Universidad de las Ciencias Informáticas

Facultad 3

Título: Desarrollo de un sistema para la gestión de las excepciones y trazas de software en una aplicación implementada sobre Arquitectura .NET.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informática

Autor: Abdel Pérez López

Tutor(es): Ing. Yunier Pérez Barroso

Ing. Jorge Yuniel Jorrin Perdomo

Junio 2009

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de ________ del año ________.

<nombre autor>

____
Firma del Autor

<nombre tutor>	**<nombre tutor>**
_______________	_______________
Firma del Tutor	Firma del Autor

DATOS DE CONTACTO

Autor: *Abdel Pérez Lopez*

Correo Electrónico: *aplopez@estudiantes.uci.cu*

Tutor: *Ing. Yunier Pérez Barroso*

Correo Electrónico: *ypbarroso@uci.cu*

Tutor: *Ing. Jorge Yuniel Jorrin Perdomo*

Correo Electrónico: *jyjorrin@uci.cu*

AGRADECIMIENTOS

Agradezco a Dios por darme sabiduría y fuerzas para recorrer este largo camino que ha sido mi educación profesional.

A Mami y a Papi, que con sus consejos, educación y apoyo lograron convertirme en el hombre que soy hoy.

A mi Hermano, que lo quiero mucho y sin su ejemplo nunca hubiera intentado lograr llegar a donde estoy hoy.

A mi Tati que sin su ayuda no habría llegado hasta aquí, por ser mi amiga, mi novia, mi todo, mi vida.

A mis abuelos Elita, Oscar gracias por su amor y en especial a mamá vieja (Abigail) que aunque no se encuentre ya conmigo, se que desde algún lugar esta observándome y cuidándome como siempre lo hizo.

A mis tutores Yunier y Jorrin por haberme soportado todo este tiempo gracias por su apoyo y ayuda.

A mis amigos todos que aunque no lo sepan siempre los tengo en mi corazón y son parte importante de mi vida.

En general a todos los que me brindaron su ayuda y apoyo; aunque fuera pequeña, en toda mi vida de estudiante.

DEDICATORIA

A mis padres que siempre han estado allí cuando los he necesitado

A mi hermano gracias por existir no imagino uno mejor que tú.

A mi toda mi familia que siempre me han apoyado de una forma u otra.

A mi Tati gracias por estar siempre a mi lado apoyándome y dándome tanto amor, eres mi alegría y el encanto de mis días, no imagino una vida sin ti, Te amo.

A esa segunda madre (Sandra) que entró a mi vida gracias a mi amor.

A todos mis amigos que son parte importante de mí.

RESUMEN

El trabajo presentó una propuesta de solución informática para la gestión de las excepciones y trazas de software, en el desarrollo de aplicaciones con arquitectura .NET. Se expuso la plataforma sobre la cual fue desarrollado el sistema. Además las técnicas de programación empleadas en el desarrollo del software, así como los artefactos obtenidos durante el mismo. Y finalmente se mostró el proceso de prueba al que se sometió al sistema obtenido; para su validación.

PALABRAS CLAVES

Excepciones, Arquitectura .NET, desarrollo.

ÍNDICE

DATOS DE CONTACTO III

AGRADECIMIENTOS IV

DEDICATORIA V

RESUMEN VI

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 7

1.1. Tratamiento de excepciones 7

1.1.1. ¿Qué es excepción de software? 7

1.1.2. Excepciones en C#.NET 8

1.1.3. Análisis de Sistemas Existentes 11

1.2. Paradigmas de Programación 13

1.2.1. Programación Orientada a Objetos 13

1.2.2. Programación Orientada a Eventos 15

1.2.3. Programación Orientada a Componentes 16

1.3. Proceso de Desarrollo de Software 18

1.4. Metodología de Desarrollo de Software 19

1.4.1. Programación Extrema (XP) 19

1.4.2. Rational Unified Process (RUP) 21

1.4.3. Breve análisis de la metodología seguir. 23

1.5. Artefactos de la Disciplina de Diseño 24

1.5.1. Principales Artefactos 25

1.6. Artefactos de la Disciplina de Implementación 28

1.6.1. Principales Artefactos 28

1.7. Plataforma de Desarrollo Microsoft.Net....30
1.7.1. Lenguaje de Programación C-Sharp (C#)....33
1.8. Patrones de Diseño....34
1.8.1. Fábrica Abstracta (Abstract Factory)....35
1.8.2. Singleton....36
1.8.3. Fachada (Facade)....36
1.9. Herramientas para el modelado....37
1.9.1. Visual Paradigm....38
1.9.2. Rational Rose....38
1.9.3. Herramienta de Modelado Enterprise Architect....39
1.10. Métricas....41
1.10.1. Métricas del Modelo de Diseño....41
1.10.2. Métricas de Prueba....45
1.11. Arquitectura de software....47
1.12. Conclusiones....48
CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA....49
2.1. Introducción....49
2.2. Breve descripción de la Arquitectura....49
2.2.1. Arquitectura en Capas....49
2.2.2. Modelo basado en capas....52
2.3. Modelo de Diseño....55
2.3.1. Clases del Diseño....55
2.3.2. Patrones de Diseño....60
2.3.3. Principales Casos de Uso del Sistema....61

2.3.4. Modelo de Datos....69
2.3.5. Modelo de despliegue....71
2.4. Estándar de Codificación. 71
2.4.1. Reglas a seguir....72
2.4.2. Pautas de Diseño de la Base de Datos....73
2.5. Implementación del sistema....75
2.5.1. Componente desarrollado....76
2.5.2. Diagrama de Componentes 76
2.6. Conclusiones 77
CAPÍTULO 3: ANÁLISIS DE RESULTADOS.....79
3.1. Métricas Aplicadas....79
3.1.1. Métrica Tamaño de clase (TC)....79
3.1.2. Árbol de Profundidad de Herencia 83
3.1.3. Métrica de la Complejidad Ciclomática. 84
3.2. Pruebas de unidad....89
3.2.1. Pruebas de Caja Negra. 89
3.3. Conclusiones 99
CONCLUSIONES....100
RECOMENDACIONES....101
BIBLIOGRAFÍA....102
ANEXOS 105
Anexo 1: Estructura del Patrón Abstract Factory (Fábrica Abstracta)....105
Anexo 2: Estructura del Patrón Singleton....105
Anexo 3: Estructura del Patrón Facade....105

Anexo 4: Diagrama de secuencia “Manejar Excepción” escenario Reemplazar Excepción. 106
Anexo 5: Diagrama de secuencia “Manejar Excepción” escenario Visualizar Excepción......................107
Anexo 6: Diagrama de secuencia “Gestionar XML Configurador” escenario Cargar XML. 108
Anexo 7: Diagrama de secuencia “Adicionar Contexto”. 109
Anexo 8: Diagrama de secuencia “Gestionar Módulo” escenario adicionar módulo. 110
Anexo 9. Metodo de la clase grtConfigurador GlobalExcepciones. 111
Anexo 10. Metodo de la clase gestora del intérprete AppThreadException. 111
Anexo 11. Metodo de la clase frmPrincipal btnGenerarXML_Click que implementa el evento click del boton btngenerarXML. 111
Anexo 12. Metodo de la clase frmPrincipal cargarXMLConfiguradorToolStripMenuItem_Click que implementa el evento click del menú principal la opción Cargar XML Configurador. 113
Anexo 13. Metodo ejecutar de la clase Cubrir. 113
Anexo 14. Metodo GuardarContexto de la clase gestora del módulo configurador..............................113
GLOSARIO..116
A 116
C 116
D 117
E 117
I...117
P 118
S 118
W 118

ÍNDICE DE FIGURAS

Figura 1 Proceso de Desarrollo...18

Figura 2 RUP en Dos Dimensiones......22
Figura 3 Artefactos y Trabajadores del Flujo de Trabajo Análisis y Diseño......25
Figura 4 Artefactos y Trabajadores del Flujo de Trabajo de implementación. 28
Figura 5 Marco de Trabajo de .Net 32
Figura 6 Arquitectura en Capas. 52
Figura 7 Clase Formulario......56
Figura 8 Formulario Generar XML configurador......56
Figura 9 Clase gestor GtrConfiguradorGlobalExcepciones 57
Figura 10 Clase gestor GtrProcesamientoGlobalExcepcion......57
Figura 11 Clase entidad Contexto......58
Figura 12 Clase fachada FachadaModeloConfiguracion......59
Figura 13 Clase de acceso a datos Conexion. 60
Figura 14 Diagrama de Clases “Manejar Excepción”. 63
Figura 15 Diagrama de Secuencia “Manejar Excepción” escenario Cubrir Excepción. 64
Figura 16 Diagrama de clases “Gestionar XML Configurador”. 65
Figura 17 Diagrama de secuencia “Gestionar XML Configurador” escenario Generar XML......66
Figura 18 Diagrama de clases “Adicionar Contexto”. 67
Figura 19 Diagrama de clases “Gestionar Módulo”. 68
Figura 20 Diagrama de secuencia “Gestionar Módulo” escenario Eliminar Módulo......69
Figura 21 Modelo de datos del Módulo Configurador de Excepciones......70
Figura 22 Diagrama de despliegue del módulo Configurador de Excepciones......71
Figura 23 Diagrama de Componente del intérprete de excepciones. 77
Figura 24 Diagrama de Componente del Módulo Configurador de Excepciones......77
Figura 25 Una jerarquía de Clases. 84

Figura 26 Grafo método Adicionar Módulo. 85
Figura 27 Grafo método AppThreadException. 86
Figura 28 Grafo método btnGenerarXML_Click 86
Figura 29 Grafo método cargarXMLConfiguradorToolStripMenuItem. 87
Figura 30 Grafo método Ejecutar. 87
Figura 31 Grafo método GuardarContexto 88

ÍNDICE DE TABLAS

Tabla 1 Descripción de los Actores del Sistema. 62
Tabla 2 Prefijo de las clases. 72
Tabla 3 Prefijos de las clases de interfaz de Usuario. 73
Tabla 4 Clases de la capa de Negocio Intérprete. 80
Tabla 5 Clases de la capa de Negocio Módulo Configurador. 81
Tabla 6 Resultados métrica Tamaño de Clases. 82
Tabla 7 Complejidad Ciclomática vs Evaluación del Riesgo 89
Tabla 8 Caso de prueba 1 Adicionar Módulo. 90
Tabla 9 Caso de prueba 2 Eliminar módulo. 91
Tabla 10 Caso de prueba 1 Cargar XML. 92
Tabla 11 Caso de prueba 2 Generar XML 93
Tabla 12 Caso de prueba 1 Adicionar contexto con una operación y una acción. 97
Tabla 13 Caso de prueba 2 Adicionar contexto con dos operaciones y dos acciones. 97
Tabla 14 Caso de prueba 1 Excepción controlada que se configuró para que sea cubierta con una nueva. 99

INTRODUCCIÓN

Desde hace algún tiempo la sociedad está siendo testigo de un cambio que ha transformando la forma de entender el mundo. Ninguna de las revoluciones técnicas sucedidas a lo largo de la humanidad; como fueron la imprenta o la electricidad, tiene comparación con los avances tecnológicos que se han implantado en los últimos veinte años. Como parte de estos adelantos se encuentra el desarrollo de la computación y la informática.

En conjunto con el desarrollo de estas tecnologías se creó un nuevo producto que permite interactuar con la computadora y realizar diversas operaciones sobre la misma; como escribir una carta, hacer cálculos matemáticos, entre otras; a estos productos se les llamó software o programas de computación. Al ir evolucionando estas tecnologías también fueron aumentando su complejidad los programas, y por ende se complicó el proceso de creación y desarrollo de los mismos, evolucionando al punto de convertirse en los últimos años en una ingeniería.

Al crear nuevos software para su uso en las nuevas tecnologías, los informáticos se han encontrado con algunos problemas, entre estos están errores que ocurren durante la ejecución de un programa y requiere código fuera del flujo normal de control, a lo que han llamado excepción. En el desarrollo para resolver este problema, las excepciones se han convertido básicamente; entre los diferentes lenguajes de programación, en un objeto que se genera cuando en tiempo de ejecución se produce algún error y que contiene información sobre el mismo.

Tradicionalmente, el sistema que en algunos lenguajes y plataformas se ha venido usando para informar estos errores; consistía simplemente en hacer que los métodos en cuya ejecución pudiesen producirse, devolvieran códigos que informasen si se han ejecutado correctamente, o en caso contrario, sobre cuál fue el error producido.

El uso de excepciones proporcionan las siguientes ventajas frente a este sistema:

- Claridad: El uso de códigos especiales para informar de errores suele dificultar la legibilidad del código fuente, debido a que se mezclan las instrucciones propias de la lógica del mismo con las instrucciones propias del tratamiento de los errores que pudiesen producirse durante su ejecución.

Con el uso de excepciones solamente se lanzan y se tratan cuando son necesarias contribuyendo a la claridad del código.

- Más información: A partir del valor de un código de error puede ser difícil deducir las causas del mismo, y conseguirlo muchas veces implica tener que consultar la documentación proporcionada sobre el método que lo provocó, pudiendo la misma no especificar claramente su causa. Por el contrario, una excepción es un objeto que cuenta con campos que describen las causas del error y a cuyo tipo suele dársele un nombre que resume claramente su causa.

