ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

Tema:

"ESTUDIO COMPARATIVO DE LINQ Y EXPRESIONES LAMBDA COMO PARADIGMAS DE PROGRAMACIÓN EN EL SISTEMA "HADE" APLICADO A COMPROTEC-ESPOCH"

Tesis de grado previa la obtención del título de

Ingeniería en Sistemas Informáticos

PRESENTADO POR:

Olimpia Elizabeth Mayorga Vemus

Rosario del Carmen Carrera Salgado

Riobamba-Ecuador

2010

DEDICATORIA

Esta investigación que representa un esfuerzo por superarme tanto en mi vida profesional como en la personal, se lo dedico a Dios que me da fortaleza espiritual en los momentos difíciles;

Especialmente con todo mi amor a mi familia, Para mis padres Juan y Carmen, por su comprensión y ayuda en todo momento. Me han enseñado con su ejemplo a encarar las adversidades sin perder nunca la dignidad ni desfallecer en el intento, a querer ser mejor cada día, a entender que no hay nada imposible y que sólo hay que esmerarse y sacrificarse, si es necesario, para lograr las metas que nos planteamos. Me han dado todo lo que soy como persona, mis valores, mis principios, mi perseverancia y mi empeño, y todo ello con una gran dosis de amor y sin pedir nunca nada a cambio.

A mis hermanos, cuñadas, sobrinos por acompañarme y apoyarme siempre en el transcurso de mi vida; también, a esa personita especial que llegó a formar parte vida.

A todos ellos,

muchas gracias de todo corazón.

Rosario Carrera

DEDICATORIA

La presente investigación se la dedico a Dios que me da la salud y la sabiduría suficiente para seguir con pasos firmes a un futuro prometedor, a toda mi familia en especial a mis padres Juan y Nelly que en todo este tiempo han sido el pilar fundamental en toda mi vida, siempre me han apoyado y están a mi lado orientándome e impulsándome en todo momento a conseguir mis metas, se que para darme todo lo necesario los dos se han sacrificado mucho por todo esto y mucho mas gracias, de manera especial a Johnny Ordoñez es una persona fundamental en mi vida juntos hemos logrado superar satisfactoriamente todas las adversidades que se nos han presentado, el es mi fuerza, mi hombro de consuelo, mis ganas de seguir adelante, gracias por toda tu ayuda amor.

Olimpia Mayorga

AGRADECIMIENTO

Primero nos gustaría agradecer sinceramente a nuestra directora y tutora de Tesis, Ing. Lorena Aguirre, su esfuerzo y dedicación, sus conocimientos, sus orientaciones, su manera de trabajar, su persistencia, su paciencia y su motivación han sido fundamentales para nuestra formación como investigadoras. Ella ha inculcado en nosotras un sentido de seriedad, responsabilidad y rigor académico sin los cuales no podríamos tener una formación completa como investigadoras. A su manera, ha sido capaz de ganarse nuestra lealtad y admiración; también a la Ing. Pamela Buñay, de manera especial al Ing. Jonny Ordoñez, a los catedráticos de nuestra Escuela y Facultad por permitirnos alcanzar una meta importante en nuestras vidas como es lograr obtener nuestro título profesional.

Para ellos,

muchas gracias por todo.

AUTORAS

DERECHOS DE AUTORÍA.

Nosotras, Olimpia Elizabeth Mayorga Vemus y Rosario del Carmen Carrera Salgado, somos responsables de las ideas, doctrinas y resultados expuestos en esta Tesis y el patrimonio intelectual de la misma pertenece a la Escuela Superior Politécnica de Chimborazo.

FIRMAS RESPONSABLES Y NOTAS

ING. IVÁN MENES
DECANO DE LA FACULTAD DE INFORMÁTICA Y ELECTRÓNICA _____ _____

ING. RAÚL ROSERO
DIRECTOR DE LA ESCUELA DE INGENIERÍA EN SISTEMAS _____ _____

ING. LORENA AGUIRRE
DIRECTORA DE TESIS _____ _____

ING. PAMELA BUÑAY
MIEMBRO DE TESIS _____ _____

Tec. CARLOS RODRÍGUEZ
DIRECTOR DEL CENTRO DE DOCUMENTACIÓN _____ _____

NOTA DE LA TESIS _____

ÍNDICE GENERAL

ÍNDICE DE CONTENIDO:

INDICE DE TABLAS

INDICE DE FIGURAS

CAPÍTULO I

MARCO REFERENCIAL

1.1 INTRODUCCIÓN

Los Paradigmas de Programación

Desde el principio del desarrollo de programas de computación, el mundo de la programación se ha visto enmarcado en un conjunto paradigmas o filosofías para la construcción de software definidas por las necesidades de automatización, las características del programa y su aplicación. En función de la naturaleza del programa a crear, un paradigma resulta más apropiado que otro, inclusive en la práctica es muy común que se mezclen dando lugar a la programación “multiparadigma” explotando las ventajas de los enfoques utilizados. En términos generales, se puede describir dos estilos fundamentales de programación del cual se derivan los paradigmas conocidos y que contienen a los lenguajes de programación más utilizados: la Programación Declarativa y la Programación Imperativa.

En la programación declarativa las sentencias que se utilizan lo que hacen es describir el problema que se quiere solucionar, pero no las instrucciones necesarias para

solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información a partir de la descripción realizada. En la programación imperativa, a diferencia de la declarativa se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar el problema.

Derivado de la programación declarativa, el paradigma de programación funcional está basado en la utilización de funciones matemáticas cuyo objetivo es conseguir un lenguaje "*expresivo y matemáticamente elegante*". Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo, sino como funciones puramente matemáticas; carecen de asignaciones de variables y no poseen construcciones estructuradas como la secuencia o la iteración lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas.

El Enfoque de los nuevos lenguajes de programación frente a la Programación Tradicional

En la actualidad, con la aparición de nuevos lenguajes de programación de alto nivel, resulta muy interesante estudiar los nuevos enfoques de desarrollo que brindan, las ventajas y desventajas que poseen frente a los esquemas cotidianos de construcción de software; su utilidad, complejidad y productividad en comparación a las filosofías comúnmente utilizadas. La importancia del estudio se incrementa a medida que las aplicaciones siguen creciendo en complejidad y tamaño, haciendo evidente la necesidad de mejorar las técnicas de programación para obtener software con mejores características en desempeño, calidad y con mejores índices en productividad.

LINQ y expresiones Lambda como paradigma de Programación Funcional

Los programadores, ahora familiarizados con conceptos como las clases, objetos y métodos, tablas y sentencias SQL pueden expandir su capacidad creativa combinando técnicas y paradigmas para crear un programa consistente, expresivo, entendible, flexible y elegante en menos tiempo de desarrollo. Una de las la tecnologías para lograrlo es *LINQ (Language Integrated Query)* que integra sintaxis del acceso a base de datos de la programación tradicional en los lenguajes C# y Visual Basic elevando la comprensión y el manejo de elementos de programación. Este código imperativo enriquecido con conceptos y técnicas de la programación funcional, como expresiones Lambda, permite a los desarrolladores incrementar su eficiencia cumpliendo todos los requerimientos con menos líneas de código.

1.2 PROBLEMATIZACIÓN

1.2.1 TÍTULO DE LA INVESTIGACIÓN

ESTUDIO COMPARATIVO DE LINQ Y EXPRESIONES LAMBDA COMO PARADIGMAS DE PROGRAMACIÓN EN EL SISTEMA “HADE” APLICADO A COMPROTEC-ESPOCH.

1.2.2 DETERMINACIÓN DE LA PROBLEMÁTICA

HADE (Sistema de Gestión de Proyectos de la ESPOCH), el Sistema Multiparadigma

La Comisión de Proyectos y Transferencia de Tecnología (COMPROTEC), de la ESPOCH, se encarga de presentar al Consejo de Investigación y Desarrollo el plan operativo anual, identificar, formular y gestionar proyectos de prestación de servicios, consultoría y asesoría en coordinación con las unidades académicas; establecer y mantener la cooperación interinstitucional con empresas públicas y privadas para el

desarrollo científico y tecnológico; colaborar con organismos, instituciones, empresas públicas y privadas, nacionales e internacionales para la investigación.

COMPROTEC posee un registro con los PROYECTOS/CONCURSOS que son los proyectos que se plantean y son válidos pero que por falta de apoyo -en especial económico- no se llevan a cabo aún y se necesita darlos a conocer utilizando cualquier mecanismo de publicación, para poder conseguir el auspicio e inicio de ejecución de los mismos. La unidad COMPROTEC se encarga de llevar el registro pertinente del monitoreo, seguimiento y evaluación de proyectos en donde se plasman las actividades planificadas, ejecutadas, los resultados obtenidos y evaluación de los objetivos esperados y logrados de los proyectos que se ejecutaron.

Para aportar con la investigación y el desarrollo en COMPROTEC se propone desarrollar una aplicación que contenga las publicaciones de los Proyectos/Concursos en la Web y el Sistema de Control de Monitoreo, Seguimiento y Evaluación de Proyectos, todo esto como una contribución para generar información útil y eficaz a partir de los datos que posee COMPROTEC mejorando su gestión, planificación y administración. La adecuada aplicación del nuevo sistema evitara pérdidas de tiempo, esfuerzo y recursos económicos, así como permitirá la consecución de los objetivos y metas propuestas por parte del personal que conforma la comisión.

1.3 DEFINICIÓN DE OBJETIVOS

1.3.1 OBJETIVO GENERAL

Realizar el estudio comparativo de LINQ y Expresiones Lambda como Paradigmas de programación funcional frente al de programación imperativa, en la implementación del Sistema de Gestión de Proyectos “HADE” aplicado a COMPROTEC-ESPOCH.

1.3.2 OBJETIVOS ESPECÍFICOS

- Estudiar el Lenguaje de Consultas Integrado LINQ junto al uso de Expresiones Lambda como Paradigma de Programación Funcional y sus ventajas frente al esquema de programación imperativa;

- Definir parámetros y aspectos de comparación entre los paradigmas de programación Funcional e Imperativa, orientados a brindar mayor productividad al programador;

- Realizar el estudio comparativo mediante la construcción de dos prototipos de prueba frente a las filosofías y técnicas de programación referidas en este proyecto, que permitan establecer la que brinde mayor productividad al programador;

- Desarrollar el Sistema de Gestión de Proyectos “HADE” de acuerdo al paradigma de programación más adecuada.

1.4 JUSTIFICACIÓN DEL PROYECTO DE TESIS

Se ha visto la necesidad de realizar este estudio comparativo de LINQ y expresiones lambda como paradigma de programación funcional, frente al paradigma de programación imperativa ya que el avance de la tecnología incide en el desarrollo de la sociedad desde varias perspectivas.

LINQ define operadores de consulta estándar que permiten filtrar, enumerar y crear proyecciones a nivel de programación de varios tipos de colecciones: arreglos, clases enumerables, XML, conjuntos de datos desde bases de datos relacionales y orígenes de datos de terceros; usando la misma sintaxis del lenguaje de acceso a datos SQL.

De esta forma brinda enorme simplificación que significa usar muy pocas líneas de código en el lenguaje de programación natural para realizar operaciones y obtener resultados productivamente. Al usar LINQ junto con expresiones Lambda para usar funciones personalizadas en las operaciones sobre los enumerables se otorga mayor expresividad a la programación permitiendo resolver problemas complejos en menos líneas de código.

Adicionalmente, la plataforma .NET Framework 3.5 y el lenguaje C# 3.0 proveen un conjunto de mejoras y herramientas para el soporte y explotación de LINQ. Para garantizar el buen desempeño del desarrollo y ejecución de la aplicación se utilizará la metodología de desarrollo Microsoft Solutions Framework (MSF) para el control de las etapas del proyecto de manera que asegure que todos los elementos participantes: personas, procesos y herramientas puedan ser favorablemente administrados; el Sistema Operativo XP, el motor de Base de Datos SQL Server 2000, la herramienta de programación Visual Studio.Net 2008. Todos los anteriormente mencionados serán utilizados para la creación y ejecución de nuestra aplicación, ya que la ESPOCH cuenta con los licenciamientos respectivos; utilizando el lenguaje de programación C#.

El desarrollo de la aplicación permitirá difundir la información coherente y eficaz de los Proyectos/Concurso para que los usuarios o empresas nacionales e internacionales interesadas en apoyar un proyecto en particular puedan contactarse con el director de COMPROTEC, también le permitirá a la persona autorizada administrar por completo el control de monitoreo, seguimiento y evaluación de proyectos que se están ejecutando lo cual sin duda ayudará en la toma de decisiones a la COMISIÓN DE PROYECTOS Y TRANSFERENCIA TECNOLÓGICA.

1.5 PLANTEAMIENTO DE LA HIPÓTESIS

“El estudio comparativo del LINQ y Expresiones Lambda como Paradigmas de programación funcional frente al de programación imperativa, permitirá seleccionar la técnica que mejore la productividad en el desarrollo del Sistema de Gestión de Proyectos “HADE”.

CAPÍTULO II:

MARCO TEÓRICO

2.1 INTRODUCCIÓN

El presente trabajo tiene como objetivo general el desarrollo de un sistema web utilizando la tecnología LINQ para el acceso a datos y el manejo de objetos de programación. LINQ enriquecido con el uso de Expresiones Lambda, brinda alto grado de simplificación programática al poder acceder a diferentes fuentes de información de forma consistente mediante una sintaxis expresiva y familiar desde el punto de vista de desarrollo. Sin embargo, LINQ es más que un conjunto de sentencias para el acceso a bases de datos.

En su mayoría, los desarrolladores de hoy comprenden conceptos de programación orientada a objetos y junto a ellos, las tecnologías y características relacionadas: clases, métodos y objetos. Microsoft ha unificado y empaquetado en las nuevas versiones del .NET Framework, las ventajas de un modelo de consulta estandarizado para la manipulación de objetos, XML, colecciones y datos, elevando la utilidad de LINQ. Ahora, es posible disfrutar de los beneficios de un patrón declarativo expresado en un lenguaje de programación imperativo de .NET.

La recopilación a continuación es el resultado de una investigación responsable y enfocada, cuyo objetivo es ofrecer una visión general y completa de la tecnología LINQ y Visual Studio 2008 así como resaltar sus principales ventajas y características. De esta forma, este capítulo servirá de base para comprender el esquema en la que se encuentra construido el Sistema de Gestión de Proyectos COMPROTEC.

2.2 MICROSOFT VISUAL STUDIO 2008: UNA VISIÓN GENERAL

2.2.1 Características y Mejoras del Ambiente de Desarrollo

Visual Studio 2008 es la herramienta que más posibilidades ofrece para los desarrolladores de aplicaciones que elaboran programas e interfaces en los departamentos de TI, puesto que las mejoras que trae consigo junto al aprovechamiento de las nuevas tecnologías propias y las introducidas por sus coprotagonistas de estreno, crean un ambiente propicio para el desarrollo simplificado de aplicaciones inteligentes. Y gracias a la integración con Windows Vista y Office System 2007 será posible crear aplicaciones seguras, controlables y confiables.

A las mejoras de desempeño, escalabilidad y seguridad con respecto a la versión anterior, se agregan entre otras, las siguientes novedades:

Figura 2.1: MICROSOFT VISUAL STUDIO 2008

- Logrado mejoramiento en las capacidades de Pruebas de Unidad permiten ejecutarlas más rápido independientemente de si lo hacen en un entorno IDE o desde una línea de comandos. Se incluye además un nuevo soporte para diagnosticar y optimizar un sistema a través de herramientas de pruebas de Visual Studio. Con ellas se podrán ejecutar perfiles durante las pruebas para que ejecuten cargas, prueben procedimientos contra un sistema y registren su comportamiento; y utilizar herramientas integradas para perfilar, depurar y optimizar.

- Con Visual Studio Tools for Office (VSTO) integrado con Visual Studio 2008 Professional Edition es posible desarrollar rápidamente aplicaciones de alta calidad basadas en la interfaz de usuario (UI) de Office que personalicen la experiencia del usuario y mejoren su productividad en el uso de Word, Excel, PowerPoint, Outlook, Visio, InfoPath y Project. Una completa compatibilidad para implementación con ClickOnce garantiza el entorno ideal para una fácil instalación y mantenimiento de las soluciones Office.

- Visual Studio 2008 permite incorporar características del nuevo Windows Presentation Foundation sin dificultad tanto en los formularios de Windows existentes como en los nuevos. Ahora es posible actualizar el estilo visual de las aplicaciones al de Windows Vista debido a las mejoras en Microsoft Foundation Class Library (MFC) y Visual C++. Visual Studio 2008 permite mejorar la interoperabilidad entre código nativo y código manejado por .NET. Esta integración más profunda simplificará el trabajo de diseño y codificado.

- LINQ (Language Integrated Query) es un nuevo conjunto de herramientas diseñado para reducir la complejidad a través de extensiones para C++ y Visual Basic así como para Microsoft .NET Framework, que permiten filtrar, enumerar, y crear proyecciones de muchos tipos y colecciones de datos utilizando todos la misma sintaxis prescindiendo del uso de lenguajes especializados como SQL o XPath.

- Visual Studio 2008 ahora permite la creación de soluciones multiplataforma adaptadas para funcionar con las diferentes versiones de .Net Framework: 2.0. (Incluido con Visual Studio 2005), 3.0 (incluido en Windows Vista) y 3.5 (incluido con Visual Studio 2008).

- NET 3.5 incluye biblioteca ASP.NET AJAX para desarrollar aplicaciones más eficientes, interactivas y altamente personalizadas que funcionen para todos los browsers más populares y utilicen las últimas tecnologías y herramientas Web, incluyendo Silverlight y Popfly

Las nuevas características y mejoras del lenguaje de Visual Studio 2008, en Visual C# y Visual Basic. Net, junto con LINQ permite tomar algunas ventajas sobre los otros lenguajes de programación.

Visual Studio incluye varios solucionadores de diseño que pueden ayudar a los desarrolladores a crear muchos aspectos SQL como entidades, clases y asociaciones. Por ejemplo el Objeto diseñador Relacional (O/R Designer), proporciona una interfaz visual para la creación y el diseño de Linq.

2.3 LA EVOLUCIÓN DE C# EN EL VS 2008: C# 3.0

2.3.1 Programación Funcional

Para conocer sobre la programación funcional se debe considerar en primer lugar que es un paradigma. La palabra paradigma viene del griego paradeigma que significaba originalmente: ejemplo ilustrativo. Un paradigma de programación es una forma de pensar a la hora de programar, una manera de construir programas. Existen tres formas esenciales de enfocar la programación:

Programación imperativa: orientada a la máquina de Von Newman;
Programación funcional: basada en el concepto de función matemática;
Programación lógica: basada en la lógica de predicados.

Se denomina **programación declarativa**, en contraposición a la programación imperativa, al conjunto de la programación funcional y la programación lógica. En la programación declarativa los programas describen "qué" se desea obtener.

La Programación funcional es un paradigma de programación declarativa basado en la utilización de funciones matemáticas. Los lenguajes funcionales se caracterizan por expresar, a través de funciones, **qué se desea conseguir** en un programa.

El primer lenguaje de programación funcional fue **LISP** (McCarthy 1959), basado en el procesamiento de listas. Otros lenguajes funcionales: APL (1962), ISWIM (1966), SCHEME (1975), FP (1977), HOPE (1980), MIRANDA (1985), ML (1986), HASKELL (1988), etc.

Su creación es debida a las necesidades de los investigadores en el campo de la inteligencia artificial y en sus campos secundarios del cálculo simbólico, pruebas de teoremas, sistemas basados en reglas y procesamiento del lenguaje natural.

Evolución de la Programación Funcional

- ✓ Inicios: Cálculo lambda de Church (años 30)

- ✓ El primer lenguaje funcional desarrollado fue LISP (1958), que se aplicó en el área de Inteligencia Artificial. Es la base del actual Scheme, que no es puramente funcional

- ✓ Años 60 y 70: ISWIM, FP y ML

- ✓ Años 80: Eclosión de lenguajes funcionales SASL, KRC y Miranda, Haskell, Hope, Wadler, CAML, etc.

- ✓ Haskell: Nace a partir de la conferencia FPCA'87 y fue desarrollado por las universidades de Yale y Glosgow, con objeto de reunir todas aquellas características fundamentales del paradigma funcional

Características de la programación funcional

Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo, sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades como la transparencia referencial (el significado de una expresión depende únicamente del significado de sus subexpresiones), y por tanto, la carencia total de efectos laterales.

Otras características propias de estos lenguajes son la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas).

Existen dos grandes categorías de lenguajes funcionales: los funcionales puros y los híbridos. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al admitir conceptos tomados de los lenguajes imperativos, como las secuencias de instrucciones o la asignación de variables. En contraste, los lenguajes funcionales puros tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial, algo que no se cumple siempre con un lenguaje funcional híbrido.

Por ejemplo al usar la programación estructural.

▼ Tabla I: Programación en Pascal con efectos laterales

```
PROGRAM ConEfectosLat;
VAR estado: boolean;
        FUNCTION Efectos (n:integer): integer;
        BEGIN
            IF (estado) THEN
                Efectos :=  n
            ELSE
                Efectos := 2*n+1;
            estado := NOT estado;
        END;  (* Efectos *)
BEGIN
        estado := TRUE;
        WriteLn (Efectos(1), ' ', Efectos(1));
        WriteLn (Efectos(2), ' ', Efectos(2));
END.
```

En los programas funcionales, se dispone de diferentes tipos de datos predefinidos (enteros, reales, booleanos y caracteres), y de tipos de datos definidos por el programador. No hay noción de posición de memoria y, por tanto, no hay necesidad de una instrucción de asignación:

- Gestión de memoria -> libera de carga al programador
- No existe el concepto de posición de memoria
- x = <expresión> => x, es, por definición, igual a <expresión>
- En el mismo programa no puede aparecer otra vez x = …, no sería correcto

Los bucles se modelan a través del uso de la recursividad, ya que no hay manera de incrementar o decrementar el valor de una variable.

Estructuras de control en los programas funcionales:

1) composición funcional
2) construcción condicional

3) recursividad

Funciones orden superior:

- Funciones tratadas como valores de primera clase -> Almacenadas en estructuras de datos, pasadas como argumentos y devueltas como resultados

▼ Tabla II: Ejemplo de una Función de Orden Superior

```
reaplica (f:Integer -> Integer, x:Integer) : Integer      - Programa Principal
Begin
        reaplica:= f ( f (x) );                                  ……
end                                                              write(replica (incr,0));
incr (n: Integer) : Integer
Begin                                                  ……
        incr:= n + 1;
end
```

Tipado fuerte -> detección de errores en tiempo de compilación

- El programador no está obligado a declarar el tipo de las expresiones
- El compilador, mediante un algoritmo, infiere el tipo de las expresiones
- Si se declara el tipo de una expresión -> compilador lo chequea

▼ Tabla III: Ejemplo del uso de un Tipeado fuerte

```
eligeSaludo x = if x then “adios” else “hola”
```

Polimorfismo -> aumenta la reusabilidad

- El tipo de una función depende de un parámetro

▼ Tabla IV: Ejemplo del uso de polimorfismo

```
long L = If vacia(L) then 0 else 1 + long(cola(L))
```

Orden de evaluación normal y evaluación perezosa:

- Las expresiones (argumentos formales, operaciones aritméticas, etc.) se evalúan en el momento en que se necesiten.

▼ Tabla V: Ejemplo del uso de expresiones

```
g (x:Integer) :Integer
Begin
      (*Bucle infinito*)
      while true do x:=x
End;

f (x,y: Integer): Integer
Begin
            f:=x + 3
End;
Evaluación tradicional
(ansiosa) -> Bucle infinito

Evaluación perezosa -> 7
```

```
{Programa Principal}
Begin
write(f(4,g(5)));

End.
```

2.3.2 Expresiones Lambda

Una expresión lambda es una función sin nombre que calcula y devuelve un solo valor. Se pueden utilizar las expresiones lambda dondequiera que un tipo de delegado sea válido. La instrucción **RemoveHandler** es una excepción. No puede pasar ninguna expresión lambda para el parámetro de delegado de

RemoveHandler.

El ejemplo siguiente es una expresión lambda que incrementa su argumento y devuelve el valor.

Function (num As Integer) num + 1

Dado que una expresión lambda es una expresión, sólo se puede utilizar como parte de una instrucción.

Expresiones Lambda en Consultas

En Language-Integrated Query (LINQ), las expresiones lambda están debajo de muchos de los operadores de consulta estándar. El compilador crea las expresiones lambda para capturar los cálculos definidos en los métodos de consulta básicos como **Where, Select, Order By, Take While,** etc. La sintaxis de una expresión lambda se parece a la de una función estándar. Las diferencias son las siguientes:

- Una expresión Lambda no tiene nombre.
- Las expresiones lambda no pueden tener modificadores, como **Overloads** u **Overrides.**
- Las expresiones lambda no utilizan una cláusula **As** para designar el tipo de valor devuelto de la función. En su lugar, el tipo se deduce del valor en el que se evalúa el cuerpo de la expresión lambda. Por ejemplo, si el cuerpo de la expresión lambda es **Where cust.City = "London"**, su tipo de valor devuelto es **Boolean**.
- El cuerpo de la función debe ser una expresión, no una instrucción. El cuerpo puede estar formado por una llamada a un procedimiento de función, pero no una llamada a un subprocedimiento.
- No hay ninguna instrucción **Return**. El valor devuelto por la función es el valor de la expresión del cuerpo de la función.
- No hay ninguna instrucción **End Function**.

- Todos los parámetros deben de tener tipos de datos especificados o se deben deducir.
- No se permite los parámetros Optional y ParamArray.
- No se permiten los parámetros Generic.

Como resultado de estas restricciones y de las maneras en las que se utilizan las expresiones lambda, éstas suelen ser cortas y sencillas.

Contexto

Una expresión lambda comparte su contexto con el método dentro del que se define. Tiene los mismos derechos de acceso que cualquier código escrito en el método contenedor. Esto incluye el acceso a las variables miembro, funciones y subs, **Me**, así como a los parámetros y las variables locales del método contendor.

El acceso a las variables locales y a los parámetros del método contenedor se puede ampliar una vez transcurrida la duración de dicho método. En tanto que un delegado que hace referencia a una expresión lambda no esté disponible para la recolección de elementos no utilizados, se conserva el acceso a las variables del entorno original.

Por Ejemplo: En el cálculo del interés simple, dentro del campo de las matemáticas financieras, entran en juego una serie de ecuaciones que deseamos implementar en nuestro código.

Interés = Capital * Tiempo * Tasa de interés

Valor Futuro = Capital * (1 + Tasa * Tiempo)

Capital = Valor Futuro * (1+ Tasa * Tiempo) -1

C# de los .Net Frameworks 3.0 y 3.5 incluye mediante las expresiones lambda una sintaxis mucho más clara y concisa. Una posible implementación de las ecuaciones implicadas en el cálculo del interés podría ser la siguiente:

▼ Tabla VI: Definición de una Expresión Lambda

```
public static class InteresSimple
        {
                public static Func<double, int, Single, double> Interes =
                        (C, t, i) => (C * t * i);
                public static Func<double, int, Single, double> ValorFuturo =
                        (C, t, i) => (C * (1 + i * t));
                public static Func<double, int, Single, double> Capital =
                        (VF, t, i) => (VF * Math.Pow((1 + i * t), -1));
        }
```

2.3.3 Cerradura (Closure)

Las cerraduras –también denominadas Cerraduras Léxicas- representan una mejora en los lenguajes de programación incluidos dentro de la Especificación Común de Lenguajes (Common Languages Specification o CLS por sus siglas en inglés) basados en el .NET Framework 3.5. Consiste principalmente en que una función o expresión Lambda puede referirse y alterar los valores de las variables definidas dentro del mismo contexto de declaración de la función. En C#, las cerraduras se implementan mediante el uso de delegados anónimos. A continuación se muestra un ejemplo el uso de Cerradura en C#:

▼Tabla VII: Utilización de un método extensor

```
//Declaración de Variable
int x = 10;

//Uso de la variable dentro del Delegado Anónimo
Action<int> addToX = new Action<int>(
                delegate(int y)
                  {
                     x = x + y;
                  } );
addToX(10);
Console.WriteLine(x);
```

En el ejemplo anterior se observa que la variable x puede ser accedida desde el interior de la declaración del delegado anónimo. Algo muy parecido sucede con el uso de expresiones lambda, como se muestra a continuación:

▼Tabla VIII: Utilización de un método extensor con expresiones lambda

```
public static double Operar(int valor)
{
   //Declaración de Variable
   double factor = 3.141516;

  //Declaración de Expresión Lambda. Usando la variable
  //factor dentro del cuerpo de la expresión Lambda
   Func<int, double> Calcular = x => x * factor;

   return Calcular(valor);
}
```

2.4 LINQ (EL LENGUAJE INTEGRADO DE CONSULTAS)

2.4.1 Introducción a LINQ: ¿Qué es y Cómo Funciona?

Esta sección ofrece un panorama general de las capacidades de LINQ y el nivel de consulta de operadores. El Lenguaje Integrado de Consultas o LINQ –por sus siglas

en inglés: *Language Integrated Query*– es un modelo uniforme de programación para todo tipo de información. LINQ permite ejecutar consultas y manipular información con un modelo consistente que es independiente de las fuentes de datos. Existen varias concepciones o percepciones de LINQ, descritas a continuación:

- LINQ es otra herramienta para consultas SQL embebidas en código;
- LINQ es todavía una capa de abstracción de datos;
- LINQ es un conjunto de operadores de consulta estándar que ofrece potentes servicios de consulta dentro .Net Framework, así como en los lenguajes de C# y VB.NET.

Todas estas descripciones son correctas, pero hay que tomar en cuenta algo importante:

LINQ puede hacer más de simples consultas SQL embebidas, es más simple y fácil que un modelo de programación uniforme, y es más que una capa de modelado de información.

Figura 2.2: Arquitectura de LINQ

LINQ es una tecnología que simplifica y unifica la implementación de cualquier tipo de acceso a datos. LINQ no obliga a utilizar una arquitectura específica, si no que te facilita la implementación de arquitecturas ya existentes para el acceso a datos.

Actualmente la información que se maneja por un programa puede pertenecer a diferentes dominios de datos: un arreglo de objetos, un documento XML, una base de datos, un archivo de texto, una clave de registro, un mensaje de correo electrónico, un mensaje de SOAP, un archivo de Excel de Office; en fin muchos recursos de datos.

Cada dominio de datos tiene un modelo de acceso específico, por ejemplo si se desea hacer una consulta a base de datos ocupamos el lenguaje SQL, si se desea navegar en un XML usamos DOM o XQuery, si se desea obtener información de un arreglo, se hace a través de una iteración, si se desea ver un documento de texto se puede utilizar Office, y así por el estilo.

Realmente el problema es poder unificar todo este tipo de estructuras de acceso, que es lo que se ha venido intentando anteriormente. LINQ viene a resolver este tipo de problemas, brindándonos una manera uniforme de acceso y manipulación de información.

Componentes

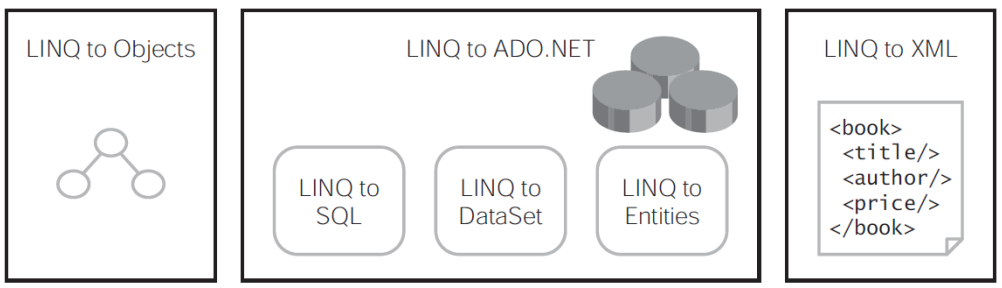


Figura 2.3: Componentes de LINQ

Debido a la flexibilidad y potencia de LINQ, se podrá observar un conjunto de sistemas y productos compatibles con LINQ. Prácticamente todos los datos almacenados pueden ser buenos candidatos para soportar consultas en Linq. Esto incluye bases de datos, Microsoft Active Directory, el registro, archivos del sistema, un archivo Excel, y así sucesivamente**.**

Linq para Objetos

LINQ para Objetos (LINQ to Objects) es el nombre otorgado a la interfaz IEnumerable<T> para el manejo de los Operadores de Consulta Estándar. Es este LINQ To Objects que permite ejecutar consultas contra arreglos y colecciones de datos en memoria. Los Operadores de Consulta Estándar son métodos estáticos de la clase System.Linq.Enumerable que son usados por las consultas LINQ to Objects.

Uno de los aspectos del diseño de LINQ es que las consultas pueden ser ejecutadas a través de algunos orígenes de datos. Si un objeto implementa la interfaz IEnumerable, entonces LINQ puede acceder a la data a tras del objeto. Por ejemplo, suponga que necita buscar varios usuarios en la carpeta y sub-carpetas ***My Documents***

▼ Tabla IX: Búsqueda dentro de la carpeta My Documents

```
using SIO = System.IO;
string[] files;
string mydocs;
mydocs =
Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
files = SIO.Directory.GetFiles(mydocs, "*.*", SIO.SearchOption.AllDirectories);
var query = from file in files
let lasthour = DateTime.Now.Subtract(new TimeSpan(0, 1, 0, 0))
where SIO.File.GetLastAccessTime(file) >= lasthour &&
(SIO.File.GetAttributes(file) & SIO.FileAttributes.System) == 0
select file;
```

Note la presencia de la sentencia **let**, la cual permite la definición de las variables locales dentro de la consulta, **let** se usa para mejorar la legibilidad y eficiencia a través de factores de operación comunes.

Este ejemplo no es muy sorprendente, sin embargo lo que se está haciendo realmente es iterar a través de un arreglo de nombre de archivos (“files”), y no del sistema de archivos como tal. Pero esto es parte del diseño de .NET Framework no una limitación de LINQ. Un similar ejemplo de búsqueda de archivos, el cual es fácilmente usado en LINQ por iteraciones de líneas de archivos.

▼ Tabla X: Búsqueda de Archivos

```
string filename = ...; // file to search
var lines = from line in SIO.File.ReadAllLines(filename)
where line.Contains("class")
select line;
```

En este ejemplo, se puede leer todas las líneas dentro de un arreglo y entonces se puede buscar con **Select** las líneas que contienen la secuencia de caracteres “class”.

Linq para XML

Linq para XML (Linq to XML) es el nombre otorgado a la interfaz LINQ API dedicado a trabajar con XML. Esta interfaz fue conocida anteriormente como XLinq. Microsoft ha añadido bibliotecas XML necesarias para Tabla XLIIcon Linq, ha corregido otras deficiencias en XML DOM, lo que hace más fácil que nunca trabajar con XML. Para tomar ventaja de LINQ para XML, se debe referenciar al ensamblado (o a la librería) System.Xml.Linq.dll en el proyecto, y tener una directiva Using tal como se muestra a continuación: *using System.Xml.Linq.*

Desde un inicio, LINQ fue diseñado para manipular datos XML tal fácil como manipular datos relacionales. LINQ to XML representa una nueva API para el desarrollo basado en XML, equivalente al poder de XPaht y XQuery pero todavía más simple para la mayoría de desarrolladores.

Por ejemplo, asuma que la fuente de datos para nuestra aplicación de horarios de hospital es un documento XML almacenado en el archivo *SchedulingDocs.xml*. Aquí la estructura básica del documento

▼ Tabla XI: Estructura del documento SchedulingDocs.xml

```
<?xml version="1.0" standalone="yes"?>
<SchedulingDocs>
<Calls>
<Call>
<Initials>mbl</Initials>
<DateOfCall>2006-10-01T00:00:00-05:00</DateOfCall>
</Call>
.
.
.
</Calls>
<Doctors>
<Doctor>
<Initials>ay</Initials>
<GivenFirstName>Amy</GivenFirstName>
<FamilyLastName>Yang</FamilyLastName>
<PagerNumber>53300</PagerNumber>
<EmailAddress>ayang@uhospital.edu</EmailAddress>
<StreetAddress>1400 Ridge Ave.</StreetAddress>
<City>Evanston</City>
</Doctor>
.
.
.
</Doctors>
<Vacations>
<Vacation>
<Initials>jl</Initials>
<DateOfDayOff>2006-10-03T00:00:00-05:00</DateOfDayOff>
</Vacation>
.
.
.
</Vacations>
</SchedulingDocs>
```

Al usar LINQ, se puede cargar este documento como se muestra a continuación

▼ Tabla XII: Cargar los datos del documento XML usando LINQ

```
import System.Xml.XLinq; // LINQ to XML
XElement root, calls, doctors, vacations;
root = XElement.Load("SchedulingDocs.xml");
calls = root.Element("Calls");
doctors = root.Element("Doctors");
vacations = root.Element("Vacations");
```

Se puede ahora tener acceso a los tres elementos principales del documento XML: Doctors, Calls y Vacations. Para seleccionar todos los doctores y encontrar todos los doctores que viven en Chicago, se usa la sentencia de consulta:

▼ Tabla XIII: Acceso y Búsqueda en la Aplicación Hospital

```
var docs = from doc in doctors.Elements()
select doc;

// Busque de doctores que viven en Chicago

var chicago = from doc in doctors.Elements()
where doc.Element("City").Value == "Chicago"
orderby doc.Element("Initials").Value
select doc;
```

Como se puede observar, al consultar el documento XML con LINQ es conceptualmente parecido a databases relational, DataSets y otros objetos. La diferencia es la estructura del documento XML que se le debe tomar en cuenta, es decir, el diseño jerárquico del documento y el uso de los elementos sobre los atributos.

Un importante diseño de LINQ es la habilidad que tiene para transformar datos a varios formatos. En el mundo de XML, transforma un lugar común dado la necesidad de crear documentos XML así como también traduce desde un esquema a otro. Por

ejemplo: se necesita un nuevo documento XML que contenga los nombres de los doctores, con sus iniciales como un atributo.

▼ Tabla XIV: Creación de un documento XML

```
<?xml version="1.0" standalone="yes"?>
<Doctors>
<Doctor Initials="bb">Boswell, Bryan</Doctor>
<Doctor Initials="lg">Goldstein, Luther</Doctor>
.
.
.
</Doctors>
```

Linq para ADO.NET

Linq para Base de Datos (LINQ to ADO.NET) incluye diferentes implementaciones de LINQ que se necesita para manipular datos relacionales. Dentro de la arquitectura de LINQ to ADO.NET se ha concebido la construcción de tres especializaciones: Linq to SQL, LINQ to Entities y LINQ to DataSets:

Linq to SQL

Linq para Sql (Linq to Sql) es el nombre otorgado a la interfaz IQueryable<T> API esto permite que las consultas LINQ trabajen con las bases de datos de Microsoft SQL Server Esta interfaz anteriormente era conocida como DLinq. Para tomar ventaja de LINQ para SQL, se debe tener una referencia al ensamblado System.Data.Linq.dll en el proyecto y tienen un uso directiva tales como las siguientes: using System.Data.Linq.

Linq to Entities

LINQ para Entidades (LINQ to Entities) es una API alternativa de LINQ que es usada como interface a la base de datos. Esta API desacopla el modelo de objetos

entidades del diseño físico de la base de datos incorporando un mapeo lógico entre ambos. En concreto, si el programa necesita un acoplamiento flexible entre las entidades y el modelo de base de datos, integración de objetos de datos procedentes de múltiples tablas o, más flexibilidad en el modelado de objetos, LINQ para Entidades puede ser la respuesta.

Linq to DataSets

Linq para DataSets(Linq to DataSets), LINQ soporta las consultas sobre DataSets tipados y no tipados. Al seguir con la programación de la aplicación del Hospital, suponga que tiene un *namespace DataSets* con un tipo de datasets SchedulingDocs contiene tres tablas: Doctors, Calls y Vacations. La tabla Doctors contiene un registro de cada doctor, la tabla Calls mantiene un registro de que doctor ha sido llamado cada dia, y la tabla Vacations anota las solicitudes de vacaciones. El DataSets se resume en la siguiente figura:

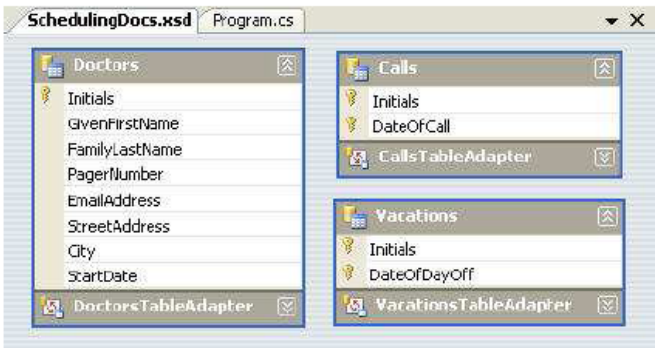


Figura 2.4: Tablas dentro del DataSET tipado en SchedulingDocs

Permite asumir una instancia de **SchedulingDocs** que puede ser creada y llenada; también para encontrar los doctores que solo viven en chicago, la consulta es exactamente como se ve a continuación:

▼ **Tabla XV: Creación de un DataSet y Búsqueda de un registro**

```
DataSets.SchedulingDocs ds = new DataSets.SchedulingDocs(); //
create dataset
.
. // open connection to a database and fill each table?
.
// Busqueda para encontar los doctors que viven en Chicago
var chicago = from d in ds.Doctors
where d.City == "Chicago"
```

LINQ no es sólo para consultas

Inicialmente, se podría entender que LINQ es sólo para consultas o una capa de acceso a datos, ya que significa Lenguaje Integrado de Consultas, pero, LINQ no se enfoca solamente a este contexto. Su poder trasciende a mucho más. Es preferible pensar en LINQ como un motor de iteración de datos, donde los resultados de la invocación de un método se almacenan dentro de una estructura, y éstos se necesitan convertir a otra estructura antes de invocar a otro método. Por ejemplo: Se tiene un método A y el método A retorna un arreglo de strings que contienen valores numéricos almacenados como cadenas. Se necesita llamar el método B, pero el método B requiere un arreglo de enteros. Normalmente se escribiría un bucle que recorra cada item del array de strings convirtiéndolo a entero, llenando una nueva estructura de arreglo de enteros para pasar al método B. Con LINQ el pequeño problema se resolvería con el siguiente algoritmo:

▼ **Tabla XVI: Conversión de un Array de Strings a Enteros con LINQ**

```
string[] numbers = { "0042", "010", "9", "27" };
int[] nums = numbers.Select(s => Int32.Parse(s)).ToArray();

foreach (int num in nums)

Console.WriteLine(num);
```

Eso es todo. Aquí está la salida muestra los enteros:

▼ Tabla XVII: Conversión de un Array de Strings a números Enteros y su Clasificación

```
42
10
9
27
```

▼ Tabla XVIII: Conversión un Array de Strings a números Enteros y su Clasificación

```
string[] numbers = { "0042", "010", "9", "27" };
int[] nums = numbers.Select(s => Int32.Parse(s)).OrderBy(s =>
s).ToArray();
foreach(int num in nums)
```

▼ Tabla XIX: Aquí está el resultado:

```
9
10
27
42
```

Ahora se hará un ejemplo más complejo. Suponga que cuenta con algún código común que contiene una clase Empleados. Esta clase Empleados cuenta con un método para devolver todos los empleados. También asuma que se tiene otro código base que contiene una clase Contactos, y en esa clase cuenta con un método para publicar contactos. Ahora, que tiene la necesidad de publicar todos los empleados como contactos.

La tarea parece bastante simple, pero hay una cosa. El método empleado común que recupera los empleados y devuelve a los empleados en un ArrayList de objetos

empleados, y el método de contacto publica los contactos que requiere una variedad de tipo de contacto. Aquí el código:

▼ Tabla XX: Método que devuelve empleados en un ArrayList

```
namespace LINQDev.HR
{
public class Employee
{
public int id;
public string firstName;
public string lastName;
public static ArrayList GetEmployees()
{

// Of course the real code would probably be making a database query

// right about here.

ArrayList al = new ArrayList();

// Man, do the new C# object initialization features make this a snap.
al.Add(new Employee { id = 1, firstName = "Joe", lastName = "Rattz"} );
al.Add(new Employee { id = 2, firstName = "William", lastName = "Gates"} );
al.Add(new Employee { id = 3, firstName = "Anders", lastName = "Hejlsberg"} );
return(al);
}
}
}
namespace LINQDev.Common
{
public class Contact
{
public int Id;
public string Name;
public static void PublishContacts(Contact[] contacts)
{
// This publish method just writes them to the console window.
foreach(Contact c in contacts)
Console.WriteLine("Contact Id: {0} Contact: {1}", c.Id, c.Name);
}
}
}
```

Como se puede observar, la clase Employee y el método GetEmployees se encuentran en el espacio de nombres LINQDev.HR, y GetEmployees devuelve un ArrayList. El método PublishContacts de la clase Contacts se encuentran en el nombre de espacio LINQDev.Common, y requiere una serie de objetos de contacto para ser invocado. Anteriormente, esto se podría resolver iterando a través de la ArrayList devuelto por el método GetEmployees y la creación de una nueva gama de tipos de contacto que se pasa al método PublishContacts. LINQ hace que la tarea sea fácil, como se muestra:

▼ Tabla XXI: Llamada al Código Común

```
ArrayList alEmployees = LINQDev.HR.Employee.GetEmployees();
LINQDev.Common.Contact[] contacts = alEmployees
.Cast<LINQDev.HR.Employee>()
.Select(e => new LINQDev.Common.Contact {
Id = e.id,
Name = string.Format("{0} {1}", e.firstName, e.lastName)
})
.ToArray<LINQDev.Common.Contact>();
LINQDev.Common.Contact.PublishContacts(contacts);
```

Para convertir un ArrayList de objetos Empleado a un arreglo de objetos Contacto, se debe convertir el arreglo de Empleados a una secuencia de tipo IEnumerable<T> usando el operador de consulta estándar 'Cast'. Esto es necesario debido a la legalidad de la clase ArrayList usada. Sintácticamente hablando, lo que se almacena en un ArrayList son objetos del tipo System.Object y no objetos de tipo Empleado. Entonces, es necesario convertirlos a objetos Empleado. Se tenía también el método GetEmployees que retorna un lista genérica, pero este no ha sido necesario. Sin embargo, el tipo de colección no será disponible cuando la legalidad del código sea escrita.

A continuación, se hace un llamamiento al metodo Select este regresa una secuencia de los empleados y los objetos en la expresión lambda, el código aprobado en el interior de la llamada al método Select, instancia e inicializa un contacto, la instanciación de un nuevo objeto utilizando las nuevas características el nuevo C # 3.0 que permite asignar los valores en la construcción misma del elemento contacto.

