Proyecto de Grado 2008

Estudio, prototipación y análisis comparativo de LINQ

INFORME FINAL

Facultad de Ingeniería
Universidad de la República Oriental del Uruguay

Antonio Malaquina

Tutores: Dr. Hermann Steffen, Ing. Camila Pedocchi

Resumen

El desarrollo de software requiere la utilización de modelos, metodologías y herramientas eficientes. A lo largo de las ya varias décadas de historia múltiples han sido las propuestas de mejoras, muchas de ellas significativas. Una de las preocupaciones permanentes ha sido la de mejorar el manejo de base de datos, procurando obtener un creciente nivel de abstracción y de independencia, así como de mejorar la eficiencia de desarrollo y el confort del programador. Con relación a las épocas primitivas (en el sentido de iníciales), donde los datos en disco eran manejados explícitamente a través de direcciones físicas (disco, pista, sector), la aparición de archivos, archivos indexados y luego de las bases de datos Jerárquicas y Redes representó un enorme salto adelante. Más adelante el Modelo Relacional y el lenguaje SQL se presentan como la última y definitiva solución a la independencia y al nivel de abstracción en el manejo de datos. El enorme avance que esta tecnología representó y la universalidad de su utilización no deben ocultar muchos de sus inconvenientes: la manipulación de SQL desde los lenguajes de programación imperativos, incluso los basados en el principio de la Orientación a Objetos, sufre del inconveniente del uso de dos paradigmas diferentes (uno para la base de datos, otro para la programación), y del carácter operativo de SQL. La resolución de estas dificultades ha generado varias propuestas diferentes, como es el caso del uso de Base de Datos Orientadas a Objetos (para disponer de un único paradigma para datos y programas), la gestión de la Persistencia de Datos (ocultando la presencia explícita del manejo y trasiego de datos el ambiente de programación y el del manejador de base de datos), y más recientemente la idea de aumentar el nivel de programación ocultando el uso explicito de SQL y aplicando técnicas de programación funcional. Este último enfoque ha dado lugar históricamente a varias herramientas, más o menos dependientes de los manejadores de bases de datos o de un entorno integrado y auto contenido de programación (Lenguajes de 4ta Generación, Generadores de Aplicaciones), populares más bien en la década de los 80. Dentro de esta misma orientación y mucho más recientemente, Microsoft ha presentado una interesante propuesta denominada LINQ, bajo la forma de una extensión al ambiente de programación .NET (multi-lenguaje). Este enfoque puede verse como la síntesis de intentos de hacer más transparente la manipulación de la base de datos, aumentando el nivel de abstracción no solo a nivel de la estructura de los datos sino también de la operativa de manipulación, integrando tecnología de programación funcional dentro de lenguajes populares y de amplia difusión, sin imponer al usuario un cambio de ambiente de desarrollo o el uso de un lenguaje propietario específico.

LINQ es una herramienta incluida en el Framework de .Net (3.0), la cual permite resolver diferentes problemas como:

- Poder verificar la correctitud de las sentencias utilizadas para realizar consultas a tablas en base de datos Relacional, a documentos XML, a archivos de texto u otros orígenes de datos, en tiempo de compilación.
- Permitir al desarrollador manipular siempre el mismo conjunto de sentencias para cualquier origen de datos, o sea no hay necesidad de saber cómo se escribe tal o cual sentencia aplicada a determinado tipo de datos.
- Introduce a la Programación Funcional dentro del paradigma de Orientación Objetos y Programación Imperativa. Esta unión permite que se puedan utilizar funciones de programación funcional como parte de una consulta de LINQ.

El propósito del presente Trabajo de Grado es realizar un estudio del alcance y características de LINQ, incluyendo en la tarea no solo la investigación literaria sino también la experimentación y prototipación con fines de evaluación, tanto para aspectos de alcance funcional, facilidad de uso y performances de esta herramienta.

Para evaluar la pertinencia de la herramienta LINQ, se lo compara con herramientas que realizan tareas similares. La comparación anterior se encuentra basada en:

- Un estudio comparativo de performance, para lo cual se realizaron dos aplicaciones con las mismas características, pero donde una desarrolla sus funcionalidades con LINQ y la otra utilizando herramientas tradicionales. En esta comparación se ha incluido la observación de los tiempos de respuesta obtenidos, así como en el consumo de memoria RAM y del acceso a disco de ambas aplicaciones. Un primer eje de análisis comparativo está orientado a un aspecto de difícil definición: la facilidad de uso del ambiente de trabajo. Para ello se ha medido cuanto trabajo llevó realizar las respectivas aplicaciones, cantidad de líneas de código, y en cuan sustentable es (sustentable en el sentido de que el código escrito es "claro" y fácil de mantener).

Dentro del estudio comparativo se muestra además de qué forma fueron inculcadas las funcionalidades de LINQ en los proyectos realizados por la empresa Active Software.

Luego se desarrollo una aplicación, la cual fue construida sobre una arquitectura en tres capas, bajo el Framework 3.0 de .Net, escrita en lenguaje C#, y basada en una aplicación similar desarrollada en la empresa Active Software (dicha aplicación se encarga de gestionar el tiempo dedicado por una persona a realizar un determinado trabajo dentro de la empresa).

La realización de la aplicación posee como objetivo primordial poner a prueba a LINQ to ENTITY, una de las implementaciones de LINQ, encargada de manejar base de datos Relacional provenientes de diferentes orígenes. Esta permite manipular a las tablas como si fueran objetos, permitiendo que el acceso y manipulación de la capa de datos, pensando en un sistema en tres capas, sea sencillo y eficiente.

El segundo objetivo es el de realizar un análisis comparativo entre LINQ to ENTITY y el ambiente de desarrollo (Framework) construido por Active Software para el desarrollo de cierto tipo de aplicaciones. El análisis se focaliza en mostrar las ventajas y desventajas que poseen cada uno de esos ambientes para la realización de sus funciones, se dan a conocer y/o a tener en cuenta que características debería de tener un modelador de base de datos a objetos.

Palabras Claves:

LINQ, LINQ to ENTITY, AF, Framework .Net, Arquitectura en "tres capas", Active Software, LINQ to OBJECT, LINQ to SQL, LINQ to XML, MVC.

1. Índice

2. Introducción

2.1. Problema a Resolver

El informe trata de responder a la pregunta ¿es útil adoptar a LINQ en el desarrollo de software?

Antes de responder a la pregunta anterior, primeramente hay que resolver las siguientes interrogantes: ¿Qué tiene de interesante LINQ?, ¿Qué nos ofrece que resulta tan impactante? ¿Qué nos motivaría a utilizarlo?

Para responder a las inquietudes antes planteadas se puede primeramente empezar por mostrar cual es la situación en la que se encuentra el desarrollo de software actualmente, más específicamente con la creación de aplicaciones de escritorio (con Web sucede lo mismo) con persistencia en una base de datos Relacional, lo cual siempre traen consigo distintos problemas e inconveniencias. Como por ejemplo el escribir consultas como un *String*, donde no hay chequeo alguno del compilador si las sentencias escritas son validas o no, siendo ésta una fuente de errores de ejecución del programa. Además de que el obtener, manipular y guardar datos en las base trae sus inconvenientes, porque se trabajan con tipos de datos distintos y en mucho casos pueden llegar a ser incompatibles en varios aspectos.

Como solución surgió una herramienta llamada LINQ to SQL (encargada del manejo de base de datos con manejador SQL Server) y LINQ to ENTITY (encargada de la manipulación de diferentes manejadores de base de datos) las cuales muestran una solución a problemas planteados, permitiendo poder manejar los datos de las tablas y acceder a ellas de tal manera que el compilador se encargue de chequear su validez.

Pero casi inmediatamente después de haberse comenzado a utilizar, la comunidad (con comunidad se hace referencia a los desarrolladores de software que utilizan el Framework de .Net) duda de cuan beneficioso pueda ser para un producto de software el usar LINQ.

De la duda planteada surge la primera motivación para la realización de este proyecto. El poder mostrar si en verdad estas tecnologías LINQ to SQL y LINQ to ENTITY son pertinentes de utilizar en el desarrollo de una aplicación real basada en base de datos Relacional.

Al buscar más a fondo se encontró que estas (las tecnologías antes nombradas) se originaron de un proyecto en común llamado LINQ, el cual introduce el paradigma de la programación funcional, siendo ésta la segunda motivación, el poder escribir funciones basadas en programación funcional para el desarrollo de software.

LINQ posee algunas declinaciones, a las anteriormente nombradas LINQ to SQL, LINQ to ENTITY, también se suman, LINQ to OBJECT, LINQ to XML, LINQ to DATASET y muchas más. Entonces en base a esas diversas implementaciones se decidió ver cuán útiles podrían ser.

Para eso se propuso resolver dos problemas:

- Realizar un estudio sobre LINQ. En el mismo se indicaran las características que se pueden obtener del mismo, para luego realizar un análisis de performance donde se realizarán pruebas sobre estas implementaciones, en comparación con las tecnologías que se utilizan tradicionalmente.

  Además se van a mostrar códigos de proyectos realizados por la empresa Active Software donde se presentan las funcionalidades impuestas por LINQ en el código.

- El segundo resultado esperado es un producto basado en LINQ to ENTITY llamado "Active.Management.HoursLinq", el mismo se va a encargar de gestionar el tiempo dedicado por los empleados a la realización de una tarea dentro de una empresa (la aplicación se encuentra basada en una muy similar de la empresa Active Software).

  La realización de esta aplicación surge con el objetivo de poder aplicar en un caso real a LINQ to ENTITY, implementación de LINQ encargada de mapear las tablas de las base de datos a objetos (ORM), realizando el diseño de toda una capa de persistencia, permitiendo al desarrollador manipular información sin importar de que origen provino la misma. Esta aplicación se va a comparar con una ya realizada en la empresa Active Software, que tiene la misma funcionalidad que esta pero con la diferencia que se encuentra diseñada con el Framework de dicha empresa. La comparación consta de comparar las características que ofrece una u otra herramienta para el desarrollo de esta aplicación.

2.2. Contexto

Un punto a tener en cuenta son las decisiones que se tomaron a lo largo de este trabajo de Grado, tanto en el estudio de LINQ como en la realización del aplicativo, estas estuvieron influidas por y para la conveniencia de la empresa Active Software. Por ende muchas decisiones que se tomaron en las pruebas de performance, en la implementación y diseño se encuentran respaldadas por requerimientos propios de la empresa.

2.3. Objetivo

El objetivo de este Trabajo de Grado es el de evaluar hasta dónde LINQ es una herramienta muy importante para el desarrollo de software. Para ello se harán medidas de performance en comparación a las tecnologías tradicionales (para eso se la desea introducir en la realización de proyectos para la empresa Active Software). Además se desea analizar más en profundidad el uso de la herramienta LINQ to ENTITY, comparándola con el Framework desarrollado por la empresa Active Software.
De ambas experiencias de evaluación concreta se aspira a obtener conclusiones lo más generales posibles, que puedan resultar de utilidad para toda la comunidad de desarrolladores de software.

2.4. Resultados Esperados

De este proyecto como ya se nombro anteriormente se esperan dos resultados: uno de ellos es un estudio de LINQ, en el cual se trata de dar una justificación objetiva a través de diferentes casos de prueba, de la utilidad o no del uso de LINQ como herramienta fundamental en el desarrollo de aplicaciones basadas en tecnología .NET más específicamente basadas en lenguaje C#. Mostrando además usos que se hicieron de las herramientas en los diferentes proyectos desarrollados por la empresa Active Software.

El otro resultado esperado es la realización de una aplicación de mediano porte donde se utiliza como mediador de acceso a la base de datos a LINQ to ENTITY. Los objetivos de la misma son: observar cuán bien funciona este en el manejo de base de datos Relacional, y también poder comparar al mismo con la implementación de la capa de datos del Framework de Active Software, el cual se basa en el patrón de diseño MVC (Model View Controller).

2.5. Organización del Documento

Para poder comprender a grandes líneas sobre que conceptos se basa el proyecto es recomendable dirigirse primeramente a capítulo Conceptos Relevantes, el cual da una introducción a los diferentes temas que se tratan a lo largo del informe permitiendo entender con claridad los resultados esperados.

Luego el capítulo Introducción explica los cometidos y motivaciones que dieron cabida a la realización del proyecto. Además explica cuales son los resultados esperados así como la organización del siguiente documento en sí.

Luego el capítulo Desarrollo profundiza en los detalles de la realización de los resultados esperados. Para el caso del análisis y puesta en uso de LINQ por parte de Active Software, se explica: contenido del mismo, detalles sobre los diferentes análisis realizados así como también las conclusiones que se obtuvieron.

En la sección Aplicación Desarrollada se describe específicamente la aplicación de referencia, exponiendo los resultados que se obtuvieron de la comparación con el Framework de Active Software, mostrando además algunos diagramas de análisis y de diseño, etc.

En el capítulo Conclusiones y Trabajo a Futuro, el último de este informe, muestra las conclusiones finales que se obtuvieron, tanto de los resultados esperados como del proyecto en sí, así como también de algunas propuestas de trabajos a futuro, y algunas autocríticas finales.

3. Conceptos Relevantes

3.1. Introducción

El paradigma de la Programación Orientada a Objetos es utilizado cada vez más frecuencia en el mundo de los desarrolladores. En ese contexto, el manejar datos y estructuras que no responden al modelo objetos, representa una actividad inconveniente. Resulta un desafío la creación de una aplicación que se comunique con una base de datos Relacional, porque la misma a parte de manejar tipo de datos distintos a los que maneja el lenguaje con el cual fue construida la aplicación, su actualización requiere adaptaciones y cambios para la aplicación. LINQ es una de las recientes propuestas para solucionar estos problemas.

LINQ cuyas siglas significan Lenguaje Integrado de Consulta, fue desarrollado como un proyecto de investigación de Microsoft llamado **Cω**, como parte de una extensión del lenguaje C#, cuyos objetivos eran el de experimentar con consultas integradas entre este lenguaje y SQL, XML y demás. Entre los muchos involucrados en este proyecto se encuentran Eric Meijer, Wolfram Schulte y Gavin Bierman, pero el que logro integrar Cω a C# y crear LINQ fue Andres Hejlsberg.

Pero la idea de LINQ no fue solo manejada por C#, VB, y demás lenguajes basados en el Framework de .Net. Cabe resaltar que el mismo fue implementado para poder ser utilizado sobre Java, en este caso se llama Queare, y fue creado por Andres Noras. La idea del mismo es similar a la idea original de LINQ, en el hecho de manipular un conjunto de sentencias similares, y que las consultas hechas con la misma, logren ser verificadas (la correctitud de las mismas) en tiempo de compilación.

LINQ, además de saber manipular un conjunto de sentencias similares a SQL (SQL), introduce el paradigma de la Programación Funcional para la realización de consultas. Estas consultas operan sobre conjuntos de objetos que se encuentran cargados en memoria. A continuación se ve un ejemplo:

```
IEnumerable<Order> orders = customers.Where(c => c.Country == "Denmark").SelectMany(c => c.Orders);
```

La sentencias **Where** define una función lambda que indica que posee una entrada de tipo *Customer* (el tipo lo deriva de la colección a la que esta asignada) y la salida *bool* (se deriva de la expresión), y cuya finalidad es la de filtrar a los elementos de la colección que cumplan una determinada condición. Se puede apreciar el parecido que tiene esta definición con la de **Filter** función definida por el lenguaje **Hazkell**, como se muestra en la siguiente figura.

Filter

Selecciona los elementos de una lista que cumplen con una propiedad.

```
filter :: (a -> Bool) -> [a] -> [a]

filter p [] = []                                -- (filter.1)
filter p (x:xs)
  | p x         = x : filter p xs               -- (filter.2)
  | otherwise   =     filter p xs               -- (filter.3)
```

Definido con listas por comprensión

```
filter' p xs = [ x | x <- xs , p x ]
```

Generalmente para el desarrollo de una aplicación no se utiliza LINQ directamente, sino a sus implementaciones o declinaciones. Las mismas están basadas en las ideas impuestas por el mismo, pero agregan otras funcionalidades. A continuación se presenta la lista de las más importantes mostrando alguna de las funcionalidades que ofrecen:

- Con la utilización de LINQ to OBJECT, un desarrollador puede manipular a un conjunto de objetos de una manera muy simple, permitiendo por ejemplo ordenar una colecciones de objetos en pocas líneas de código. Una característica que deben de tener esas colecciones es que para poder manipularlas con LINQ, deben de implementar IEnumerable o IQuerable

- ```
  var B = from dep in departamentos
                        orderby dep.idPais, dep.idDepartamento ascending
                        select dep;
  ```

- Con LINQ to XML, se puede manipular el contenido de un documento XML, introduciendo un conjunto de clases nuevas que permiten integrar varios modelos de XML, como XPath, XQuery.

- LINQ to Data Set ofrece la posibilidad de extender las funcionalidades que se tienen sobre el manejo de los DataSet (Framework .NET), haciendo que realizar consultas con los mismos sea claramente más sencillo

- LINQ to SQL , LINQ to ENTITY, estas dos herramientas son parte de la solución al problema central que desea solucionar LINQ, estos problemas son:
  - Las consultas son escritas como un *String*, lo cual resulta que los errores de sintaxis de la misma no surjan en tiempo de compilación sino en tiempo de ejecución.
  - Además si se realiza una consulta para SQL, no se puede reutilizar para realizar la misma en otro manejador de base de datos.
  - Problemas con los tipos que maneja una base de datos y los utilizados por el lenguaje de Programación.

  Estas dos herramientas son ORM (Mapeador de base de datos a objeto), LINQ to SQL está orientada a SQLServer, en cambio LINQ to ENTITY está orientada a un gran conjunto de manejadores.

  Las ventajas que ofrece LINQ to ENTITY son llamativas para un ORM, pero no quitan que venga a la mente de los diferentes desarrolladores otros enfoques emparentados, como es el caso de NHibernate (es una implementación para .Net de la tecnología Hibernate hecha por Java, cuya finalidad es similar a la de LINQ to ENTITY). Ambos poseen funcionalidades similares, pero con diferencias importantes. Más adelante en el informe se abordará también este tema.

A continuación se van a desarrollar sobre otras características que ofrece LINQ así como otros motivos que llevaron a su creación.

Y más adelante se indicará con más detalle cada una de las implementaciones de LINQ anteriormente nombradas.
  ```

Luego se realizara una exploración por las distintas mejoras que trajo con si el Framework 3.0 de .Net, estas mejoras son la base sobre la cual trabaja LINQ. Las mismas son por ejemplo, el poder introducir el paradigma de la programación funcional al de la programación a objeto, así como también el poder desarrollar aplicaciones sobre datos Relacional accediendo a los mismos de la forma más directa posible.

3.2. LINQ

3.2.1. Relational Model vs Herarchical

El problema existente con el manejo de objetos, y la manipulación de datos provenientes de una base de datos o de un documento XML es que, entre si no son compatibles, por que originalmente no fueron construidos para trabajar juntos.

La información es manipulada por la aplicación, escrita usando OOP (Programación Orientada a Objetos), utilizando lenguajes como C#, Visual Basic, etc. Pero para trasladar un objeto dentro de otra representación tales como tuplas de una base de datos Relacional, ahí se origina un problema para el desarrollador.

Con LINQ se resuelve este problema "**Data!= Object**".

A continuación se muestra una lista de problemas que se tienen al realizara el mapeo de Objetos a Relacional.

- Las base de datos Relacional y los lenguajes orientados a objetos no comparten el mismo conjuntos de tipos de datos primitivos, el caso más notorio es el manejo de *string*, que en el caso de la base de datos tiene un limite determinado y el *string* del lenguaje puede no tenerlo (solo los limites que le asigna el compilador).
- OPP y Relacional son teorías que manejan distintos modelos de datos, las base de datos Relacional manejan tablas, Primary Key, en cambio en OPP, se manejan objetos con identidad.
- En base de datos Relacional hay una clara separación entre que es el código y cuales son los datos, en cambio en Objetos esa separación no existe.
- Los conceptos de herencia y composición que maneja OPP, no son soportados por una base de datos Relacional, lo cual muestra que los datos se representan de manera distinta según el modelo utilizado.

3.2.2. Tabla Cometidos – Motivación.

A continuación se muestra una tabla donde se indica cuales fueron los cometidos y motivaciones que llevaron ala realización de LINQ.

Cometidos	**Motivación**
Integrar objetos con datos Relacional	-Unificar las sintaxis de las consultas, permitiendo acceder a objetos de distintos orígenes a través de la misma sintaxis. - Un único modelo para procesar todos los datos sin importar cual sea su origen o la representación en memoria.
Tener más poder en el acceso a sentencias SQL y XQuery	- Las consultas se encuentren integradas dentro del lenguaje de programación.
Tipos seguros (Type Safety)	-En tiempo de compilación se pueden encontrar errores que solo se descubren en tiempo de ejecución (Run time). -El compilador encontraría errores en las consultas.
Poder “debuguear” consultas	-Permite a los desarrolladores debuguear las consultas de LINQ, pasó a paso y con una rica información del debug.
Extenso soporte de IntelliSense	Asiste al desarrollador cuando escribe consultas para aumentar la productividad y aumentar la velocidad para acostumbrarse a la nueva sintaxis El editor podría guiar al programador a escribir consultas.
Permanece 100% compatible con herramientas anteriores	Ser capaz de usar colecciones genéricas y estándares, DataBinding, y muchas otras herramientas

3.2.3. Algunas Consultas

A continuación se muestran algunos ejemplos donde se emplea LINQ para la resolución de diferentes problemas.

Primero se va a plantear una realidad sobre la cual se basan los ejemplos. La misma dice así: “Los países se encuentran divididos en Departamentos donde todos estos tienen una capital que los gobierna”.

Ejemplo 1

“Obtener `cant_elementos` *departamentos desde* `inicio_filtro`*”*

```
var D = departamentos.Skip(inicio_filtro).Take(cant_elementos);
```

Skip esta operación permite devolver una colección de departamentos cuyos índices dentro de la colección departamentos, son mayores a `inicio_filtro`.

Take permite obtener `cant_elementos` de una determinada colección.

Ejemplo 2

"Ordenar todos los departamentos por el identificador del país al cual pertenece y por el identificador del mismo"

```
var B = from dep in departamentos
        orderby dep.idPais, dep.idDepartamento ascending
        select dep;
```

OrderBy esta operación permite ordenar una colección de objetos por los campos indicados a continuación, y luego se puede definir que la ordenación sea ascendente o descendente.

Ejemplo 3

"Preguntar si existe algún departamento donde la cantidad de habitantes sea menor a 1000"

```
bool result = departamentos.Any(p => p. var cant_Hab < 1000)
```

Any permite verificar si algún elemento dentro de una colección cumple una determinada condición, en este caso la condición era `cant_Hab < 1000`.

Exist es igual a la operación anterior, verifica que existe por lo menos un elemento dentro de una colección que cumple con una determinada condición.

All es una operación similar a la anterior, pero en este caso todos los elementos dentro de la colección tienen que cumplir con una determinada condición.

Ejemplo 4

"Obtener el total de habitantes de todos los departamentos de Uruguay"

```
var CantHab = departamentosUruguay.Sum(p => p.cant_Hab);
```

Sum permite sumar un determinado campo de una de una determinada colección de objetos.

Ejemplo 5

"Obtener cual es el departamento que tiene menor cantidad de habitantes del Uruguay"

```
departamentosUruguay.Min(p => p.cant_Hab);
```

Min permite obtener de todos los elementos de una colección el mínimo referido a un campo específico, por ejemplo en este caso `cant_Hab`.

Max permite obtener de todos los elementos de una colección el máximo referido a un campo específico, por ejemplo en este caso `cant_Hab`.

Ejemplo 6

"Obtener la cantidad de departamentos cuyo total de habitantes no supera los 10000"

```
var CantDep = departamentosUruguay.Count(p => p.cant_Hab < 10000);
```

Count esta operación cuenta la cantidad de objetos de una colección que cumplen con una determinada condición, en este caso que `cant_Hab < 10000`.

Los ejemplos anteriores muestran algunas de las funcionalidades ofrecidas, mostrando los diferentes comandos de consultas utilizados. Para conocer otras funcionalidades, otros comandos ver http://aspnetresources.com/blog/linq_sqo__cheat_sheet.aspx

3.2.4. LINQ to OBJECT

Es una parte de LINQ, encargada de realizar consultas sobre colecciones de objetos que se encuentran residiendo en memoria.
A continuación se nombraran cuales son las colecciones que soportan las consultas realizadas a través de LINQ to OBJECT:

Arrays

Generic Lists

- 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>

Generic Dictionaries

- System.Collections.Generic.Dictionary<TKey,TValue>
- System.Collections.Generic.SortedDictionary<TKey, TValue>
- System.Collections.Generic.SortedList<TKey, TValue>

String

Además de las anteriores colecciones, se puede también usar LINQ to OBJECT, en cualquier colección cuyo tipo implemente *IEnumerable<T>* o *IQueryable<T>*.

Para entender un poco más que es LINQ to OBJECT a continuación se muestran unos ejemplos:

```
List<Comprador> compradores;// Lista de compradores
…
IEnumerable<Comprador> personas = compradores.Where(c => c.Country == "Uruguay")
```

En el ejemplo anterior se realizo una búsqueda dentro de una la colección de compradores para buscar todos aquellos cuyo país es Uruguay.