- Tratamiento asegurado: Cuando se utilizan códigos de error nada obliga a tratarlos en cada llamada al método que los pueda producir, e ignorarlos puede provocar más adelante en el código comportamientos inesperados de causas difíciles de descubrir. Cuando se usan excepciones, siempre se asegura que el programador trate toda excepción que pueda producirse o que, si no lo hace, se aborte la ejecución del programa mostrándose un mensaje que indica donde se ha producido el error.

A la gestión de excepciones en los programas muchas veces no se le da la importancia que lleva. En la actualidad, se desarrollan aplicaciones más complejas y más grandes pero sin darle demasiada relevancia al tratamiento de errores. Al crear software nos centramos sobre todo en el desarrollo para el comportamiento normal, olvidándonos de las excepciones. Realmente su uso simplifica enormemente el trabajo, permitiendo escribir por un lado el flujo normal de un método o función y por otro mantener el control de los errores ocurridos en tiempo de ejecución. Existen aplicaciones en las que una excepción producida, no tratada o gestionada provoca grandes daños, ejemplos de estas son los software en que están en juego vidas humanas, o que tienen un gran impacto en la economía de empresas.

El uso excepciones es el mecanismo recomendado en la plataforma .NET para la propagación de errores que se producen durante la ejecución de los programas (divisiones por cero, intentos de lectura de archivos dañados, etc.) ,ya que en esta plataforma desaparecen los problemas de complicar el compilador y dificultar las optimizaciones; como existían en otros lenguajes de programación, porque es el CLR[1] quien se encarga de detectar y tratar las excepciones y es su recolector de basura quien asegura la correcta

[1] CLR (Common Language Runtime) en español Lenguaje común en tiempo de ejecución.

destrucción de los objetos. El código seguirá siendo algo más lento; comparado con uno que no se traten adecuadamente, pero es un pequeño sacrificio que merece la pena hacer, en tanto que ello asegura que nunca se producirán problemas difíciles de detectar derivados de errores ignorados.

Por el aumento de las complejidades del software, se hace necesario que el proceso de tratamiento y manipulación de excepciones se realice de forma eficiente y transparente a la programación del mismo. Debido a que las estructuras de excepciones de las aplicaciones crecen mucho y se hace muy difícil mantener una manipulación estándar; donde están implicados varios programadores a la vez.

A partir de la situación problemática antes expuesta, el presente trabajo de diploma pretende dar respuesta mediante la solución del siguiente ***Problema Científico***:

¿Cómo lograr un eficiente manejo de las excepciones y trazas en un software desarrollado sobre arquitectura .net, de forma que el proceso sea paralelo e independiente al trabajo del programador?

Lo cual lleva al ***Objeto de Estudio*** que es el **Proceso de desarrollo de software**. Y el ***Campo de Acción*** se centra en el **Diseño e Implementación de sistema para la gestión de las excepciones y trazas de software, en el desarrollo de programas con arquitectura .net.**

El ***Objetivo General*** de la investigación es:

Diseñar e implementar un sistema para la gestión de las excepciones y trazas de software, en el desarrollo de aplicaciones con arquitectura .net, para facilitar claridad, más información y tratamiento asegurado de estos factores importantes durante la implementación.

Al cual se le dará cumplimiento mediante la obtención de los siguientes ***Objetivos Específicos***:

- Elaborar el marco teórico de la investigación.
- Realizar los Diagramas de Clases del Diseño.
- Realizar los Diagramas de Secuencia.

- Realizar el Diagrama de Componentes.
- Implementar los componentes.
- Validar el sistema obtenido mediante las técnicas adecuadas.

Hipótesis:

Si se realiza el diseño e implementación de una aplicación para la gestión de las excepciones y trazas de software, en el desarrollo de proyectos con arquitectura .NET entonces se contribuirá al mejoramiento del proceso de implementación en los proyectos.

Metodología de la Investigación

Para realizar la parte investigativa de este trabajo se siguieron los siguientes pasos:

1. Recopilar información necesaria para el proyecto.
    - Análisis de la bibliografía disponible.
    - Consulta de libros con relación al tema.
    - Búsqueda de textos disponibles en Internet.
    - Entrevistas con los especialistas.
2. Seleccionar y organizar datos necesarios para el proyecto.

Método utilizado para recopilar información

Para recopilar la información se utilizaron:

- Análisis Bibliográfico.
- Entrevistas.

Revisión bibliográfica

Se basó fundamentalmente en consultar un conjunto de fuentes de información referidas al tema, como fueron libros, artículos, revistas, publicaciones, boletines y toda una variedad de materiales escritos y digitales, además se localizaron lecturas especializadas sobre el tema para documentar la base teórica de este trabajo. Toda esta fuente de información fue debidamente referenciada para ulteriores consultas, pues la misma servirá a aquellas personas que deseen estudiar más a fondo el resto de los capítulos de este trabajo, aunque es bueno destacar que no representa en manera alguna una bibliografía profunda sobre el tema.

Técnica de Entrevista

Mediante esta forma específica de interacción social, se logró obtener información y detalles del sistema que deseaban obtener finalmente los clientes. A partir de las repuestas y aclaraciones que brindaron los especialistas se obtuvieron las informaciones que se buscaban. Gracias a que esta técnica permite reunir datos primarios, generalmente con un carácter más práctico y menos teórico, se logró definir todos los detalles requeridos para el diseño del sistema.

Método utilizado para seleccionar y organizar los datos

La selección de información de la bibliografía consultada se realizó considerando a aquellas que tuviesen relación directa con el problema en cuestión. Esta se organizó utilizando el método Inductivo – deductivo, partiendo de la idea de comprender los problemas más generalizadores y a partir de estos resultados interpretar conceptos con menor nivel de generalización.

Tareas de la investigación:

- Estudio de los procesos de tratamiento de excepciones en la arquitectura que brinda Visual Studio .NET.
- Estudio de las diferentes teorías de tratamiento de excepciones con C#.NET.

- Estudio del modelado de sistemas informáticos existentes dedicados al tratamiento de excepciones.
- Estudio de la plataforma de desarrollo y las herramientas asociadas a ésta.
- Estudio de las Metodologías de Desarrollo de Software.
- Estudio de los Paradigmas de Programación.
- Estudio y selección de los Patrones de Diseño más factibles para esta propuesta de solución.
- Realización de los principales artefactos del diseño.
- Realización de los principales artefactos de la implementación.
- Validación el sistema mediante las métricas de diseño.
- Validación el sistema mediante casos de prueba.

.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se exponen diferentes conceptos referentes al tratamiento de excepciones en el desarrollo de aplicaciones Además se realiza un estudio sobre las diferentes tendencias y tecnologías actuales en el desarrollo de aplicaciones informáticas.

1.1. Tratamiento de excepciones

1.1.1. ¿Qué es excepción de software?

En el desarrollo de la tecnología de la computación; al ir evolucionando el software, los programadores se han encontrado con algunos errores que ocurren mientras se están ejecutando los programas, a los que diferentes autores han definido de la siguiente manera:

- En la ejecución de un programa pueden darse muchas situaciones inesperadas: dispositivos que fallan, datos que se entran incorrectamente, valores fuera del rango esperado (etc.) Incorporar todas las posibilidades de fallo y casos extraños en la lógica de un algoritmo puede entorpecer su legibilidad y su eficiencia; no tenerlas en cuenta produce que los programas aborten de forma incontrolada y puedan generarse daños en la información manejada. Los mecanismos de control de excepciones permiten contemplar este tipo de problemas separándolos de lo que es el funcionamiento "normal" del algoritmo; una *excepción* en Ada 95 es un objeto que identifica la ocurrencia de una situación anormal. ()

- Una excepción es un evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias. (Pérez)

- Situaciones anómalas que se pueden producir durante la ejecución de un programa, error en tiempo de ejecución. (Martínez)

Las definiciones analizadas convergen a una idea común, sobre la cual se apoyará la presente investigación: Una excepción de software es una interrupción por un evento inesperado en tiempo de ejecución. La forma de cómo tratarla y manejarla varía entre los distintos lenguajes de programación,

durante este capítulo se va a tomar la forma adoptada por la tecnología .NET por la finalidad que tiene el mismo.

1.1.2. Excepciones en C#.NET

Como se ha mencionado hasta el momento, las excepciones son el mecanismo recomendado en la plataforma .NET para propagar los errores que se produzcan durante la ejecución de las aplicaciones (divisiones por cero, lectura de archivos no disponibles, etc.) Básicamente, son objetos derivados de la clase System.Exception que se generan cuando en tiempo de ejecución se produce algún error y que contienen información sobre el mismo.

En .NET todas las excepciones derivan de un tipo predefinido en la BCL[2] llamado System.Exception. Algunos de los métodos que heredan de este son:

- string Message {virtual get;}: Contiene un mensaje descriptivo de las causas de la excepción. Por defecto este mensaje es una cadena vacía (“”)
- string Source {virtual get; virtual set;}: Almacena información sobre cuál fue la aplicación u objeto que causó la excepción.
- MethodBase TargetSite {virtual get;}: Almacena cuál fue el método donde se produjo la excepción en forma de objeto System.Reflection.MethodBase. Puede consultar la documentación del SDK para obtener información sobre las características del método a través del objeto MethodBase.
- string HelpLink {virtual get;}: Contiene una cadena con información sobre cuál es la URI donde se puede encontrar información sobre la excepción. El valor de esta cadena puede establecerse con virtual Exception SetHelpLink (string URI), que devuelve la excepción sobre la que se aplica, pero con la URI ya actualizada.

Entre otros que son de gran ayuda a la hora de recoger y manejar información sobre la excepción capturada. Para crear excepciones se debe heredar de algunos de los miembros

[2] BLC : Biblioteca de la Clase Base

pertenecientes a la jerarquía de excepciones del sistema; teniendo en cuenta que la mayor jerarquía o el padre de todo el árbol es System.Exception.

En la práctica, cuando se crean nuevos tipos derivados de System.Exception no se suele redefinir sus miembros ni añadirles nuevos, sino que sólo se hace la derivación para distinguir una excepciones de otra por el nombre del tipo al que pertenecen. Pero, es conveniente respetar el convenio de darles un nombre acabado en Exception y redefinir los tres constructores:

1. Exception()
2. Exception(string msg)
3. Exception(string msg, Exception causante)

El primer constructor crea una excepción cuyo valor para Message será “” y no causada por ninguna otra excepción (InnerException valdrá null). El segundo la crea con el valor indicado para Message y el último la crea con además la excepción causante indicada.

En el espacio de nombres System de la BCL hay predefinidas múltiples excepciones derivadas de System.Exception que se corresponden con los errores más comunes que pueden surgir durante la ejecución de una aplicación. En la ayuda MSDN del Visual Studio se puede consultar información sobre estas.

Para informar de un error no basta con crear un objeto del tipo de excepción apropiado, sino que también hay que pasárselo al mecanismo de propagación de excepciones del CLR. A esto se le llama lanzar la excepción, y para hacerlo se usa la instrucción:

throw <objetoExepcionALanzar>

Al igual que en varios de los lenguajes predecesores, esta instrucción rompe con la ejecución del código y da comienzo al mecanismo de propagación de excepciones.

Una vez lanzada una excepción es posible escribir código que se encargue de tratarla. Por defecto, si este código no se escribe la excepción provoca que la aplicación aborte mostrando un mensaje de error en el que se describe la excepción producida y dónde se ha producido.

La estructura que permite la captura y el tratamiento que se desea es la siguiente:

```
Try
{ <instrucciones>}
Catch(<exepcion1>)
{ <instrucciones>}
Catch(<exepcion2>)
{ <instrucciones>}
finally
{ <instrucciones>}
```

Esto significa que si durante la ejecución de las <instrucciones>; que se encuentran entre las llaves del *try*, se lanza una excepción de tipo <excepción1> (o alguna subclase suya) se ejecutan las instrucciones <tratamiento1>, si fuese de tipo <excepción2> se ejecutaría <tratamiento2>, y así hasta que se encuentre una cláusula *catch* que pueda tratar la excepción producida, si no se encuentra ninguna y la instrucción *try* está anidada dentro de otra, se examina en los *catch* de su *try* padre y se repite el proceso. Si al final se recorren todos los *try* padres y no se encuentra ningún catch compatible, entonces se busca en el código desde el que se llamó al método que produjo la excepción. Si se termina llegando al método que inició el hilo donde se produjo la excepción y tampoco allí se encuentra un tratamiento apropiado, se aborta dicho hilo; y si ese hilo es el principal (el que contiene el punto de entrada) se aborta el programa y se muestra el mensaje de error con información sobre la excepción lanzada.

La idea de todo este mecanismo de excepciones es evitar mezclar el código normal con el código de tratamiento de errores. Así, en <instrucciones> se escribiría el código como si no se pudiesen producir errores, en las cláusulas *catch* se tratarían las posibles excepciones, y en la cláusula *finally* se incluiría código a ejecutar tanto si produjesen errores como si no (suele usarse para liberar recursos ocupados, como fichero o conexiones de red abiertas). Además, cuando se relance una excepción en un *try* con cláusula *finally*, antes de pasar a reprocesar la excepción en el *try* padre del que la relanzó se ejecutará dicha cláusula. (Martin Torres, y otros, 2002)

El Visual Studio .net tiene un elemento importante para la detección y captura de excepciones que es el evento ThreadException. Este evento permite que la aplicación de formularios Windows Forms controlen excepciones que se producen en los subprocesos de formularios Windows Forms y que, de otro modo, no estarían controladas. Se le asocian los controladores de eventos al evento ThreadException para tratar estas excepciones, ya que dejarán la aplicación en un estado desconocido. (Microsoft, 2008)

La plataforma .NET brinda una buena cantidad de recursos para el tratamiento de excepciones. En este epígrafe se quiso brindar una introducción al sistema de excepciones con el que van a trabajar.