Por último, se convierte la secuencia de Contactos a un arreglo de objetos, que es el tipo de entrada para el método PublishContacts, utilizando el operador ToArray. Aquí se muestran los resultados:

▼ Tabla XXII: Ejecución del Método Public Contacts

```
Contact Id: 1 Contact: Joe Rattz
Contact Id: 2 Contact: William Gates
Contact Id: 3 Contact: Anders Hejlsberg
```

LINQ puede hacer mucho, además de sólo consultar datos. Linq tiene muchas características las mismas que han mencionado anteriormente.

2.4.2 Arquitectura

La arquitectura de LINQ está diseñada alrededor de una abstracción de objetos de alto nivel. Esto significa que las consultas LINQ son escritas en C 3.0 contra la noción de una colección de objetos y son traducidas por el compilador IL (Lenguaje Intermedio) para la ejecución contra una fuente de datos particular. Esto provee muchos beneficios, incluyendo:

- Capacidad de apuntar a diferentes tipos de datos
- Capacidad de usar LINQ con los objetos existentes de .Net y 2.0

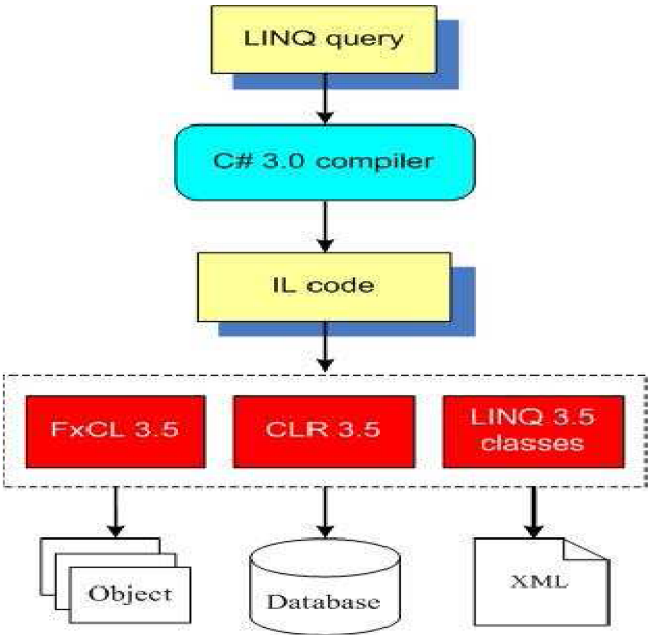


Figura 2.5: Arquitectura de LINQ

2.4.2.1 Árboles de Expresiones (Expressions Trees)

Un árbol de expresión es una eficiente representación de datos en forma de árbol de una consulta correspondiente a la ejecución de un operador o expresión Lambda. Estas representaciones de datos pueden ser evaluadas todas al mismo tiempo de modo que una única consulta se puede construir y ejecutar contra una fuente de datos, tales como una base de datos.

Los árboles de expresiones representan el código de nivel del lenguaje en forma de datos. Los datos se almacenan en una estructura con forma de árbol. Cada nodo del árbol de expresión representa una expresión, por ejemplo, una llamada al método o una operación binaria, como x < y.

En la ilustración siguiente se muestra un ejemplo de una expresión y su representación en forma de un árbol de expresión. Las diferentes partes de la expresión tienen un color distinto para hacerlas coincidir con el nodo correspondiente

del árbol de expresión. También se muestran los diferentes tipos de los nodos del árbol de expresión.

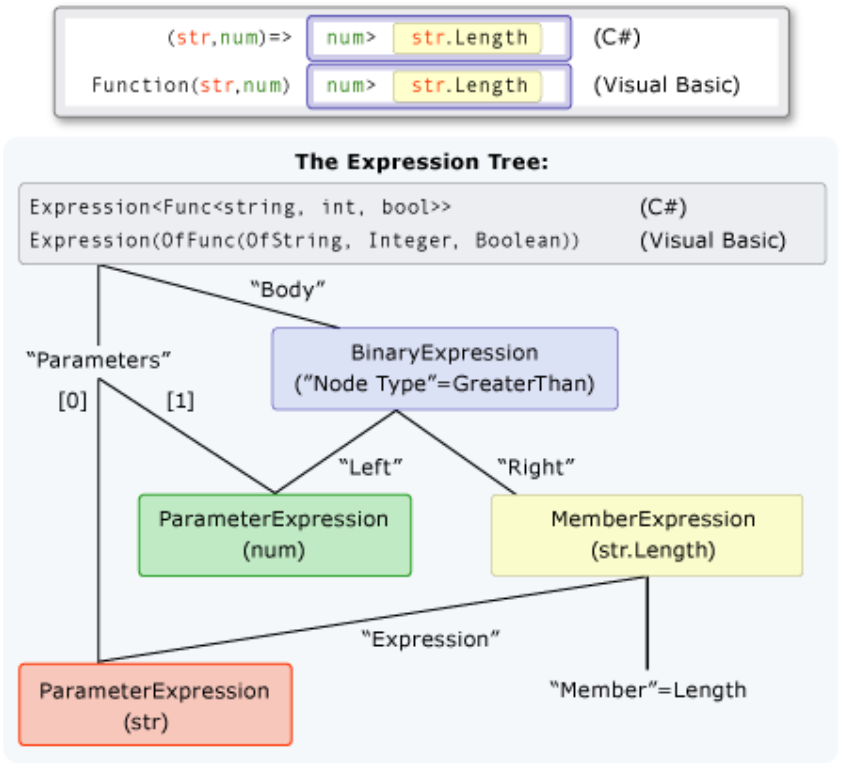


Figura 2.6: Arboles de expresiones Lambda

En el ejemplo de código siguiente se muestra cómo el árbol de expresión que representa la expresión lambda num => num < 5 (C#) o Function(num) num < 5 se puede descomponer en partes.

▼ **Tabla XXIII: Representación de algunas Expresiones Lambda**

```
C#
// Add the following using directive to your code file:

// using System.Linq.Expressions;


//Create an expression tree.

Expression<Func<int,bool>>exprTree=num=>num<5;


//Decompose the expression tree.

ParameterExpression param = (PaqrameterExpression)exprTree.Parameters[0]);
BinaryExpressionoperation = (BinaryExpression)exprTree.Body;
ParameterExpression left = (ParameterExpression)operation.Left;
ConstantExpression ringht =(ConstantExpression)operation.Right;



Console.WriteLine("Decomposed expression:{0}=>{1}{2}{3}"),
          Param.Name, left.Name, operation.NodeType,right.Value);


/* This code produces the following output:



   Decomposed expression: num=>num LessThan 5
*/
```

Generar árboles de expresiones

El espacio de nombres System.Linq.Expressions proporciona una API para la compilación manual de árboles de expresiones. La clase Expression contiene métodos de generador estáticos que crean nodos del árbol de expresión de tipos específicos, por ejemplo, un objeto ParameterExpression, que representa una expresión de parámetro con nombre, o un objeto, MethodCallExpression, que representa una llamada a un método. ParameterExpression, MethodCallExpression y los demás tipos de árboles de expresiones específicos de la expresión se definen

también en el espacio de nombres System.Linq.Expressions. Estos tipos se derivan del tipo abstracto Expression.

El compilador también puede generar un árbol de expresión. Un árbol de expresión generado por el compilador siempre tiene como raíz un nodo de tipo Expression<(Of <(TDelegate>)>); es decir, su nodo raíz representa una expresión lambda.

▼ Tabla XXIV: Generación de arboles de expresiones

```
C#

//add the following using directive to your code file:
//using system.Linq.Expressions;


//Manually build the expression tree for
//the lambda expression num => < 5.


ParameterExpression numParam = Expression.Parameter(typeof(int),"num");
ConstanExpressionfive =Expression.Constan(5,typeof(int));
BinaryExpression numLessThanFive = Expression.LessThan(numParam,five);
Expressiom<func<int,bool>>lambda1=
        Expression.Lambda<func<int,bool>>(
                numLessthanfive,
                new ParameterExpression[]{numParam});

//Let the compiler generate the expression tree for
//the lambda expression num => <5.

Expression<Func<int,bool>>lambda2=num=>num<5
```

En el ejemplo de código siguiente se muestran dos mecanismos para crear un árbol de expresión que representa la expresión lambda num => num < 5 (C#) o Function(num) num < 5.

Cuando una expresión lambda está asignada a una variable de tipo Expression<(Of <(TDelegate>)>), el compilador emite un árbol de expresión que representa la expresión lambda. Por ejemplo, algunos métodos de operador de consulta estándar que se definen en la clase Queryable tienen parámetros de tipo Expression<(Of <(TDelegate>)>). Al llamar a estos métodos, puede pasar una expresión lambda y el compilador generará un árbol de expresión.

El tipo Expression<(Of <(TDelegate>)>) proporciona el método Compile, que compila el código representado por el árbol de expresión en un delegado ejecutable. Este código ejecutable es equivalente al código ejecutable que se habría generado si originalmente la expresión lambda hubiera estado asignada a un tipo de delegado.

2.4.2.2 Lazy Evaluation

La Evaluación Perezosa (***Lazy Evaluation***) o Evaluación Retardada (***Delayed Evaluation***) es una técnica de programación que consiste en demorar el cálculo computacional de un conjunto de valores hasta que los resultados del cálculo sean necesarios. La idea principal es suspender la evaluación de algún elemento o alguna función hasta que los resultados de estos elementos se utilicen más adelante en el algoritmo.

Dentro de LINQ, la evaluación perezosa es una característica heredada particularmente de lenguajes funcionales. Cuando se usa Lazy Evaluation, una expresión no es atendida al mismo instante en el que se enlaza dicho resultado a una variable, sino más tarde cuando un evaluador obliga a obtener el resultado de una expresión y alojarla en otro espacio de memoria. Observe el siguiente el ejemplo:

▼ Tabla XXV: Uso de Lazy Evaluation

```
//Definiendo el arreglo de cadenas
string[] _nombres = { "Edison", "Daniel", "Edgar",
"Esperanza", "Johnny", "Jorge","Omar", "Olimpia", "Charito"};

//Se filtran la cadena que sea igual a "Olimpia"
string nombreDiosaOlimpo = _nombres.Where(n => n ==
"Olimpia");

//Hasta este momento la variable "nombreDiosaOlimpo" es un
objeto //diferido que representa una cadena, pero no contiene
la cadena en //sí. La consulta no se ha ejecutado aún.


//Al momento de hacer referencia a la variable
"nombreDiosaOlimpo", convertir el valor a mayúscula y
asignarla a otra variable es cuando verdaderamente se evalúa
la consulta y se obtiene el valor de "nombreDiosaOlimpo".
string nombreEnMayusculas = nombreDiosaOlimpo.ToUpper();
```

En la sentencia ***nombreEnMayusculas*** = ***nombreDiosaOlimpo.****ToUpper()* claramente se llama a la expresión `nombreDiosaOlimpo`, obligando de esta forma la evaluación de la operación que representa y asignando el resultado de la consulta a una nueva variable `nombreEnMayusculas`. Sin embargo, hasta este punto, el valor de la variable `nombreEnMayusculas` es irrelevante hasta que sea referenciado más tarde en otra expresión, haciendo la evaluación diferida.

La evaluación Perezosa otorga grandes ventajas a LINQ, entre ellas: incrementa el desempeño debido a que evita cálculos innecesarios, evita condiciones de error en la evaluación de expresiones compuestas, la posibilidad de construir estructuras de datos infinitas y hacer cálculos sobre ellas sin tener que implementar bucles infinitos, y la capacidad de definir estructuras de control como funciones regulares en lugar de construirlas como tipos primitivos. Adicionalmente, mediante la evaluación perezosa es posible utilizar paso de valores por nombre, recordando los valores de los

argumentos ya calculados para evitar recalcularlos. A esta estrategia se denomina "***Paso de Parámetros por Necesidad***" o ***Call By Need***.

2.4.2.3 Tiempo de Evaluación de Consultas (Query Evaluation Time)

LINQ usa un esquema de evaluación de expresiones y consultas basado en tres características fundamentales para el tratamiento de la información: Expresiones Lambda, Árboles de expresión y Lazy Evaluation.

Los operadores de consulta permiten al desarrollador ejecutar operaciones de filtrado, proyección o extracción de datos. La construcción de consultas se facilita bajo el concepto de Expresiones Lambda, que proporcionan una conveniente forma de escribir funciones que manejan argumentos para su subsecuente evaluación. Las expresiones Lambda poseen la ventaja de proporcionar una sintaxis más directa y compacta de redacción, pero más importante; las expresiones Lambda pueden ser compiladas como código o datos, los cuales permiten a las expresiones lambda ser procesadas en tiempo de ejecución por optimizadores, traductores y evaluadores.

El namespace System.Linq.Expressions define un distinguido tipo genérico denominado Expression<T>, el cual indica que un 'árbol de expresión ha sido creado a partir de una expresión lambda, en lugar del tradicional método basado en IL. Los arboles de expresión son eficientes representaciones de datos en memoria de las expresiones lambda, que construyen una estructura de expresión transparente y explicita.

2.4.2.4 Fundamentos de la Sintaxis de LINQ: Sentencias y Operadores

Aquí se analiza el nivel de consulta de los operadores de consulta. Estos operadores contienen la sintaxis básica tanto de C# como en Visual Basic.

A continuación están los operadores utilizados con mayor frecuencia, tienen un dedicado lenguaje y sintaxis con palabras clave, razón por la cual son usados como parte de expresiones de consultas.

▼ Tabla XXVI: Operadores de Consulta Estandar LINQ

Lista de Operadores de Consulta Estándar LINQ		
Operador	**Diferido**	**Descripción**
Aggregate	No	Aplica una función a una secuencia , manteniendo un unico valor.
All	No	Aplica una función a una secuencia para ver si todos los elementos satisfacen la función
Any	No	Aplica una función a una secuencia para ver si solo un elemento satisface la función
Average	No	Calcula el promedio de una secuencia de números.
Cast	Si	Transforma una secuencia de elementos de un tipo a otro.
Concat	Si	Concatena dos secuencias S1 y S2
Contains	No	Busca dentro de una secuencia para ver si esta contiene un elemento dado.
Count	No	Cuenta el número de elementos de una secuencia y devuelve como resultado un entero.
DefaultIfEmpty	Si	Dado una secuencia S, retorna S . Si la secuencia no es vacía o el valor predeterminado en caso de que la secuencia sea vacía.
Distinct	Si	Retorna una secuencia sin valores duplicados.
ElementAt	No	Retorna el elemento de la secuencia de la posición especificada.
ElementAtOrDefault	No	Retorna el elemento de la secuencia de la posición especificada, o el valor predeterminado si la secuencia es vacía.
Empty	Si	Retorna una secuencia vacía.

EqualAll	No	Compara dos secuencias que sean iguales.
Except	Si	Dado dos secuencias S1 y S2, retorna el conjunto de la diferencia entre S1 – S2.
First	No	Retorna el primer elemento de la secuencia.
FirstOrDefault	No	Retorna el primer elemento de la secuencia, o el valor predeterminado de la secuencia si esta es vacia.
GroupBy	Si	Agrupa los elementos de la secuencia por clave.
GroupJoin	Si	Une dos secuencias S1y S2, y agrupa jerárquicamente los elementos de las mismas.
Intersect	Si	Dado dos secuencias S1 y S2, retorna el conjunto de la intersección entre S1 y S2.
Join	Si	Ejecuta un tradicional xjoin o producto cruz de dos secuencias dadas S1y S2.
Last	No	Retorna el último elemento de una secuencia dada.
LastOrDefault	No	Retorna el último elemento de una secuencia dada, o el valor predeterminado si la secuencia es vacía.
LongCount	No	Cuenta el número de elementos de la secuencia , retorna como resultado un entero largo.
Max	No	Retorna el número máximo de una secuencia.
Min	No	Retorna el número mínimo de una secuencia
OfType	Si	Retorna los elementos de una secuencia que correspondan a un tipo dado.
OrderBy	Si	Ordena una secuencia de elementos por clave en orden ascendente.
OrderByDescending	Si	Ordena una secuencia de elementos por clave en orden descendente.
Range	Si	Retorna una secuencia de enteros de un rango dado.
Repeat	Si	Retorna una secuencia de valores repitiendo un valor dado n veces.
Reverse	Si	Invierte los elementos de una secuencia.
Select	Si	Aplica una función de proyección a una secuencia, retornando una nueva secuencia.
SelectMany	Si	Aplica una función de proyección para seleccionar la secuencia principal de las secuencias dadas.
Single	No	Retorna un elemento único de una secuencia.
SingleOrDefault	No	Retorna un elemento único de una secuencia, o el valor predeterminado si la secuencia es vacía.
Skip	Si	Salta el primer elemento de n elementos de una secuencia, , retorna el elemento actual.
SkipWhile	Si	Dada una function F y una secuencia S, se mueve al elemto inicial de S si F es verdadero (true).

CAPÍTULO III

LINQ PARA LA MANIPULACIÓN DE OBJECTOS - LINQ TO OBJECTS

3.1 INTRODUCCIÓN

Uno de los pilares de la programación y por tanto una de las tareas más comunes, es la manipulación de objetos. Para que la información pueda ser procesada a través de un conjunto de rutinas de programación, ésta debe representada adecuadamente en estructuras y objetos cuyo estado es modificado en función de obtener el resultado deseado a partir de los requerimientos del problema a resolver.

Otra de las ventajas de LINQ, es la forma con la que se integra sutilmente al lenguaje de programación .NET elegido, C# o Visual Basic. En lugar de tener un nuevo conjunto de clases necesarias para acceder los beneficios de LINQ, es posible utilizar las mismas colecciones, arreglos y estructuras dentro de las clases pre-existentes, lo que significa que se puede explotar las ventajas de las consultas LINQ con pequeñas modificaciones en el código existente.

En este capítulo se describen los operadores y mecanismos en los cuales se basa la funcionalidad de LINQ para el manejo de objetos de programación.

3.2 LA VISIÓN DE LINQ PARA OBJECTOS

LINQ se enfoca principalmente en la creación y ejecución de consultas, que pueden devolver un conjunto de objetos, un solo objeto, o un subconjunto de campos de un objeto. En LINQ, a este conjunto de objetos de retorno se los llama **Secuencia.** La mayoría de secuencias en LINQ son de tipo **IEnumerable<T>**, donde **T** es el tipo de dato de los objetos almacenados en la secuencia. Por ejemplo, una secuencia de enteros se puede almacenar en una variable de tipo IEnumerable<int>. Con el uso de LINQ, se podrá observar el uso frecuente del objeto IEnumerable<T>, ya que muchos de los operadores de consulta estándar retornan un objeto secuencia. La funcionalidad de LINQ To Objects se basa en la implementación de la interfaz `IEnumerable<T>`, de secuencias y un conjunto de operadores de consulta estándar; los mismos que se explican a continuación.

3.2.1 IEnumerable<T>, Secuencias, y Operadores de Consulta Estándar

`IEnumerable<T>` *(cuya lectura es "I enumerable de T")*, es una interfaz que implementa toda clase que representa una colección genérica de objetos. Una ***secuencia*** es un término lógico para enunciar una colección que implementa la interfaz `IEnumerable`. Cuando se tiene una variable del tipo `IEnumerable<T>`, significa que dicha variable alberga una ***"secuencia"*** de objetos de tipo `T`, siendo `T` un tipo genérico que puede guardar cualquier tipo de dato.

LINQ se basa en un conjunto de Operadores de Consulta, definidos como métodos de extensión (de la clase estática `System.Linq.Enumerable`), y que trabajan principalmente sobre cualquier objeto que implemente la interfaz `IEnumerable<T>`. Esta característica hace que LINQ se convierta en un framework de consulta de

propósito general, ya que se puede extender la funcionalidad de los operadores estándar a cualquier colección de objetos. Para las colecciones no genéricas de C#, es posible ejecutar consultas LINQ usando *"unboxing"* o casteo de tipos.

De la misma manera, la infraestructura de consulta es también muy extensible, ya que los desarrolladores pueden especializar el comportamiento de un método para un tipo de dato particular, otorgando la posibilidad de ejecutar consultas complejas de datos sobre cualquier secuencia.

3.2.2 Retornando IEnumerable<T>, Yielding y Consultas Diferidas

Es importante recordar que mientras muchos de los Operadores de Consulta Estándar son prototipos que retornan un `IEnumerable<T>` que a su vez representan una secuencia, ésta en realidad no es una secuencia de objetos que se obtiene cuando el operador es invocado. En su lugar, los operadores retornan un único objeto diferido del enumerado, que al momento de referenciarse en otra sección del código, retornan en cada instante un elemento de la secuencia. Durante la enumeración de la secuencia a retornar es cuando la consulta verdaderamente se ejecuta. Uno a uno, se toma cada elemento de la secuencia y el elemento actual es ***"diferido"***, aplazando la ejecución del resto de la secuencia. El elemento actualmente diferido será retornado como salida de la secuencia. De esta forma, se dice que la consulta ha sido diferida.

▼ Tabla XXVII: Ejemplo. Lista de cadenas

```
//Definiendo el arreglo de cadenas
string[] _nombres = { "Edison", "Daniel", "Edgar",
"Esperanza", "Johnny", "Jorge","Omar", "Olimpia", "Charito"};

//Se filtran las cadenas que comienzan con la letra "E"
IEnumerable<string> expresion = _nombres.Where(n =>
n.StartsWith("E"));

//Hasta este momento la variable "expresion" es un objeto
diferido //que representa una secuencia y no una secuencia en
sí. La consulta //Where no se ha ejecutado aún.


//Al momento de hacer referencia a la variable "expresion" y
comenzar //a enumerarlo (recorrer sus elementos), es cuando
verdaderamente la //consulta Where se ejecuta.
//Como la ejecución del operador no ha sido inmediata, es que
se //llama al operador Where, como operador Dieferido o
Consulta //Diferida
 foreach (string elemento in expresion)
    Console.WriteLine(elemento);
```

Hasta este momento antes del bucle *foreach*, la variable *"expresion"* es un objeto diferido que representa una secuencia y no una secuencia en sí. En este instante el operador *Where* no se ha ejecutado aún. Más abajo en el código, al momento de hacer referencia a la variable *"expresion"* dentro del bucle *foreach* y comenzar a enumerarlo, es decir al comenzar a recorrer sus elementos, es cuando verdaderamente la consulta se ejecuta. Como la ejecución de la consulta no ha sido inmediata, es que se llama al operador *Where* como ***Operador Dieferido***.

Para crear consultas diferidas más fácilmente, se ha incorporado la palabra clave ***yield*** a los lenguajes de programación.

Las consultas diferidas pueden ser muy convenientes en varios escenarios; por ejemplo cuando se ejecuta una consulta que exige el consumo de cierta cantidad de recursos y el resultado de esta consulta se lo albergaba en una variable en memoria; independientemente de que si se hiciera referencia o no a esta variable en el código

posterior, la consulta se ejecuta de todas formas. Si por algún descuido en el código, no se hace referencia a dicho resultado, se habrán utilizado recursos en vano en la ejecución de una consulta que no se necesita. Otro escenario común, es cuando se desea obtener una cantidad determinada de elementos de una secuencia infinita; se puede hacer la consulta y traer los elementos necesarios, obviamente sin tener que esperar que se termine de obtener el último elemento.

Sin embargo, al no ejecutarse la consulta de forma inmediata, es posible que un error pueda ocurrir en la consulta y no ser detectado hasta que la enumeración tome lugar. Esto puede convertir a una consulta diferida en un beneficio, o en un dolor de cabeza. Si se desea obligar a la ejecución inmediata de un operador diferido, existen operadores de conversión que no retornan un `IEnumerable<T>` y que crean diferentes estructuras de datos en memoria donde se alojan los resultados que no cambiarán si la fuente de dichos datos cambia. Algunos de estos operadores de conversión son: `ToArray`, `ToList`, `ToDictionary`, `ToLookup`; entre otros que se estudiarán en secciones más adelante.

3.2.3 Delegados a Funciones

Muchos de los Operadores de Consulta Estándar de LINQ están diseñados a tomar un Delegado a Función como argumento, en otras palabras, pueden recibir una función o método como parámetro. Un delegado es un puntero a una función, es decir un tipo que hace referencia a un método. Cuando se asigna un método a un delegado, éste se comporta exactamente como el método. El método delegado se puede utilizar como cualquier otro método, con parámetros y un valor devuelto. A continuación alguna de las formas de declarar delegados a funciones:

▼ Tabla XXVIII: Ejemplo. Declaración de Delegados a Funciones

```
public delegate TR Func<TR>();
public delegate TR Func<T0, TR>(T0 a0);
public delegate TR Func<T0, T1, TR>(T0 a0, T1 a1);
public delegate TR Func<T0, T1, T2, TR>(T0 a0, T1 a1, T2
a2);
public delegate TR Func<T0, T1, T2, T3, TR>(T0 a0, T1 a1, T2
a2, T3 a3);
```

Cada declaración, `TR` se refiere al tipo de dato retornado, que también es colocado al final de la lista de parámetros en cada sobrecarga. `T0`, `T1`, `T2`, y `T3` corresponden a parámetros de entrada pasados al método. Las múltiples declaraciones existen debido a que algunos Operadores de Consulta Estándar poseen argumentos de delegados que requieren más parámetros que otros. A continuación la declaración del operador estándar de consulta `Where`:

▼ Tabla XXIX: Ejemplo. Declaración Operador de Consulta Where

```
public static IEnumerable<T> Where<T>(this IEnumerable<T>
source,
Func<T, bool> predicate);
```

El argument `predicate` especifica un delegado a función o expresión Lambda que recibe un parámetro `T` y retorna un valor booleano: **`Func<T, bool>`**. La idea es que una variable de este tipo: función o expresión Lambda, reciba un valor `T`, y retorne un bool a partir de este valor `T`. La razón por la que se deba colocar el tipo de retorno al final de la lista de parámetros en el delegado indica el tipo de dato que deberá retornar la función resultado del procesamiento del parámetro `T`. A continuación un ejemplo de la creación de un delegado simple y su uso:

▼ Tabla XXX: Ejemplo. Creación de un Delegado simple y su uso

```
//Definiendo el arreglo de enteros
int[] _numeros = new int []{1, 2, 3, 4, 5, 6, 7, 8, 9, 0};

//Declarando el Delegado
Func<int, bool> EsMayorQueDos = i => i > 2;

//Ejecutando la consulta
IEnumerable<int> enterosMayoresQueDos =
_numeros.Where(EsMayorQueDos);


//Mostrando los datos resultado
 foreach (int elemento in enterosMayoresQueDos)
    Console.WriteLine(elemento);
```

LINQ PARA OBJETOS EN EL CODIGO

Linq para objetos permite a los desarrolladores de .NET escribir consultas sobre colecciones de objetos, Ya fuera de la caja, Linq posee un gran conjunto de operadores de consulta que proveen una funcionalidad similar a la esperada con cualquier lenguaje SQL trabajando con datos relacionales.

Tradicionalmente, al trabajar con colecciones de objetos significa que se debe escribir varias rutinas de barrido usando bucles o foreach para iterar a través de una lista de valores, filtrando usando sentencias if, y algunas acciones como mantener una variable total donde se irá sumando los valores cada vez. LINQ libera al desarrollador de tener que escribir código de iteración, permitiéndole escribir consultas que filtran una lista o hacer cálculos mediante funciones de agregación sobre los elementos.

Se puede escribir consultas solo a través de colecciones de tipos llamados IEnumerble(y también a través de una nueva interfaz llamada IQueryable). Esto es casi cualquier tipo de colección construida dentro de las librerías de clases .NET,

incluyendo arreglos como string[], o int[] y cualquier colecciones de tipo List<T> que se haya definido. Revise los siguientes ejemplos simples, ya que contienen una estructura básica.

▼ Tabla XXXI: Ejemplo. Linq en el código

```
int[] nums = new int[] {0,4,2,6,3,8,3,1};


var result = from n in nums
             where n < 5
             orderby n
             select n;


foreach(int i in result)
    Console.WriteLine(i);


Output:
0
1
2
3
3
4
```

▼ Tabla XXXII: Ejemplo. Calculo de la suma total de elementos

```
int[] nums = new int[] {0,4,2,6,3,8,3,1};


var result = from n in nums
             where n < 5
             orderby n
             select n;


foreach(int i in result)
    Console.WriteLine(i);


Output:
0
1
2
3
3
4
```

Linq para Objeto se extiende para todo tipo de objetos que vengan de IEnumerable(desde una gran colección de clases en .Net, o hasta un simple arreglo de List<T>)), para soportar los operadores de consulta similares a los disponibles en SQL. Se puede escribir consultas usando algún built-in Estándar Query Operators, o añadiendo nuestros propios operadores si fueran necesarios. Los operadores estándares cubren una amplia variedad de categorías, en la actualidad existen más de secuencia que forman parte de LINQ. Para tener una idea de su alcance, aquí está una lista de los operadores a nuestra disposición.

▼ Tabla XXXIII: Lista de Operadores

Operador Tipo	Operador Nombre
Aggregate	Aggregate, Average, Count, LongCount, Max, Min, Sum
Conversion	Cast, OfType, ToArray, ToDictionary, ToList, ToLookup
Element	DefaultIfEmpty, ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault
Equality	EqualAll
Generation	Empty, Range, Repeat
Grouping	GroupBy
Joining	GroupJoin, Join
Ordering	OrderBy, ThenBy, OrderByDescending, ThenByDescending, Reverse
Partitioning	Skip, SkipWhile, Take, TakeWhile
Quantifiers	All, Any, Contains
Restriction	Where
Selection	Select, SelectMany
Set	Concat, Distinct, Except, Intersect, Union

La mayoría de los operadores se le harán familiares si alguna vez trabajo con una Base Datos relacional. Una distinción importante entre la escritura y las consultas en SQL es que el orden de los operadores se invierten si usted ha usado Select-From-

Where-OrderBy, podría pasar un tiempo hasta habituarse a utilizar de la siguiente forma From-Where-OrderBy-Select.

▼ Tabla XXXIV: Ejemplo. Orden de los Operadores en Linq

```
List<Contacts> contacts = Contacts.SampleData();
 var q = from c in contacts
       where c.DateOfBirth.AddYears(35) > DateTime.Now
       orderby c.DateOfBirth descending
       select c.FirstName + " " + c.LastName +
            " b." + c.DateOfBirth.ToString("dd-MMM-yyyy");

foreach(string s in q)
   Console.WriteLine(s);
 Output:
Mack Kamph b.17-Sep-1977
Armando Valdes b.09-Dec-1973
```

El ejemplo anterior muestra cómo obtener una lista de contactos que están a menos de 35 años de edad clasificados en orden decreciente según la edad. Esta consulta crea una lista de formato de cadenas como el resultado, pero cualquier tipo pueden ser devueltos, incluso un tipo anónimo..

▼ Tabla XXIV: Ejemplo. Agrupación de código que permite crear un sub-conjunto de elementos sobre la base de un valor con el grupo por construir.

```
List<Contacts> contacts = Contacts.SampleData();
var q = from c in contacts
        group c by c.State;
 foreach(var group in q) {
    Console.WriteLine("State: " + group.Key);
        foreach(Contacts c in group)
            Console.WriteLine("  {0} {1}", c.FirstName,
c.LastName);
}
Output:
State: CA
  Barney Gottshall
  Jeffery Deane
State: WA
  Armando Valdes
  Stewart Kagel
  Chance Lard
State: AK
  Adam Gauwain
State: FL
  Collin Zeeman
State: TX
  Blaine Reifsteck
  Mack Kamph
State: OR
```

Un aspecto clave de acceso a datos relacionales es el concepto de adhesión (joining). El lenguaje SQL tienen la capacidad de unirse a consultas escritas a través de de normalización de datos, esto permite que los datos no se repita, separando los datos a través de múltiples tablas que se unen por un valor en común. LINQ le permite unirse a múltiples colecciones de objetos usando sintaxis similar a SQL. A continuación se muestran los datos de los contactos y los registros de las llamadas.

▼ Tabla XXXVI: Resultado de la Ejecución del Código

Number	Duration(mins)	Incoming	Date	Time
885 983 8858	2	TRUE	7-Aug-2006	8:12
165 737 1656	15	TRUE	7-Aug-2006	9:23
364 202 3644	1	FALSE	7-Aug-2006	10:5
603 303 6030	2	FALSE	7-Aug-2006	10:35
546 607 5462	4	TRUE	7-Aug-2006	11:15
885 983 8858	15	FALSE	7-Aug-2006	13:12
885 983 8858	3	TRUE	7-Aug-2006	13:47
546 607 5462	1	FALSE	7-Aug-2006	20:34
546 607 5462	3	FALSE	8-Aug-2006	10:10
603 303 6030	23	FALSE	8-Aug-2006	10:40
848 553 8487	3	FALSE	8-Aug-2006	14:0
848 553 8487	7	TRUE	8-Aug-2006	14:37
278 918 2789	6	TRUE	8-Aug-2006	15:23
364 202 3644	20	TRUE	8-Aug-2006	17:12

Para unir la información del registro de llamadas y obtener el nombre del contacto que coincide con el número de teléfono es necesaria la siguiente consulta.

▼ Tabla XXXVII: Consulta que permite obtener el nombre del contacto con su respectivo número telefónico.

```
List<Contacts> contacts = Contacts.SampleData();
   List<CallLog> callLog = CallLog.SampleData();


   var q = from call in callLog
           join contact in contacts on call.Number equals contact.Phone
           select new {contact.FirstName, contact.LastName,
                       call.When, call.Duration};

    foreach(var call in q)
       Console.WriteLine("{0} – {1) {2) ({3}min)",
                         call.When.ToString("ddMMM HH:m"),
                         call.FirstName, call.LastName, call.Duration);

Output:
07Aug 08:12 - Barney Gottshall (2min)
07Aug 09:23 - Ariel Hazelgrove (15min)
07Aug 10:5 - Mack Kamph (1min)
07Aug 10:35 - Collin Zeeman (2min)
07Aug 11:15 - Stewart Kagel (4min)
07Aug 13:12 - Barney Gottshall (15min)
07Aug 13:47 - Barney Gottshall (3min)
07Aug 20:34 - Stewart Kagel (1min)
08Aug 10:10 - Stewart Kagel (3min)
08Aug 10:40 - Collin Zeeman (23min)
08Aug 14:0 - Armando Valdes (3min)
08Aug 14:37 - Armando Valdes (7min)
08Aug 15:23 - Chance Lard (6min)
08Aug 17:12 - Mack Kamph (20min)
```

Esta consulta contiene todos los registros con los datos de los contactos vía número telefónica:

Dando un paso mas halla y resumiendo los datos de múltiples colecciones a través de la combinación, de filtros, grupos, uniones y funciones de agregación todos en una expresión de consulta, se puede demostrar el poder de Expresiones de Consulta sobre datos de memoria.

▼ Tabla XXXVIII: Ejemplo. Muestra los registros de llamadas entrantes para cada contacto y la agregación de las estadísticas de las llamadas.

```
List<Contacts> contacts = Contacts.SampleData();
    List<CallLog> callLog = CallLog.SampleData();

    var q = from call in callLog
            where call.Incoming == true
            group call by call.Number into g
            join contact in contacts on g.Key equals
contact.Phone
            orderby contact.FirstName, contact.LastName
            select new { contact.FirstName, contact.LastName,
                         Count = g.Count(),
                         Avg   = g.Average( c => c.Duration
),
                         Total = g.Sum( c => c.Duration )};

    foreach(var call in q)
        Console.WriteLine("{0} {1} - Calls:{2}, Time:{3}mins,
Avg:{4}mins",
            call.FirstName, call.LastName,
            call.Count, call.Total, Math.Round(call.Avg, 2));

Output:
Ariel Hazelgrove - Calls:1, Time:15mins, Avg:15mins
Armando Valdes - Calls:1, Time:7mins, Avg:7mins
Barney Gottshall - Calls:2, Time:5mins, Avg:2.5mins
Chance Lard - Calls:1, Time:6mins, Avg:6mins
Mack Kamph - Calls:1, Time:20mins, Avg:20mins
Stewart Kagel - Calls:1, Time:4mins, Avg:4mins
```

Este ejemplo muestra operaciones de filtro, ordenamiento, agrupación, unión y selección usando variables de agregación.

CAPÍTULO IV

ESTUDIO COMPARATIVO DE LINQ Y EXPRESIONES LAMBDA FRENTE AL PARADIGMA DE PROGRAMACIÓN IMPERATIVA

4.1 INTRODUCCION

El presente capítulo tiene como objetivo presentar un esquema para la medición y evaluación de aspectos cualitativos y cuantitativos en la comparación de los paradigmas de Programación enunciados en el presente trabajo de Tesis: Programación Imperativa, Programación Funcional usando LINQ y Programación Funcional usando LINQ y Expresiones Lambda.

El estudio se enfoca en principalmente en la identificación de escenarios claves que permita evaluar la robustez, fiabilidad y eficiencia de las técnicas de programación que se examinarán en el presente análisis. Para lograrlo, se plantea un modelo de evaluación que engloba un grupo actividades que van desde la preparación del PC de pruebas, el diseño de los ejercicios, definición de métricas, selección de herramientas, captura y tabulación de resultados e interpretación de los resultados obtenidos.

Al final de este capítulo se dedica una sección en donde se resumen el análisis frente a los datos obtenidos y consideraciones del uso de los paradigmas.

4.2 DEFINICION DEL MODELO DE EVALUACION

El modelo de pruebas utilizado en el presente análisis consiste en la ejecución de seis fases principales:

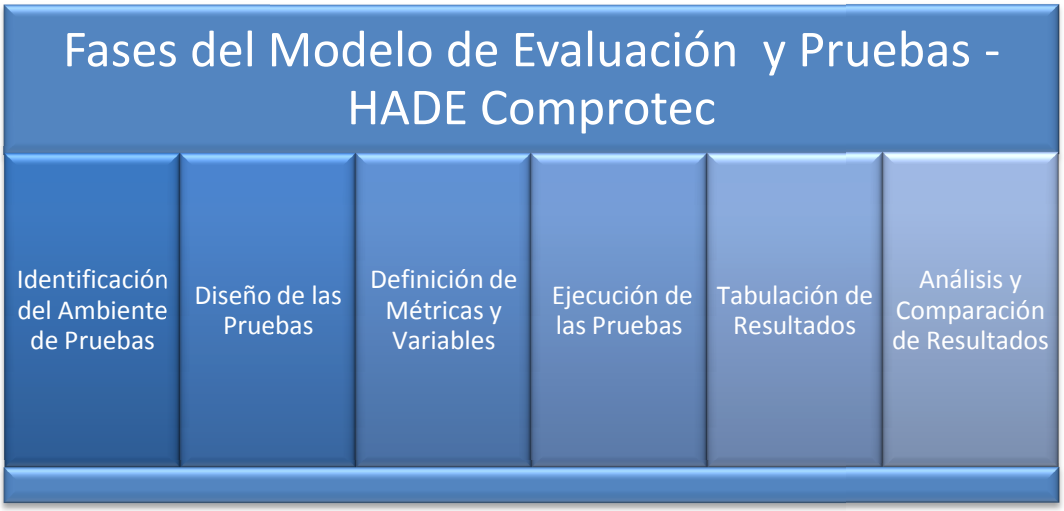


Figura 4.1: Fases del Modelo de Evaluación y Pruebas HADE

FASE 1: Identificación del Ambiente de Pruebas: Consiste principalmente en conocer las características del entorno y condiciones sobre los cuales se ejecutarán las pruebas. Se identifican aspectos que puede intervenir en las pruebas y su mitigación. Adicionalmente se exploran también las características de las herramientas de análisis de aplicaciones que se utilizarán en el estudio y una breve guía o secuencia de pasos que deben seguirse para la ejecución de una prueba y para la captura de datos utilizando las herramientas de evaluación seleccionadas.

FASE 2: Diseño de las Pruebas: las actividades dentro de esta fase se enfocan en la construcción de casos de prueba o ejercicios que se ejecutarán en el ambiente de prueba.

FASE 3: Definición de Métricas y Variables: las actividades dentro de esta fase comprenden la definición de métricas y variables que se usarán en el estudio.

FASE 4: Planeamiento y Ejecución de las Pruebas: las actividades dentro de esta fase tienen como objetivo definir apropiadamente los paquetes de pruebas o grupo de casos de prueba a ejecutarse y la recurrencia en las ejecuciones.

FASE 5: Tabulación de Resultados: durante la ejecución de esta fase se recogen los datos capturados y se presentan en formatos adecuados para su tabulación y posterior interpretación. Incluye gráficos y variables estadísticas.

FASE 6: Análisis y Comparación de Resultados: durante la ejecución de esta fase se enfrentan los datos para obtener criterios acerca del desempeño de las técnicas empleadas.

4.3 FASE 1: IDENTIFICACION DEL AMBIENTE DE PRUEBAS

Durante esta fase se revisarán las características hardware y software y recursos que posee la máquina sobre la que se realizarán las pruebas. Adicionalmente se identificarán aspectos que pueden intervenir y alterar las pruebas y controlarlos en lo posible para obtener datos más reales acerca de la respuesta del PC frente a las pruebas realizadas; y por último se evaluará el uso de herramientas especializadas en la realización de pruebas de desempeño y análisis de aplicaciones .NET,(ver **Anexos)**

4.3.1 CARACTERÍSTICAS DEL PC DE PRUEBAS

A continuación el detalle de las capacidades y recursos con el que cuenta el PC sobre el cual se realizarán las pruebas:

DATOS **generales** DEL PC

- Nombre del Sistema: OLIMPIAMAYORGA
- Fabricante: Sony Corporation
- Modelo: VGN-FE660G
- Tipo: x86

BIOS

- Fabricante: Phoenix Technologies LTD
- Versión: R0130J3
- Fecha: 11/05/3006
- Versión del SMBIOS: 2.40

SISTEMA OPERATIVO

- Nombre: Microsoft Windows XP Professional
- Versión: 5.1.2600 Service Pack 3 (Build 2600)
- Fabricante: Microsoft Corporation

PROCESADOR

- Fabricante: INTEL
- Frecuencia: 1662,308MHz
- Architectura: IA32
- Velocidad del Bus: 400MHz

MEMORIA

- Física Total: 1,024.00 MB
- Virtual Total: 2,00GB
- Espacio de Archivo de Paginación: 2,38 GB

UNIDAD DE DISCO DURO Y PARTICIÓN DE PRUEBA

- Fabricante del Hardware: Unidad Estándar Sony
- Velocidad Lectura Escritura: 7200 rpm
- Sistema de Archivos: NTFS
- Compresión: No
- Número de serie del volumen de Prueba: 0C9945A1
- Tamaño de partición de Prueba: 34,18 GB (36.701.167.104 Bytes)
- Desplazamiento inicial de partición: 32.256 bytes
- Tecnología de Conexión: IDE

SOFTWARE ADICIONAL

- Versión del .NET Framework: 2.0.50727.3053

4.3.2 IDENTIFICACIÓN DE ASPECTOS QUE AFECTAN LA EVALUACIÓN (VARIABILIDAD EXTERNA)

Durante la ejecución de un programa cualquiera en el sistema operativo, se ejecutan de forma clandestina un conjunto de procesos y programas más pequeños que mantienen el funcionamiento básico del propio sistema operativo y de los servicios alojados en el computador.

Comúnmente, el Benchmarking de aplicaciones puede verse afectado -en menor o mayor grado- por estos servicios vivos. Ésta Variabilidad Externa hace que la ejecución de una prueba sea en ocasiones más rápida o más lenta arrojando variación en los resultados en varias ejecuciones de una misma rutina (un buen benchmark debe poseer un Coeficiente de Variación por debajo del 1% - véase sección: 4.5.2.9. Coeficiente de Variación).

Típicamente, los aspectos que mayormente causan variabilidad externa y que afectan las mediciones son:

- **Actividad** en Background no controlada:
  - Aplicaciones de usuario abiertas que poseen tareas de ejecución automática no controlada (por ejemplo: Funcionalidad de Auto Guardado de Microsoft Word);
  - Procesos de usuario que corresponden a programas de usuario instalado y cuya ejecución (continua, ocasional o programada) afecta el desempeño del computador aunque sea por poco tiempo (por ejemplo: software Antivirus y parecidos);
  - Jobs o Tareas automáticas que se ejecutan sobre la instancia de base de datos.
  - Diferencias en la prioridad de ejecución del proceso de prueba entre un ambiente y otro.
- **Interferencias por Dispositivos conectados y Complementos:**
  - Interferencias causadas por paquetes de red entrantes o salientes de dispositivos hardware conectados al PC de prueba que solicitan atención por el procesador (por ejemplo: Impresoras);
  - Sincronización de equipos con un servidor dentro de una red corporativa (por ejemplo: sincronización de fecha y hora local con la del servidor);

- Paquetes de red entrantes provocados por otras fuentes de interrupción (por ejemplo: programas de Chat).

- **Actividad de Red externa:**
  - Solicitudes Web de aplicativos y procesos (por ejemplo: barras y complementos del explorador);
  - Sistema de Actualizaciones Automáticas o sistema de Notificaciones que abren conexiones para la entrega o recepción de datos por internet.
- **Errores Humanos en la toma de datos:**
  - Mediciones no homogéneas en cuanto a los datos de entrada o salida que reciben las rutinas o procesos a evaluar (por ejemplo: una consulta SQL que se ejecuta con el código 1, y una nueva ejecución con el código 10 traen cada una diferente cantidad de datos);
  - Diferencias en la utilización de Unidades de Medida que arrojan las herramientas empleadas;
  - Mediciones del mismo software en ambientes de prueba diferentes;
  - Error en cálculos de las métricas;
  - Diferencias en la configuración de ambiente de prueba.

Para obtener unas pruebas más objetivas y por ende mediciones más precisas, es necesario mitigar en lo posible estos factores con lo que seguramente mejorará el estudio. Es posible que no todos los procesos puedan ser identificados y detenidos, sobre todo los procesos de sistema y sin ellos el computador dejaría de funcionar; sin embargo un correcto mantenimiento y configuración del PC de pruebas garantizan mejores y más precisos resultados.

4.4 FASE 2: DISEÑANDO LAS PRUEBAS

El diseño de las pruebas se enfoca principalmente a la elaboración de casos de prueba o ejercicios que se ejecutarán en el ambiente de pruebas. Estos ejercicios o problemas de programación son diseñados en base a algunos criterios que en sí mismos intentan relacionarse de manera directa con los objetivos de la investigación. Algunos de estos son sin duda: cómo se comporta LINQ para el acceso a datos de orígenes relacionales, el manejo de conexiones, la conversión de datos relacionales a objetos y así por estilo.