```
bool b = productos.Any(p=>p.PrecioPorUnidad >= 100 && p.UnidadesEnStock == 0)
```

En el ejemplo anterior lo que se hace es consultar a una colección de productos si alguno de esos productos no tiene unidades en stock y el precio por unidad es mayor a 100.

3.2.5. LINQ to XML

3.2.5.1. Introducción

XML (Extensible Markup Language) es un metalenguaje extensible, su utilización juega un papel fundamental en el manejo de documentos, ya que permite el intercambio de una gran cantidad de datos entre distintos sistemas de forma fiable, fácil y segura. En la actualidad la mayoría de los Framework de desarrollo manejan XML y sus diferentes especificaciones, algunas de ellas se describen a continuación:

XML Schema Definition (XSD): Define la estructura de los documentos de XML, a través de la cual se puede verificar consistencia de distintos archivos XML generados.

Extensible Stylesheet Language for Transformations (XSLT): Transforma documentos XML desde un esquema a otro.

Extensible Stylesheet Language (XSL): El objetivo principal que cumple es el de estructurar, diseñar y presentar cualquier contenido tanto en un navegador Web, como en un dispositivo móvil

XPath y XQuery: Permite acceder a partes de un documento XML viendo al mismo como un árbol de valores y atributos.

Document Object Model (DOM): Maneja representación en memoria de documentos XML.

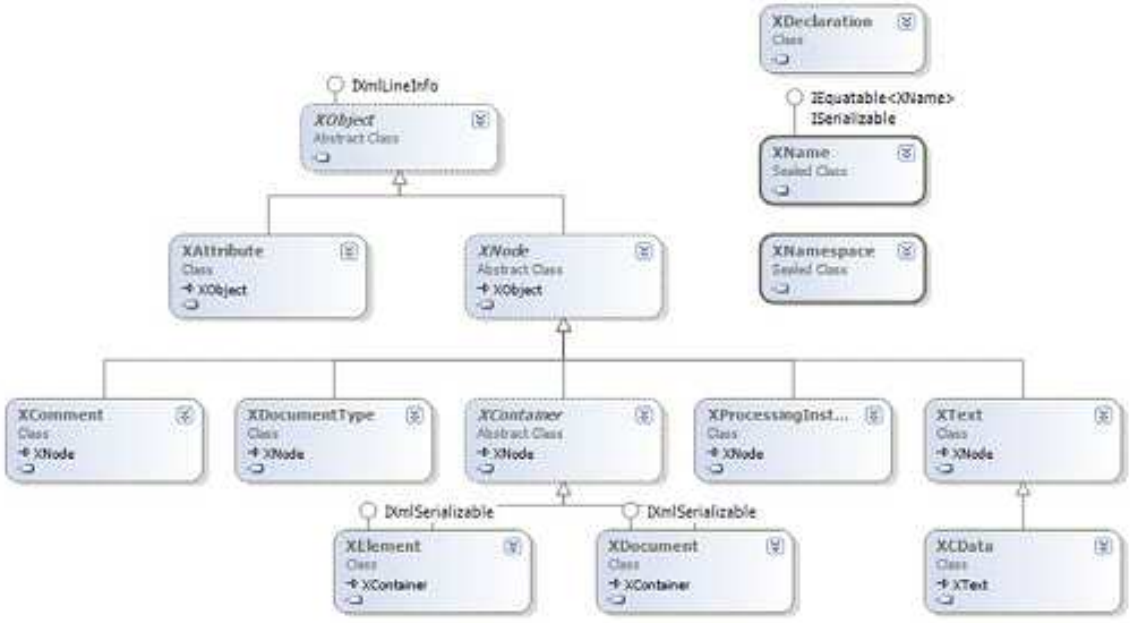
Simple Object Acces Protocol (SOAP) services: Realiza la interoperabilidad entre plataformas usando mensajes XML.

3.2.5.2. Características

Se puede observar que en .NET hay muchas APIs que se encargan del manejo de XML, el manejo de los XML puede realizarse a través de distintos modelos como pueden ser; XSLT, *XPath*, *XQuery*, XML DOM, etc., usando la tecnología existente para manejar cada uno de esta se necesitan herramientas distintas, se puede observar entonces que el problema es que no son genéricas estas APIs y de alguna manera hay que poder trabajar con ellos de forma unificada.

En solución a estos problemas surge LINQ to XML, el cual permite realizar operaciones de consulta y transformación utilizando *XQuery* y *XPath* integrados dentro de los lenguajes .NET. Además de resolver este problema LINQ to XML, da características nuevas a usar en el manejo de los XML, como los son el manejo de nulos, construcciones funcionales, además de los avances que provee C#, como son los *Extension Method*, *Anonymous Type*, etc.

LINQ to XML además ofrece una API nueva para el manejo tradicional que se tiene utilizando DOM, la cual se describe a continuación con el siguiente diagrama y la descripción de los componentes más importantes:



A continuación se comentara algunas clases que son más relevantes:

XNode – Es la clase base para todas las clases de LINQ to XML, encargada de representar a los nodos. Posee operaciones de actualización como lo son *AddBeforeSelf, Remove*, etc.

XElement – Es de las clases fundamentales de esta API, representa a los elementos de XML (nodos) que contienen otros elementos (nodos hijos). Aparte de poseer las operaciones de actualización heredadas de sus antecesores posee *Attributes, AncestorsAndSelf, RemoveAll, SetElementValue, SetAttributeValue*, etc. Además posee las operaciones *Load* y *Save* que permiten cargar un XML desde un recurso externo y guardar un XML en otro recurso externo, respectivamente. También contiene la operación *Parse* que permite cargar un *XElement* desde un *string*.

XAttribute – Son objetos formados por un par nombre – valor que representan los atributos en LINQ to XML que están asociados a un *XElement* object.

XDocument – A simple vista parece que ofrece las mismas operaciones que *XElement*, pero tiene diferencias como:
- Posee un solo *XElement* origen
- Permite la declaración de un XML
- Y de un tipo de documento XML

XName – Representa el nombre completo para un *XElement* o un *XAttribute.*

XNameSpace – Es la clase que representa la porción del *XName* referida a al espacio de nombres.

3.2.6. LINQ to DATASET

Data Set es uno de los componentes más ampliamente utilizados en ADO.NET. Es importante para la programación sin conexión y por permitir almacenar explícitamente en caché valores de diferentes orígenes de datos. A pesar de su importancia tiene sus capacidades de consultas muy limitadas.

Los métodos *Select*, *GetChildRows* y *GetParentRow*, permiten ordenar, filtrar, pero no realizar tareas más complejas que esas, para tareas más complejas hay que realizar consultas más personalizadas, lo que lleva a que la aplicación tenga un bajo rendimiento y cuyo mantenimiento sea muy bajo. LINQ to DATASET facilita la realización de la consulta permitiendo a los desarrolladores escribir consultas usando la sintaxis dada por el lenguaje, en lugar de usar un lenguaje de consulta distinto (por ejemplo SQL).

Por ejemplo a continuación se ve una aplicación de esta herramienta

```
System.Data.DataTable productos = ds.Tables["Producto"];

IEnumerable<DataRow> query =
    from product in products.AsEnumerable()
    select product;

Console.WriteLine("Nombres de Producto:");
foreach (DataRow p in query)
{
    Console.WriteLine(p.Field<string>("Nombre"));
}
```

En el ejemplo anterior se puede observar que esta extensión de LINQ permite manejar a las tablas como lista de filas y poder realizar diversas operaciones sobre las mismas.

El uso de LINQ en DATASET es similar al de LINQ to OBJECT, LINQ to XML, etc. Pero hay que recordar que antes de realizar una consulta a un objeto *DataSet* este debe de ser “rellenado”, a esto se refiere a que debe de adquirir los valores del *DataSet* antes de ser utilizado, unas de las formas para lograr este cometido puede ser a través de *DataAdapter* o LINQ to SQL.

3.2.7. LINQ to SQL

3.2.7.1. Introducción

En LINQ to SQL, el modelo de datos de una base de datos Relacional se asigna a un modelo de objetos expresado en el lenguaje de programación del desarrollador.

Cuando la aplicación se ejecuta, LINQ to SQL convierte a SQL las consultas integradas al lenguaje en el modelo de objetos y las envía a la base de datos para su ejecución. Cuando la base de datos devuelve los resultados, LINQ to SQL los vuelve a convertir en objetos con los que puede trabajar en su propio lenguaje de programación.

Esta herramienta surge de los siguientes factores:

- Imposibilidad de chequear en tiempo de compilación consultas hechas a la base de datos Relacional, porque en el programa se realizan sobre un string a lo cual el compilador no hace ningún chequeo de sintaxis.
- La mayoría de las aplicaciones, de sistemas de información, se programan generalmente usando algún lenguaje orientado objetos, en cambio hay que usar otro lenguaje para manejar la base de datos, esto lleva a: tener diferentes interpretaciones de la información y además se deben de conocer dos, o más lenguajes de programación, etc.

3.2.7.2. Primeros Pasos

A continuación se va a explicar cuáles son los componentes que se deben manejar y conocer para poder manejar LINQ to SQL.

Lo primero que se debe hace es definir a los objetos que van a representar a los datos, estos son las **clases de entidad**, para eso se debe de usar una serie de atributos, como por ejemplo "**Table**" que a su vez posee una propiedad llamada "**Name**" a través de la cual se define el nombre de la tabla de la base de datos a la cual vamos a representar con este objeto (si no se indica un nombre se toma el nombre de la clase como el nombre de la tabla).

Otro atributo que también se define es "**Column**" al cual también se le puede dar varias características para permitir un mapeo lo más exacto posible entre los campos de la clase y las columnas de la tabla. Solamente los campos declarados como columnas serán persistidos o recuperados de la base de datos.

```
[Table (Name = "Customer")]
    public class Cliente
    {
        [Column(Name = "Cedula_Identidad", Id = true)]
        private string ci;
        [Column(Name = "Nombre")]
        private string nombre;
    }
```

Un último atributo a destacar es el atributo "**Association**", el mismo es el encargado de representar a las relaciones, que pueden existir entre distintas tablas de la base de datos.

Esta forma de resolver las relaciones permite definirlas como una propiedad de la clase, en cambio en la base de datos Relacional para representar esta relación se modelan normalmente como valores de F*oreing-Key* que hacen referencia a *Primary-Key* de otras tablas, y para realizar la navegación hay que aplicar una operación de *join* entre las tablas.

A continuación se muestra un ejemplo de cómo se utiliza este atributo, viendo el caso de un Cliente y las órdenes de compra del mismo.

```
[Table(Name="Customer")]
public class Cliente
{
    [Column(Name="Cedula_Identidad",Id=true)]
    public string ci;

    [Column(Name = "Nombre")]
    private string nombre;

    [Association(OtherKey = "Cedula_Identidad")]
    private EntitySet<Orden> _Ordenes;


    public EntitySet<Orden> Ordenes
    {
        get { return this.Ordenes; }
        set { this.Ordenes.Assign(value); }
    }
}
```

La propiedad *Órdenes* es de tipo "**EntitySet**", este tipo indica que es una relación de 1-N entre el cliente y sus órdenes de compra, y la propiedad "**OtherKey**" especifica cómo se logra la asociación indicando el nombre de la propiedad de la clase a la que se está asociado.

Otra forma de indicar esta relación entre el cliente y sus órdenes de compras es invertir la situación e indicar que cliente posee la orden de compra indicada, para esto se utilizó el siguiente código.

```
[Table (Name="Order")]
    public class Orden
    {
        [Column(IsPrimaryKey = true)]
        public int IdOrden;

        [Column (Name="Cedula_Identidad")]
        public string _IdCliente;

        [Column (Name="Fecha")]
        public DateTime _Fecha_Ini;


        private EntityRef<Cliente> _Cliente;
        [Association(Storage = "_Cliente", ThisKey = "Cedula_Identidad")]
        public Cliente  Cliente
        {
            get { return this._Cliente.Entity; }
            set { this._ Cliente.Entity = value; }
        }

    }
```

En este caso se utiliza el tipo "E*ntityRef* "para describir la relación entre el Cliente y la Orden de Compra.

En este ejemplo se ve una relación uno a varios pero también pueden suceder los siguientes casos:

- Uno a uno: Para representar este tipo de relación se debe incluir en ambas clases de la relación el tipo *EntityRef* en ambos lados.
- Varios a varios: En este caso se debe de implementar una clase auxiliar llamada tabla de vínculos (también conocida como tabla de unión), donde la clave principal que contiene, suele estar formada por un conjunto de claves externas de las otras dos tablas.

3.2.7.3. DataContext

Es la clase que actúa como medio de conducción encargada de devolver los objetos de la base de datos y también de actualizar la base de datos luego de haber realizado alguna consulta, modificación o eliminación.
Para representar una base de datos se puede definir un *DataContext* de la siguiente manera, donde se declara la tabla base como miembro del contexto (por eso se lo llama en este caso *fuertemente tipado*).

```
 public partial class ClienteContext : DataContext
{

        public Table<Cliente> Personas;
        public ClienteContext(string connection) : base(connection) { }

}
```

Lo que posee esta forma de utilización es que, cada vez que se quiera acceder a la tabla de clientes llamada Personas tengo que hacer lo siguiente.

```
ClienteContext persContext= new ClienteContext(@"string de conexion");
Table<Cliente> cliente = persContext.Personas;
```

En cambio si usara directamente la clase **DataContext** entonces lo que tendría que hacer es utilizar la operación *GetTable<Clase> ()* de la siguiente manera:

```
DataContext persContext  = new DataContext(@"string de conexion");
Table< Cliente > persona = persContext.GetTable< Cliente >();
```

Para más información visitar el siguiente link: http://msdn.microsoft.com/es-ar/library/bb425822(en-us).aspx#Mtps_DropDownFilterText

LINQ to ENTITY

3.2.7.4. Introducción

A pesar de muchos avances en la interacción entre base de datos y entornos de desarrollo, aun existe una pequeña brecha entre los dos, para resolver esto se utilizan entidades (Entity) que permiten acortar esta brecha entre las filas lógicas y los objetos.

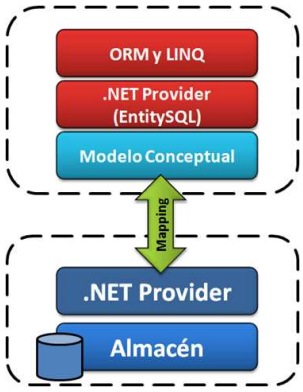
Más específicamente, la mayoría de los desarrolladores (que realizan aplicaciones basadas en una arquitectura de 3 capas) tienen que por lo menos saber dos lenguajes de programación uno para el diseño de la capa de negocio y presentación (generalmente un lenguaje orientado a objetos como C# por ejemplo) y otro lenguaje para la capa de persistencia física (algún lenguaje que maneje el accesos a una base de datos como el caso de *SQL*). El hecho de manejar dos lenguajes que necesitan estar relacionados puede producir lagunas en el código, como por ejemplo en el caso de *SQL* y C#, el valor nulo es tratado de distinta manera en uno u otro caso, además las consultas que se escriben en C# basadas en la sintaxis *SQL*, no son verificadas antes de su ejecución.

Otra razón, es el tener que programar para cada administrador de base de datos de una manera totalmente diferente. El inconveniente es que cada administrador usa su propia sintaxis, pueden ser similares entre sí pero no son iguales (por ejemplo *SQL*, *ORACLE*).

Por esto y más razones que se indicaran más adelante en este informe se utiliza LINQ con *ADO.NET Entity Framework* (Entity Framework es un nuevo componente de la familiar ADO.NET) para solucionar estas problemáticas. Esta nueva herramienta permite olvidar con que capa de persistencia se está trabajando, además nos asegura mantenibilidad e independencia de plataforma (Entity Framework). Se trabaja con un EDM (Modelo de Entidad de Datos), que nos permite ese nivel de abstracción que se desea, permitiendo además realizar consultas a una lista de entidades abstrayéndose de la capa de datos físico, usando la sintaxis de LINQ.

3.2.7.5. Funcionamiento y Características

Básicamente se traduce en una arquitectura formada por dos partes bien diferenciadas, el Modelo de Datos y el Modelo Conceptual (EDM), donde el pasaje de una capa a la otra se realiza a través de mapeo, para el cual se utilizan proveedores.NET que permiten interactuar con los datos en cada una de las capas.



La idea es que el provider .NET de la capa de datos se encargue de traducir las operaciones de datos realizadas en la capa conceptual a un lenguaje entendible por la BD (SQL). Por su parte, el proveedor .NET (EntitySQL) de la capa conceptual traduce las consultas LINQ a instrucciones que interactúan con el modelo conceptual definido mediante el proceso de mapeo.

ADO.NET Entity Framework

Como ya se dijo anteriormente es un componente que aparece en la nueva versión del Framework que pertenece a la familia de ADO.NET.

El modelo de entidad de datos (ADO.NET Entity Framework) permite tener un nivel de abstracción, tan grande que permite al desarrollador olvidarse de la capa de persistencia con la que se está trabajando, con esto se quiere decir que permite trabajar con los datos sin importar cual sea el origen de donde estos provienen.

Para cumplir con estas funcionalidades el mismo se divide en tres capas:

Conceptual, definida mediante un archivo.*CSDL* (Conceptual Schema Definition Language, con información relativa a las entidades del modelo), es la que define las entidades y las relaciones como se encuentra definida en la capa de negocio

De asignación, definida con *MSL* (Mapping Schema Language, posee la información para poder mapear la capa conceptual y la capa lógica), establece el mapeo entre las entidades del modelo conceptual y sus correspondencias con la BD.

Lógica, definida con *SSDL* (Store Schema Definition Language), representa el esquema de la BD.

Para conocer más sobre las diferentes capaz visitar los siguientes links: [44]
http://msdn.microsoft.com/en-us/library/bb399292.aspx
http://msdn.microsoft.com/en-us/magazine/cc163399.aspx

3.3. Otras Herramientas

A continuación explican las tecnologías que acompañaron a LINQ, en el Framework 3.0 de .Net.

3.3.1. Local Type Inference

Esta nueva funcionalidad que posee C# permite declarar una variable sin definirle el tipo directamente, encargándose más adelante el compilador de determinar el tipo, infiriéndolo de la expresión que tiene asignada la variable.
A continuación vemos un ejemplo

```
var a = '3'    //esto nos indica que a es de tipo char
      char b = a    // esta asignación internamente realiza un casteo a char
```

Es lo mismo que escribir

```
char a='3'
char b=a
```

Este ejemplo es importante porque muestra que la declaración de tipo, realizada con *var* es segura (type-safe), por lo tanto si luego de definir `var a = '3',` se hiciera `a = 3 ,` ocurriría un error.

3.3.2. Extensions Method

Es un mecanismo a través del cual se puede añadir métodos a clases existentes sin tener necesidad de utilizar ningún mecanismo de herencia. En realidad el aporte más importante que ofrece es el de otorgar la posibilidad de poder extender clases selladas (*sealed*), privadas y/o internas.

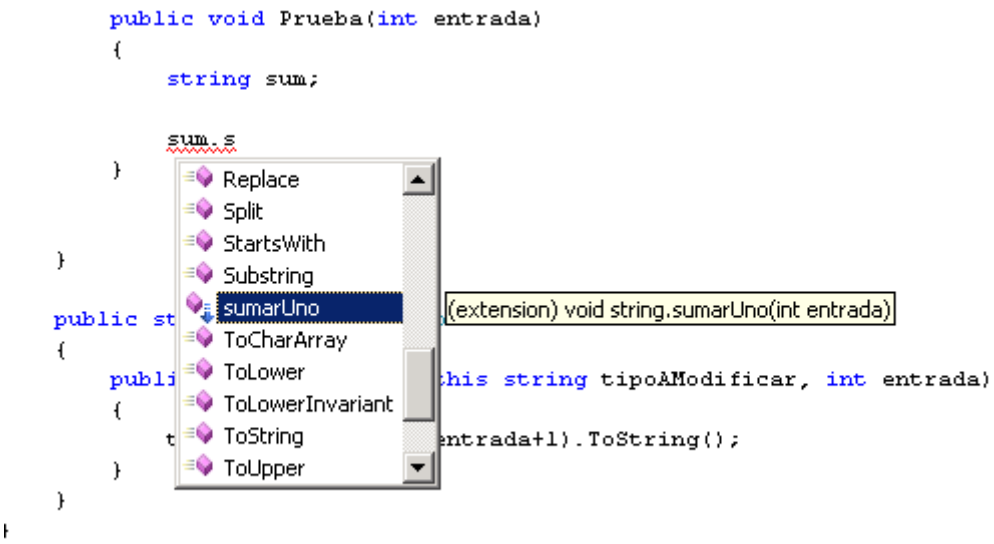
Se implementan a nivel de compilador, no ejerciendo influencia el CLR (Command Language Runtime) en ellos.

Una característica que cumplen es que deben ser estáticos y públicos, y deben ser declarados dentro de una clase estática, además el primer parámetro que tienen estos métodos es this seguido de la clase que se quiere extender.

A continuación se muestra un ejemplo de cómo se definiría una Extension Method, de la clase `string.`

```
public static class StringModificada
    {
        public static void sumarUno(this string tipoAModificar, int entrada)
        {
            tipoAModificar =  (entrada+1).ToString();
        }
    }
```

Cuando se va a utilizar una Extension Method uno puede diferenciarlos de los demás métodos propios de la clase, porque en el listado de funciones aparece una flecha azul, como se muestra en la siguiente imagen:



3.3.3. *Object Inicialization Expresión*

Esta funcionalidad ofrece la posibilidad de poder inicializar las propiedades de un objeto pertenecientes a una clase definida (con esto se quiere decir que se conoce cuáles son sus propiedades), sin ser necesario tener que definir un constructor para cada combinación de propiedades de la clase que necesito inicializar, por ejemplo si se define una clase de la siguiente manera:

```
public class Punto
{
    public int x{get{return x;}set{x=value;}}
    public int y{get{return y;}set{y=value;}}

}
```

Si se quiere inicializar una instancia de la clase anterior dándole valores a x e y, usando la funcionalidad anteriormente nombrada con solo escribir la siguiente línea:

```
Punto p = new Punto{12, 12};
```

Ya se estaría inicializando al objeto p.

3.3.4. *Anonymous Type*

El tipo anónimo es generado por el compilador y permite declarar objetos de clases anónimas (se refiere a que no se sabe de qué tipo son), sin necesidad de declarar la clase.

A continuación con el siguiente ejemplo se podrá observar cuan útil es combinándolo con LINQ (que ya se explicará con más detalles de que se trata).

