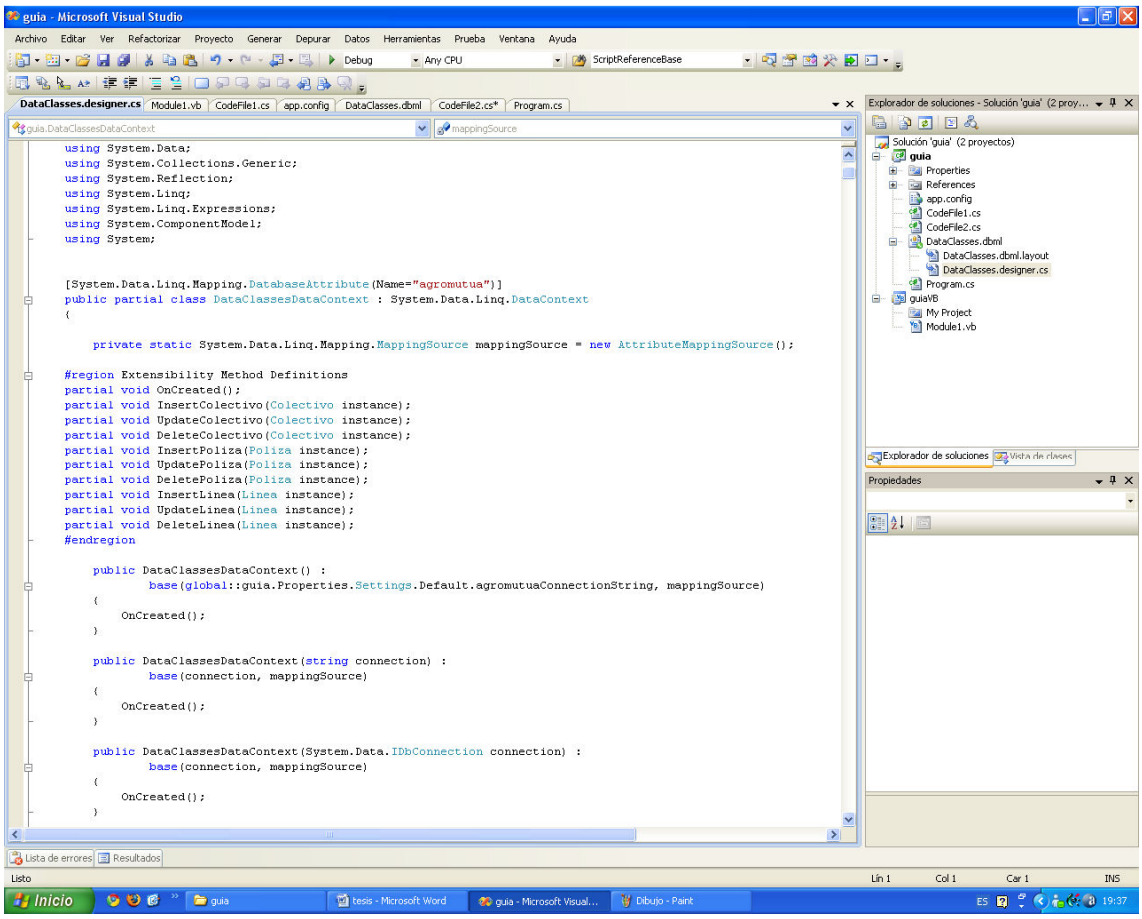
Por defecto las herramientas mencionadas en el punto 4.5.1 (el diseñador y el SqlMetal) generan las definiciones de las clases como clases parciales automáticamente, así va a permitir agregar al usuario otras clases parciales para realizar validaciones extra, por ejemplo.

Otra novedad en los lenguajes C#3.0 y VB 9.0 es la incorporación de una característica de lenguaje denominada métodos parciales para permitir incluir funcionalidad en un método.

Normalmente, cuando trabajamos con entidades de negocio tenemos que proporcionar procesado adicional como parte de un constructor o durante un cambio de propiedad. Anteriormente a C#3.0 y VB 9.0 se deberían crear clases abstractas y permitir que las nuevas clases se anulen al implementar las clases. Utilizar esta forma de herencia puede ser problemático en LINQ to SQL debido al modelo de implementación de herencia, de ahí la necesidad de disponer los métodos parciales.

Con los métodos parciales, se pueden insertar stubs de método en nuestro código generado. Si se implementa el método en nuestro código de negocio, el compilador añadirá la funcionalidad. Si no lo implementamos, el compilador eliminara el método vacio. Cuando generamos un mapeo con el diseñador de LINQ to SQL o SqlMetal se insertan en las definiciones de clase.

En la siguiente captura se ve la clase parcial con los métodos parciales que ha generado el diseñador de la tabla *Línea*.



5. LINQ EN LA ARQUITECTURA MULTICAPA

La aparición de LINQ supone una revolución en lo que se refiere con la interacción de la base de datos y el tratamiento de estos. Pero a la hora de comenzar un nuevo desarrollo al desarrollador se le añade una cuestión nueva referente al lugar que debe ocupar LINQ dentro de sus aplicaciones.

Durante los capítulos anteriores se han realizando ejemplos de pequeñas consultas y operaciones que utilizaban LINQ pero no se abordado el problema de donde incluir la interacción de LINQ con la base de datos dentro de una arquitectura.

En este capítulo se va abordar el debate de las diferentes opciones que se pueden tomar respecto al lugar de LINQ dentro del desarrollo de una aplicación multicapa, teniendo en cuenta que LINQ no solamente es una tecnología de consulta genérica sino un conjunto completo de herramientas que puede tratar con bases de datos relacionales, XML, DataSets, colecciones de objetos en memoria y muchas más fuentes de datos gracias a su ampliabilidad. Esto significa, que para el desarrollador la utilización de LINQ en su código se convierta en algo dominante.

Primero se repasará una arquitectura multicapa de 3 capas, la más básica y se verá el lugar que ocupa LINQ to SQL dentro de ella, que influirá bastante en la arquitectura por lo que es necesario para el desarrollador deba pensar en cómo utilizarla. Después se analizará donde pueden ser útiles LINQ to XML y LINQ to Objects en general.

5.1 LINQ TO SQL Y LA CAPA DE ACCESO A DATOS

EL tipo de LINQ que tiene probablemente más impacto sobre la arquitectura es LINQ to SQL. Por esta razón en este punto del capítulo vamos a analizar la típica arquitectura de tres capas y qué función desempeña LINQ to SQL en ella, que puede ser tan importante hasta el punto de reconsiderar en muchas ocasiones la propia naturaleza de una capa de acceso a datos.

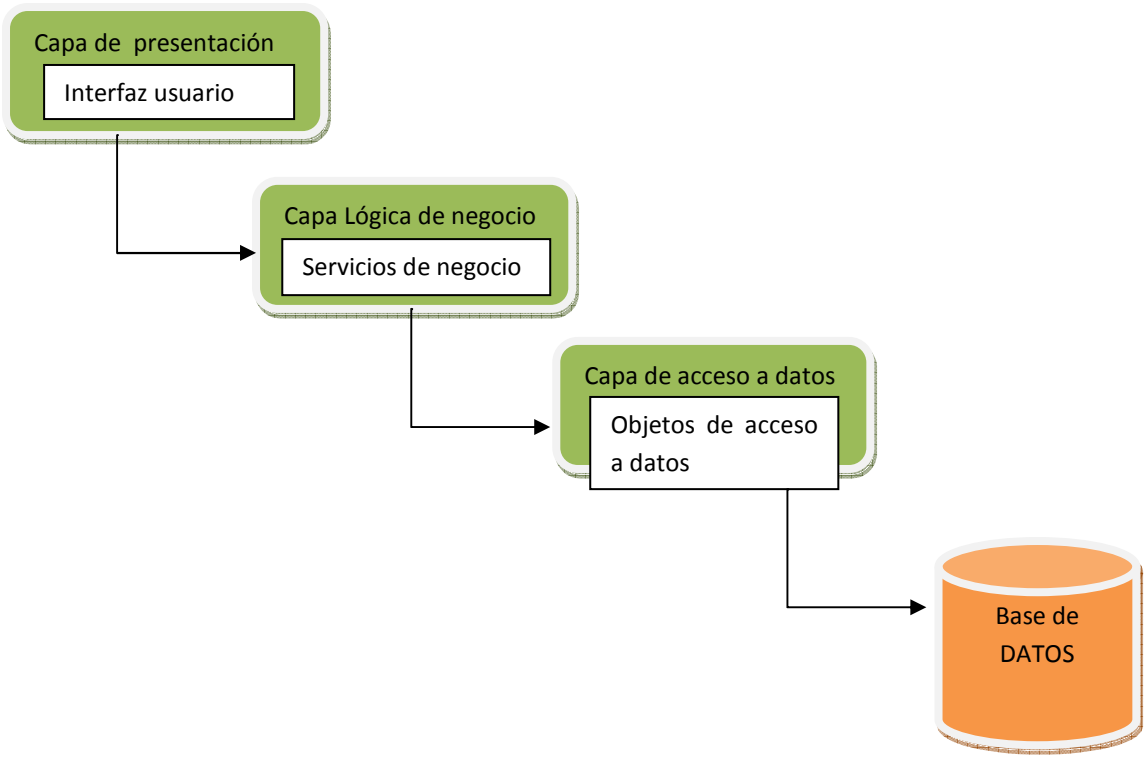
5.1.1 LA ARQUITECTURA TRADICIONAL DE TRES CAPAS

Aunque muchas aplicaciones en una arquitectura multicapa se dividen en más de tres capas, en este punto nos servirá para exponer el impacto de LINQ to SQL con una arquitectura básica de tres capas.

Una arquitectura de tres capas es cualquier sistema que fuerza una separación general entre las siguientes tres partes:

- **La capa de presentación**: gestiona el nivel más alto de la aplicación, la interfaz de usuario. La función principal de esta capa es traducir tareas y resultados en algo que el usuario pueda entender. La capa de presentación posee componentes necesarios para interactuar con el usuario de la aplicación: páginas web o formularios de cliente entre otros.
- **La capa lógica o capa de negocio:** Coordina la aplicación, procesa comandos, toma decisiones y evalúa lógicas y lleva a cabo cálculos. También mueve y proceso datos entre las dos capas de alrededor.

- **La capa de acceso a datos o persistencia:** almacena y recupera información de una base de datos, sistema de archivo o cualquier otro almacén. La información se pasa a la capa de lógica de negocio para procesarla.