1.1.3. Análisis de Sistemas Existentes

Actualmente existen Soluciones Informáticas, con el objetivo de garantizar un sistema de control y manipulación de excepciones en el proceso de desarrollo de software. Es importante estudiar y tener en cuenta también, las formas en que diferentes proyectos realizan el proceso de manipulación y tratamiento de errores.

A continuación se hace referencia a las Soluciones Informáticas estudiadas para el desarrollo de este sistema, las cuales permitieron enfocar el software a desarrollar directamente al proceso de tratamiento automático de excepciones.

- **Microsoft Exception Management Application Block para .NET**

Es un simple y extensible Framework para guardar información de excepciones. Permitiendo anotar los detalles de la excepción a otras fuentes de datos o notificar a operadores sin afectar el código de la aplicación. Al usar el Framework se puede reducir la cantidad de código escrito usualmente para el

tratamiento de errores. (Microsoft Corporation, 2003) Sin embargo al usar la solución se presentan problemas:

- El momento de identificar las excepciones y dar tratamiento a las mismas; no es claro y sencillo para el programador provocando que tenga que dedicarle tiempo a la captura y configuración de una excepción.
- El Framework no brinda una herramienta sencilla para la configuración de las excepciones a manejar.
- No abstrae al programador del uso de código para tratamiento de excepciones durante el desarrollo.

Este Framework a tenido actualizaciones como son la liberación de la Enterprise Library 2005 y la Enterprise Library 2007 en las cuales se ha corregido algunos problemas de complejidad de uso del Framework pero se han mantenidos lo problemas mencionado.

- **Proceso de tratamiento de excepciones en soluciones de software sobre .NET en la universidad.**

Se contacto con algunos proyectos de desarrollo en la universidad desarrollados sobre arquitectura .NET para investigar como manejaban sus excepciones. Llegando a la conclusión que en general se usaba la estructura de código recomendada para el tratamiento de errores en .NET; o sea los bloques código mencionados en el epígrafe anterior. Ejemplo del uso de esta estructura es el proyecto ONE primera etapa el cual fue desarrollado sobre la plataforma Visual Studio .NET 2005. Se encontró también que en el proyecto Registro y Notarias en subsistema de Administración Financiera desarrollaron un intérprete de excepciones que resolvió el problema de independizar la manipulación y tratamiento de errores. Pero es un componente muy integrado a la arquitectura del software que ellos usan y no era aplicable a cualquier solución en .NET.

1.2. Paradigmas de Programación

1.2.1. Programación Orientada a Objetos

La Programación Orientada a Objetos es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Es una forma especial de programar, más cercana a como se expresaría las cosas en la vida real que otros tipos de programación.

La unidad básica de este paradigma no es la función (unidad básica de la programación estructurada), sino un ente denominado objeto. Un objeto es la representación de un concepto para un programa, y contiene toda la información necesaria para abstraer dicho concepto: los datos que describen su estado y las operaciones que pueden modificar dicho estado, y determinan las capacidades del objeto.

Existen una serie de principios fundamentales para comprender cómo se modela la realidad al crear un programa bajo el paradigma de la orientación a objetos. Estos principios son: de abstracción, de encapsulamiento, de modularidad, de jerarquía, del paso de mensajes, de polimorfismo, de la herencia.

- **Principio de Abstracción**

Mediante la abstracción, la mente humana modela la realidad en forma de objetos. Para ello busca semejanzas entre la realidad y la posible implementación de objetos del programa que simulen el funcionamiento de los objetos reales. Significa extraer las propiedades esenciales de un objeto que lo distinguen de los demás tipos de objetos y proporciona fronteras conceptuales definidas respecto al punto de vista del observador. Es la capacidad para encapsular y aislar la información de diseño y ejecución.

- **Principio de Encapsulamiento**

El encapsulamiento permite a los objetos elegir qué información es publicada y qué información es ocultada al resto de los objetos. Para ello los objetos suelen presentar sus métodos como interfaces públicas y sus atributos como datos privados e inaccesibles desde otros objetos.

Para permitir que otros objetos consulten o modifiquen los atributos de los objetos, las clases suelen presentar métodos de acceso. De esta manera el acceso a los datos de los objetos es controlado por el programador, evitando efectos laterales no deseados.

Con el encapsulado de los datos se consigue que las personas que utilicen un objeto sólo tengan que comprender su interfaz, olvidándose de cómo está implementada, y en definitiva, reduciendo la complejidad de utilización.

- **Principio de Modularidad**

Mediante la modularidad, se propone al programador dividir su aplicación en varios módulos diferentes (ya sea en forma de clases, paquetes o bibliotecas), cada uno de ellos con un sentido propio.

Esta fragmentación disminuye el grado de dificultad del problema al que da respuesta el programa, pues se afronta el problema como un conjunto de problemas de menor dificultad, además de facilitar la comprensión del programa.

- **Principio de Jerarquía**

La mayoría de las personas ve de manera natural el mundo, como objetos que se relacionan entre sí de una manera jerárquica. Por ejemplo, un perro es un mamífero, y los mamíferos son animales, y los animales seres vivos. Del mismo modo, las distintas clases de un programa se organizan mediante la jerarquía. La representación de dicha organización, da lugar a los denominados árboles de herencia.

- **Principio del Paso de Mensajes**

Mediante el denominado paso de mensajes, un objeto puede solicitar de otro objeto que realice una acción determinada o que modifique su estado. El paso de mensajes se suele implementar como llamadas a los métodos de otros objetos. Desde el punto de vista de la programación estructurada, esto correspondería con la llamada a funciones.

- **Principio de Polimorfismo**

Polimorfismo quiere decir "un objeto y muchas formas". Esta propiedad permite que un objeto presente diferentes comportamientos en función del contexto en que se encuentre. Por ejemplo un método puede presentar diferentes implementaciones, en función de los argumentos de entrada, recibir diferentes

números de parámetros para realizar una misma operación, y realizar diferentes acciones dependiendo del nivel de abstracción en que sea llamado.

- **Principio de la Herencia**

Es la propiedad que permite a los objetos construirse a partir de otros objetos. La clase base contiene todas las características comunes. Las sub-clases contienen las características de la clase base más las características particulares de la sub-clase. Si la sub-clase hereda características de una clase base, se trata de herencia simple, si hereda de dos o más clases base, herencia múltiple.

((Universidad de Burgos, 1999) (Teso)

1.2.2. Programación Orientada a Eventos

Los lenguajes visuales extienden las posibilidades de los lenguajes convencionales incorporando elementos nuevos, que poseen un comportamiento predefinido, orientados a facilitar el diseño de la interfaz de la aplicación. Estos elementos, también llamados componentes[3], pueden modificarse tanto en la apariencia como en la respuesta esperada al interactuar con ellos.

El uso de herramientas que combinen lo visual con lo algorítmico llevó a cambiar el estilo de Programación Estructurada convencional hacia lo que se conoce como Programación Orientada a Eventos[4], la cual gira en torno al momento en que las señales del mundo real ocurren (Lanzarini). Este tipo de programación es un poco más complicada que la Secuencial pero con los lenguajes visuales de hoy, se hace sencilla y agradable (Orellana, 2006). En la Programación Secuencial es el programador quien define cuál va a ser el flujo del programa mientras que en la Dirigida por Eventos es el propio usuario quien dirige este flujo.

Dentro de este nuevo paradigma, el programador podrá incorporar un botón a su aplicación, darle la apariencia deseada e indicar, por ejemplo, qué hacer al dar clic sobre él. También puede centrar más su

---

[3] Un componente es un objeto particular que puede ser reutilizado en diferentes contextos.

[4] Un evento es un hecho que se produce en un momento dado bajo ciertas condiciones y puede desencadenar reacciones. Constituye las acciones que pueda ejecutar un usuario sobre el programa que está utilizando en ese momento.

atención al análisis y diseño de la solución, así como a la selección de las estructuras de datos más adecuadas para el problema. Es importante destacar, que el aspecto visual es manejado ahora por el lenguaje de programación. Esto último permite reducir el tiempo de desarrollo, ya que no sólo resuelve el problema de la interfaz con el usuario, sino que facilita el mantenimiento del software y reduce errores (Lanzarini).

Los Programas Orientados a Eventos son los programas típicos de Windows, tales como Word, Excel, editores de imágenes como Adobe Photoshop, IDE[5] como Visual Studio 2005 y otras aplicaciones. Cuando uno de estos programas ha arrancado, lo único que hace es quedarse a la espera de las acciones del usuario, que en este caso son llamadas eventos. El usuario dice si quiere abrir y modificar un fichero existente, o bien comenzar a crear un fichero desde el principio. Estos programas pasan la mayor parte de su tiempo esperando las acciones del usuario y respondiendo a ellas (Orellana, 2006).

Algunos ejemplos de eventos son: el clic hecho sobre un botón, el ingreso de un valor por parte del usuario, la selección de una opción dentro de un menú desplegable, el clic hecho sobre un archivo para abrirlo, arrastrar un ícono, pulsar una tecla o combinación de ellas o simplemente mover el mouse.

1.2.3. Programación Orientada a Componentes

La Programación Orientada a Componentes, es una variante natural de la Orientada a Objetos con el objetivo de construir un mercado global de componentes cuyos usuarios son los propios desarrolladores de aplicaciones que necesitan reutilizar componentes ya hechos y probados para construir sus aplicaciones de una forma más rápida (Navarro, y otros).

Denominamos Componente a la unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio (Navarro, y otros).

[5] IDE (Interfaz de desarrollo): Son aquellas aplicaciones informáticas que están especializadas en uno o varios lenguajes de programación que brinda varios servicios y facilidades entorno a estos optimizando el uso de los mismos.

Existen tres tecnologías de componentes predominantes en la actualidad e incompatibles entre sí (Universidad de Jaén, 2008):

- VBX/OCX/ActiveX de Microsoft: Los VBX surgieron como una forma de poder distribuir controles entre los desarrolladores de Visual Basic. Visual Basic puede ser considerado como el primer Entorno de Desarrollo Orientado a Componentes que tuvo realmente éxito y aceptación. Los OCX y ActiveX son evoluciones posteriores de los VBX.

- VCL de Borland: Es la tecnología de componentes utilizados por los entornos de desarrollo Delphi y C++ Builder. Curiosamente estos dos entornos también permiten la utilización de componentes OCX y ActiveX.

- JavaBeans de Sun: Es la tecnología de componentes basada en Java. En principio no permite la utilización de otro lenguaje de programación. Al contrario que las otras tecnologías, JavaBeans funciona en cualquier plataforma.

Características de los componentes (Universidad de Jaén, 2008) (Navarro, y otros)

- Constituyen una unidad software compilada reutilizable, con una interfaz bien definida.

- Se distribuyen en un único paquete instalable, que contiene en sí todo lo necesario para su funcionamiento, con ninguna o muy pocas dependencias con otros componentes o librerías.

- Pueden estar implementados en cualquier lenguaje de programación (aunque los Orientados a Objetos son los más adecuados), y ser utilizados también para el desarrollo en cualquier lenguaje de programación. Se ejecutan en diferentes plataformas y sistemas operativos.

- Pueden constituir un producto comercial de calidad, realizado por un fabricante especializado. Además pueden existir componentes gratuitos.

- Pueden ser modificables y sujetos a evolución por ampliación, desaparición, sustitución de componentes o reconfiguración de las relaciones entre ellos.

1.3. Proceso de Desarrollo de Software

Cada año se producen en el mercado computadoras más potentes y eficaces, con grandes capacidades de almacenamiento y velocidad. Debido a esto, la tendencia actual en el desarrollo de software hace que aparezcan sistemas más grandes y más complejos y que por lo tanto, los usuarios esperen más de ellos. Esta tendencia también se ve afectada por el auge que ha tenido Internet en los últimos años en el intercambio de información, como fotos, textos, multimedia. Los usuarios quieren ver productos mejorados, adaptados a sus necesidades y que sean cada vez más rápidos. La comunidad de desarrolladores necesita una forma coordinada de trabajar, una forma de que otros entiendan su trabajo sin haber participado directamente en él, necesita un proceso que integre las múltiples facetas de desarrollo, necesita un método común. (Jacobson, y otros, 2002).

Un Proceso de Desarrollo de Software es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. (Jacobson, y otros, 2002) (Véase Figura 1 Proceso de Desarrollo).Un proceso define *quién* está haciendo *qué, cuándo* y *cómo* alcanzar un determinado objetivo que en ingeniería de software sería construir un producto software o mejorar uno existente. Es necesario que este sirva como guía común para todos los participantes (clientes, usuarios, desarrolladores y directores ejecutivos), que sea nuevo, que sea el mejor.

Un proceso que es efectivo proporciona normas para el desarrollo eficiente de un producto de calidad además de capturar y presentar las mejores prácticas que el estado actual de las tecnologías permite, también reduce el riesgo y hace al proyecto más predecible. Como consecuencia de esto se crea un método común. (Jacobson, y otros, 2004)

Figura 1 Proceso de Desarrollo

1.4. Metodología de Desarrollo de Software

Al inicio el desarrollo de software era artesanal en su totalidad. La ausencia de procesos formales, lineamientos claros, determinaron que se importara la concepción y fundamentos de metodologías existentes en otras áreas, y adaptarlas al desarrollo de software. Esta nueva etapa de adaptación contenía el desarrollo dividido en etapas de manera secuencial, que de algo mejoraba la necesidad latente en el campo del software.