```
public class Persona
{
   public String Nombre { get { return Nombre; } set { Nombre = value; } }
   public String Apellido { get { return Apellido; } set { Apellido = value; } }
   public String Telefono { get { return Telefono; } set { Telefono = value; } }
   public String Ci { get { return Ci; } set { Ci = value; } }
}
```

```
IEnumerable<Presentacion> listaPersonas = from p in Persona
                                          where p.Nombre == "Martin"
                                            select new Presentacion { Nombre = p.Nombre, Apellido = p.Apellido, Telefono =
        p.Telefono, Ci = p.Ci};
```

La consulta anterior indica que se debe declarar, una clase llamada *Presentacion* la cual va a estar formada por 4 propiedades {Nombre, Apellido, Telefono, Ci}. Pero usando *Anonymous Type* se puede escribir el código anterior de la siguiente manera

```
var listaPersonas = from p in Persona
                    where p.Nombre == "Martin"
                  select new { Nombre = p.Nombre, Apellido = p.Apellido, Telefono = p.Telefono, Ci = p.Ci};
```

Se puede observar que no se debe, como en el caso anterior, tener que definir la clase *Presentacion*, en cambio es posible ver la creación de un tipo anónimo (vemos que en luego del *select* se pone un new sin tener necesidad de poner el nombre de la clase), y en este caso al no saber el tipo que devuelve la consulta se está obligado a declarar la variable *listaPersonas* usando *var*.

3.3.5. Query Expressions

Son expresiones que poseen una sintaxis similar a la que maneja el lenguaje SQL, son usadas para manipular datos (va a ser uso cotidiano en LINQ).

A continuación se ve un ejemplo de la utilización de *Query Expressions*

```
var query = from c in boleta
        where c.Cantidad > 3
        where c.Precio < 6.3
        orderby c.RUC
        select new { Nombre = c.Nombre, Valor = c.Precio / c.Cantidad };

foreach (var x in query)
   Console.WriteLine(x);
```

La salida es:

```
{ Nombre = E.SRL, Valor = 1,4 }
{ Nombre = F.SRL, Valor = 1,375 }
```

Una *query expression* comienza con una clausura *form* y finaliza con una clausura *select* o *group*. La clausura *form* es la que especifica el objeto sobre el cual se van a aplicar diversas operaciones.

3.3.6. Lambda Expressions

Esta nueva herramienta es la que acerca el paradigma de la programación funcional. La misma permite definir funciones anónimas que pueden contener expresiones e instrucciones.

Todas las expresiones lambda utilizan el operador lambda = >, que se lee como "va a". El lado izquierdo del operador lambda especifica los parámetros de entrada (representados como una lista delimitada de parámetros delimitador por “,”), mientras que el lado derecho contiene el bloque de expresiones o instrucciones.
Por ejemplo la expresión lambda x => x * x se lee "x va a x veces x". Esta expresión se puede asignar a un tipo de delegado del siguiente modo:

```
delegate int del(int i);
del myDelegate = x => x * x;
int j = myDelegate(5); //j = 25
```

El operador => tiene la misma prioridad que la asignación (=) y es asociativo por la derecha.

En los próximos ejemplos se muestra como fue evolucionando la escritura de códigos en C#, para eso se muestra como se puede escribir el mismo código con las herramientas que se posee en cada versión de C#.

Por ejemplo vemos las siguientes líneas de código donde se define

```
public event EventHandler CuandoTermine; //Declaro un evento

this.CuandoInicie += new EventHandler(ClaseEventos_CuandoInicie);  //Se agrega un handler para ese evento el cual esta definido mas abajo

void ClaseEventos_CuandoInicie(object sender, EventArgs e)
   {
      // Lineas de codigo
   }
```

Luego con C# 2.0 aparecieron los delegados anónimos donde el código anterior queda:

```
public event EventHandler CuandoInicie;

this.CuandoInicie += delegate(object sender, EventArgs e)
{
        // Lineas de codigo
};
```

Y ahora con la introducción de las Lambdas Expressions, provistas por C# 3.0 el código anterior se puede escribir así.

```
public event EventHandler CuandoInicie;
this.CuandoInicie += (o, e) =>
{
          // Lineas de codigo
};
```

Las expresiones lambda son muy usadas en la realización de consultas de LINQ.

4. Desarrollo

4.1. Introducción

LINQ por lo que se pudo ver en capítulos anteriores es una herramienta con grandes beneficios, que a quien quiera usarlo le va a traer buenos resultados en la realización de aplicaciones. El único inconveniente es que no se conoce muy bien cuan performante puede llegar a ser, si el código que se genera utilizando el mismo es sustentable o totalmente desechable, si en verdad es útil para resolver todos los problemas o solo un conjunto de ellos.

Por lo tanto la idea de este capítulo es la de realizar un análisis en profundidad de LINQ, el cual constara de las siguientes tres partes bien identificadas:

1. Realizar un análisis comparativo de LINQ, frente a tecnologías actuales, dicha comparación consta de la realización de dos aplicativos donde uno de ellos utiliza una de las implementaciones de LINQ, y el otro una tecnología similar usada tradicionalmente, luego en base a las mismas se realiza un estudio de performance. El mismo consta de medir a las aplicaciones en términos de tiempo de ejecución (medida en milisegundos), uso de recursos como la CPU (medida en porcentaje de utilización), memoria RAM (medida en byte utilizados), el tiempo de acceso al disco duro (medida en milisegundos) y además el trabajo que llevó realizar las mismas, medida en cantidad de líneas de código escritas.
2. Para complementar ese análisis comparativo, se analizan las principales características que ofrecen las implementaciones de LINQ, en base a las mismas y al estudio de performance, se sacan conclusiones las cuales se encargan de mostrar las ventajas de utilizar la herramienta, indicando además en que situaciones es buena su utilización y en otras no tanto.
3. La tercer parte de este análisis es el de mostrar de que manera se fue introduciendo la utilización del mismo, como parte de la realización de los diferentes proyectos que se realizaron en la empresa Active Software. La idea es la de mostrar cuales fueron los aportes que se realizaron, cuales son los problemas que soluciono, mostrando código utilizado.

Como conclusión del análisis realizado se concluye cuan pertinente puede llegar a ser la utilización de LINQ tanto para la empresa Active Software como para cualquier otra empresa que la utilice.

A continuación se describe como se hizo el análisis, indicando que herramientas se utilizaron, mostrando las pruebas realizadas y los resultados obtenidos.

Antes de entrar en el análisis mismo, se van a mostrar diversos algoritmos que muestran los beneficios de usar LINQ, en términos de productividad y legibilidad.

CASO 1) Extraer todas las líneas de caracteres de un archivo de texto.

File.ReadAllLines	Se utilizo una clase de la librería *System.IO*	Posee una gran desventaja y es que carga todas las líneas en memoria, y después realizo operaciones sobre todas esas líneas que se encuentra en memoria, lo cual resulta en un gasto grande de memoria.
StreamReader reader = new StreamReader("archivo.txt"); var books = from line in reader.Lines() where !line.StartsWith("#") let parts = line.Split(',') select new { Title = parts[1], Publisher = parts[0] };	Se utilizó LINQ, más precisamente LINQ to Object. Cabe destacar que el código de *Lines* es una *extension method* cuyo código se puede observar en la sección códigos auxiliares[1].	Usando LINQ, la ventaja obtenida es la de no tener que cargar todas las líneas del archivo cuando se realiza la consulta (esto es por que LINQ posee ejecución en modo diferido), y permite trabajar sobre la línea sin tenerla aun cargada en memoria, lo mismo pasaría si se trabajara con el conjunto de líneas. Recién se cargan las líneas una vez que se accede a *books*. Igual hay que tener cuidado con la ejecución diferida, comandos como **orderby**, **max**, **min**, etc, hacen que la consulta este obligada a recorrer todas las líneas.

CASO 2) En el caso anterior se pudo ver que si uno no maneja bien las distintas consultas que se pueden realizar con LINQ, puede pasar de un aplicación eficiente a una aplicación menos eficiente, para reafirmar esto se van a mostrar los siguientes códigos que resultan del algoritmo encargado de encontrar el máximo de una lista.

Book maxBook = null;
foreach (var book in books)
{
if ((maxBook == null) || (book.PageCount > maxBook.PageCount))
maxBook = book;
}
	Esta operación se encuentra libre de consultas LINQ, es de O(n), siendo matemáticamente lo mejor basándose en que la *books* (lista de libros) no se encuentra ordenada.

var sortedList = from book in books orderby book.PageCount descending select book; var maxBook = sortedList.First();	En este caso si se utilizan consultas de LINQ, y la estrategia que se utiliza para obtener, el libro que contiene más cantidad de páginas de una lista, lo que se debe de hacer es: ordenarla y obtener el primero de la misma. Este algoritmo al tener que ordenar la lista tiene O(nlogn), lo cual es menos eficiente que el ejemplo anterior
var maxList =from book in books where book.PageCount == books.Max(b => b.PageCount) select book; var maxBook = maxList.First();	Se puede ver en este caso que se está preguntando dentro de la consulta, si el elemento que se está recorriendo es el libro que posee la mayor cantidad de

	páginas. Esto lleva internamente a dos consultas lo cual posee un orden O (n2).

var maxPageCount = books.Max(book => book.PageCount);
var maxList =
from book in books
where book.PageCount == maxPageCount
select book;
var maxBook = maxList.First();
	A continuación se observa una mejora en la consulta mostrada en la fila anterior, y mejorando el orden es O(n).

CASO 3) Este caso lo que muestra es el overhead de LINQ, mostrándose para eso el resultado de los tiempos de las distintas corridas de algoritmos con LINQ y sin LINQ (todos estos resultados se obtuvieron de la siguiente referencia bibliográfica [4]).

	Tiempo promedio	Tiempo medio	Tiempo máximo

var results = new List<Book>();
foreach (var book in books)
{
if (book.PageCount > 500)
results.Add(book);
}
	68	47	384

var results = new List<Book>();
for (int i = 0; i < books.Count; i++)
{
Book book = books[i];
if (book.PageCount > 500)
results.Add(book);
}
	59	42	383

var results = from book in books where book.PageCount > 500 select book;	91	74	404
var results = books.FindAll(book => book.PageCount > 500);	62	51	278

Observando los números “fríamente” se puede observar que el overeada que produce el uso de LINQ, es un poco grande (ahí viene el porqué muchas personas están desconformes con su utilización)

CASO 4) En este caso se va a mostrar una de las razones por las cuales un programador puede optar por LINQ, esa razón es la “prolijidad” del código.

“Agrupar libros por autores”

Con LINQ	**Sin LINQ**

var results =
from book in libros
group book by book.autor.Name into autorlibros
orderby autorlibros.Key
select autorlibros;
	var results = new SortedDictionary<String, IList<Book>>();
	foreach (var book in SampleData.libros)
	{
	IList<Book> autorlibros;
	if (!results.TryGetValue(book.autor.Name,
	out autorlibros))
	{
	autorlibros = new List<Book>();
	results[book.autor.Name] = autorlibros;
	}
	autorlibros.Add(book);
	}

4.2. Pruebas a realizar

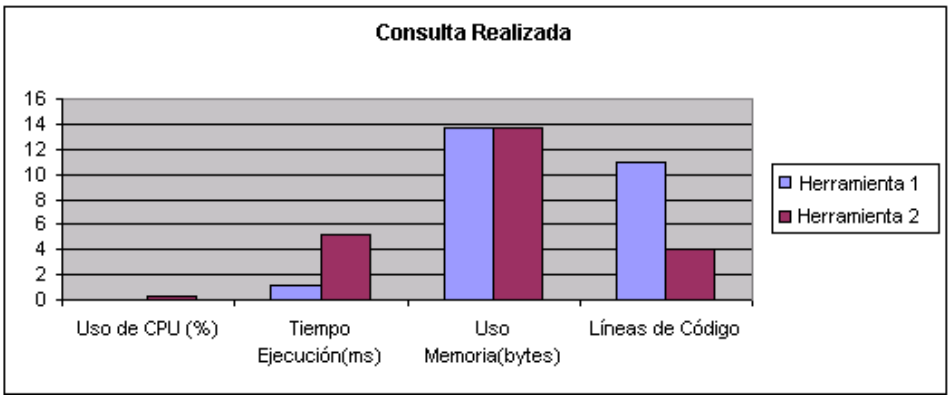
Las pruebas que se van a realizar van a medir la performance en términos de uso de CPU (medida en porcentaje de utilización), uso de memoria (medida en cantidad de bytes utilizados), cantidad de líneas, tiempo de ejecución (medido en milisegundos) y uso de disco duro (se realizara esta medida para los casos de LINQ to XML, LINQ to SQL y LINQ to ENTITY, y se realizara en milisegundos).

Para poder obtener datos certeros de el tiempo de ejecución que puede tardar un programa o el uso de memoria que este necesita para su ejecución, se utiliza un programa que se adapta al Visual Studio 2008 como plugin, el mismo se llama dotTrace de la compañía JetBrains (en las referencias se indica más sobre esta herramienta), el cual realiza profiling sobre la aplicación que se está ejecutando para obtener las distintas mediciones.

Para obtener mayor exactitud en los datos obtenidos se realizaron 10 corridas de las mismas pruebas y en base al cálculo del promedio de las distintas medidas obtenidas, se pudieron obtener las conclusiones. Para visualizar cada resultado se van a guardar en una carpeta todos los snapshot de las distintas corridas (en el caso, del uso de memoria, por incapacidad de la herramienta al no poder obtener esta medida junto con las demás, solo se va a obtener un snapshot distinto para la misma, para luego en base a estos completar la siguiente tabla.

Nº Corrida	*Uso de CPU (%)*	*Uso Memoria(bytes)*	*Líneas de Código*	*Uso Disco Duro(ms)*
1 . . 10				
Promedio				

Además para que los resultados sean comparativamente mejor visualizados, para su mejor entendimiento se realizó por cada par de tablas una grafica que indicando las diferentes propiedades por lo cual se realiza la comparación de dos tecnologías. A continuación se muestra un ejemplo de la realización de esta grafica:

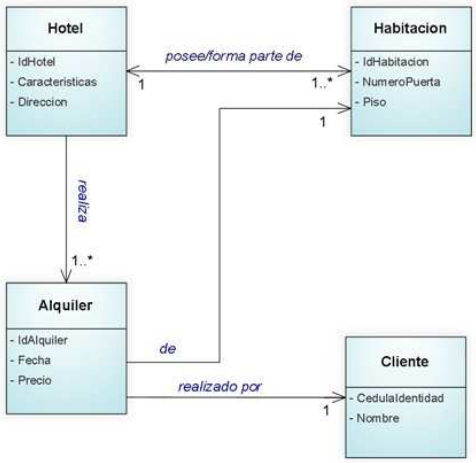


Cabe destacar que las tablas no se incluyeron en este informe final si en los anexos.

4.2.1. LINQ to OBJECT

4.2.1.1. Introducción

Se desea mostrar cuan eficiente es utilizar LINQ to OBJECT en comparación con el manejo habitual que se tiene de las colecciones de objetos. Para comprobar esta eficiencia se realizaron distintas tipos de consultas, las cuales representan casos de uso comunes. Para realizar las mismas se basó en la siguiente realidad: "Un complejo hotelero desea conocer cuáles fueron los clientes, que alquilaron habitaciones y en qué momento", para implementar esto se diseño el siguiente modelo de dominio.

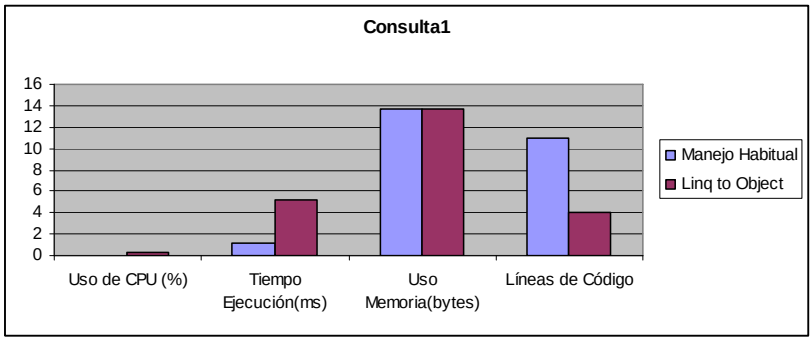


4.2.1.2. Resultados

A continuación se van a nombrar las consultas que se realizaron sobre las diferentes tecnologías y en base a eso se va a mostrar una grafica mostrando los resultados que se obtuvieron de forma comparativa.

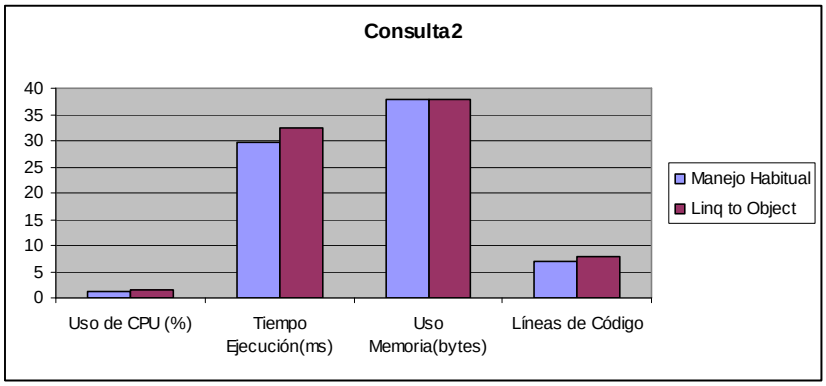
Consulta 1

"Desplegar el Piso y el numero de puerta de las habitaciones en el hotel. "



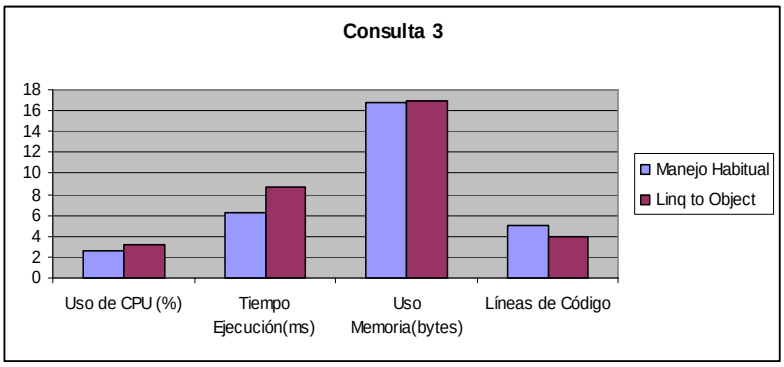
Consulta 2

"Que habitaciones alquilo un cliente en un hotel y a qué precio las alquilo".



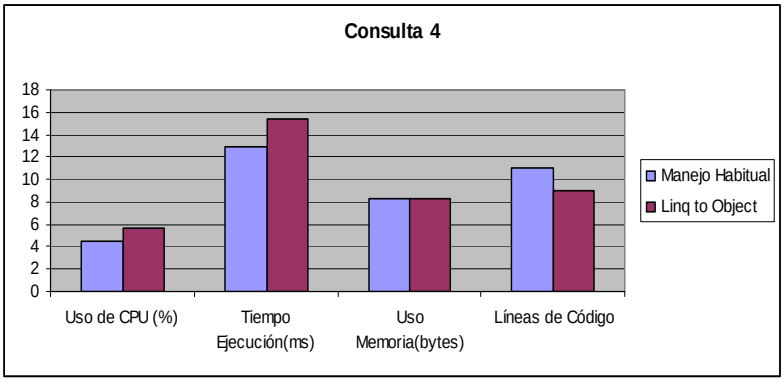
Consulta 3

"Qué cantidad de alquileres se realizaron en un periodo de tiempo determinado".



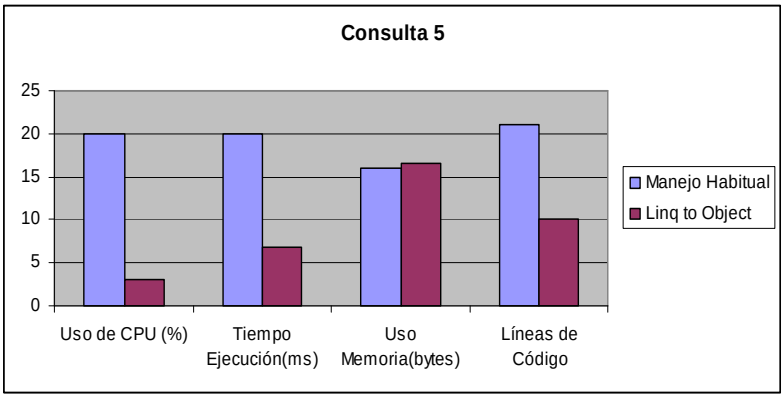
Consulta 4

"Devolver que habitación de un hotel, a qué precio y por quien fue alquilada en un periodo de tiempo determinado".



Consulta 5

"Devolver todos los clientes de un hotel que hayan alquilado alguna habitación cuya cedula es menor a un determinado valor". El resultado tiene que estar ordenado por nombre del cliente.



4.2.1.3. Ejemplos de Códigos

A continuación se muestran los códigos utilizados para realizar la Consulta 5, tanto utilizando LINQ como utilizando el manejo tradicional de colecciones.

Manejo Tradicional

```
List<Cliente> clientesAux = new List<Cliente>();
List<Cliente> clientes = new List<Cliente>();
Hotel[] hoteles = new Hotel[complejoHotelero.Count()];
int indice = 0;
int i = 0,j=0;
complejoHotelero.CopyTo(hoteles, 0);

while (hoteles[indice].Nombre != NomHotel)
{
    indice++;
}
foreach (Alquiler alquiler in hoteles[indice].Alquileres)
{
    if (alquiler.Cliente.Ci < Cedula)
    {
        clientesAux = new List<Cliente>();
        while ((i < clientes.Count) &&
 (clientes[i].Nombre.CompareTo(alquiler.Cliente.Nombre) <= 0))
        {
            i++;
        }
        for (j = 0; j < i ; j++)
        {
            clientesAux.Add(clientes[j]);
        }
        clientesAux.Add(alquiler.Cliente);
        if (i < clientes.Count)
        {
            for (j = i + 1; j < clientes.Count; j++)
            {
                clientesAux.Add(clientes[j]);
            }

        }
        clientes = new List<Cliente>(clientesAux);
    }
}
return clientes;
```

LINQ to OBJECT

```
List<Cliente> clientes = new List<Cliente>();
var solucion = from hotel in complejoHotelero
               where hotel.Nombre == NomHotel
               select (from alq in hotel.Alquileres
                       where alq.Cliente.Ci < Cedula
                       orderby alq.Cliente.Nombre ascending
                       select alq.Cliente);


foreach (var elemento in solucion.Single())
{
    clientes.Add(elemento);
}
return clientes;
```

Si se desea saber cuál es el código utilizado para la realización de las diferentes pruebas, se debe de ir a los anexos donde se encuentran los códigos de las pruebas anteriores.

4.2.1.4. Conclusiones Obtenidas

En base a las pruebas que se realizaron se puede afirmar que es útil la utilización de LINQ para el manejo de objetos, pero no siempre conveniente, dependiendo mucho de la situación en que se lo quiera utilizar.

Igual se puede destacar que:

- La cantidad de líneas de código que se necesitan para realizar una consulta con LINQ, es mucho menor que hacer la misma consulta con el manejo habitual de colecciones.
- A medida que la consulta es cada vez más compleja (con compleja se hace referencia a que para obtener el resultado de la misma se necesita iterar sobre muchas listas involucradas entre sí) es cada vez más recomendable usar LINQ, porque el uso de recursos es cada vez menor. Esto se puede ver en las graficas de las distintas pruebas, se observa que cuanto más compleja es la consulta los tiempos de LINQ mejoran (en comparación al otro).

4.2.2. LINQ to XML

4.2.2.1. Introducción

Para saber si en verdad es conveniente utilizar LINQ to XML en vez de la tecnología actual provista por .NET Framework (MSXML). MSXML (Microsoft XML Core Services), esta API implementa DOM (Document Object Model) la cual es un modelo de datos que permite acceder a XML (en las referencias se puede encontrar más información sobre esta tecnología en sus diferentes versiones).

Las distintas pruebas que realizan están basadas en la siguiente estructura XML, la cual describe lo siguiente: "Mostrar de cada país, las bibliotecas existentes en el mismo, donde se sabe su nombre, dirección y los libros que posee, donde cada libro se identifica por el nombre del mismo y el autor que lo escribió".