La idea principal de dividir la aplicación en varias capas es separar las funcionalidades comunes para agrupar intereses y optimizar independientemente de otras funciones. De esta forma un fallo en determinada capa solo afectara a esta y no hará que otras capas fallen y en general facilitar entender, diseñar y gestionar complejos interdependientes que se solapen lo menos posible.

5.1.2. La capa de acceso a datos y LINQ to SQL

Durante el desarrollo de la tesina se han ido incluyendo ejemplos de pequeñas consultas LINQ to SQL, pero nunca se ha hecho referencia a su utilización en una arquitectura. Eso es porque hasta ahora se ha utilizando LINQ to SQL de una forma RAD (*Rapid Application Development*, o Desarrollo Rápido de Aplicaciones). El RAD es una metodología que implica un desarrollo de aplicaciones iterativo y de construcción de prototipos.

En este punto se va a explicar que cuando un desarrollador decide utilizar LINQ to SQL puede seguir una o dos direcciones: bien utilizar LINQ to SQL de forma transparente en toda su aplicación haciendo uso de él cuando lo vea necesario (forma RAD que se ha indicado) o puede seguir el modelo tradicional de tres capas con acceso a datos que se construye utilizando LINQ to SQL.

Con la primera opción el programador tendrá todos los beneficios de LINQ to SQL, con la segunda opción obtiene los beneficios de tener una capa de acceso a datos bien definida.

Se describirá a continuación cada opción por separado.

5.1.2.1. UTILIZAR LINQ TO SQL COMO CAPA DE ACCESO A DATOS

Cuando incluimos consultas en cualquier parte de nuestra aplicación, LINQ to SQL está actuando como una capa de acceso a datos mínima. Las clases que generamos con el LINQ to SQL Designer y su posterior archivo .dbml son las entidades de datos que forma el modelo de objeto de datos. El código que SQL que lleva a cabo las llamadas a la base de datos para cargar o guardar datos de las entidades de datos se generan por un DataContext LINQ to SQL basado en consultas que se escriben en algún lenguaje de programación .NET.

En la presentación de este capítulo se comenta que a utilizar LINQ sin crear una capa de datos real podría obtener todos los beneficios de LINQ to SQL, a continuación se detallan dichos beneficios.

Cuando se desarrollan aplicaciones a menudo, el desarrollador se da cuenta que en cada página necesita consultas distintas o puntuales para obtener algún dato concreto. Es muy posible que esta consulta que se haga en un sitio puntual no se vaya a realizar en más partes de la aplicación. Por tanto no existe una capa de datos real que encaje en todos lados. Es decir, si existe cualquier cada de acceso a datos que tiene código genérico suficiente para satisfacer las necesidades de cada página, puede ser acosta del rendimiento.

De lo descrito en el párrafo anterior se obtiene la conclusión de que si se escribe directamente en la página la consulta puede obtener consultas que coincidan exactamente con las necesidades del programador para esa página en concreto. Reduciendo la sobrecarga de la base de datos y el tráfico de red.

Veamos un ejemplo:

```
var query = from cultivo in bd.Cultivo
            where cultivo.Descripcion.Length > 20
            select cultivo;
```

En este ejemplo se muestra un ejemplo de una consulta que podría ser utilizada solo en una página por su alto nivel de detalle, esta obteniendo solo cultivos con una descripcion mayor de 20 caracteres.

Otro beneficio a destacar, es cuando se recuperan datos de la base de datos, se pueden seleccionar facilmente solo los campos con los que se van a trabajar y así evitar cargar datos inncesarios que no se van a utilizar.

Por ejemplo y utilizando la consulta anterior, solo se quieren recuperar las lineas de los cultivos con la descripción mayor de 20 caracteres:

```
var query = from cultivo in bd.Cultivo
            where cultivo.Descripcion.Length > 20
            select cultivo.Linea;
```

Posiblemente al disponer de una capa de datos real, hubiera sido necesario obtener todo el objeto C*ultivo*, no solamente las lineas. Siempre se puede diseñar una funcion y seleccionar los datos a devolver pero esto complica el codigo y posiblemente se dispondrá de muchos metodos y funciones que pocas veces se utilizan.

Tambien, cuando realizamos las consultas genericas a servidor de base de datos, la ordenación y los agrupamientos los hara siempre de la misma forma por lo que tambien se le pueden indicar más operaciones.

```
var query = from cultivo in bd.Cultivo
            where cultivo.Descripcion.Length > 20
            orderby cultivo.Plan
            select cultivo.Linea;
```

Para resumir los beneficios que se han mencionado, se podria indicar que la genericidad tiene un coste. Lo que permite utilizar LINQ to SQL de una forma RAD es la personalizacion.

Pero teniendo en cuenta que existe una segunda posibilidad, se exponen una serie de limitaciones que supone aplicar consultas LINQ to SQL de una forma RAD:

- Escribir consultas de base de datos en su capa de presentación no es tan malo como utilizar codigo SQL en ella, pero sigue siendo una practica cuestionable. Ya que se mezcla codigo de acceso a datos con logica de negocio y codigo de presentacion, y por tanto se estan abandonando las buenas practicas de la separación en capas.
- No existiría un solo lugar donde mirar y ver todo lo realcionado con el acceso a datos, ya que todas las consultas estarían dispersas por el codigo. Por tanto cualquier modificación acarrearia dificiles tareas de actualización y mantenimiento.
- La reutilización de codigo pasa a un segundo plano, ya que no se podrían compartir consultas. Se podría enriquecer la clase DataContext con consultas predefinidas o código de validacion, pero carecería de mucho sentido comprado con una capa real de acceso a datos.
- Mapear esta limitado al modelo de tabla por clase. Si se desea tener entidades que se extienden varias tablas debser ser mejor utilizar algo diferente de LINQ to SQL o crear una capa de aplicación que abstrae esta limitación y devuelve entidades enriquecidas.

Para evitar este tipo de uso pasamos a explicar la opcion opuesta, en la que LINQ to SQL se utiliza como capa de acceso a datos real.

5.1.2.2. UTILIZAR LINQ TO SQL COMO CAPA DE ACCESO A DATOS REAL

A la hora de poseer LINQ to SQL como capa de acceso a datos en nuestros proyectos podemos disponer de verdaderos beneficios de una capa real de acceso a datos y algunos de LINQ to SQL.

La capa de persistencia o acceso a datos es la última capa de la arquitectura multinivel, así una caracterisitca de esta capa es el ocultamiento de todo tipo de datos hacia la base de datos y la tecnologia de acceso utilizada, tanto como si es una herramienta de mapeado objeto-relacional, SQL generado a mano, llamadas a procedimientos, etc. Consiguiendo mayor abstracción en capas superiores, aspecto importante en las arquitecturas multinivel.

Así la capa de acceso a datos tampoco debe imponer ninguna restriccion significativa en el diseño del modelo de objeto de negocio (o modelo de dominio).

Teniendo en cuenta los dos aspectos anteriores, dicha capa podria ser totalmente reemplazable y modificable con un impacto minimo en capas superiores, consiguiendo un mayor desacoplamiento. Se recuerda que cuando utilizabamos la forma RAD disponiamos de multiples consultas diseminadas por todo el código de la aplicación.

A la hora de crear una capa de acceso a datos LINQ del tipo LINQ to SQL se deben tener unos aspectos en cuenta:

- El ratio de codigo de ejemplo y codigo real se puede comparar con codigo directo LINQ to SQL
- Si se devuelven entidades o consultas LINQ to SQL desde su cada de acceso a datos, estos objetos soportan ejecucion diferida y búsqueda perezosa, debilitando su división.
- La capa de acceso a datos deberá devolver objetos que se pueden pasar entre componentes en diferentes capas y ser tratados debidamente en cada uno de ellas.

Referente al primer punto, es cierto que mucho código de la capa de acceso a datos puede parecer inútil porque sea sencillo o no se utilice. Sin embargo si se desea adoptar este tipo de arquitectura será algo que se deba aceptar. De hecho es mejor disponer todo ese codigo que consultas literales SQL.

Veamos el segundo punto. Cuando se escribe un metodo en la capa de acceso a datos el programador puede estar tentado de escribir metodos del estilo:

```
public class CultivoDataAccessObject
{
    private DataClassesDataContext bd = new DataClassesDataContext();

    public IQueryable<Cultivo> GetCultivoByLinea(String linea)
    {
        return from cultivo in bd.Cultivo
               where cultivo.Linea = linea
               select cultivo;
    }

}
```

Puede observarse un metodo `GetCultivoByLinea` que devuelve un tipo `IQueryable<Cultivo>`. Si como en el ejemplo se crea una capa de acceso a datos y devuelve tipos de tipo `IQueryable` o parecido se debe tener en cuenta que se estan devolviendo consultas, que es diferente a devolver colecciones del tipo `Cultivo`, en este caso. En esta situación y debido a la consulta diferida de LINQ la consulta se ejecutaria fuera de la capa de acesso a datos. Por tanto es mejor devolver ya las propias consultas enumerads y ejecutadas (Utilizando por ejemplo ToList, ToArray, etc.).

```
public class CultivoDataAccessObject
{
    private DataClassesDataContext bd = new DataClassesDataContext();

    public IQueryable<Cultivo> GetCultivoByLinea(String linea)
    {
        var query = from cultivo in bd.Cultivo
                where cultivo.Linea = linea
                select cultivo;

        return query.ToList();
    }

}
```

Si se utiliza como en este caso la carga perezosa, se producen llamadas a la base de datos por medio de llamadas transparentes a LINQ to SQL. Se pueden enviar los objetos valor con la carga perezosa desactivada utilizando la propiedad `DeferredLoadingEnabled = false` del `DataContext`.

Por tanto devolver objetos de valor es importante cuando necesitamos pasar objetos entre capas remotas.

5.1.3. LINQ TO XML Y LINQ TO OBJECTS EN LA ARQUITECTURA

El uso de LINQ to XML y LINQ to Objects en una arquitectura es indiferente y no importara en que parte de las aplicaciones esten situadas.

LINQ to XML se puede utilizar para leer o crear XML. Dado el amplio uso de XML en estos momentos, se puede esperr encontrar LINQ to XML empleado en todas las capas de aplicaciones. Por ejemplo en combinacion con datos de una base de datos, para importar datos XML, crear RSS feeds y más.