Un proceso de software detallado y completo suele denominarse “Metodología”. Las metodologías imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente. Lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar inspirado por otras disciplinas de la ingeniería.

En un proyecto de desarrollo de software la metodología define *Quién debe hacer Qué*, *Cuándo y Cómo debe hacerlo*. Una metodología es un proceso. No existe una metodología de software universal.

Las características de cada proyecto (equipo de desarrollo, recursos) exigen que el proceso sea configurable. Existen dos grupos en los cuáles se dividen estas metodologías: los métodos tradicionales y los procesos ágiles con marcadas diferencias. En la actualidad existen varias metodologías OO (Orientada a Objeto) basadas en UML entre las que se encuentran RUP[6] y XP[7]. En los próximos epígrafes una breve descripción de estas metodologías de desarrollo de software, en las que se apoyara el proceso de desarrollo de la solución propuesta. Se escogen estas metodologías porque son de las más utilizadas, con mejores resultados y las más conocidas en la universidad.

1.4.1. Programación Extrema (XP)

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre

[6] RUP (Rational Unified Process) Proceso unificado de desarrollo en español.

[7] XP (Extreme Programing) Programación extrema en español.

el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos:

- El cliente define el valor de negocio a implementar.
- El programador estima el esfuerzo necesario para su implementación.
- El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
- El programador construye ese valor de negocio.
- Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración. (H.Canós, y otros, 2003)

Características de la metodología de desarrollo XP:

- **Pruebas Unitarias:** Pruebas realizadas a los principales procesos, de forma tal que se pruebe la aplicación teniendo en cuenta posibles errores.

- **Re-fabricación:** Reutilización de código, a través de la creación de patrones o modelos estándares, siendo así más flexible al cambio.

- **Programación en pares:** Se trata de dos programadores escribiendo código en una sola estación de trabajo donde cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. (Mendoza, 2004)

1.4.2. Rational Unified Process (RUP)

Las metodologías tradicionales son referenciadas por diferentes nombres, por ejemplo, pesadas, ortodoxas, clásicas, predictivas, formales, la metodología tradicional más significativa y empleada en la actualidad en los procesos de desarrollo de software es RUP.

Se le llama de esta forma por la extensión en tiempo de desarrollo, así como el gran número de especialistas necesarios para la confección del software. Los negocios de gran magnitud encajan perfectamente en esta metodología puesto que son estos los que involucran a gran cantidad de especialistas de la materia de que se trate.

Esta metodología es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo, a través del UML, y trabajo de muchas metodologías utilizadas por los clientes. La versión que se ha estandarizado vio la luz en 1998 y se conoció en sus inicios como Proceso Unificado de Rational 5.0; de ahí las siglas con las que se identifica a este proceso de desarrollo.

Como RUP es un proceso, en su modelación define como sus principales elementos:

- **Trabajadores (“quién”):** Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.

- **Actividades (“cómo”):** Es una tarea que tiene un propósito claro, es realizada por un trabajador y manipula elementos.

- **Artefactos (”qué”):** Productos tangibles del proyecto que son producidos, modificados y usados por las actividades. Pueden ser modelos, elementos dentro del modelo, código fuente y ejecutables.

- **Flujo de actividades (“Cuándo”):** Secuencia de actividades realizadas por trabajadores y que produce un resultado de valor observable.

En RUP se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo. Ver Figura 2 RUP en Dos Dimensiones..

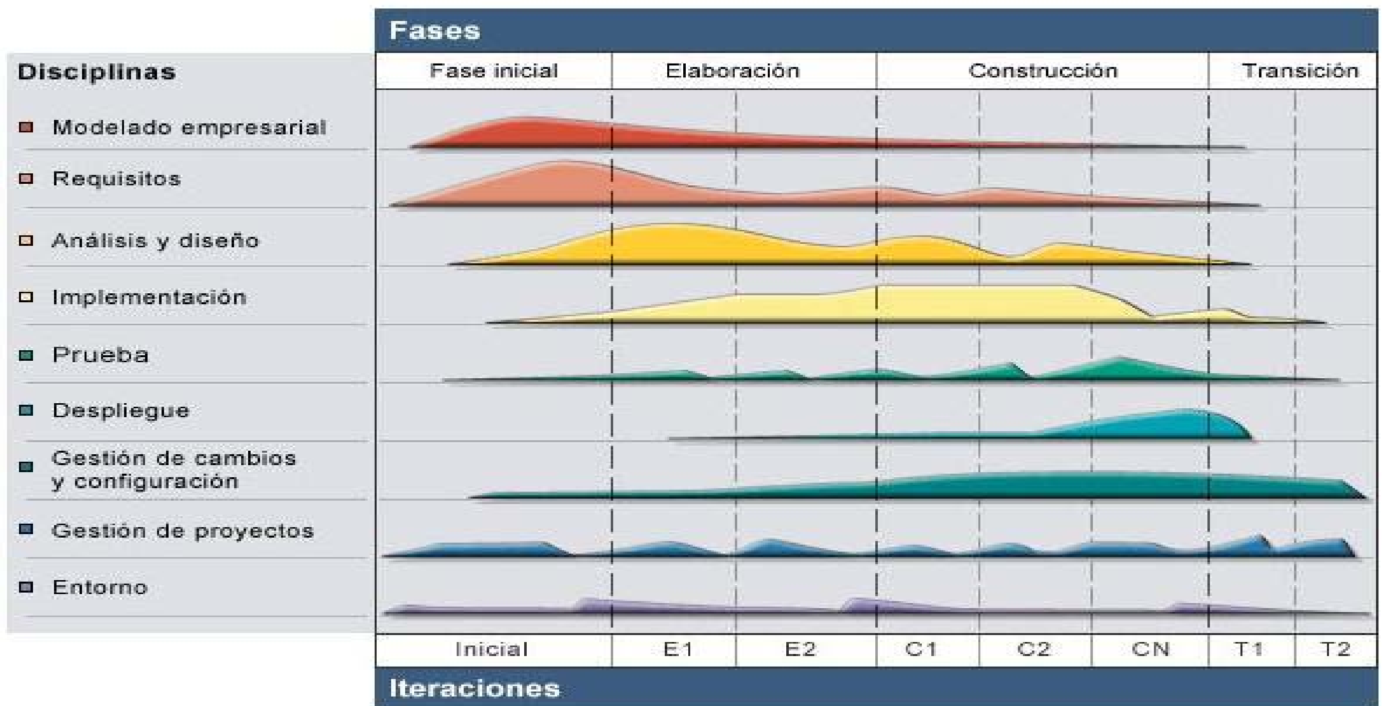


Figura 2 RUP en Dos Dimensiones.

El ciclo de vida de RUP se caracteriza por:

- **Dirigido por casos de uso:** Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).

- **Centrado en la arquitectura:** La arquitectura muestra la visión común del sistema completo con la que el equipo del proyecto y los usuarios deben estar de acuerdo, ya que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios

como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los casos de uso relevantes desde el punto de vista de la arquitectura. El modelo de arquitectura se representa a través de vistas en las que se incluyen los diagramas de UML.

- **Iterativo e Incremental:** RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros.

1.4.3. Breve análisis de la metodología seguir.

Por las características que tiene la solución que se propone, entre las que están:

- Es un software de pequeña envergadura.
- Es implementada por un desarrollador.
- La solución está destinada a ser una herramienta a utilizar por desarrolladores de software.
- Por ser parte de un trabajo de diploma es necesario ilustrar las acciones realizadas en el proceso de diseño e implementación del mismo.

No se va a utilizar una metodología específica, sino que se van a tomar los elementos de las antes descritas que más aportan al desarrollo del presente software y a la ilustración de los artefactos diseñados e implementados. En este sentido se incorporan los elementos necesarios para la construcción de software que brinda XP; tal como la interacción entre el desarrollador con los clientes finales. Así como también se tomaran los principales artefactos generados por los flujos de diseño e implementación que define RUP para una mejor comprensión de la solución.

No se escoge alguna de las dos metodologías para el desarrollo de la solución porque las características del sistema y su proceso de desarrollo no se enmarcan en el proceso que define alguna de las ellas. Las dos metodologías en su concepto son adaptables a las características que se desean aprovechar los desarrolladores, pero hay que seguir un grupo de reglas mínimas para aplicarlas, que no se adaptan en ninguno de los dos casos a lo que el autor requiere. Por eso se decidió aprovechar los

elementos que le brindaban alguna ventaja; para el desarrollo del sistema, que tienen ambas metodologías.

1.5. Artefactos de la Disciplina de Diseño

Como se había analizado anteriormente el presente trabajo se desarrollará e ilustrará utilizando los artefactos que propone la metodología RUP en los flujos de trabajo de Análisis y Diseño e Implementación. Por lo que a continuación se hace un breve estudio sobre los principales artefactos que se generan en estos flujos de trabajos.(Rational Software Corporation, 2003)

La disciplina de Diseño, perteneciente al Flujo de Trabajo de Análisis y Diseño, tiene como propósitos transformar los requerimientos en un diseño del sistema a crear; definir y desarrollar una arquitectura robusta para el sistema, crear una entrada apropiada y un punto de partida para actividades de implementación.

La relación que tiene con el Flujo de Trabajo de Requerimientos es que éste constituye su entrada primaria y con el de Implementación es que éste implementa precisamente ese diseño que se conciba.

A continuación, en la Figura 3 se muestran los Artefactos y Trabajadores del Flujo de Trabajo Análisis y Diseño, siendo los más importantes en el Diseño los que se mencionan en el epígrafe siguiente.

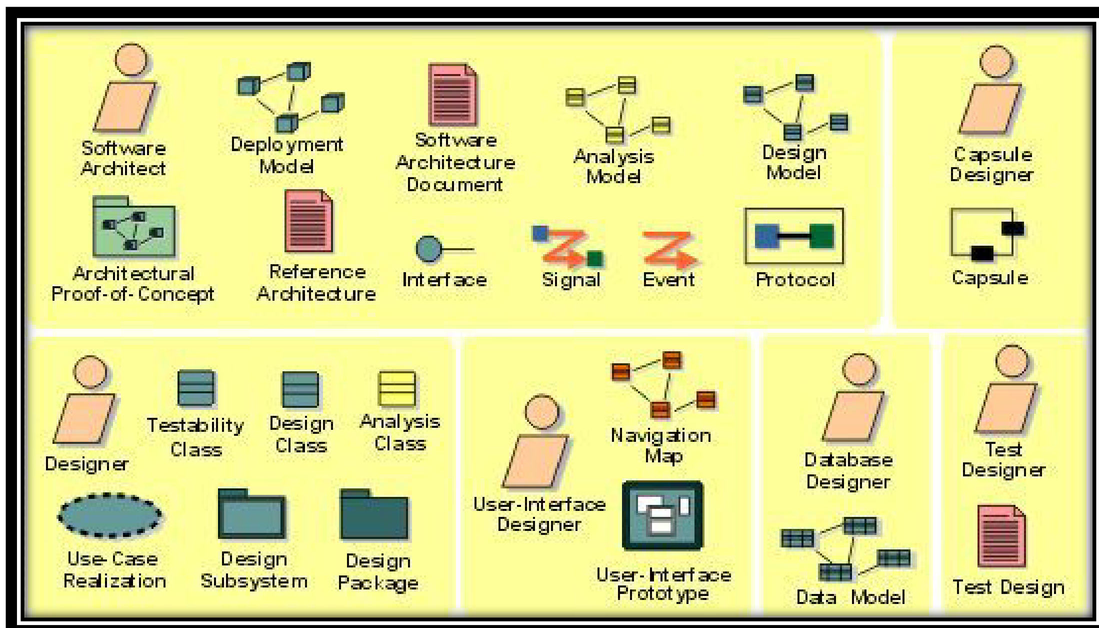


Figura 3 Artefactos y Trabajadores del Flujo de Trabajo Análisis y Diseño

1.5.1. Principales Artefactos

- **Modelo de Diseño**

El Modelo de Diseño es un modelo de objetos que describe la realización de casos de uso y sirve como abstracción del Modelo de Implementación y del código fuente. Constituye una entrada esencial a las actividades de Implementación y Prueba.

Ante todo, establece la arquitectura, pero también es usado como un vehículo de análisis durante la fase de Elaboración, refinándose mediante un diseño detallado durante la fase de Construcción. El Arquitecto de Software es el responsable de la integridad de este modelo, pero no de los paquetes, clases, relaciones, diagramas y realizaciones de casos de uso.

- **Documento de Arquitectura de Software**

El Documento de Arquitectura de Software provee una vista comprensiva de la arquitectura del sistema, utilizando 4+1 vistas arquitectónicas para representar diferentes aspectos de éste. Las vistas son

las siguientes*: Vista de Casos de Uso, Vista Lógica, Vista de Procesos, Vista de Implementación y Vista de Despliegue*; pero todas están regidas por la de *Vista de Casos de Uso*.

Es desarrollado fundamentalmente durante la fase de Elaboración, ya que uno de los objetivos de ésta es establecer una arquitectura sólida. El Arquitecto de Software es el responsable de este documento, además, de mantenerlo actualizado.

- **Modelo de Despliegue**

El Modelo de Despliegue muestra la configuración de los nodos procesadores en tiempo de ejecución, los enlaces de comunicación entre ellos y los componentes y objetos que residen en ellos. Consta de uno o varios *nodos* (elementos de procesamiento con al menos un procesador, memoria y posiblemente otros dispositivos); *dispositivos* (nodos estereotipados sin capacidad de procesamiento) y *conectores* (expresan las conexiones entre los nodos y entre éstos y los dispositivos). El Arquitecto de Software es el responsable de este modelo.