```
<Paises>
  <Pais name="nomPais">
    <Biblioteca name="nomBiblioteca"  address="dirBiblioteca">
      <Libro genero="tipoGenero">
        <Nombre>  nomLibro </Nombre>
        <Autor> nomAutor </Autor>
      </Libro>
      <Libro genere="...">
        ....
      </Libro>
      ....
    </Biblioteca>
    <Biblioteca name="..." address="...">
        .....
    </Biblioteca>
    .....
  </Pais>
  <Pais name="">
    .......
  </Pais>
  .......
</Paises>
```

Las pruebas realizadas son consultas, modificaciones, creaciones y eliminaciones de elementos de un XML, así como también se realizan pruebas sobre el manejo de NameSpace en el mismo.

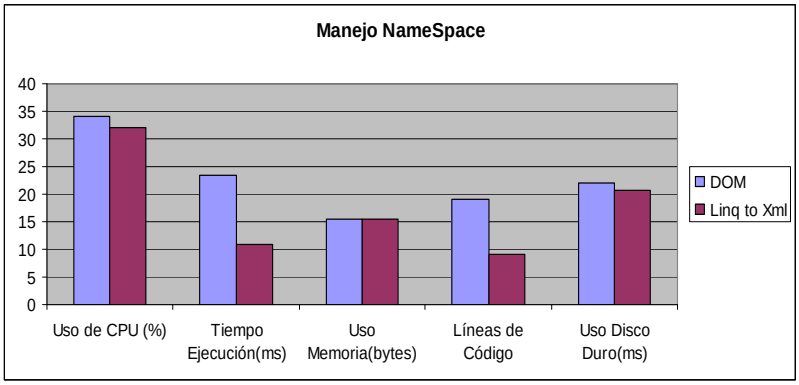
4.2.2.2. Resultados

Prueba 1 (Manejo de NameSpace)

Esta prueba trata sobre realizar la consulta de un documento XML, el cual se encuentra formado por dos NameSpace llamados `www.fing.edu.uy/biblioteca` y `www.fing.edu.uy/libreria` el documento tiene el siguiente formato:

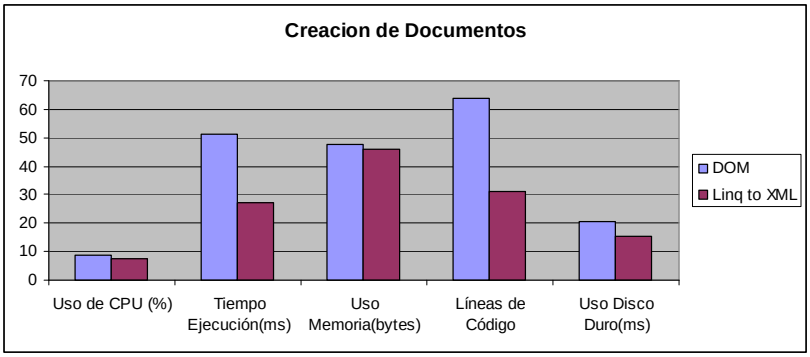
```
<Paises>
  <Pais name="Uruguay">
    <biblioteca:Biblioteca biblioteca:name="nombre0"
biblioteca:address="direccion0" xmlns:biblioteca="www.fing.edu.uy/biblioteca">
      <biblioteca:Libro biblioteca:genero="genero0">
        <biblioteca:Nombre>nombre0</biblioteca:Nombre>
        <biblioteca:Autor>autor0</biblioteca:Autor>
      </biblioteca:Libro>
      <biblioteca:Libro biblioteca:genero="genero1">
        <biblioteca:Nombre>nombre1</biblioteca:Nombre>
        ......
        .....
      </biblioteca:Libro>
      <libreria:Libreria libreria:name="nombre15" libreria:address="direccion15"
xmlns:libreria="www.fing.edu.uy/libreria">
        <libreria:Libro libreria:genero="genero25">
          <libreria:Nombre>nombre25</libreria:Nombre>
          <libreria:Autor>autor25</libreria:Autor>
        </libreria:Libro>
        <libreria:Libro libreria:genero="genero26">
          <libreria:Nombre>nombre26</libreria:Nombre>
          .......
          ........
        </libreria:Libro>
      </Pais>
       </Paises>
```

El cometido de la consulta es: *"obtener todos los libros que pertenecen a las bibliotecas y que no pertenecen a las librerías "* (pueden tener valores repetidos).



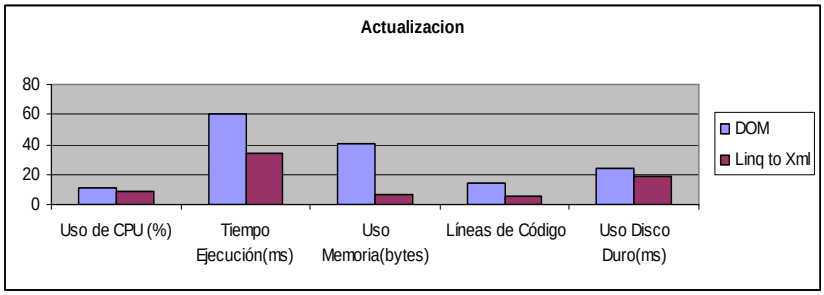
Prueba 2 (Creación de Documentos)

Se realizó la creación de un documento XML, el cual se almacena en disco duro manteniendo la estructura nombrada en la *Introducción*.



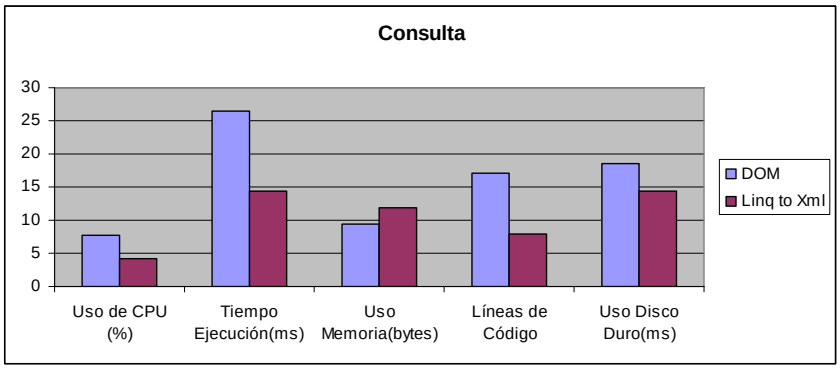
Prueba 3 (Actualización)

En esta prueba se desea poder modificar algún elemento de los que conforman al archivo xml. En este caso preciso se desea "*modificar el nombre de un libro de un autor determinado*".



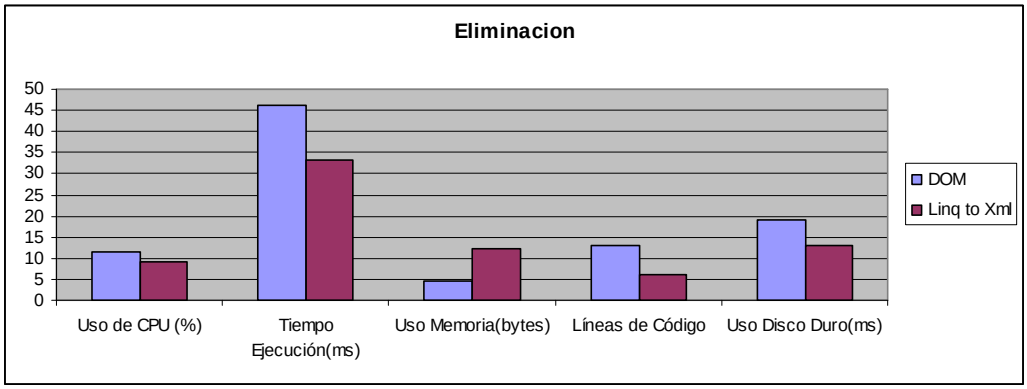
Prueba 4 (Consulta)

Lo que se quiere realizar con esta operación es: "*obtener todos los libros de un autor, indicando además en que biblioteca se encuentran*".



Prueba 5 (Eliminación)

En esta prueba se desea poder modificar algún elemento de los que conforman al archivo xml. En este caso preciso se desea "*modificar el nombre de un libro de un autor determinado*".



4.2.2.3. Ejemplos de Códigos

A continuación se muestran los códigos utilizados para realizar la Consulta, tanto utilizando LINQ to XML como utilizando DOM.

DOM

```
List<String> listaBibliotecas = new List<String>();
XmlDocument documento = new XmlDocument();
XmlNodeList paises, bibliotecas, libros, elementos, autores;
FileStream docIn = new FileStream(path + @"\Comun.xml", FileMode.Open);
documento.Load(docIn);
paises = documento.GetElementsByTagName("Pais");
foreach (XmlNode nodo in paises)
{
    bibliotecas = nodo.ChildNodes;
    foreach (XmlNode biblioteca in bibliotecas)
    {
        libros = biblioteca.ChildNodes;
        foreach (XmlNode libro in libros)
        {
            elementos = libro.ChildNodes;
            foreach (XmlNode elemento in elementos)
            {
                if ((elemento.Name == "Autor")
                    && (elemento.InnerText == autorentrada))
                {
                    listaBibliotecas.Add("Biblioteca: "
                    + biblioteca.Attributes.GetNamedItem("name").Value
                    + "----  Pais: " +
                    nodo.Attributes.GetNamedItem("name").Value); ;
                }
            }
        }

    }
}
docIn.Close();
return listaBibliotecas;
```

LINQ to XML

```
XElement elemento = XElement.Load(path + @"\Comun.xml");
var solucion = from pais in elemento.Elements("Pais")
               select (from biblioteca in pais.Elements("Biblioteca")
                       select (from libro in biblioteca.Elements("Libro")
                               select (from autor in libro.Elements("Autor")
                                       where autor.Value == autorentrada
                                       select "Biblioteca: "
                                       + biblioteca.Attribute("name").Value
                                       + "--------- Pais: "
                                       + pais.Attribute("name").Value)));
return solucion.ElementAt(0).ElementAt(0).ElementAt(0).ToList<string>();
```

Si se desea saber que código se utilizó para la realización de las diferentes pruebas, se debe de ir a los anexos donde se encuentran los códigos de las pruebas anteriores.

4.2.2.4. Conclusiones Obtenidas

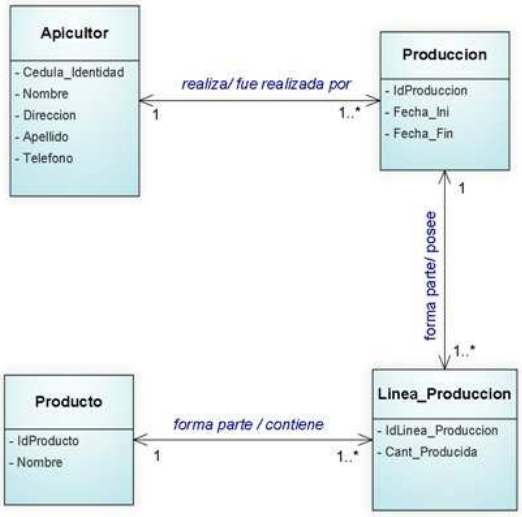
A continuación en base a las pruebas realizadas, se puede afirmar que LINQ to XML es la opción a elegir sobre otras tecnologías de manejo de XML para .Net, visto que en todas las pruebas que se realizaron, y observando las graficas obtenidas se puede ver que el uso de LINQ to XML no solo realiza las pruebas en la menor cantidad de código (cualidad repetida en todas las implementaciones de LINQ), sino que también lo hace utilizando muy pocos recursos. Aparte de las pruebas realizadas cabe destacar que:

- El manejo que ofrece sobre los espacio de nombres y prefijo de nombres, hace que el manipularlos sea muy sencillo, lo cual para las tecnologías tradicionales resulta un problema.
- Un documento XML se divide en una determinada cantidad nodos. Para actualizar un valor de un determinado nodo, primero se lo debe de eliminar del documento y luego agregar. LINQ to XML permite resolver este problema actualizando el nodo.

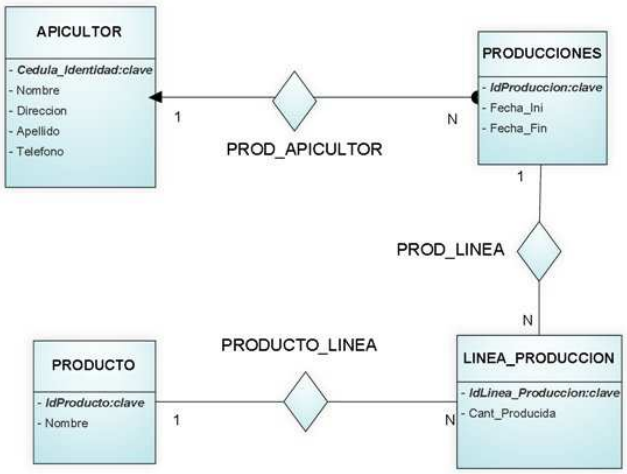
4.2.3. LINQ to SQL

4.2.3.1. Introducción

En las siguientes pruebas se quiere mostrar cuan eficiente es usar LINQ to SQL frente a la acceso directo a la base de datos a través de consultas escritas en SQL.



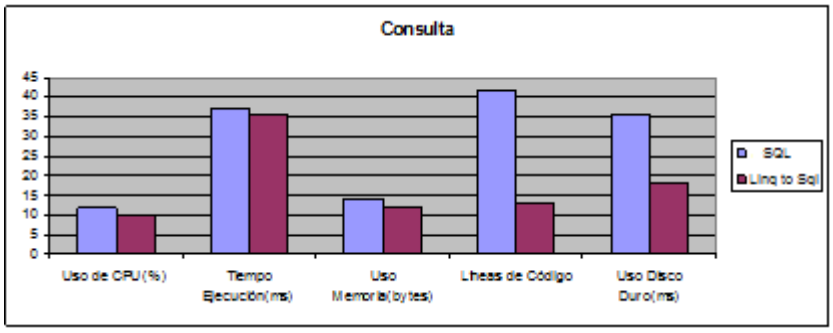
Las pruebas a realizar se basan en la siguiente realidad: "Administración de la producción de un Apicultor". El diseño de clases en el cual se basa la implementación de la capa lógica se muestra a la derecha, y el MER utilizado para diseñar las tablas de las base de datos donde se va a persistir (el manejador de base de datos es SQL Server), se muestra en el diagrama de abajo.



4.2.3.2. Resolución

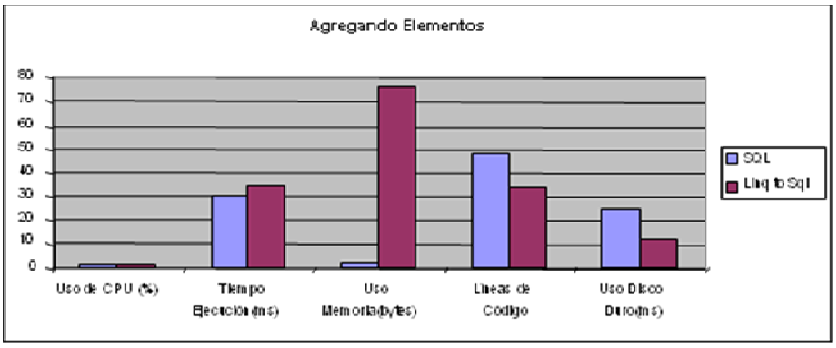
Prueba 1 (Consulta)

En esta prueba se realiza una consulta para ver cuán eficiente es LINQ, la consulta dice así: "Que productos y qué cantidad de estos, produjo un Apicultor en un periodo de tiempo estipulado".



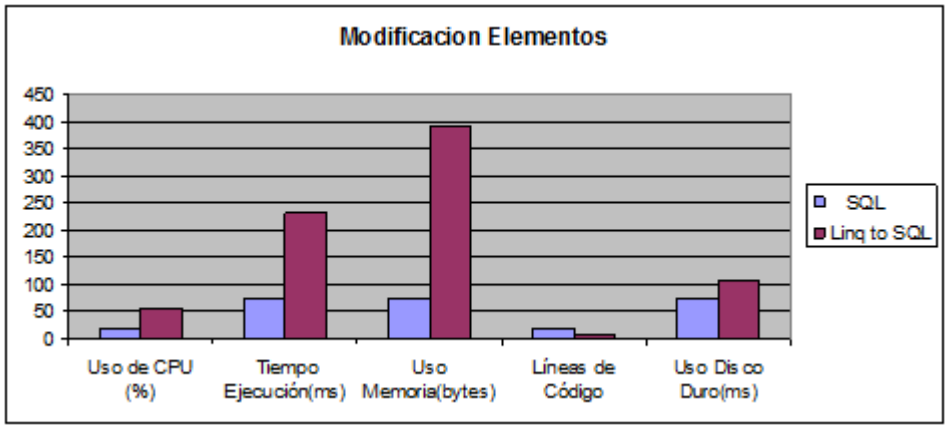
Prueba 2 (Agregando Elementos)

Se va a prueba cuán bien trabaja LINQ to SQL frente al agregado de elementos a la base de datos. En esta ocasión la agregación se va a producir implementado lo siguiente: “En un periodo de tiempo un Apicultor realizo la producción de dos productos, ya conocidos por el sistema”.



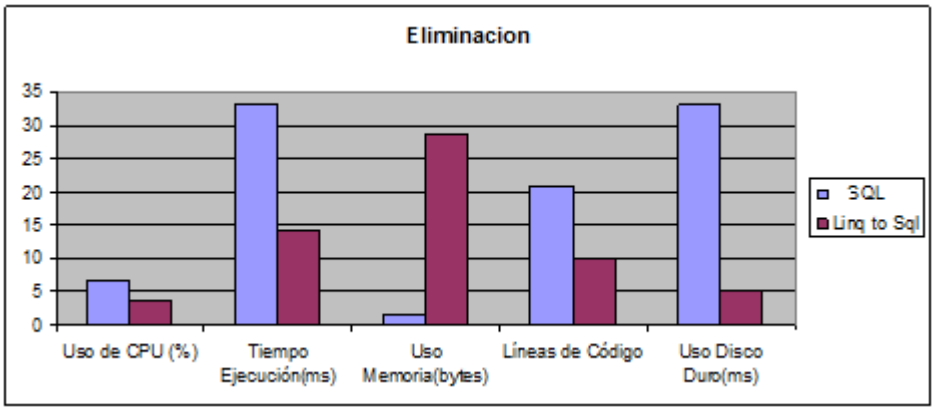
Prueba 3 (Modificar Elementos)

En este caso, se prueba la modificación de elementos de la base de datos, siguiendo lo siguiente: “Modificar la cantidad producida de un Producto en un periodo de tiempo determinado”.



Prueba 4 (Eliminación)

Se prueba como funciona LINQ to SQL, en la eliminación de elementos de una tabla de una base de datos, para eso se implementará lo siguiente: “Eliminar todas las producciones que realizo un Apicultor de un producto determinado”.



4.2.3.3. Ejemplos de Códigos

A continuación se muestran los códigos utilizados para realizar la Consulta, tanto utilizando LINQ to SQL como utilizando SQL.

SQL

```
DataTable table = new DataTable();
SqlCommand command = conexion.CreateCommand();

command.CommandText = @"SELECT [t4].[Nombre] AS [NombreProducto], [t2].[Cant_Producida], (
                        SELECT COUNT(*)
                        FROM [dbo].[Produccion] AS [t5]
                        INNER JOIN [dbo].[Linea_Produccion] AS [t6] ON [t5].[IdProduccion] = ((
                            SELECT [t7].[IdProduccion]
                            FROM [dbo].[Produccion] AS [t7]
                            WHERE [t7].[IdProduccion] = [t6].[IdProduccion]
                            ))
                        INNER JOIN [dbo].[Producto] AS [t8] ON [t8].[IdProducto] = [t6].[IdProducto]
                        WHERE ([t5].[Fecha_Ini] > @Fecha_Ini) AND ([t5].[Fecha_Fin] < @Fecha_Fin)
                        AND ([t5].[Cedula_Identidad] = [t0].[Cedula_Identidad])
                        ) AS [value]
                    FROM [dbo].[Apicultor] AS [t0]
                    LEFT OUTER JOIN ([dbo].[Produccion] AS [t1]
                        INNER JOIN [dbo].[Linea_Produccion] AS [t2] ON [t1].[IdProduccion] = ((
                            SELECT [t3].[IdProduccion]
                            FROM [dbo].[Produccion] AS [t3]
                            WHERE [t3].[IdProduccion] = [t2].[IdProduccion]
                            ))
                        INNER JOIN [dbo].[Producto] AS [t4] ON [t4].[IdProducto] = [t2].[IdProducto]) ON
([t1].[Fecha_Ini] > @Fecha_Ini) AND ([t1].[Fecha_Fin] < @Fecha_Fin) AND ([t1].[Cedula_Identidad] = [t0].[Cedula_Identidad])
                    WHERE (CONVERT(NVarChar,[t0].[Cedula_Identidad])) = @Ci_Apicultor
                    ORDER BY [t0].[Cedula_Identidad], [t1].[IdProduccion], [t2].[IdLineaProduccion], [t4].[IdProducto]
                    ";


 command.Parameters.Add("@Fecha_Ini",System.Data.SqlDbType.DateTime);
 command.Parameters["@Fecha_Ini"].Value = Fecha_Ini ;

 command.Parameters.Add("@Fecha_Fin",System.Data.SqlDbType.DateTime);
 command.Parameters["@Fecha_Fin"].Value = Fecha_Fin ;

 command.Parameters.Add("@Ci_Apicultor", System.Data.SqlDbType.VarChar);
 command.Parameters["@Ci_Apicultor"].Value = Ci_Api;

 adapter = new SqlDataAdapter(command);
 adapter.Fill(table);
 grid.DataSource = table;
 command.Dispose();
 adapter.Dispose();
```

LINQ to SQL