Por otro lado, un criterio válido que es necesario tomar en cuenta, es el manejo de recursos (sobre todo la memoria) que posee LINQ en la resolución de problemas sobre Enumerables o secuencias, el alojamiento y compilación de los árboles de expresión generados (véase Capítulo II, sección 2.4.2.1 Árboles de Expresiones), la interpretación de las estructuras, la conversión a objetos, el aporte de la ejecución retardada o Lazy Evaluation (véase Capítulo II, sección 2.4.2.2 Lazy Evaluation) para la evaluación de consultas, entre las características más importantes estudiadas en los capítulos anteriores.

Por el lado de la importancia del negocio, los ejercicios deben englobar algunas de las tareas más comunes de un aplicativo tradicional. Estos deben aproximarse a una lógica de negocio real que brinde un punto de vista más consistente y apropiado, más allá de una demostración con funcionalidad muy básica. De este modo, los ejercicios deben ir escalando de complejidad hasta satisfacer en lo mayormente posible los objetivos del análisis.

4.4.1 Definición de Casos de Prueba

Básicamente se plantean dos grupos de ejercicios: Consultas de Bases de Datos, para probar la funcionalidad de Linq to SQL; y Consultas sobre Colecciones de Objetos,

para probar la funcionalidad de Linq to Objects. La complejidad de los ejercicios es incremental, tratando de abarcar las tareas de programación más comunes dentro de un programa.

Los ejercicios para Consultas de Bases de Datos son:

▼ Tabla XXXIX: Ejercicios realizados para Consultas de Bases de Datos

Nº EJERCICIC	CONSULTA SQL	DESCRIPCIÓN	COMPLEJIDAD
Ejercicio #1	SELECT * FROM PROYECTO	Consulta de datos plana sobre una tabla.	Baja
Ejercicio #2	SELECT * FROM PROYECTO WHERE CodigoArea = 13 and CodigoEstadoProyecto = 1	Consulta de datos plana sobre una tabla con filtros con operador WHERE.	Baja
Ejercicio #3	SELECT dbo.PROYECTO.CodigoProyecto, ISNULL(dbo.PROYECTO.CodigoUNESCO, 'No Asignado') AS CodigoUNESCO, dbo.PROYECTO.Titulo, dbo.AREA.Nombre AS Area, dbo.INSTITUCION.Nombre AS Institucion, dbo.PROYECTO.FechaInicio, dbo.PROYECTO.FechaFin, dbo.PROYECTO.Duracion, dbo.UNIDAD_TRABAJO.Nombre AS UnidadTrabajo, dbo.ESTADOS_PROYECTO.Nombre AS EstadoProyecto, dbo.ESTADOS_PROYECTO.Icono AS IconoEstadoProyecto, dbo.PROYECTO.AvanceGeneral, dbo.PROYECTO.CostoTotal FROM dbo.PROYECTO INNER JOIN dbo.AREA ON dbo.PROYECTO.CodigoArea = dbo.AREA.CodigoArea INNER JOIN dbo.INSTITUCIONES_PARTICIPANTES ON dbo.PROYECTO.CodigoProyecto = dbo.INSTITUCIONES_PARTICIPANTES.CodigoProyecto INNER JOIN dbo.INSTITUCION ON dbo.INSTITUCIONES_PARTICIPANTES.CodigoInstitucion = dbo.INSTITUCION.CodigoInstitucion INNER JOIN dbo.TIPO_PARTICIPACION_INSTITUCION ON dbo.INSTITUCIONES_PARTICIPANTES.CodigoTipoParticipaci dbo.TIPO_PARTICIPACION_INSTITUCION.CodigoTipoParticipacion INNER JOIN dbo.UNIDAD_TRABAJO ON dbo.PROYECTO.CodigoUnidadTrabajo = dbo.UNIDAD_TRABAJO.CodigoUnidadTrabajo INNER JOIN dbo.ESTADOS_PROYECTO ON dbo.PROYECTO.CodigoEstadoProyecto = dbo.ESTADOS_PROYECTO.CodigoEstadoProyecto WHERE (dbo.TIPO_PARTICIPACION_INSTITUCION.CodigoTipoParticipacion = 1)	Consulta de datos de múltiples tablas usando operador INNER JOIN y filtros con operador WHERE.	Media

Los ejercicios para **Consultas sobre Colecciones de Objetos** son:

▼ **Tabla XL: Ejercicios realizados para Consultas sobre Colecciones de Objetos**

Nº EJERCICIO	TIPO DE CONSULTA	COMPLEJIDAD
Ejercicio #1	Ordenamiento Ascendente de Arreglo de Enteros	Baja
Ejercicio #2	Filtrado de elementos de un Arreglo de Cadenas	Baja
Ejercicio #3	Obteniendo el Promedio de un arreglo de Decimales	Media

4.4.2 Definición de Paquetes de Prueba

A continuación, los Paquetes para la medición del desempeño de los paradigmas frente al tiempo de ejecución otorgado por el procesador:

▼ **Tabla XLI: Paquete de Prueba 1**

ID:	**Paquete 1**
Clase:	**Performance**
Ejercicio:	1
Tipo de Ejercicio:	Consulta SQL
Número de Muestras:	5
Número de Ejecuciones:	1
Métricas Aplicables:	• Líneas de Código • Número de Ejecuciones • Número de Ejecuciones por Segundo • % Eficiencia frente al Paradigma Tradicional • Relación de Desempeño • Desviación Estándar • Coeficiente de Variación

▼ **Tabla XLII: Paquete de Prueba 2**

ID:	**Paquete 2**
Clase:	**Performance**
Ejercicio:	2
Tipo de Ejercicio:	Consulta SQL
Número de Muestras:	5
Número de Ejecuciones:	100
Métricas Aplicables:	• Líneas de Código • Número de Ejecuciones

	• Número de Ejecuciones por Segundo • % Eficiencia frente al Paradigma Tradicional • Relación de Desempeño • Desviación Estándar • Coeficiente de Variación

▼ **Tabla XLIII: Paquete de Prueba 3**

ID:	**Paquete 3**
Clase:	**Performance**
Ejercicio:	3
Tipo de Ejercicio:	Consulta SQL
Número de Muestras:	5
Número de Ejecuciones:	500
Métricas Aplicables:	• Líneas de Código • Número de Ejecuciones • Número de Ejecuciones por Segundo • % Eficiencia frente al Paradigma Tradicional • Relación de Desempeño • Desviación Estándar • Coeficiente de Variación

Seguidamente, los paquetes de prueba para la medición y análisis del uso de la memoria de cada implementación son:

▼ **Tabla XLIV: Paquete de Prueba 4**

ID:	**Paquete 4**
Clase:	Memoria
Ejercicio:	4
Tipo de Ejercicio:	Consulta sobre Colección de Objetos
Número de Muestras:	5
Número de Ejecuciones:	1
Métricas Aplicables:	• Líneas de Código • Número de Ejecuciones • Uso de Memoria por Ejecución • % Eficiencia frente al Paradigma

	Tradicional • Relación de Desempeño • Desviación Estándar • Coeficiente de Variación

▼ **Tabla XLV: Paquete de Prueba 5**

ID:	**Paquete 5**
Clase:	Memoria
Ejercicio:	5
Tipo de Ejercicio:	Consulta sobre Colección de Objetos
Número de Muestras:	5
Número de Ejecuciones:	1
Métricas Aplicables:	• Líneas de Código • Número de Ejecuciones • Uso de Memoria por Ejecución • % Eficiencia frente al Paradigma Tradicional • Relación de Desempeño • Desviación Estándar • Coeficiente de Variación

▼ **Tabla XLVI: Paquete de Prueba 6**

ID:	**Paquete 6**
Clase:	Memoria
Ejercicio:	6
Tipo de Ejercicio:	Consulta sobre Colección de Objetos
Número de Muestras:	5
Número de Ejecuciones:	1
Métricas Aplicables:	• Líneas de Código • Número de Ejecuciones • Uso de Memoria por Ejecución • % Eficiencia frente al Paradigma Tradicional • Relación de Desempeño • Desviación Estándar • Coeficiente de Variación

4.4.3 Acerca de los Paradigmas de Programación e Implementaciones

Como se menciona al inicio del presente capítulo, básicamente se pretende evaluar y comparar cualitativa y cuantitativamente dos paradigmas de programación: Programación Imperativa y Programación Funcional.

En la **Programación Imperativa** los programas consisten de rutinas que contienen instrucciones que son ejecutadas mayormente en secuencia. Las estructuras de control, tales como decisiones y ciclos, y las llamadas a subprogramas (rutinas) son usadas para alterar la secuencia. Debido a esto último, la programación imperativa también se le conoce como programación estructurada y también como procedimental. El estado de un programa puede ser determinado observando los valores de las variables.

Por otro lado, en la **Programación Funcional** los programas consisten de funciones que reciben como argumentos los resultados de otras funciones. Las funciones en estos lenguajes pueden aceptar funciones como parámetros y también pueden devolver funciones como resultados. En la mayoría de los lenguajes funcionales, las listas dinámicas son un tipo de datos provisto. Por lo tanto, el programador cuenta con operaciones pre elaboradas para manejarlas.

Cada lenguaje o Framework de programación en su estructura responde a un paradigma u otro, de tal forma que existen muchas formas de implementar un algoritmo o problema utilizando una técnica u otra, un lenguaje u otro. El estudio se extiende más allá, cuando se involucra un elemento clave dentro del análisis: el framework de **LINQ** (*Lenguaje Integrado de Consultas por sus siglas en inglés*), que provee un amplio conjunto de clases y recursos para la programación funcional sobre métodos y funciones de .NET, para la realización de consultas al estilo SQL sobre colecciones o

secuencias de objetos programáticos o persistentes (objetos que mapean tablas de bases de datos relacionales).

De tal forma, dentro de los lenguajes de .NET es posible implementar el paradigma funcional usando diferentes técnicas o características que provee LINQ para la resolución de problemas

4.5 FASE 3: DEFINICIÓN DE VARIABLES Y MÉTRICAS

Un aspecto vital en el diseño de las pruebas es la definición de variables y métricas que permitan mostrar de la forma más clara los resultados de las mediciones y que su interpretación enriquezca el análisis para conclusiones más acertadas. Sin embargo, no todo son números. Aspectos cualitativos importantes se escapan a las mediciones pero que son igualmente significantes en el análisis.

El grupo de variables cuantitativas o métricas está conformado de un conjunto de indicadores de desempeño que son obtenidos directamente de las herramientas utilizadas para el monitoreo del comportamiento de las aplicaciones; y otras variables calculadas provenientes de la estadística descriptiva que nos proporcionan indicadores valiosos para el análisis de la variabilidad en las series de datos capturados.

Las variables cualitativas dan una perspectiva desde el punto de vista del desarrollador, dentro de las cuales se evalúa la claridad del código, la facilidad de codificación, la facilidad en el manejo de estructuras de programación y técnicas de programación en general.

4.5.1 Variables Cualitativas

4.5.1.1 Expresividad del Código

Indica la claridad en la "legibilidad" de un conjunto de rutinas de programación, claridad en el entendimiento del algoritmo que las líneas de código desean expresar y el grado de facilidad en el Mantenimiento del código.

4.5.1.2 Manejo de Estructuras de Programación

Indica el grado de facilidad que presenta un paradigma para el manejo, adecuación y uso de estructuras de programación dentro de una la rutina. Representa la manera en la que la técnica de programación empleada utilizada las estructuras de programación conocidas: bucles, bloques de decisión, bifurcaciones, condiciones.

4.5.1.3 Acoplamiento

La facilidad en el paradigma se ajusta al entorno de programación empleado: lenguaje y marco de clases base provistos (Framework). Representa la naturalidad que posee la técnica de adaptarse al lenguaje de programación y el mejor aprovechamiento de los recursos de programación disponibles en el .NET Framework 3.5 (librerías, Clases especializadas, tareas predefinidas, mejoras del lenguaje, entre otras).

4.5.1.4 Complejidad Computacional

Indica el esfuerzo que hay que realizar para aplicar un algoritmo y lo costoso que resulta su implementación en los paradigmas estudiados.

4.5.2 Variables Cuantitativas (Métricas)

4.5.2.1 Número de Líneas de Código

La cantidad de líneas de código que posee una rutina de programación. Define el tamaño del algoritmo.

4.5.2.2 Número de Ejecuciones

La cantidad de invocaciones o de ejecuciones consecutivas de una rutina o ejercicio.

4.5.2.3 Tiempo de Ejecución (Rendimiento):

La cantidad de tiempo de atención del procesador en milisegundos (ms) otorgado a la ejecución de un proceso o a un grupo de ejecuciones del mismo proceso. Este valor es tomado directamente de la herramienta de medición.

4.5.2.4 Número de Ejecuciones Por Segundo (Throughput):

Esta variable se aplica a los ejercicios que realizan consultas a la base de datos. Indica la cantidad de consultas a base de datos ejecutadas en 1 segundo. Cuando las aplicaciones procesan un flujo de tareas, la tasa de llegada es más importante que el peso computacional de cada tarea, el tiempo que toma procesar una de ellas no es tan importante como la cantidad puede ser procesada en un período finito de tiempo. Su fórmula se define como:

$$Numero\ de\ Ejecuciones\ Por\ Segundo\ = \frac{Número\ de\ Ejecuciones}{Tiempo\ de\ Ejecución} \times 1000$$

4.5.2.5 Ratio de Eficiencia en Ejecución

Indica la proporción en términos porcentuales en la que una técnica de programación es más eficiente en términos de ejecución frente a la técnica de programación tradicional tomada como base del análisis. La fórmula es la siguiente:

$$\boldsymbol{Eficiencia\ frente\ al\ Paradigma\ Tradicional} = \frac{\textbf{Número de Operaciones Por Segundo}_{\textbf{Técnica N}}}{\textbf{Número de Operaciones Por Segundo}_{\substack{\textbf{Técnica} \\ \textbf{Tradicional}}}} \times \mathbf{100}$$

Si el valor obtenido es mayor a 100, se interpreta que la Técnica de Programación N es más eficiente que la técnica tradicional por ejecutar mayor cantidad de operaciones por segundo. Si el valor obtenido es menor a 100, se interpreta que la Técnica de Programación N es menos eficiente que la técnica tradicional por ejecutar menor cantidad de operaciones por segundo. Si el valor obtenido es 100, significa que el número de operaciones ejecutadas en un segundo por ambas técnicas es el mismo.

4.5.2.6 Uso de Memoria

La cantidad en bytes de memoria que utiliza un proceso durante su ejecución o grupo de ejecuciones del mismo proceso. Este valor es tomado directamente de la herramienta de medición.

4.5.2.7 Relación de Desempeño

Indica la cantidad de veces en el que una técnica de programación o paradigma es más rápida o más lenta frente al paradigma tradicional. Su fórmula es:

$$\textbf{Relación de Desempeño} = \textbf{Eficiencia frente al Paradigma tradicional} - \mathbf{100}$$

Si el valor obtenido es mayor a cero, indica que el paradigma es más rápido que el paradigma tradicional; si es menor a cero indica la cantidad de veces en la que un paradigma es más lento que el tradicional. Si por el contrario el valor obtenido es cero, indica que la eficiencia del paradigma evaluado frente al paradigma tradicional es la misma.

4.5.2.8 Desviación Estándar

La desviación estándar es una medida del grado de dispersión de los datos de una serie con respecto al valor promedio de la misma. Dicho de otra manera, la desviación estándar es simplemente el "promedio" o variación esperada con respecto a la media aritmética. Su interpretación dentro del análisis es indicar la variabilidad entre los datos obtenidos en las diferentes capturas. Su fórmula es:

$$\textbf{Desviación Estándar} = \sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} (x_i - \bar{x})^2}$$

Donde N es el número de muestras o captura de valores, y $\bar{x}$ es el promedio de los valores de las muestras.

4.5.2.9 Coeficiente de Variación

En estadística el coeficiente de variación es una medida de dispersión útil para comparar dispersiones a escalas distintas pues es una medida invariante ante cambios de escala. Indica la homogeneidad de los valores de las muestras tomadas en la medición. A mayor valor de Coeficiente de Variación mayor heterogeneidad de los valores de las muestras y viceversa. Un buen benchmark debe poseer un Coeficiente de Variación menor a 1, es decir; los valores de las muestras no deben ser demasiados

dispersos. En caso de que el valor obtenido sea mayor a 1, se sospecharía que existen factores de Variabilidad Externa que han adulterado las mediciones *(véase la sección 4.3.2. Identificación de Aspectos que Afectan la Evaluación)*. Su fórmula es:

$$\boldsymbol{Coeficiente\ de\ Variación} = \frac{\boldsymbol{Desviación\ Estándar}}{\boldsymbol{Promedio}} \times \mathbf{100}$$

4.6 FASE 4: EJECUCIÓN DE PRUEBAS

Dentro de esta fase se definen los paquetes de prueba que serán ejecutados sobre el ambiente, la forma y condiciones con las que se ejecutarán y las propiedades o criterios de medición de dicho paquete. Un **Paquete de Prueba** consiste en la ejecución de las diferentes implementaciones de un determinado ejercicio o Caso de Prueba -en los paradigmas estudiados-, un determinado número de veces, medido bajo el criterio de métricas especificadas acordes al tipo de medición que se desea estudiar.

4.6.1 Ejecución de las Pruebas: Aplicación HADE COMPROTEC para el Análisis de Paradigmas

Seguidamente, los paquetes de prueba para la medición y análisis del uso de la memoria de cada implementación:

Para objeto del estudio actual y como parte del esquema de medición planteado en éste capítulo, cada ejercicio definido en la sección *4.4.1.Definición de Casos de Prueba* debe ser resuelto usando los paradigmas o técnicas de programación planteados: Programación Imperativa usando el estilo tradicional, Programación Funcional usando LINQ y Programación Funcional usando LINQ y Expresiones Lambda. Para lograrlo, se ha creado una herramienta que contiene las

implementaciones de cada ejercicio, una en cada técnica de programación planteada. La aplicación de pruebas es de tipo Escritorio (Windows Forms), la misma que se encuentra dentro de la solución del sistema, y hace referencia al componente de Capa de Datos, específicamente a la clase **AnalisisParadigmas.cs**, que contiene las implementaciones de los ejercicios en los diferentes paradigmas, así como las consultas a la base de datos.

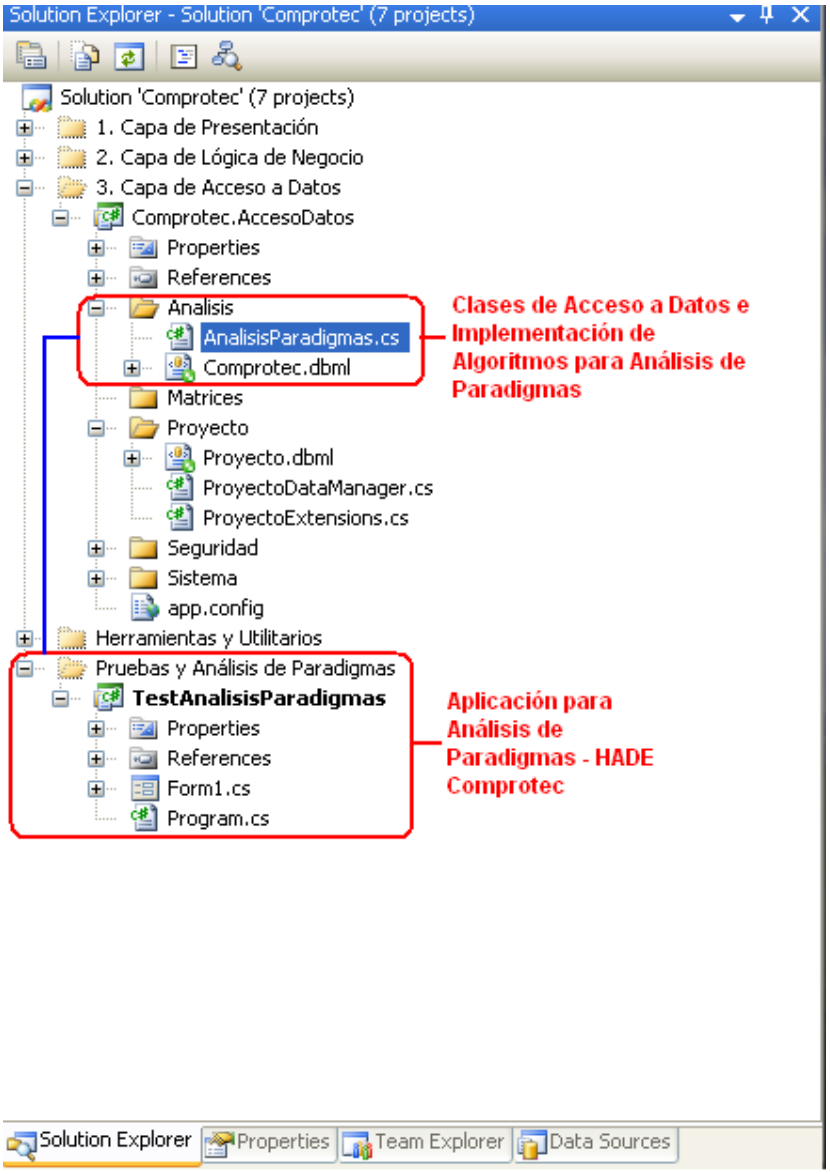


Figura 4.2: Esquema de la Aplicación de Prueba para elegir el mejor Paradigma

El formulario principal de la herramienta posee varios grupos de botones agrupados por ejercicio definido, un grupo por cada ejercicio.

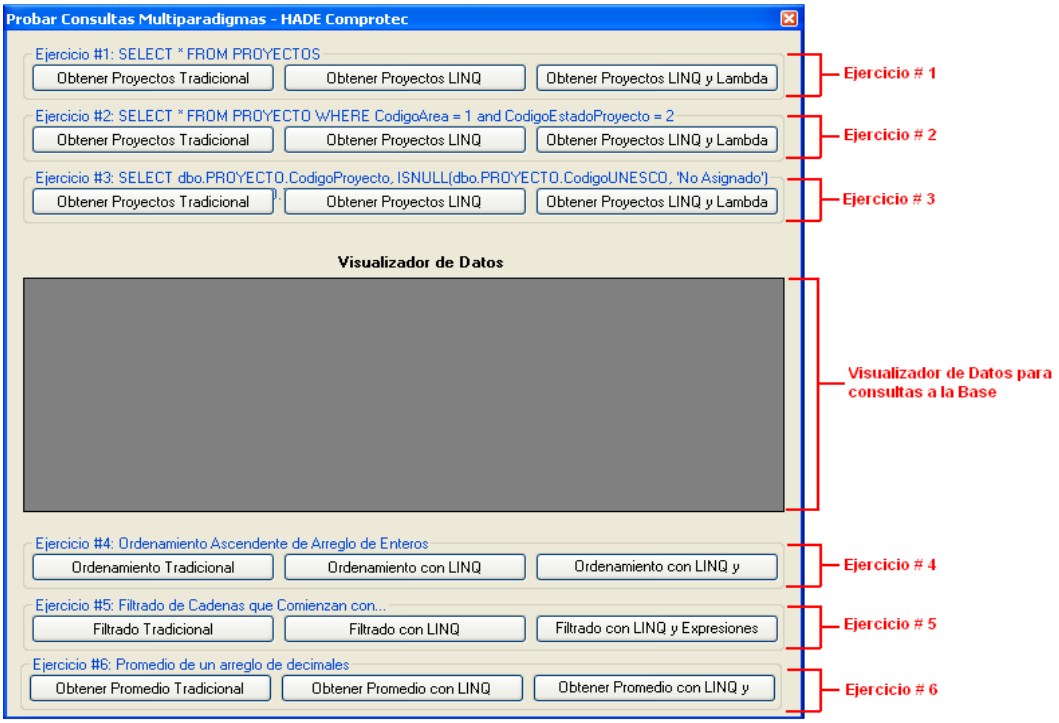


Figura 4.3: Lista de Ejercicios y sus Implementaciones en cada Paradigma

Dentro de cada grupo existen 3 botones, donde cada uno representa la implementación del ejercicio en un paradigma o técnica de programación determinado.

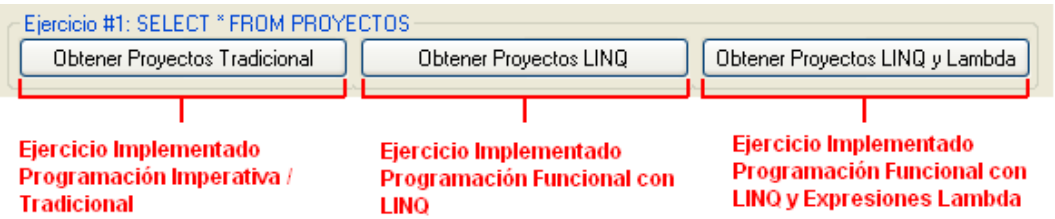


Figura 4.4: Paradigmas a ser comparados

El evento Click de cada botón se encuentra enlazado a un método que invoca a la implementación del algoritmo en el paradigma respectivo dentro de la clase AnalisisParadigmas en el proyecto de Acceso Datos.

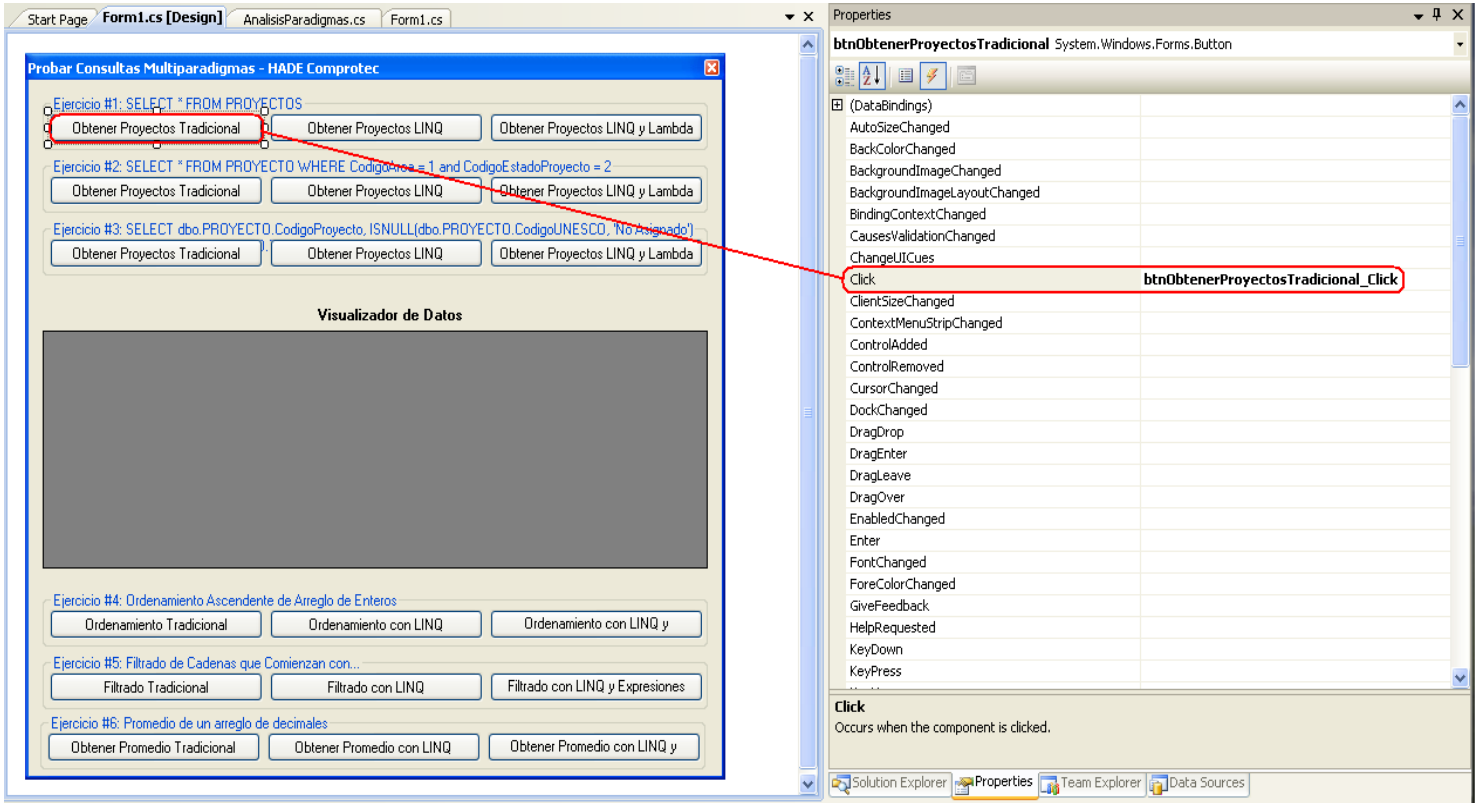
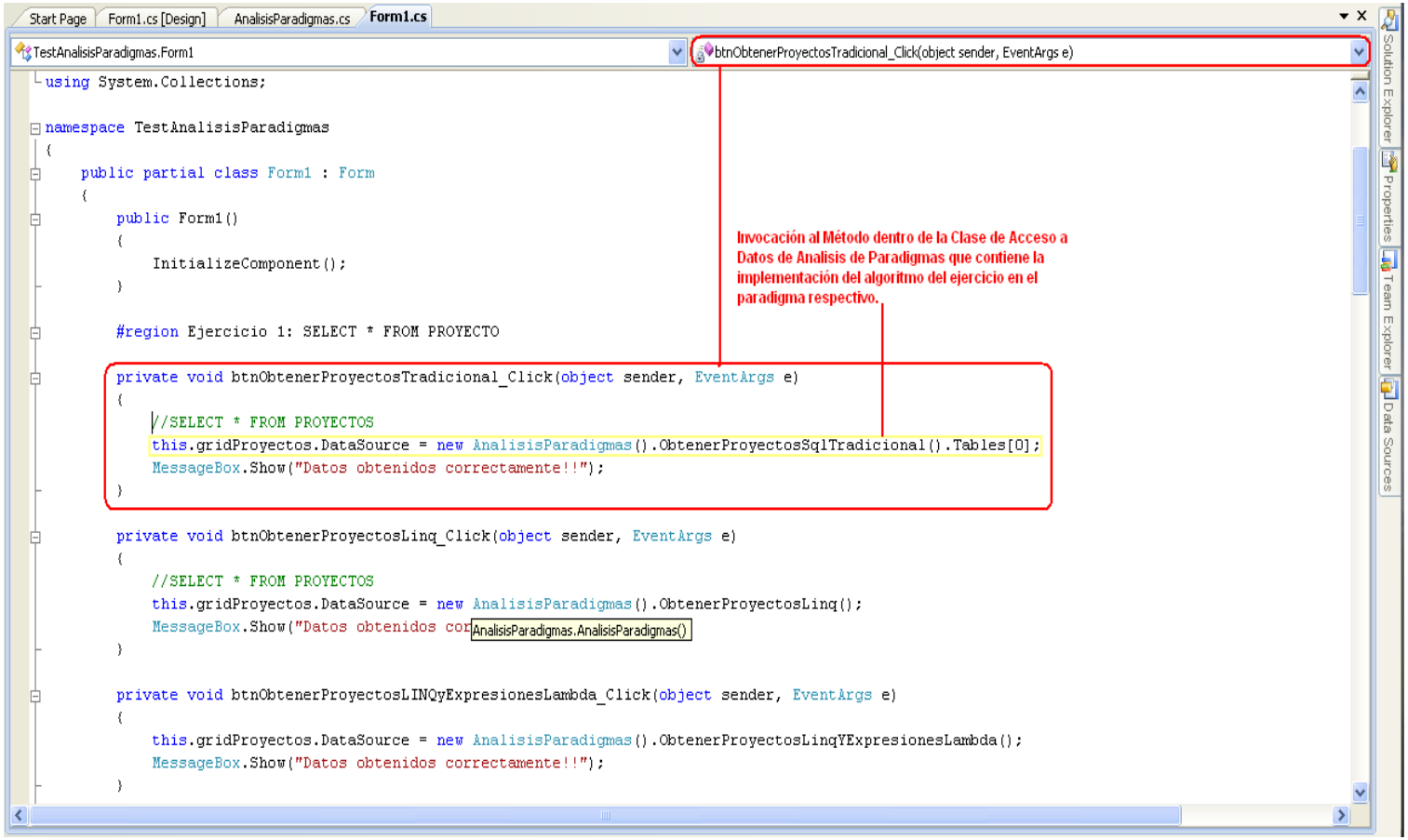


Figura 4.5: Evento Clic del Botón de un Paradigma seleccionado




```
using System.Collections;

namespace TestAnalisisParadigmas
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        #region Ejercicio 1: SELECT * FROM PROYECTO

        private void btnObtenerProyectosTradicional_Click(object sender, EventArgs e)
        {
            //SELECT * FROM PROYECTOS
            this.gridProyectos.DataSource = new AnalisisParadigmas().ObtenerProyectosSqlTradicional().Tables[0];
            MessageBox.Show("Datos obtenidos correctamente!!");
        }

        private void btnObtenerProyectosLinq_Click(object sender, EventArgs e)
        {
            //SELECT * FROM PROYECTOS
            this.gridProyectos.DataSource = new AnalisisParadigmas().ObtenerProyectosLinq();
            MessageBox.Show("Datos obtenidos cor
        }

        private void btnObtenerProyectosLINQyExpresionesLambda_Click(object sender, EventArgs e)
        {
            this.gridProyectos.DataSource = new AnalisisParadigmas().ObtenerProyectosLinqYExpresionesLambda();
            MessageBox.Show("Datos obtenidos correctamente!!");
        }
```

AnalisisParadigmas.AnalisisParadigmas()

Figura 4.6: Invocación al Método dentro de la clase de Acceso a Datos de Análisis de Paradigmas que contiene la implementación del Algoritmo

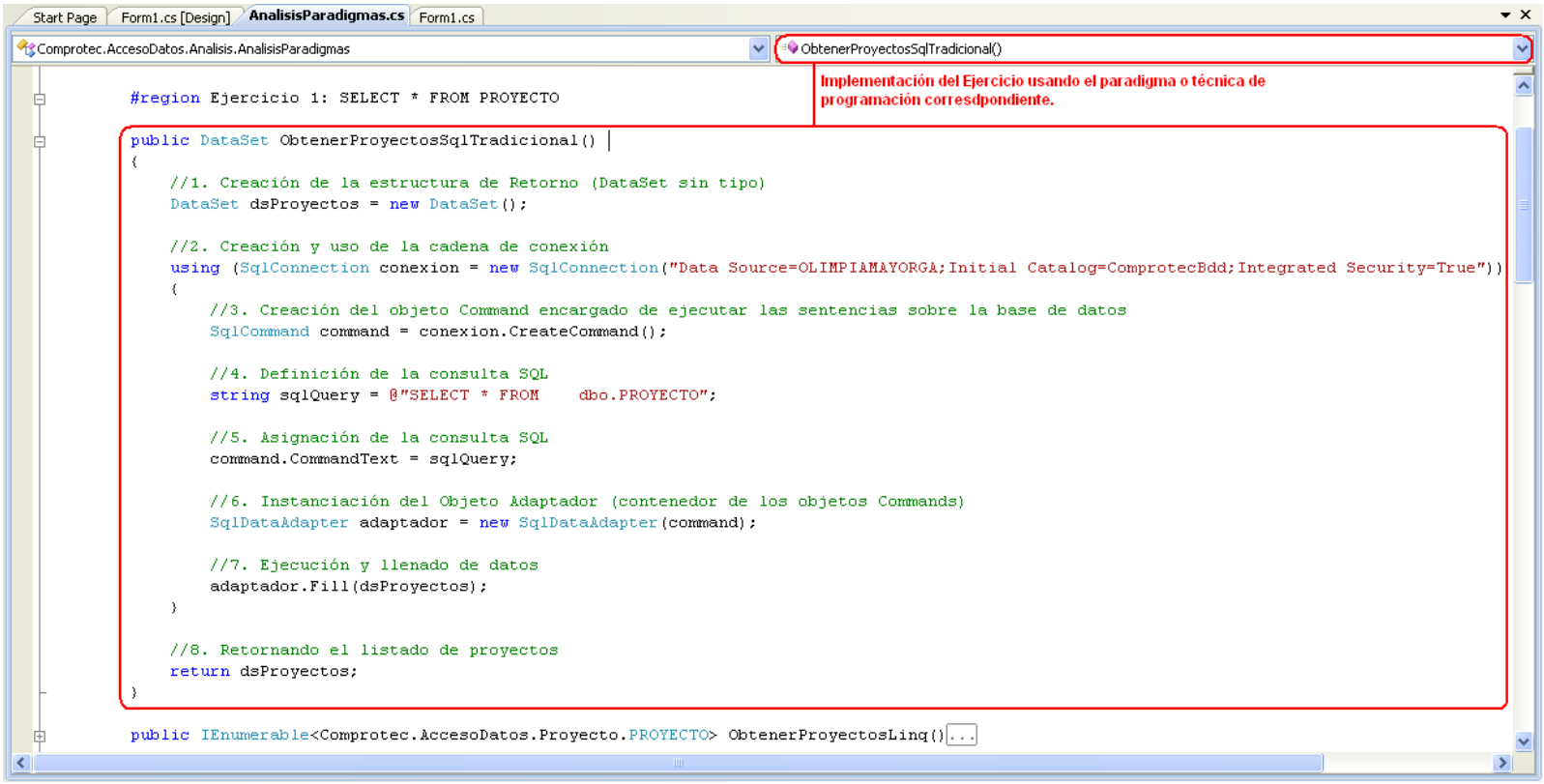


Figura 4.7: Implementación del Ejercicio usando el Paradigma Respectivo

4.6.2 Acerca de la Toma de Datos

Para todos los casos, las mediciones se basan en capturas parciales de los datos que las herramientas arrojan al probar una implementación. Una **Muestra** es la captura del valor de una magnitud arrojado por la herramienta de medición utilizada, que resulta de la ejecución de un método por un determinado número de invocaciones. Cada muestra se toma en ejecuciones aisladas una de la otra; es decir, que para tomar una muestra primero se inicia la aplicación de Pruebas de Paradigmas Comprotec, se ejecuta la implementación o método el número de veces especificado, se toman los valores y se termina la ejecución del programa. De esta manera, cada muestra es independiente del ámbito de ejecución una de la otra.

Para el Caso de Mediciones de Performance (Tiempo de Ejecución), las muestras utilizadas para el cálculo del Coeficiente de Variación corresponden a aquellas capturadas luego de la primera invocación posterior a la ejecución del aplicativo de pruebas; es decir exceptuando la **Muestra de Inicialización o Muestra Inicial MI**. Esta consideración se hace especialmente por el diseño de ejecución de los objetos .NET, en donde a la primera ejecución se consume la mayor cantidad de recursos, instanciando y alojando objetos que serán reutilizados en ejecuciones siguientes; por ende las ejecuciones a partir de la muestra inicial serán más eficientes y brindarán medidas más homogéneas.

Si no se toma en cuenta esta consideración, el valor del Coeficiente de Variación calculado puede incrementarse, debido a que posiblemente la primera muestra poseerá un valor muy alto (hablando en términos de tiempos de ejecución) que las subsiguientes. De esta forma, se generará una desviación mayor en las muestras, causando distorsión en las medidas. A simple vista, esta distorsión fundamentaría la sospecha de la existencia de Factores de Variabilidad Externa que intervienen de forma

latente en todo el proceso de Benchmark. Por lo tanto, el excluir a la Muestra de Inicialización del conjunto de Muestras estudiadas nos brinda un alto nivel de confiabilidad en los resultados del análisis.

4.6.3 Acerca de la Evaluación Retardada en las Pruebas

En la sección *2.4.2.2 Lazy Evaluation* del capítulo II del presente documento, se introduce el concepto de Evaluación Retardada dentro de las características del empleo de LINQ sobre consultas sobre secuencias en general. Básicamente, la Evaluación Retardada consiste en demorar el cálculo computacional de una consulta LINQ hasta que los resultados de dicha consulta sean utilizados. Esto implica que si se asigna a una variable el resultado de una consulta y ésta variable nunca es utilizada como entrada de un proceso o para el despliegue del valor en una pantalla –por ejemplo-, la consulta LINQ nunca se ejecutará, lo que en sí representa una buena práctica y mejora el desempeño general de un proceso.

Para la realización de pruebas Benchmarking, es necesario eliminar la Evaluación Retardada o Lazy Evaluation de las consultas a medir; debido a que se puede obtener mediciones erróneas sobre un conjunto de invocaciones continuas de una misma consulta, cuyos valores son asignados a la misma variable. El problema radica en que si se ejecuta un método un número definido de iteraciones, y para cada iteración el resultado se almacena en la misma variable que seguidamente será usada luego del bucle, el valor de la medición del tiempo de ejecución de ese conjunto de invocaciones, será igual al tiempo de ejecución de la última consulta.

El problema se ilustra en el siguiente bloque de código:

▼ **Tabla XLVII: Bloque de código que ilustra un ejemplo típico con Lazy Evaluation**

```
//Manejador del evento Click del Botón btnObtenerProyectosLinq
private void btnObtenerProyectosLinq_Click(sender, e)
{

//Variable de Salida en Pantalla
IEnumerable<Comprotec.AccesoDatos.Proyecto.PROYECTO> proyectos =
null;

//Invocando al método LINQ un número de Iteraciones
for (int i = 0; i < Convert.ToInt32(txtNumeroEjecuciones.Text); i++)
{
  //Ejecutando el método LINQ y guardando los resultados en variable
  proyectos = analisisParadigmasDataManager.ObtenerProyectosLinq();
}

//Mostrando los resultados en grilla de datos
 this.gridProyectos.DataSource = proyectos;
 MessageBox.Show("Datos obtenidos correctamente!!");
}
```

En la asignación del valor del método ObtenerProyectosLinq en la variable proyectos, cada iteración reemplazará el valor de la variable desechando el valor anterior. Entonces, la consulta LINQ correspondiente al valor anterior nunca será ejecutada ya que su valor jamás se utilizó. Esto implica que, las herramientas de medición no registrarán la ejecución del método y por ende no se podrá obtener un valor real del resultado de ejecutar N veces un proceso. Para evitar el Lazy Evaluation para las mediciones de Performance es necesario forzar a la ejecución de la consulta LINQ cada vez. Para lograrlo se puede utilizar operadores No Diferidos, que obligan la compilación del árbol de expresión respectivo y la ejecución de la consulta. Para las pruebas de Performance de HADE – Comprotec se ha utilizado el operador **ToList.**

De tal forma las invocaciones se utilizarían de la siguiente forma:

▼ **Tabla XLVIII: Evitando Lazy Evaluation a través de ToList**

```
//Manejador del evento Click del Botón btnObtenerProyectosLinq
private void btnObtenerProyectosLinq_Click(sender, e)
{

//Variable de Salida en Pantalla
IEnumerable<Comprotec.AccesoDatos.Proyecto.PROYECTO> proyectos =
null;

//Invocando al método LINQ un número de Iteraciones
for (int i = 0; i < Convert.ToInt32(txtNumeroEjecuciones.Text); i++)
{
  //Ejecutando el método LINQ y guardando los resultados en variable
  proyectos =
analisisParadigmasDataManager.ObtenerProyectosLinq().ToList();
}

//Mostrando los resultados en grilla de datos
 this.gridProyectos.DataSource = proyectos;
 MessageBox.Show("Datos obtenidos correctamente!!");
}
```

4.7 FASE TABULACIÓN DE RESULTADOS

A continuación se encuentran los resultados obtenidos de cada una de las métricas cualitativas y cuantitativas que se han utilizado para realizar el estudio comparativo.

4.7.1 Métricas Cualitativas

Para interpretar los valores de las tablas de datos presentadas en esta fase en cuanto a las variables cualitativas, es necesario revisar la tabla de valoraciones y su significado que a continuación se detalla:

Categoria	Valor	Expresividad de Código	Manejo de Estructuras de Programación	Acoplamiento	Complejidad Computacional
Baja	1	A medida que aumenta el grado de complejidad de la consulta es mas dificil interpretar dicho codigo implementado.	Dificultad en el manejo de estructuras de programación	No se acopla a la mejoras del lenguaje y no mantine compatibilidad con los objetos del framework	La complejidad de la consulta se reduce usando los operadores de consultas estándar, pero puede incrementar la complejidad en función de la complejidad de la consulta.
Media	2	Legibilidad del codigo es poco clara y su grado e facilidad para el mantenimiento de codigo es mas compleja	Facilidad en el manejo de estructuras de consulta utilizando estructuras conocidas	Manejo únicamente de objetos para el acceso a datos provistos por el Framework ADO.NET y el namespace System.Data. Permite el manejo directo y consumo de prodecimientos almacenados.	La complejidad de la consulta se reduce usando los operadores de consultas estándar, pero puede incrementar la complejidad en función de la complejidad de la consulta.
Alta	3	Definición de la consulta de datos en lenguaje SQL, comprensión directa de la consulta.	Utiliza estructuras de consulta	Se acopla a las mejoras del lenguaje de .NET y mantiene compatibilidad con las clases y objetos clásicos del framework	La complejidad del algoritmo es proporcional a la complejidad de la consulta.
Muy Alta	4	Interpretación más natural de la consulta SQL en sintaxis de programación de acuerdo al lenguaje empleado y usando funciones con nombres familiares (Where, Join, etc.).	Las secuencias soportan todas las esctructuras de programación: bucles, condiciones, etc.	Se acopla a las mejoras del lenguaje de .NET y mantiene compatibilidad con las clases y objetos clásicos del framework. Soporte de expresiones Lambda y árboles de expresión.	La complejidad de la consulta no se reduce usando los operadores de consultas estándar, la complejidad de la consulta obliga a la creación de consultas sobre consultas. Es necesario la creación de muchas variables internas necesarias para las consultas parciales dentro de la expresión.