- **Clases del Diseño**

Las Clases del Diseño son una descripción de un conjunto de objetos que comparten las mismas responsabilidades, relaciones, operaciones, atributos, y semántica. Forman parte del Modelo de Diseño y fundamentales en un enfoque orientado a objetos. Las Clases del Diseño Arquitectónicamente significativas son identificadas y descritas durante la fase de Elaboración, las restantes, durante la de Construcción. El Diseñador es el responsable de la integridad de la clase.

- **Realizaciones de Casos de Uso**

Una Realización de Casos de Uso describe cómo un caso de uso particular es realizado dentro del Modelo de Diseño, en términos de colaboración de objetos. Forman parte del Modelo de Diseño y son creadas en la fase de Elaboración para los casos de uso arquitectónicamente más significativos, los restantes, son realizados en la de Construcción. Presentan una descripción de flujos de eventos textual, diagramas de clases y diagramas de interacción, etc. El Diseñador es el responsable de estas realizaciones.

- **Subsistemas de Diseño**

Un Subsistema de Diseño es una parte de un sistema que encapsula comportamiento, expone un conjunto de interfaces, paquetes y otros elementos del modelo. Desde afuera, un subsistema es un elemento sencillo del Modelo de Diseño que colabora con otros elementos del modelo para realizar sus responsabilidades. Las interfaces visibles externamente y su comportamiento se conoce como subsistema de especificación. En el interior, un subsistema es una colección de elementos del modelo (clases del diseño y otros subsistemas) que realizan las interfaces y el comportamiento del subsistema de especificación. Esto se conoce como subsistema de realización. Forma parte del Modelo de Diseño y es precisamente el Diseñador el responsable de su integridad, no el Arquitecto.

- **Paquetes de Diseño**

Un Paquete de Diseño es una colección de clases, relaciones, realizaciones de casos de usos, diagramas y otros paquetes. Es usado para estructurar el Modelo de Diseño dividiéndolo en partes más pequeñas. Deben usarse fundamentalmente como herramienta organizacional para agrupar elementos relacionados, si se requiere un comportamiento o semántica se deben usar subsistemas. Forma parte del Modelo de Diseño y el Diseñador es el responsable de su integridad, pero dentro de él, sólo de las Clases del Diseño.

- **Modelo de Datos**

El Modelo de Datos describe las representaciones lógicas y físicas de los datos persistentes usados por la aplicación. En casos donde ésta utilice un Sistema de Gestión de Bases de Datos Relacional (SGBD), el Modelo de Datos puede incluir elementos como procedimientos almacenados, triggers, restricciones (constraints), etc., que definen la interacción de los componentes de la aplicación con el SGBD. El Diseñador de Bases de Datos es el responsable de la integridad del modelo, asegurando que sea completamente correcto, entendible y consistente.

- **Diagramas de Interacción**

Los Diagramas de Interacción se utilizan para modelar los aspectos dinámicos de un sistema. Muestran una interacción que consiste en un conjunto de objetos y sus relaciones, incluyendo los

mensajes que se pueden enviar entre ellos. Existen dos tipos: Los Diagramas de Secuencia y los de Colaboración. El primero destaca la ordenación temporal de los mensajes y el segundo destaca la organización estructural de los objetos que envían y reciben mensajes.

1.6. Artefactos de la Disciplina de Implementación

(Rational Software Corporation, 2003)

El Flujo de Trabajo de Implementación tiene como propósitos definir la organización del código en términos de subsistemas de implementación organizados en capas; implementar elementos de diseño en términos de elementos de implementación tales como ficheros fuente, binarios, ejecutables, etc.; probar los componentes desarrollados como unidades; integrar los resultados producidos por los implementadores individuales (o los equipos) en un sistema ejecutable.

La disciplina de Diseño describe cómo desarrollar un Modelo de Diseño el cual representa la intención de la implementación y es la entrada primaria al Flujo de Trabajo de Implementación.

A continuación en la Figura 4 se muestran los Artefactos y los Trabajadores del Flujo de Trabajo de Implementación.

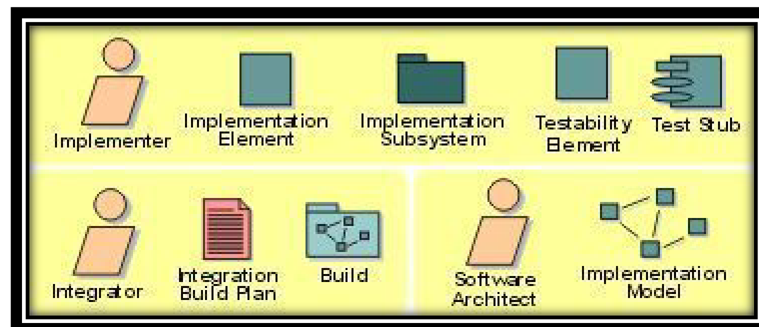


Figura 4 Artefactos y Trabajadores del Flujo de Trabajo de implementación.

1.6.1. Principales Artefactos

- **Modelo de Implementación**

El Modelo de Implementación representa la composición física de la implementación en términos de Subsistemas de Implementación y Elementos de Implementación tales como directorios y ficheros incluyendo código fuente, datos y ejecutables. El Arquitecto de Software es el responsable de la integridad de este modelo.

- **Subsistema de Implementación**

Los Subsistemas de Implementación son una serie de Elementos de Implementación. Estructuran el Modelo de Implementación al dividirlo en pequeñas partes que puedan ser integradas y probadas individualmente. Son análogos a los Paquetes de Diseño y pertenecen a la Vista de Implementación. El Implementador es el responsable del subsistema.

- **Elementos de Implementación**

Constituyen la parte física que compone una implementación incluyendo archivos y directorios. Contienen archivos de código (fuente, binarios o ejecutables), archivos de datos y de documentación como por ejemplo, archivos de ayuda en línea. El Implementador es el responsable de los elementos.

- **Build**

Un Build es una versión operativa de un sistema o de una parte de un sistema que demuestra un subconjunto de las capacidades que se proveerán en el producto final. Consta de uno o varios Elementos de Implementación (a menudo ejecutables), cada uno construido a partir de otros elementos, por lo general por un proceso de compilación y vinculación de código fuente. El Integrador es el responsable de la producción de los Builds.

- **Plan de Integración del Build**

El Plan de Integración del Build provee un plan detallado para la integración dentro de una iteración. Define el orden en el cual los componentes deben ser implementados, cuáles Builds crear cuando se integra el sistema y cómo estos van a ser evaluados. El Integrador es el responsable de este plan y además, debe mantenerlo actualizado.

- **Diagrama de Componentes**

Muestran la estructura de los componentes[8], incluyendo clasificadores que los especifican y artefactos que los implementan. También son usados para mostrar la estructura de alto nivel del Modelo de Implementación en términos de Subsistemas de Implementación y las relaciones entre los Elementos de Implementación.

1.7. Plataforma de Desarrollo Microsoft.Net

.NET es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma y que permita un rápido desarrollo de aplicaciones. Basado en esta plataforma, Microsoft intenta desarrollar una estrategia horizontal que integre todos sus productos, desde el Sistema Operativo hasta las herramientas de mercado. (Cuevas Guerrero)

Ésta plataforma desembocó en el panorama empresarial en el año 2001 y ofrece a los desarrolladores algunas ventajas interesantes, entre las que se pueden mencionar el soporte para múltiples lenguajes, una perfecta integración con el resto de los productos de Microsoft, dispone del Visual Studio .Net como una herramienta muy potente que ofrece un entorno homogéneo de desarrollo, los desarrolladores poco experimentados pueden utilizar lenguajes como Visual Basic .NET que hacen muy sencilla la creación de aplicaciones empresariales. (Ávila, 2007)

Con esta plataforma, Microsoft incursiona de lleno en el campo de los Servicios Web y establece el XML como norma en el transporte de información en sus productos y lo promociona como tal en los sistemas desarrollados utilizando sus herramientas. .NET intenta ofrecer una manera rápida y económica pero a la vez segura y robusta de desarrollar aplicaciones o como la misma plataforma las denomina, “soluciones”, permitiendo a su vez una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

---

[8] Los componente de Software son todo aquel recurso desarrollado para un fin concreto y que puede formar solo o junto con otro/s, un entorno funcional requerido por cualquier proceso predefinido. Son independientes entre ellos, y tienen su propia estructura e implementación. Parte modular de un sistema, desplegable y reemplazable. Típicamente contiene clases y puede ser implementado por uno o más artefactos.

A largo plazo Microsoft pretende reemplazar la API Win32 o Windows API con la plataforma .NET. Esto es debido a que la API Win32 o Windows API fue desarrollada sobre la marcha, careciendo de documentación detallada, uniformidad y cohesión entre sus distintos componentes, provocando múltiples problemas en el desarrollo de aplicaciones para el sistema operativo Windows. La plataforma .NET pretende solventar la mayoría de estos problemas proporcionando un conjunto único y expandible con facilidad, de bloques interconectados, diseñados de forma uniforme y bien documentados, que permitan a los desarrolladores tener a mano todo lo que necesitan para producir aplicaciones sólidas.

Debido a las ventajas que la disponibilidad de una plataforma de este tipo puede darle a las empresas de tecnología y al público en general, muchas otras empresas e instituciones se han unido a Microsoft en el desarrollo y fortalecimiento de la plataforma .NET, ya sea por medio de la implementación de la plataforma para otros sistemas operativos aparte de Windows (Proyecto Mono de Ximian/Novell para Linux/MacOS X/BSD/Solaris), el desarrollo de lenguajes de programación adicionales para la plataforma (ANSI C de la Universidad de Princeton, NetCOBOL de Fujitsu, Delphi de Borland, entre otros) o la creación de bloques adicionales para la plataforma (como controles, componentes y bibliotecas de clases adicionales); siendo algunas de ellas software libre, distribuibles bajo la licencia GPL. (Cuevas Guerrero)

Entre las principales desventajas de esta plataforma, se pueden mencionar que no soporta múltiples sistemas operativos y que por ser exclusiva de Microsoft, es ésta sola empresa la única que puede añadir y quitar características según crea necesaria. (Ávila, 2007)

En el caso específico de C#, Microsoft lo diseñó desde su base para aprovechar el nuevo entorno .NET Framework. Como C# forma parte de este nuevo mundo .NET, deberá comprender perfectamente lo que proporciona .NET Framework y de qué manera aumenta su productividad. (Ferguson, y otros, 2003)

La base de la plataforma.NET la constituye el "Framework" o marco de trabajo, y este denota la infraestructura sobre la cual se reúnen un conjunto de lenguajes, herramientas y servicios que simplifican el desarrollo de aplicaciones en un entorno de ejecución distribuido.

Los principales componentes del marco de trabajo son: (Ver Figura 5):

- El conjunto de lenguajes de programación.

- El Entorno Común de Ejecución para Lenguajes o CLR.
- La Biblioteca de Clases Base o BCL.
- El entorno ASP.NET.

(Cuevas Guerrero)

Figura 5 Marco de Trabajo de .Net

.NET Framework fue diseñado con tres objetivos en mente. Primero, debía lograr aplicaciones Windows mucho más estables, aunque también debía proporcionar una aplicación con un mayor grado de seguridad. En segundo lugar, debía simplificar el desarrollo de aplicaciones y servicios Web que no sólo funcionan en plataformas tradicionales, sino también en dispositivos móviles. Por último, el entorno fue diseñado para proporcionar un solo grupo de bibliotecas que pudieran trabajar con varios lenguajes.

Los componentes del .NET Framework proveen los "ladrillos" necesarios para construir las aplicaciones Web, los servicios Web y cualquier otra aplicación dentro de Visual Studio .NET. Microsoft

ofrece herramientas de desarrollo de extrema productividad que se centran en la fase de construcción de códigos del ciclo de vida de la aplicación.

Con la introducción de Microsoft Visual Studio 97, esta perspectiva del ciclo de vida de la aplicación comenzó a ampliarse, incluyendo compatibilidad para el análisis, el diseño y el desarrollo basado en equipos. En la actualidad, con Visual Studio .NET 2005, Microsoft garantiza nuevas características del ciclo de vida empresarial que ayudan a las organizaciones a programar, analizar, diseñar, generar, probar y coordinar los equipos que producen aplicaciones y servicios Web XML. (Ferguson, y otros, 2003) (Ávila, 2007)

1.7.1. Lenguaje de Programación C-Sharp (C#)

C# (C-Sharp) es un lenguaje de programación Orientado a Objetos desarrollado por Microsoft, el cual fue presentado al público por primera vez en la Professional Developer's Conference en Orlando, Florida, en el verano del año 2000. Combina las mejores ideas de diferentes lenguajes como C, C++, Java y Visual Basic con las mejoras de productividad de .NET Framework, aunque también ha sido influenciado por Modula 2 y SmallTalk. Brinda una experiencia de codificación muy productiva, tanto para los nuevos programadores como para los más experimentados.

El principal objetivo de Microsoft fue la creación del primer lenguaje Orientado a Componentes, al estilo de Visual Basic, siendo flexible y potente como C++ pero sin muchas de sus complejidades. Se diseñó de modo que retuviera casi toda la sintaxis de C y C++. Los programadores que estén familiarizados con estos dos lenguajes pueden escoger el código C# y empezar a programar de forma relativamente rápida. Sin embargo, la gran ventaja de C# consiste en que sus diseñadores decidieron no hacerlo compatible con los anteriores C y C++.