```
List<Solucion> solucion = new List<Solucion>();
var sol = from apicultor in _Apicultor
          where apicultor.Cedula_Identidad.ToString() == CI_Api
          select from produccion in apicultor.Producciones
                 where produccion.Fecha_Ini > Fecha_Ini && produccion.Fecha_Fin < Fecha_Fin
                 join linea_produccion in _LineaProduccion on
                 produccion.IdProduccion equals linea_produccion.Produccion.IdProduccion
                 select new {NombreProducto=  linea_produccion.Producto.Nombre ,
                     Cant_Producida =  linea_produccion.Cant_Producida };

grid.DataSource = new BindingSource();
((BindingSource)grid.DataSource).DataSource =sol.Single().ToList();
```

Si se desea saber que código se utilizó para la realización de las diferentes pruebas, se debe de ir a los anexos donde se encuentran los códigos de las pruebas anteriores.

4.2.3.4. Conclusiones Obtenidas

Sin lugar a dudas se ve que accediendo directamente a la base de datos usando SQL es mucho más performante que usar LINQ, entonces se concluiría que la utilización de LINQ puede resultar poco performante, aunque el esfuerzo que lleva realizar una aplicación basada en LINQ es mucho menor que usando SQL.

Aunque no sea performante, se puede afirmar a través de las siguientes razones que la utilización de esta herramienta, puede resultar muy útil en el desarrollo de aplicaciones con base de datos Relacional:

- Como se puede ver en el documento de Factibilidad de LINQ, la cantidad de líneas de los diferentes códigos escritos usando LINQ es mucho menor que hacerlo directamente usando SQL.
- Además una ventaja que le da LINQ to SQL, al programador es el poder chequear en tiempo de ejecución la validación de sus consultas, lo cual era imposible usando directamente SQL, visto que las mismas son escritas sobre un *string.*.
- LINQ to SQL además ofrece acceso a diferentes elementos de la base de datos, permitiendo tratar a cada tabla de la misma como si fuera un objeto, lo cual permite solventar el abismo que existe entre la orientación a objeto y el modelo Relacional.
- Para tener una aplicación sostenible y sustentable es necesario siempre dividirla en distintas capas, a esto me refiero con que si tengo que almacenar algún dato en la base de datos es de esperarse que se tenga una capa de persistencia y una capa de negocio. Bueno LINQ to SQL realiza esta división permitiéndonos trabajar con objetos sin importar de donde provinieren los datos que contiene.

4.2.4. LINQ to ENTITY

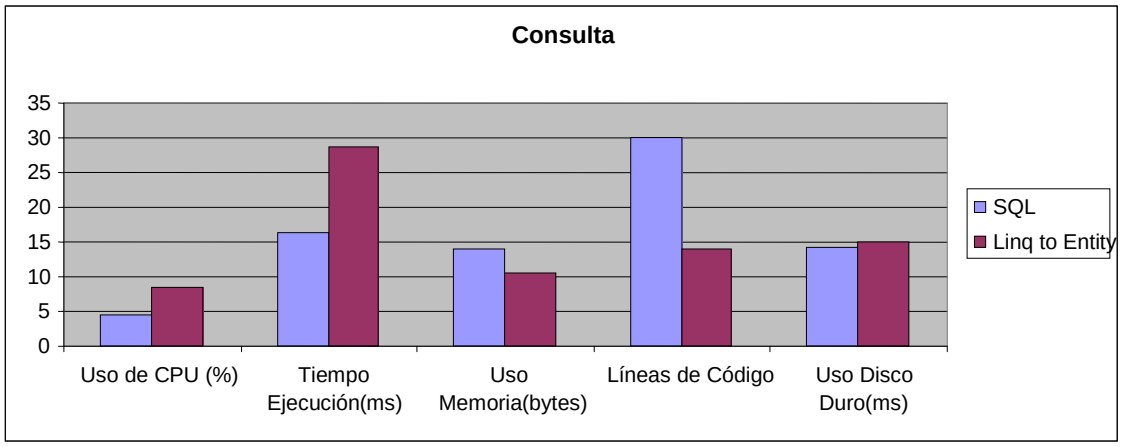
4.2.4.1. Introducción

Las pruebas que se realizaron se basan en el modelo ya propuesto en el análisis de LINQ to SQL, además las pruebas que se van a utilizar para comparar LINQ to ENTITY frente al acceso a una base de datos utilizando SQL (son las mismas que se utilizaron para con LINQ to SQL).

4.2.4.2. Resolución

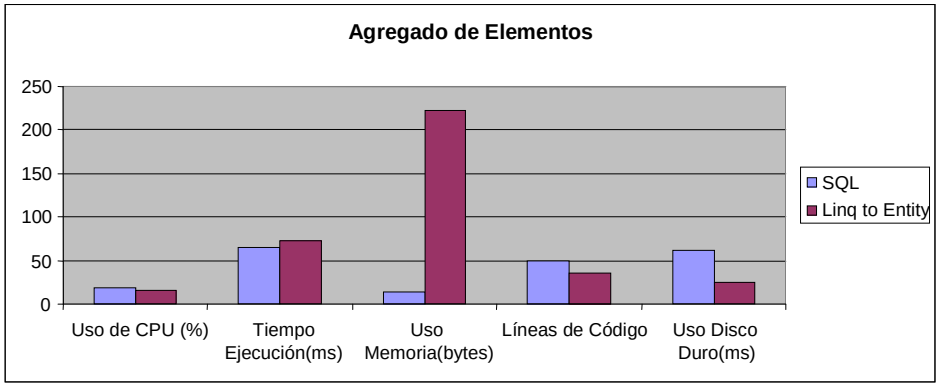
Prueba 1 (Consulta)

En esta prueba se va a realizar una consulta para ver cuán eficiente es LINQ, la consulta dice así: "Que productos y qué cantidad de estos, produjo un Apicultor en un periodo de tiempo estipulado".



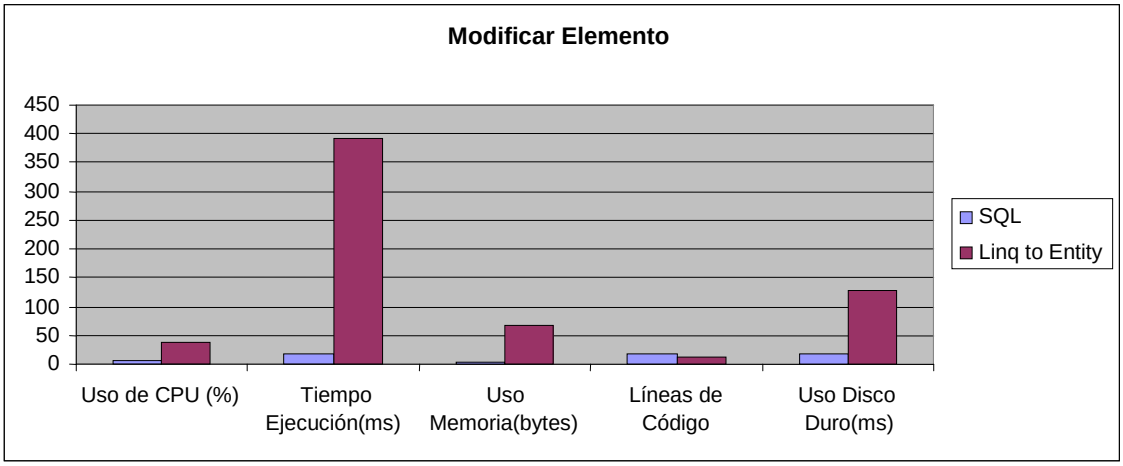
Prueba 2 (Agregado de Elementos)

Se va a prueba cuán bien trabaja LINQ to SQL, frente al agregado de elementos a la base de datos. En esta ocasión la agregación se va a producir implementado lo siguiente: "En un periodo de tiempo un Apicultor realizo la producción de dos productos, ya conocidos por el sistema".



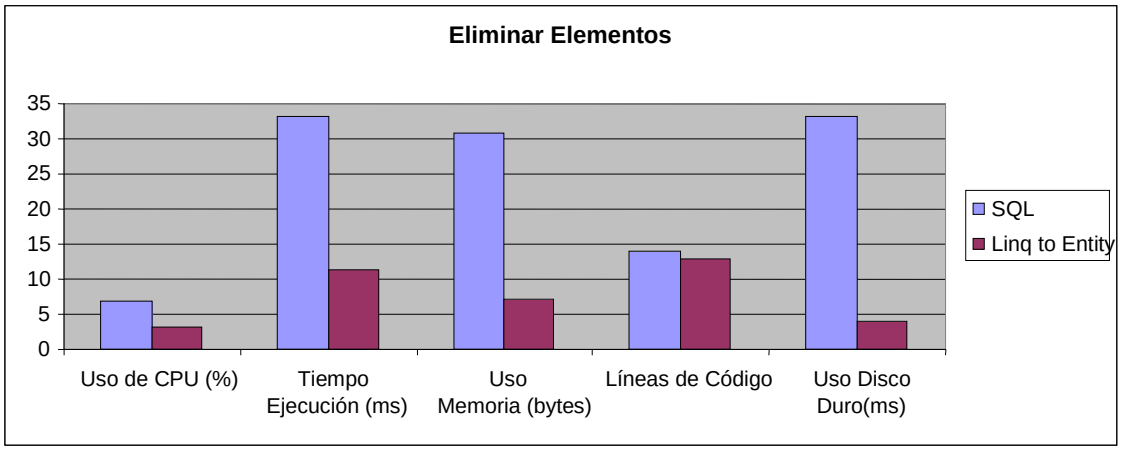
Prueba 3 (Modificar Elementos)

En este caso, se prueba la modificación de elementos de la base de datos, siguiendo lo siguiente: “Modificar la cantidad producida de un Producto en un periodo de tiempo determinado”.



Prueba 4 (Eliminación)

Se prueba como funciona LINQ to SQL, en la eliminación de elementos de una tabla de un base de datos, para eso se implementará lo siguiente: “Eliminar todas las producciones que realizo un Apicultor de un producto determinado”.



4.2.4.3. Ejemplos de Códigos

A continuación se muestran los códigos utilizados para realizar la Consulta, tanto utilizando LINQ to ENTITY como utilizando SQL.

SQL

```
DataTable table = new DataTable();
SqlCommand command = conexion.CreateCommand();

command.CommandText = @"SELECT [t4].[Nombre] AS [NombreProducto], [t2].[Cant_Producida], (
                        SELECT COUNT(*)
                        FROM [dbo].[Produccion] AS [t5]
                        INNER JOIN [dbo].[Linea_Produccion] AS [t6] ON [t5].[IdProduccion] = ((
                            SELECT [t7].[IdProduccion]
                            FROM [dbo].[Produccion] AS [t7]
                            WHERE [t7].[IdProduccion] = [t6].[IdProduccion]
                            ))
                        INNER JOIN [dbo].[Producto] AS [t8] ON [t8].[IdProducto] = [t6].[IdProducto]
                        WHERE ([t5].[Fecha_Ini] > @Fecha_Ini) AND ([t5].[Fecha_Fin] < @Fecha_Fin)
                        AND ([t5].[Cedula_Identidad] = [t0].[Cedula_Identidad])
                        ) AS [value]
                    FROM [dbo].[Apicultor] AS [t0]
                    LEFT OUTER JOIN ([dbo].[Produccion] AS [t1]
                        INNER JOIN [dbo].[Linea_Produccion] AS [t2] ON [t1].[IdProduccion] = ((
                            SELECT [t3].[IdProduccion]
                            FROM [dbo].[Produccion] AS [t3]
                            WHERE [t3].[IdProduccion] = [t2].[IdProduccion]
                            ))
                        INNER JOIN [dbo].[Producto] AS [t4] ON [t4].[IdProducto] = [t2].[IdProducto]) ON
([t1].[Fecha_Ini] > @Fecha_Ini) AND ([t1].[Fecha_Fin] < @Fecha_Fin) AND ([t1].[Cedula_Identidad] = [t0].[Cedula_Identidad])
                    WHERE (CONVERT(NVarChar,[t0].[Cedula_Identidad])) = @Ci_Apicultor
                    ORDER BY [t0].[Cedula_Identidad], [t1].[IdProduccion], [t2].[IdLineaProduccion], [t4].[IdProducto]
                    ";


 command.Parameters.Add("@Fecha_Ini",System.Data.SqlDbType.DateTime);
 command.Parameters["@Fecha_Ini"].Value = Fecha_Ini ;

 command.Parameters.Add("@Fecha_Fin",System.Data.SqlDbType.DateTime);
 command.Parameters["@Fecha_Fin"].Value = Fecha_Fin ;

 command.Parameters.Add("@Ci_Apicultor", System.Data.SqlDbType.VarChar);
 command.Parameters["@Ci_Apicultor"].Value = Ci_Api;

 adapter = new SqlDataAdapter(command);
 adapter.Fill(table);
 grid.DataSource = table;
 command.Dispose();
 adapter.Dispose();
```

LINQ to ENTITY

```
int cedula = Int32.Parse(CI_Api);
var sol = from apicultor in api.Apicultor.Include("Produccion")
          where apicultor.Cedula_Identidad == cedula
          from produccion in apicultor.Produccion
          where  produccion.Fecha_Ini > Fecha_Ini && produccion.Fecha_Fin.Value < Fecha_Fin
          join linea_produccion in api.Linea_Produccion.Include("Produccion") on
          produccion.IdProduccion equals linea_produccion.Produccion.IdProduccion
          select new { NombreProducto = linea_produccion.Produccion.Nombre,
              Cant_Producida = linea_produccion.Cant_Producida };


grid.DataSource = new BindingSource();
((BindingSource)grid.DataSource).DataSource = sol.ToList();
```

Si se desea saber que código se utilizó para la realización de las diferentes pruebas, se debe de ir a los anexos donde se encuentran los códigos de las pruebas anteriores.

4.2.4.4. Conclusiones Obtenidas

Observando el resultado (visualizados a través de las graficas) de las distintas pruebas que se realizaron y tomando en cuenta que las mismas se realizaron utilizando directamente el acceso a base de datos, a través de las sentencias SQL, por cada prueba se elimina la memoria cache en cada corrida (lo cual produce que los tiempos de ejecución, uso de CPU, etc., sean aun mayores), se puede concluir que la performance LINQ to ENTITY no es una de sus virtudes, pero en cambio ofrece otras características que se nombraran a continuación que hacen opacar esta desventaja:

- Permite dividir la aplicación en dos capas bien diferenciadas la capa de negocio (donde se manejan los objetos) y la capa de persistencia (donde se encuentra la base de datos), donde la capa de negocio no tiene noción a que manejador de base de datos esta accediendo, lo cual es una gran ventaja para el desarrollador de la capa de negocio de la aplicación por que no tiene que pensar con que lenguaje de base de datos tiene que tratar.
- Ofrece la posibilidad de manipular objetos, siendo estos generados por el mapeo de distintas tablas de la base de datos. El manejar objetos ofrece grandes ventajas sobre el manejo directo sobre tablas, algunas de esas ventajas son:
    - Permitir la corrección de la sintaxis de una consulta que accede a información provista por la base de datos en tiempo de ejecución.
    - Ofrece la posibilidad de no necesitar aprender uno o varios lenguajes dedicados al manejo de base de datos, ya con saber la sintaxis provista por C# es suficiente.

Cabe tener en cuenta el siguiente link que habla sobre la performance de cada componente que conforma a LINQ to ENTITY:
http://blogs.msdn.com/adonet/archive/2008/02/04/exploring-the-performance-of-the-ado-net-entity-framework-part-1.aspx

4.3. Aplicabilidad de LINQ en Active Software

4.3.1. Introducción

Active Software es una empresa de desarrollo de aplicaciones y servicios. La gran mayoría de las aplicaciones que desarrolla se encuentran basadas en el Framework de .Net (más específicamente en lenguaje C#), por eso es bueno ver cómo fue su experiencia con el manejo de esta herramienta y en qué casos empleo la misma.

La utilización de LINQ por parte de la empresa, se fue realizando de forma paulatinamente en el desarrollo de los diferentes productos, a medida que se iban observando distintos beneficios que ofrecía para ciertas situaciones, siendo en muchos casos una solución menos costosa, y en otros casos resolviendo grandes dolores de cabeza, como por ejemplo en el caso que se quiere ordenar una colección bajo una determinada propiedad, resolver este problema de forma tradicional (sin la utilización de LINQ) podría llevar una cantidad de líneas de código más que al hacerlo utilizando LINQ.

La utilización de la función *Where*, junto con funciones lambda, se ve como una gran idea a utilizar (ya se puede ver en ejemplos de códigos realizados por la empresa esta idea en ejecución), visto que se puede armar un filtro por fuera y luego se lo puede pasar como parámetro a dicha función, para lograr el cometido de obtener un subconjunto de una colección con elementos que cumplen con una determinada condición.

Además una vez visto lo beneficioso que resulto el uso de LINQ to OBJECT, y al necesitar manipular XML, y al ver el resultado positivo del análisis hecho sobre LINQ to XML, entonces surgió la posibilidad de utilización de la misma para la manipulación de XML (también se puede observar el manejo del mismo en los códigos mostrados a continuación).

LINQ to SQL, LINQ to ENTITY, no se utilizaron por la empresa visto que se posee un Framework encargado de realizar el mismo trabajo, aunque si se realizo un análisis comparativo entre este y LINQ to ENTITY (será desarrollado en el capitulo Aplicación Desarrollada).

LINQ to DataSet, no llego a utilizarse por no haber nuevos proyectos donde existiese la necesidad de manipular el contenido de DataSets.

4.3.2. Ejemplos

A continuación se van a mostrar distintos trozos de códigos, los cuales fueron sacados de los diferentes proyectos realizados por la empresa Active Software. Dichos códigos muestran de qué manera se utilizo LINQ para resolver diferentes problemas, en diversas situaciones.

Nombre Proyecto: Jouve

Funcionalidades: Este producto una de las tantas funcionalidades que tiene es la de leer XML, para luego analizar su contenido, para luego más tarde que el usuario realiza unos cambios sobre el mismo (vía una interfaz de usuario amigable), se actualizan los XML para ser enviados a otro destinatario.

Función: El siguiente trozo de código se corresponde a un trozo de la función `CheckXMLResult`, el objetivo de la misma es chequear que lo valores leídos a partir de un documento XML, cumplen con ciertas condiciones establecidas.

Dicho código que se muestra a continuación utiliza para el acceso a la información del XML, a la herramienta LINQ to XML (la utilización de la misma se encuentra incluida dentro de los cuadros rojos).

```
documento = XDocument.Load(file);
var items = from item in documento.Descendants("image")
            select item;

if (items.Count() != _ImageObjList.Count)
{
    msgError = String.Format("Problema de Integridad en {0}. No coincide la cantidad de imagen
    return false;
}

foreach (XElement item in items)
{
    image = item.Attribute("nomimage").Value;
    idImage = Convert.ToInt32(image.Substring(0, image.Length - 4));

    var champs = from champ in item.Descendants("champ")
                 select champ;

    imgObj = _ImageObjList.ToList().Find(im => im.Name == image);
    if (imgObj == null)
    {
        msgError = String.Format("Problema de Integridad en {0}. No existe la imagen {1}.", _R
        return false;
    }

   foreach (XElement campo in champs)
   {
       fieldObj = imgObj.FieldObjList.ToList().Find(f => f.Name == campo.Attribute("id").Value
       if (fieldObj == null)
       {
           msgError = String.Format("Problema de Integridad en {0}. No existe el campo {1}.",
           return false;
       }

       if (fieldObj.ValueFieldFinal.Replace(".", "").Replace(",", "").Replace("(", "").Replace
       {
           msgError = String.Format("Problema de Integridad en {0}. No coinciden los valores e
           return false;
       }

       etat = 0;
       foreach (XAttribute at in campo.Attributes())
       {
           if (at.Name == "etat")
           {
               if (at.Value == "illisible")
               {
                   etat = 1;
               }
```

Función: El siguiente trozo de código que se muestra en la figura más abajo, se corresponde a un trozo de la función `LoadLotFromXML`, el objetivo de la misma es el de crear distintos tipos de clases con la información obtenida del documento XML. Dicho código utiliza para obtener los datos del XML, la herramienta LINQ to XML (mostrada en la siguiente figura bajo recuadros rojos).

```
documento = XDocument.Load(pathXML);
var items = from item in documento.Descendants("image")
            select item;
_ImageObjList.Clear();
countFields = 0;
foreach (XElement item in items)
{
    image = item.Attribute("nomimage").Value;
    idImage = Convert.ToInt32(image.Substring(0, image.Length - 4));

    var champs = from champ in item.Descendants("champ")
                 select champ;

    countFields = countFields + champs.Count();
    imgObj = new OM.Image();
    imgObj.IdLot = idLot;
    imgObj.IdFolder = idFolder;
    imgObj.IdImage = idImage;
    imgObj.Name = image;
    imgObj.CountFields = champs.Count();
    imgObj.FieldObjList.Clear();

    _ImageObjList.Add(imgObj);
    foreach (XElement campo in champs)
    {
        fieldObj = new Field();
        fieldObj.Name = campo.Attribute("id").Value;
```

Nombre Proyecto: Framework de Active Software (AF)

Funcionalidad: Ver anexo que explica más en detalle las características de AF

Función: El siguiente trozo de código que se muestra en la figura más abajo, se corresponde a un trozo de la función **DoList**, cuyo objetivo es el de obtener determinados elementos de una colección de datos genéricos, esos datos obtenidos cumplen con ciertas condiciones impuestas por la función lambda pasada como parámetro. A continuación se marca en la figura a través de recuadros rojos en que momentos se utiliza LINQ o conocimientos relacionados al mismo.

Algunas cosas que se pueden rescatar del siguiente código son:

- La utilización de funciones lambda, pasadas como parámetro a un *Where*, lo cual funciona como filtro para una determinada colección de objetos.

- La utilización de la sentencias *orderby* de LINQ, que permite ordenar una colección de objetos en 3 líneas de código.

```
internal void DoList(Func<T, bool> filt, ........)        {
 ......
 ......
if (_Filt == null)
{
    result = listComplete;
}
else
{
    result = listComplete.Where(_Filt).ToList<T>();
}
if (selectedText != "")
{
    if (exact)
    {
        result = result.Where(k => ((IKeyword)k).TextKeyword.ToUpper() == selectedText.ToUpper()).ToList<T>();
    }
    else
    {
        result = result.Where(k => ((IKeyword)k).TextKeyword.ToUpper().Contains(selectedText.ToUpper())).ToLis
    }
}
if (result.Count < 1)
{
    return;
}
if (Page == 0)
{
    result = (from e in result
              orderby ((IKeyword)e).TextKeyword
              select e).ToList<T>();
}
else
{
  .....
        result = (from e in result
                  orderby ((IKeyword)e).TextKeyword.Split('-')[0], ((IKeyword)e).TextKeyword.Split('-')[1]
                  select e).ToList<T>();
```

Nombre Proyecto: Cotton

Funcionalidad: Desarrollo de software de administración para una empresa algodonera.

Función: El siguiente trozo de código que se muestra en la figura más abajo, se corresponde la definición de una clase llamada *FicheDecadaire*, donde se puede observar que algunos de sus propiedades utiliza LINQ to OBJECT para devolver información requerida

En el código se puede observar que se utiliza LINQ, empleando la sintaxis *Sum*, que permite sumar la propiedad de los objetos perteneciente a una determinada colección de objetos.
Además se utiliza *Where*, el cual permite obtener un subconjunto de objetos de una colección, que cumplen con una determinada condición.

```
public class FicheDecadaire : FicheDecadaireBase
    {
         .....
        public void SetSupTotaleSeme()
        {
            if ((_FicheDecadaireDecadesSemisObjList == null) ||
            (_FicheDecadaireDecadesSemisObjList.Count() == 0 ))
            {
                _SupTotalSemee = 0;
            }
            else
            {
                _SupTotalSemee = _FicheDecadaireDecadesSemisObjList.Sum(p => p.SupSemee);
            }
           .....
        }
        public void SetPourcSemeAvantDate()
        {
           ......
          if ((_FicheDecadaireDecadesSemisObjList != null) )
          {
              lista = (List<FicheDecadaireDecadesSemis>)(_FicheDecadaireDecadesSemisObjList
              .Where(d => d.DecadeSemisObj.DateDebut < DateTime.Parse(CTE_JOUR_LIMITE
                  + "/" + CTE_MOIS_LIMTE + "/" + _CampagneObj.Fin.Year)).ToList());
              ......
          }
          else
          {
               PourcSemeAvantDate = 0;
          }
         }
```

Función: El siguiente trozo de código que se muestra en la figura más abajo, se corresponde a un trozo de una función encargada de obtener los centros (lugar donde se procesa el algodón) que se encuentra dentro de una región (por las que se encuentra dividido el país). En esta función también se emplea LINQ, para chequear que la lista solución no contenga dos elementos iguales.

En el código se puede observar que se utiliza LINQ, empleando la sintaxis *Count*, que permite contabilizar cuantos elementos de una determinada colección cumple con una determinada condición.