Figura 4.8: Valoraciones y su significado establecido para las variables cualitativas

Descripción detallada de cada métrica cualitativa utilizada en el estudio:

	Expresividad del Código		
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	3	3	4
Ejercicio 2	3	3	4
Ejercicio 3	3	2	1
Ejercicio 4	3	3	4
Ejercicio 5	3	3	4
Ejercicio 6	3	2	4
Promedios	**3**	**2,666666667**	**3,5**

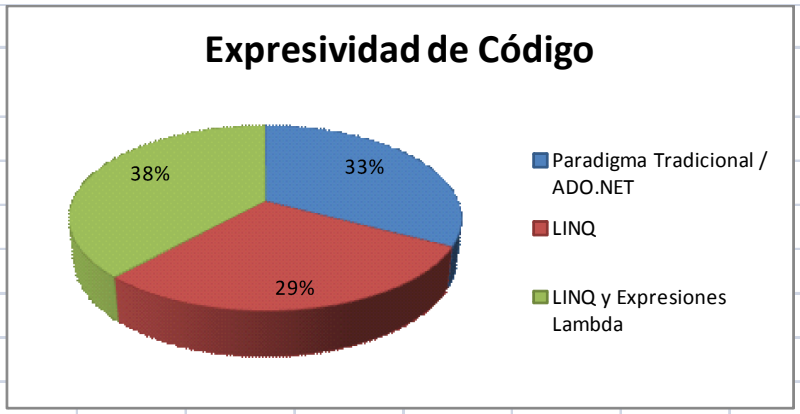


Figura 4.9: Representación Gráfica de la métrica Expresividad de Código

	Manejo de Estructuras de Programación		
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	1	4	4
Ejercicio 2	2	3	4
Ejercicio 3	3	3	2
Ejercicio 4	2	3	4
Ejercicio 5	2	3	4
Ejercicio 6	2	3	4
Promedios	**2**	**3,166666667**	**3,666666667**

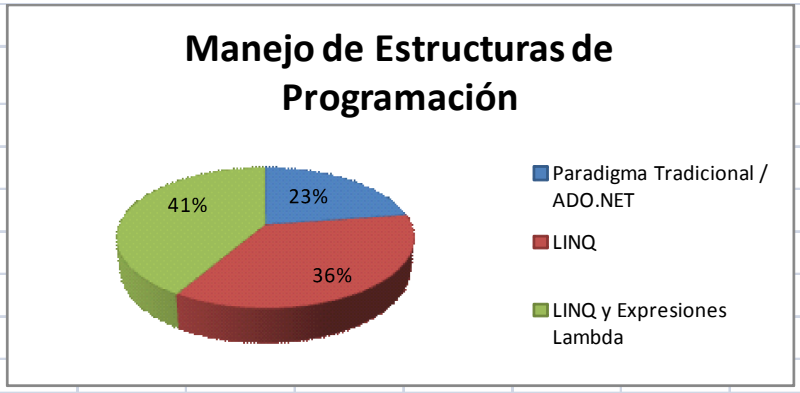


Figura 4.10: Representación Gráfica de la métrica Manejo de Estructuras de Programación

	Acoplamiento		
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	2	3	4
Ejercicio 2	2	3	4
Ejercicio 3	4	3	2
Ejercicio 4	2	3	4
Ejercicio 5	3	3	3
Ejercicio 6	3	3	3
Promedios	**2,666666667**	**3**	**3,333333333**

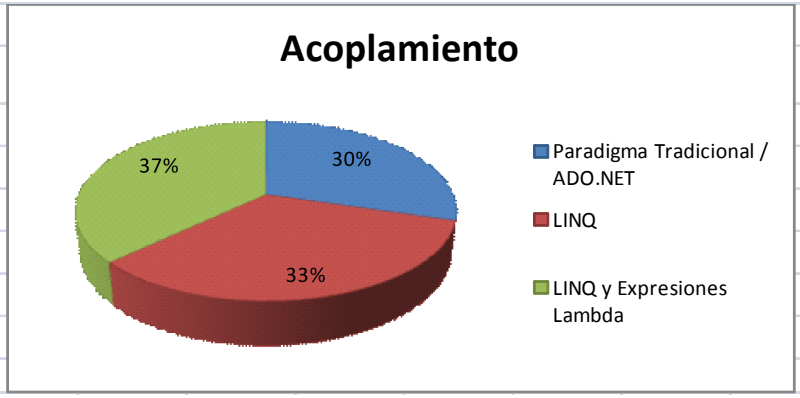


Figura 4.11: Representación Gráfica de la métrica Acoplamiento

	Complejidad Computacional		
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	3	3	2
Ejercicio 2	3	3	2
Ejercicio 3	2	4	4
Ejercicio 4	2	2	1
Ejercicio 5	2	2	1
Ejercicio 6	2	2	1
Promedios	**2,333333333**	**2,666666667**	**1,833333333**

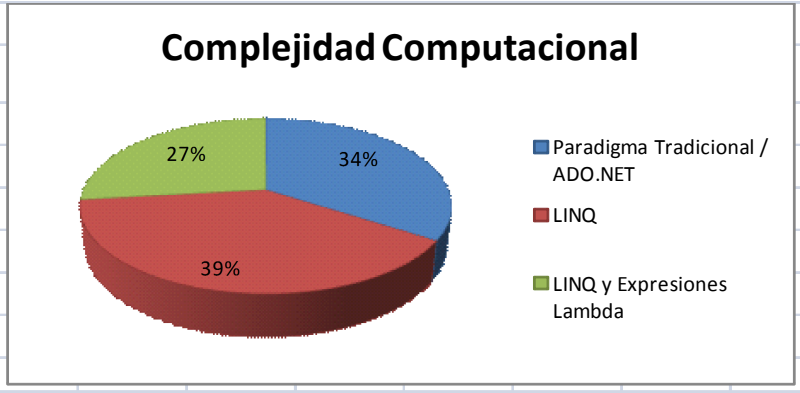


Figura 4.12: Representación Gráfica de la métrica Complejidad Computacional

Métricas Cuantitativas

	Número de Líneas de Código		
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	10	2	1
Ejercicio 2	15	4	2
Ejercicio 3	35	30	75
Ejercicio 4	5	2	1
Ejercicio 5	5	2	1
ejercico 6	5	3	1
Promedios	**12,5**	**7,166667**	**13,5**

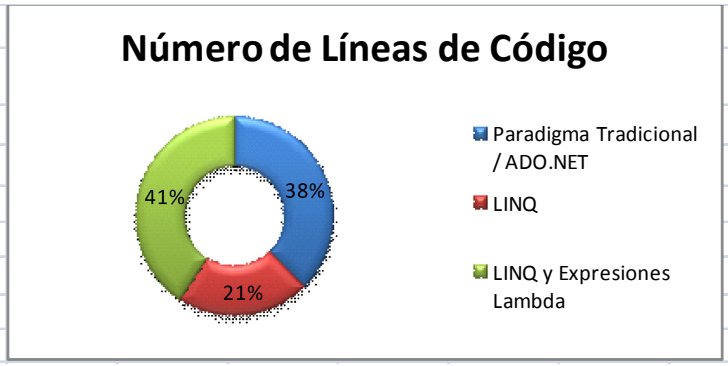
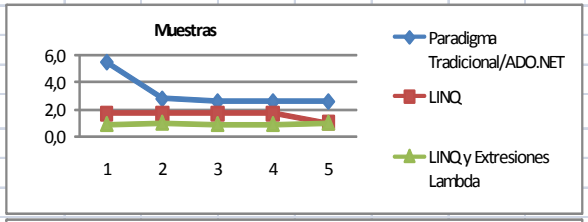


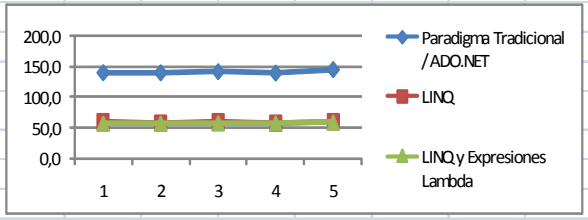
Figura 4.13: Representación Gráfica de la métrica Número de Líneas de Código

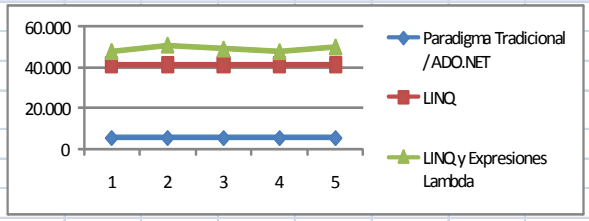
	Número de Ejecuciones		
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	1	1	1
Ejercicio 2	100	100	100
Ejercicio 3	500	500	500

Figura 4.14: Representación Gráfica de la métrica Número de Ejecuciones

		Tiempo de Ejecución (ms)		
		Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
	M	1.103,0	1.244,0	1.183,0
Ejercicio 1	M1	5,5	1,7	0,9
	M2	2,8	1,7	1,0
	M3	2,6	1,7	0,9
	M4	2,6	1,7	0,9
	M5	2,6	1,0	1,0
	Promedio	3,22	1,56	0,94
	M	1.282	1.285	1.230
Ejercicio 2	M1	140,3	60,0	57,0
	M2	140,2	57,4	57,1
	M3	141,9	59,0	57,6
	M4	139,7	58,0	57,5
	M5	145,2	60,0	59,0
	Promedio	141,5	58,9	57,6
	M	6.664	43.944	44.911
Ejercicio 3	M1	5.629	41.096	48.096
	M2	5.658	41.139	51.107
	M3	5.609	41.284	49.233
	M4	5.597	41.265	48.010
	M5	5.588,00	41.304	50.247
	Promedio	5.616	41.218	49.339
Total	**Promedios**	**1.920**	**13.759**	**16.466**







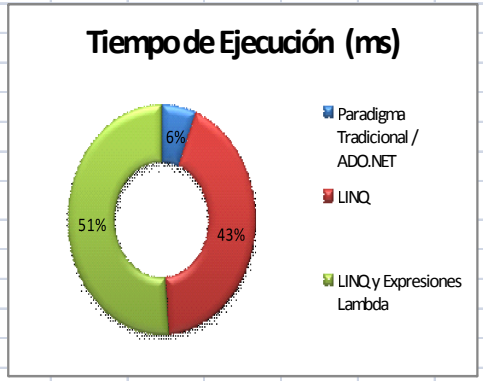


Figura 4.15: Representación Gráfica de la métrica Tiempo de Ejecución (ms)

Número Ejecuciones/seg			
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	310,5590062	641,02564	1063,829787
Ejercicio 2	706,7137809	1697,7929	1736,111111
Ejercicio 3	89,03133903	12,130623	10,1339711
Promedios	**368,7680421**	**783,6497**	**936,6916231**

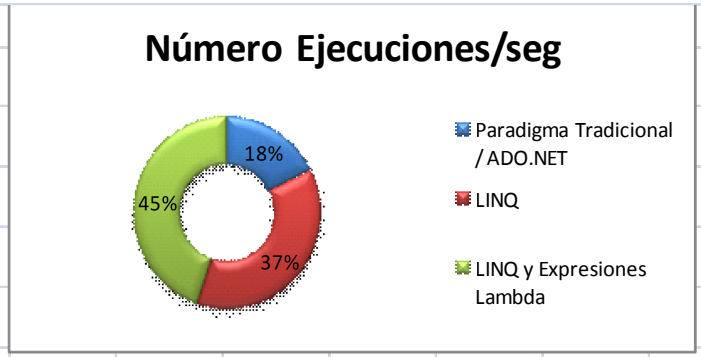


Figura 4.16: Representación Gráfica de la métrica Número Ejecuciones/seg

% de Eficiencia frente al Tradicional			
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	100,000%	206,410%	342,550%
Ejercicio 2	100,000%	240,250%	245,420%
Ejercicio 3	100,000%	13,630%	11,380%
Ejercicio 4	100,000%	37,500%	75,000%
Ejercicio 5	100,000%	100,000%	100,000%
ejercico 6	100,000%	34,340%	68,000%
Promedios	**100%**	**105%**	**140%**

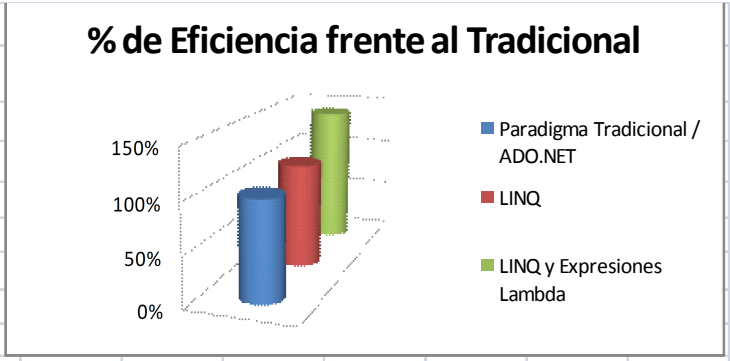


Figura 4.17: Representación Gráfica de la métrica % de Eficiencia frente al Paradigma Tradicional

	Uso de Memoria																	
	Programación Imperativa / ADO.NET Tradicional						Programación Funcional / LINQ						Programación Funcional / LINQ y Expresiones Lambda					
	M1	M2	M3	M4	M5	Promedio	M1	M2	M3	M4	M5	Promedio	M1	M2	M3	M4	M5	Promedio
Ejercicio 4	24,0	24,0	24,0	24,0	24,0	24,0	64,0	64,0	64,0	64,0	64,0	64,0	32,0	32,0	32,0	32,0	32,0	32,0
Ejercicio 5	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0	60,0
Ejercicio 6	34,00	34,00	34,00	34,00	34,00	34,0	99,00	99,00	99,00	99,00	99,00	99,0	50,00	50,00	50,00	50,00	50,00	50,0
Promedios						**39,3**						**74,3**						**47,3**

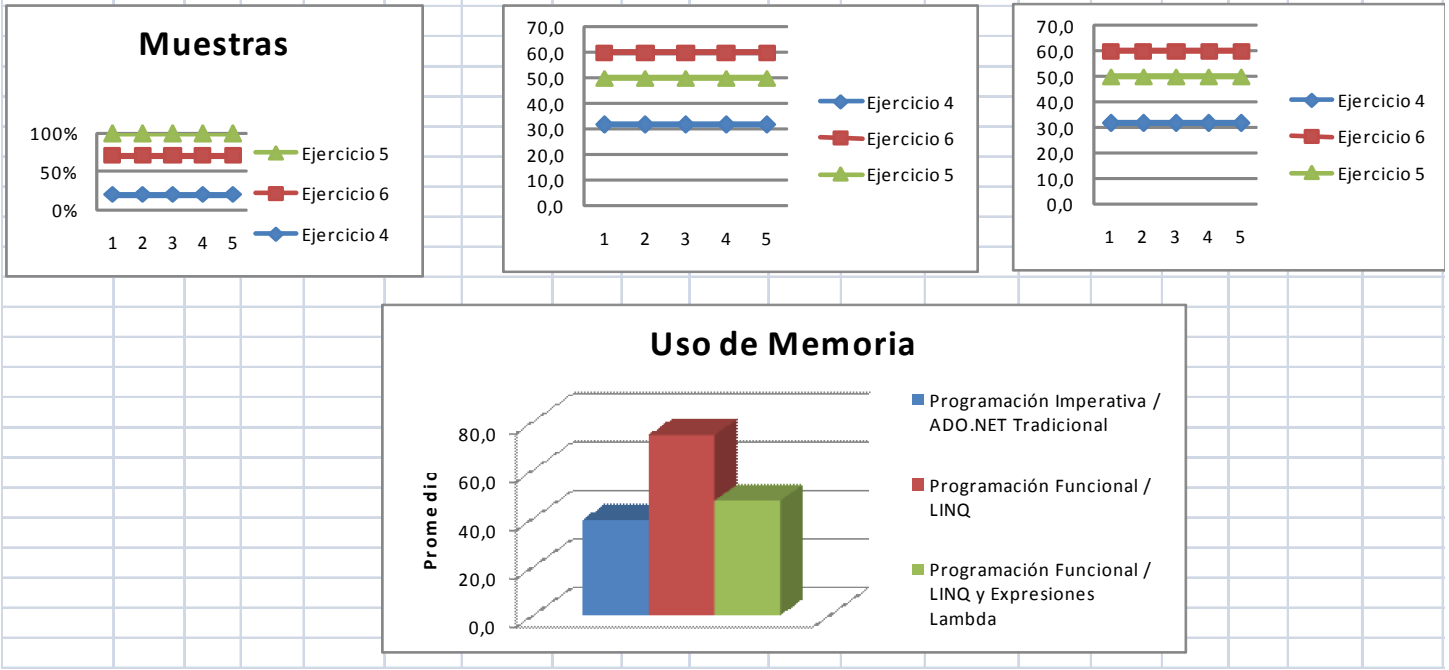


Figura 4.18: Representación Gráfica de la métrica Uso de Memoria

	Relación de Desempeño		
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	0,00	1,06	2,43
Ejercicio 2	0,00	1,40	1,45
Ejercicio 3	0,00	-0,86	-0,89
Ejercicio 4	0,00	-0,63	0,00
Ejercicio 5	0,00	0,00	0,00
Ejercicio 6	0,00	-0,66	0,00
Promedios	**0**	**0,051666667**	**0,498333333**

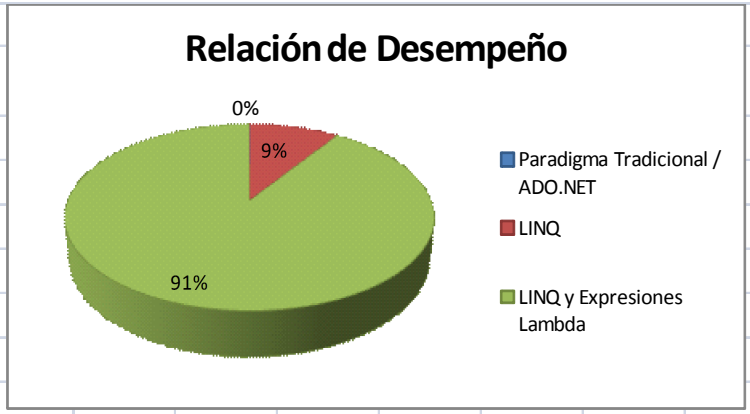


Figura 4.19: Representación Gráfica de la métrica Relación de Desempeño

	Desviación Estándar		
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	1,14	0,28	0,050
Ejercicio 2	2,01	1,05	0,72
Ejercicio 3	25,01	83,77	1206,05
Ejercicio 4	0,00	0,00	0,00
Ejercicio 5	0,00	0,00	0,00
Ejercicio 6	0,00	0,00	0,00
Promedios	**4,693333333**	**14,18333333**	**201,1366667**

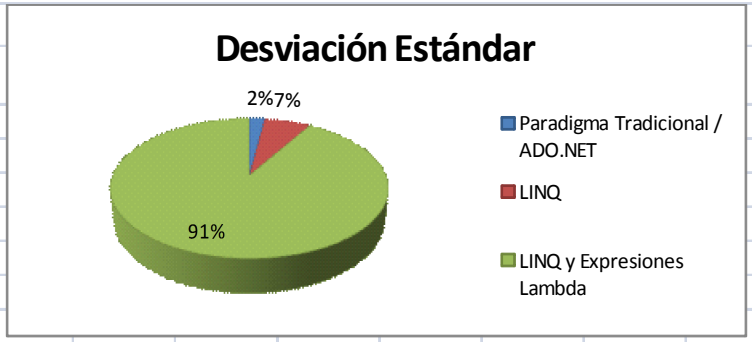


Figura 4.20: Representación Gráfica de la métrica Desviación Estándar

	Coeficiente de Variación		
	Paradigma Tradicional / ADO.NET	LINQ	LINQ y Expresiones Lambda
Ejercicio 1	0,3549	0,1795	0,0521
Ejercicio 2	0,0142	0,0178	0,0124
Ejercicio 3	0,0000	0,0000	0,0200
Ejercicio 4	0,0000	0,0000	0,0000
Ejercicio 5	0,0000	0,0000	0,0000
Ejercicio 6	0,0000	0,0000	0,0000
Promedios	**0,061516667**	**0,032883333**	**0,014083333**

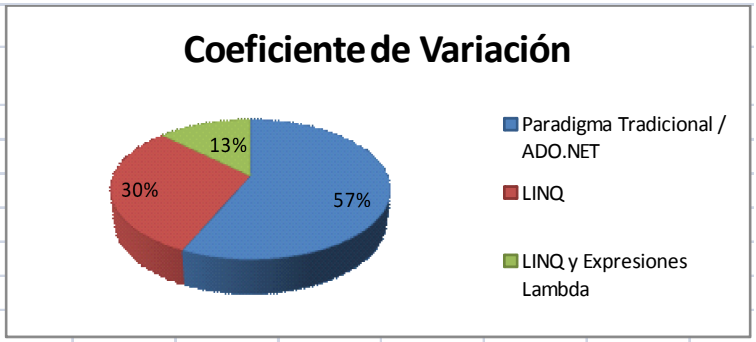


Figura 4.21: Representación Gráfica de la métrica Coeficiente de Variación

4.8 FASE 6: ANÁLISIS Y COMPARACIÓN DE RESULTADOS

El análisis de enfocará en tres aspectos principales: el Tiempo de Ejecución, el Uso de la Memoria y el uso del Paradigma desde el punto de vista programático.

4.8.1 Análisis de los Resultados en Tiempo de Ejecución

A continuación se presenta un resumen de las mediciones obtenidas para la categoría Tiempo de Ejecución de los paquetes de prueba ejecutados:

EJERCICIO 1	Tiempo de Ejecución (ms)		Número de Ejecuciones	Número Ejecuciones /seg	% Eficiencia frente Tradicional	Relación de Desempeño	Evaluación Preliminar	
	MI	Promedio					Desempeño	El Mejor Paradigma
Programación Imperativa / ADO.NET Tradicional	1.103,0	3,22	1	310,5590062	100,00%	0,00		
Programación Funcional / LINQ	1.244,0	1,56	1	641,025641	206,41%	1,06		
Programación Funcional / LINQ y Expresiones Lambda	1.183,0	0,94	1	1063,829787	342,55%	2,43		

Figura 4.22: Resumen de Resultados de Prueba de Tiempo de Ejecución-caso de prueba 1

El Paquete de Prueba #1 consiste en una única ejecución de una consulta SQL “plana” sobre una tabla (consulta de todos los registros y todos los campos). Los valores registrados para la columna Medición Inicial (MI) muestran que el Paradigma Imperativo Tradicional basado en el uso de objetos ADO.NET, es más eficiente en la instanciación primaria de los objetos necesarios para ejecutar las consultas y alojarlos en memoria A partir de la Medición Inicial, los tiempos de ejecución se reducen notablemente, gracias el esquema de instanciación de objetos de .NET (*véase la sección 4.6.4. Acerca de la Toma de Datos – Medición Inicial*). El tiempo de ejecución promedio para una invocación revela una significativa diferencia entre los tiempos obtenidos de la programación Imperativa usando ADO.NET y las implementaciones funcionales usando LINQ. De esta forma, a primera instancia se tiene que el

mecanismo más eficiente es la Programación funcional usando LINQ y Expresiones Lambda con un tiempo de 0,94 ms frente a los 3,22 ms de la programación Imperativa usando ADO.NET. En términos de Número de Ejecuciones/segundo, se muestra claramente una gran capacidad de ejecuciones simultáneas en el lapso de un segundo para LINQ y Expresiones Lambda, en una proporción de 342,55% en comparación con el paradigma imperativo ADO.NET, lo que indica una relación de eficiencia de 2,43x; es decir, LINQ y Expresiones Lambda es 2,43 veces más rápido en la ejecución de esta consulta frente al paradigma imperativo con ADO.NET.

EJERCICIO 2	Tiempo de Ejecución (ms)		Número de Ejecuciones	Número Ejecuciones/seg	% Eficiencia frente Tradicional	Relación de Desempeño	Evaluación Preliminar	
	M	Promedio					Desempeño	El Mejor Paradigma
Programación Imperativa / ADO.NET Tradicional	1.282	141,5	100	706,9136152	100,00%	0,00		
Programación Funcional / LINQ	1.285	58,9	100	1698,369565	240,25%	1,40		
Programación Funcional / LINQ y Expresiones Lambda	1.230	57,6	100	1734,906315	245,42%	1,45		

Figura 4.23: Resumen de Resultados de Prueba de Tiempo de Ejecución-caso de prueba 2

En la tabla anterior se muestran un resumen de las mediciones y resultados para el Paquete de Prueba #2, que consiste en la ejecución de 100 invocaciones continuas de una consulta SQL sobre una tabla incluyendo operadores de restricción y condiciones lógicas (WHEREs, ANDs y ORs), y estableciendo ciertos campos de salida, incrementando levemente el nivel de complejidad con respecto a la consulta anterior. Los valores registrados para la columna Medición Inicial (MI) son bastante homogéneos, con una ligera ventaja para el Paradigma Funcional implementado con LINQ y Expresiones Lambda. El tiempo de ejecución promedio para las 100 invocaciones muestra nuevamente una superioridad para las implementaciones LINQ del paradigma funcional. El menor tiempo empleado para la ejecución de este paquete fue obtenido por la implementación funcional de LINQ y Expresiones Lambda, registrando 57,6 ms frente a 141,5 ms de la implementación tradicional ADO.NET. En

función al tiempo de ejecución registrado, la capacidad de Ejecuciones por segundo de LINQ y Expresiones Lambda (1734,906315 consultas/segundo) sobrepasa por más del doble de la capacidad de ejecuciones continuas de ADO.NET (706,91316151 consultas/segundo); estableciendo una proporción de 245,42% más de eficiencia de LINQ y Expresiones Lambda, lo que indica una relación de desempeño de 1,45 veces más veloz que el paradigma tradicional.

EJERCICIO 3	Tiempo de Ejecución (ms)		Número de Ejecuciones	Número Ejecuciones/seg	% Eficiencia frente Tradicional	Relación de Desempeño	Evaluación Preliminar	
	M	Promedio					Desempeño	El Mejor Paradigma
Programación Imperativa / ADO.NET Tradicional	6.664	5.616	500	89,02816851	100,00%	0,00		
Programación Funcional / LINQ	43.944	41.218	500	12,13074027	13,63%	-0,86		
Programación Funcional / LINQ y Expresiones Lambda	44.911	49.339	500	10,13405326	11,38%	-0,89		

Figura 4.24: Resumen de Resultados de Prueba de Tiempo de Ejecución-caso de prueba 3

El Paquete de Prueba #3 consiste en la ejecución de 500 invocaciones continuas de una consulta SQL más elaborada, que incluye lógica de Negocio y cuyo nivel de complejidad respecto a las consultas anteriores es mayor. Los resultados para el Paquete de Prueba #3 son inesperados, basados en el comportamiento de los paradigmas en los ejercicios anteriores. En primer lugar, se evidencia una notable diferencia en los valores de la Medición Inicial, en donde el valor de instanciación y alojamiento inicial de objetos para la programación Imperativa usando ADO.NET es aproximadamente 7 veces menor a las mediciones de los paradigmas implementados con LINQ. Subsecuentemente, el promedio del tiempo de ejecución favorece por primera ocasión al Paradigma Imperativo - ADO.NET con 5.5616 ms usados en la ejecución de 500 consultas. De igual forma, el número de ejecuciones / segundo es aproximadamente 8 veces mayor, lo que indica una gran capacidad de ejecuciones continuas de tareas de datos complejas en comparación a los paradigmas LINQ, que registraron 12,13074027 ms

para LINQ puro y 10,1340532564767 ms para LINQ y Expresiones Lambda. En términos de Desempeño, se tiene un ratio de 13,63% para LINQ y 11,38% para LINQ y Expresiones Lambda, o en otras palabras; LINQ con -0,86x y LINQ con Expresiones Lambda con -0,89x, son más lentos que el consumo de datos ADO.NET para este ejercicio.

4.8.2 Análisis de los Resultados en Uso de la Memoria

Una característica de las mediciones de Memoria, es que los valores de las muestras tomados son bastante homogéneos, tal como lo indica el Coeficiente de Variación para cada grupo de medidas, inclusive en algunos casos es cero, lo que indica una completa homogeneidad en las mediciones y que el mecanismo de alojamiento de objetos de .NET es siempre el mismo para el número de objetos generados para la misma ejecución, sin importar cuantas veces se ejecute el ejercicio.

A continuación se presenta un resumen de las mediciones obtenidas para la categoría Uso de Memoria de los paquetes de prueba ejecutados:

EJERCICIO 4	Uso de Memoria (bytes)	% Eficiencia frente Tradicional	Relación de Desempeño	Evaluación Preliminar	
	Promedio			Desempeño	El Mejor Paradigma
Programación Imperativa / ADO.NET Tradicional	24,0	100,00%	0,00		
Programación Funcional / LINQ	64,0	37,50%	-0,63		
Programación Funcional / LINQ y Expresiones Lambda	32,0	75,00%	-0,25		

Figura 4.25: Resumen de Resultados de Pruebas de Uso de Memoria-Caso de Prueba 4

Para el paquete de prueba #4, con un promedio de 24,0 bytes la implementación del Paradigma Imperativo mediante el uso de objetos tradicionales de .NET registra el

menor uso de memoria. Las mediciones para LINQ y LINQ con Expresiones Lambda son: 64,0 bytes y 32,0 bytes respectivamente. Evidentemente, los valores correspondientes a las mediciones de las implementaciones funcionales LINQ y LINQ con Expresiones Lambda son inferiores en relaciones de desempeño de -0,63 y -0,25; notándose una diferencia de 50% en el desempeño entre las propias implementaciones funcionales.

EJERCICIO 5	Uso de Memoria (bytes)	% Eficiencia frente Tradicional	Relación de Desempeño	Evaluación Preliminar	
	Promedio			Desempeño	El Mejor Paradigma
Programación Imperativa / ADO.NET Tradicional	60,0	100,00%	0,00		
Programación Funcional / LINQ	60,0	100,00%	0,00		
Programación Funcional / LINQ y Expresiones Lambda	60,0	100,00%	0,00		

Figura 4.26: Resumen de Resultados de Pruebas de Uso de Memoria-Caso de Prueba 5

Curiosamente, las mediciones para el paquete de prueba #5 muestran un “empate” entre todos los paradigmas. Con valores promedios de uso de memoria de 60,0 bytes en todas las mediciones de las implementaciones; lo que puede ser interpretado en que ningún paradigma es mejor o más eficiente que otro para el ejercicio planteado.

EJERCICIO 6	Uso de Memoria (bytes)	% Eficiencia frente Tradicional	Relación de Desempeño	Evaluación Preliminar	
	Promedio			Desempeño	El Mejor Paradigma
Programación Imperativa / ADO.NET Tradicional	34,00	100,00%	0,00		
Programación Funcional / LINQ	99,00	34,34%	-0,66		
Programación Funcional / LINQ y Expresiones Lambda	50,00	68,00%	-0,32		

Figura 4.27: Resumen de Resultados de Pruebas de Uso de Memoria-Caso de Prueba 6

Las mediciones para el caso de prueba #6, nuevamente la ventaja es para el Paradigma Imperativo usando un estilo de programación tradicional. Con 34,00 bytes de uso de memoria para la ejecución del ejercicio 6, la programación imperativa se

impone ante los 99,99 bytes de la programación al estilo LINQ puro, y los 50,00 bytes de la programación funcional usando LINQ y Expresiones Lambda. En términos de eficiencia, el ratio marcado por LINQ puro es de 34,34% lo que da como resultado un valor de 0,66 veces menos eficiente en comparación a la programación tradicional. De igual forma, la implementación LINQ y Expresiones Lambda muestra un ratio de 68,00% que comparándolo contra el paradigma tradicional, se exhibe una relación de desempeño de 0,32 veces menos eficiente.

4.8.3 Análisis de los Aspectos de Programación

Para este análisis se tomarán las evaluaciones de las variables cualitativas para todos los ejercicios, y adicionalmente se añadirán al estudio los valores de la variable cuantitativa Líneas de Código. Las valoraciones se interpretan usando la tabla de valoración definida en la sección *4.7.1 . Métricas Cualitativas - Figura 4.8: Valoraciones y su significado establecido para las variables cualitativas*:

A continuación se presentan las tablas de resumen de las valoraciones cualitativas realizadas para el código correspondiente a cada implementación de los ejercicios definidos:

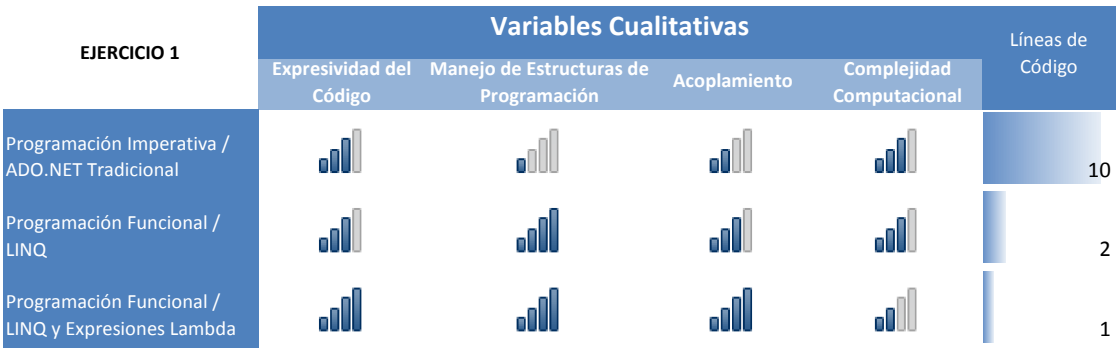


Figura 4.29: Resumen de Valoración de Variables Cualitativas y Aspectos de Programación – Caso de Prueba 1

Para el código correspondiente al Ejercicio #1, se puede observar que la implementación LINQ y Expresiones Lambda presenta las mejores evaluaciones. En tan sólo 1 línea de código, LINQ y Expresiones Lambda demuestra perfectamente la intención del algoritmo y las potencialidades programáticas que posee.

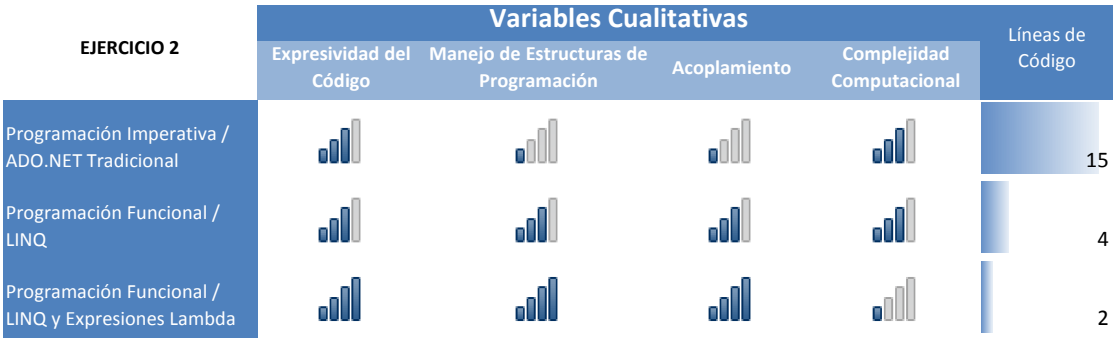


Figura 4.30: Resumen de Valoración de Variables Cualitativas y Aspectos de Programación – Caso de Prueba 2

Para la implementación del Ejercicio #2, se puede observar que las implementaciones con LINQ poseen menos líneas de código en comparación con la programación tradicional. Al igual que en el ejercicio anterior, LINQ y Expresiones Lambda logra construir la consulta con menor esfuerzo y con los índices más altos para los aspectos programáticos estudiados.

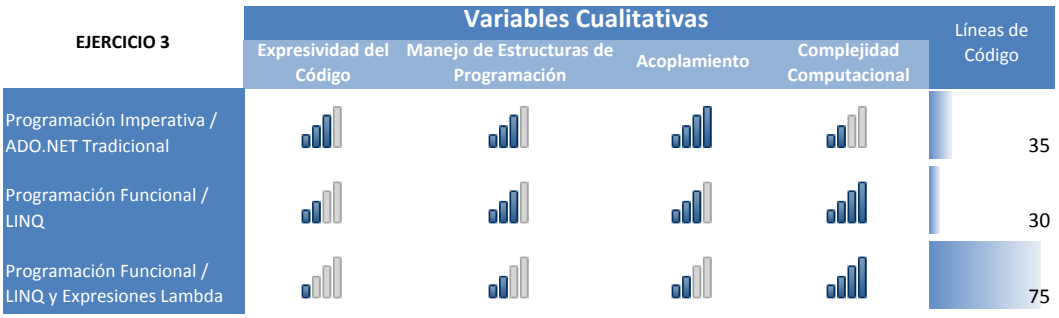


Figura 4.31: Resumen de Valoración de Variables Cualitativas y Aspectos de Programación – Caso de Prueba 3

Para la implementación del Ejercicio #3, se puede observar que la cantidad de líneas de código para todas las implementaciones se ha incrementado, esto debido a la complejidad de la consulta SQL. Sin embargo, las mejores evaluaciones de los aspectos programáticos favorecen a la programación tradicional, que en términos de estructura de algoritmo, no ha variado significativamente en comparación a los ejercicios anteriores, simplemente aumentó el números de líneas de código en función de la escritura de la consulta SQL. Contrariamente, las implementaciones usando LINQ puro y LINQ con Expresiones Lambda requirieron más esfuerzo para lograr construir la misma consulta; por ende sus evaluaciones para expresividad de código disminuyeron y la complejidad computacional de los algoritmos aumentó significativamente haciendo poco legible al código.

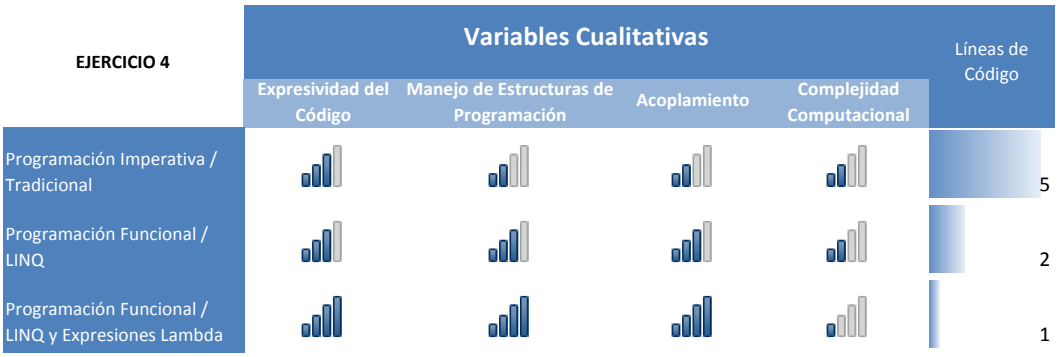


Figura 4.32: Resumen de Valoración de Variables Cualitativas y Aspectos de Programación – Caso de Prueba 4

Los siguientes ejercicios se enfocan más a tareas de programación sobre objetos en donde LINQ to Objects juega un papel primordial –a diferencia de los ejercicios anteriores donde se focalizaba más las tareas para la obtención de datos y LINQ to SQL.

Para la implementación del Ejercicio #4, las valoraciones favorecen a LINQ, donde LINQ y Expresiones Lambda presentan los mejores índices programáticos en menos líneas de código.

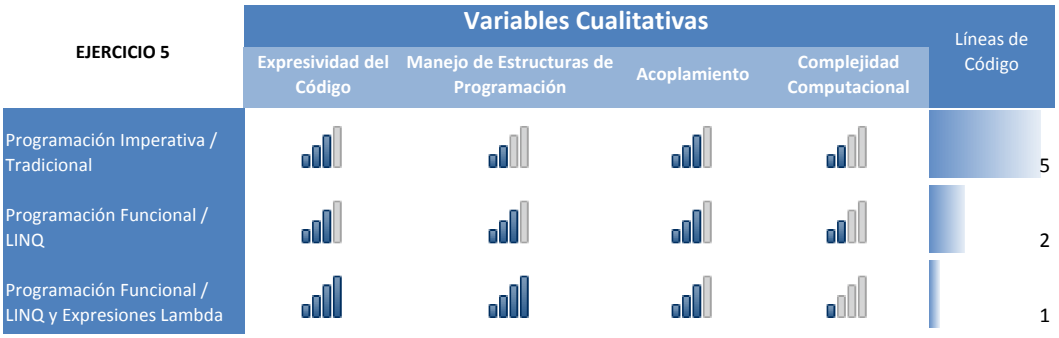


Figura 4.33: Resumen de Valoración de Variables Cualitativas y Aspectos de Programación – Caso de Prueba 5

Las valoraciones para las implementaciones del Ejercicio #5 muestran una clara ventaja -desde el punto de vista de desarrollo- al uso de LINQ y Expresiones Lambda, que con 1 línea de código demuestra el más alto indicador de Expresividad y un adecuado Manejo de Estructuras de Programación, logrando disminuir la complejidad de la implementación del algoritmo y el esfuerzo de desarrollo.

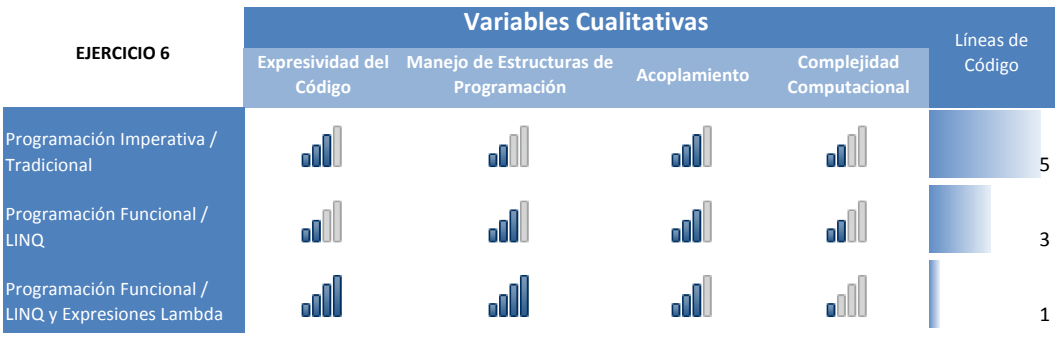


Figura 4.34: Resumen de Valoración de Variables Cualitativas y Aspectos de Programación – Caso de Prueba 6

Para las implementaciones del Ejercicio #6, LINQ y Expresiones Lambda nuevamente muestra una superioridad en cuanto al esfuerzo de programación, legibilidad del código, uso de clases y objetos programáticos, y una complejidad de desarrollo menor. En cuanto a líneas de código, se puede observar que LINQ y Expresiones Lambda

puede lograr en 1 línea, lo que en una programación tradicional se logra en 5, sugiriendo que el uso de LINQ y Expresiones Lambda aumenta la productividad al necesitarse menos esfuerzo de desarrollo.

Para obtener una perspectiva general del comportamiento de los paradigmas estudiados, se presenta a continuación un balance general de los valores obtenidos para las variables cualitativas y cuantitativas utilizadas para el análisis:

Parámetros	Paradigma Tradicional(Promedio)		Paradigma Funcional(Promedio)	
1. Cualitativas				
1.1 Expresividad de Código	3	✗	3,5	✓
1.2 Manejo de Estructuras de Programación	2	✗	3,666666667	✓
1.3 Acoplamiento	2,666666667	✗	3,333333333	✓
1.4 Complejidad Computacional	2,333333333	✗	1,833333333	✓
2. Cuantitativas				
2.1 # Líneas Código	12,5	✗	13,5	✓
2.2 Tiempo de Ejecución(ms)	1.920	✓	16.466	✗
2.3 Número Ejecuciones/seg	368,7680421	✗	936,6916231	✓
2.4 %Eficiencia	1	✗	1,4	✓
2.5 Uso de Memoria	39,3	✓	47,3	✗
2.6 Relación de Desempeño	0	✗	0,498333333	✓
2.7 Desviación Estándar	4,693333333	✗	201,1366667	✓
2.8 Coeficiente de Variación	0,06151667	✗	0,014083333	✓
Promedio	196,360241	✗	1473,239503	✓

Figura 4.35: Resumen Consolidado de Resultados para las Métricas utilizadas para el Análisis

4.9 CONCLUSIONES Y RECOMENDACIONES DEL USO DE PARADIGMAS

4.9.1. Acerca de los Aspectos de Programación

- Las Variables Cualitativas brindan una apreciación desde el punto de vista de programación. En base a las valoraciones otorgadas para estas variables, se puede apreciar que las implementaciones del Paradigma Funcional representan técnicas de programación más productivas que reducen el esfuerzo del programador en la construcción de algoritmos.
- La simplicidad y reducción en las líneas de código son características claves en las implementaciones elaboradas en LINQ con Expresiones Lambda, que otorgan al programador la capacidad de crear código más legible, compacto y optimizado, permitiéndole simplificar la resolución de problemas y reducir la complejidad en el desarrollo.
- Gracias a los operadores de consulta de LINQ, el programador cuenta con un conjunto de tareas predefinidas listas para usar, que representan útiles herramientas que ahorran esfuerzo de implementación de tareas conocidas sobre cualquier colección de objetos. De esta forma, LINQ se convierte en una válida alternativa de desarrollo que ayuda a los programadores a ser más productivos reduciendo significativamente el tiempo de codificación y mejorando notablemente la calidad del código.

4.9.2. Acerca del Tiempo de Ejecución

- En las aplicaciones que procesan tareas sobre repositorios de datos, la tasa de ejecuciones simultáneas de consultas es mucho más importante que el peso computacional del procesamiento de una consulta; es decir, el tiempo que toma procesar una de ellas no es tan importante como la cantidad que puede

ser procesada en un período finito de tiempo. Dentro de este contexto, los resultados de las variables cuantitativas de ejecución del cuadro anterior favorecen a las implementaciones del paradigma Funcional.

- En los casos estudiados, LINQ se muestra mucho más robusto en la ejecución continua de tareas de acceso a datos, haciéndolo apropiado para la implementación de aplicaciones transaccionales.
- El modelo de Mapeo de Estructuras Relacionales a Objetos de Programación de LINQ, ofrece un valor agregado al brindarle al desarrollador la capacidad de utilizar los objetos de la base de datos como objetos de programación y aplicar características de Orientación a Objetos y otros beneficios, abriendo un abanico de posibilidades dentro del código.
- El manejo de relaciones entre tablas de bases de datos y su Cardinalidad, es expresada como una propiedad de los objetos dentro del Contexto de Datos de LINQ, lo que facilita considerablemente la navegación entre objetos relacionados.
- Las tareas CRUD (Creación, Lectura, Actualización y Eliminación) sobre las tablas son características naturales del Contexto de Datos de LINQ, que en cortas líneas de código es posible manejar de forma eficiente y fiable la inserción, modificación y eliminación de objetos mediante el uso de operadores especializados y amigables que encapsulan la lógica relacional.

4.9.3. Acerca del Uso de Memoria

- Los valores promedios de la Variable de Uso de Memoria presentados en la **Figura 4.35** de Resumen Consolidado de Resultados muestran un consumo de memoria ligeramente superior de las implementaciones del Paradigma Funcional frente al valor promedio de consumo de memoria de la programación tradicional en lenguaje .NET. La razón básicamente es que el entorno de Desarrollo de .NET (o .NET Framework) está diseñado para

manejar de forma óptima los objetos nativos de .NET, de esta forma el alojamiento de objetos como secuencias ArrayList y parecidos son rápidamente interpretados y mediante un esquema mejorado de punteros, la colocación en memoria se hace de forma natural. Por otro lado, las consultas LINQ internamente son construidas en base a Árboles de Expresión, que son estructuras optimizadas para la organización jerárquica de datos. Estas estructuras constituyen una representación eficiente de alojamiento de datos que facilita la evaluación e interpretación del compilador; pero el alojamiento y manejo de estas estructuras en memoria es un poco más complicado para el .NET Framework. A medida que la consulta crece, el número de ramas del árbol aumenta, estableciendo más punteros para mantener la ligación de los datos; en otras palabras, si la consulta LINQ es mayor o más compleja, mayor es el árbol de expresión correspondiente. Cada rama del árbol puede ser un dato específico u otro árbol de expresión, por lo que la complejidad en el manejo aumenta.