Por otra parte, y en general, LINQ to Objetcs se va a utilizar en combinacion con LINQ to XML y LINQ to SQL y por tanto va a estar presente en todas las capas. Además la importancia de LINQ to Objects va a ser importante como ya hemos comentado en los primeros capitulos de esta tesina en los desarrollos que se hagan apartir de ahora utilizando el Framework 3.5 y posteriores ya que nos permitira reducir codigo y manejar colecciones de formas muy sencillas y intuitivas, ya que el desarrollador en cualquier momento va tener la necesidad de consultar tablas, colecciones, listas o diccionarios y encontrara en LINQ to Objects una herramienta útil.

6. MEJORAS Y AVANCES FUTUROS

En este capítulo se van a mostrar aspectos importantes que el desarrollador debe tener en cuenta durante la construcción de la aplicación. Los problemas mencionados, en muchas ocasiones no son problemas sino más bien son problemas surgidos fruto de un uso incorrecto de LINQ y que se debe corregir o hacerlo de forma correcta, en otras ocasiones no y es que cuando se trabaja con LINQ hay que tener en cuenta que existe un desajuste de paradigma entre la programación orientada a objetos y las bases de datos relacionales, que de alguna forma se tiene que salvar.

6.1. ASPECTOS A MEJORAR EN LINQ

En ocasiones cuando se usa LINQ se pueden encontrar con muchos aspectos que posiblemente necesiten una revisión en ediciones futuras, pero el hecho de ser criticados en esta tesina no significa que estén mal desarrollados, simplemente que durante el desarrollo de nuestra aplicación de Agromutua.Web nos han producido problemas.

6.1.1. PROBLEMAS COMUNES

Para mencionar todos los problemas encontrados, se introducirá el problema y debajo de él se dará la explicación. Alguno de estos problemas son producidos por el desajuste de paradigma que se aborda en el punto 6.1.2.

LINQ parece estar diseñado en ocasiones para una arquitectura de dos niveles.

En ocasiones el programador de la aplicación tiende a olvidarse de una arquitectura de tres niveles, comenzado a diseminar por la aplicación consultas y operaciones de la base de datos, que a la larga pueden acarrear problemas. LINQ facilita mucho el desarrollo rápido pero se debe tener en cuenta la arquitectura que se emplea y no salirse de ella.

LINQ to SQL genera SQL dinámico. La tecnología motiva el uso de SQL dinámico en vez de procedimientos almacenados.

En ocasiones un procedimiento almacenado puede resolver un problema de una gran consulta o un problema relacionado con el rendimiento de esta. LINQ realiza un enfoque para utilizar siempre el uso de SQL dinámico, es decir, construcciones de consultas que variaran según condiciones en vez de utilización de procedimientos almacenados con tablas temporales o SQL dinámico en el propio procedimiento almacenado.

El problema del desajuste de la cache. LINQ to SQL carga una vez los objetos en la cache conduciendo en ocasiones a resultados inesperados.

Más que un problema es una característica de LINQ to SQL, el DataContext de LINQ utiliza una carga de datos por las claves primarias Por lo que esto puede acarrear problemas con aserciones y cabe la necesidad de siempre que se desee actualizar registros deberá disponer la tabla de claves primarias. De lo contrario la actualización se reflejará con éxito pero el cambio en la base de datos realmente no se realizará.

Métodos `First()` y `Single()` ocasionan problemas cuando no obtienen resultados esperados.

Estos métodos ejecutan la consulta LINQ y nos devuelven en el caso de `First()` el primer resultado, y en caso de `Single()` el único resultado. Pero los problemas vienen cuando `First()` no obtiene ningún resultado o `Single()` no obtiene ningún resultado o obtiene más de uno. Lógicamente se produce un error en tiempo de ejecución que deberá ser tratado, pero a simple vista parecen innecesario que esto produzca errores, existiendo métodos como `FirstOrDefault()` o `SingleOrDefault()` que devuelven valores nulos cuando se dan las situaciones mencionadas. En futuras versiones de LINQ (en el Entity Framework) está previsto eliminar los métodos para evitar confusiones.

Relaciones "muchos a muchos" y asociaciones

Como veremos en el siguiente punto, el desajuste del paradigma entre la orientación a objetos y las base de datos relacionales producen en ocasiones una inestabilidad dentro de los modelos equivalentes que se supone que nos proporciona el mapeo.

El choque producido entre filas de datos unidas por identificadores de columna y construcciones de memoria que contienen colecciones de objetos que pueden contener adicionalmente otras colecciones de objetos y así sucesivamente van a producir problemas a la hora de obtener los resultados con LINQ, al igual que la imposibilidad de representar una relación "muchos a muchos" en el paradigma de la orientación a objetos, ya que en las bases de datos relacionales se representa con una tabla intermedia. En el punto siguiente se analizará el desajuste del paradigma.

6.1.2. EL DESAJUSTE DE PARADIGMA

El hecho de que los desarrolladores modernos tengan que trabajar simultáneamente con lenguajes de programación de aplicación general, datos relaciones, SQL, documentos XML, etc., significa que necesitamos dos cosas:

- Ser capaces de trabajar con cualquiera de estas tecnologías o lenguajes de una forma individual.
- Mezclar y hacerlas coincidir para crear una solución robusta y coherente.

El problema surge cuando la programación orientada a objetos, el modelo de bases de datos relacional y XML, entre otros, no se crearon originalmente para trabajar juntos. Por tanto representan diferentes paradigmas que no interactúan bien entre ellos.

Traducir un modelo de una base de datos relacional en código de lenguaje de programación requiere, a menudo, un trabajo tedioso. El problema general que LINQ aborda ha sido indicado por Microsoft como "Datos != Objetos", y más específicamente para LINQ to SQL como "Datos relacionales != Objetos", igual con LINQ to XML, por ejemplo, "Datos XML != Objetos" y por lógica "Datos XML !=Datos relacionales".

A continuación se muestran algunos de los desajustes existentes entre el paradigma orientado a objetos y el paradigma relacional:

- **Las bases de datos relacionales y los lenguajes orientados a objetos no comparten el mismo conjunto de tipos primitivos de datos:** Un ejemplo de esto son las cadenas que normalmente en las bases de datos tienen una longitud preestablecida y en los lenguajes como C# y VB.Net no.

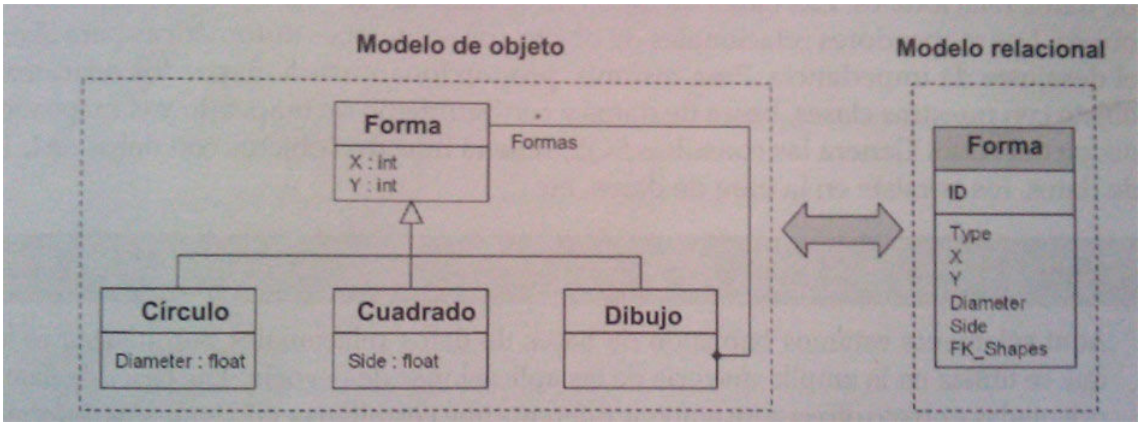
- **OOP y las teorías relacionales vienen con diferentes modelos de datos:** Normalmente por razones de rendimiento principalmente las bases de datos están normalizadas. La normalización es un proceso que se encargar de la eliminación de la redundancia, organiza los datos eficientemente y reduce el potencial por anomalías durante las operaciones de datos y mejora la consistencia de datos. Las bases de datos están normalizadas en tablas y relaciones, mientras los objetos utilizan herencia, composición y gráficos de referencia complejos. La equivalencia entre ambos requiere el uso de "trucos" adicionales por el programador.

- **Modelos de programación:** En SQL escribimos consultas y así obtenemos una forma declarativa de más alto nivel de expresar el conjunto de datos en los que estamos interesados. Con los lenguajes de programación imperativos como C# o VB.NET tenemos que escribir bucles y sentencias.

- **Encapsulación**: Los objetos son autónomos y tienen un comportamiento. En las bases de datos, los registros de datos no tienen comportamiento de por sí. En las bases de datos el código y los datos están claramente separados.

Este desajuste es el resultado de las diferencias entre una base de datos relacional y la jerarquía típica de clase orienta a objetos.

Veamos un ejemplo para demostrar el desajuste de paradigma:



Los conceptos como herencia o composición no se pueden soportar directamente por las bases de datos relacionales, lo que significa que no podemos representar los datos de la misma forma en los dos

modelos. El problema es que en algún punto necesitamos hacer que el modelo de objeto y el modelo relacional trabajen conjuntamente. Esto no es una tarea sencilla puesto que los lenguajes de programación orientados a objetos y .NET implican clases de entidad, reglas de negocio, relaciones complejas y herencia, mientras que una fuente de datos relacional implica tablas, filas, columnas y claves primarias y externas.

Ninguna solución a día de hoy es perfecta y los mapeadores relacionales de objeto podrían mejorar. Algunas de sus limitaciones principales incluyen lo siguiente:

- Un buen conocimiento de las herramientas es necesario antes de ser capaz de ser utilizadas de forma eficiente y evitar problemas de rendimiento.
- Un uso óptimo requiere conocimiento sobre base de datos relacionales.
- Las herramientas de mapeado no son siempre tan eficientes como el código de acceso a datos escrito a mano.
- No todas las herramientas vienen con soporte para la validación en tiempo de compilación.

6.2. EL FUTURO DE LINQ

Las novedades de LINQ surgirán a partir de la nueva versión de Microsoft Visual Studio 2010, que vendrá con el nuevo Framework 4.0. Por lo que respecta a LINQ las últimas novedades surgidas, entre ellas PLINQ, ya se pudieron apreciar en la Parallel Extension y en el SP1 del Framework 3.5, por lo que no se esperan grandes cambios. A día de hoy es una incógnita.