```
public bool GetRegionCentre(int idRegion, out int[] idsCentre, out int i)
{
    ........
        .......
                dt = LoadTableSelectBuild(0, listSelect, null, null, listNum, null, null, null, null);
                if (dt.Rows.Count != 0)
                {
                    foreach (DataRow items in dt.Rows)
                    {
                        if (listaCentres.Count(p => (p ==
    (Convert.ToInt32((items[this.IdCentreCC.NameTable_NameColumn]).ToString()))))) == 0)
                        {
                            listaCentres.Add(
    (Convert.ToInt32((items[this.IdCentreCC.NameTable_NameColumn]).ToString())));
                        }
                    }
                }
    ........
}
```

5. Aplicación Desarrollada

5.1. Introducción

En el anterior capitulo se analizo en profundidad a LINQ, pero una de sus aplicaciones LINQ to ENTITY se observa que ofrece un gran conjunto de características que permiten mapear las tablas de la base de datos a objetos, para la mejor utilización por parte del desarrollador en el manejo de base de datos.

En este capítulo la idea es darle gran importancia a la utilización de esta herramienta (LINQ to ENTITY), para poder analizar qué ventajas y desventajas ofrece en el desarrollo de una aplicación real. Además es importante ver que errores, si es que existen, con la utilización del mismo.

La aplicación que se desarrollo se llama Active.Management.HoursLinq, trata sobre el manejo y contabilización de las horas que un empleado realiza en la empresa en la cual trabaja.

El nombre del producto proviene de una aplicación ya desarrollada por la empresa AS, cuyas funcionalidades son iguales a la que ofrece esta (basadas ambas en la misma interfaz grafica).

Entonces ¿porque realizar una aplicación que ya está hecha y funciona?, la respuesta es que el cometido principal de esta aplicación no es el de resolver el problema planteado, sino:

- Ver cuán bien funciona como implementación de una capa de datos basada en una arquitectura en tres capas la utilización de LINQ to ENTITY.
- Comparar a LINQ to ENTITY con la herramienta realizada por la empresa AS como parte de su Framework para el manejo de los datos provenientes de las tablas de la base de datos, esta comparación está orientada a facilidades de utilización y características, que el diseño e implementación de las mismas ofrece.

A continuación se va a realizar un análisis en busca de funcionalidades que debe de tener la aplicación, y restricciones que debe de esta cumplir. Más adelante se indicará mediante diagramas y explicaciones, la arquitectura que se eligió para implementar la aplicación. Luego se van a indicar los detalles de implementación que se tuvieron en cuenta para la realización de la misma.

Y por último se va a mostrar el análisis comparativo entre los Framework (AF y LINQ to ENTITY), mostrando características, ventajas y desventajas de uno y otro. Así como también se indicaran los errores que se encontraron al trabajar con LINQ to ENTITY.

Además otro tema que se va a desarrollar es el de explicar la diferencia, si es que existe, entre LINQ to ENTITY y NHibernate (ORM basado en Hibernate de Java).

5.2. Análisis

La idea es desarrollar una aplicación basada en el paradigma de orientación a objetos. La misma como ya se menciono en la introducción va a manejar cuanto tiempo de trabajo realizó un determinado empleado en la empresa. Pero resta completar esta descripción diciendo que las horas que va a ingresar cada empleado quedan clasificadas según la tarea que esté realizando dentro de un determinado proyecto.

Cada proyecto a realizar se divide en tareas en las cuales se debe de indicar a los empleados a los que tiene asignado para realizar la misma, así como también las cantidad de horas que tiene asignado para su realización, como su fecha de comienzo y finalización. Además en la definición del proyecto se debe de saber cuál fue su fecha de inicio y de finalización, así como también el empleado que va a cumplir la función de jefe del proyecto.

A continuación se van a listar algunas de las restricciones que se pudieron extraer de la aplicación.

5.3. Restricciones

Las siguientes restricciones se obtienen del uso de los clientes de la aplicación ya realizada por la empresa AS.

- El empleado debería de tener la aplicación funcionando todo el tiempo, pero seria de utilidad tener alguna manera de acceder a ella solo cuando se la necesita.
- Se debe de poder acceder a la información de todos los empleados que tiene la empresa y de los que tuvo.
- Si un empleado llega a la empresa en un tiempo considerablemente después a la hora que tiene prevista que entre a trabajar, entonces se le enviara a los jefes que tiene asignado una notificación por mail.
- La aplicación debe ser fácil de instalar desde cualquier sitio y debe de poder actualizarse a nuevas versiones de forma automática sin necesidad de realizar una re-instalación.
- Los datos almacenados por la aplicación deben de ser almacenados en una base de datos Relacional.
- Los datos relevados de cada cliente debe de ser almacenados en una base de datos, para su posterior manipulación
- La interfaz grafica de la aplicación debe ser similar a la desarrollada por la empresa Active Software.
- Esta restricción se desprende de la anterior, la aplicación debe de ser desarrollada sobre .Net, más específicamente sobre el lenguaje C# (para ser compatible con la aplicación antes nombrada).

Las anteriores restricciones indicaban la usabilidad esperada por el usuario, pero resta indicar las restricciones de funcionalidad, las cuales se listan a continuación.

- No puede haber dos empleados con el mismo usuario.
- El empleado no puede ser jefe de sí mismo.
- La hora a la que debe de entrar el empleado a trabajar debe ser menor a la que debe de salir.
- La fecha en que un empleado ingreso a la empresa debe ser menor a la fecha de egreso de la misma.
- No puede haber dos tareas con el mismo nombre.
- Para una misma tarea no se puede asignar dos veces a la misma persona.
- Las horas que se le asignan a cada tarea debe de ser menor a las del proyecto a las que están asociadas.
- No se puede tener un proyecto en estado de finalización sin antes haber indicado cual es la fecha de finalización del mismo.
- La fecha de finalización debe ser mayor a la fecha de inicio del proyecto.
- No puede haber dos proyectos con el mismo nombre
- Las horas trabajadas por un empelado nunca puede ser un valor negativo
- La fecha y hora a la que un empleado comenzó a realizar una tarea debe de ser menor a la fecha y hora en que finalizo de realizar la misma.
- Se debe de poder ver la lista de empleados, lista de tareas, lista de tipos de tareas, lista de proyectos y las horas realizadas por un empleado en la empresa.

5.4. Requisitos Técnicos

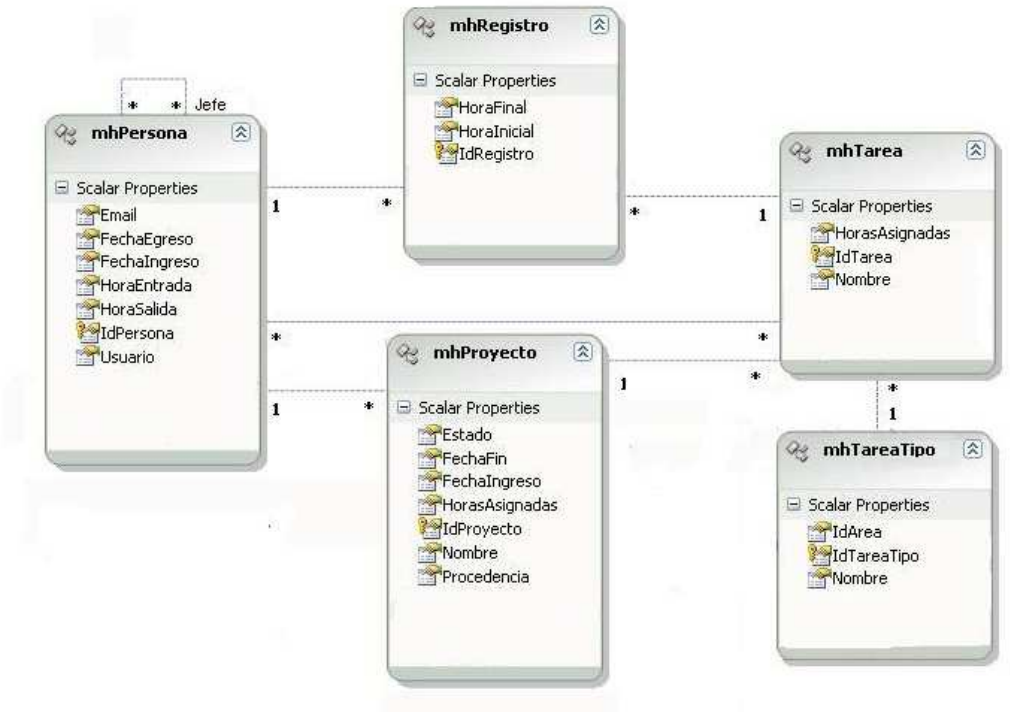
Las restricciones técnicas nos indican que tecnología se debe de tener instalada en la maquina que vaya a utilizar la aplicación.

- La aplicación debe de poder ejecutarse sobre el sistema operativo Windows XP(o superior).
- Como se dijo en las restricciones se va a almacenar la información en base de datos Relacional, en este caso se eligió a Microsoft SQL Server 2005.
- La aplicación tiene como requisito que debe de estar implementada sobre el Framework .Net 3.0

5.5. Descripción

En esta sección se va a indicar cuáles son las funcionalidades que debe de tener la aplicación así como también se muestran diagramas y descripciones de las clases obtenidas en este análisis.

Basados en la realidad planteada se muestra el siguiente diagrama que representa el modelo de dominio de la aplicación.



A partir del modelo de dominio se definen las siguientes clases, de las cuales se va a indicar que operaciones (funcionalidades, responsabilidades) tiene asignado:

Persona:

Esta clase representa a los Empleados en la empresa, para la misma se puede encontrar operaciones que permitan dar de alta y permitir modificar los diferentes datos que describen al empleado (en las cuales se debe de indicar el horario que realiza en la empresa), cabe recordar que un empleado puede tener un jefe distinto en cualquier momento.

Tipo de Tarea:

Esta clase representa los distintos tipos de tarea que se manejan, a partir de los cuales se van a clasificar las tareas.

Proyecto:

Esta clase representa todos los proyectos de la empresa, los mismos pueden haber comenzado, haberse suspendido o haber finalizado.
Los proyectos se dan de alta especificando la fecha de inicio y fin del mismo, así como también la cantidad de horas/hombre que tiene asignadas.

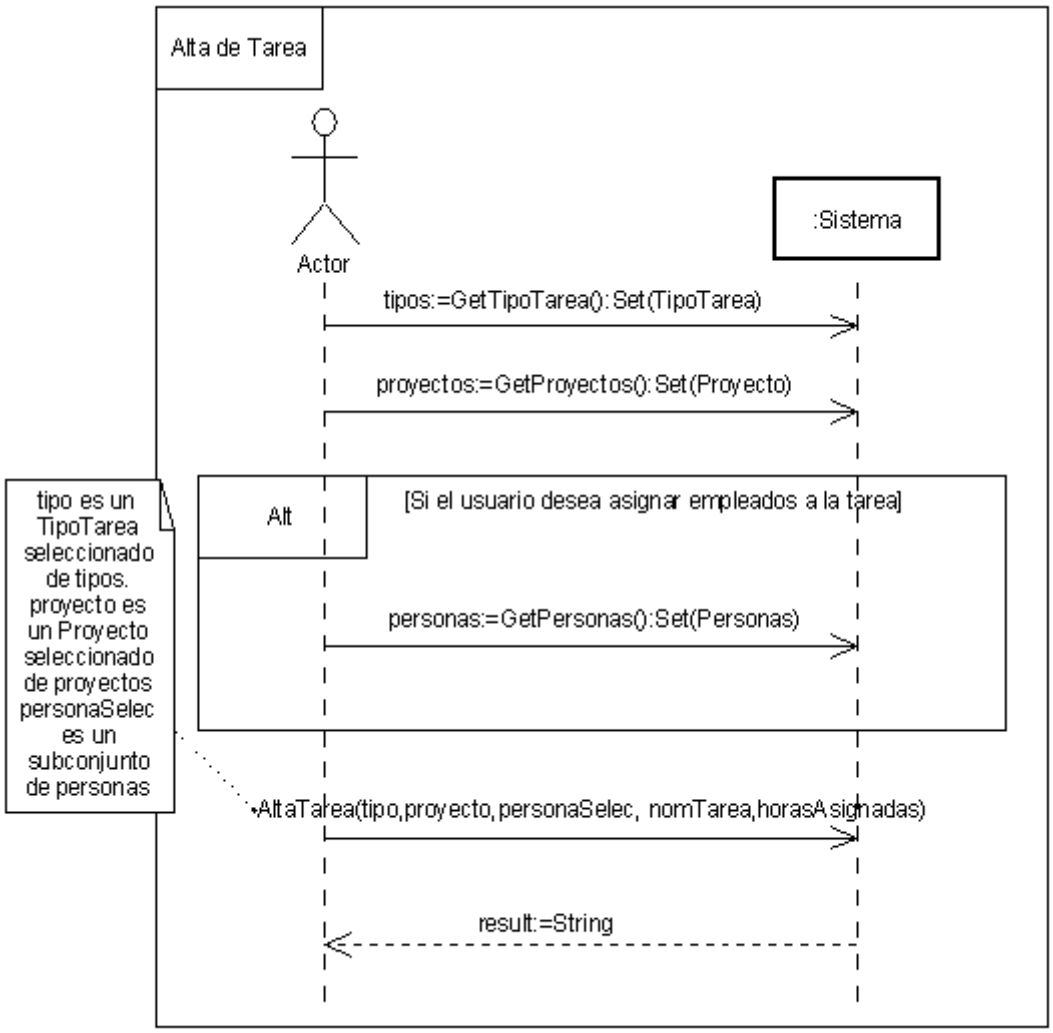
Tarea:

Esta clase representa a las tareas obtenidas del análisis del proyecto al cual la misma pertenece, cuando se da de alta la misma especifica que empleados deben de estar asignados, y dentro de qué tipo de clasificación de tareas se encuentra (recordar que las tareas se clasifican por el tipo de tarea).

Registro:

Esta clase es la principal por que se encarga de registrar las horas que un empleado trabajó sobre una determinada tarea (perteneciente a un proyecto), dentro de la empresa.

A continuación se va a definir uno de los flujos de la operación de dar de ***alta una tarea***, dicho flujo se muestra a través del siguiente diagrama de secuencia.



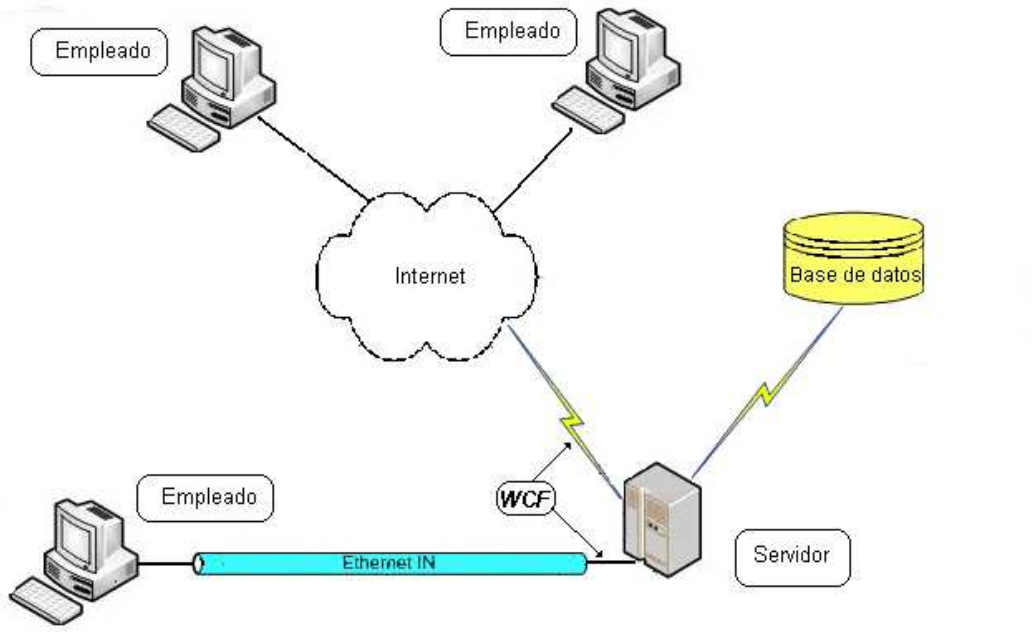
5.6. Diseño

Observando los distintos requerimientos propuestos y conociendo el funcionamiento del software en sí, entonces se puede afirmar que, para que la aplicación tenga una buena aceptabilidad e incorporación por parte de los empleados la misma debe de ser de tipo Smart Client, (para más información sobre la descripción de esta herramienta se puede encontrar en los anexos) para poder tener una aplicación cuya instalación sea sencilla, con el aspecto de una de escritorio, para realizarla se utilizó la tecnología Click Once de Microsoft la cual permite (para más información sobre la descripción de esta herramienta se puede encontrar en los anexos) realizar este tipo de aplicaciones clasificadas como Smart Client.

La arquitectura está basada en el diseño arquitectónico llamado de “tres capas” (la descripción de la misma se encuentra desarrollada en los anexos), el cual divide la aplicación en tres capas bien distintas; la de datos, lógica y presentación (también llamada vista). Para poder acceder a la capa de datos se utilizo una de tantas tecnologías que permiten la interconexión entre sistemas, la misma se llama WCF (Windows Comunication Foundation).

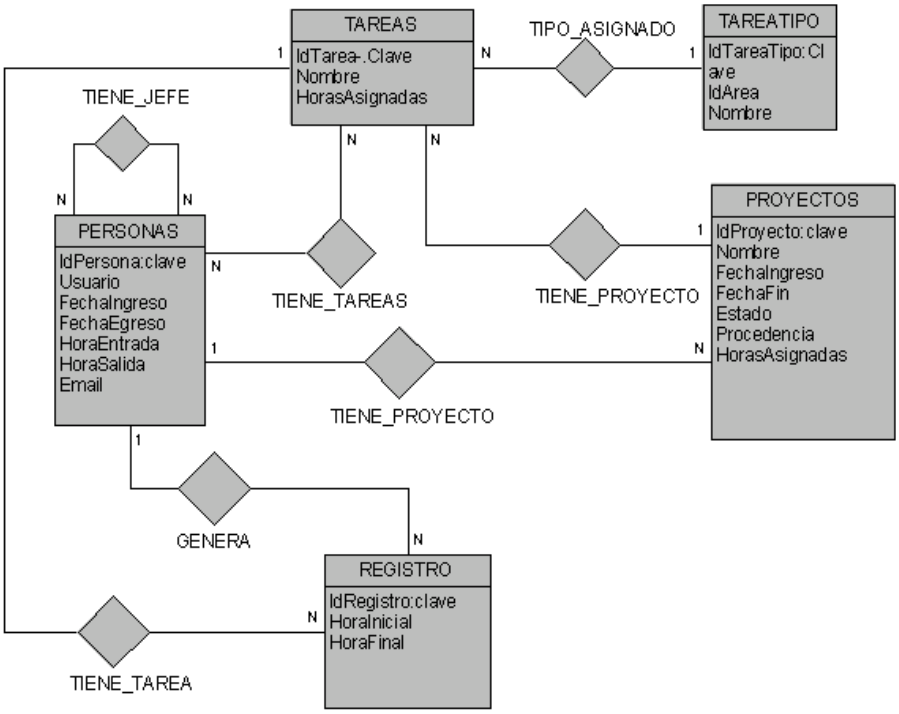
5.6.1. Arquitectura

A continuación se muestra un diagrama global que indica la arquitectura en grandes rasgos que se va a manejar para implementar la aplicación requerida.

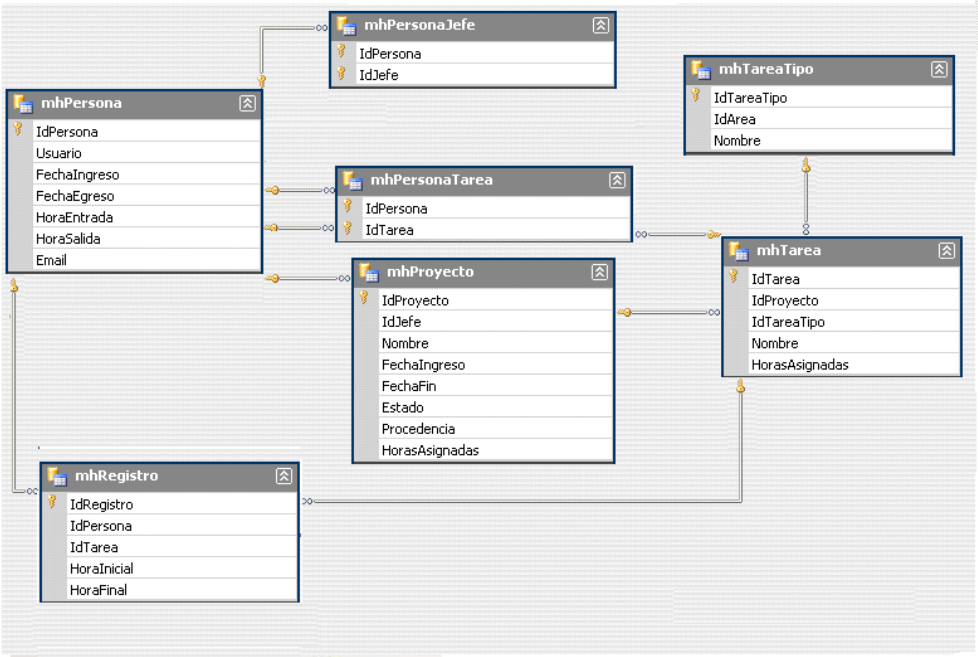


5.6.2. Base de Datos

A continuación se muestra un diagrama que representa el MER de la base de datos que se va a adoptar, sobre la cual se van a almacenar la información referida a la aplicación a desarrollar.

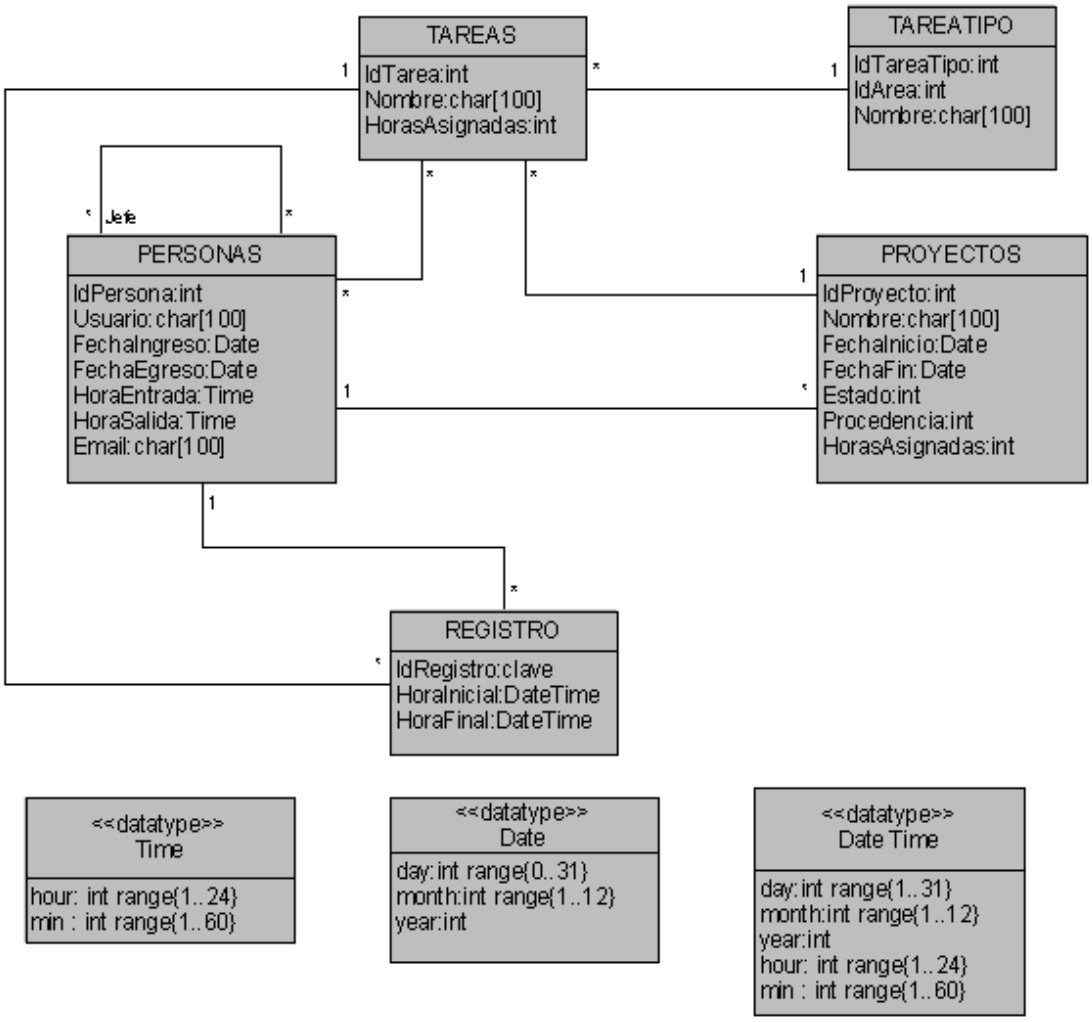


Luego se realizó el pasaje del MER a la base de datos Relacional, dando como resultado a las siguientes tablas que se muestran en el siguiente diagrama.



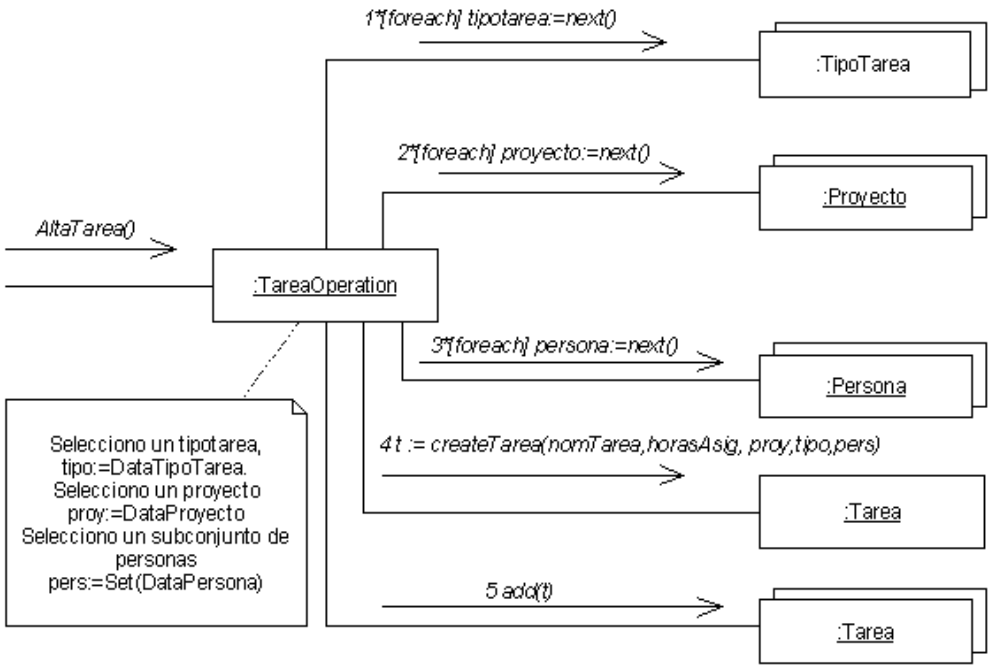
5.6.3. Diseño de Clases

El siguiente diagrama muestra el diseño de clases que se siguió (como se puede observar la idea es implementar una aplicación orientada a objetos).
Donde la mayoría de los tipos de datos usados son los comúnmente manejados int (enteros), char[](arreglo de caracteres), con excepción de Date, Date Time y Time (las cuales son descripciones de la fecha con diferentes agregados).



A continuación se muestra el diagrama de colaboración para la operación alta de una tarea.

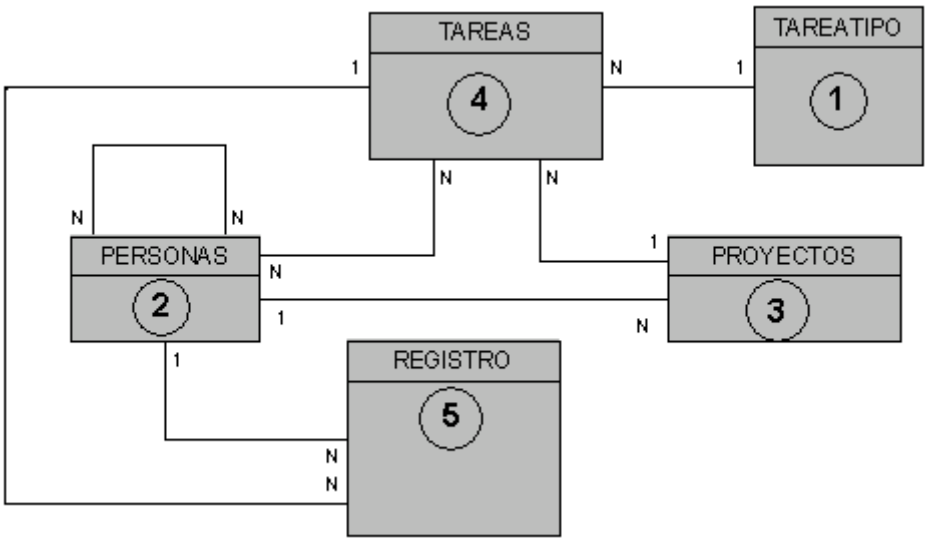
5.6.4. Diagrama de Colaboración



5.7. Implementación

Para el desarrollo de la aplicación, se comenzó primero con las clases a las que ninguna otra hace referencia hasta llegar a las más referenciadas.

A continuación se muestra en el siguiente diagrama el orden que se tomo para desarrollar las distintas clases.



5.7.1. Decisiones de Implementación Tomadas

A continuación se van a describir decisiones e imposiciones que se hicieron sobre el desarrollo:

- La aplicación se desarrollo basándose en una arquitectura de tres capas: capa de datos, capa de negocio y de presentación (en los anexos se describe con más profundidad, cuales son las características de una arquitectura en tres capas).
- El entorno grafico de la aplicación debe ser igual al de la aplicación de la empresa (AS), sabiendo esto se opto por utilizar el Framework de la misma para realizar la capa de presentación de la aplicación (Una descripción a grandes rasgos del Framework de AS se puede encontrar en los anexos presentados junto a este informe).
- El obtención de los datos de las tablas de la base de datos se realizo a través de LINQ to ENTITY (nombrado anteriormente), generando diversas entidades las cuales quedan almacenadas en un servidor central, y para tener acceso al mismo se utilizo la tecnología WCF, la cual permite poder llegar a los datos almacenados en el servidor desde cualquier punto (cumpliendo uno de las restricciones pedidas).
- Para cumplir los requerimientos, una de las restricciones pedidas que debe de cumplir la aplicación es que debe de informar a los jefes de los empleados que hayan llegado tarde vía e-mail, para resolver esto se utilizo

una librería del Framework de .Net llamada `System.Net.Mail.MailMessage,` la cual permite el envió de mail a través del protocolo SMTP, tomando los valores del emisor del mail(los datos son la dirección del emisor, el servidor SMTP de servidor de mail del emisor, el usuario y la contraseña). Como no hubo una indicación directa de cuánto tiempo puede llegar tarde el empleado, se consideró que ese tiempo sea igual a 30 minutos, otra cosa a considerar es que la clave del emisor de mail no esta encriptado (por no haber restricciones sobre ello).

- Se había indicado que se quería tener a la aplicación todo el tiempo ejecutándose pero al mismo tiempo es incomodo tenerla abierta y minimizarla también molestaría de igual forma al empleado. Para resolver esto se utilizo una librería del Framework de .Net llamada `System.Windows.Forms.NotifyIcon`, el cual permite agregar a la aplicación como icono en el área de notificación, asignándole un "Menú Contextual" y manejo de eventos, a través del icono se puede suspender la aplicación haciendo que la misma desaparezca de la vista de la persona, aunque la misma continua ejecutándose, y luego cuando se desea utilizar se selecciona el icono indicando que se quiere abrir la aplicación.
- Para almacenar en la base de datos una determinada hora, por ejemplo, la hora de ingreso se guarda con la fecha mínima, esa fecha mínima se considera "14/05/1989".
- Para actualizar a las personas que están asignadas a una determinada tarea, lo que se hace es eliminar a las personas que ya no se encuentran asignadas y luego, agrega a las nuevas (todo esto se realiza en el servidor).
- La implementación de la capa lógica está muy ligado al Framework de AS, por que al ser gráficamente compatible con el mismo, hay que implementar ciertas interfaces que hacen que la implementación de esta capa se encuentre orientada a la metodología impuesta por el mismo.