Sin embargo, el desempeño de LINQ con el uso de Expresiones Lambda reduce notablemente el consumo de memoria mediante un esquema de estructuras precompiladas que son mucho más fáciles de manejar y alojar en la memoria. Como consecuencia el promedio de uso de memoria es semejante al mostrado por la implementación tradicional .NET. Con este aspecto en mente y tomando en cuenta las ventajas del uso de LINQ en la programación, su uso en la implementación de algoritmos es ampliamente recomendada ya que brinda enormes beneficios a un costo de recursos relativamente menor para la complejidad del problema a resolver.

4.9.4. Recomendaciones Generales del Uso Adecuado de LINQ de acuerdo al Escenario y Necesidad

En base al análisis realizado, se ha logrado identificar algunos escenarios en los hay que tomar en cuenta algunos aspectos para el manejo adecuado de LINQ. A continuación se detallan algunas recomendaciones generales:

- **Para el Acceso a Datos:**
    - El uso de LINQ es bastante apropiado para la construcción de páginas de catálogos que utilizan consultas planas sobre las tablas de una base de datos, es decir consultas en donde se obtiene todos los campos de la tabla.
    - Si en la definición de la consulta intervienen aspectos de negocio que hace que la complejidad de la consulta aumente; por ejemplo: múltiples INNER JOINs, Subconsultas, invocación de funciones o procedimientos almacenados anidados o campos calculados; es recomendable utilizar un procedimiento almacenado para obtener la información.
    - LINQ no es apropiado para la elaboración de consultas entre bases de datos vinculadas. Los contextos de datos de LINQ poseen una única conexión hacia una base de datos.
    - Es altamente recomendable el uso de Operadores Estándar y Expresiones Lambda. La naturaleza de ser objetos precompilados aceleran la ejecución haciendo que el rendimiento en general mejore al procesar un lote de consultas.

- **Para el Manejo de Objetos en la Lógica de Negocio:**

  LINQ es altamente recomendado para el manejo de objetos de programación y tareas sobre colecciones de objetos. Mediante el uso de LINQ se evitan el

uso de bucles y recorridos simplificando enormemente la codificación. El uso de LINQ to Objects es recomendado en todas las capas de la aplicación: Acceso a Datos, Lógica de Negocio y Presentación.

CAPÍTULO V

"CASO APLICATIVO: HADE (Sistema de Gestión de Proyectos de la ESPOCH), el Sistema Multiparadigma"

5.1 INTRODUCCIÓN

Actualmente, la Escuela Superior Politécnica de Chimborazo cruza por una etapa de madurez tecnológica e informática que gracias a la eficiencia de su infraestructura de soporte (red institucional, dispositivos, elementos de red activos y pasivos, PCs, etc.), posee una gran cantidad de aplicaciones informáticas construidas con tecnologías heterogéneas encargadas de brindar beneficios a la comunidad politécnica. Dentro de éste contexto La Comisión de Proyectos y Transferencia de Tecnología (COMPROTEC), de la ESPOCH, brinda un apoyo fundamental para la institución, mediando la organización y presentación de proyectos de Investigación y otros que aportarán al crecimiento de la ESPOCH.

Sin embargo, ésta importante Unidad o departamento, no posee una herramienta informática que agilice los procesos al interior de la misma. Por lo cual se ha visto la imperiosa necesidad de implementar un Portal Web que brinde a investigadores, directores, empresas, instituciones y público en general -dentro y fuera del país- que se

convierta en una útil herramienta de gestión, seguimiento y emprendimiento de proyectos en beneficio de la ESPOCH y del país.

Este capítulo se enfoca principalmente al análisis de la problemática y de las alternativas de solución, al diseño arquitectónico y funcional, al estudio comparativo de tecnologías, y especificaciones técnicas de desarrollo de software para lograr construir una aplicación fiable, de fácil uso, acoplable, compatible y segura para cualquier usuario involucrado en la gestión.

5.2 PRESENTACIÓN

El proyecto propuesto tiene por objetivo construir un sitio web de última generación que integrado a la infraestructura de tecnologías de la ESPOCH, se convierta en una óptima y eficiente herramienta para la gestión de proyectos que maneja la Unidad COMPROTEC.

Mediante la utilización de HADE (Sistema de Gestión de Proyectos) – COMPROTEC se conseguirá:

- Mostrar una vista pública detalla acerca de los proyectos para motivar la participación de empresas e instituciones nacionales y extranjeras.
- Sitio de creación de proyectos de investigación y la integración de investigadores al mismo.
- De cada proyecto se mostrará sus características, actividades, recursos y presupuestos.
- Seguimiento de Proyectos, donde se podrá visualizar el estado actual del proyecto, las actividades realizadas, el presupuesto, galerías de fotografías

- Sitio de administración, en el cual los administradores pueden manejar datos de configuración y datos del sistema como: áreas, sectores, Instituciones, contactos, unidades de medidas temporales, etc.

Además, Se construirán un conjunto de componentes englobando las mejores prácticas y patrones de seguridad dentro de sus aplicaciones.

Al mismo tiempo se brindará a las personas involucradas en la Unidad de COMPROTEC contar la facilidad de contar con información útil y eficaz a partir de la automatización de la automatización de los procesos que se lleva a cabo actualmente mejorando su gestión, planificación y administración. La adecuada aplicación del nuevo sistema evitara pérdidas de tiempo, esfuerzo y recursos económicos, así como permitirá la consecución de los objetivos y metas propuestas por parte del personal que conforma la comisión.

Conjuntamente, se proporcionará una herramienta de Administración Web para el control de los usuarios dependiendo del tipo de perfil al que pertenecen.

5.3 METODOLOGÍA DE DESARROLLO

La Metodología de desarrollo a utilizar es Microsoft Solutions Framework (MSF). MSF es un conjunto de modelos, principios y guías para diseñar y desarrollar soluciones empresariales en una manera que asegura que todos los elementos de un proyecto, personas, procesos, herramientas puedan ser satisfactoriamente administrados. MSF también provee prácticas mejoradas para el planeamiento, diseño, desarrollo y despliegue satisfactorio de aplicaciones empresariales.

5.4 VISIÓN

Posterior a la implementación del Sistema de Gestión de Proyectos “HADE”, la Escuela Superior Politécnica del Chimborazo a través de COMPROTEC brindará una herramienta útil, eficiente, eficaz, de fácil acceso y manejo para difundir la información coherente de los Proyectos/Concurso a las personas, instituciones, empresas nacionales e internacionales interesadas en apoyar un proyecto; así, como también se mejorará la toma de decisiones mediante la administración del monitoreo, seguimiento y evaluación de proyectos existentes.

5.5 METAS DE DISEÑO

Las metas de diseño describen las características operacionales de las infraestructuras y los sistemas empresariales.

5.5.1 Rendimiento

El rendimiento debe ser el adecuado para el tipo de conexiones a Internet utilizadas por los usuarios del sistema, las páginas de los sitios Web serán optimizadas en tamaño para proveer rapidez en el intercambio de la información. Adicionalmente, se emplearán tecnologías y mecanismos que permitirán que el desarrollo y ejecución sea más ágil, optimizando recursos.

5.5.2 Disponibilidad

La disponibilidad del sistema debe ser las 24 horas al día, y 7 días a la semana, debido a que el ingreso al sitio se podrá hacer desde cualquier lugar y en cualquier momento.

5.5.3 Fiabilidad

Debido a la necesidad de disponibilidad 24x7, el sistema deberá contar con información consistente y actualizada que es extraída de los repositorios de datos y presentada a través de nuestras interfaces.

5.5.4 Escalabilidad

Debido al crecimiento del número de usuarios, el sistema será capaz de escalar horizontalmente y verticalmente.

5.5.5 Seguridad

La seguridad del Sitio se maneja en varios niveles. El sistema contará con un primer nivel de acceso denominado “Sitio Público”, dentro del mismo se presentará información a usuarios invitados (no restringidos) acerca de los proyectos actuales que necesitan patrocinio y apoyo de instituciones, empresas o recursos para que se ejecuten. (Proyectos Pendientes)

El siguiente nivel de Acceso, denominado “Sitio de Gestión de Proyectos” contendrá interfaces mediante las cuales los usuarios con perfiles de Director e Investigador podrán crear proyectos y visualizar su información, crear actividades, asignar recursos, establecer presupuestos y demás información concerniente a los proyectos en ejecución.

El último nivel de acceso denominado “Sitio de Administración” está dedicado a los usuarios administradores, los cuales podrán ejecutar tareas de administración y configuración de datos del sistema, como: áreas, sectores, estados de proyectos,

instituciones, unidades de medición tiempo, usuarios, perfiles o roles, permisos de acceso, entre otros.

Figura 5.1: Esquema de Seguridad del Sistema

5.5.6 Interoperabilidad

El sistema es una plataforma aislada que no interactúa con otros sistemas, este posee su propio nivel de seguridad y autentificación en el caso de que los usuarios sean:

investigadores y administradores que deseen acceder al sistema; también cuenta con su propia base de datos.

5.6 FASE INTRODUCTORIA

5.6.1 Introducción

La Fase Introductoria tiene como objetivo obtener una perspectiva clara de las metas y objetivos del proyecto en base a la problemática que se desea resolver. Durante ésta fase se analiza los escenarios de desenvolvimiento, se definen los problemas que el proyecto pretende resolver y se visualiza el camino que tomará el sistema.

5.6.2 Análisis de la Problemática

5.6.2.1 Antecedentes

La Comisión de Proyectos y Transferencia de Tecnología (COMPROTEC), de la ESPOCH, se encarga de presentar al Consejo de Investigación y Desarrollo el plan operativo anual, identificar, formular y gestionar proyectos de prestación de servicios, consultoría y asesoría en coordinación con las unidades académicas; establecer y mantener la cooperación interinstitucional con empresas públicas y privadas para el desarrollo científico y tecnológico; colaborar con organismos, instituciones, empresas públicas y privadas, nacionales e internacionales para la investigación.

5.6.2.1.1 LINEAMIENTOS GENERALES PARA LA INVESTIGACIÓN CIENTÍFICA Y TECNOLÓGICA EN LA ESPOCH

En base al análisis realizado de conceptos operativos sobre la investigación, a los tipos de investigación reconocidos en nuestro país por el CONESUP, a los programas y

proyectos que merecen financiamiento por parte de la entidad rectora de la ciencia en Ecuador la SENACYT, a la problemática que atraviesa esta función universitaria y a efectos de orientar el quehacer científico y tecnológico, la Escuela Superior Politécnica de Chimborazo a través de los Centros de Investigación de las Facultades y de las instancias institucionales auspiciará proyectos que respondan a los siguientes lineamientos generales:

1. Que planteen soluciones claras a problemas de carácter nacional, regional y local
2. Que faciliten la integración o la vinculación de la institución con los sectores sociales y/o productivos del país, la región y la provincia, y respondan a sus reales necesidades.
3. Que se encuentren articulados a las políticas y líneas de investigación institucional y nacional.
4. Que se trabajen bajo la perspectiva del desarrollo sustentable.
5. Que las acciones previstas para su ejecución sean compatibles con los valores de la ética y la moral.
6. Que garantice la formación de equipos interinstitucionales y faciliten trabajos conjuntos con otros centros de investigación públicos y/o privados del país y del extranjero.
7. Que incorporen al trabajo las dimensiones de la interdisciplinaridad, multidisciplinaridad y transdisciplinaridad.
8. Que su planificación contemple acciones de difusión y aplicación del conocimiento bajo los conceptos de popularización de la ciencia y pueda ser aprovechada socialmente.
9. Que contribuyan a innovar procesos, elevar la productividad y los niveles de competitividad del sector productivo.
10. Que contribuyan al fomento del espíritu emprendedor y al autoempleo.
11. Que permitan la articulación investigación, docencia y extensión, de acuerdo al modelo educativo de la ESPOCH.

12. Que aporten a la formación de investigadores, incorporando estudiantes que desarrollen trabajos investigativos a nivel de cátedras, pasantías, tesis de grado y postgrado.

13. Que faciliten la consecución de recursos externos.

14. Que aporten al mejoramiento de la infraestructura científica-académica (equipos, instrumentos, laboratorios, bibliografía etc.) de las unidades académicas que forman la ESPOCH

15. Que respondan a las líneas de investigación definidas para cada unidad académica.

16. Que se encuentren articulados a la misión y visión institucional y consecuentemente a la misión y visión de la universidad ecuatoriana.

17. Que se realicen bajo metodologías cuantitativas y/o cualitativas.

18. Que sean formulados con rigor científico y sistematizados adecuadamente

5.6.2.1.2 POLITICAS Y ESTRATEGIAS GENERALES PARA LA INVESTIGACIÓN CIENTÍFICA Y TECNOLÓGICA EN LA ESPOCH

La Escuela Superior Politécnica de Chimborazo con la finalidad de contribuir al fortalecimiento de la función investigativa establece las siguientes políticas y estrategias institucionales.

A. POLITICAS

1. Fortalecer el fondo de contraparte institucional para proyectos de investigación y transferencia de tecnología.
2. Reconocer en el escalafón docente y administrativo los trabajos de investigación ejecutados por los servidores institucionales siempre y cuando estos respondan los lineamientos de investigación institucional

3. Desarrollar capacitación permanente para la formación de investigadores, bajo los conceptos de educación continúa y postgrado

4. Establecer una política de difusión y transferencia de tecnología de la investigación

5. Auspiciar la presentación de proyectos previa certificación de la Comisión de Proyectos y Trasferencias Tecnológicas.

6. Monitorear y evaluar los proyectos a través de la COMPROTEC

7. Crear una infraestructura mínima para la investigación actualizando laboratorios y espacios útiles para la práctica investigativa.

8. Auspiciar la ejecución de diferentes tipos de investigación de acuerdo a la oferta que realizan organismos financieros nacionales e internacionales

9. Auspiciar la ejecución de proyectos en el ámbito educativo

B. ESTRATEGIAS

1. Participar en concursos nacionales e internacionales para la investigación que permitan lograr apoyos financieros-económicos

2. Formar redes de investigación.

3. Establecer alianzas estratégicas con el sector productivo y de servicios.

4. Establecer alianzas estratégicas con el gobierno nacional, gobiernos locales, ONGS y otro tipo de organizaciones.

5. Crear un banco de proyectos institucionales y/o por unidades académicas para procurar financiamiento.

6. Definir un equipo procurador de fondos para el financiamiento de proyectos.

7. Crear incentivos no económicos para investigadores y estudiantes que destaquen en el aspecto investigativo y de la transferencia de tecnología

8. Promover y financiar la participación de los investigadores en eventos nacionales e internacionales

5.6.2.1.3 LINEAS PRIORITARIAS DE INVESTIGACIÓN

Las necesidades y problemas sociales y productivos son tantos y tan diversos que es difícil la tipificación de sectores y áreas, sin embargo para la elaboración de las líneas de investigación de la ESPOCH, se han considerado los sectores y áreas que han sido declarados como prioritarios por la SECRETARIA NACIONAL DE CIENCIA Y TECNOLOGIA en el año 2006, para el financiamiento de proyectos con fondos CEREPS y que responden a la política nacional de ciencia, tecnología e innovación Ecuador 2005. Además se han considerado las líneas de investigación definidas en forma participativa con los representantes de los sectores sociales y productivos de la provincia de Chimborazo en los talleres realizados el 8 de mayo del 2007, a los cuales se dieron cita 30 técnicos y funcionarios en representación de los gobiernos locales, gobernación, ONGS y empresa privada, cuyos resultados (anexo 3) en la práctica son tópicos que coinciden con los de la SENACYT.

En el año 2008 la SENACYT ha declarado para el trabajo investigativo nacional campos de acción y las áreas de trabajo que se citan en el literal C y D. y que son muy similares a las 2006 Y 2007, por lo que los lineamientos institucionales se fundamentan en los siguientes sectores y áreas del conocimiento

A. SECTORES

1. Salud y Nutrición
2. Seguridad Alimentaria
3. Educación
4. Economía y Producción
5. Vivienda
6. Recursos Naturales y Ambiente

B. ÁREAS DEL CONOCIMIENTO

1. Ciencias Básicas y de materiales
2. Ciencias Sociales y Humanas
3. Medio Ambiente y Hábitat
4. Biotecnología
5. Desarrollo Agropecuario
6. Salud
7. Estudios Científicos en Educación
8. Ciencias del Mar y de la Tierra
9. Ciencias de Computación y TICs
10. Energía y Minería

C. CAMPOS DE ACCION 2008

1. Desarrollo humano y social
2. Fomento agropecuario y agricultura sostenible
3. Sectores estratégicos: agua, petróleo, energía, minería
4. Investigación en el sector publico
5. Biotecnología
6. Fomento productivo
7. Tecnologías de la información y comunicación

D. AREAS 2008

1. Energía
2. Recursos naturales
3. Medio ambiente
4. Tecnologías de la información y comunicación
5. Desarrollo humano y social
6. Fomento agropecuario y agricultura sostenible

Por la importancia y pertinencia, cada unidad académica con sus escuelas y carreras ha establecido sus líneas de investigación basadas en las políticas establecidas por la SENACYT, lo han realizado en base a áreas académicas a través de las cuales desarrollan su actividad académica y en otros casos han elaborado las líneas en función de procesos productivos, además han agregado otras líneas que demanda la naturaleza propia de su trabajo y lo que demandan los sectores sociales productivos de la provincia de Chimborazo, que es la zona de influencia más cercana a la ESPOCH.

5.7 FASE DE PLANIFICACIÓN

5.7.1 Introducción

La Fase de Planificación tiene como objetivo enumerar las especificaciones funcionales y requerimientos del sistema propuesto.

5.7.2 Especificación de Requerimientos del Sistema

5.7.2.1 INFORMACIÓN DE NECESIDADES TECNOLOGICAS

▼ **Tabla XLIX: Requerimientos para las Necesidades Tecnológicas del Sistema**

Id.	Tipo de Requerimiento	Requerimiento	Detalle
REQ-1	Infraestructura	Implantación y Despliegue sobre la infraestructura de soporte actual	La solución a implementarse podrá ser desplegada sobre la infraestructura de red y servidores que actualmente posee la ESPOCH; así como también podrá utilizar la infraestructura de red o el servidor de Chimborazo Emprered integrándose al conjunto de recursos tecnológicos presentes aprovechando al máximo sus capacidades y bondades.
REQ-2	Arquitectura	Consolidación de Información.	Como parte de la solución se creara una base de datos para que la institución cuente con el registro de todas las investigaciones que se realizan a nivel institucional. La base de datos será implementada en SQL Server 2000. La solución deberá estar diseñada de tal forma que se pueda acceder a la información de los proyectos necesaria, organizarla y consolidarla en función a un diseño arquitectónico y de base de datos eficiente.
REQ-3	Arquitectura	Integración con infraestructura de Autenticación	El sistema deberá integrarse con una infraestructura de autenticación mediante una interfaz programática que brinde el acceso respectivo a los miembros que accedan al sistema (Administrador, Investigador, Director y Usuario).

5.7.2.2 Información del modulo de seguridad

▼ **Tabla L: Requerimientos para el modulo de Seguridad**

Id.	Tipo de Requerimiento	Requerimiento	Detalle
REQ-4	Almacenamiento	Registro en la Base de Datos de los perfiles que se les asignara a los usuarios	El sistema debe ser capaz de almacenar los diferentes perfiles que se les asignará al administrador, investigadores y usuarios que manipulen la aplicación; asignándoles los diferentes roles y permisos de navegación
REQ-5	Funcionalidad	Acceso a la Base de datos por perfiles de Usuario creados.	El sistema debe ser capaz de permitir e identificar el acceso a la Base de datos dependiendo del usuario que acceda al sistema.
REQ-6	Funcionalidad	Administración del sistema	El administrador del sistema estará en la capacidad de tener el control absoluto de la información registrada en la base de Datos; así como también tendrá todos los permisos de navegación.
REQ-7	Funcionalidad	Acceso a la información por perfil de usuario	El sistema deberá permitir el acceso a la información acorde al perfil asignado restringiendo la navegabilidad por el tipo de usuario.

5.7.2.3 Información del módulo de administración

▼ **Tabla LI: Requerimientos para el modulo de Administración**

Id.	Tipo de Requerimiento	Requerimiento	Detalle
REQ-8	Funcionalidad	Dar soporte para el mantenimiento del sistema.	El sistema debe ser capaz de dar soporte en el mantenimiento a las tablas de la base de datos existente y al catalogo del sistema

5.7.2.4 Información del Modulo de Proyectos

▼ **Tabla LII: Requerimientos para el modulo de Proyectos**

Id.	Tipo de Requerimiento	Requerimiento	Detalle
REQ-9	Almacenamiento	Registro en Base de datos de todos los proyectos que se realizan en la ESPOCH.	El sistema debe ser capaz de almacenar todos los proyectos que se realizan en la institución para llevar el control, seguimiento y ejecución de los mismos.
REQ-10	Almacenamiento	Registro en la Base de Datos del administrador, investigadores e instituciones involucradas en los proyectos	El sistema debe de ser capaz de almacenar la información del administrador y de todos los investigadores e instituciones que forman parte de cada uno de los proyectos.

REQ-11	Gestionar	Gestionar los proyectos que posee la ESPOCH.	El sistema deberá permitirle al administrador gestionar la información de todos los proyectos
REQ-12	Mostrar	Mostrar la información de los proyectos existentes	El sistema deberá mostrar la información de los proyectos a todos los usuarios que ingresen al sistema.
REQ-13	Mostrar	Mostrar la información de los investigadores	El sistema deberá permitir mostrar los investigadores que posee cada uno de los proyectos; así como también los roles que desempeñan en el mismo.
REQ-14	Mostrar	Mostrar las actividades, recursos y sub-actividades de los proyectos	El sistema deberá mostrar todas las actividades, sub-actividades y recursos de los proyectos existentes.

5.8 FASE DE ANÁLISIS

5.8.1 Introducción

La Fase de Análisis tiene como objetivo describir casos de uso del sistema y sus actores, definir el modelo conceptual de la base de datos y construir diagramas lógicos.

5.8.2 Identificación de Actores

▼ **Tabla LIII: Identificación de los Actores del Sistema**

No.	Nombre	Perfil	Tipo de Acceso	Descripción
1	Usuarios/Invitado (Público en General, Instituciones, empresas nacionales e internacionales)	Miembros de la Comunidad	Web	Usuario que puede navegar en la aplicación Web y visualizar en detalle cada uno de los proyectos existentes sin la necesidad de identificarse.
2	Administrador del Sistema “HADE”	Administrador	Windows/Web	Usuario que debe ser autentificado ya que es el encargado de las gestiones de configuración, seguridad y administración del Sistema de Gestión de proyectos “HADE”. empleado en la Unidad de COMPROTEC de la ESPOCH.
3	Investigadores	Investigador	Windows/Web	Usuario que se debe autentificar y que podrá gestionar los proyectos realizados por él; visualizar actividades y sub-actividades que se siguen alcanzando para culminar con éxito los proyectos dados.
4	Directores	Director	Windows/Web	Usuario que deberá autentificarse para poder realizar cambios

				en el proyecto que dirige

5.8.3 Definición de Casos de Uso

Un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico

5.8.3.1 CASO DE USO GENERAL O DE ALTO NIVEL

▼ Tabla LIV: Caso de Uso General: Uso del Sistema "HADE"

CASO DE USO:	USO DEL SISTEMA HADE
ACTORES:	Administrador, Investigadores, Directores y Usuarios/Invitados
TIPO:	Alto nivel Esencial
PROPÓSITO:	Automatizar el proceso manual que se lleva actualmente a cabo para realizar el control, monitoreo, seguimiento y Evaluación de los Proyectos; mediante el Sistema de Gestión de Proyectos de la ESPOCH denominado "HADE"; éste servirá como una contribución para generar información útil y eficaz a partir de los datos que posee COMPROTEC mejorando su gestión, planificación y administración.
DESCRIPCIÓN:	El sistema de Gestión de Proyectos "HADE" permitirá a la comunidad (administrador, investigadores, directores y usuario/invitado), visualizar mediante una Aplicación Web los proyectos existentes que se realizan en la ESPOCH. En el caso de que el usuario sea un administrador se debe autentificar para ingresar al Sistema ingresando su ID y Contraseña, inmediatamente el sistema lo valida interactuando con la base de datos propia de la aplicación; un administrador será el encargado de la configuración, seguridad y administración completa de la Aplicación otorgando los perfiles, permisos y reglas necesarios para el acceso de cada usuario, también

	podrá realizar búsquedas de usuarios, visualizar todos los Catálogos del Sistema, ingresar, registrar, eliminar, actualizar datos del mismo. En el caso de que el usuario sea un investigador al igual que un administrador se debe autentificar para ingresar a la Aplicación ingresando su ID y Contraseña, inmediatamente el sistema lo valida interactuando con la base de datos propia del Sistema; un investigador podrá observar el listado general de todos los proyectos, el detalle de cada uno de los mismos, también podrá visualizar las actividades y sub-actividades que se llevan a cabo para finalizar el proyecto. En el caso de que el usuario sea un director, el mismo deberá ingresar su ID y Contraseña para autentificarse, el Sistema verifica sus datos ingresados y los valida interactuando con la base de datos del mismo; un director tendrá los permisos para poder editar la información pertinente a sus proyectos, visualizar los proyectos de manera general y detallada, actividades y sub-actividades que se siguen cumpliendo en el transcurso del desarrollo del proyecto.

5.8.3.2 CASOS DE USO PARTICULARES

5.8.3.2.1 Caso de Uso – Gestión de Información del Sistema

▼ **Tabla LV: Caso de Uso: Gestión de Información del Sistema**

Caso de Uso	CU-01
Nombre	Gestión de Información del Sistema
Actores	Administrador del Sistema HADE (Sistema de gestión de Proyectos)
Función	Iniciar Sesión, Visualización, Ingreso, Edición, Eliminación de datos del Sistema
Descripción	Describe brevemente la interacción entre el usuario politécnico y el Sistema HADE (Sistema de Gestión de Proyectos).
Algoritmo	1. El Usuario Arriba al Sitio Web 2. La página de Autenticación solicita las credenciales del usuario. 3. El Usuario ingresa sus credenciales. 4. La pagina de autenticación valida las credenciales y en caso exitoso, inicia sesión. 5. El Sitio realiza gestión de autorización en base a perfiles. EL usuario ingresa al sitio. 6. El Administrador solicita pagina de catalogo de tablas del

	sistema. 7. El sistema presenta página solicitada. 8. El Usuario selecciona una tabla del sistema del catalogo. 9. El sistema muestra la información dentro de la tabla. 10. El usuario decide añadir, editar o eliminar u registro de la tabla
Condiciones	

5.8.3.2.2 Caso de Uso – Gestión de Proyectos

▼ **Tabla LVI: Caso de Uso: Gestión de Proyectos**

Caso de Uso	CU-02
Nombre	Gestión de Proyectos
Actores	Administrador , Director, Investigador, Invitado
Función	Iniciar Sesión. Creación, Visualización, Ingreso, Edición, Listado de Proyectos
Descripción	Describe brevemente la interacción entre los usuarios y el Sistema HADE (Sistema de Gestión de Proyectos).
Algoritmo	1. El Usuario (Administrador, Director, Investigador, Invitado) arriba al Sitio Web. 2. El Usuario solicita ver listado de proyectos. 3. El sistema presenta la página solicitada. 4. El Usuario solicita ver información detallada del proyecto. 5. El sistema presenta la página con la información detallada del proyecto seleccionado. 6. El Usuario navega por las fichas de información del proyecto. 7. El Usuario (Administrador, Director, Investigador) solicita página de inicio de sesión para tareas de Creación, Modificación, Actualización de proyectos. 8. El sistema presenta página de Inicio de sesión. 9. La página de Autenticación solicita las credenciales del usuario. 10. El Usuario ingresa sus credenciales. 11. La pagina de autenticación valida las credenciales y en caso exitoso, inicia sesión. 12. El Sitio realiza gestión de autorización en base a perfiles. EL usuario ingresa al sitio. 13. El Administrador solicita página de Creación de Proyectos. 14. El sistema presenta página. 15. El Administrador ingresa datos del nuevo proyecto. 16. El sistema procesa información y notifica el éxito del proceso. 17. El Director solicita ver información detallada de un proyecto. 18. El sistema presenta la página con la información detallada del proyecto seleccionado. El usuario navega a través de las fichas

	de información del proyecto. 19. El Director edita información del proyecto seleccionado. 20. El sistema procesa información y notifica el éxito del proyecto. 21. El usuario (Administrador, Director, Investigador) solicita ver información de las Tareas y Sub-Tareas de un proyecto determinado. 22. El sistema presenta información.
Condiciones	

5.8.3.2.3 Caso de Uso – Gestión de Seguridad Comprotec

▼ **Tabla LVII: Caso de Uso: Gestión de Seguridad COMPROTEC**

Caso de Uso	CU-03
Nombre	Gestión de Seguridad Comprotec
Actores	Administrador del Sistema HADE (Sistema de gestión de Proyectos)
Función	Iniciar Sesión, Visualización/Creación de Perfiles, Visualización/Creación de Usuarios, Configuración de Acceso.
Descripción	Describe brevemente la interacción del usuario administrador con el sistema para la gestión de tareas administrativas
Algoritmo	1. El Usuario Arriba al Sitio Web 2. La página de Autenticación solicita las credenciales del usuario. 3. El Usuario ingresa sus credenciales. 4. La pagina de autenticación valida las credenciales y en caso exitoso, inicia sesión. 5. El Usuario visualiza perfiles de Usuario.
Condiciones	

5.8.4 Diagramas de Casos de Uso

5.8.4.1 DIAGRAMA DE CASO DE USO GESTION DE PROYECTOS

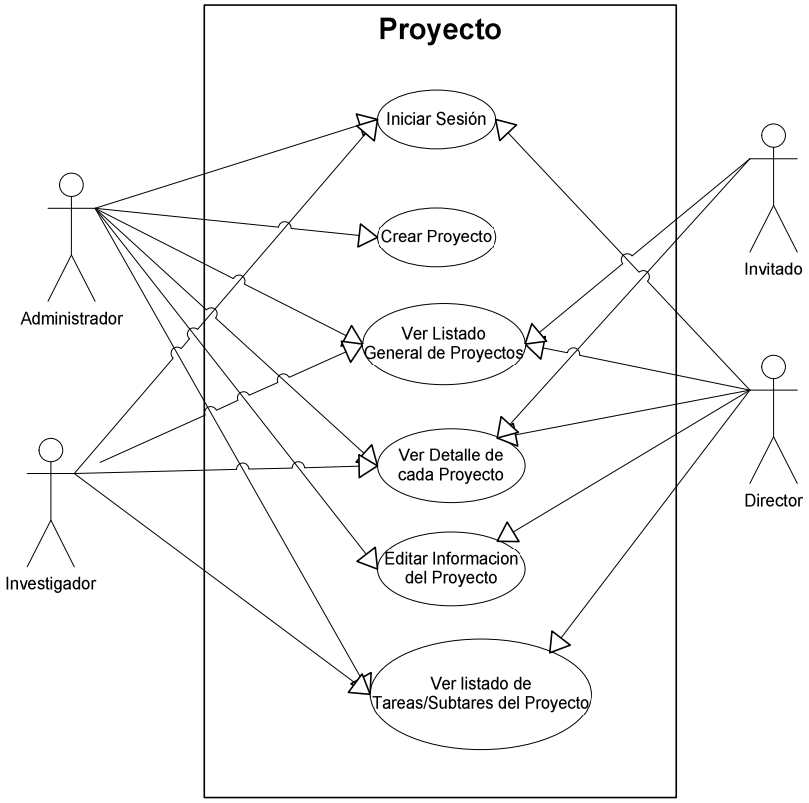


Figura 5.2: Diagrama de Caso de Gestión de Proyectos

5.8.4.2 DIAGRAMAS DE CASOS DE USO PARTICULARES

Inicio de Sesión

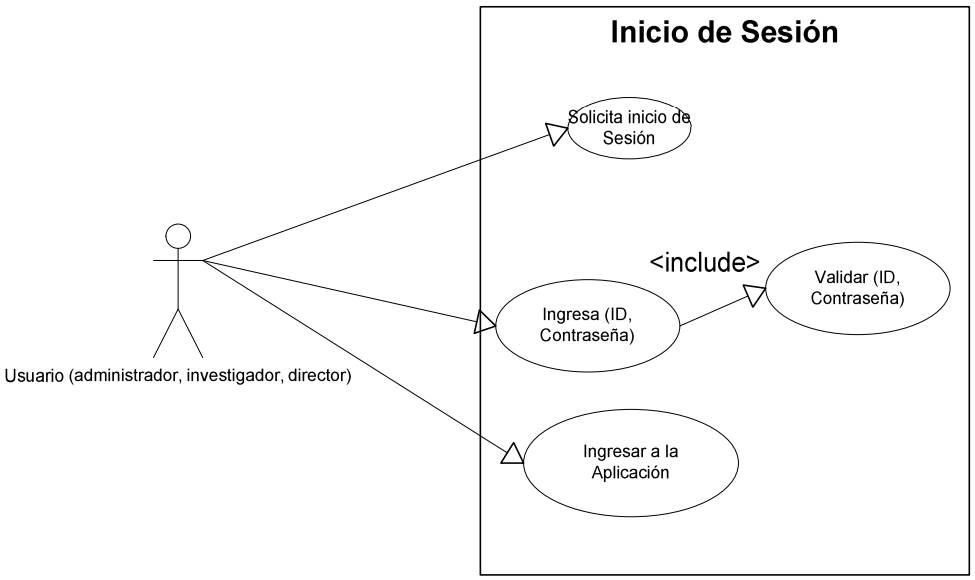


Figura 5.3: Diagrama de Caso de Uso del Inicio de Sesión

Gestión de Información del Sistema

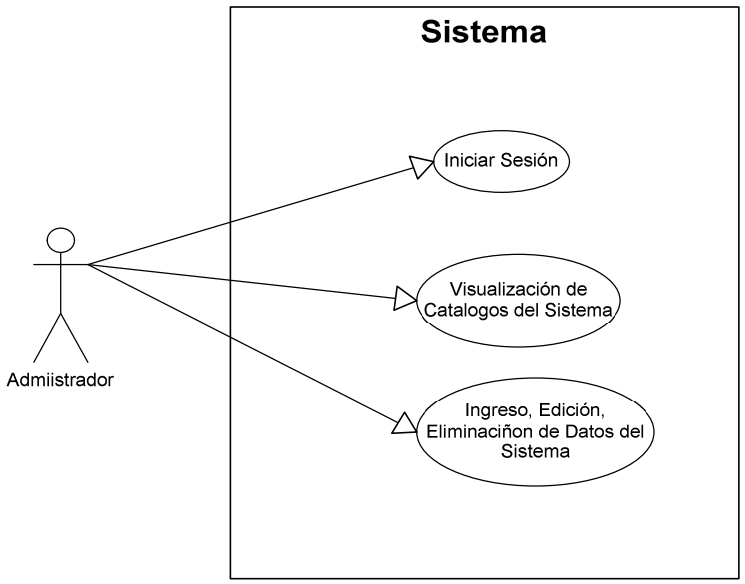


Figura 5.4: Diagrama de Caso de Uso de Gestión de Información del Sistema

Gestión de Seguridad Comprotec

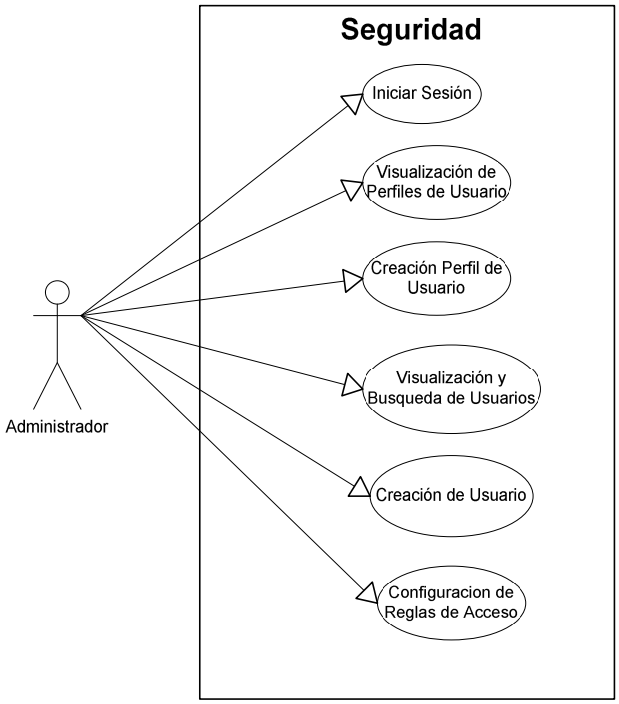


Figura 5.5: Diagrama de Caso de Uso de Gestión de Seguridad del Sistema

5.8.5 Construcción del Modelo Conceptual

5.8.5.1 Diagrama Conceptual Datos Generales del Proyecto e Instituciones Participantes

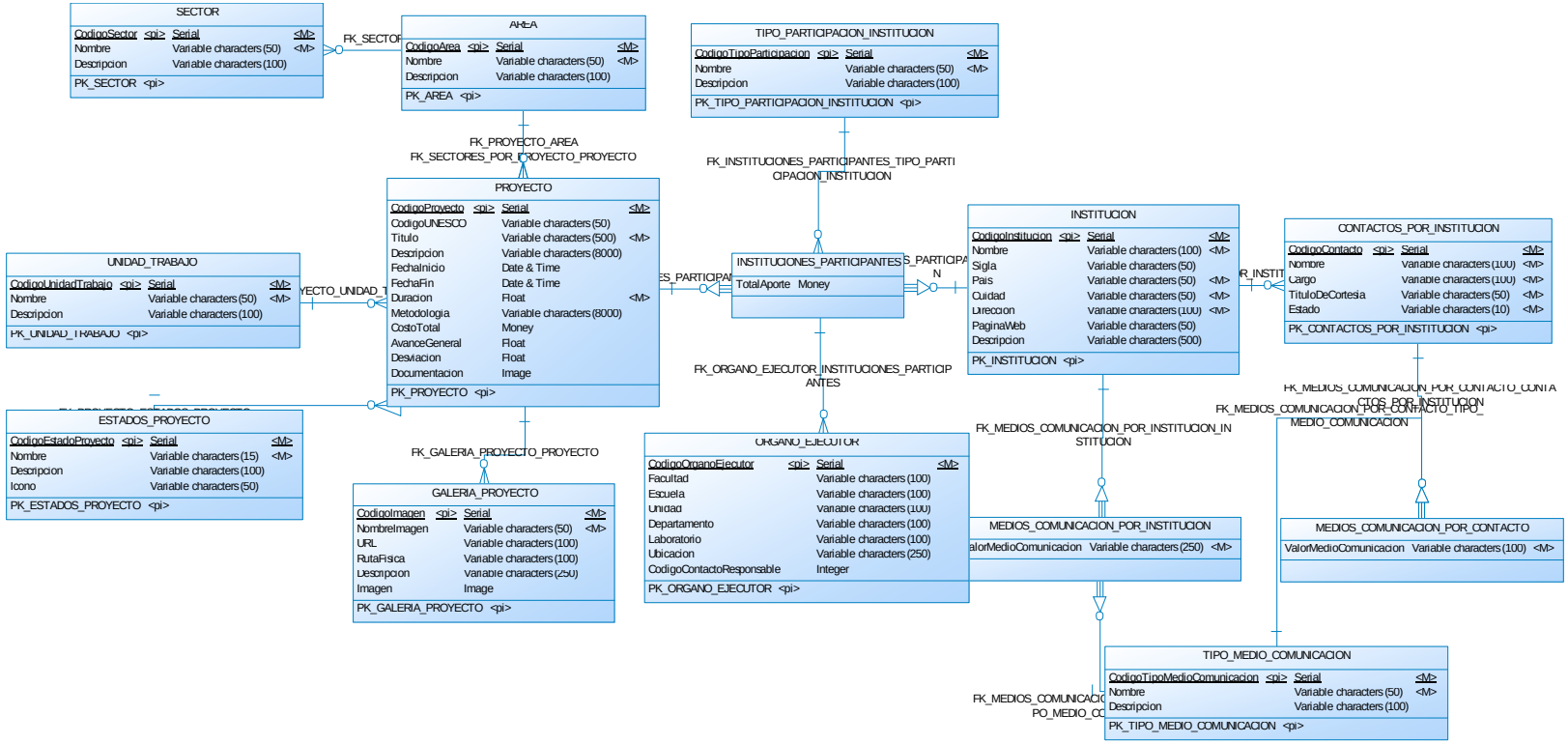


Figura 5.6: Diagrama Conceptual de Datos Generales del Proyecto e Instituciones Participantes