C# elimina las cosas que hacían que fuese difícil trabajar con estos dos lenguajes, por ejemplo, los punteros. Como todo el código C es también código C++, éste tenía que mantener todas las rarezas y deficiencias de C, sin embargo, C# parte de cero y sin ningún requisito de compatibilidad, así que mantiene los puntos fuertes de sus predecesores y descarta las debilidades que complicaban las cosas a los programadores de C y C++.

Cuenta con la mayoría de las características de orientación a objetos de C++. Su entidad de primer nivel es la *clase,* de la cual se crean objetos y se derivan nuevas clases. No pueden faltar por supuesto, la encapsulación, herencia y polimorfismo. Las clases pueden agruparse en *espacios de nombres o namespaces.* No existen los conceptos de definición e implementación como elementos separados en una clase, todo se realiza en la misma, simplificando la codificación.

C# asimila, e incluso supera, las tradicionales facilidades que da Visual Basic a la hora de usar y diseñar componentes sin perder un ápice de potencia y flexibilidad. Se puede usar C# para fabricar los componentes que encapsulan la lógica de negocio y acceden a bases de datos, las interfaces de usuario que actúan como clientes de dichos componentes y el código de servidor que actúa detrás de las interfaces Web.

La destrucción de objetos y liberación de la memoria asociada se produce de manera automática, gracias a la existencia de un recolector de basura. El compilador no genera código ejecutable en ningún caso, sino que produce código MSIL (Microsoft Intermediate Language) por lo que el programador no tiene que preocuparse del sistema operativo o procesador en el que va a ejecutarse su aplicación.

(Charte Ojeda, 2002) (Ferguson, y otros, 2003)

1.8. Patrones de Diseño

Los patrones son soluciones simples y elegantes a problemas específicos y comunes. Son soluciones basadas en la experiencia, que se ha demostrado que funcionan y pueden emplearse en diferentes contextos, existen diferentes tipos de patrones, a continuación se abordarán los Patrones de Diseño por su utilidad en la presente investigación.

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. Identifican: Clases, Instancias, Roles, Colaboraciones y la distribución de responsabilidades.

1.8.1. Fábrica Abstracta (Abstract Factory)

Es un patrón que ofrece una interfaz para la creación de familias de productos relacionados o dependientes sin especificar las clases concretas a las que pertenecen. Es conocido también como *Kit*. Se debe usar cuando:

- Un sistema debe ser independiente de cómo se crean, se componen y se representan sus productos.
- Un sistema se debe configurar con una de entre varias familias de productos.
- Una familia de productos relacionados están hechos para utilizarse juntos (hay que hacer que esto se cumpla).
- Se desea ofrecer una biblioteca de productos a partir de su interfaz sin revelar su implementación.

Estructura (Ver Anexo 1)

Las clases participantes en la estructura son las siguientes:

AbstractFactory: Declara una interfaz para la creación de objetos de productos abstractos.

ConcreteFactory: Implementa las operaciones para la creación de objetos de productos concretos.

AbstractProduct: Declara una interfaz para los objetos de un tipo de productos.

ConcreteProduct: Define un objeto de producto que la correspondiente factoría concreta se encargará de crear, a la vez que implementa la interfaz de producto abstracto.

Client: Utiliza solamente las interfaces declaradas en la factoría abstracta y en los productos abstractos.

Las clases AbstractFactory a menudo son implementadas con un método factoría (Factory Method[9]), pero también usando prototipos (Prototype[10]). Una factoría concreta a veces es un Singleton.

---

[9] Patrón de Creación que define una interfaz para crear un objeto, pero deja a las subclases decidir qué clases van a instanciar.

(Dodero, y otros, 2002-2003) (Gamma, y otros, 1998)

1.8.2. Singleton

El patrón Singleton asegura que sólo se pueda crear una instancia de una clase, y ofrece un punto de acceso global a ella. Se debe usar cuando:

- Existe la necesidad de que una clase se instancie una sola vez de modo que todos los clientes puedan acceder a esa única instancia desde un punto conocido.
- La instancia única se puede ampliar mediante subclases y los clientes deben ser capaces de utilizar las instancias de las subclases sin tener que modificar su código.

Estructura (Ver Anexo 2)

La clase participante en la estructura es la siguiente:

Singleton: Es el responsable de la creación de su única instancia. Define una operación para crear instancias que ofrece a todos los clientes la misma y única instancia. Éste método debe ser estático.

Diferentes patrones pueden ser implementados usando el Singleton; entre ellos Abstract Factory, Builder[11] y Prototype.

(Gamma, y otros, 1998) (Dodero, y otros, 2002-2003)

1.8.3. Fachada (Facade)

El patrón Facade tiene como propósito ofrecer un interfaz de acceso único a un conjunto de interfaces en un subsistema. Define una interfaz de alto nivel que hace al subsistema fácil de usar. Se debe usar cuando:

---

[10] Patrón de Creación que especifica los tipos de objetos a crear usando una instancia prototípica y crea nuevos objetos mediante la copia de estos prototipos.

[11] Patrón de Creación que separa la construcción de un objeto complejo de su representación para que éste mismo proceso de construcción pueda crear diferentes representaciones.

- Se quiera proporcionar una interfaz sencilla para un subsistema complejo.
- Se quiera desacoplar un subsistema de sus clientes y de otros subsistemas, haciéndolo más independiente y portable.
- Se quiera dividir los sistemas en niveles: las fachadas serían el punto de entrada a cada nivel.

Estructura (Ver Anexo 3)

Las clases participantes en la estructura son las siguientes:

Fachada: Conoce las clases del subsistema responsables de un pedido y delega las peticiones de los clientes en los objetos del subsistema.

Clases del subsistema: Implementan la funcionalidad del subsistema y llevan a cabo las peticiones que les envía la Fachada, aunque no la conocen.

Abstract Factory puede ser usado junto a Facade, proporcionando una interfaz para crear objetos subsistemas. Por lo general, sólo un objeto Facade es requerido. De este modo, los objetos Facade son a menudo Singleton.

(Gamma, y otros, 1998)

1.9. Herramientas para el modelado.

La realización de un nuevo software requiere que las tareas sean organizadas y completadas en forma correcta y eficiente. Las Herramientas CASE fueron desarrolladas para automatizar los procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de desarrollo de software.

La principal ventaja de la utilización de una herramienta CASE, es la mejora de la calidad de los desarrollos realizados y, en segundo término, el aumento de la productividad. Para conseguir estos dos objetivos es conveniente contar con una organización y una metodología de trabajo, además de la propia

herramienta. En los siguientes subepígrafes se hará el análisis de algunas de estas herramientas CASE, con el propósito de seleccionar la que será utilizada en el desarrollo del presente trabajo.

1.9.1. Visual Paradigm.

Visual Paradigm es una herramienta CASE profesional que soporta la última versión de UML 2.1 así como el ciclo de vida completo del desarrollo de software, análisis y diseño orientados a objetos, construcción, pruebas y despliegue, exportación desde Rational Rose, exportación/importación XML, generación de informes y edición de figuras.

Visual Paradigm ofrece además:

- Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa e inversa.
- Modelo y código que permanecen sincronizados en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.
- Ingeniería Inversa Java, C++, Esquemas XML, XML, NET exe/dll, CORBA IDL.

El Visual Paradigm permite escribir toda la especificación de un caso de uso sin necesidad de utilizar una herramienta externa como editor de texto. Es posible crear Especificaciones de Casos de Uso utilizando plantillas que se encuentran definidas, o que pueden ser creadas por los usuarios.

1.9.2. Rational Rose.

- Herramienta de modelación visual que provee el modelado basado en UML.

- Es en la actualidad, una de las herramientas CASE más potentes, es la herramienta que comercializan los desarrolladores de la Corporación Rational.
- Soporta de forma completa la especificación de UML.
- Basado principalmente en el nivel de integración que tiene este con el resto de las herramientas que lo acompañan en la Suite entre las que aparecen:
    - Rational Clear CASE, para el control de versiones.
    - Rational Clear Quest, para el control de cambios.
    - Rational Model Integrator, para la integración de los artefactos.
    - Rational Requisite Pro, herramienta de administración de requerimientos.
- Brinda la posibilidad de generar y realizar ingeniería inversa (Rational, 2007) en una buena cantidad de lenguajes de programación en su versión XDE24 y el número de Framework que vienen predefinidos, entre los cuales se pueden citar .Net, J2EE, C++, Visual Basic.
- Permite que existan varias personas trabajando a la vez en el proceso iterativo controlado, para ello posibilita que cada desarrollador opere en un espacio de trabajo privado que contiene el modelo completo y tenga un control exclusivo sobre la propagación de los cambios en ese espacio de trabajo.

Rational Rose es una herramienta CASE potente que se integra bien con el Proceso de Desarrollo de Software (RUP) y encaja bien en procesos de negocio complejos y por ende proyectos grandes. Sin embargo, en la Universidad de las Ciencias Informáticas se encuentra en su versión de 2003 aún. El Lenguaje Unificado de Modelado se versiona y en la actualidad existe otra herramienta, en este caso Enterprise Arquitect que resulta compatible con UML 2.1 y que al igual que Rational Rose describe bien a RUP.

1.9.3. Herramienta de Modelado Enterprise Architect

(Sparx Systems, 2007)

Es una herramienta de modelado desarrollada por Sparx System [12]que permite tanto modelar en últimas versiones de UML (UML 2.1), como rastrear la información importante de proyecto con artefactos de diseño.

Enterprise Architect es una herramienta comprensible de diseño y análisis UML, que cubre el desarrollo de software desde el paso de los Requerimientos a través de las etapas del Análisis, Diseño, Pruebas y Mantenimiento. EA es una herramienta multiusuario, basada en Windows, diseñada para ayudar a construir software robusto y fácil de mantener. Ofrece una salida de documentación flexible y de alta calidad.

El Lenguaje Unificado de Modelado provee beneficios significativos para ayudar a construir modelos de sistemas de software rigurosos y donde es posible mantener la trazabilidad de manera consistente. Enterprise Architect soporta este proceso en un ambiente fácil de usar, rápido y flexible.

Sus bases están construidas sobre la especificación de UML 2.0. Tiene soportes para los diferentes diagramas de éste (Diagrama de Clases, de Objetos, Secuencia, Componentes, Casos de Uso etc.)

Provee una trazabilidad completa desde el análisis de Requerimientos hasta los artefactos de Análisis y Diseño, a través de la Implementación y el Despliegue. Combinados con la ubicación de recursos y tareas incorporados, los equipos de Administradores de Proyectos y Calidad están equipados con la información que ellos necesitan para entregar proyectos en tiempo.

Enterprise Architect suministra una generación poderosa de documentos y herramientas de reporte con un editor de plantilla completo. Ayuda a administrar la complejidad con herramientas para rastrear las dependencias, tiene soporte para modelos muy grandes y control de versiones con proveedores CVS o SCC.

Soporta generación e ingeniería inversa de código fuente para muchos lenguajes populares, incluyendo C++, C#, Visual Basic.Net, Java, Delphi, Visual Basic y PHP. También hay disponibles Add-ins

[12] Sparx Systems es una empresa australiana Líder en el Análisis y Diseño de Sistemas y cuenta con varios Productos para realizar estas actividades que están basadas en el estándar UML 2.1 de la OMG.

gratis para CORBA y Python. Cuenta con un editor de código fuente con "resaltador de sintaxis" incorporado.

EA soporta transformaciones de Arquitectura avanzada dirigida por Modelos (MDA) usando plantillas de transformaciones de desarrollo y fáciles de usar. Con transformaciones incorporadas para DDL, C#, Java, EJB y XSD, se pueden desarrollar soluciones complejas desde los simples "Modelos Independientes de Plataforma" (MIP) que son el objetivo en "Modelos Específicos de Plataforma" (MEP). Un Modelo Independiente de Plataforma se puede usar para generar y sincronizar múltiples modelos, proveyendo un aumento de productividad significativo.

(Sparx Systems, 2007)

1.10. Métricas

La aplicación de métricas de la calidad de software en las primeras etapas del ciclo de vida de un software tiene gran importancia en los resultados finales con respecto a la calidad. La medición permite tener una visión más profunda proporcionando un mecanismo para la evaluación objetiva. (Pressman, 2002)

A continuación se muestran algunas métricas que se aplican actualmente para validar la calidad de software. Proporcionan una medida de cuán evolucionado se encuentra el desarrollo de la aplicación informática desde la visión interna que proporcionan los parámetros que estas definen.

1.10.1. Métricas del Modelo de Diseño

Las métricas del Modelo de Diseño proporcionan al diseñador una mejor visión interna y ayudan a que el diseño evolucione a un nivel superior de calidad. Se dividen en:

- Métricas de diseño arquitectónico.
- Métricas de Diseño a nivel de componente.

A continuación se muestran algunas de las métricas del Modelo de Diseño.

Métricas de Diseño Arquitectónico

Las métricas de diseño arquitectónico están contenidas dentro de las Métricas del Modelo de Diseño y se concentran en las características de la arquitectura del programa, con especial énfasis en la estructura arquitectónica y en la eficiencia de los módulos o clases. Se componen de:

- Métricas de complejidad propuestas por Card y Glass en 1990.
- Complejidad estructural: S (i) = f2 out (i), donde fout (i) = expansión del módulo i (número de módulos inmediatamente subordinados al módulo i).
- Complejidad de Datos. Sobre la interfaz interna del módulo: D (i) = v(i) / [fout (i) + 1], donde v(i) es el número de variables que entran o salen del módulo.
- Complejidad del Sistema: C (i) = S(i) + D(i).