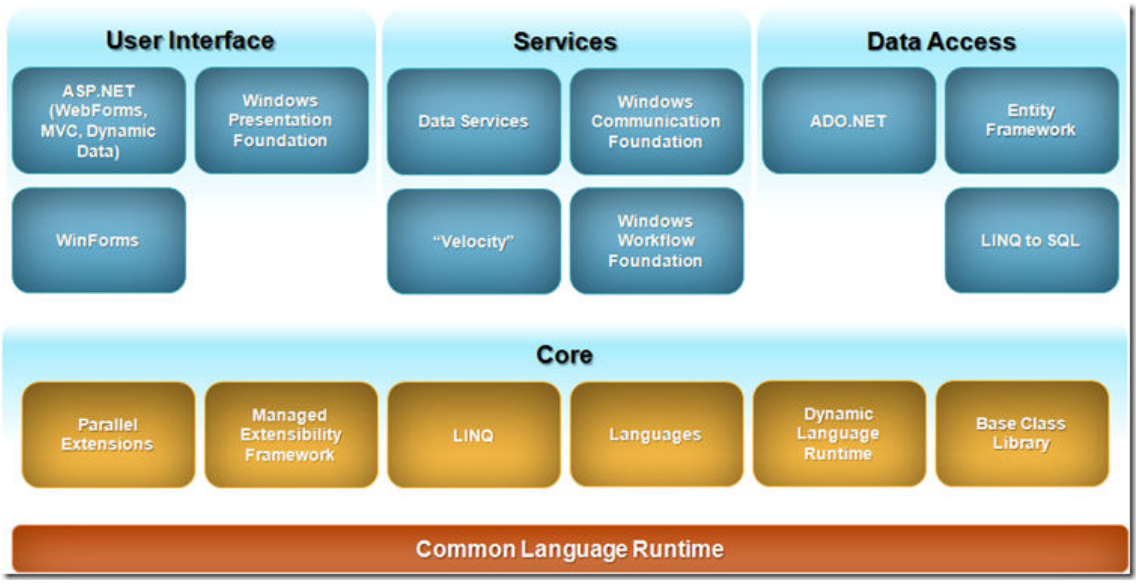
6.2.1 VISUAL STUDIO 2010 Y FRAMEWORK .NET 4

La web de Visual Studio 2010 [15] nos permite observar las nuevas novedades que traerá la nueva versión del producto para desarrolladores más importante de Microsoft:

- Experiencia de usuario mejorada: nueva y mejorada UI, menor complejidad, editores mejorados, mejorado el soporte de ventanas y documentos así como la focalización de documentos y animaciones.
- SharePoint Development: mayor integración de Sharepoint en Visual Studio, permitiendo personalizarlo y agregando opciones para trabajar más ágilmente.
- Democratización de la administración de ciclo de vida de las aplicaciones: nuevas capacidades para Microsoft Visual Studio Team System 2010 como gestores avanzados y evaluadores, así como nuevos diagramas, pruebas y control de versiones.
- Seguir facilitando las tareas a los programadores: compatibilidad con versiones anteriores de Visual Studio, desarrollo intuitivo, actualizar las aplicaciones a Windows 7, entre otras.
- Desarrollo de Cloud Computing: utilización de herramientas de Windows Azure.
- Desarrollo Web: mejorado el motor Intellisense para adaptarse a los estándares JavaScript, publicar de forma rápida los sitios web y completo soporte para Silverligth.
- Programación paralela: simplificación de las tareas de programación de paralela con soporte IDE, nuevas bibliotecas de tareas, extensiones paralelas para .NET Framework ya incluidas, concurrencia en tiempo de ejecución y herramientas para su depuración.
- Mayor soporte a las bases de datos: ahora ya no será SQL Server la única base de datos con la que se podrá trabajar y se podrán utilizar otras como Oracle o IBM DB2.

Por lo que respecta a las nuevas características concretas del Framework .NET 4 que se podrán apreciar respecto a sus antecesoras son:

- . NET Framework 4 trabaja codo con codo con el Framework versión 3.5 SP1. Las aplicaciones que se basan en las versiones anteriores del Framework seguirá funcionando en esta versión. Sólo un subconjunto de la funcionalidad es compartida por todas las versiones del Framework.
- Innovaciones en los lenguajes Visual Basic y C #, por ejemplo, declaraciones lambdas, continuaciones de líneas implícitas, envío dinámico, y llamada / parámetros opcionales.
- El ADO.NET Entity Framework, nuevas características para los desarrolladores que trabajan con bases de datos relacionales con una mayor nivel de abstracción, incluye soporte para POCO, test-driven, nuevos operadores LINQ y funciones de modelo.
- Mejoras de ASP.NET:
    - Nueva interfaz de usuario
    - Plantillas JavaScript con capacidad de enlace de datos para Ajax. Nuevo control de gráfico de ASP.NET.
- Mejoras en WPF:
    - Añadido soporte en Windows Presentation Foundation (WPF) para Windows 7 Multi-Touch, los controles de la cinta, y las características de extensibilidad de la barra de tareas.
    - Se ha agregado soporte de WPF para la superficie 2.0 SDK.
    - Nueva línea o de negocio controles y gráficos de control, smart edit, data grids, y otras que mejoran la experiencia de los desarrolladores que crean aplicaciones centradas en datos.
    - Mejoras en el rendimiento y la escalabilidad.
    - Mejoras visuales en la claridad de texto, diseño de píxel de romperse, la localización y la interoperabilidad.
- Mejoras de Windows Workflow (WF) que permiten a los desarrolladores mejorar la interaccion con los flujos de trabajo. Estos incluyen un modelo mejorado de programación de la actividad, una experiencia de diseño mejorado, un nuevo estilo de modelado de diagrama de flujo, una paleta de mayor actividad, las normas de flujo de trabajo de integración y características de la nueva correlación de mensajes. . NET Framework también ofrece importantes mejoras de rendimiento para flujos de trabajo basados WF.
- Mejoras a Windows Communication Foundation (WCF), como el apoyo para los servicios de WCF programas de flujo de trabajo que permitan el flujo de trabajo con las actividades de mensajería, soporte de correlación, duradera comunicación de dos vías y las amplias capacidades de acogida. Además,. NET Framework 4 proporciona nuevas características de WCF, como el descubrimiento de servicios, servicio de router, configuración simplificada y una serie de mejoras de cola, el resto solo, diagnósticos y de rendimiento.
- Nuevas e innovadoras características de programación en paralelo, como el apoyo loop paralelo, Grupo de la Biblioteca Paralelo (TPL), LINQ Paralelo (PLINQ), y las estructuras de coordinación de datos que permiten a los desarrolladores aprovechar la potencia de los procesadores multi-núcleo.



6.2.2. PARALLEL LINQ (PLINQ)

Parallel Language Integrated Query (PLINQ), también conocido como Parallel LINQ, se trata de una implementación de LINQ to Objects que ejecuta consultas en paralelo. PLINQ está basado en Parallel Extensions de .NET Framework y actualmente se debe descargar para poderlo utilizar [16] (a partir de Visual Studio 2010 ya vendrá con el entorno).

PLINQ solo puede utilizarse con consultas que se ejecuten en memoria, así como aquellas sobre proveedores IQueryable y requiere el conocimiento de las implicaciones de la programación multihilo, así como los efectos secundarios de la ejecución de consultas, como el acceso de escritura a objetos compartidos.

Para empezar se debe tener instalado el paquete de Parallels Extensions de .NET Framework y si todo ha ido correctamente se debe poder importar el ensamblado `System.Threading` que contiene toda al implementación de PLINQ.

Ahora ya se puede utilizar los métodos y las clases que vienen con PLINQ, para ello se comentará lo que trae esta extensión de LINQ y la utilización.

6.2.2.1 LOS MÉTODOS Y LAS CLASES DE PARALLELS EXTENSIONS

Existen tres métodos que utilizaremos para ejecutar las consultas en paralelo:

- `Parallel.For`
- `Parallel.ForEach`
- `Parallel.Invoke`

Veamos la utilización de los métodos `Parallel.For` y `Parallel.ForEach` en comparación con la utilización normal:

```
//*******************************************************//
/////////////////METODOS FOR/////////////////////////////
//*******************************************************//
static void SequentialFor()
{
    Stopwatch sw = Stopwatch.StartNew();
    for (int index = 0; index < 100; index++)
    {
        ProcessData(index);
    }
    long elapsed = sw.ElapsedMilliseconds;
    Console.WriteLine("Sequencial For: {0} milisegundos",elapsed);
}

static void ParallelFor()
{
    Stopwatch sw = Stopwatch.StartNew();
    Parallel.For(0, 100, (index) => { ProcessData(index); });
    long elapsed = sw.ElapsedMilliseconds;
    Console.WriteLine("Parallel For: {0} milisegundos", elapsed);
}

//*******************************************************//
/////////////////METODOS FOREACH/////////////////////////
//*******************************************************//
static void SequentialForEach(int[] data)
{
    Stopwatch sw = Stopwatch.StartNew();
    foreach (int value in data)
    {
        ProcessData(value);
    }
    long elapsed = sw.ElapsedMilliseconds;
    Console.WriteLine("Sequencial ForEach: {0} milisegundos", elapsed);
}

static void ParallelForEach(int[] data)
{
    Stopwatch sw = Stopwatch.StartNew();
    Parallel.ForEach(data, (value) => { ProcessData(value); });
    long elapsed = sw.ElapsedMilliseconds;
    Console.WriteLine("Parallel ForEach: {0} milisegundos", elapsed);
}
```

Cuando utilizamos este código en una maquina de más de un núcleo la ejecución paralela es más rápida, ya que ejecuta diferentes hilos para invocar al método `ProcessData` que se supone que es un proceso muy pesado, mientras sí una maquina es de un solo núcleo la ejecución en paralelo es más lenta.

El uso de los métodos `Parallel.For` y `Parallel.ForEach` requiere que la operación de cada ciclo sea independiente de las otras. En el caso de que el código ejecutado durante varios ciclos tuvo que almacenar resultados en un objeto compartido, por ejemplo una lisa, tendrá que asegurarse de que el acceso a el objeto compartido está protegido contra concurrencia.

Por otra parte existe el método `Parallel.Invoke` que recibe una matriz de delegados que pueden ejecutarse en cualquier orden basándose en los recursos disponibles

```
//*******************************************************//
//////////////////METODO DO//////////////////////////////////
//*******************************************************//
static int Operation(int index)
{
    ProcessData(index);
    Console.WriteLine("Operation {0}",index);
    return index * 10;
}

static void DemoDo_Sequencial()
{
    Console.WriteLine("Llamada sequencial");
    Operation(1);
    Operation(2);
    Operation(3);
    Operation(4);
}

static void DemoDo_Parallel()
{
    Console.WriteLine("Llamada paralela");
    Parallel.Invoke(
        () => Operation(1),
        () => Operation(2),
        () => Operation(3),
        () => Operation(4));
}
```

Observamos el resultado:

La implementación de `Parallel.Invoke` crea varias instancias de la clase Task, sin exponerlas al control de los programadores.