5.8.5.2 Diagrama Conceptual de actividades y Sub-Actividades del Proyecto

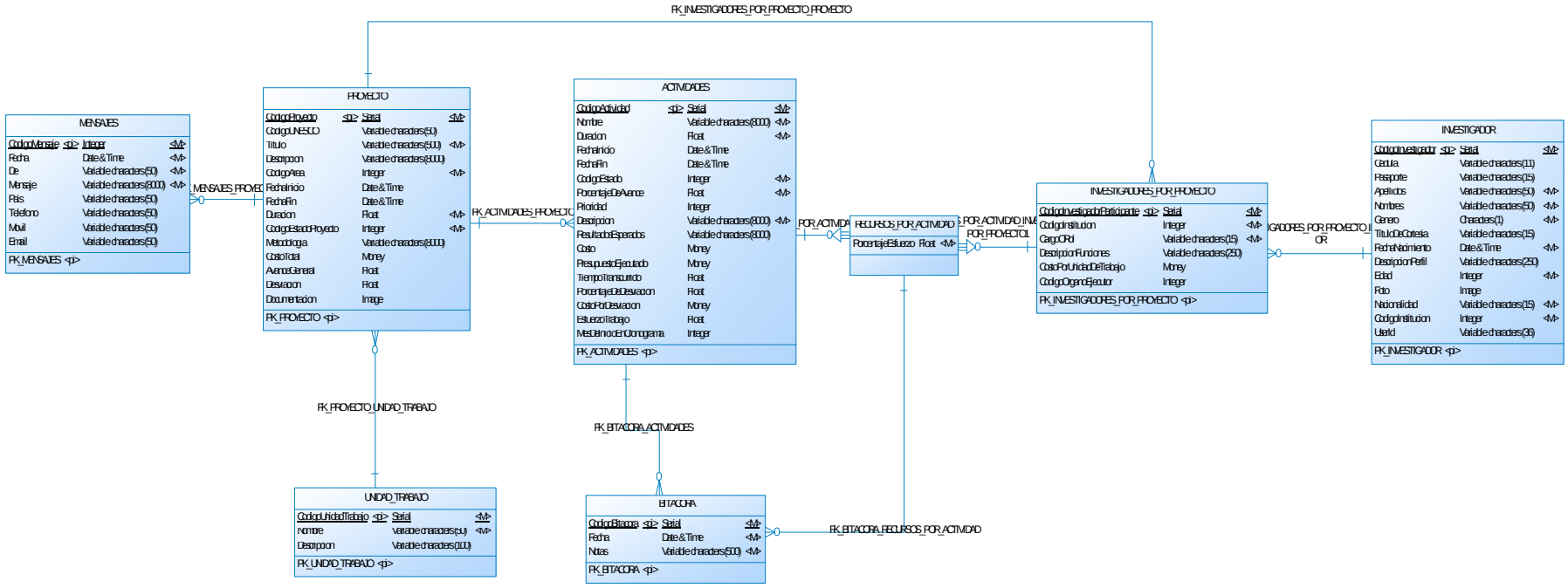


Figura 5.7: Diagrama Conceptual de Actividades y Sub-Actividades del proyecto

5.8.5.3 Diagrama Conceptual De Investigadores Asociados a un Proyecto

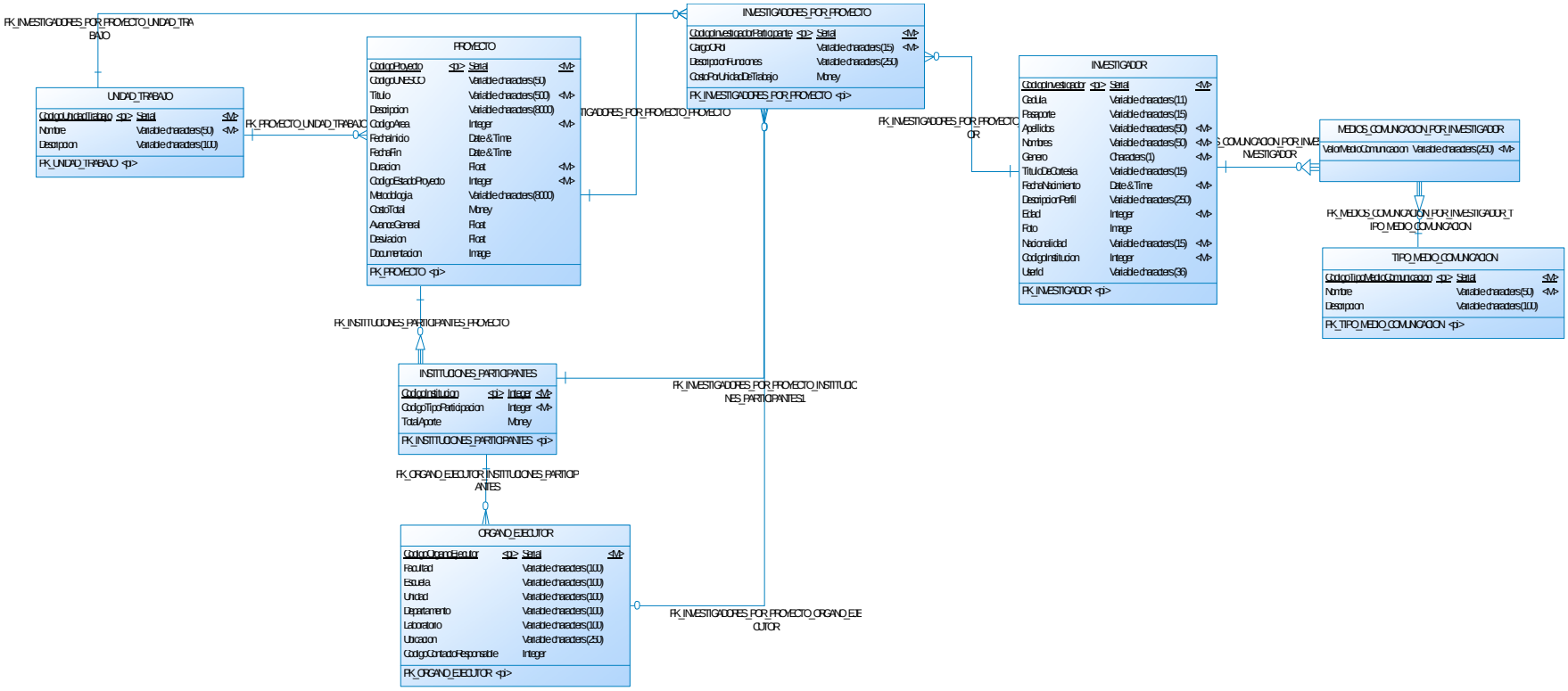


Figura 5.8: Diagrama Conceptual de Investigadores Asociados a un Proyecto

5.8.5.4 Diagrama Conceptual Presupuesto y Financiamiento del Proyecto

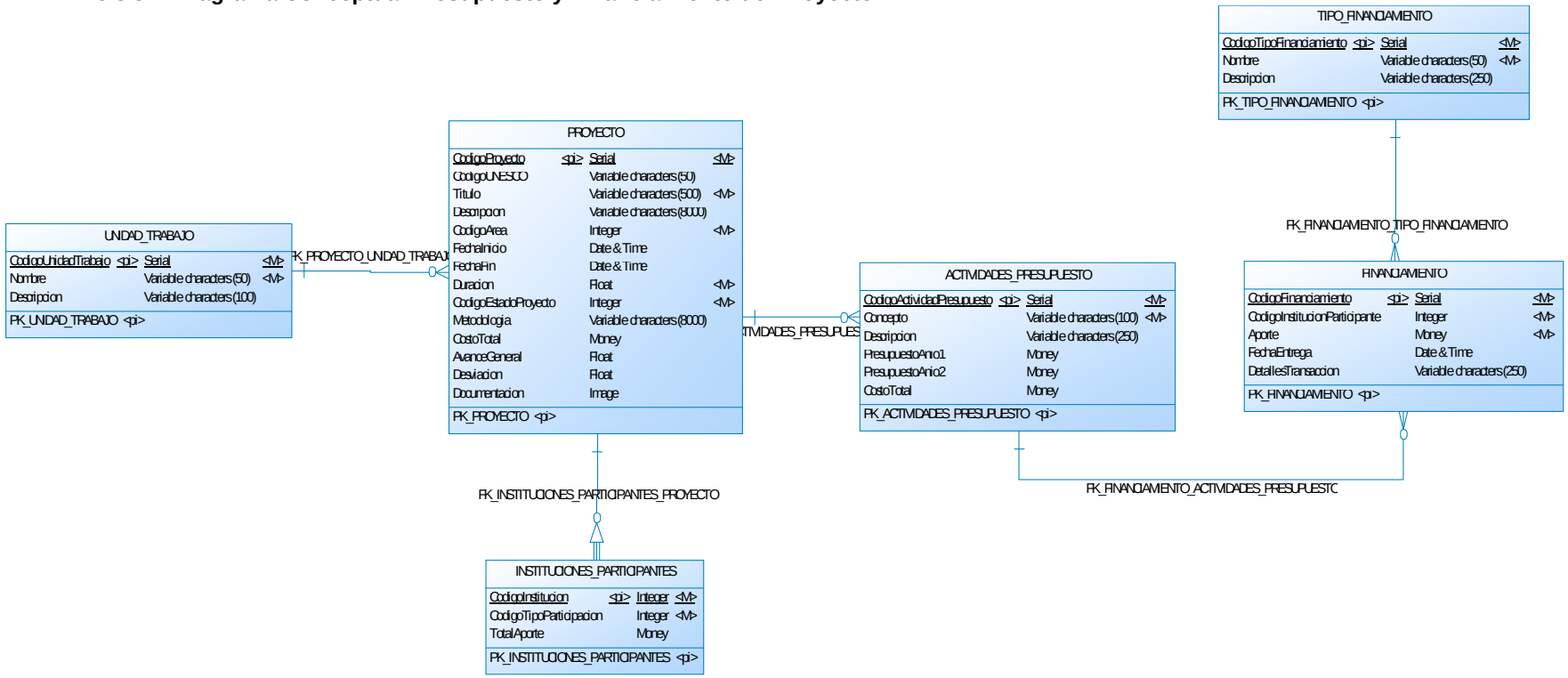


Figura 5.9: Diagrama Conceptual: Presupuesto y Financiamiento del Proyecto

5.8.5.5 Diagrama Conceptual Esquema de Seguridad y Acceso a la Aplicación

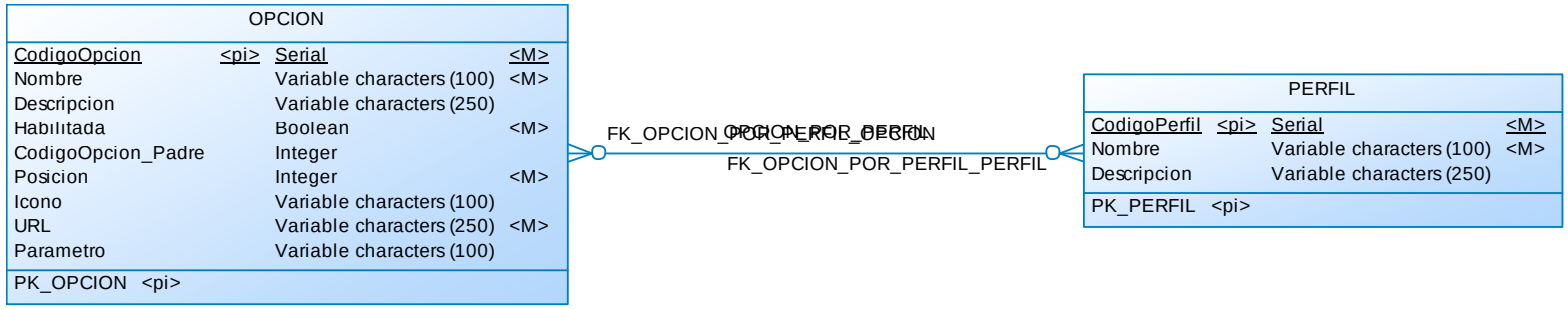


Figura 5.10: Diagrama Conceptual: Esquema de Seguridad y Acceso a la Aplicación

5.8.5.6 Diagrama de Seguridad y Autenticación MEMBERSHIP ASP.NET

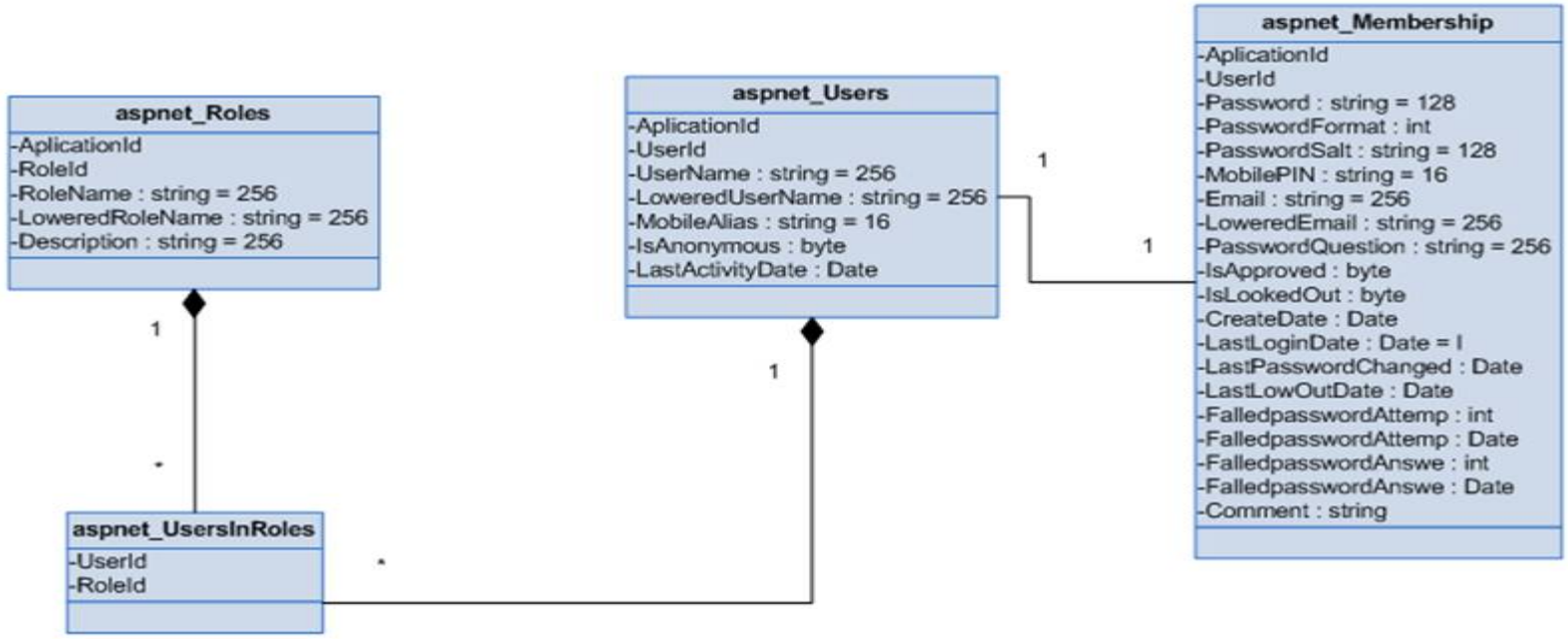


Figura 5.11: Diagrama de Seguridad y Autenticación MEMBERSHIP ASP.NET

5.8.6 Diagrama De Secuencia

5.8.6.1 Diagrama de Secuencia Administrador

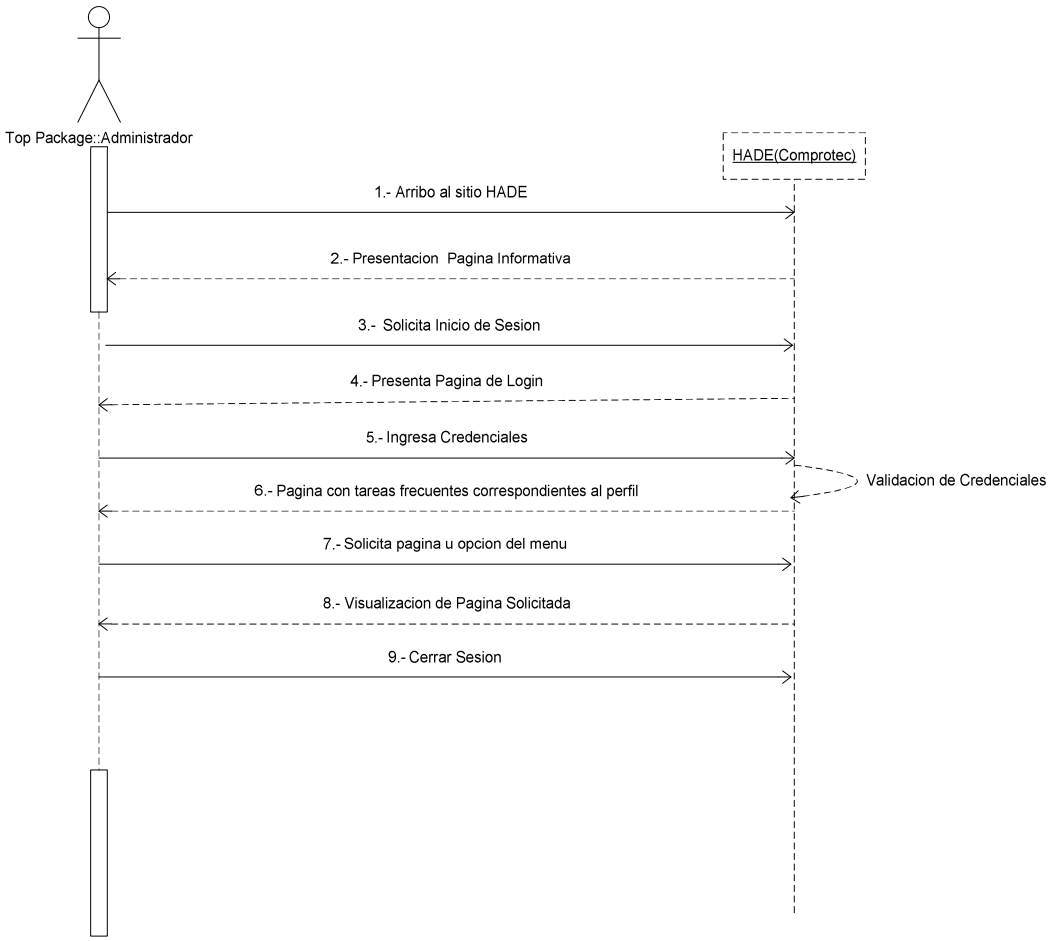


Figura 5.12: Diagrama de Secuencia: Administrador

5.8.6.2 Diagrama de Secuencia Director

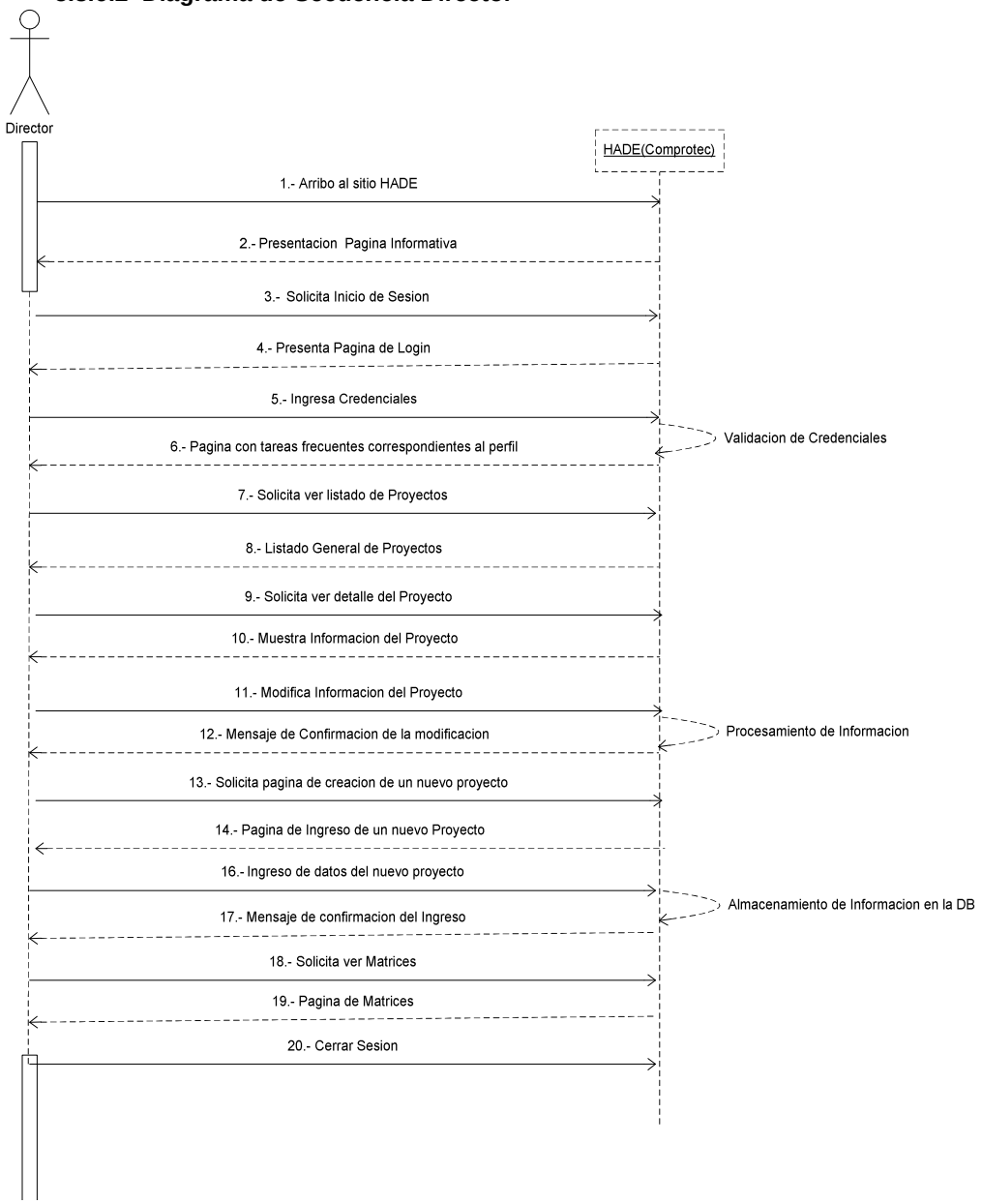


Figura 5.13: Diagrama de Secuencia: Director

5.8.6.3 Diagrama de Secuencia Investigador

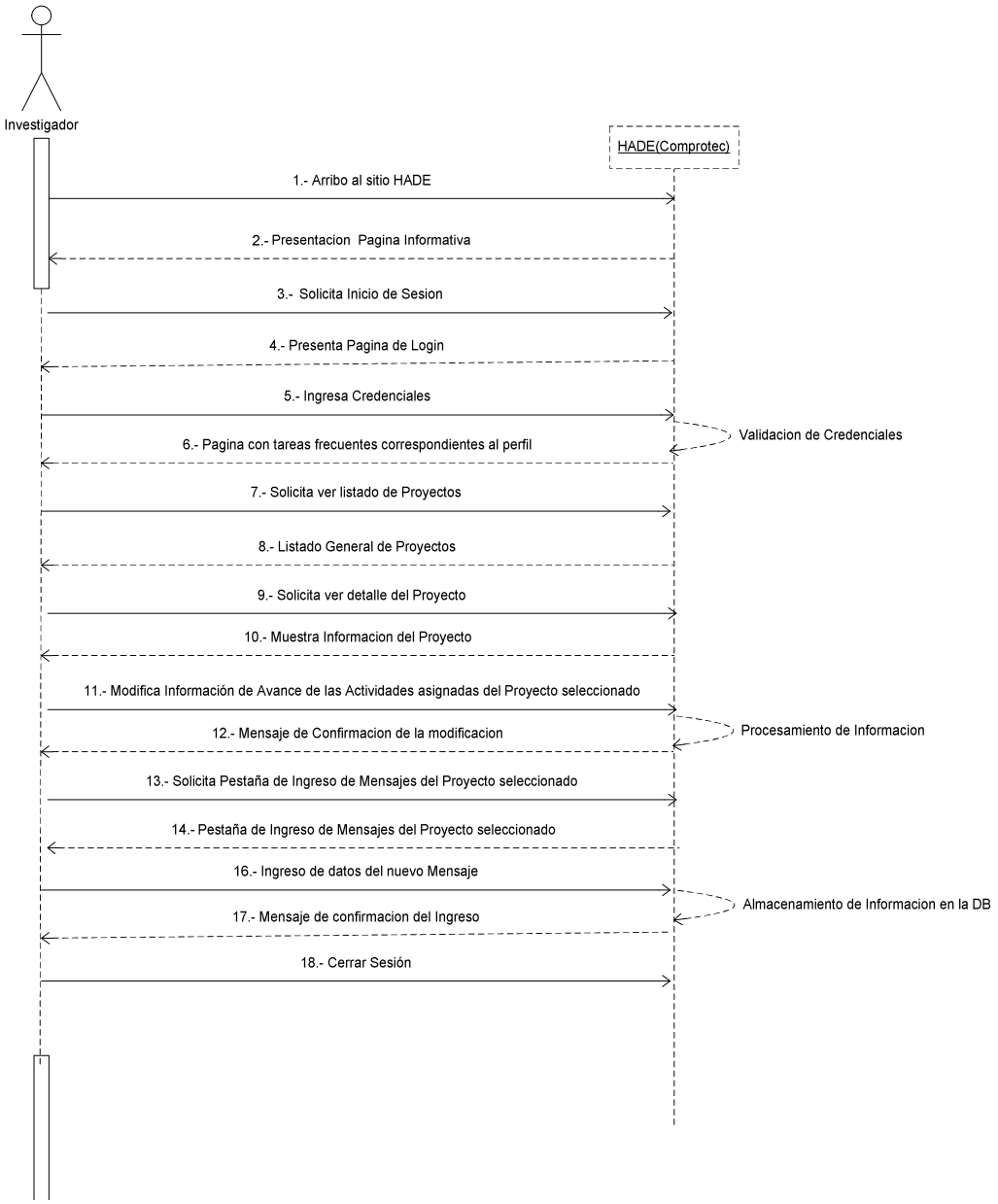


Figura 5.14: Diagrama de Secuencia: Investigador

5.9 FASE DE DISEÑO

5.9.1 Introducción

La Fase de Diseño tiene como objetivo definir la arquitectura de la solución, los módulos y unidades lógicas que la conforman, los componentes de negocio que encapsulan la funcionalidad requerida y el esquema de despliegue y distribución, basándose en las abstracciones obtenidas en las Fases de Planificación y de Análisis.

5.9.2 Diagramas de Diseño Arquitectónico

5.9.2.1 ARQUITECTURA DEL SISTEMA HADE

Figura 5.15: Arquitectura del Sistema HADE

En la figura 5.15 se muestra la arquitectura de la Solución HADE Comprotec. Consiste básicamente en tres capas lógicas de aplicación, en donde se agrupa la funcionalidad asociada de cada aspecto de negocio en componentes empresariales.
Las tareas, consultas y procesos de negocio se encuentran encapsuladas y organizadas en gestiones de negocio que juntos resuelven las necesidades de automatización de la solución planteada.

De tal forma se tiene las siguientes Gestiones de Negocio:

- **Proyecto:** Encapsula las tareas de Creación, Consulta y Modificación de Proyectos que se manejan en la unidad Comprotec. Los componentes dentro de ésta gestión se encargan de agrupar y resumir la información de proyectos para presentarla al usuario. El manejo de los Datos Generales de los Proyecto, Información de Avance y Seguimiento, las Actividades dentro del proyecto y su avance, Cronogramas de Trabajo, Instituciones Participantes, Investigadores Asociados, Archivos Adjuntos y Mensajes; hacen de esta gestión el corazón del sistema.

- **Seguridad:** Contiene la lógica para el manejo del esquema de Autenticación y Autorización de la aplicación. Los componentes dentro de esta gestión se encargan de validar la identidad del usuario que ingresa al sistema y de otorgar los privilegios de navegación y visibilidad sobre los recursos. Gestiona la información de usuarios, estados de sesión, mantenimiento de contraseñas, bloqueo de usuarios, establecimiento de permisos sobre los recursos, entre otras tareas relacionadas.

- **Sistema:** Los componentes dentro de esta gestión de negocio se encargan de brindar mantenimiento a los datos del Sistema; entendiéndose por estos a los datos de parametrización inicial que requiere el sistema para su

funcionamiento. La información de Catálogos o Tablas del sistema, Parámetros de Configuración de la aplicación y otros datos necesarios para el funcionamiento del Sistema son manejados dentro de esta lógica de negocio.

- **Análisis:** La Gestión de Análisis contiene las implementaciones de los distintos paradigmas estudiados en el presente documento. Los componentes de Análisis contienen las definiciones de los ejercicios de prueba y sus algoritmos en las implementaciones respectivas que serán analizadas para el estudio detallado del comportamiento de los paradigmas planteados. La lógica dentro de esta Gestión es invocada desde el aplicativo de Pruebas HADE Comprotec.

Adicionalmente, la arquitectura cuenta con una Capa Vertical denominada **Entidades Extendidas** que contiene la definiciones de contenedores de datos que agrupan la información que es cargada por los componentes de la Capa de Acceso a Datos y son transmitidas hasta las páginas en la Capa de Presentación. Una entidad extendida es básicamente una clase que contiene propiedades sobre las cuales se define un contexto de información en función de la necesidad de negocio. En el entorno de .NET, a estas entidades se las conoce como POCOs (*Plain Old CRL Objects* por sus siglas en inglés, o Objetos Viejos Planos del Lenguaje de Ejecución Común del entorno de desarrollo .NET), que son objetos diseñados para la comunicación entre capas de aplicaciones construidas con la tecnología .NET.

A continuación se describe brevemente cada capa de la aplicación HADE Comprotec:

- **Base de Datos:** La base de datos es el repositorio de información utilizada en toda la aplicación. Para el sistema HADE, el repositorio de datos está representado en una Base de Datos relacional del Motor DBMS SQL Server 2000 de Microsoft. Permite alojar en las tablas toda la información necesaria y

relacionarla; adicionalmente permite la creación de objetos programáticos como funciones y procedimientos almacenados que encapsula cierta lógica del acceso a datos utilizada en la aplicación.

- **Capa de Acceso a Datos:** Los componentes al interior de esta capa contienen la lógica de consulta, inserción, modificación y eliminación de datos sobre las tablas de la base de datos de la aplicación.

- **Capa de Lógica de Negocio:** Encapsula la funcionalidad propia del negocio que es utilizada para la realización de cálculos, validaciones y controles sobre los datos obtenidos por la Capa de Acceso a Datos. Permite exponer los datos obtenidos por la Capa de Acceso a Datos y pasarlos a la Capa de Presentación para que sean mostrados.

- **Capa de Presentación:** Contiene los Elementos de Interfaz Gráfica que le permite al usuario la interacción con los datos del sistema. La capa de Presentación es representada por un proyecto de tipo Aplicación Web de .NET construido en base a la tecnología ASP.NET. Dentro de este proyecto se ubica una estructura de directorios y de elementos de presentación: Páginas HTML, Páginas ASPX, Páginas Maestras, Hojas de Estilos, Imágenes, Archivos Javascript, Archivos de Configuración, entre otros. Todos estos elementos son encargados de invocar a los componentes de la Capa de Lógica de Negocio, extraer la información y mostrarla de forma adecuada para agilizar la interacción con el usuario.

5.9.3 DIAGRAMAS FÍSICOS DE BASES DE DATOS

5.9.3.1 Diagrama De Actividades Y Sub-Actividades Del Proyecto

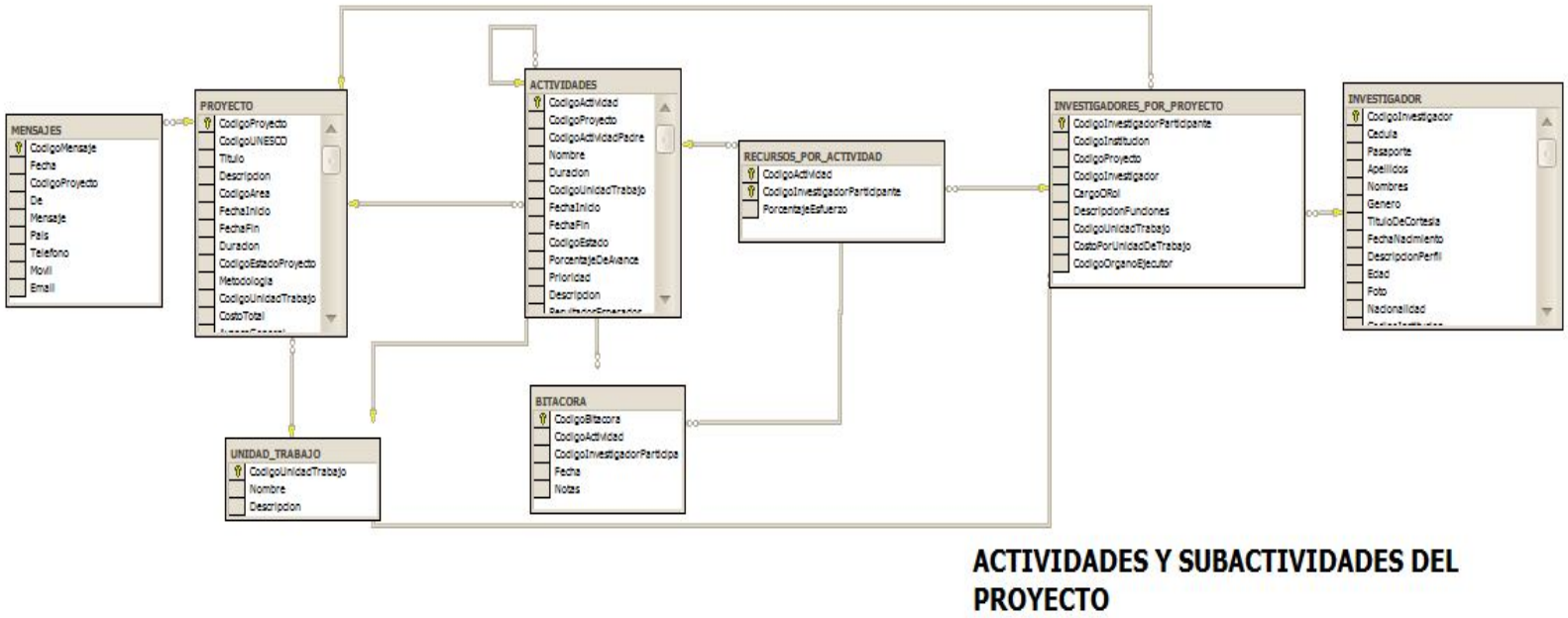


Figura 5.16: Diagrama Físico de Actividades y Sub-Actividades del Proyecto

5.9.3.2 Diagrama Físico De Datos Generales Del Proyecto E Instituciones Participantes

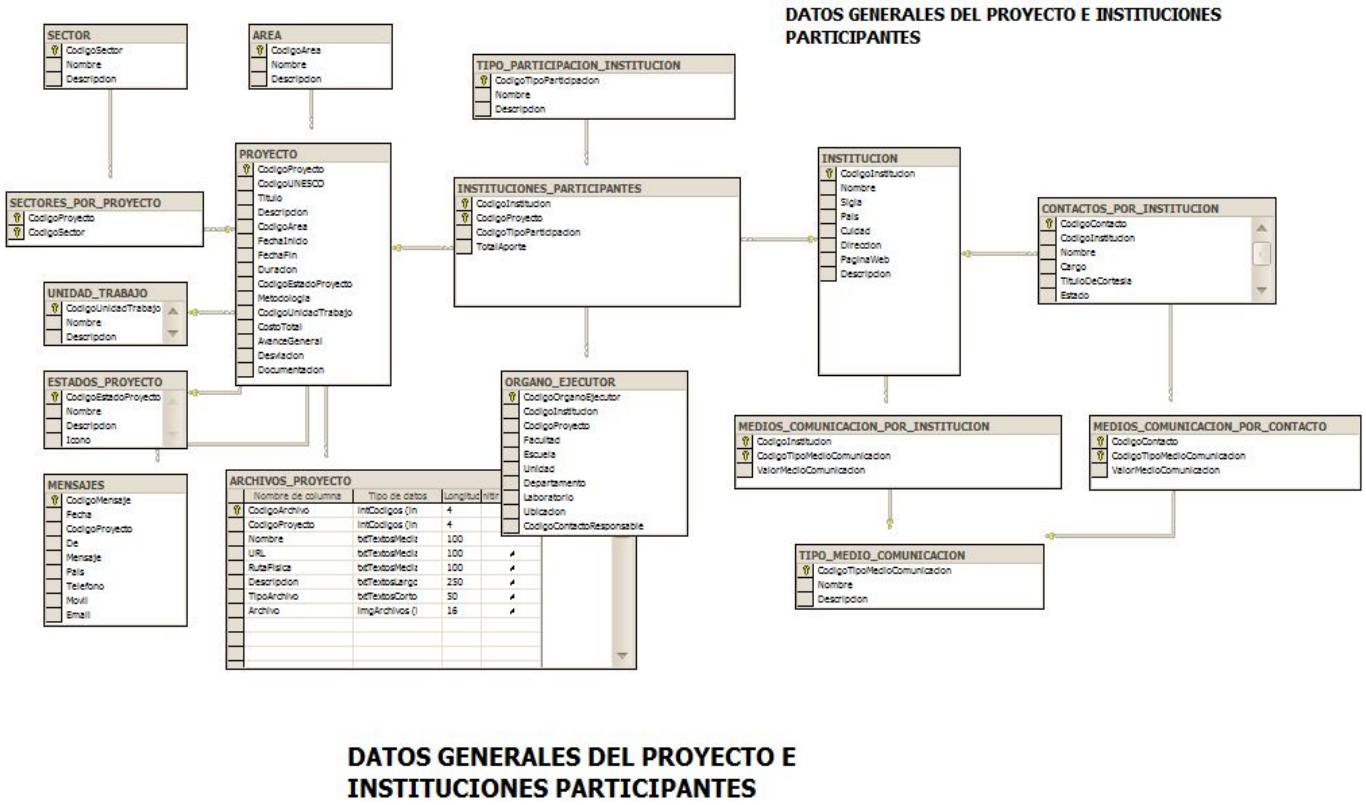


Figura 5.17: Diagrama Físico de Datos Generales del Proyecto e Instituciones Participantes

5.9.3.3 Diagrama Físico de Investigadores Asociados a un Proyecto

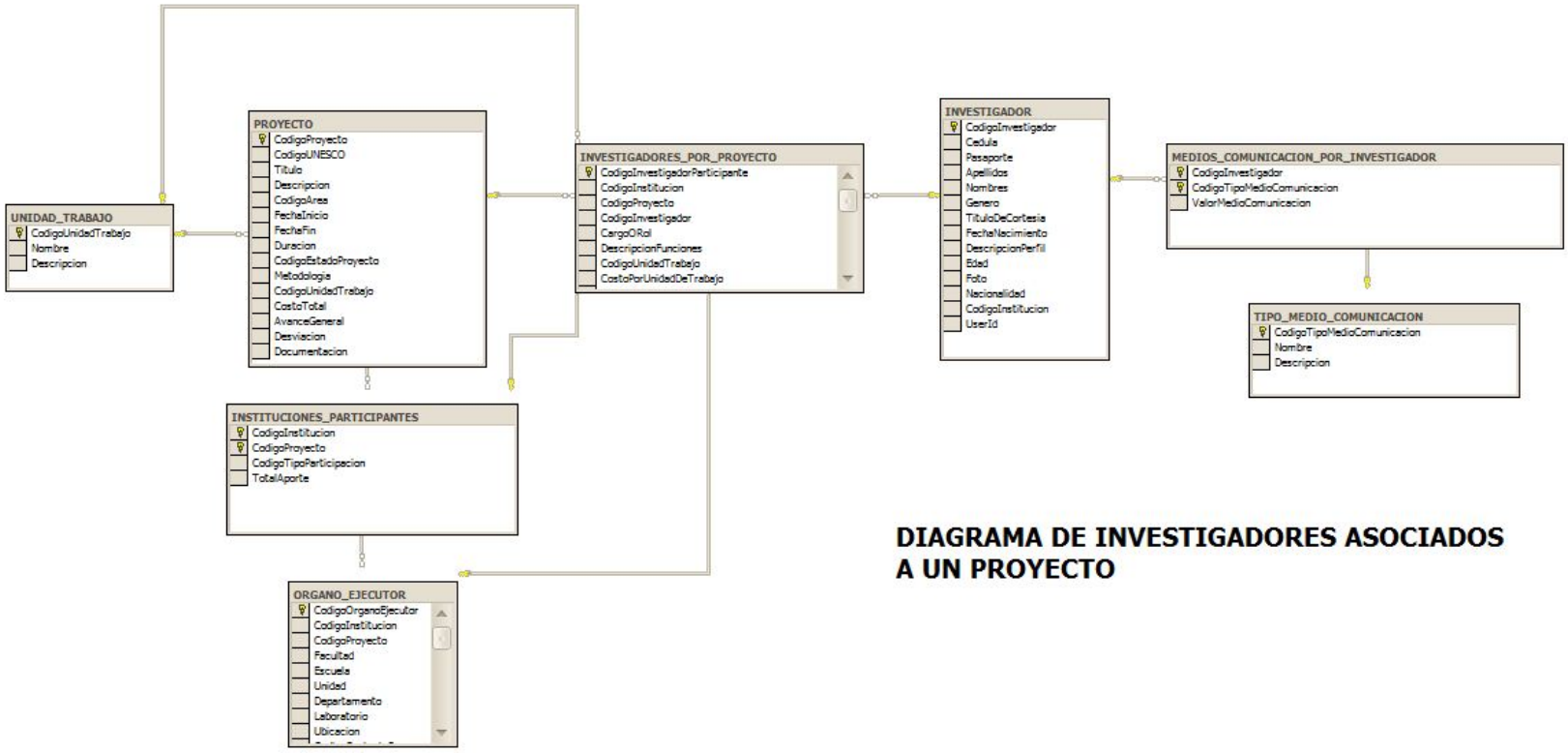


Figura 5.18: Diagrama Físico de Investigadores Asociados a un Proyecto

5.9.3.4 Diagrama Físico Presupuesto y Financiamiento del Proyecto

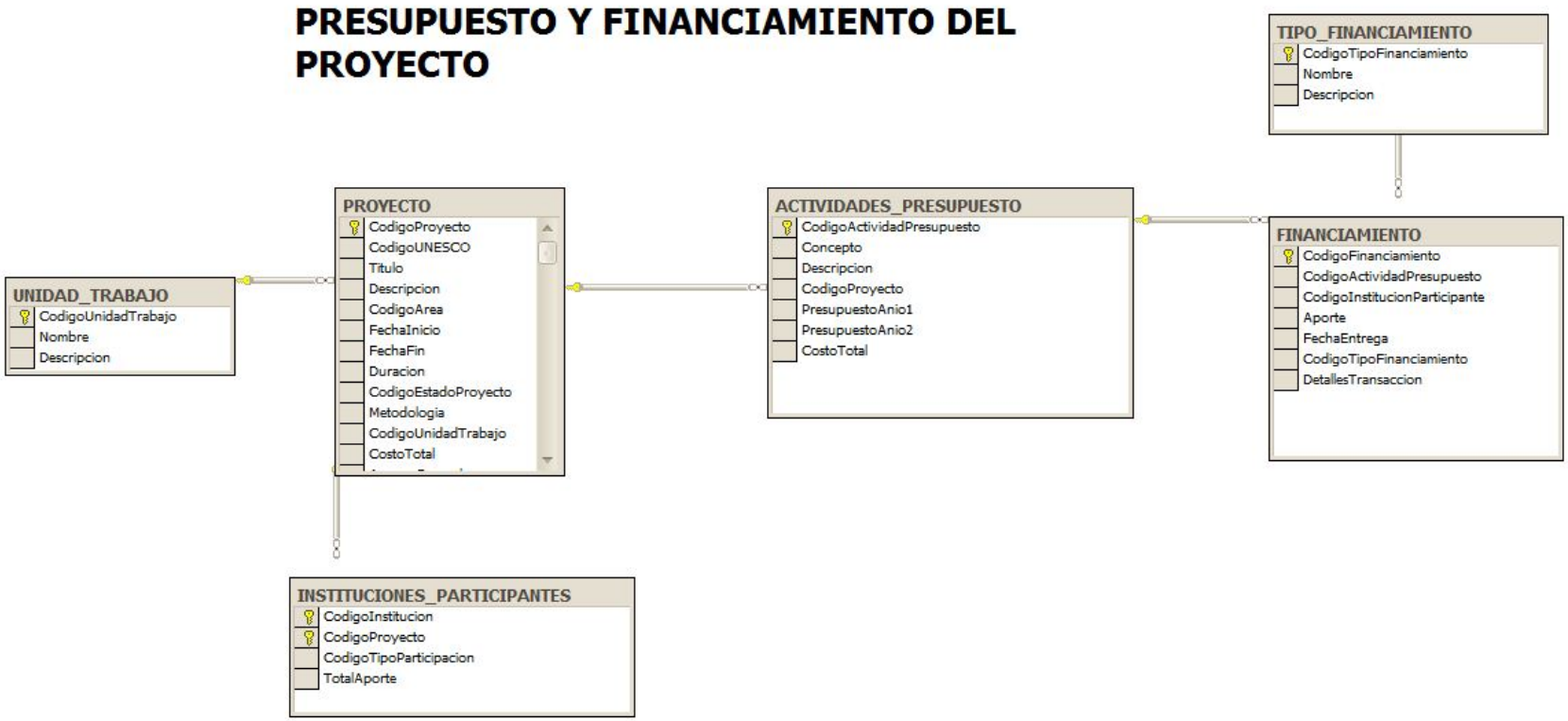


Figura 5.19: Diagrama Físico del Presupuesto y Financiamiento del Proyecto

5.9.3.5 Diagrama Físico de Seguridad

Figura 5.20: Esquema de Seguridad y Acceso a la Aplicación

5.9.3.6 Diagrama Físico de Seguridad y Autenticación MEMBERSHIP ASP.NET

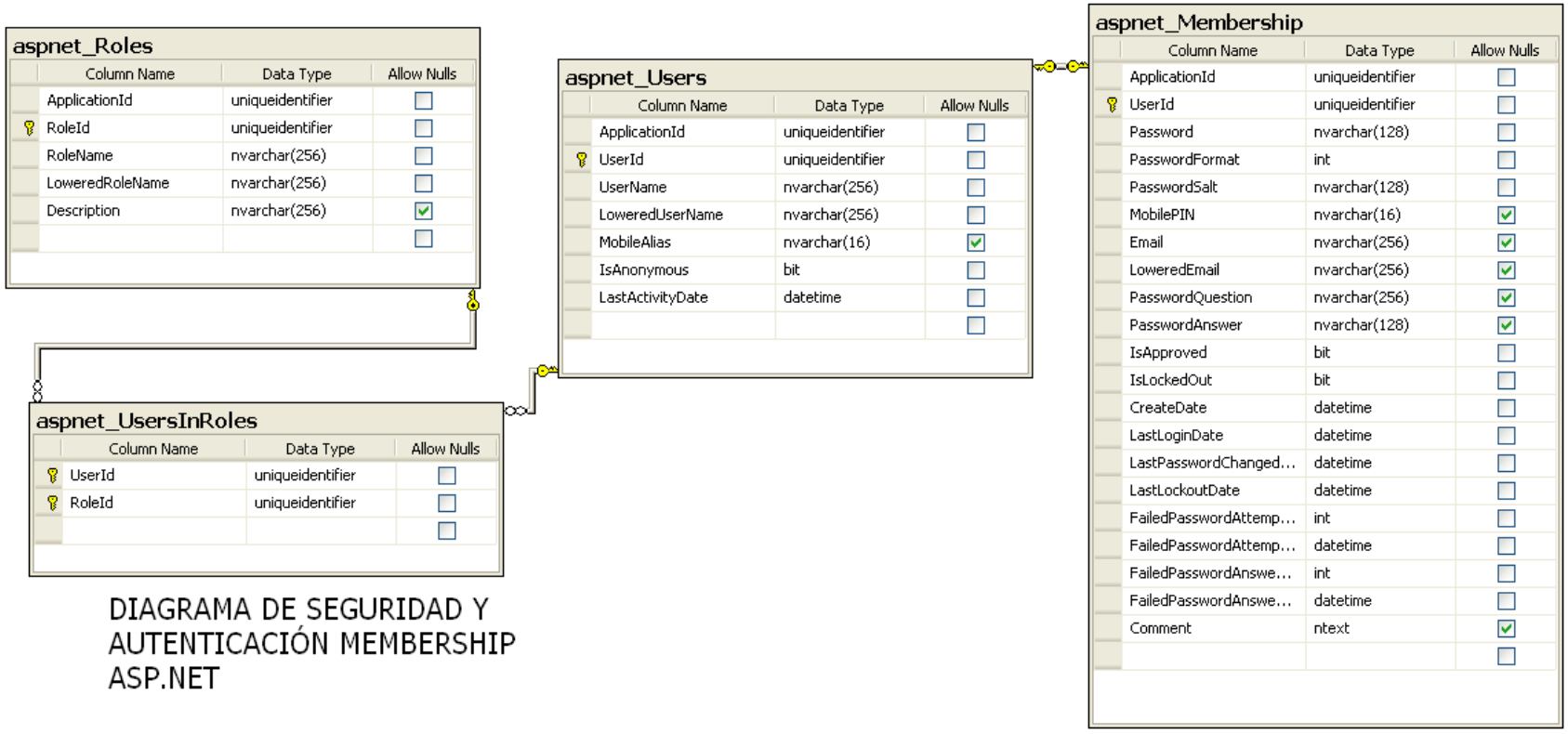


Figura 5.21: Diagrama de Seguridad y Autenticación MEMBERSHIP ASP.NET

5.9.4 DIAGRAMA DE COMPONENTES

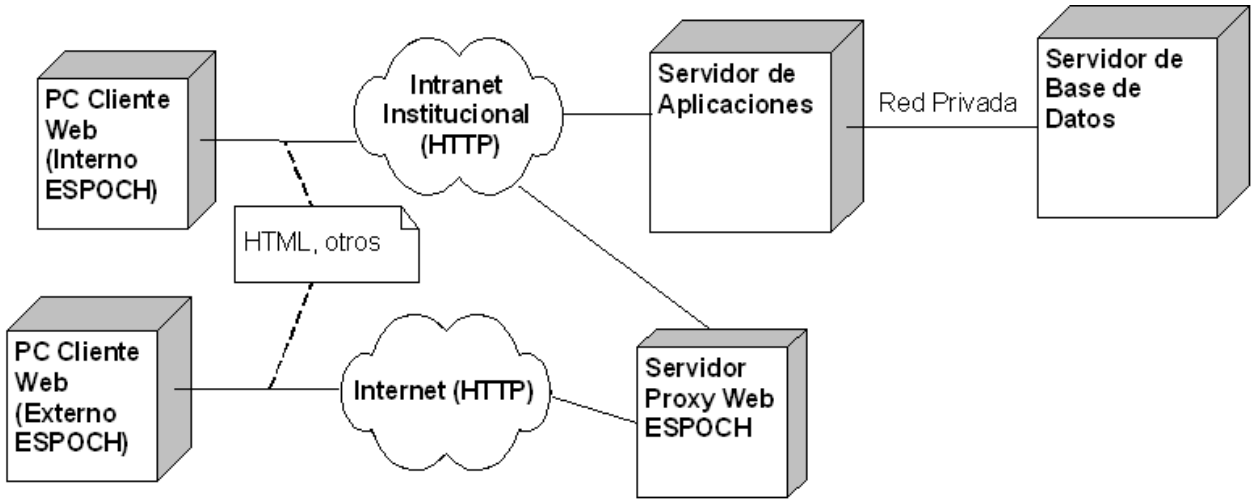


Figura 5.22: Diagrama de Componentes del Sistema HADE

5.9.5 Diagrama de Despliegue

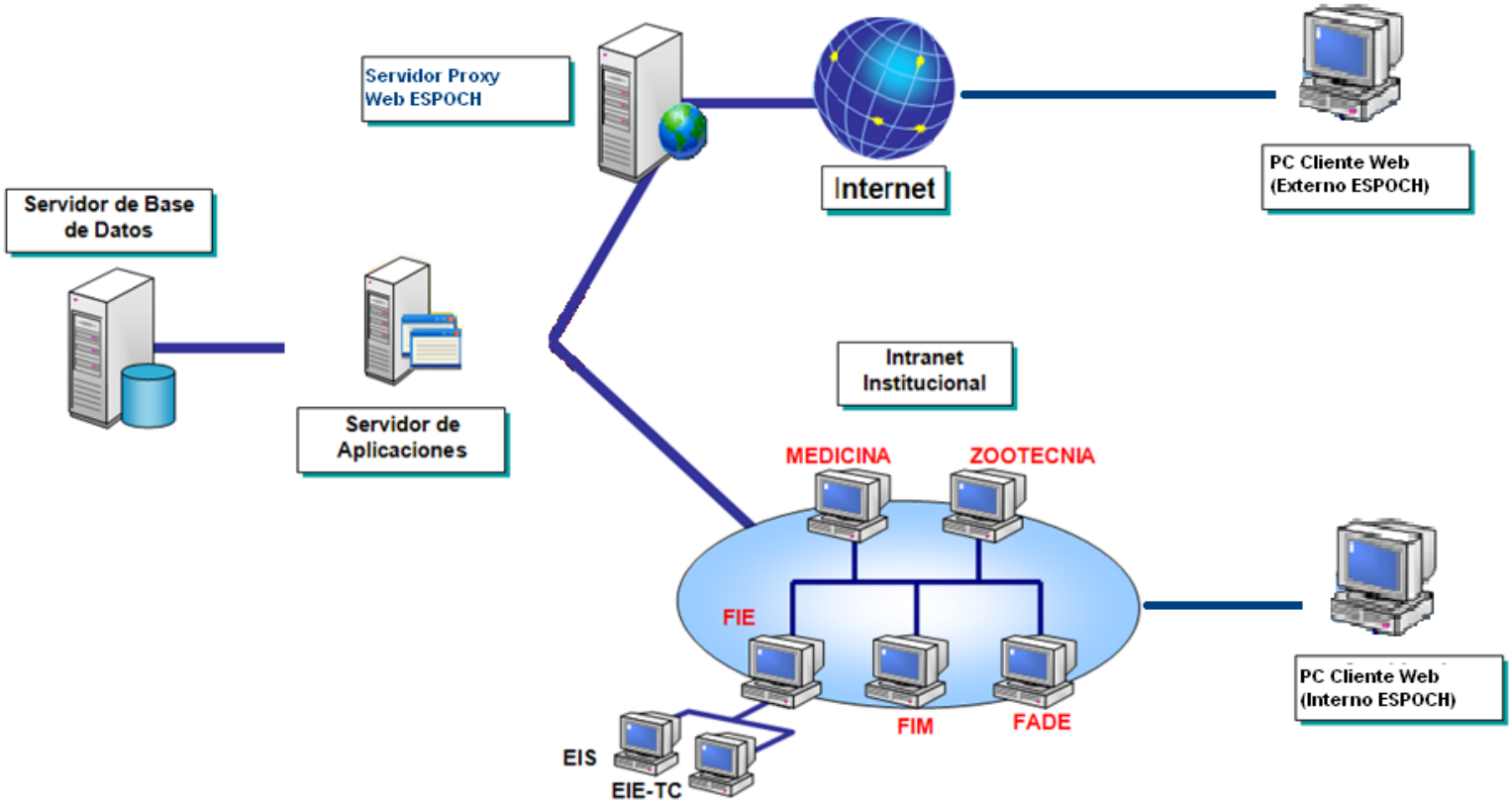


Figura 5.23: Diagrama de Despliegue del Sistema HADE

5.10 FASE DE IMPLEMENTACIÓN

5.10.1 Introducción

La etapa de implementación inicia con los resultados del diseño e “implementa” el sistema en términos de componentes, es decir, archivos de código fuente, scripts, archivos de código binario, ejecutables y similares. La mayor parte de la arquitectura del sistema es capturada durante el diseño, entonces el propósito principal de la implementación es desarrollar la arquitectura y el sistema como un todo.