A medida que crecen los valores de complejidad, crece la complejidad arquitectónica del sistema.

En 1981 Henry y Kafura proponen la métrica de complejidad de expansión – concentración, que considera las estructuras de datos que recoge (concentran) o actualizan (expansión): MHK = longitud (i) x [fin (i) + fout (i)]2 donde longitud (i) = número de sentencias en lenguaje de programación. En 1991, Fenton propone medidas de morfología simples basadas en la estructura de módulos jerárquicos del sistema:

- Tamaño = n + a (número de nodos + número de aristas).
- Profundidad.
- Anchura.
- Relación arco-nodo, r = a/n.

Métricas Orientadas a Clase

Estas métricas se incluyen dentro de las Métricas de Diseño Arquitectónico y según los autores Chidamber y Kemerer han propuesto seis métricas basadas en clases para sistemas OO:

- Métodos ponderados por clase (MPC).
- Árbol de profundidad de herencia (APH).
- Número de descendiente (NDD).
- Acoplamiento entre clases objeto (ACO).
- Respuesta para una clase (RPC).
- Carencia de cohesión en los métodos (CCM).

En su libro sobre métricas OO, Lorenz y Kidd separan las métricas basadas en clases en cuatro amplias categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas al tamaño para las clases OO se centran en el recuento de atributos y operaciones para cada clase individual, y los valores promedio para el sistema OO como un todo. Las métricas basadas en la herencia se centran en la forma en que las operaciones se reutilizan en la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y los aspectos orientados al código; las métricas orientadas a valores externos, examinan el acoplamiento y la reutilización.

- Tamaño de clase (MPC).
- Número de operaciones redefinidas por una subclase (NOR).
- Número de operaciones añadidas por una subclase (NOA).

Métricas de Diseño a Nivel de Componentes

Se incluyen dentro de las Métricas del Modelo de Diseño y se centran en las características internas de los componentes del software e incluyen las medidas de las “3C”: Cohesion (Cohesión); Coupling (Acoplamiento) y Complexity (Complejidad). Pueden aplicarse antes o después de tener el código.

Métricas de Cohesión

Son Métricas de Diseño a Nivel de Componentes y se encargan de medir si la cohesión se puede usar en el trabajo de Bieman y Ott, el cual define varios conceptos:

- Número de elementos (tokens).
- Rebanada de datos (data slice).
- Adhesivo (glue) y superadhesivo (superglue).

Con ellas se calcula:

Cohesión funcional fuerte: CFF = número de superadhesivos / número de elementos.

Cohesión Funcional Débil: CFD = número de adhesivos / número de elementos.

Mientras más cercano sean CFF ó CFD a 1, mayor será la cohesión del módulo.

Métricas de Acoplamiento

Son Métricas de Diseño a Nivel de Componentes y proporcionan una indicación de la conectividad de un módulo con otros propuestos por Dhama en 1995. Constituyen medidas para el acoplamiento de flujo de datos de control. Incluye algunos parámetros a tener en cuenta:

- di = número de parámetros de datos de entrada.
- ci = número de parámetros de control de entrada.
- do = número de parámetros de datos de salida.
- co = número de parámetros de control de salida.

Como medidas para el acoplamiento global esta métrica incluye el número de variables globales usadas como datos (gd) y el número de variables globales usadas como control (gc) y dentro de las medidas para el acoplamiento de entorno incluye el número de módulos llamados (w) y el número de módulos que llaman al módulo en cuestión (r). El indicador de acoplamiento de módulo: mc = k / M, donde

k = 1 y la constante de proporcionalidad es M = di + a x ci + do + b x co+gd + c x gc + w + r, con a = b = c = 2. Cuando mayor es mc, menor es el acoplamiento del módulo.

Métricas del Código Fuente

- La forma de calcularlas es con las siguientes cantidades:
- n1 = número de operadores diferentes en el programa.
- n2 = número de operandos distintos en el programa.
- N1 = número total de veces que aparecen los operadores.
- N2 = número total de veces que aparecen los operandos.

Luego las métricas de Halstead, serían:

- Longitud global del programa: N = n1 log2 n1 + n2 log2 n2.
- Volumen del programa: V = N log2 (n1 + n2).
- Volumen compacto (ya que V varía dependiendo del lenguaje): L = 2 / n1 x n2 / N2.

1.10.2. Métricas de Prueba

Las métricas de prueba que existen se concentran en el proceso de prueba y no en las características técnicas de la prueba. Los responsables de la prueba, se guían por las métricas de análisis, diseño y código.

- Puntos de Función.
- Bang.
- Halstead.
- Complejidad ciclomática.

Métricas de Pruebas de Unidad

Se incluyen dentro de las Métricas de Prueba. El término prueba de unidad se refiere a la prueba individual de unidades separadas de un sistema de software. En sistemas orientados a objetos, estas unidades son, típicamente, clases y métodos. Así, en el contexto que ocupa, prueba de unidad se refiere a la prueba individual de métodos y clases.

Las **pruebas de caja blanca** permiten examinar la estructura interna del programa realizando un seguimiento del código fuente según va ejecutando los casos de prueba, de manera que se determinan concretamente las instrucciones, bloques en los que existen errores.

Para la realización de las pruebas de unidad se diseñan casos de prueba que se encargan de examinar la lógica del sistema. Los casos de prueba garantizan que se ejerciten todos los caminos independientes de cada módulo o clase y todas las decisiones lógicas así como que se ejecuten todos los bucles y las estructuras de datos internas.

Las **pruebas de caja negra** se llevan a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa. Los casos de prueba de la caja negra pretenden demostrar que las funciones del sistema son operativas; que las entradas son aceptadas de forma adecuada y que las salidas correspondientes son las correctas. Además garantizan que la información externa se mantiene.

Los errores esperados durante la realización de las pruebas de caja negra al sistema informático que se propone se agrupan en las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Las pruebas de regresión por su parte son una estrategia de prueba en la cual las pruebas que se han ejecutado anteriormente se vuelven a realizar en la nueva versión modificada, para asegurar la calidad después de añadir la nueva funcionalidad. El propósito de estas pruebas es asegurar que los defectos identificados en la ejecución anterior de la prueba se han corregido y que los cambios realizados no han introducido nuevos defectos o reintroducido defectos anteriores. (Pressman, 2002)

1.11. Arquitectura de software

En Informática, la Arquitectura de Software según (Pressman, 2002 pág. 238) es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos.

La arquitectura constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y de cómo trabajan juntos sus componentes facilitando así la comunicación entre todas las partes implicadas en el desarrollo del software

Según (A field guide to Boxology: Preliminary classification of architectural styles for software systems., Abril de 1996), la arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.

Otras de las definiciones de arquitectura de software la brinda el documento de la IEEE[13] Std 1471-2000 que plantea: "La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución". Esta es la definición que se ha tomado como oficial por todas las instituciones que desarrollan software a nivel mundial, como ha sido por ejemplo Microsoft.

[13] IEEE, (Institute of Electrical and Electronics Engineers, Inc., Instituto de electrónica e ingenieros electrónicos) es la asociación líder mundial de profesionales. Su propósito fundamental es adoptar innovaciones tecnológicas y la excelencia en beneficio de la humanidad. Reconocida mundialmente por las contribuciones tecnológicas.

1.12. Conclusiones

Las investigaciones realizadas acerca del estado actual de las metodologías de desarrollo de software y las herramientas de modelado han permitido adoptar como ya se expuso en los anteriores epígrafes los artefactos propuestos por la metodología RUP, ya que es una de las más aplicadas en la actualidad y que documenta con mejores detalles el desarrollo de una solución de software. Se tomaran las ventajas que brinda la metodología XP por su flexibilidad a cambios e interacción con los clientes finales. Sin embargo no se adoptará una metodología en específico pues el desarrollo de la solución no cumple con todas las características necesarias para adoptar una u otra metodología.

El lenguaje y plataforma de desarrollo seleccionada fueron Microsoft Visual Studio 2005 con CSharp.NET ya que la solución propuesta es una herramienta que servirá de apoyo al desarrollo en la plataforma .NET.

La Herramienta Case seleccionada para la solución es Enterprise Architect por ser una herramienta comprensible de análisis y diseño UML, cubriendo el desarrollo de software desde el modelado de los procesos del negocio hasta su fases finales de despliegue. Permite crear un repositorio, logrando de esta forma un control versiones que posibilita coordinar el trabajo sobre los componentes del software, siendo posible la recuperación de versiones anteriores. Además que brinda opciones de integración con el lenguaje y la plataforma de desarrollo seleccionada.

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

2.1. Introducción

En el capítulo anterior se llegó a la conclusión de adoptar los artefactos generados por la metodología RUP como guía para la construcción del diseño e implementación del sistema propuesto, además las facilidades que brindan los elementos de comunicación con los clientes y cambio de requisitos que plantea la metodología XP. El objetivo del presente capítulo es desarrollar los artefactos más importantes para estos flujos de trabajos, partiendo de la fundamentación del tema, el estudio del arte de las diferentes herramientas y tecnologías existentes. ¿Pero exactamente que es el Diseño de Software Orientado a Objetos? Según define Pressman, requiere la definición de una arquitectura de software multicapa, la especificación de subsistemas que realizan funciones necesarias y proveen soporte de infraestructura, una descripción de objetos (clases), que son los bloques de construcción del sistema y una descripción de los mecanismos de comunicación, que permiten que los datos fluyan entre las capas, subsistemas y objetos.

2.2. Breve descripción de la Arquitectura.

2.2.1. Arquitectura en Capas

La Arquitectura en Capas constituye uno de los estilos que aparecen con mayor frecuencia mencionados como categorías mayores del catálogo. Según (A field guide to Boxology: Preliminary classification of architectural styles for software systems., Abril de 1996), definen el estilo en capas como una organización jerárquica, tal que cada capa proporciona servicios a cada capa inmediatamente superior y se beneficia de las prestaciones que le brinda la inmediatamente inferior.

De esta arquitectura puede decirse que su finalidad es abstraer las funcionalidades de una capa, de manera tal que pueda ser totalmente reemplazada. La más común es la compuesta por tres capas, presentación, modelo o reglas del negocio y acceso a datos. De esta forma, se puede reemplazar cualquier capa sin afectar las otras, cambiando solamente las referencias de las implicadas en el cambio. Aunque sea la de tres capas la más común, esto no significa que mientras crezca la complejidad las mismas se mantengan invariantes, sino al contrario, si la complejidad crece y es necesario, entonces pueden aparecer otras capas para descomponer las funcionalidades que en estas aparezcan. A su vez

estas capas pueden estar compuestas por subcapas y una capa o subcapa puede estar compuesta por una o varias clases del diseño.

Esta característica de la arquitectura de capas permite implementar las reglas del negocio en una capa aparte, para que estas reglas puedan ser usadas por otros sistemas o por otros servicios que necesiten dichas funcionalidades, evitando la duplicación de código y mejorando la organización.

Según (Trowbridge, 2003), las ventajas que presenta la arquitectura de capas son:

- El mantenimiento y las mejoras de la solución son fáciles debido al bajo acoplamiento entre las capas, la alta cohesión de las capas y la habilidad de cambiar su implementación sin cambiar las interfaces. Esto es muy importante para el desarrollo del sistema, pues este debe permitir cambios ya sea de requisitos funcionales, agregaciones o mejoras en las funcionalidades.
- Otras soluciones pueden rehusar las funcionalidades expuestas por las diferentes capas, especialmente si las capas de las interfaces son diseñadas con la reutilización en mente.
- El desarrollo distribuido es fácil si este se puede dividir con las capas como fronteras. Permitiendo el desarrollo de forma paralela, posibilitando la especialización de los desarrolladores e incrementando la productividad.
- Distribuir las capas a lo largo de múltiples capas físicas puede mejorar la escalabilidad, tolerancia a errores y rendimiento. Esto lo que quiere decir es que como una capa física puede estar compuesta por una o más computadoras, si se logra distribuir, el rendimiento aumenta y de esta forma el desempeño, la durabilidad y la escalabilidad del sistema.
- Beneficios a la hora de realizar las pruebas, teniendo bien definidas las interfaces de las capas por la habilidad de cambiar las implementaciones de estas capas manteniendo la interfaz.

También se señalan algunas desventajas en este estilo como son:

- La sobrecarga extra de pasar los mensajes a través de las capas en lugar de llamar los componentes directamente, puede impactar de forma negativa en el rendimiento. Lo cual se puede mitigar con el uso de un modo relajado.

- El desarrollo de las interfaces de usuario puede algunas veces tomar tiempo si la estructura de las capas evita el uso de componentes de interfaz de usuario que interactúan directamente con la Base de Datos. Esto ya ha sido mitigado por la plataforma de desarrollo que se escogió para la construcción del sistema, en la cual la mayoría de los componentes interactúan con DataSet[14].

- Cambios en las interfaces de las capas inferiores tienden a propagarse a los altos niveles, especialmente si el modo relajado es usado.

El modo relajado es una de las formas de concebir la arquitectura en capas, se le llama modo relajado cuando las capas superiores pueden interactuar con las capas inferiores, esto aumenta el rendimiento pero implica menos flexibilidad en la aplicación, mientras que la otra forma de concebir esta arquitectura es que una capa esté ligada únicamente con la capa inferior inmediata, esta mantiene un bajo acoplamiento pero puede implicar impactos negativos en el rendimiento, según (Almenares, y otros, 2007).