- Como ya se nombro anteriormente se quería una aplicación que fuera Smart Client entonces para cumplir con este requisito se utilizo la tecnología Click Once la cual viene incluida en el editor Visual Studio 2008(sobre la cual se desarrollo la aplicación).

5.8. Errores Remanentes

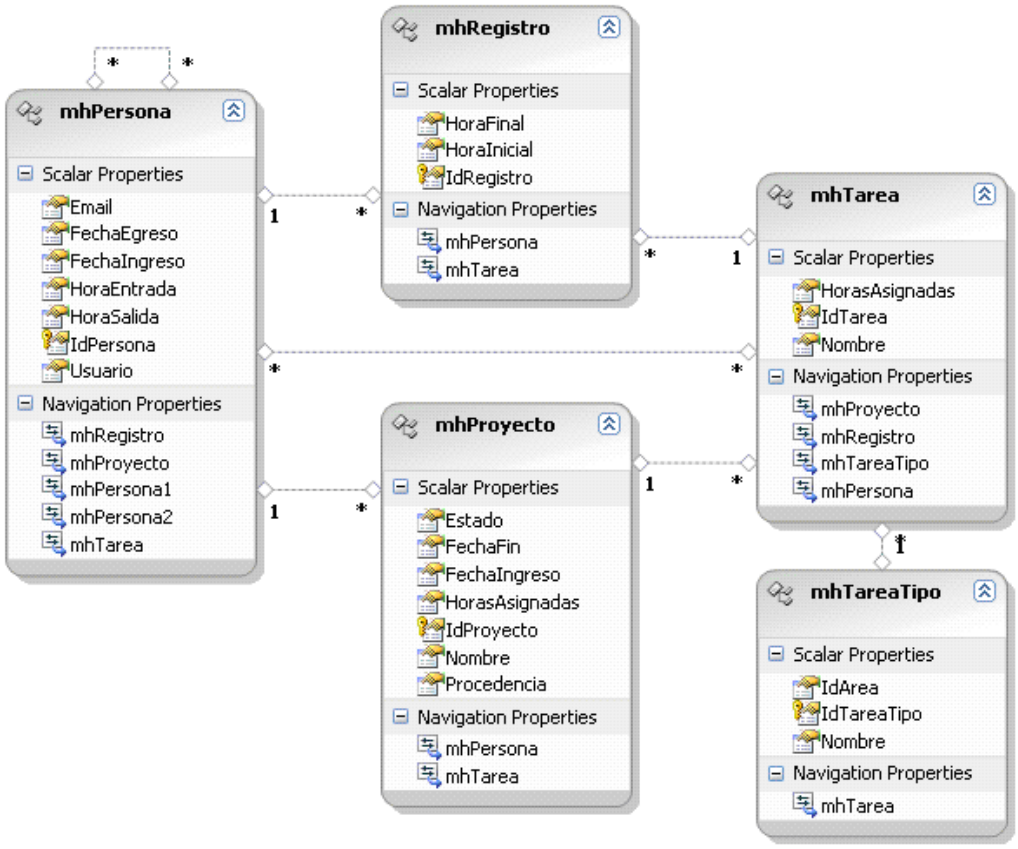
Los errores que se encontraron son los siguientes:

- En la pantalla persona la hora de entrada y salida, cuando se presiona limpiar debe de poner la hora en "00:00:00" en cambio muestra la hora vieja (la hora de entrada, que por ejemplo se uso para crear una persona anteriormente).
- En la misma pantalla sucede lo mismo con la hora de salida que con la de entrada.
- En la pantalla anterior también ocurre otro inconveniente y es que la fecha de ingreso muestra la fecha de hoy, pero cuando se desea crear una nueva persona, tira un error de que no se ha seleccionado una fecha de ingreso.

5.9. Conclusión

A continuación se van a indicar distintas características que poseen tanto el AF y LINQ to ENTITY, mostrando las diferencia entre ambos.
A continuación se listaran las características que ofrece la herramienta LINQ to ENTITY, para luego en base a estas realizar una comparación con AF:

- A partir de las tablas de las base de datos, se genera el modelo de objeto automáticamente, con las operaciones de creación, modificación y consulta. O sea se pasa del diseño de la base de datos (el diagrama de base de datos se muestra en la sección de diseño) al siguiente modelo de clases



Lo cual permite aislarse totalmente de la capa física (que manejador de base de datos se está manejando), permitiendo en todo momento trabajar con clases.

- Permite realizar el mapeo a objeto de tablas de bases de datos, cuyos manejadores pueden ser:
  - SQL Server
  - Oracle
  - Informix Data Provider

  - DB2
  - My SQL
  - PostgreSql

- Utilizando LINQ to ENTITY se trabaja con objetos, sin importar a que base de datos esta referenciado, y más aún, sin importar a que manejador de base de datos se está accediendo (ORACLE, PostgreSQL, SQLServer, etc.).

- Ofrece la posibilidad de realizar consultas a través de una sintaxis común sin importar el manejador de base de datos que se está utilizando, dicha sintaxis además:
  - Se encuentra integrada al lenguaje C# (también en VB).
  - Las consultas con esa sintaxis se verifican en tiempo de compilación.

- Las clases que se generan (también llamadas Entity), poseen un cabezal de contrato para WCF ([DataContract]), esto permite que se pueda, manejar con WCF sin hacer cambios en las Entity y sin tener que realizar clases Proxy (clases que contengan a las Entity) que se encarguen de este problema.

- Una característica que hay que tener en cuenta cuando se va a guardar una instancia de una de estas entidades, es el traer a memoria las instancias a la que hace referencia directamente, por ejemplo en el caso de esta aplicación si yo quisiera guardar una tarea, tendría que obtener primero el tipo de tarea, luego el proyecto y después las personas que están asignadas a la tarea para luego recién poder guardar la misma.

- Para mostrar la lista de las tareas vigentes se utilizan diferentes filtros, que permiten obtener información sobre tareas que cumplen tales o cuales características, esto se logra usando una clase como filtro, a la cual se le van dando valores por los cuales se tiene que realizar la búsqueda para obtener la lista de tareas deseada.

- LINQ to ENTITY permite poder cargar las entidades relacionadas a una Entity de dos formas, una diferida y otra directa, la directa se logra aplicando sobre la referencia de la entidad la operación Load (), y la diferida se aplica en la propia consulta de LINQ la operación Include (“nombre del tipo de la propiedad”). (Más información sobre la carga directa o diferida de LINQ to ENTITY ver referencia [1]).

En base a las características antes nombradas, se listaran las de AF, y se compararan con las de LINQ to ENTITY:

- El ORM, no realiza un modelo automático de las tablas de la base a las clases y además las clases que se definen no son las que uno esperaría (viendo el diagrama clases realizado), porque lo que realiza este software es, pasar cada tabla a una clase, de tal manera que las columnas de las tablas son las propiedades de la clase. Esto es una opción aceptable

de mapeo, pero rompe en algunos casos el diseño de clases que se pretende.

- Permite realizar el mapeo a objeto de tabla de diferentes manejadores de base de datos, como los que se listan a continuación:
  - SQL Server
  - Oracle

  Vemos que la lista es inferior a la que soporta LINQ to ENTITY.

- Permite al igual que LINQ to ENTITY olvidarse del manejador de base de datos con el cual se está manejando, pero la consulta en si resulta un poco más complicada porque se manejan transacciones, se dice que parámetros tiene y que no, en cambio con LINQ to ENTITY se realiza la consulta directamente sobre las Entidades
- Ofrece la posibilidad de realizar consultas a través de una sintaxis común sin importar el manejador de base de datos que se está utilizando, pero a diferencia de LINQ to ENTITY, posee la contra que la sintaxis sigue siendo sobre un *string*, lo cual provoca que no se puede verificar si la sintaxis usada es correcta en tiempo de compilación.

- En el caso del Framework de AS, como ya se nombró anteriormente, se deben de definir dos clases una que define la tabla en sí y la otra que indica los contratos con WCF ([DataContract]) que va a tener la nueva clase creada.

- Una característica similar a LINQ to ENTITY es, cuando se quiere guardar una instancia de una entidad hay que traer a memoria las instancias a la que hace referencia directamente, por ejemplo en el caso de esta aplicación si yo quisiera guardar una tarea, tendría que obtener primero el tipo de tarea, luego el proyecto y después las personas que tiene asignada la tarea para luego recién poder guardar la misma. Para realizar esto cuando se define la clase se utiliza una funcionalidad del Framework que permite, al definir las referencias (que son clases) que se desean cargar junto con la clase definida. Esta funcionalidad se logra agregando estas referencias como SubObject de la clase definida. Esta funcionalidad no la tiene LINQ to ENTITY, este necesita utilizar las operaciones ***Include*** o ***Load*** para lograr lo mismo.

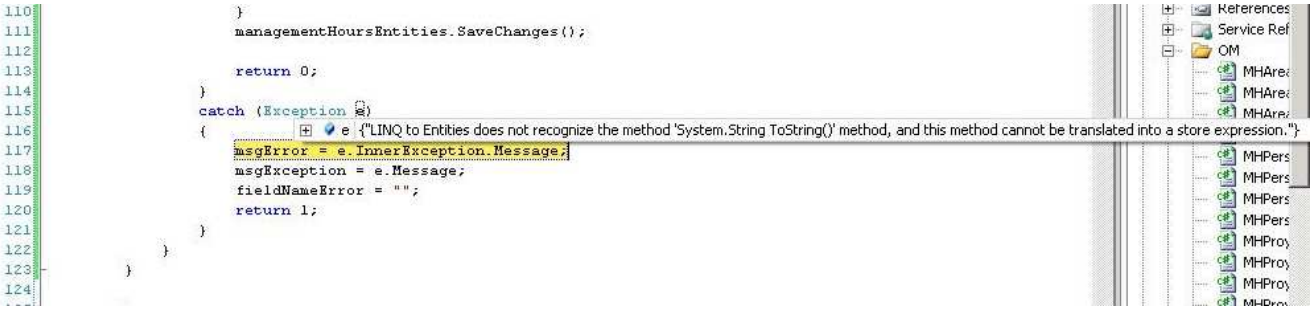
- Para realizar el filtro, por ejemplo de mostrar una lista de tareas vigentes, se realiza algo más “manual”, se utiliza una lista de cada tipo de las propiedades por ejemplo, una lista de enteros, una lista de fechas, etc, y en base a estos datos realiza un filtro de las tareas en la base de datos.

5.9.1.1. Dificultades de LINQ to ENTITY

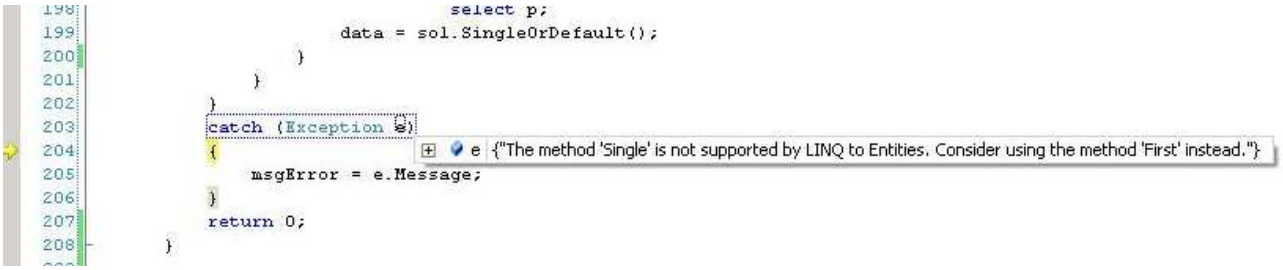
El hecho de manejar una versión de una herramienta que no es la final, produce que el desarrollar con ella surjan determinados errores, produciendo así que el desarrollo de un producto no sea de la mejor manera.

A continuación se listan algunos de los errores y dificultades encontradas al manejarlo:

- La definición de un Entity Model de forma manual, no es muy sencilla y para nada intuitiva.
- Cuando LINQ to ENTITY (más específicamente Entity Framework) es usado en un proyecto (esto hace referencia a proyectos generados con Visual Studio), accedido por muchos desarrolladores, ocurren varios conflictos, como por ejemplo cuando dos desarrolladores hacen cambios a un diagrama de entidad, el control de cambios se realiza con errores. El problema es causado por que el formato usado para guardar los modelos de entidad mantiene tanto elementos de visualización, como del modelo mismo.
- Tiene algún problema que no sucede con LINQ to SQL, si se quiere aplicar la operación *toString ()* a un atributo de un modelo de entidad cuyo tipo es un entero (por ejemplo), salta un mensaje de error (en tiempo de ejecución) lo cual puede confundir al programador pensando que es un error de él siendo error de LINQ to ENTITY.



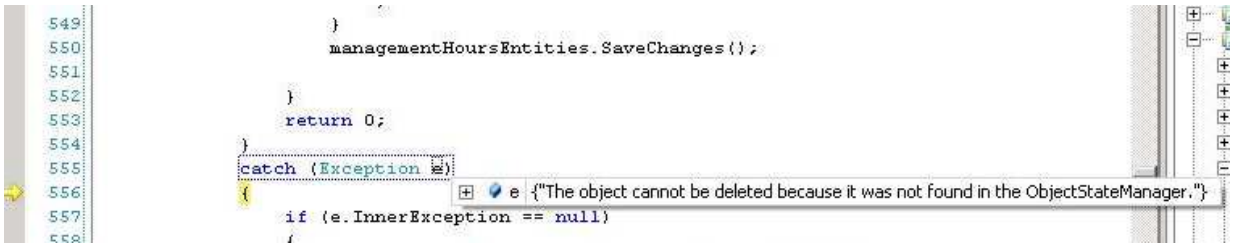
- Además de la utilización de la operación *toString()* LINQ to ENTITY tiene problemas con la operación *SingleorDefault()* o *Single()*, para las cuales se avisa con un mensaje acorde, sugiriendo utilizar la operación *First()* para sustituirlas.



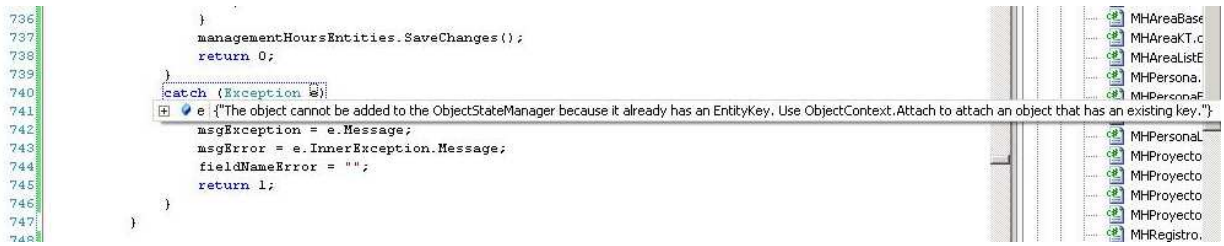
- También aparecen algunos problemas al usarlo junto a WCF, por ejemplo si manda una "función lambda" como parámetro de un procedimiento ofrecido por WCF y luego se utiliza esa función en una consulta LINQ da error de serialización. Otro error es cuando se manda por WCF una Entity para ser

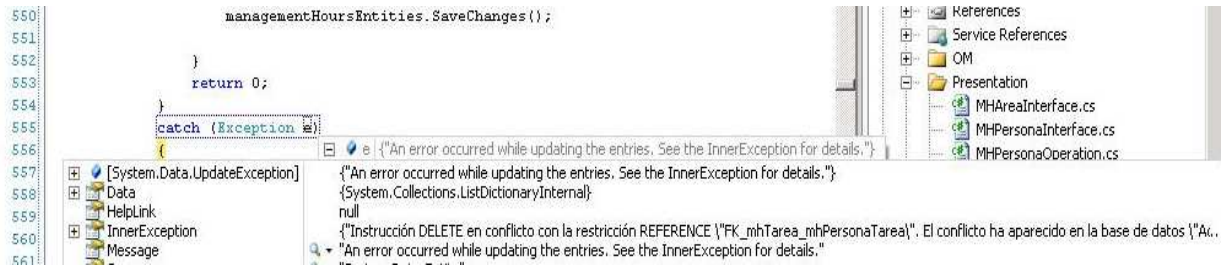
persistida, en esas circunstancias generalmente el EntityKey de la entidad queda vacío provocando que se produzcan errores de compilación relacionados a esto, para solucionarlo hay que crear una entidad nueva y copiarle los valores de la primera.

- Cuando se va a guardar una entidad, y esta hace referencia a otra que tiene un DateTime como campo, sino se la carga en memoria, puede tirar errores referidos al límite del DateTime permitidos por la base, en vez de avisar que no se tiene el objeto directamente referenciado cargado en memoria.
- En el caso que se quiera guardar una entidad que se pasa como parámetro de una operación, se produce un error, al igual que si se quiere eliminar la misma. El error está relacionado en el caso de la eliminación con que la clave de la entidad no existe.



- En el caso que el modelo que ofrece LINQ to ENTITY, los objetos que participan se encuentren en una relación de muchos a muchos se debe de tener cuidado al modificar uno de estos objetos o eliminar, porque tiene que cargar en memoria al objeto al cual está haciendo referencia o si no lo errores que pueden ocurrir son los siguientes, pudiendo ser confusos.





- No actualiza bien el modelo, da errores, por ejemplo si agrego una columna nueva a una tabla o cambio de tipo de la columna, los cambios no se actualizan automáticamente en el modelo.

5.9.1.2. Usar LINQ to SQL o LINQ to ENTITY

LINQ to SQL es una forma simple de mapear datos, aunque para muchas aplicaciones se necesita algo más sofisticado como lo es LINQ to ENTITY (que mapea los datos de forma más compleja).

Para poder decidir en qué circunstancias conviene quedarse con uno u otro (siempre basados en la hipótesis que se está utilizando el manejador de base de datos, SQLServer), a continuación se muestra la siguiente tabla la cual compara ambas tecnologías.

LINQ to SQL	LINQ to ENTITY
Es solo válida para SQL Server.	Es válida para distintos sistemas gestores de base de datos.
Usa una solución ORM donde la base de datos mantiene una relación 1:1 con el objeto de modelo	Usa una solución ORM donde las clases cumplen una relación 1:1 con la base de datos, pudiendo tener una estructura muy diferente al del esquema de base de datos.
Usa una solución ORM con herencia que son almacenadas en una tabla simple.	Usa una solución ORM con herencia jerárquica que pueden tener esquemas de almacenaje alternativo (una tabla simple para la jerarquía, una tabla simple para cada clase, y una tabla para todos los datos relacionados con el tipo específico).
Permite la utilización de *Store Procedures*	Permite la utilización de *Store Procedures*

5.9.1.3. LINQ to ENTITY vs NHibernate

A continuación se va a realizar una comparación entre ambas herramientas, visto que al parecer realizan exactamente lo mismo, pero poseen sus diferencias.

NHibernate es un Framework de ORM, producto de la conversión de Hibernate de Java a C#, para una integración en la plataforma .NET, conteniendo la funcionalidad de mapear los objetos de una aplicación .Net a una base de datos Relacional, las clases mapeadas son llamadas entidades o clases persistentes. El mapeo es realizado a través de archivos XML, donde se especifica de que manera se mapean las columnas de las tablas con las propiedades de las clases. Permitiendo características que ofrece LINQ to ENTITY como tener la posibilidad de olvidarse de escribir sentencias SQL en el código.

En performance son similares, en la realización de consultas a la base LINQ to ENTITY es mejor que NHibernate, pero cuando se requiere actualizar un registro en NHibernate es más rápido.

Tal vez las desventajas que podrá tener NHibernate no vienen por el manejo del mismo, sino por la configuración que este requiere, y la posibilidad que ofrece LINQ to

ENTITIY de poseer un entorno grafico integrado a Visual Studio 2008 que permite una mejor actualización del modelo de entidad. Y en el caso de LINQ to ENTITY lo que se podrá afirmar es que, el mismo no es un producto tan maduro como lo es NHibernate, además este ofrece un conjunto mayor de características.

Pero la diferencia entre ambas esta en el modelo de datos de entidad. LINQ to ENTITY fue estructurado específicamente para separar el proceso de consultas, de la una forma en cómo se realiza la construcción de los objetos y el seguimiento de los cambios que se realizados. La idea de esta herramienta es que en base a esta se puedan generar un conjunto se servicios a ser utilizados.

Igual existen muchos usuarios que se encuentran de acuerdo con uno y en desacuerdo con el otro. En el caso de los seguidores de NHibernate, han realizado una página para sacarle el apoyo a LINQ to ENTITY, indicando las razones por las cuales no se debe de usar (la pagina que se encuentra en las referencias).

A continuación se muestra una tabla comparativa entre NHibernate, LINQ to SQL y LINQ to ENTITY

	NHibernate	**Linq to SQL**	**Linq to Entity**
Mapping con Atributos	Si	Si	Si
Mapping con XML	Si	Si	Si
Persistencia Transparente	Si	Si	Si
Estrategias de Fetching	Si	Si	Si
Estrategias de Herencia	Todas las estrategias	Solo las tablas con Jerarquía	Todas las estrategías
Mapping Entity sobre más tablas	No	No	Si
Mapping de Value Type	Si	No	Si
Lenguaje de Interrogación a Objetos	HQL (stringas) y	Linq aplicado a DataContext	Linq to Entities (Aplicado a ObjectContext)
Lazy load	Si	Si	Si
Mapping de llaves compuestas	Si	Si	Si
Identity Map	Si	Si	Si

6. Conclusiones y Trabajo Futuro

6.1. Conclusiones y Evaluación

A continuación se van a mostrar las conclusiones obtenidas de los distintos resultados esperados, para luego basados en los mismos obtener una conclusión final global del proyecto.

6.1.1. Conclusión del Análisis

Luego de obtenidas las conclusiones del análisis hecho sobre LINQ, cuyo objetivo era medir la utilidad de LINQ en sus diferentes implementaciones, se indicaran que fue lo que se concluyo del uso de LINQ por parte de la empresa Active Software.

A continuación se muestra la conclusión del análisis realizado, divida por las obtenidas por sus diferentes implementaciones.

6.1.1.1. LINQ to OBJECT

Las conclusiones que se obtuvieron como resultado de este estudio son las siguientes:

- Cuando se tiene que realizar una consulta a una colección de objetos y esta consulta es compleja (con compleja hace referencia a que para obtener el resultado de la misma se necesita iterar sobre muchas listas involucradas entre sí), es recomendable usar LINQ. Esto es válido aún en el caso en que el tiempo de ejecución sea mayor, ya que la calidad del código es muy superior.