La clase `Task`, encontrada en `System.Theading.Tasks.Task` representa una operación asindrona. Es un contenedor de un delegado que incluye información como el fin de la ejecución (`IsCompleted`, `IsCanceled`) o bien métodos para detener la ejecución de una clase (`Cancel`).

La clase `Task` es útil cuando queremos contralar operaciones asíncronas con la posibilidad de cancelar una o más de ellas, sin tener que implementar una gestión personalizada para cada contexto concreto.

De la clase `Task`, hereda la clase `Future<T>` para añadir significado semántico para leer el valor de una operación asíncrona sin tener que reescribir el código de sincronización. Al usar `Future<T>` se ocultan la mayoría de detalles relacionados con la sincronización de hilos, los cuales son necesarios cuando una operación asíncrona tiene que devolver datos de otro hilo.

En relación a todo lo anterior y al trabajar con la concurrencia en .NET hay que tener en cuenta que un fragmento de código es seguro si funciona correctamente durante la ejecución simultanea de varios hilos. Sin embargo, la mayor parte de las clase de .NET no están diseñadas por defecto para su uso en entornos multihilo. Al usar Parallel Extensions de .NET Framework se tiene que sincronizar ese tipo de acceso. EL enfoque más adecuado es escribir código que no comparta datos, aunque esto no siempre es posible.

6.2.2.2 USO DE PLINQ

PLINQ es una implementación de LINQ to Objects que ejecuta consultas en paralelo utilizando para ello las clases de Parallels Extensions.

Para usar PLINQ solo se debe usar el método de extensión `AsParallel()` sobre el origen de datos:

```
int[] numbers = {1,5,8,5,8,52,8,5,4,8,4,87,45,7,8};

// Standard
var oddNumbers = from i in numbers
                 where i % 2 == 1
                 select i;
// Parallel
var oddNumbersP = from i in numbers.AsParallel()
                  where i % 2 == 1
                  select i;
```

Por tanto el escaneo de la matriz de enteros se dividirá en varios hilos, usando más núcleos de la CPU que el enfoque secuencial.

`AsParallel()` dispone de diferentes versiones que le permiten configurar sus argumentos: número máximo de hilos, preservación de orden, añadir métodos, etc.

Hasta el momento se ha visto una de las tres formas que dispone PLINQ de procesar una consulta, a continuación se describirán con detalle:

- Procesado de tubería:

El procesado de tubería es el modelo de proceso por defecto. Hay un conjunto de hilos que ejecutan la consulta y un hilo aparte que consume el resultado de dicha consulta. Este hilo procesa los datos tan pronto como los otros hilos comienzan a generar resultados, este es el comportamiento por ejemplo del bucle `Parallel.Foreach`

- Procesado Stop-and-Go:

Al utilizar este tipo de procesado el hilo que va a ejecutar los datos espera que los hilos que ejecutan el procesado terminen su trabajo antes de pasar a usar los datos. De esta manera no se produce una competencia entre los diferentes hilos para utilizar los recursos de la CPU.

Cuando se utiliza un ToArray o un ToList el procesado Stop-and-Go es automático.

- Enumeración invertida:

En ocasiones es necesario conseguir un mayor grado de paralelización invocando el código del hilo que utiliza los resultados en el mismo hilo que genera un elemento durante la ejecución de la consulta. Para ello se utiliza el método de extensión `ForAll`.

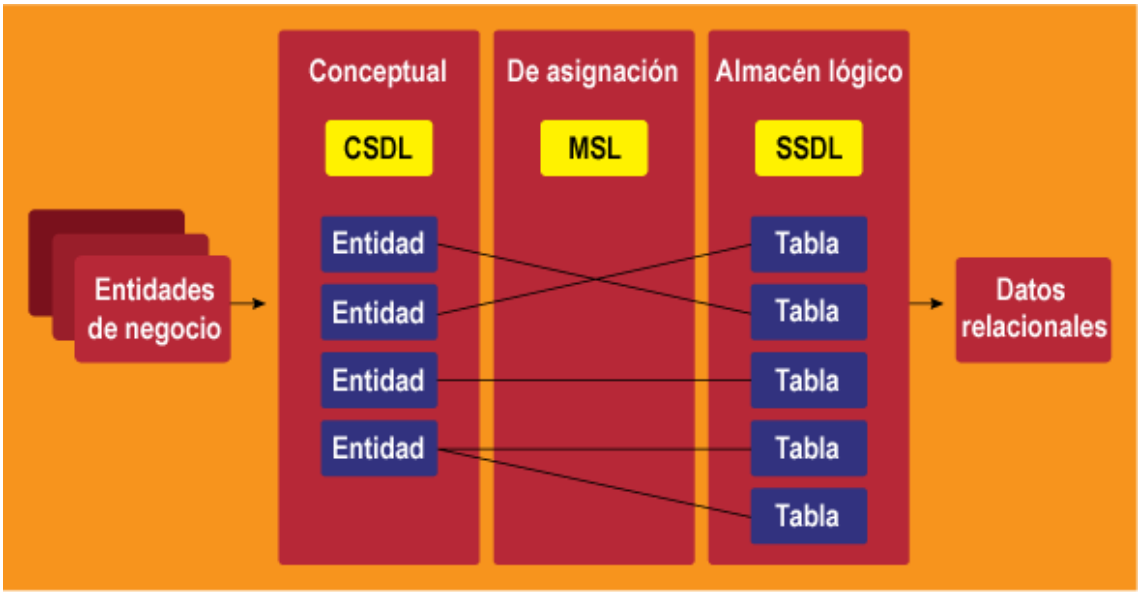
Se han enumerado los posibles procesados de las consultas en paralelo, esto puede acarrear efectos secundarios y uno de ellos es el orden de resultados, al no ser secuencial el proceso los resultados pueden obtenerse desordenados, para corregir esto podemos utilizar en nuestras consultas el operador `orderby` de LINQ, otra posibilidad es usar el argumento `ParallelQueryOptions.PreserveOrdering` en la invocación de `AsParallel()`.

6.2.3. ADO.NET Entity Framework

El ADO.NET Entity Framework surge después de LINQ, oficialmente en el SP1 del Framework .NET 3.5 y viene a ser un ORM que ofrece una tecnología para relaciones de mapeado más avanzadas.

LINQ to SQL cubría las necesidades básicas de CRUD pero en determinadas situaciones de bases de datos más grandes, la base de datos se desarrollará mejor con una estructura más normalizada.

Cuando en LINQ to SQL realizamos una vista perdemos los metadatos necesarios para actualizar registros en las tablas originales. Esto lo soluciona ADO.NET Entity Framework con la incorporación del EDM (Entity Data Model o Modelo de Datos de Entidades) que incluye información conceptual sobre la fuente de datos para que las herramientas tengan la información suficiente creación de entidades de modelos. El EF separa los modelos físico del lógico al utilizar unas series de archivos de mapeado basados en XML: CSDL, MSL y SSDL.



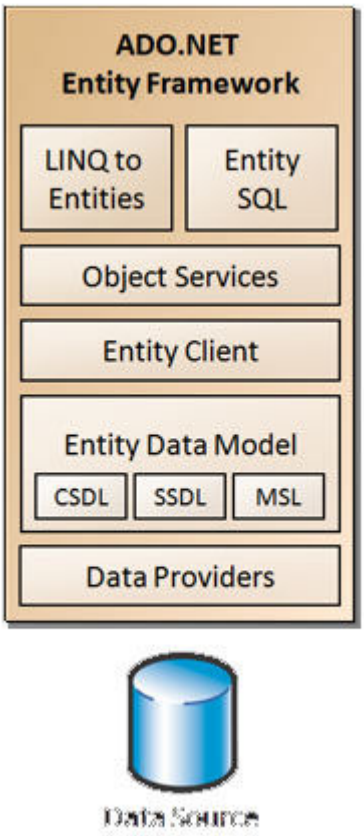
Las entidades lógicas se definen mediante un archivo SSDL (Store Schema Definition Language) basado en XML. Estos mapeados son similares a los que se definen en LINQ to SQL.

EL EF se mueve más allá de LINQ to SQL para utilizar otro archivo basado en XML, el MSL (Mapping Schema Language) para mapear el modelo lógico con un modelo conceptual. El modelo conceptual es otro archivo XML utilizando un CSDL (Conceptual Schema Definition Language). Estas entidades conceptuales se pueden convertir además en objetos fuertemente tipados si se desea.

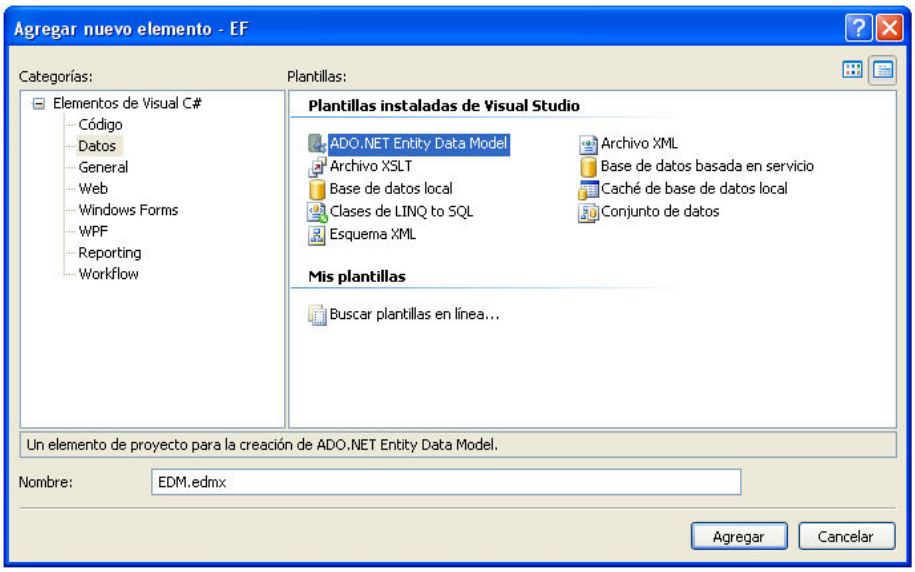
Con el EDM establecido, podemos consultarlo con un lenguaje de consulta basado en cadenas denominado Entity SQL. Además, se pueden aplicar conocimientos de LINQ utilizando LINQ to Entities sobre el EDM. Puesto que el EDM supone una verdadera capa de abstracción entre la aplicación y la base de datos, podemos modificar nuestra base de datos y el archivo de mapeado EDM y reestructurar el almacén de datos sin tener que recompilar la aplicación.