De forma específica la implementación planifica varias “construcciones” para el sistema y las integraciones necesarias. Distribuye el sistema asignando los componentes ejecutables a los diferentes nodos de cómputo. Implementa las clases y subsistemas encontrados durante el diseño, prueba los componentes individualmente y a continuación los integra compilándolos y enlazándolos en uno o más ejecutables.

El principal resultado de la implementación es el modelo de implementación. Este modelo denota la implementación actual del sistema en términos de componentes y subsistemas de implementación. El modelo de implementación describe cómo los elementos del modelo de diseño (como las clases) se implementan en términos de componentes (como archivos de código fuente, ejecutables, etc.). El modelo de implementación también describe cómo se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación (para este proyecto Visual Studio .Net 2008) y el lenguaje o lenguajes de programación utilizados (en este caso Visual C# .Net), y cómo dependen los componentes unos de otros. Este capítulo ofrece la descripción del modelo de implementación y cómo ha sido estructurado en Visual Studio .Net.

5.10.2 Descripción de los Principales Elementos de Implementación de .Net

Cada elemento en el modelo de diseño tiene su correspondencia en el modelo de implementación que se indica en el diagrama mediante una dependencia de residencia (*reside*). Por ejemplo, una capa de abstracción lógica en el diseño se corresponde con un *namespace* en el modelo de implementación. Una página de servidor corresponde con un archivo .aspx y una la clase que define a la página de servidor se implementa como un archivo .aspx.cs (*code behind*).

El archivo .aspx contiene el código HTML que define la interfaz Web de la página ASP.Net, además incluye ciertas etiquetas especiales que representan a los controles de servidor. El archivo .aspx.cs (code behind) contiene el código fuente de la clase que se ejecuta cuando la página del servidor es instanciada en memoria.

Los directorios y sus contenidos: archivos .aspx, archivos de code behind y muchos otros tipos de archivo se agrupan formando un Proyecto. Un proyecto se compilará generalmente en un componente ejecutable (*assembly* en el vocabulario de .Net). Un proyecto también incluye dependencias hacia otros assemblies desarrollados en .Net (ej: System.Xml). Finalmente, varios proyectos se agrupan para formar una Solución (*solution*). La solución es el contenedor más grande para el modelo de implementación.

5.10.3 ORGANIZACIÓN DE LOS ELEMENTOS DE IMPLEMENTACIÓN DEL SISTEMA HADE

La solución lleva como nombre **Comprotec** y su estructura en el Visual Studio .Net se esquematiza en la figura a continuación.

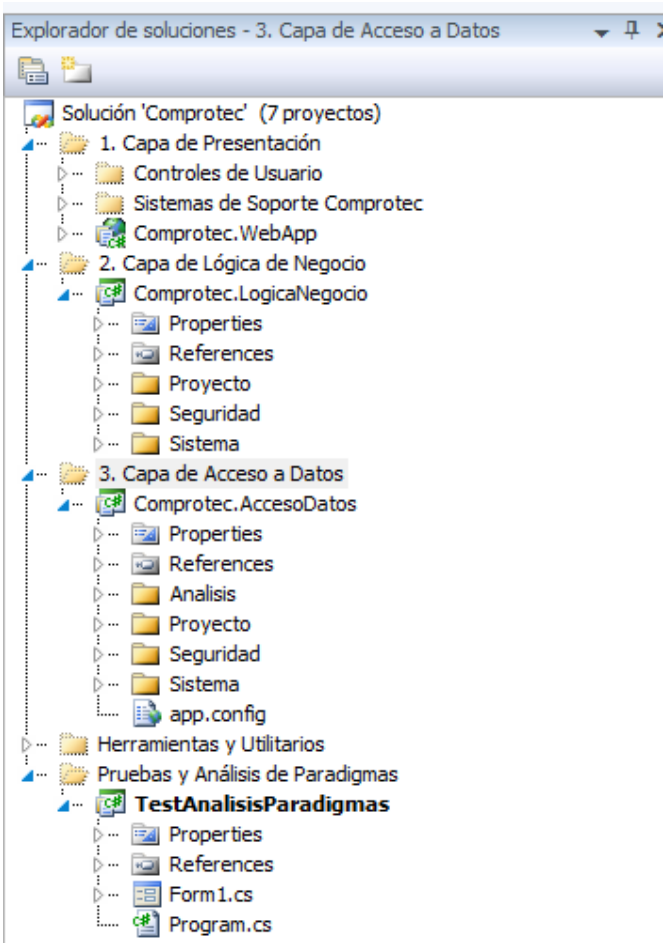


Figura 5.24: Organización de la Solución del Sistema HADE

5.10.3.1 Base de Datos de Comprotec

La base de datos del sistema HADE está representada en una Base de Datos relacional del Motor DBMS SQL Server 2000 de Microsoft denominada **ComprotecBdd**, que contiene un conjuntos de objetos que permiten alojar y manejar toda la información necesaria y relacionarla. Adicionalmente existen objetos programáticos como: Tipos de Datos de Usuario, reglas, funciones de Usuario; procedimientos almacenados que son usados sobre las tablas que

contienen la información. A continuación se brinda una vista de la estructura de la base de datos de Comprotec dentro del Administrador Corporativo de SQL Sever 2000:

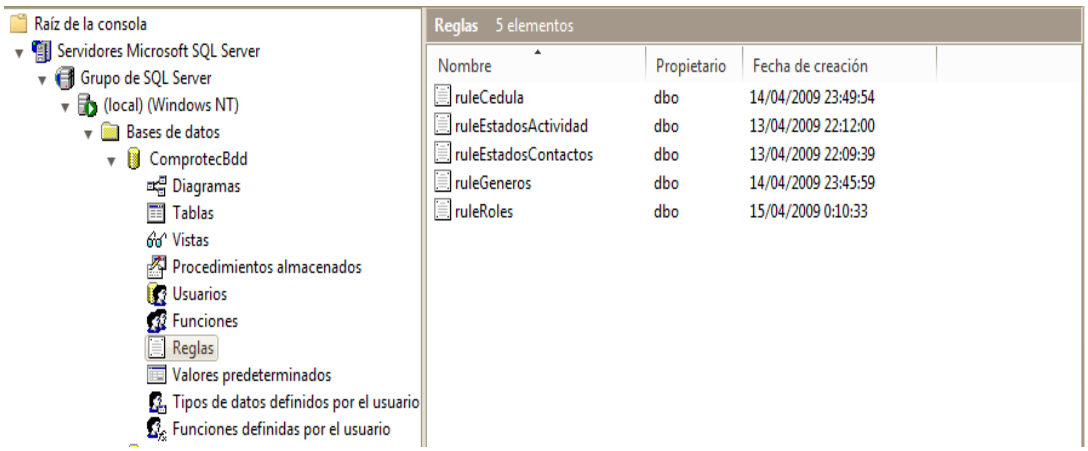


Figura 5.25: Organización de los Objetos de la Base de Datos ComprotecBdd

5.10.3.2 Capa Acceso Datos

Es la encargada de acceder a los datos. El proyecto **Comprotec.AccesoDatos** representa a la capa de acceso a datos especificada en la arquitectura lógica del diseño del sistema HADE *(véase Figura 5.15: Arquitectura del Sistema HADE)***.** El proyecto contiene varios directorios en donde cada directorio representa una Gestión de Negocio. Cada directorio posee los siguientes archivos:

- ***<Nombre de la Gestión de Negocio>*.dbml**: que representa al objeto `DataContext` en donde se mapea la estructura relacional de las tablas y vistas de la base de datos y se define el entorno de datos sobre el cual se realizarán tareas de almacenamiento y recuperación de información desde la capa de negocio. Mediante este archivo pueden hacer consultas LINQ a la base de datos, que

básicamente se comporta como un contenedor de clases, donde cada clase es una tabla de la base de datos, pero no se agregan todas las tablas de la base de datos sino aquellas tablas de las cuales se desea extraer información para la implementación de las tareas que le corresponden a la gestión correspondiente (por ejemplo; el DataContext de seguridad solo contiene las tablas de seguridad y no las de Proyecto).

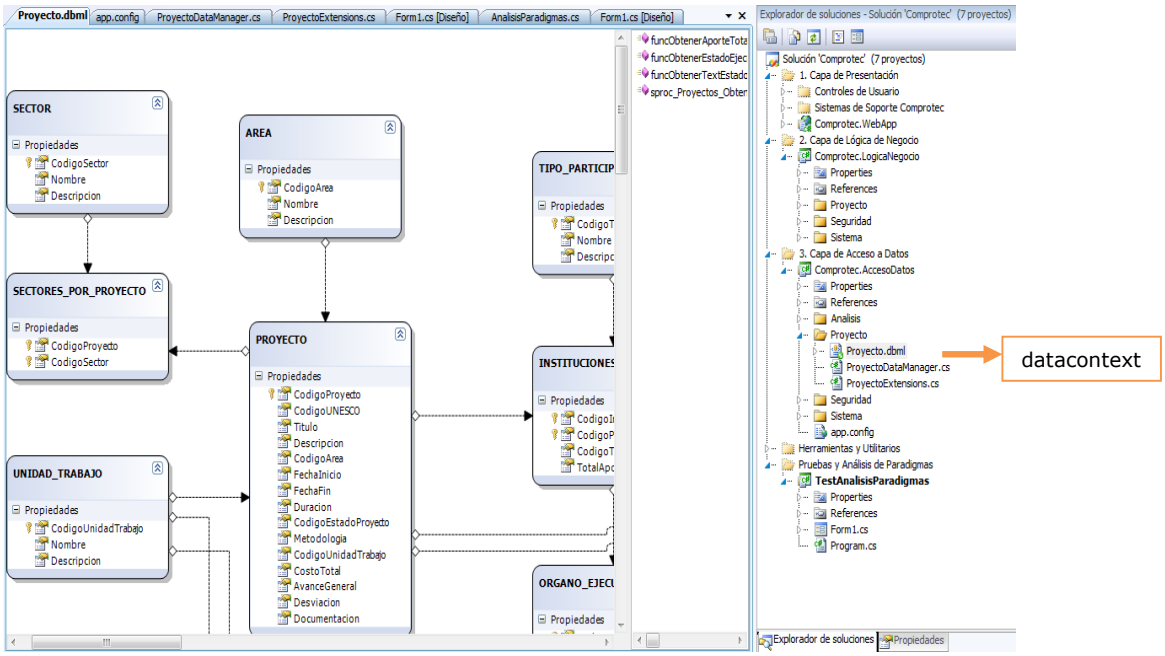


Figura 5.26: Representación del DataContext en la Capa Acceso a Datos del Sistema HADE

- ***<Nombre de la Gestión de Negocio>DataManager*.cs**: que representa a la clase que contiene las consultas que se hacen sobre el objeto DataContext. Los métodos dentro de esta clase representan las operaciones de datos: consulta, inserción, actualización y eliminación de datos, sobre las colecciones de objetos del DataContext y posteriormente a la base de datos. Las consultas LINQ se definen dentro de esta clase.

- ***<Nombre de la Gestión de Negocio>*Extensions.cs:** Este archivo contiene las Entidades Extendidas que sirven para el agrupamiento y paso de la información a través de las capas de aplicación. Las clases dentro de este archivo constituyen los objetos POCOs utilizados en el entorno .NET para la comunicación entre las capas de una aplicación, encapsulando información relacionada y simplificándola *(véase Figura 5.15: Arquitectura del Sistema HADE)***.** El archivo de Entidades Extendidas no se encuentra necesariamente en todas las gestiones de negocio de la capa de datos, únicamente en aquellas que requieren obtener cierta información agrupada utilizada en la capa de presentación.

5.10.3.3 Capa Lógica de Negocio

El proyecto **Comprotec.LogicaNegocio** representa a la capa de negocio especificada en la arquitectura lógica del diseño del sistema HADE *(véase Figura 5.15: Arquitectura del Sistema HADE)***.** . Está compuesto por un conjunto de clases dentro de estas clases se encapsula la inteligencia de los mecanismos necesarios para la interacción con la siguiente capa. Básicamente, presenta la misma estructura de directorios que la Capa de Acceso a datos, donde igualmente cada directorio representa una Gestión de Negocio. Dentro de cada directorio se encuentra una clase denominada ***<Nombre de la Gestión de Negocio>*LogicManager.cs** en donde se implementa las invocaciones a los métodos respectivos de la capa de negocio.

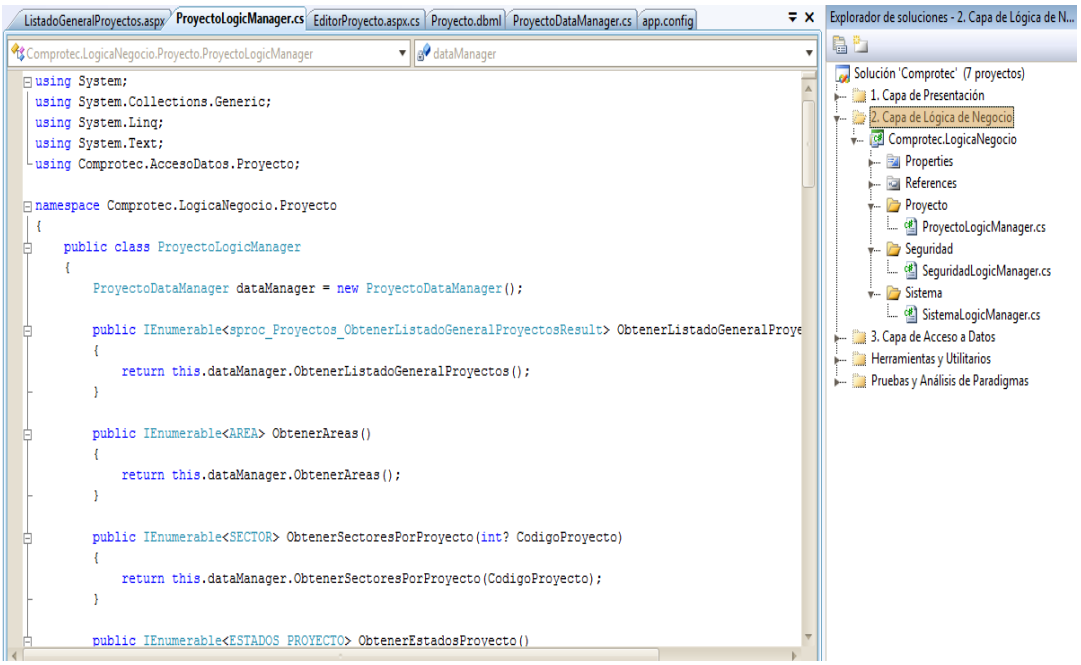


Figura 5.27: Representación de la Capa lógica de Negocio del Sistema HADE

5.10.3.4 Capa de Presentación

Es la que ve el usuario se la denomina también "capa de usuario", presenta el sistema al usuario, le comunica la información y captura la información del usuario dando un mínimo de proceso. Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz grafica y debe tener la característica de ser amigable (entendible y fácil de usar) para el usuario. La capa de Presentación de Comprotec está conformada de 3 proyectos: dos aplicaciones Web y un proyecto de tipo Librería de clases, donde el Proyecto principal es **Comprotec.WebApp**, que mostrará la información al usuario.

Comprotec.WebApp: Es una aplicación Web ASP.NET 3.5 posee un conjunto de elementos de presentación entre los cuales enumeramos: páginas maestras, páginas ASPX, hojas de estilo, imágenes, archivos javascript y archivos de configuración.

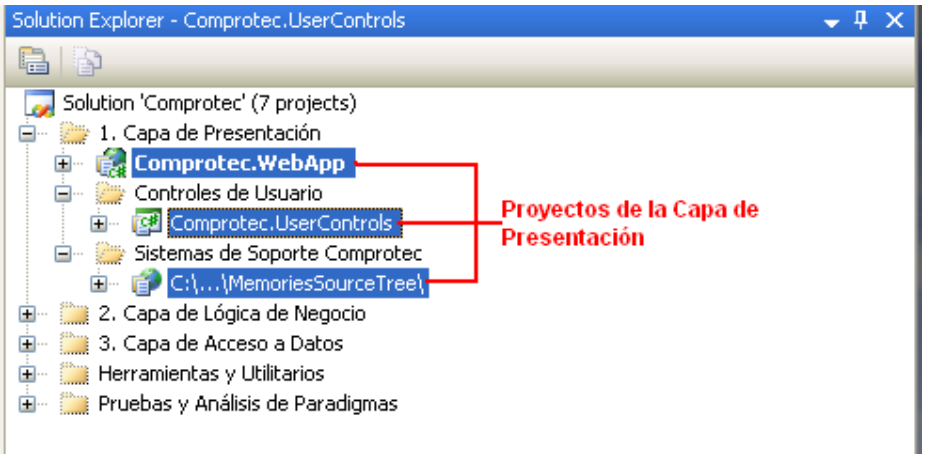


Figura 5.28: Representación de la Capa de Presentación del Sistema HADE

El proyecto **Comprotec.WebApp** se encuentra estructurado de forma similar a la estructura de la Capa de Negocio. Contiene un conjunto de directorios de Negocio que principalmente alojan las páginas que muestran y permiten el ingreso de información relacionada al negocio de Comprotec: Información de Proyectos, Configuración de Seguridad, Datos del Sistema, etc. Las páginas y clases dentro de los directorios de Negocio invocan a los componentes de la Capa de Lógica de Negocio.

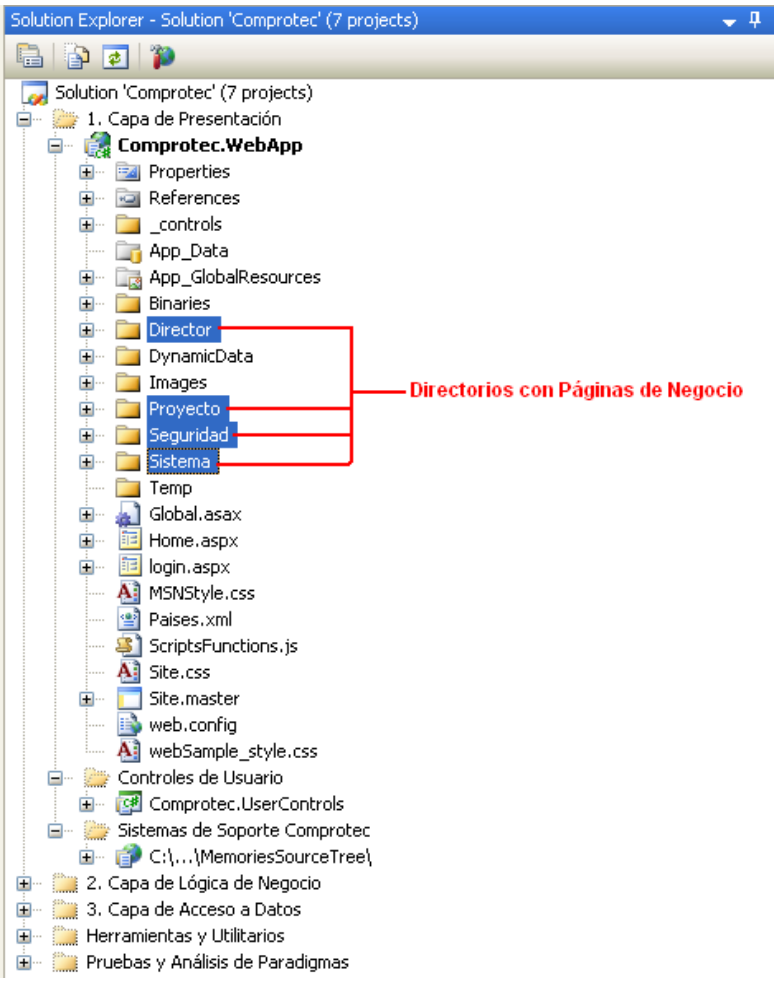


Figura 5.29: Representación de Directorios con Páginas de Negocio

Páginas Contenidas dentro de los Directorios de Negocio

Los directorios de negocio son: **Director, Proyecto, Seguridad y Sistema**. Dentro de cada directorio existen páginas ASPX y en algunas clases **.ashx** denominados también **HttpHandlers**, que son objetos que invocan automáticamente al solicitar un tipo de archivo particular (por ejemplo dentro del directorio se tienen imágenes de tipo .jpeg).

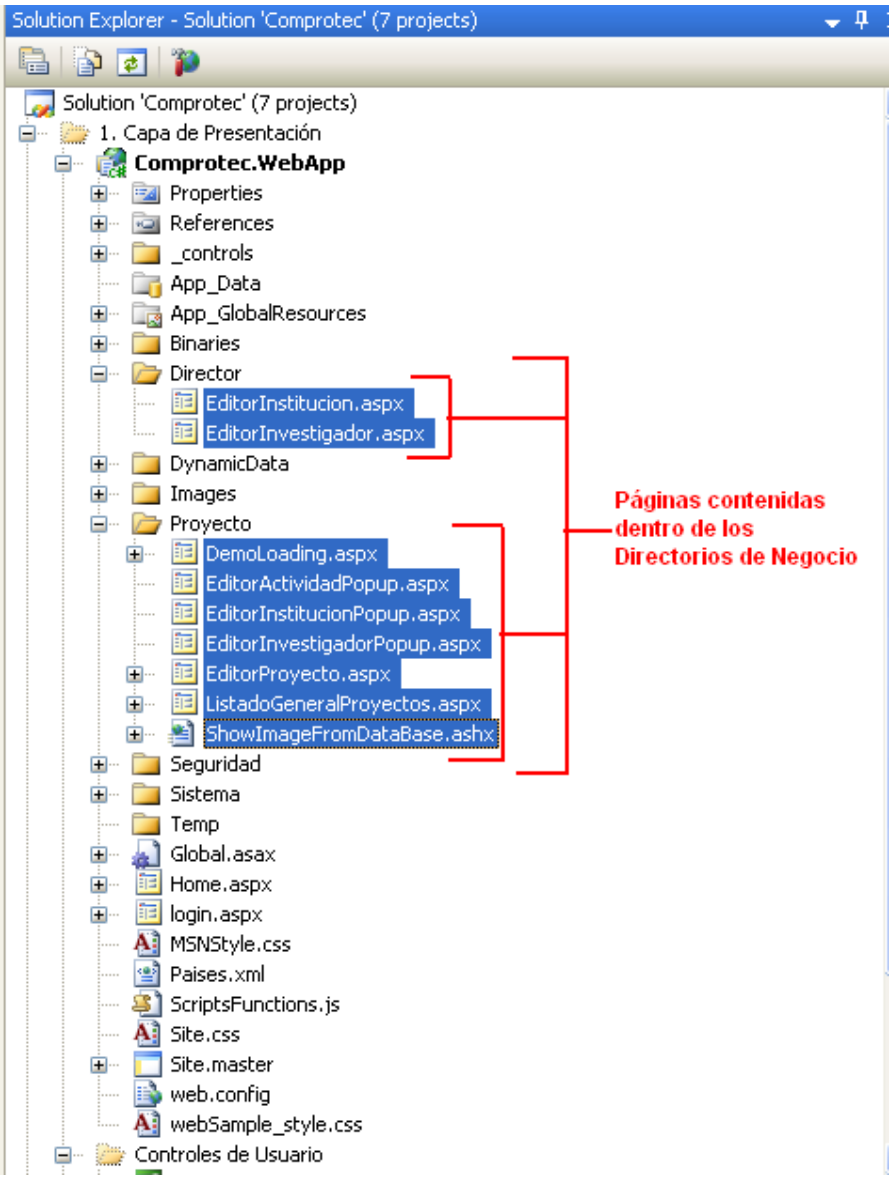


Figura 5.30: Directorio de páginas de Negocio

Directorio DynamicData

Este directorio posee un conjunto de páginas .aspx y controles de usuario .ascx utilizados para la generación de catálogos dinámicos que dan mantenimiento a los datos de las tablas del sistema.

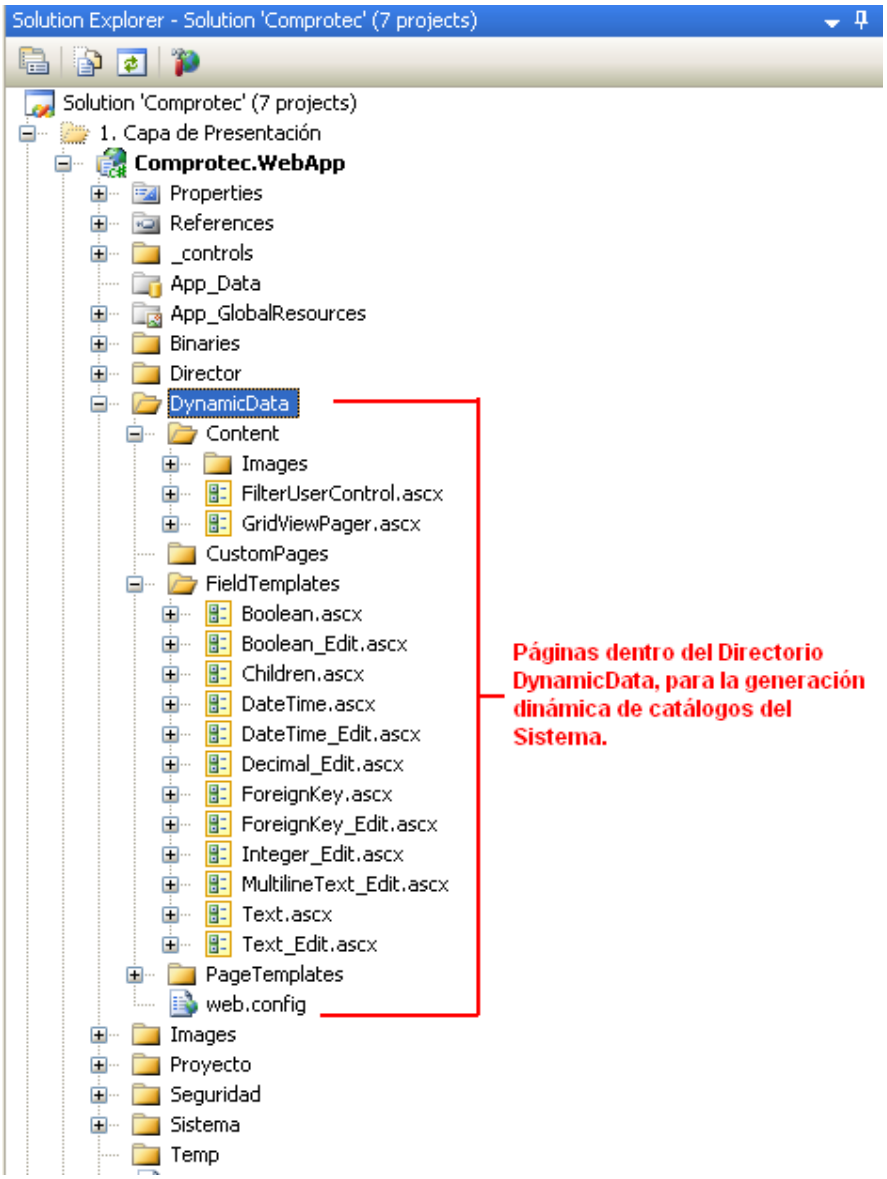


Figura 5.31: Directorio de DynamicData

Directorio Binaries

Dentro de este directorio se encuentran librerías gratuitas de terceros, que corresponden a controles de usuario enriquecidos y funcionalidad especializada para el manejo de ciertos aspectos de programación dentro de la aplicación HADE Comprotec; por ejemplo: controles de calendario, editores HTML de texto, etc.

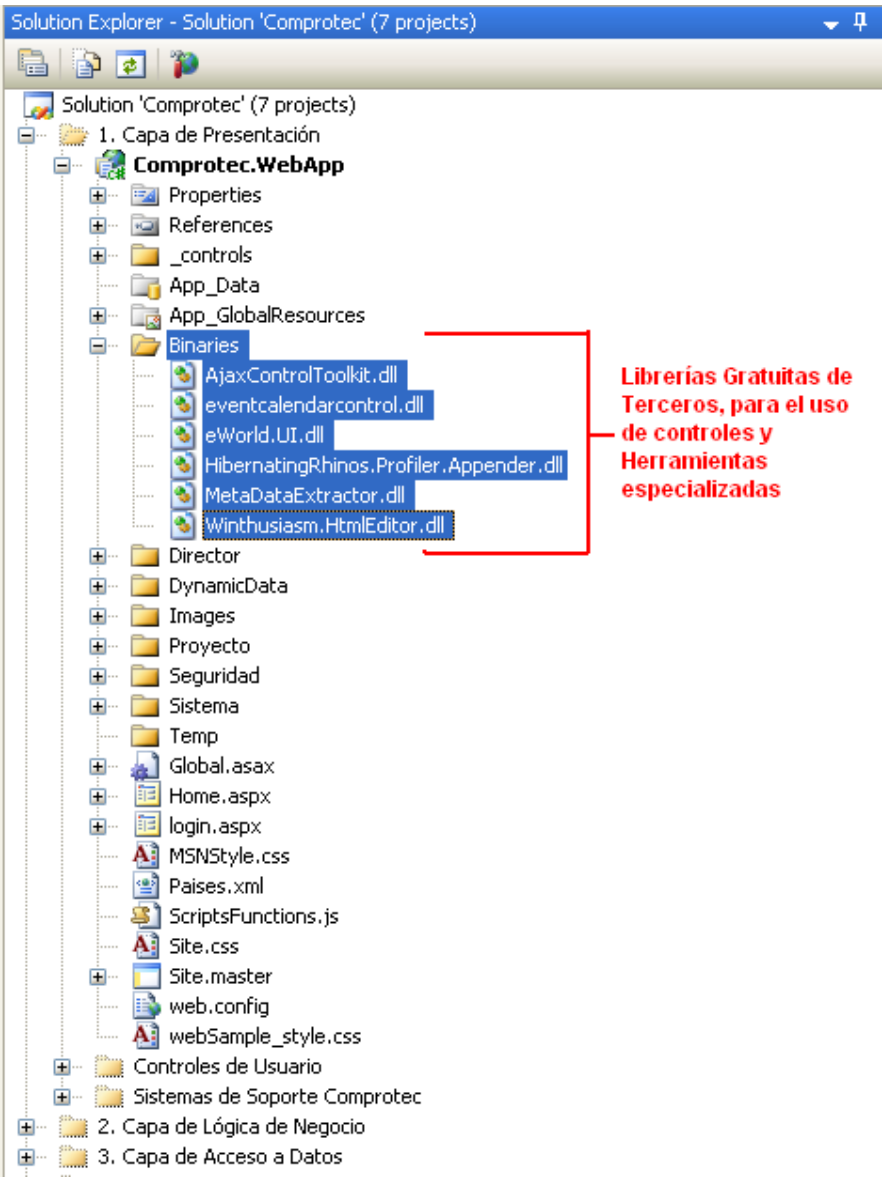


Figura 5.32: Directorio Binaries

Directorio Temp

Es el directorio donde se almacenan datos temporales generados por el sistema; por ejemplo: cuando se decide ver un documento adjunto, el documento que se encuentra

en la base de datos, se reconstruye y se guarda en el disco duro dentro del archivo temporal, para luego ser accedido a partir de su nueva ubicación física.

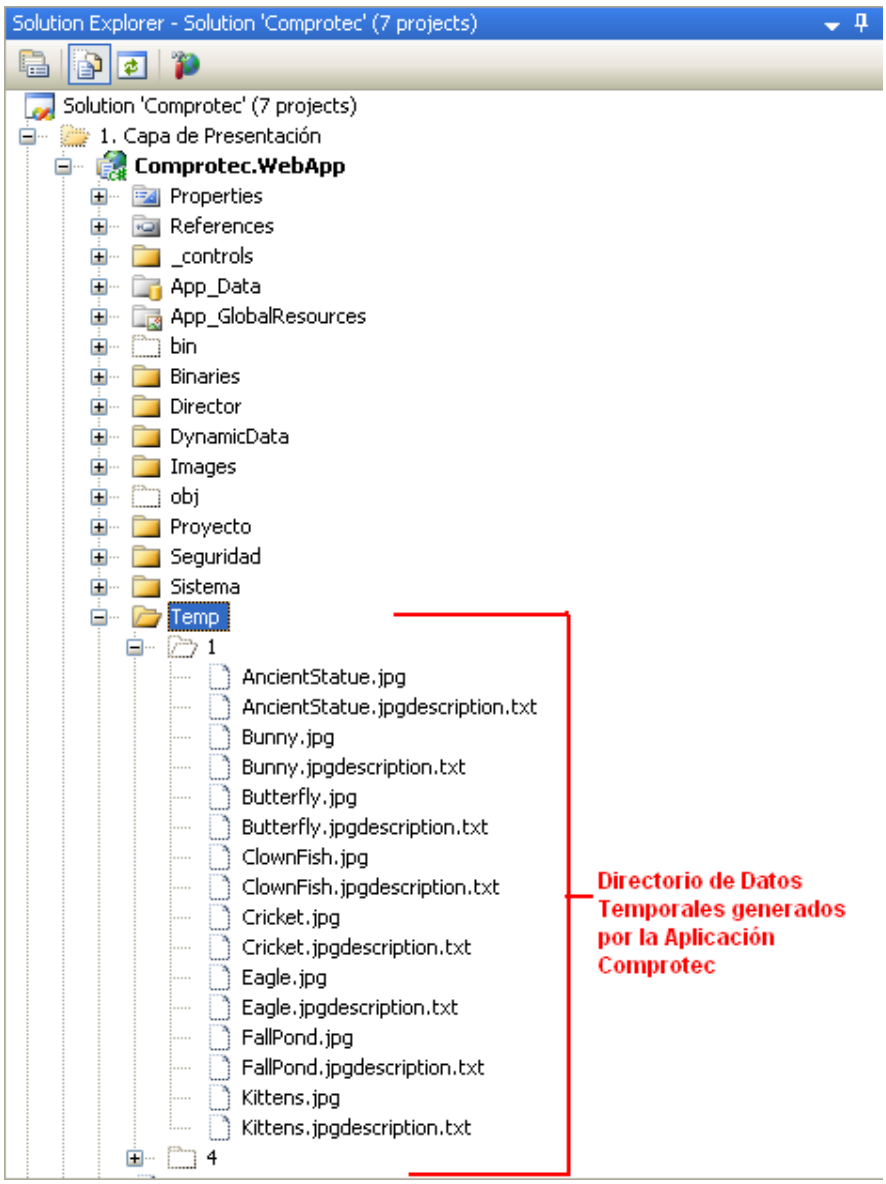


Figura 5.33: Directorio de Datos Temporales

Directorio de Imágenes

Contiene un conjunto de imágenes utilizadas en las páginas del proyecto.

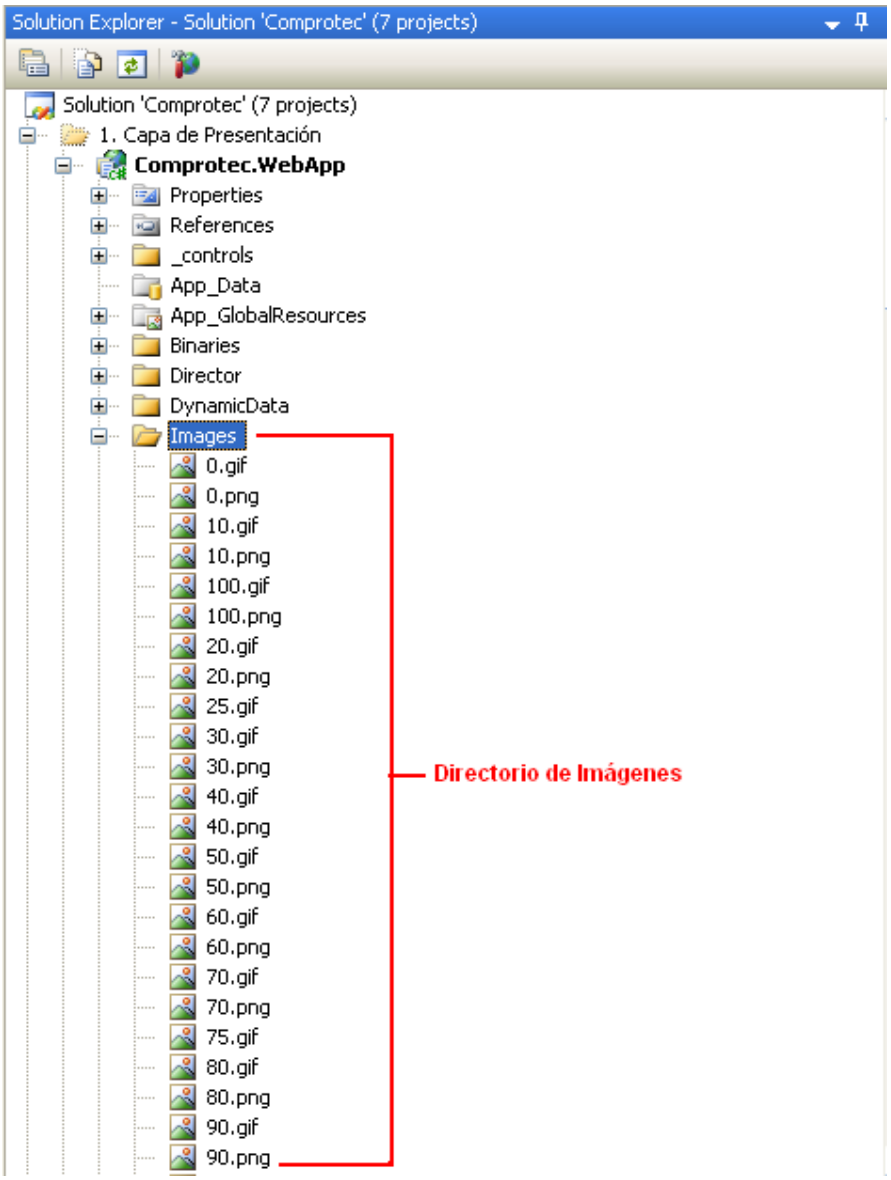


Figura 5.34: Directorio de Imágenes

Directorio App_Data y App_GlobalResources

Son directorios añadidos a la solución por el Visual Studio .Net. El directorio **App_Data** es encargado de almacenar archivos de base de datos de uso local; por ejemplo el archivo **ASPNETBD.mdf**, que contiene la Base de Datos para alojar la información de autorización y autenticación usada por el mecanismo de Seguridad **ASP.NET Membership**. El directorio **App_GlobalResources** aloja archivos de recursos usados por el aplicativo, estos archivos de recursos son predefinidos y generados por el .Net para la ejecución del sistema (por ejemplo, archivo para el manejo de múltiples idiomas).

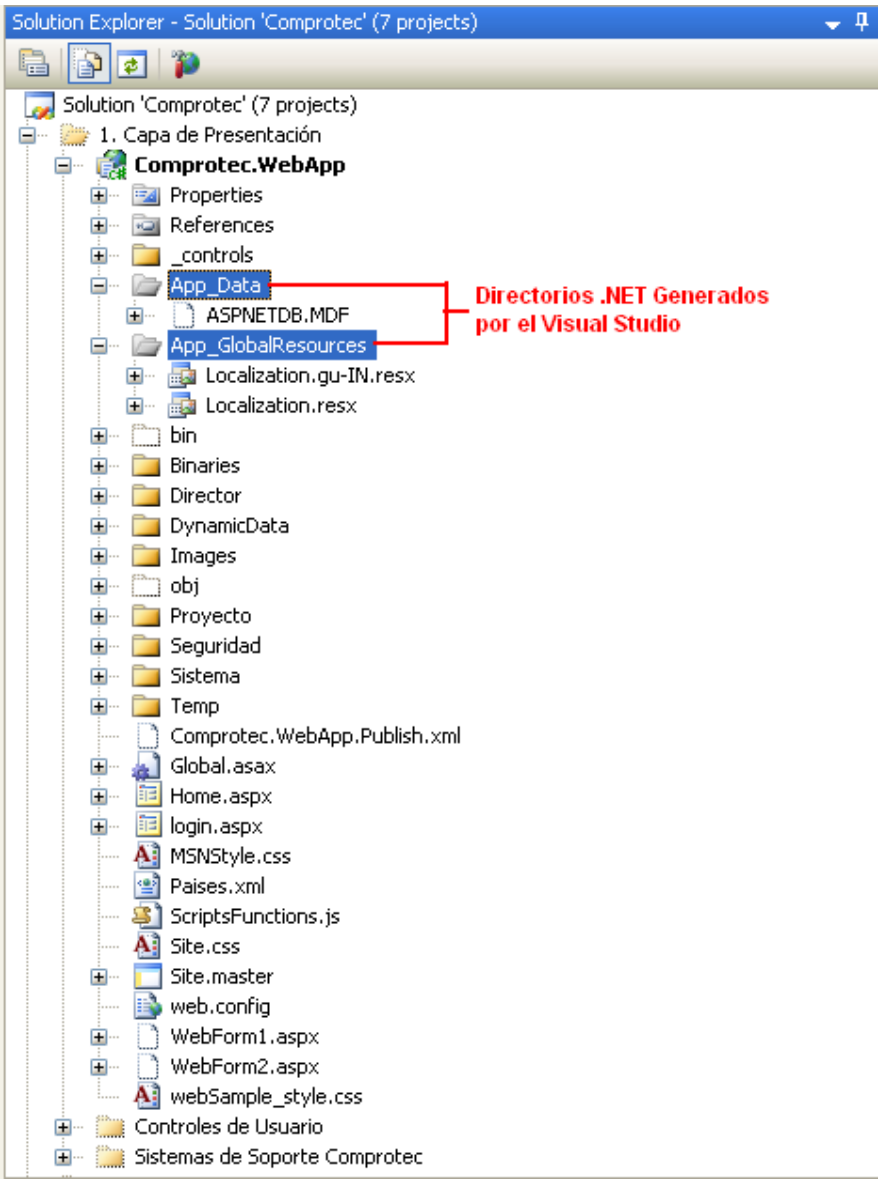


Figura 5.35: Directorios .NET generados por Visual Studio

Archivos en la Raíz de la Aplicación

Los archivos ubicados en la raíz son de acceso público y normalmente son el punto de partida a la aplicación, contiene la página de bienvenida, la página de inicio de sesión, etc.