- En el caso de que se desee ordenar el resultado de una consulta que se aplica a una colección de objetos, se recomienda utilizar LINQ, tanto por la importante reducción de la complejidad que permite obtener.
- Para realizar consultas simples sobre una colección de objetos es aconsejable utilizar el manejo habitual de colecciones de objetos, visto que se las puede resolver sin ningún inconveniente, obteniendo buenos tiempos de ejecución y prolijidad en el código.

Resumiendo se puede afirmar que se puede utilizar LINQ para resolver distintos problemas relacionados con colecciones de objetos, visto que la prolijidad en el código y la disminución de esfuerzo es alta, aunque para las cosas simples es recomendable seguir con el manejo “tradicional” que se realiza sobre una colección de objetos.

6.1.1.2. LINQ to XML

Las conclusiones que se obtuvieron como resultado de este estudio son las siguientes:

- LINQ permite realizar cualquier tipo de consultas y actualizaciones sobre archivos XML de forma muy eficiente.
- El control que tiene LINQ sobre nombres, espacios de nombres (NameSpace) y prefijos de espacios de nombre, es sencillo. En contraposición al manejo que ofrecen las tecnologías tradicionales.

- Un problema que se suele tener al utilizar DOM, es la actualización del nombre de un nodo dentro de un archivo XML. Para realizar esto se debe primero crear un nuevo nodo, y luego copiar el contenido del nodo viejo al nuevo. Utilizando LINQ este problema se resuelve de forma sencilla.
- Posee la capacidad de validar el contenido de un archivo XML, verificando que el contenido del mismo sea compatible a un esquema determinado.

En resumen LINQ soluciona muchos aspectos que antes no se resolvían o se resolvían pero no de la forma más apropiada. Por las razones anteriormente nombradas se puede afirmar que es útil usar LINQ to XML, como herramienta para la manipulación de documentos XML, pero hay que tener en cuenta que muchas veces puede resultar más eficiente utilizar otra herramienta (como en el caso de XmlReader).

6.1.1.3. LINQ to SQL

Las conclusiones que se obtuvieron como resultado de este estudio son las siguientes:

- La cantidad de líneas de código necesarias para la realización de una aplicación usando esta herramienta es mucho menor a implementar la misma pero utilizando sentencias SQL.
- El usarlo le otorga al programador la facilidad de poder chequear en tiempo de ejecución la validez de sus consultas, lo cual no posible utilizando directamente SQL.
- Es un ORM, lo cual quiere decir que modela la base de datos Relacional como si fuera un conjunto de clases, permitiendo que los datos de las mismas se manejen como objetos. Esto permite disminuir el abismo que existe entre la Orientación a Objeto y el modelo Relacional.
- Para la mejorar mantenibilidad de una aplicación es recomendable estructurarla en capas, incluyendo una capa de persistencia y una capa de negocios. LINQ to SQL facilita la implementación de esta división en capas, permitiendo trabajar con objetos.

Se concluye que es útil el manejo de LINQ to SQL, porque resuelve muchos de los problemas actuales que surgen al intentar manipular los datos provenientes de una base de datos Relacional. Además es de notar que la misma se encuentra integrada dentro de C# (así como también de VB), lo cual permite que la validez de las consultas sea realizada en tiempo de compilación.

6.1.1.4. LINQ to ENTITY

Las conclusiones que se obtuvieron como resultado de este estudio son las siguientes:

- Permite dividir la aplicación en dos capas bien diferenciadas: la capa de negocio (donde se manejan los objetos del negocio) y la capa de persistencia (donde se accede a la base de datos). Para la capa de negocio es transparente a que manejador de base de datos se está accediendo, lo que es una gran ventaja para el desarrollador. Esta característica es llamada muchas veces *Persistence Ingnorance.*

- Ofrece la posibilidad de manipular objetos, siendo estos generados por una base de datos Relacional. El manejar objetos ofrece grandes ventajas sobre el acceso directo a las tablas. Algunas de esas ventajas son:
    - Permitir la corrección de la sintaxis de una consulta que accede a la información provista por la base de datos en tiempo de ejecución.
    - Le ofrece la posibilidad al desarrollador de manejar un solo lenguaje integrado dentro del Framework de .Net.
- Otro punto a tener en cuenta es que LINQ to ENTITY implementa el patrón de diseño *Unit of Work*. Permite realizar el seguimiento de todas las operaciones que se realizan contra una estructura de datos, para que una vez terminadas las mismas, se puedan volcar los resultados a la base de datos, disminuyendo los accesos a la misma.
- Permite retardar la carga de un terminado objeto o colección de objetos (carga que suele ser muy costosa) en el momento justo, en que el objeto o la colección de objetos son necesarios. Esto produce una mejora en la eficiencia de las operaciones relacionadas a manipulación de objetos. Esa característica es llamada *Lazy Loading.*

En resumen LINQ to ENTITY ofrece la solución a los problemas que un desarrollador encuentra, cuando desea manipular la información provista por una base de datos Relacional.

Hay que tener en cuenta que LINQ to ENTITY no se encuentra en su versión final, lo cual produce que la utilización del mismo no sea la mejor, tanto por temas de performance, como por problemas que surgen al utilizarlo.

6.1.1.5. Active Software y LINQ

La utilización de LINQ por la empresa Active Software se ha realizado en variados proyectos, visto que permitió resolver problemas que antes resultaban más complejos de resolver, produciendo un código más sustentable y entendible. Lo más utilizado es LINQ to OBJECT, con la finalidad de realizar filtros a diferentes tipos de colecciones de objetos.

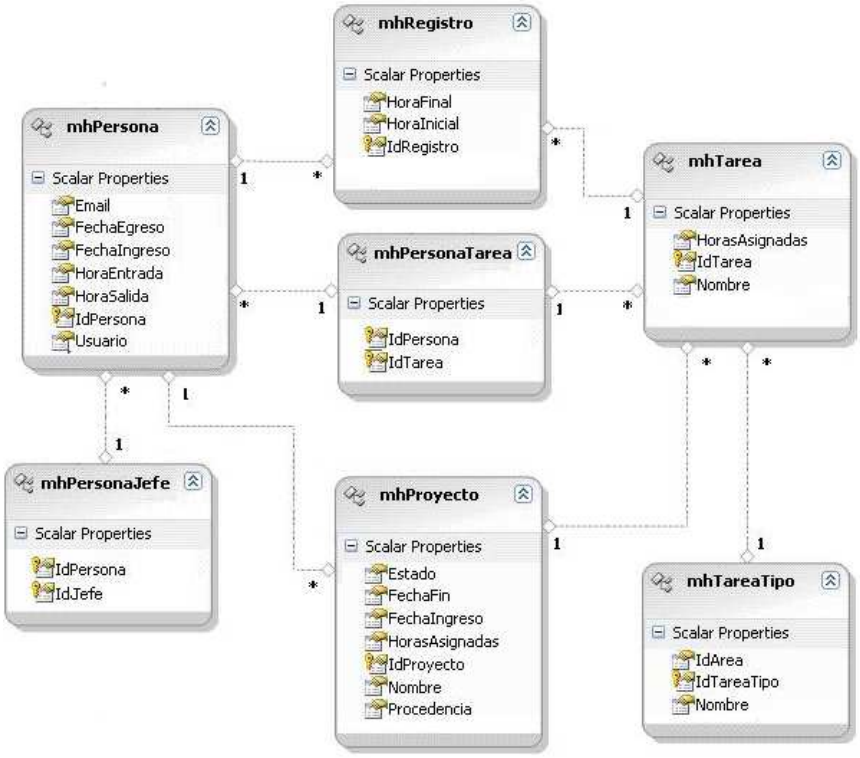
6.1.2. Conclusión del Aplicativo Desarrollado

A continuación se desarrollan las conclusiones obtenidas de la realización de la aplicación “Active.Management.HoursLinq”. Dichas conclusiones están basadas en comparaciones hechas entre, la modelización de base de datos Relacional que ofrece AF, y la que ofrece LINQ to ENTITY.

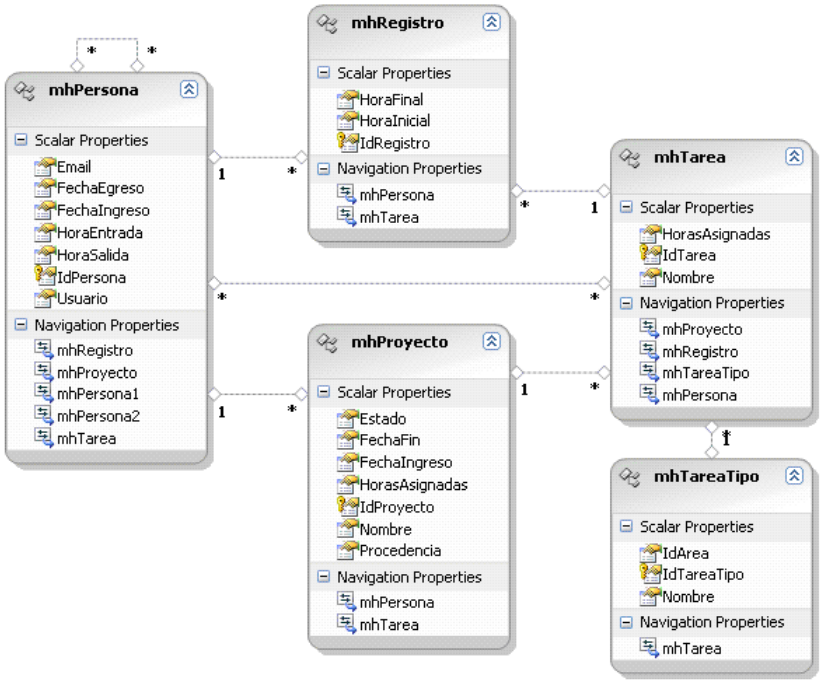
Se concluye que el modelado de la base de datos que realiza AF, es útil porque permite resolver problemas relacionados con la incompatibilidad de tipos y el acceso a los datos. Permitiendo que el usuario olvide a que origen de los datos esta accediendo (el usuario no sabe que manejador de base de datos está utilizando).

Pero AF tiene algunas características no muy buenas que se listaran a continuación que LINQ to ENTITY las soluciona:

- Realiza el mapeo directo de tabla a objeto. El mismo se puede ver en el siguiente diagrama :



En cambio con LINQ, el modelado de los objetos se realiza de forma más racionalizada, produciendo que el modelo obtenido se parezca al diseño de clases (pensando en un sistema Orientado a Objetos).Esto se muestra en el siguiente diagrama:



- A las consultas escritas utilizando AF no se le realiza ningún control, ni de la sintaxis utilizada, ni de los campos empleados en la misma. En cambio al utilizar LINQ to ENTITY, se permite manipular un conjunto de comandos que se encuentran incluidos en el lenguaje con el cual se está desarrollando, permitiendo que las consultas sean verificadas en tiempo de compilación
- Para la realización de la capa de datos utilizando AF, es necesario implementar una clase por tabla en la base de datos, esto lleva tiempo y puede ser fuente de muchos errores. En cambio con la utilización LINQ to ENTITY modelar la base de datos ser realiza de forma automática.

6.1.3. Conclusión Final

El siguiente proyecto logro sus cometidos: la realización de un análisis sobre LINQ, donde se exploran las diferentes ventajas y desventajas ofrecidas por el mismo. Basado en un estudio de performance, y un análisis de las diferentes características. Además en dicho análisis se muestra de qué manera se utilizó LINQ, para el desarrollo de variados tipos de aplicaciones dentro de la empresa Active Software.

El segundo cometido propuesto era el de construir una aplicación que permitiera ver en acción a LINQ to ENTITY, y compáralo con AF (dicha comparación estaba enfocada a características que ofrecía una u otra herramienta en el desempeño de sus funciones), lo cual se logro obteniéndose un conjunto de ventajas y desventajas de ambos.

El estudio de esta herramienta muestra en qué sentido va la evolución de las herramientas provistas por .Net, para el desarrollo de aplicaciones con persistencia en base de datos Relacional.

Como conclusión de la realización de este proyecto se pudo obtener que el manejo que impone la utilización de LINQ, los aportes positivos que tiene en la realización de aplicaciones, las alternativas que ofrece al permitir el acceso a diferentes orígenes de datos (base de datos Relacional, archivos XML y otros), y la mentalidad de poder juntar dos paradigmas que se presumían disjuntos. Permite que esta herramienta sea de gran utilidad para el desarrollo de software.

6.2. Trabajos a futuro para realizar, extensiones del trabajo

A continuación se muestra una lista de las diferentes tareas que se pueden realizar a futuro así como ideas para extender el proyecto.

- Realizar las mismas pruebas de performance que se hicieron para este informe pero con una versión mejorada de LINQ to ENTITY (una versión que resuelva los problemas que tiene la actual).
- Podría ser interesante realizar la misma implementación pero utilizando LINQ to SQL, visto que utilizan el mismo manejador de base de datos SQL Server.
- Por lo que se pudo ver en el informe las pruebas que se realizaron estaban orientadas a Aplicaciones de tipo Windows, sería muy interesante hacer las pruebas pero con Aplicaciones de tipo Web.
- Poder agregarle el manejo de usuarios y sesión al programa desarrollado, porque en realidad cualquier persona podría agregar y/o quitarle horas a otra. Así como ver información que solo debería ser relevante para un grupo selecto de personas.

6.3. Auto critica

- Se puede criticar a la organización de la documentación del proyecto, por que se podría haber hecho inicialmente un anexo conteniendo el estado del arte, donde se explicaría en detalle cada una de las herramientas con las que se estuvo trabajando y otras que están relacionadas al proyecto en sí, y luego otro conteniendo el estudio sobre la utilidad de LINQ, el cual expone las distintas pruebas realizadas y conclusiones obtenidas de las mismas, y no incluir en el mismo documento ambas cosas.

7. Glosario

Framework:

Es una estructura de soporte definidas para que a partir de ella puedan desarrollarse un sin fin de aplicaciones, relacionadas directamente a un dominio especifico, es decir, aun un conjunto de problema similares.

AF:

Es la sigla de Active Framework. Framework desarrollado por la empresa Active Software.

AS:

Es la sigla que identifica a la empresa Active Software

CLR:

Es la sigla que identifica a *Command Language Runtime*, lo cual significa Lenguaje Común en tiempo de ejecución, es el encargado de definir el ambiente de ejecución para los códigos de los programas

MVC:

Es la sigla para *Model View Controller*, patrón de diseño (desarrollado en los anexos).

Fuertemente Tipado:

Se dice que un lenguaje es *fuertemente tipado* si controla que no sean violados lo tipos de datos en un determinado código de programación.

Performance:

En este caso hace referencia a sinónimos de desempeño, rendimiento de una aplicación. Medida por el uso de diferentes recursos.

Sustentable:

Basados en la RAE, se puede decir que sustentar es: “Sostener algo para que no se caiga o se tuerza”, aplicado esta definición al desarrollo de software, se relaciona a, para que un producto de software sea sustentable el código del mismo debe ser entendible

y fácil de mantener, para que el mismo siempre pueda ser actualizado.

Protocolo:

Es un conjunto de reglas de comunicación que permiten comunicar a dos o más sistemas entre sí.

ORM:

Es la sigla para Object Relation Mapping. Es una técnica de programación para la conversión entre los tipos de datos incompatibles de una base de datos Relacional frente a un lenguaje de programación orientada a objeto.

Usabilidad:

Se refiere a la elegancia y claridad con la cual la interfase de usuario de un programa o website es diseñado. También se hace con usabilidad referencia a cuan claro es el código desarrollado.

Lazy loading:

Es un patrón de diseño útil y simple que se basa en ir cargando los distintos componentes de una clase cuando se van usando

Paradigma:

Modelo, ejemplar a seguir, desde el cual se piensa o se realizan hechos y teorías predominantes.

Profiler:

Es una herramienta de análisis de la performance, que mide el comportamiento de la ejecución de un programa, principalmente la frecuencia y duración de las llamadas a las funciones.

8. Referencias

8.1. Anexos y Libros

[1] Informe sobre evaluación de utilidad de LINQ
Anexo Proyecto Grado 2008
Autor – Antonio Malaquina Año - 2008/2009

[2] Manual de Usuario Active.Management.LinqHours
Anexo Proyecto Grado 2008
Autor – Antonio Malaquina Año - 2008/2009

[3] Conceptos Claves para la realización de Active.Management.LinqHours
Anexo Proyecto Grado 2008
Autor Antonio Malaquina Año - 2008/2009

[4] LINQ in ACTION
Autores – Fabrice Marguerie, Steve Eichert, JimWooly
Publicado en 2008

[5] Manual de Active Framework versión 1.0.
Realizado por la empresa Active Software
Autor – Active Software Año - 2008/2009

8.2. Links

[6] http://msdn.microsoft.com/es-es/library/bb397687.aspx
25 de Febrero de 2009

Glosario

[7] http://www.emprendedores.cl/estudios_trabajos/glosario.htm
02 de marzo de 2009

[8] http://www.ucsm.edu.pe/rabarcaf/vofici07.htm
17 de marzo de 2009

[9] http://www.networkdictionary.com/software/p.php
17 de marzo de 2009

FUERTEMENTE TIPADO

[10] http://www.alegsa.com.ar/Dic/tipado%20fuerte.php
02 de marzo de 2009

FRAMEWORK

[11] http://www.acm.org/crossroads/espanol/xrds7-4/frameworks.html
02 de marzo de 2009

SUSTENTAR

[12] http://buscon.rae.es/draeI/SrvltGUIBusUsual?TIPO_HTML=2&TIPO_BUS=3& LEMA=sustentar
02 de marzo de 2009

ORM

[13] http://en.wikipedia.org/wiki/Object-relational_mapping
02 de marzo de 2009

LAZY LOADING

[14] http://adeshoras.wordpress.com/2008/04/20/el-patron-de-diseno-lazy-loading/
02 de marzo de 2009

Quaere (LINQ para Java)

[15] http://www.javahispano.org/contenidos/es/quare__una_implementacion_de_linq_para_java_11/
13 de marzo de 2009

[16] http://andersnoras.com/blogs/anoras/archive/2007/09.aspx
13 de marzo de 2009

Programación Funcional

[17] http://www.fing.edu.uy/inco/cursos/progfunc/pmwiki/field.php/Materiales/Teorico
13 de marzo de 2009

LINQ to ENTITY vs NHibernate

[18] http://www.infoq.com/news/2008/05/Entity-Framework-Sparks-a-Debate
14 de marzo de 2009

[19] http://efvote.wufoo.com/forms/ado-net-entity-framework-vote-of-no-confidence/
14 de marzo de 2009

[20] http://www.mbeller.de/2007/12/reccomendation-nhibernate-linq-or.html
14 de marzo de 2009

[21] http://bugvanquisher.spaces.live.com/Blog/cns!DAED3FE79AA3FA09!434.entry?sa=556078283
14 de marzo de 2009

[22] http://blogs.msdn.com/dsimmons/archive/2008/05/17/why-use-the-entity-framework.aspx
14 de marzo de 2009