La separación de capas que obtenemos de EDM permite una separación mayor entre lo físico y lo lógico. Esto nos permite cambiar nuestro modelo de datos y estructuras de mapeado y dejar nuestra aplicación intacta. Adicionalmente, puesto que EF se ha construido sobre el modelo de proveedor ADO existente, el EF puede trabajar contra almacenes de datos distintos a SQL Server.

A continuación se muestra una figura de la arquitectura del EF (véase la explicación de esta en el punto 6.2.3.1.):

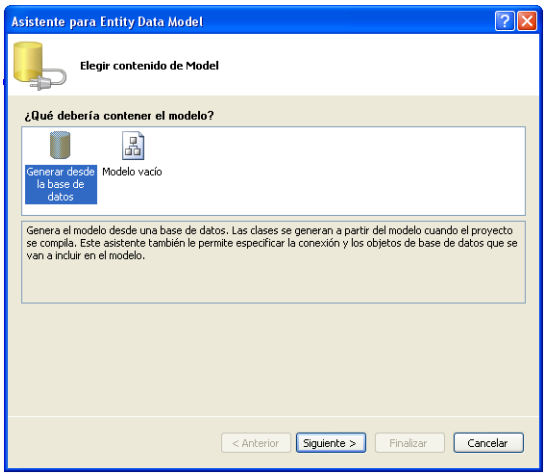


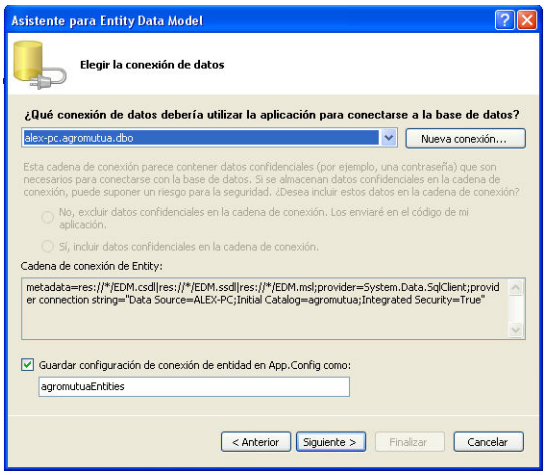
ADO.NET Entity Framework cuenta con un asistente y un diseñador integrados en Visual Studio 2008, los cuales resultan de utilidad para crear esquemas conceptuales de la nada, o bien a partir de una capa de persistencia de la base de datos soportada.



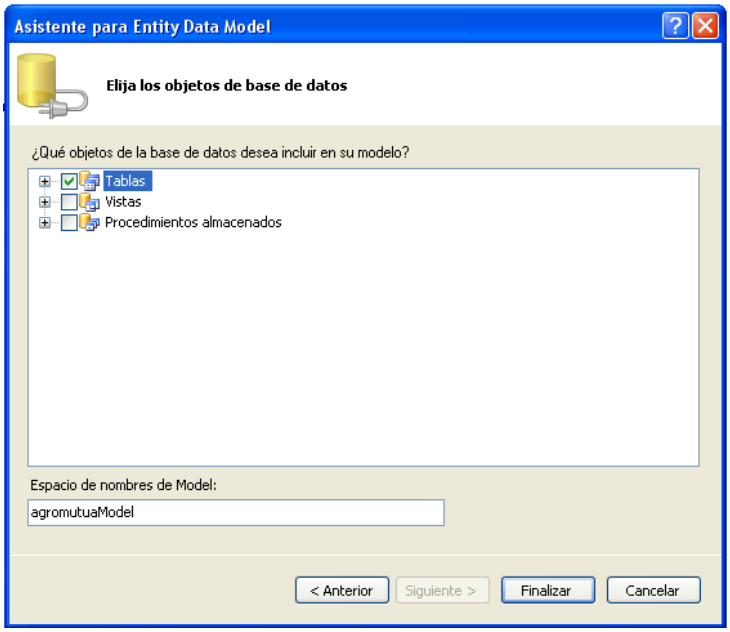
Lo primero que se debe hacer es crear un nuevo elemento de Datos del tipo ADO.NET Entity Data Model, en la figura de la parte superior se puede apreciar.

El siguiente paso pedirá al usuario si desea crear un EDM vacío o de una base de datos, en el ejemplo se ha aprovechado y se ha seleccionado "Generar desde la base de datos" ya que se dispone de la base de datos de Agromutua, que seleccionaremos en el siguiente paso.



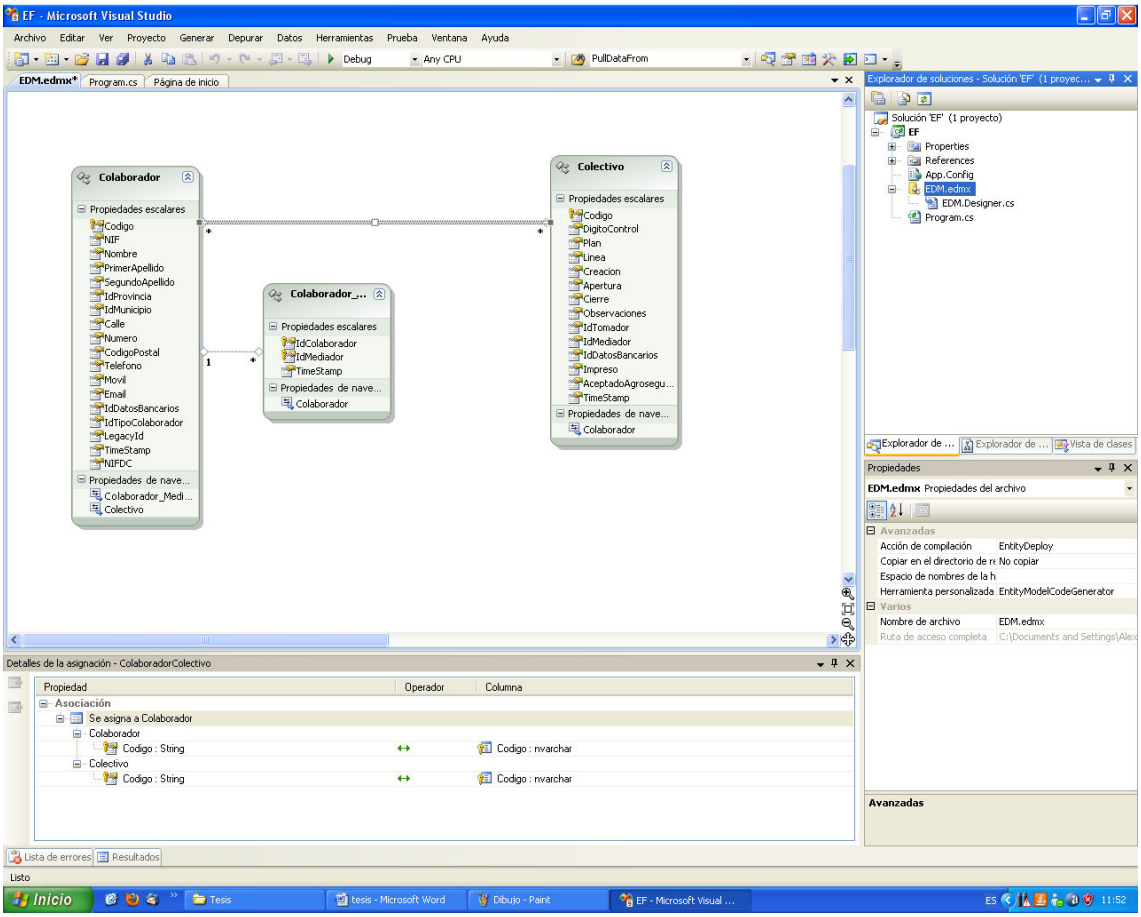


A continuación, una vez se conecta el asistente con la base de datos nos permite seleccionar las tablas, vistas y procedimientos almacenados de dicha base de datos para poder crear a partir de estos el EDM.



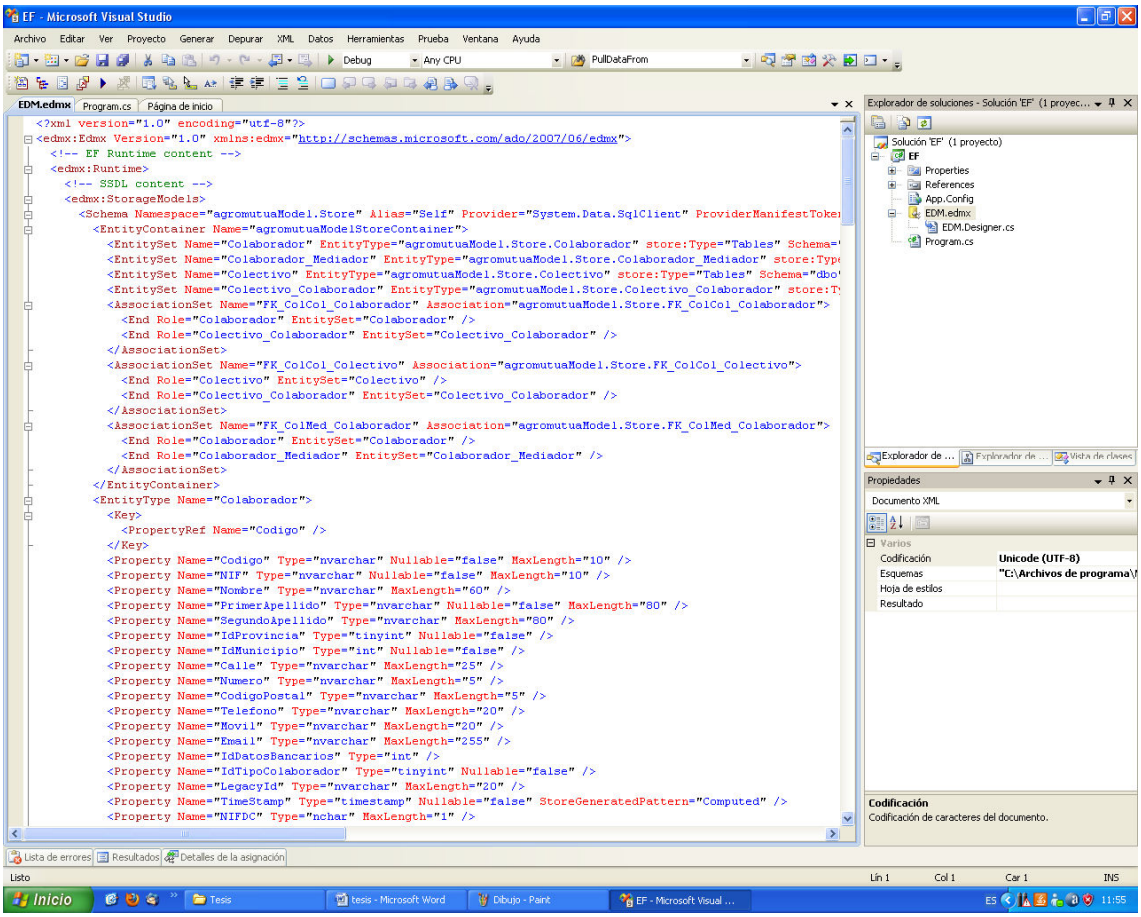
El resultado de ejecutar el asistente es un archivo .EDMX que, básicamente, es un archivo XML, a nivel interno, representa el conjunto de archivos de modelado (CSDL,SSDL y MSL).