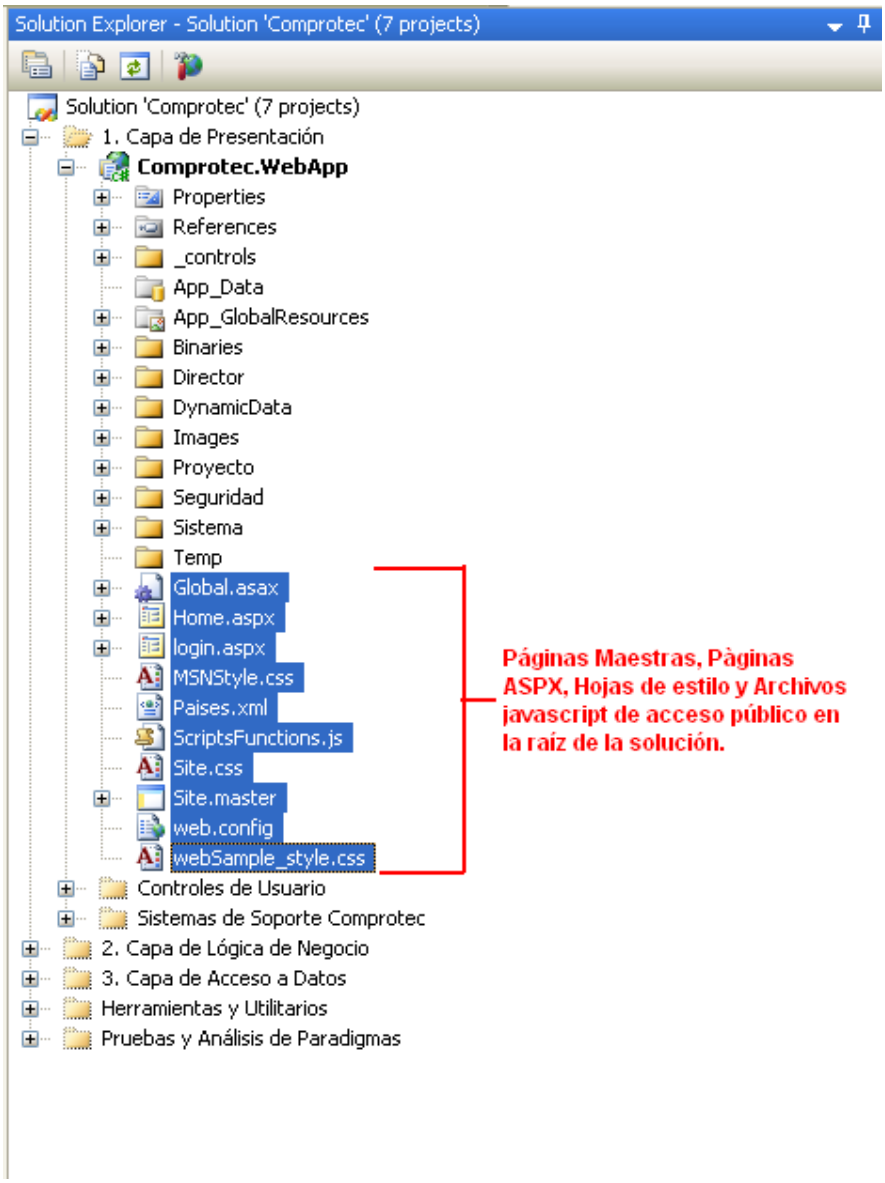


Figura 5.36: Archivos de la Raíz

Comprotec.UserControls

Es un proyecto de tipo Librería de clases, donde cada clase dentro de este proyecto corresponde a la implementación de un control de usuario personalizado utilizado en la aplicación Comprotec.WebApp.

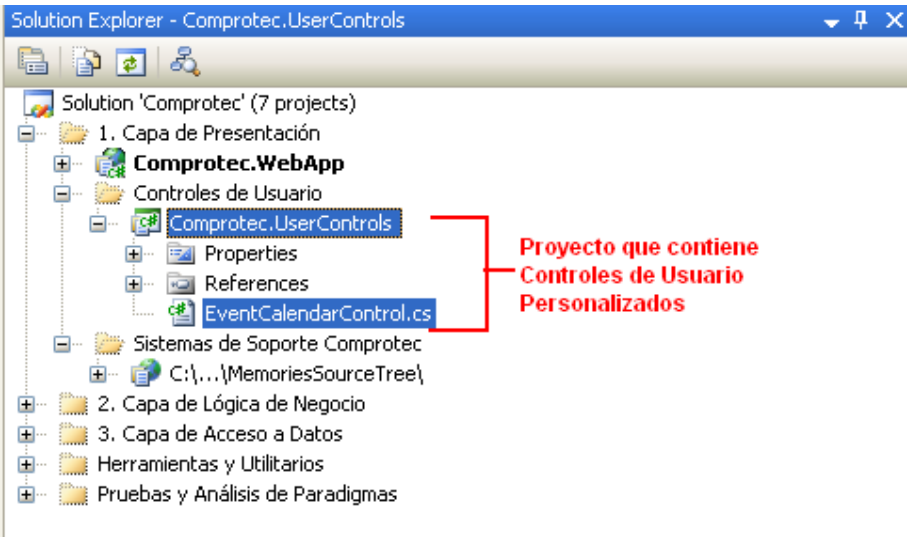


Figura 5.37: Proyecto que contiene los Controles de Usuario Personalizados

MemoriesSourceTree

Dentro del directorio Sistemas de Soporte Comprotec, el proyecto MemoriesSourceTree es un proyecto de tipo Web Site encargado de manejar y desplegar la galería de imágenes de un proyecto, las páginas de este sitio son invocadas desde la aplicación Comprotec.WebApp.

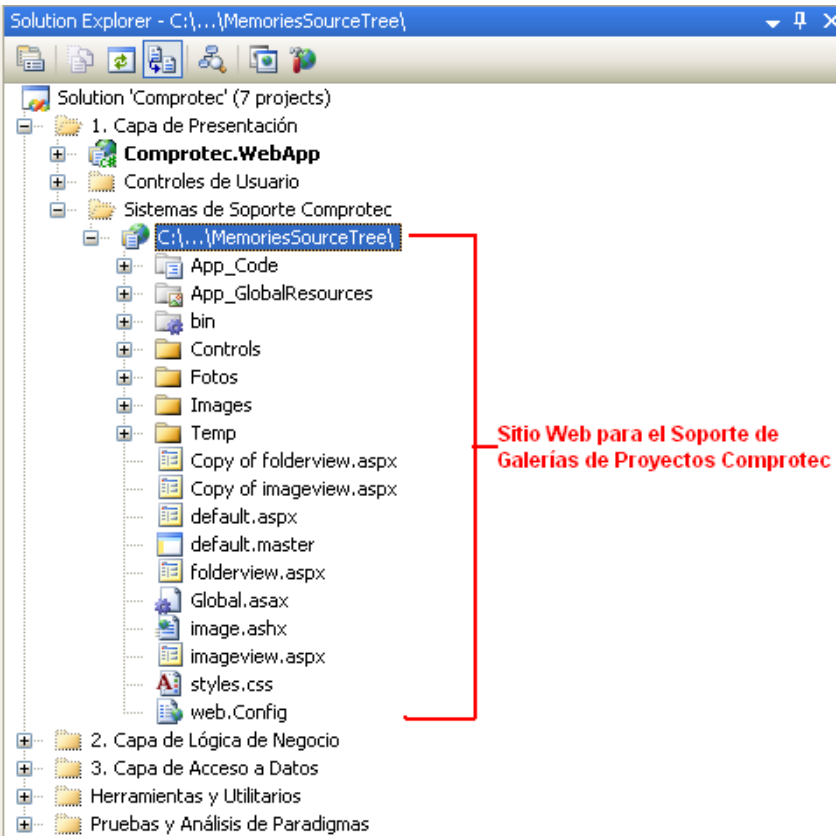


Figura 5.38: Sitio Web para el soporte de Galerías de Proyectos Comprotec

CONCLUSIONES

El modelo de Mapeo de Estructuras Relacionales a Objetos de Programación de LINQ, ofrece un valor agregado al brindarle al desarrollador la capacidad de utilizar los objetos de la base de datos como objetos de programación y aplicar características de Orientación a Objetos y otros beneficios, abriendo un abanico de posibilidades dentro del código.

Los operadores de consulta LINQ ofrecen al programador un conjunto de tareas predefinidas listas para usar; estas herramientas ahorran esfuerzo de implementación sobre cualquier colección de objetos. De esta forma, LINQ se convierte en una alternativa válida de desarrollo que ayuda a los programadores a ser mas productivos reduciendo significativamente el tiempo de codificación y mejorando notablemente la calidad del código.

La realización del Estudio Comparativo entre el Paradigma de Programación Imperativa más conocido como “Programación Tradicional” frente a “LINQ y expresiones Lambda” como paradigma de Programación Funcional para el desarrollo de una Aplicación, permitió determinar en base a algunas métricas cualitativas y cuantitativas que la Tecnología LINQ y Expresiones Lambda reduce el esfuerzo, complejidad y tiempo de codificación del programador ya que crea código comprensible, compacto y modela coherentemente los datos para trabajar en varios tipos de formatos y orígenes de datos convirtiéndose completamente en una ayuda válida de desarrollo.

Cabe resaltar que los datos obtenidos de las métricas cualitativas y cuantitativas para realizar el análisis de la hipótesis planteada, sirvieron como base para la toma de decisiones, con respecto a la elección del paradigma con el cual se desarrollo la Aplicación.

La utilización de la tecnología LINQ y Expresiones Lambda mejoró el tiempo de desarrollo del Sistema de Gestión de Proyectos denominado “HADE” aplicado a COMPROTEC-ESPOCH.

RECOMENDACIONES

El uso de LINQ es bastante apropiado para la construcción de páginas de catálogos que utilizan consultas planas sobre las tablas de una base de datos, es decir consultas en donde se obtiene todos los campos de la tabla.

Es altamente recomendable el uso de Operadores Estándar y Expresiones Lambda. La naturaleza de ser objetos precompilados aceleran la ejecución haciendo que el rendimiento en general mejore al procesar un lote de consultas.

Mediante el uso de LINQ se evitan el uso de bucles y recorridos simplificando enormemente la codificación. El uso de LINQ to Objects es recomendado en todas las capas de la aplicación: Acceso a Datos, Lógica de Negocio y Presentación.

GLOSARIO

ADO.- ActiveX Data Objects (Objetos de Datos ActiveX).

API. Application Programming Interface (Interfaz de programación de aplicación).

ASP. Active Server Pages (Páginas de servidor activas).

CIL. Common Intermediate Language (Lenguaje Intermedio Común, véase MSIL).

CLR. Common Language Runtime (Lenguaje Común de Ejecución).

CLS. Common Language Specification (Lenguaje Común de Especificación).

COM. Component Object Model (Modelo de objeto – componente).

COMPROTEC. Comisión de Proyectos y Transferencia de Tecnología

DLL. Dynamic Link Library (Librería de enlace dinámico).

ESPOCH. Escuela Superior Politécnica de Chimborazo.

IDE. Integrated Development Environment (Entorno de desarrollo integrado).

IL. Lenguaje Intermedio

IU. Interfaz de Usuario

TI. Tecnología de la información.

JSP. Java Server Pages (Páginas de Servidor Java).

LINQ. Lenguaje Integrado de Consultas(Languaje Integrated Query)

MFC. Microsoft Foundation Class Library

MI. Muestra Inicial

MSF. Microsoft Solutions Framework

MSIL. Microsoft Intermediate Language (Lenguaje Intermedio de Microsoft).

O/R Designer. Objeto Diseñador Relacional

SOAP. Simple Object Access Protocol (Protocolo de Acceso a Objeto Simple).

SQL. Structured Query Language (Lenguaje estructurado de consulta).

SRS.- Software Requirement Specification (Especificación de Requisitos de Software).

TI. Tecnología de Información

VSTO. Visual Studio Tools for Office

XML. eXtended Markup Language (Lenguaje de marcado extendido).

RESUMEN

Estudio comparativo sobre la Programación Tradicional frente a LINQ y EXPRESIONES LAMBDA para el Desarrollo del Sistema de Gestión de Proyectos “HADE” aplicado a COMPROTEC-ESPOCH esta técnica permite mejorar la productividad del programador.

Se utilizo: Visual Studio 2008, la plataforma .NET Framework 3.5, lenguaje de programación C#, LINQ y Expresiones Lambda, SQL Server 2000, Métodos de Investigación: Científico y Comparativo, Un Modelo de Evaluación y Pruebas de Microsoft aplicada para demostrar la hipótesis, Metodología Microsoft Solution Framework (MSF) para el desarrollo del Sistema.

Se obtuvo como resultado en el estudio que, en las métricas cualitativas al momento de desarrollar una aplicación con LINQ y Expresiones Lambda, dado un 100%, se tiene un 45% de productividad en la Programación Tradicional, mientras que LINQ y Expresiones Lambda muestra un 55% mejorando la programación del mismo; En el caso de las métricas cuantitativas, la programación con LINQ posee un 88% de productividad frente al 12% que presenta el programar tradicionalmente.

La implementación con LINQ y Expresiones Lambda como Paradigma Funcional representa la técnica de programación más productiva, porque reduce el esfuerzo del programador en un 76% en la construcción de sistemas informáticos.

Se recomienda a los nuevos programadores que, cuando se desarrolle un sistema informático, se utilice la nueva tecnología LINQ por ser una técnica de programación más productiva ya que permite utilizar objetos de la base de datos como objetos de programación.

SUMMARY

This is a comparative study on the Traditional Comparison against a LINQ and Expressions Lambda for the Project Management System Development "HADE" applied to COMPROTEC-ESPOCH. This technique permits to improve the programmer productivity.

Visual Studio 2008, .Net Framework 3.5 platform, C# Programming Language, LINQ and Expressions Lambda, SQL Server 2000 were used. The investigation methods were: Scientific and Comparative, as well as an Evaluation Model and Microsoft Tests applied to demonstrate the hypothesis, Microsoft Solution Framework Methodology(MSF) for the System development.

As the study result, in the qualitative meters, the moment of applying LINQ and Expressions Lambda, given a 100%, there is 45% productivity in the Traditional Programming, while LINQ and Expressions Lambda show a 55% improving its programming. In case of quantitative meters, the LINQ programming has 88% productivity against 12% presented by the traditional programming.

The LINQ and Expressions Lambda implementation as a functional paradigm represents the most productive programming technique because it reduces the programmer effort by 76% in the informatics systems construction.

The new programmers are recommend led to use the new LINQ technology when developing an informatics system, because it is the most productive programming technique, as it permits to use database objects as programming objects

BIBLIOGRAFÍA

LIBROS

1. **FERRACCHIATI**, Fabio Claudio. LINQ for Visual C# 2005. New York: Springer Verlag Inc., 2007. 174 p.

2. **HUMMEL**, Joe. LINQ: The Future of Data Access in C# 3.0. Washington: O'Reilly Media, Inc., 2007. 60 p.

3. **KLEIN,** Scott. Professional LINQ. Indianapolis: Wiley Publishing, Inc., 2008. 410 p.

4. **PIALORSI**, Paolo. RUSSO Marco. Introducing Microsoft LINQ. Washington: Microsoft Press, 2007. 203 p.

5. **RATTZ**, Joseph Jr. Pro LINQ: Language Integrated Query in C# 2008. Washington: APress, 2007. 600 p.

SITIOS WEB VISITADOS

1. **BEAN SOFTWARE**. Create Your Own Web Site Administration Tool in ASP.NET
   http://www.beansoftware.com/ASP.NET-Tutorials/Web-Site-Administration-Tool.aspx
   2009-04-19
2. **BONELLI,** Eduardo. Programación Imperativa.
   http://eabonelli.googlepages.com/clasePI1.pdf
   2008-07-25
3. **CELULA UNT.** Introducción de LINQ a XML
   http://celulaunt.net/community/blogs/edoswit/archive/2008/04/14/introduc ci-243-n-de-linq-a-xml.aspx
   2009-03-14
4. **CODEPROJECT.** Developing MS Project application using Asp.net
   http://www.codeproject.com/KB/aspnet/MS_project_and_Aspnet.asp
   2009-06-23

5. **CODEPROJECT.** Editable Nested GridView (All-in-One)
http://www.codeproject.com/aspnet/EditNestedDataGrid.asp
2009-07-26

6. **COMUNIDAD .NET MÉRIDA.** The .NET Language Integrated Query Framework
http://groups.msn.com/aluxes-net
2009-07-26

7. DEVELOPER. Configuring Your ASP.NET 2.0 Site
http://www.developer.com/net/asp/article.php/3569166
2009-06-01

8. **EXPOVISION.** Comparación Paradigma Imperativo
http://expo.itch.edu.mx/m.php?f=oop_21&t=Programaci%F3n+Orientado +a+Objetos%3A+Comparaci%F3n+Paradigma+Imperativo
2009-08-13

9. **FRANCESC,** Jaumot**.** Que es LINQ?
http://fjaumot.wordpress.com/2007/11/23/que-es-linq-i/
2009-03-19

10. **FOKKER,** Jeroen, Programación Funcional.
http://people.cs.uu.nl/jeroen/courses/fp-sp.pdf
2008-07-25

11. **GEEKS.** Hablemos de LINQ
http://geeks.ms/blogs/mredison/archive/2007/12.aspx
2009-10-06

12. **HOOKED ON LINQ.** LINQ to Objects - 5 Minute Overview
http://www.hookedonlinq.com/LINQtoObjects5MinuteOverview.ashx
2010-01-15

13. **LANGUAGE INTEGRATED QUERY.** Características del lenguaje: Orígenes de datos
http://es.wikipedia.org/w/index.php?title=LINQ&redirect=no
2009-01-12

14. **LÓPEZ,** Pablo. Paradigmas de Lenguajes de programación
http://www-2.dc.uba.ar/materias/plp/20071C/download/PF-Paradigmas-UBA.pdf
2008-08-15

15. **MICROSOFT .NET FRAMEWORK.** Creando La Capa de Acceso a Datos con LINQ
http://www.logiqsoft.com/Creando%20La%20Capa%20de%20Acceso%20a%20Datos%20con%20LINQ.pdf
2009-02-15

16. **MICROSOFT .NET FRAMEWORK.** Creando La Capa de Acceso a Datos con LINQ
http://www.logiqsoft.com/Creando%20La%20Capa%20de%20Acceso%20a%20Datos%20con%20LINQ.pdf
2009-02-15

17. **MSDN.** Árboles de expresiones
http://msdn.microsoft.com/es-es/library/bb397951.aspx
2009-09-29

18. **MSDN.** Building a LINQ Provider
http://msdn.microsoft.com/en-us/vcsharp/aa336766.aspx
2010-02-24

19. **MSDN.** Expresiones de consultas con LINQ
http://msdn.microsoft.com/es-es/library/bb397676.aspx
2009-09-26

20. **.NET FRAMEWORK DEVELOPER CENTER. LINQ**: .NET Language-Integrated Query.
http://msdn.microsoft.com/en-us/library/bb308959.aspx.
2009-01-12

21. **NET HEAVEN.** Creating ASP.NET Photo Album using FileSystem as Data
http://www.dotnetheaven.com/Articles/ArticleListing.aspx?SectionID=0&SubSectionID=11
2010-03-14

22. **OPERADORES DE CONSULTA ESTÁNDAR CON LINQ**. Descripción de las nuevas tecnologías de Microsoft.
http://geeks.ms/blogs/mredison/archive/2008/04/30/operadores_2D00_de_2D00_consulta_2D00_estndar_2D00_con_2D00_linq.aspx
2008-09-10

23. **PASIÓN POR LA TECNOLOGÍA.** Archivos de la categoría 'LINQ'
http://jcgonzalezmartin.wordpress.com/category/linq/.
2009-03-14

24. **SIDAR.** 10 Tips to Improve your LINQ to SQL Application Performance
http://www.sidarok.com/web/blog/content/2008/05/02/10-tips-to-improve-your-linq-to-sql-application-performance.html
2010-04-21

25. **TECNOLOGIAS MICROSOFT**. Introducción de la herramienta LINQ
http://www.microsoft.com/msdn/download
2008-12-13

26. **THINKING IN .NET.** Usando LINQ to SQL
http://thinkingindotnet.wordpress.com/2007/05/20/usando-linq-to-sql-1%c2%aa-parte/.
2009-01-15

27. UNTIE LINQ TO SQL CONNECTION STRING FROM APPLICATION SETTINGS. Programming & Life
http://goneale.com/
2009-05-15

28. **WIKIPEDIA**. Language Integrated Query
http://en.wikipedia.org/wiki/LINQ#Architecture
2009-08-25

29. **WIKIPEDIA.** .NET Language-Integrated Query
http://msdn2.microsoft.com/en-library/bb308959.aspx#linqoverview_topic1
2009-08-15

ANEXOS

SELECCIÓN DE HERRAMIENTAS ESPECIALIZADAS PARA EL ANÁLISIS Y MEDICIÓN

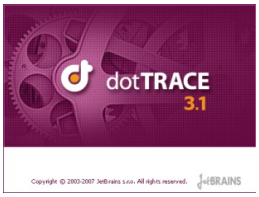
En el mercado existen muchas herramientas especializadas en la realización del Profiling y Benchmarking de aplicaciones. El Profiling consiste en la monitorización de procesos y medición de los recursos utilizados. El Benchmarking consiste en la comparación y de análisis de variables para identificar el proceso más óptimo en términos de ejecución y manejo de recursos.

Para el presente estudio, la herramientas a utilizar deberán poseer las siguientes características:

- Obtención detallada de la ejecución de un programa: Árbol de procesos y subprocesos ejecutados y el peso o porcentaje de ejecución que representa cada proceso dentro de la ejecución principal.
- Tiempo de Ejecución en procesador por cada proceso o hebra de la ejecución.
- Uso de memoria de los objetos utilizadas por cada proceso o hebra de la ejecución.
- Gran acoplamiento e integración para aplicaciones .NET.
- Generación de Snapshots o Instantáneas de una ejecución en particular.

En base a sus mejores prácticas para revisión y evaluación de aplicaciones .NET; Microsoft recomienda el uso de la herramienta: JetBrians dotTrace ya que brinda todas las características mencionadas anteriormente y un fuerte soporte para la medición precisa de aplicación .NET y del .NET Framework 3.5, que es la versión que del framework que soporta la tecnología LINQ.

JetBrains dotTrace



La herramienta **dot Trace** de la compañía Jet Brains brinda un conjunto de funcionalidades para una revisión completa de una aplicación .NET de cualquier tipo: Windows, Web y Windows Service. Para el presente estudio, usaremos la funcionalidad de profiling para aplicaciones Windows Forms.

Al lanzar la herramienta se presentará la pantalla de bienvenida mediante la cual se puede seleccionar el tipo de profiling.

Figura: Herramienta dotTRACE Profiler

Al elegir la opción Profile Application, el programa mostrará la pantalla de configuración de la traza para el profiling de la aplicación.

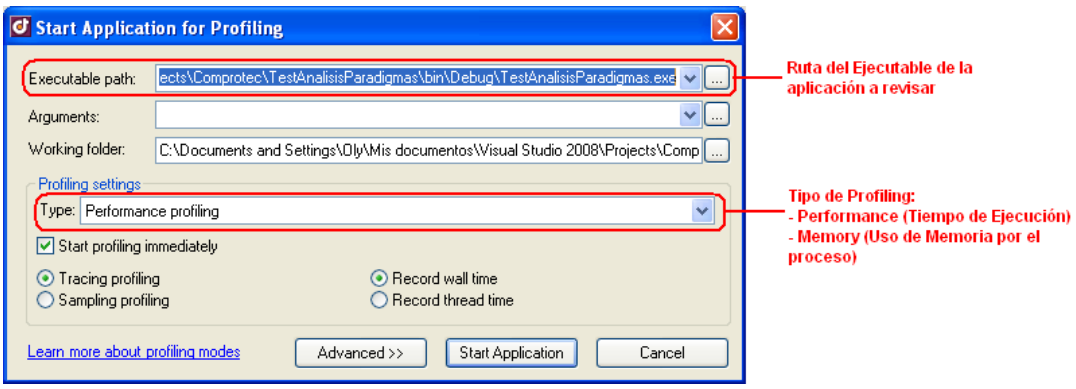


Figura: Configuración de la herramienta dotTRACE Profiler dotTRACE

Luego de seleccionar el ejecutable de la aplicación a evaluar y de escoger el tipo de profiling: Performance (Tiempo de Ejecución) ó Memory (Uso de Memoria usado por los objetos del proceso), se inicia el profiling de la aplicación seleccionada. Inmediatamente, se inicia la aplicación, junto con un tablero de Control de Profiling mediante el cual se puede controlar la ejecución.

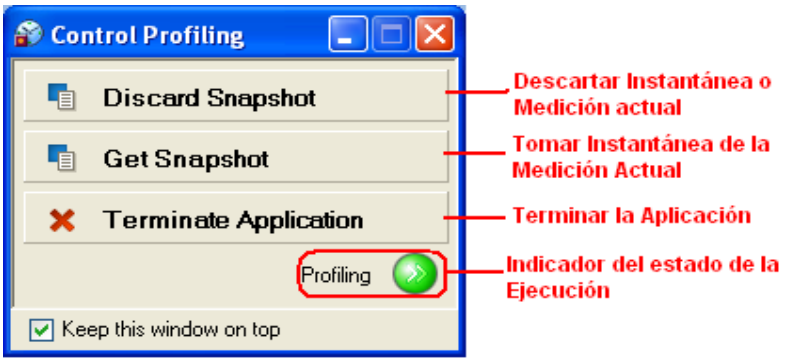


Figura : Ventana de Control Profiling

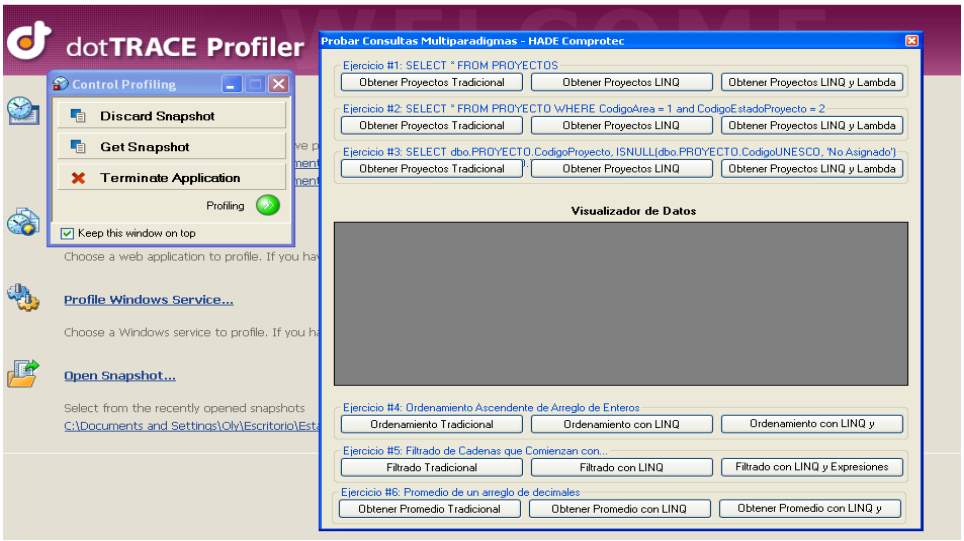


Figura : Selección del método para obtener sus mediciones

Al dar clic sobre un botón se iniciará la ejecución del método respectivo. Al mostrarse el mensaje de ejecución finalizada, es el momento para detener la traza y tomar una instantánea para revisar los datos resultantes de las mediciones.

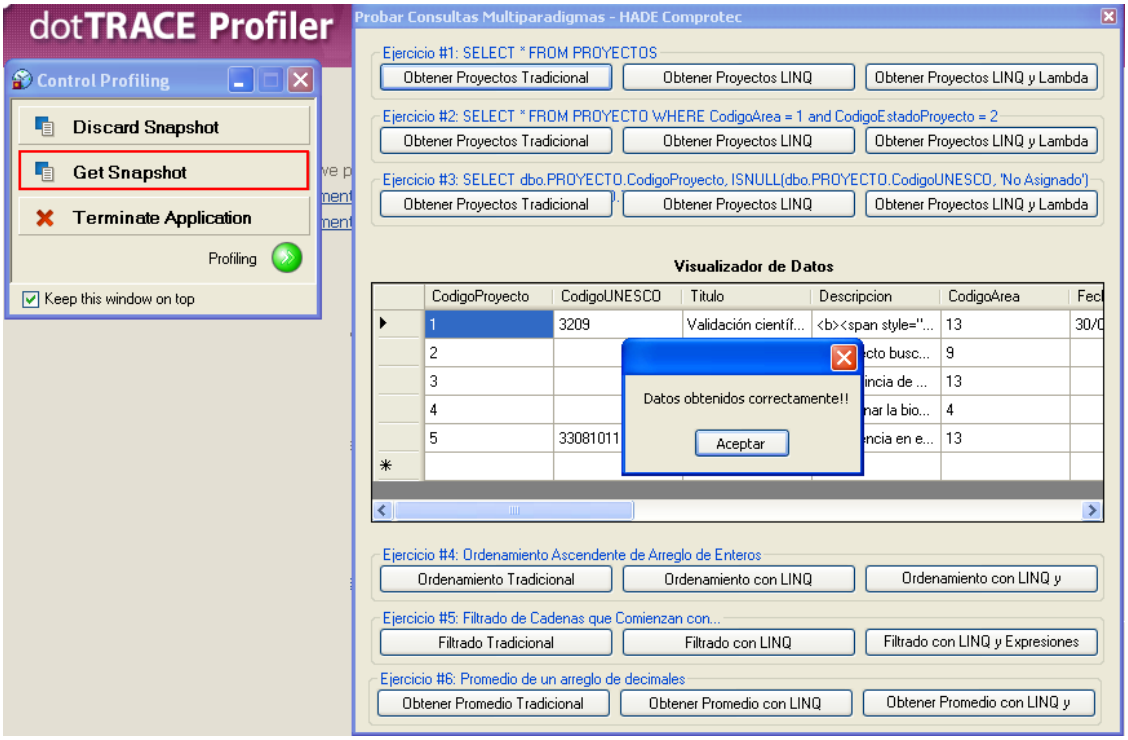


Figura : Ejecución del método con sus mediciones

Luego de dar clic en el botón Get Snapshot de la ventana de Control de Profiling, se cargarán los resultados capturados durante la ejecución de la aplicación.

Taladrando por el árbol de hebras, se puede identificar los procesos y subprocesos ejecutados durante toda la ejecución total de la aplicación. Entre estos procesos se puede el evento click del botón y dentro del ámbito de su ejecución, se identificará el método que implementa el ejercicio correspondiente.

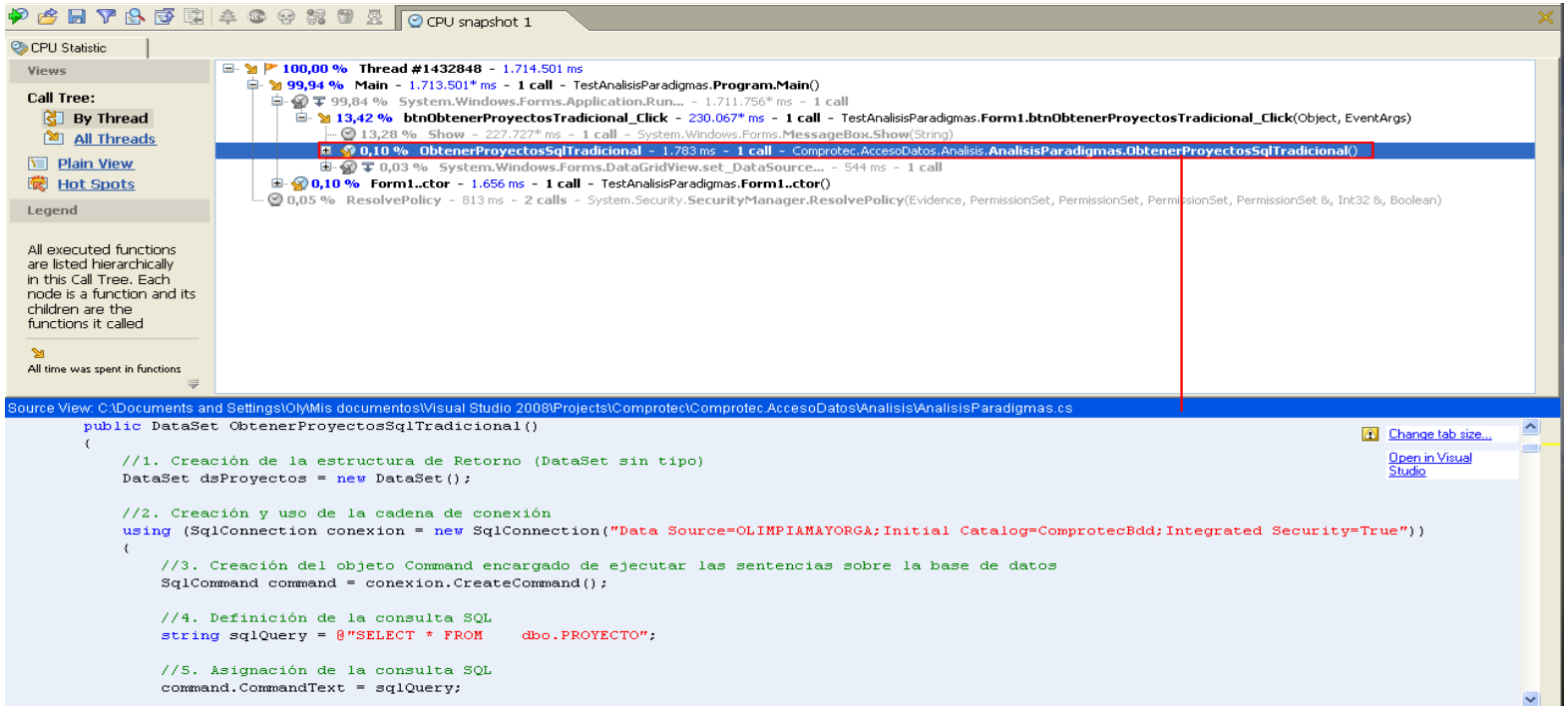


Figura : Valor de medición del Método ejecutado

Al seleccionar un proceso, se puede observar el código fuente del método seleccionado.

CRL Profiler

El CLR Profiler es una herramienta gratuita provista por Microsoft para la revisión y profiling de aplicaciones de aplicaciones .NET. Permite al usuario investigar el contenido de la gestión de pila, así como el comportamiento del recolector de basura (Garbage Collector), para identificar las porciones de código que utiliza demasiada memoria. Provee de gráficos de la ejecución de los procesos internos y gráficos estadísticos.

Su uso es bastante simple. Al iniciar el programa, se debe dar clic sobre el botón **Start Application** y seleccionar el archivo ejecutable del aplicativo que se desea revisar.

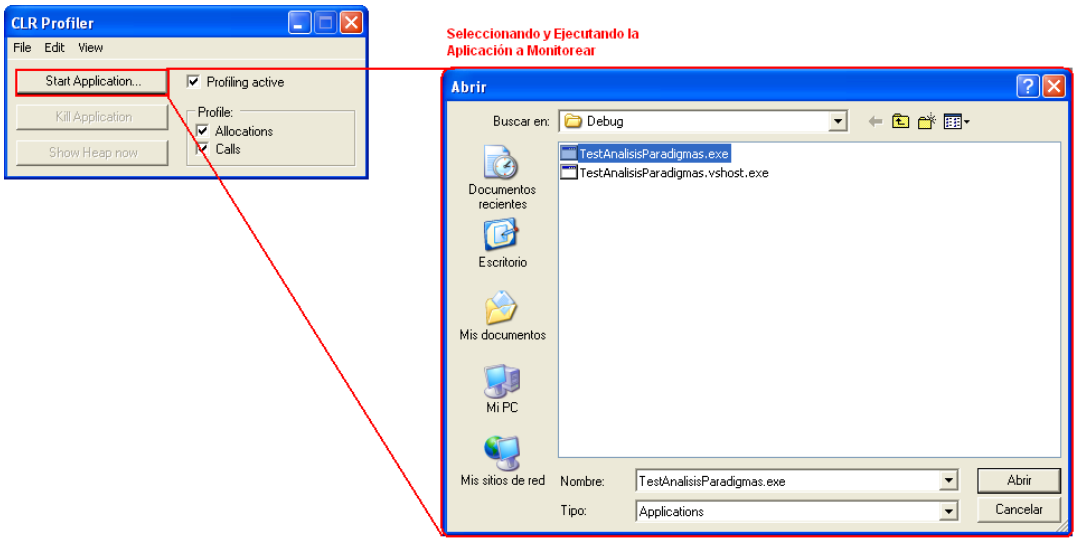


Figura : Herramienta CLR Profiler

Una vez que se encuentra en ejecución el aplicativo de prueba, se ejecuta la acción que se desea monitorear.

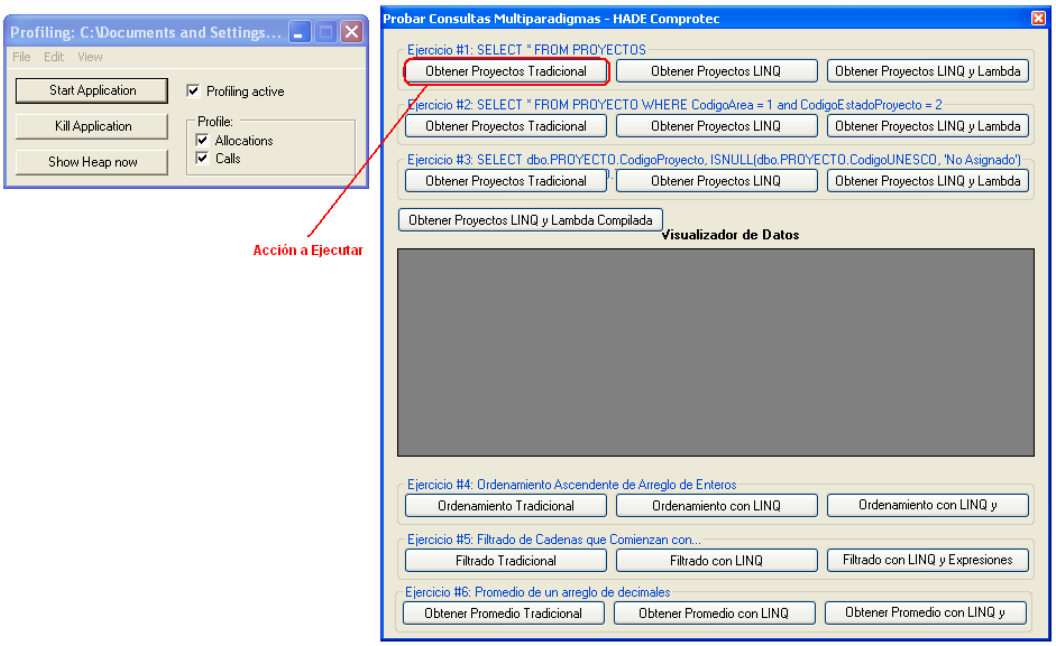


Figura : Selección de la acción a ejecutar

Luego de la acción, se finaliza la ejecución del aplicativo de prueba para posteriormente revisar los resultados.

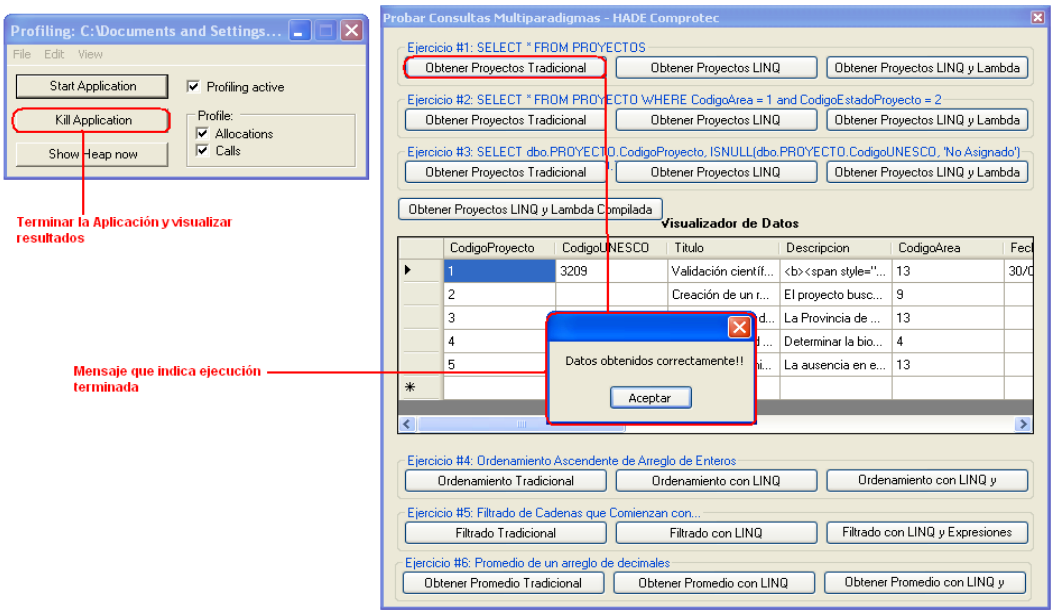


Figura : Revisión de los resultados obtenidos

Al finalizar el aplicativo de prueba, aparecerá una ventana de Control de Resultados, de donde se seleccionará la opción: **Allocation Graph,** encargada de dibujar el gráfico de la ejecución del programa evaluado.

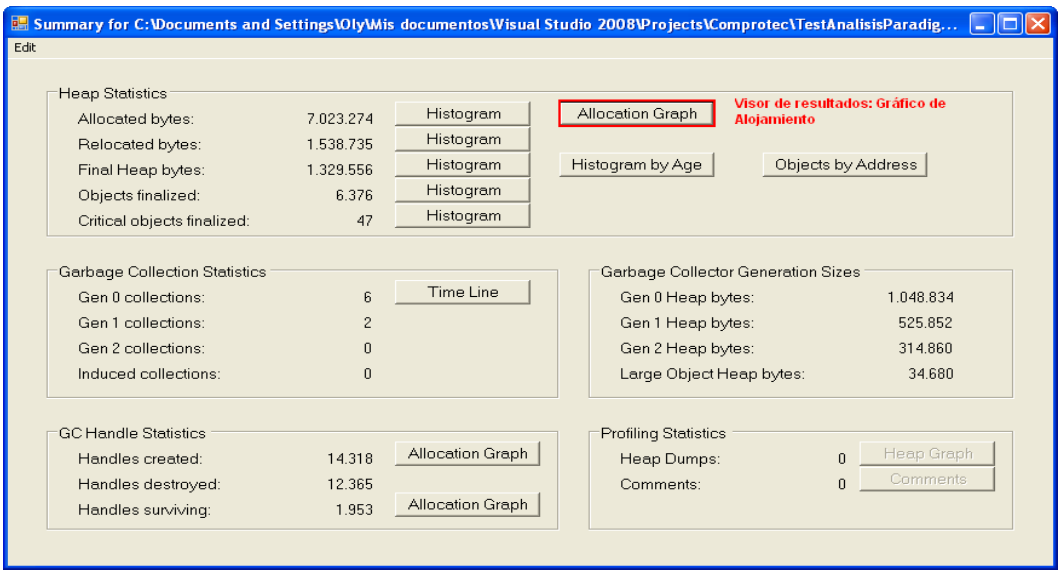


Figura : Ventana de control de resultados

Una vez dentro del gráfico, se identifica el método invocado y se podrá apreciar la cantidad de memoria usada, así como también las invocaciones internas de los objetos de .NET.

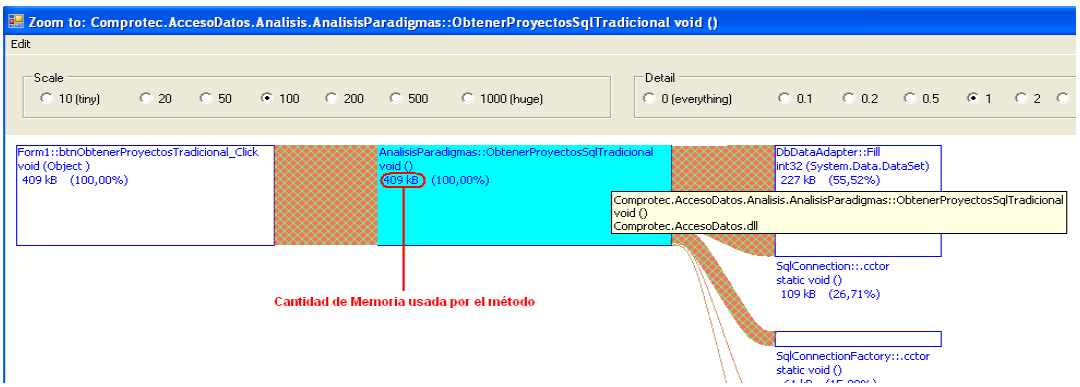


Figura : Apreciación de los resultados del uso de la memoria

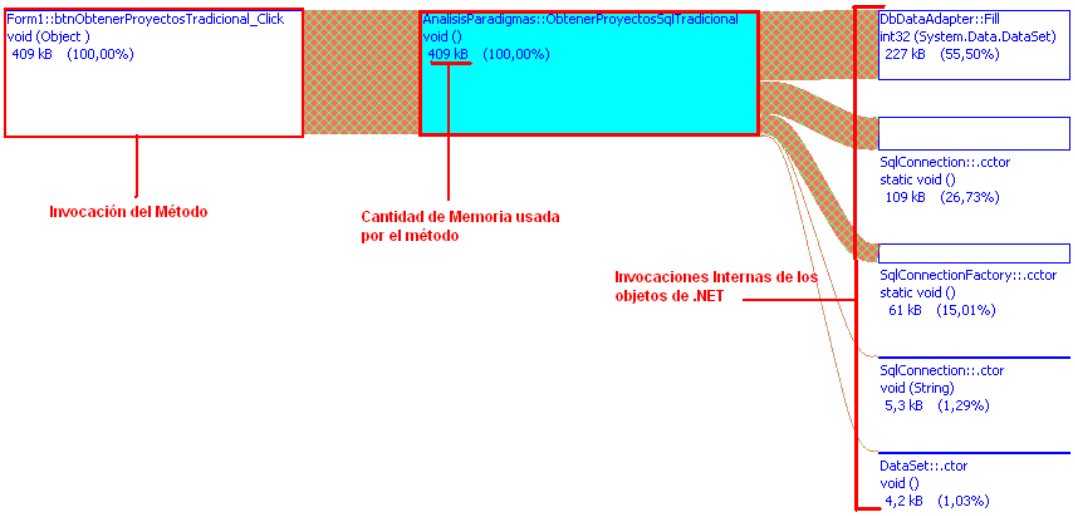


Figura : Detalles que se muestran de los resultados del uso de la memoria