El editor visual nos permite realizar los cambios pertinentes en las relaciones, tablas, etc. y de esta forma modelar las entidades.



Una de los aspectos a destacar es que si se dispone de dos tablas con una relación de muchos a muchos, el diseñador omite esa tabla y representa la relación subyacente en el esquema del modelo de entidades.

Si abrimos dicho archivo con el editor XML podemos observar que efectivamente dispone de las descripciones CSDL, SSDL y MSL:



Ahora ya se podría ejecutar una consulta sobre el nuevo modelo creado:

```
EntityConnection cn =
    new EntityConnection(ConfigurationManager.ConnectionStrings["CSagromutuaEntities"].ConnectionString);

EntityCommand cmd = new EntityCommand("SELECT Nombre, PrimerApellido FROM Colaborador",cn);

cn.Open();

DbDataReader dr = cmd.ExecuteReader(CommandBehavior.SequentialAccess);

    while (dr.Read())
    {
        Console.WriteLine((DbDataRecord)dr.GetValue(0));
    }
}
```

La gestión de datos que realiza ADO.NET Entity Framwork se realiza de una forma transparente mostrando el estado de los objetos involucrados en las operaciones de modificación de las instancias de las entidades. La clase `EntityObject` se trata de la clase padre, de la cual se hereda cada tipo de entidad definida en el EDM. `EntityObject`, junto con `ObjectContext`, forman parte del componente `ObjectServices`. Este componente también forma parte del EF, y es implementado mediante los espacios de nombres `System.DataObjects y System.Data.Objects.DataClasess`, los cuales permiten la ejecución de consultas, asi como la edición, actualización y eliminación de entidades con un enfoque que utiliza Entity SQL y LINQ to Entities.

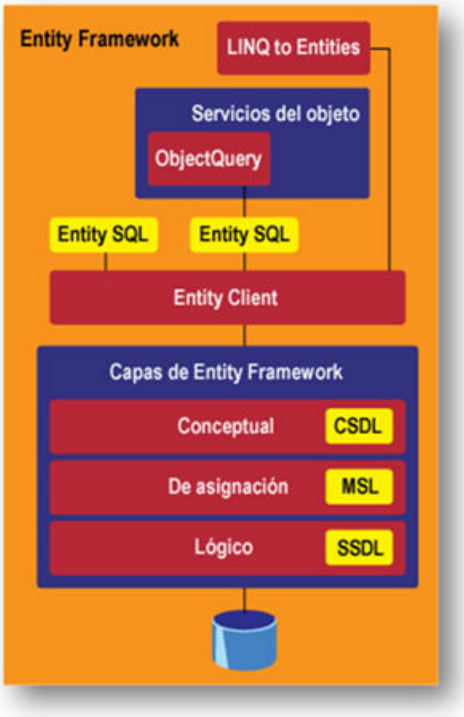
Cada instancia de la entidad creada, dispondrá de un atributo `EntityState` que podrá tomar los valores: `Added, Deleted, Detached, Modified y Unchanged.`

La actualización de la base de datos se realizará con la llamada al método `SaveChanges()` por parte de la instancia de la conexión.

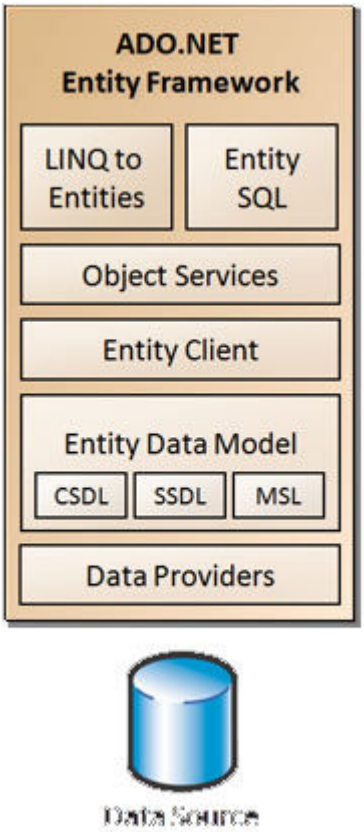
Cuando se realiza una actualización, esta se realiza por la clave de identidad de cada entidad, de aquí la importancia que esté bien definida dentro del modelo de entidades. A nivel interno, el motor `ObjectServices` garantiza que no existen varias instancias de la misma entidad, lo que sirve para evitar problemas de concurrencia dentro de la propia aplicación.

6.2.3.1 LINQ TO ENTITIES

LINQ to Entities es una implementación que permite ejecutar consultas sobre un modelo de datos entidad de ADO.NET Entity Framework, utilizando sintaxis de LINQ, por tanto hablaremos de LINQ to Entities como proveedor de consultas LINQ para ADO.NET Entity Framework.



Dentro de la arquitectura, LINQ to Entities se basa en la infraestructura de ADO.NET Entity Framework, la cual se compone de un conjunto de componentes con varias capas. Los componentes Object Services y Entity SQL están conectados directamente a LINQ to Entities. Laca capa de Object Services se encarga de la gestión de la identidad y del estado de los objetos, mientras que Entity SQL proporciona capacidades de consulta sobre el modelo de datos de la entidad. Estas capas funcionan gracias a los componentes subyacentes que, básicamente, son aquellos que permiten el mapeado entre el almacenamiento de persistencia física y su representación del modelo de datos entidad. De forma interna, Entity Framework utiliza el entorno clásico de ADO.NET para conectarse al sistema de gestión de base de datos (DBMS) y para ejecutar consultas sobre este, lo cual permite el uso de las implementaciones de `IDbConnection, IDbCommand y IDataReader.`



Se debe destacar que LINQ to Entities es compatible con todos los operadores y métodos de consulta básicos de LINQ. Por tanto, se puede elegir entre la sintaxis de consulta de LINQ o la de los métodos de extensión. Se pueden utilizar casi todos los comandos disponibles para la sintaxis de consulta de LINQ to Objects.

Existen algunas limitaciones de algunos comandos de consulta que no son soportados por LINQ to Entities, como `Agregate, Take, Skip`, etc. y esto es debido a que no todas las operaciones CLR se pueden convertir en instrucciones SQL equivalentes ya que no hay garantía de que exista la misma característica en todos los motores DBMS.

6.2.3.2 DIFERENCIAS ENTRE LINQ TO SQL Y LINQ TO ENTITIES

A primera vista LINQ to SQL y LINQ to Entities son implementaciones competitivas de LINQ. De hecho, desde el punto de vista de una consulta de LINQ ambas poseen características similares, y ambas le permiten ejecutar una consulta sobre una DBMS utilizando LINQ. Sin embargo, ambas implementaciones tienen diferencias que reflejan los diferentes escenarios a los que están orientadas.

LINQ to SQL es un mapeador relacional de objetos fácil y rápido de utilizar. Por lo general, se considera que su uso es más apropiado en contextos de desarrollo rápido de aplicaciones, donde la única base de datos es del tipo Microsoft SQL Server, y el esquema de la base de datos no tiene que ser abstraído a través de un modelo de dominios entidad.

LINQ to SQL trabaja con entidades que mapean tablas de la capa de persistencia de la base de datos de origen, siguiendo una relación 1:1. En este escenario, LINQ to SQL es muy rápido, y ofrece un rendimiento excelente al ejecutar consultas sobre Microsoft SQL Server, puesto que esta optimizado para trabajar con este producto.

LINQ to Entities es el proveedor de consultas de LINQ oara ADO.NET Entity Framework. Es un mapeador de objetos relacionales más complejo y más potente. Ha sido diseñado para ser utilizado con una gran variedad de servidores de base de datos. Puede utilizarse para definir complejos modelos de entidades sin las restricciones de un mapeado 1:1 entre entidades y un esquema de base de datos. Con ADO.NET Entity Framework se puede modelar las entidades sin tener en cuenta la estructura de datos de la capa física. También es posible mapear el modelo de datos entidad sin que sea necesaria una capa de persistencia, independientemente de que se trate de una base de datos SQL Server o no, o de que tras una entidad sencilla tengamos una tabla, un conjunto de tablas, varios procedimientos almacenados o ambos. Por supuesto estas características hacen que el modelo de datos de entidad nos permita definir consultas que son más fácilmente transferibles, lo cual puede afectar al rendimiento de estas (respecto a LINQ to SQL, que serían más eficientes). Sin embargo, se debe evaluar estas circunstancias con detalle antes de tomar una decisión respecto al modelo a utilizar.

7. CONCLUSIONES

Durante los capítulos anteriores se ha mostrado el proyecto en el que se ha trabajado, así como LINQ como una tecnología novedosa y se han enseñado sus nuevas y futuras características. Se han analizado puntos de vista diferentes respecto a las arquitecturas posibles que se pueden utilizar y se ha enseñado el camino que se sigue por parte de Microsoft y todo lo relacionado con el mapeado de base de datos relaciones.

Pues bien, lo que se quiere comentar en esta última parte de la tesina es la demostración de la experiencia y el trabajo realizado con LINQ desde el punto de vista del propio programador.

7.1 LA EXPERIENCIA PROPIA CON LINQ

LINQ to SQL y LINQ sobretodo ha supuesto una forma de revolucionar la programación convencional relacionada con el acceso a base de datos. La primera sensación que se percibe cuando programa y lo hace utilizando todas las nuevas características es una sensación de control de la situación, se está acostumbrado a realizar consultas con líneas y líneas de código con ADO.NET y sobretodo el mantenimiento que esto supone, la idea de realizar cualquier cambio suponía un reto en tiempo y recursos desperdiciados. Con LINQ el asunto mejora, se cambia la base de datos, se vuelve a mapear y automáticamente la depuración en tiempo de compilación muestra lo que se debe cambiar sin trabajar de la forma "cambiar y probar", un gran avance y todo bajo control.

Como bien se dijo en los primeros capítulos donde se repasa la arquitectura, esta está dividida en dos grandes partes, la denominada como antigua y la nueva. La antigua es donde se accede a los datos mediante ADO.NET y la nueva donde ya se utiliza LINQ to SQL. Es en ese momento cuando el programador se da cuenta del avance, ADO.NET generaba muchas más dependencias y produce muchos más problemas al producir errores en tiempo de ejecución y no en compilación.

Hay que matizar que LINQ es un gran avance pero que como todo, no siempre es la mejor opción. Existen escenarios donde podemos tener en cuenta el rendimiento del código o tiempo empleado en desarrollar ese código, en ocasiones realizar una consulta LINQ es menos eficiente que hacerlo de una forma convencional con bucles `for` o `foreach`.

En el primer caso, en el caso de que se debe tomar una decisión entre un árbol de posibilidades que nos brinda LINQ.

Se realiza una prueba en la que se dispone de una colección de objetos y quiere obtener el elemento mayor de uno de sus atributos, la tentación principal puede ser utilizar el método de extensión `Max`, pero analícese con más detenimiento:

- Utilización de un bucle `foreach` que mantiene una referencia al elemento máximo en cada momento, si el registro actual es mayor al que se tiene referencia se cambia sino se mantiene, así sucesivamente. En este caso la complejidad del problema es O(n), que matemáticamente es lo mejor que se puede conseguir sobre una lista.
- Se puede ordenar la colección y tomar el primer elemento, tiene complejidad O(n log n).
- Otra opción es usar una subconsulta, seleccionando el libro Max dentro de una comparación. La comparación convierte la operación en $O(n^2)$

- Y otra opción es utilizar dos consultas separadas, que resuelve el problema de la comparación, por tanto tiene O(n), aunque debemos iterar dos veces la colección.
- Y por último se puede hacer un operador de consulta personalizado.

Con un sencillo ejemplo encontrado en [1] nos muestra una tabla con los resultados:

Opción	Tiempo medio (en msg)
Foreach	37
OrderBy + First	1724
Subconsulta	37482
Dos consultas	66
Operador personalizado	56

Estos resultados nos muestran que el rendimiento puede variar mucho entre diferentes soluciones, es importante utilizar consultas correctas y no siempre un operador personalizado es la mejor opción, aunque si se puede siempre puede ser la mejor opción iterar una vez sobre la lista con operadores convencionales.

Por otra parte y al margen del propio rendimiento existe otro dilema que tiene que ver entre el rendimiento y la concisión del código. Si LINQ tiene una ventaja es que se puede escribir código fácilmente entendible y muy parecido al lenguaje SQL que estamos acostumbrados a utilizar, por ello existen ocasiones en las que una consulta legible es preferible a montar un fragmento de código que con el tiempo se va a convertir en tareas casi imposibles de entender, hablando de partes de un tamaño considerable.

Resumiendo, LINQ ha sido (y va ser) un importante avance para los programadores y al fin ha supuesto un importante cambio a mejor en el vacío que se encontraba la interacción del código con la base de datos. Proporcionando una interacción con las bases de datos relacionales que hasta el momento no existía en la plataforma .NET, y por tanto ganando en costes tanto de desarrollo, programación y tiempos de entrega de las aplicaciones.

7.2 Función realizada en el proyecto Agromutua.Web

Cuando entre en Prodevelop S.A. en Junio de 2008 se me introdujo en un grupo de trabajo con 2 programadores más, el equipo de trabajo de Agromutua.Web dentro del departamento de .NET de la empresa. Desde esa fecha hasta ahora se han ido desarrollando partes de la aplicación y manteniendo otras.

Nada más incorporarme al puesto de trabajo observe el importante cambio que suponía LINQ en el recién sacado Framework 3.5 de Microsoft. Mis anteriores experiencias con Frameworks de .NET por trabajos de asignaturas en la Universidad Politécnica de Valencia me permitieron observar la evolución que en poco tiempo había tenido la tecnología y el entorno de trabajo Visual Studio 2008, del que normalmente saca versiones Microsoft cada un par de años.

Como se comenta en el capítulo de la arquitectura, Agromutua.Web está dividida en una parte de acceso a datos con ADO.NET y otra ya con LINQ to SQL. En mi caso, cuando me incorpore a la empresa se estaba comenzando a migrar el acceso a datos a LINQ to SQL por lo que participe en el proceso de

mapeo y construcción de la nueva capa de datos que nos proporcionaría el acceso LINQ to SQL con la base de datos relacional SQL Server.

Durante todo este tiempo se me han asignado tareas de desarrollo de diversas partes de la aplicación.

Una de las partes que más trabajo supuso a nivel de negocio, por su importancia fue desarrollar los complejos cálculos de las pólizas de los aseguradores y a la vez el cálculo de las distintas subvenciones de comunidades autónomas que proporciona Agroseguro. Este trabajo supuso exprimir al máximo mis conocimientos de LINQ ya que necesita de la interacción con la base de datos muy frecuentemente debido que para llegar al resultado final del cálculo se deben extraer datos de múltiples tablas de la base de datos: Asegurados, Opciones Asegurables, Precios, Tarifas, Cultivos, Variedades de Cultivo, Modulación, etc…

Al igual sucedía con los cálculos para la obtención de las subvenciones de las comunidades autónomas, las cuales depende de la situación geográfica (municipio, comarca y provincia) y la variedad junto con otros campos como pueden ser la opción y la zona para realizar este cálculo que comprendía a la vez otros que cualquier fallo de uno suponía un resultado erróneo con lo que esto conlleva en una póliza.

Para todo lo relacionado con el negocio y capas intermedias LINQ permitió acelerar el trabajo notablemente, y puede ser confirmado por mi experiencia debido a que en ocasiones algunos métodos del negocio estaban implementados con la parte más antigua de la aplicación y encontrar errores y modificarlos se convertían en tareas tediosas y desesperantes para el programador, debido principalmente a que un cambio producía inconsistencia en otras partes de la aplicación solamente detectables en tiempo de ejecución con lo que esto supone a la vista del cliente.

Los cálculos suponían (como en todas las pólizas) la parte más importante, que después se deberían validar con la entidad Agroseguro, encargada de validar todas las pólizas agrarias y documentos adjuntos a nivel nacional para que no se produzcan irregularidades. Para ello se debían desarrollar envíos de archivos mediante FTP, en los cuales participe en la creación de varios documentos junto con el de la póliza.

Por otra parte mi experiencia en otros trabajos en desarrollo HTML, CSS y JavaScript me permitió desarrollar algunas de las pantallas de documentos adjuntos, referente a la capa interfaz. En estas pantallas el nivel de dificultad fue creciendo a la vez que fue creciendo la aplicación y las exigencias del cliente. La aplicación se convertía cada vez más pesada y eso nos forzó hacer uso amplio de la tecnología AJAX, por ello se desarrollaron bastantes librerías JavaScript junto con C# y VB.NET que implementaban llamadas AJAX a dichas librerías, suponiendo un incremento de la velocidad y del rendimiento de la aplicación.

Por último compaginando el desarrollo de las partes antes mencionadas una de las tareas que con más frecuencia se daba era la resolución de incidencias, apoyándonos en el gestor de proyectos e incidencias JIRA (http://www.atlassian.com/software/jira/) se iban resolviendo muchas incidencias propias del negocio con el apoyo de los informáticos de Agromutua.

Ejemplo de esto, era la pantalla de parcelas que fue y es una de las que más trabajo nos produce por el complejo negocio que existe entre los diferentes cultivos, variedades, opciones, etc… ya que en muchos casos se tenía que trabajar con esta pantalla realizada con las partes tecnológicas más viejas de la aplicación e introducir nuevas características con la llegada de la contratación de un nuevo cultivo por parte de los aseguradores.

8. BIBLIOGRAFÍA

[1] LINQ in Action. Fabrice Marguerie, Steve Eichert y Jim Wooley. 2008 Manning Publications

[2] Programación LINQ. Paolo Pialorsi, Marco Russo. Anaya Multimedia 2009

[3] La biblia de SQL Server 2005. Mike Gunderloy, Joseph L. Jorden y David W. Tschanz. Editorial Anaya Multimedia 2004.

[4] Paso a Paso ASP.NET 3.5. George Shepherd. Editorial Anaya Multimedia 2009

[5] JavaScript para desarrolladores Web. Nicholas C. Zakas. Editorial Anaya Multimedia 2005

[6] Ajax con ASP.NET. Wallace B.McClure, Scott Cate, Paul Glavich, Craig Shoemaker. Editorial Anaya 2006

[7] Manual Avanzado de Visual Basic. Jorge Serrano Perez. Editorial Anaya multimedia 2008

[8] Manual imprescindible de AJAX. Javier Mellado Dominguez. Editorial Anaya 2008

[9] ASP.NET 2.0. Bill Evjen, Scott Hanselman, Farham Muhammad, Srinivasa Sivakumar, Devin Rader. Editorial Anaya Multimedia 2006

[10] MSDN de Microsoft, http://msdn.microsoft.com/

[11] http://msdn.microsoft.com/es-es/magazine/cc337893.aspx , Operadores de Consulta con LINQ, John Papa (mmdatat@microsoft.com)

[12] http://weblogs.asp.net/scottgu/archive/2007/07/31/linq-to-sql-debug-visualizer.aspx, LINQ to SQL Debug Visualizer

[13] http://www.microsoft.com/spanish/msdn/latam/visualstudio2008/

[14] http://cmenez.wordpress.com/

[15] Visual Studio 2010, http://www.microsoft.com/spain/visualstudio/products/2010/default.mspx

[16] Parallel Extensions, http://www.microsoft.com/downloads/details.aspx?FamilyId=348F73FD-593D-4B3C-B055-694C50D2B0F3&displaylang=en

[17] Blog del CIIN, http://geeks.ms/blogs/ciin/