La solución propuesta está constituida básicamente por dos componentes. El primero lo constituye la infraestructura para interpretar y traducir las excepciones configuradas y el segundo es un sistema que permite al usuario interactuar y configurar de una manera sencilla las excepciones personalizando las acciones que desea se realicen cuando sea capturada la excepción.

La arquitectura en capa fue escogida para el desarrollo del Módulo Configurador de las excepciones dadas sus características y las ventajas que se han sido estudiadas. Lógicamente existe cierta relación entre estos los dos elementos de la solución como parte de las estructura necesaria para el procesamiento de los datos de las excepciones y su persistencia en uno o varios archivos XML que posteriormente utilizaría la aplicación que se sirva de el intérprete para el tratamiento y manipulación de

[14] Según (Bipin), es una estructura de datos del Microsoft .Net framework que encapsula un conjunto de datos y soporta un modelo relacional de una o más tablas.

las excepciones identificadas y configuradas previamente. En la Figura 6 se muestra la arquitectura del sistema.

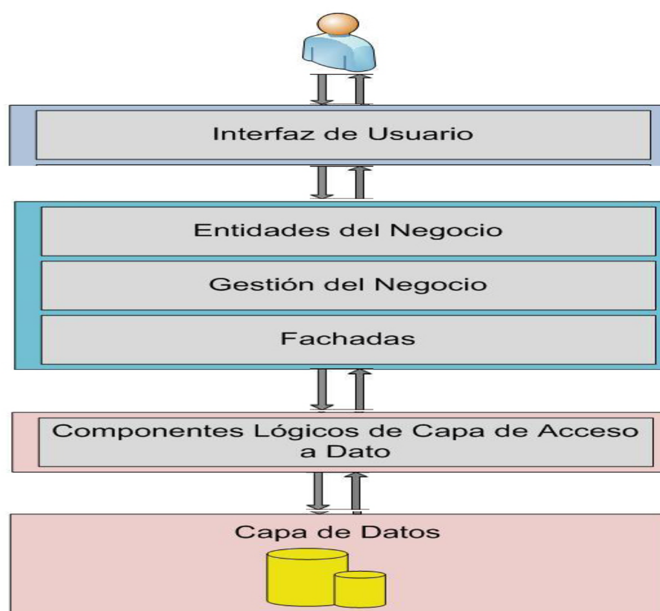


Figura 6 Arquitectura en Capas.

El intérprete que se desea construir no tiene una gran complejidad, sin embargo se adapta bien a una arquitectura en capas. En general es una librería de clases brindando varios servicios a los desarrolladores que desean utilizarla. Más adelante con el modelado de clases del diseño y los diagramas de interacción correspondientes, se le dará explicación a este componente de la solución.

2.2.2. Modelo basado en capas.

Los módulos Configurador de Excepciones y el Intérprete de excepciones utilizan el Estilo Arquitectónico basado en Capas, el cual permite distribuir el trabajo en varios paquetes.

Capa de Presentación.

Aquí se encuentran los Formularios de Interfaz de Usuario del módulo Configurador de Excepciones así como las del módulo Manejador, los cuales responden a un comportamiento y diseño estándar. Su uso se basa en la explotación de estos formularios que realmente provee el Framework de Visual Studio 2005 facilitando el desarrollo del sistema.

Capa Lógica de Negocio.

El diseño de esta capa depende directamente del negocio específico del módulo. El Negocio recibe datos o información capturada en las interfaces de usuario, gestiona o procesa la misma, de ser necesario solicitándola a la Capa de Acceso a Datos y finalmente se la envía a la de Presentación nuevamente para que ésta la presente al usuario en el punto donde se inició la petición.

Las “entidades” del negocio son clases objeto-valor que representan los datos con los que se van a trabajar en cada uno de los procesos que se están automatizando. Cada entidad se encarga de procesar sus propios datos o valores sin interactuar con los demás elementos del negocio, con esto se garantiza independencia y encapsulamiento de la información.

Los “gestores” tienen como objetivo agrupar una serie de entidades en un fin común, y así, obtener funcionalidades donde intervienen un conjunto de datos que se encuentran en dichas clases objeto-valor. Son recursos utilizados para encapsular una o varias funciones determinadas y que serán utilizados por más de un proceso de negocio en el módulo, estas funcionalidades son netamente de este nivel, no tienen Presentación pero sí pueden utilizar recursos del Acceso a Datos.

Capa Acceso a Datos.

Es la más crítica y sensible pues controla todo lo relacionado a la información que se encuentra en la fuente de almacenamiento (Capa de Datos). Al ser la capa inferior, no conoce los niveles superiores, sólo se limita al manejo de la información, ya sea para persistirla o proporcionarla para su procesamiento y propagación por la aplicación. Aquí se propone la librería ADO.NET del Visual Studio 2005.

ADO.NET es un conjunto de clases que exponen servicios de acceso a datos para el programador de .NET. ADO.NET ofrece abundancia de componentes para la creación de aplicaciones de uso compartido de datos distribuidas. Constituye una parte integral de .NET Framework y proporciona acceso a datos relacionales, XML y de aplicaciones. ADO.NET satisface diversas necesidades de desarrollo, como la creación de clientes de base de datos de aplicaciones para usuario y objetos empresariales de nivel medio que utilizan aplicaciones, herramientas, lenguajes o exploradores de Internet. (Microsoft, 2008)

La fachada es un recurso usado para buscar la abstracción, es decir, brindar una especie de interfaz con la cual se va a interactuar cada vez que se necesite comunicarse, en este caso, con el Acceso a Datos logrando una completa enajenación ante cualquier modificación que pueda ocurrir.

Esta estructura responde a la implementación del patrón Fachada y todas las clases que conformen esta capa serán estáticas debido a que no es necesario instanciarlas, solamente acceder a sus funcionalidades que realmente son un puente de acceso al resto.

En la lógica de Acceso a Datos recaen todas las funcionalidades del Acceso a Datos ya que estas clases implementan la totalidad de las operaciones de persistencia y obtención de datos explotando los recursos que brinda ADO.NET, como el trabajo con procedimientos almacenados y métodos de persistencia o consultas.

Capa de Datos.

Es aquella que corresponde a los almacenes de datos, a ella pertenece la base datos del módulo Configurador de Excepciones que está compuesta por nueve tablas interrelacionadas y 37 procedimientos almacenados. Está diseñada para instalarse en un sistema gestor de base de datos Microsoft SQL Server 2000. Es un sistema de gestión de bases de datos relacionales desarrollado por Microsoft. Para el desarrollo de aplicaciones más complejas (tres o más capas), Microsoft SQL Server incluye interfaces de acceso para la mayoría de las plataformas de desarrollo, incluyendo .NET, además de ser uno de los mejores sistemas de gestión de bases de datos relacionales (SGBD) basada en el lenguaje SQL, capaz de poner a disposición de muchos usuarios grandes cantidades de datos de manera simultánea.

2.3. Modelo de Diseño.

El punto de partida para la elaboración de este diseño fueron las especificaciones de los casos de uso del sistema, los requisitos asociados a estos y las reglas generales del negocio. El análisis de los requisitos es una tarea vital dentro del proceso de desarrollo de software, porque cubre el espacio que existe entre las ideas que forman la definición del software a nivel de procesos de negocio y el diseño del software. Este análisis permite detallar en las características funcionales del software, así como en algunas restricciones que deba cumplir el sistema.

El Diseño de un software es tanto un Proceso como un Modelo. El "Proceso" de Diseño es una secuencia de pasos que hacen posible que el diseñador describa todos los aspectos del software que se va a realizar mediante el cual los requisitos se traducen en un "plano" para construir el software. Un conocimiento creativo, gran experiencia en el tema, un sentido de lo que hace que un software sea bueno, y un compromiso general con la calidad son factores críticos de éxito para un diseño competente.

El "Modelo" es el equivalente a los planos de un arquitecto para una casa. Comienza representando la totalidad de lo que se va a construir y refina lentamente lo que va a proporcionar la guía para construir cada detalle. De manera similar, el Modelo de Diseño que se crea para el software proporciona diversos enfoques diferentes del sistema (Pressman, 2002).

2.3.1. Clases del Diseño.

El diseño del sistema se ajusta a la arquitectura definida por el proyecto desde su etapa inicial. De esta forma, son identificadas cinco tipos de clases: Formularios, Gestores, Entidades, Fachadas y de Acceso a Datos.

Formularios.

Estas son clases que contienen los controles y componentes visuales necesarios para construir las entidades que son utilizadas para la entrada de datos o para mostrar objetos almacenados. Sus nombres comienzan con las letras "frm" por lo que se pueden distinguir fácilmente de otras.

A continuación, se muestra en la Figura 7 un ejemplo de clase Formulario.

```
Form
frmConfigurarManejador
- acciones: List<Accion>
- btnAdicionarAccion: System.Windows.Forms.Button
- btnEliminar: System.Windows.Forms.Button
- btnExaminar: System.Windows.Forms.Button
- btnFinalizar: System.Windows.Forms.Button
- cbxAccion: System.Windows.Forms.ComboBox
- cbxArchivar: System.Windows.Forms.CheckBox
- cbxSalir: System.Windows.Forms.CheckBox
- Column1: System.Windows.Forms.DataGridViewTextBoxColumn
- Column2: System.Windows.Forms.DataGridViewCheckBoxColumn
- Column3: System.Windows.Forms.DataGridViewCheckBoxColumn
- components: System.ComponentModel.IContainer = null
- dgvAcciones: System.Windows.Forms.DataGridView
- epAyudaErrores: System.Windows.Forms.ErrorProvider
- groupBox3: System.Windows.Forms.GroupBox
- groupBox8: System.Windows.Forms.GroupBox
- groupBox9: System.Windows.Forms.GroupBox
- label1: System.Windows.Forms.Label
- label2: System.Windows.Forms.Label
- label3: System.Windows.Forms.Label
- label4: System.Windows.Forms.Label
- label5: System.Windows.Forms.Label
- label7: System.Windows.Forms.Label
- manejador: Manejador
- tbxAyuda: System.Windows.Forms.TextBox
- tbxMensaje: System.Windows.Forms.TextBox
- tbxNuevaExcepcion: System.Windows.Forms.TextBox

- ActualizarAcciones() : void
- btnAdicionarAccion_Click(object, EventArgs) : void
- btnEliminar_Click(object, EventArgs) : void
- btnExaminar_Click(object, EventArgs) : void
- btnFinalizar_Click(object, EventArgs) : void
# Dispose(bool) : void
- frm_adicionarExcepcion(object) : void
+ frmConfigurarManejador()
- InitializeComponent() : void
- nombre(string) : string
«event»
+ adicionarManejador() : adicionarObjetoEventHandler
+ cancelar() : cancelarAccionEventHandler
```

Figura 7 Clase Formulario.

Estas vistas de manera general heredan del formulario definido en el Framework del Visual Studio 2005 que es: Forms, que representa una ventana o un cuadro de diálogo que constituye la interfaz de usuario. La Figura 8 muestra un ejemplo de un formulario.

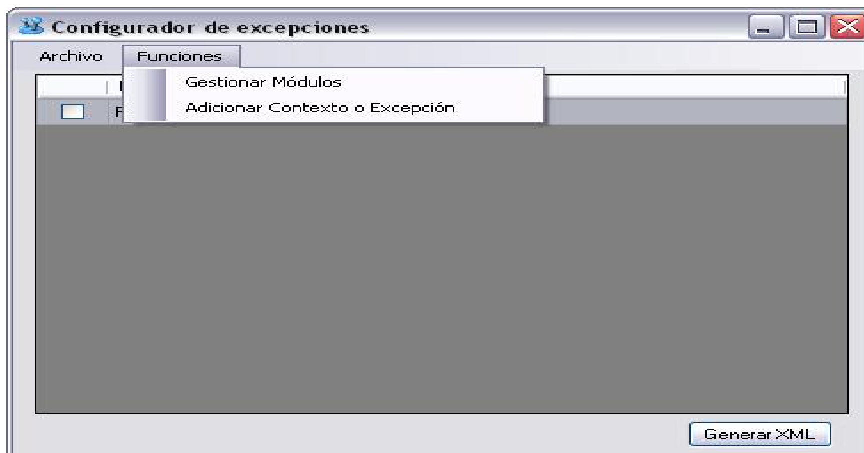


Figura 8 Formulario Generar XML configurador.

Gestores.

Son las clases que contienen la lógica del negocio y manipulan las operaciones internas del sistema y los datos. Sus nombres comienzan con el prefijo “Gtr” buscando uniformidad en el código. Se construyó un gestor por módulo, pues no implicaban gran carga de trabajo; o sea existen dos gestores uno es GtrConfiguradorGlobalExcepciones (Figura 9) y el otro es GtrProcesamientoGlobalExcepcion (Figura 10).

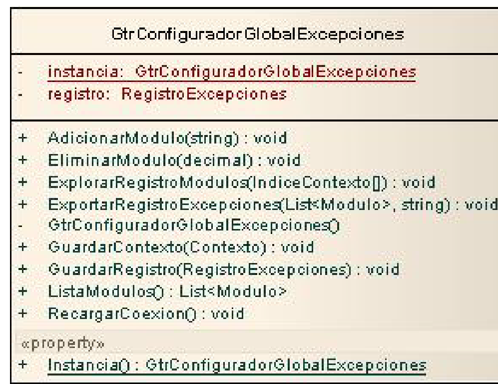


Figura 9 Clase gestor GtrConfiguradorGlobalExcepciones

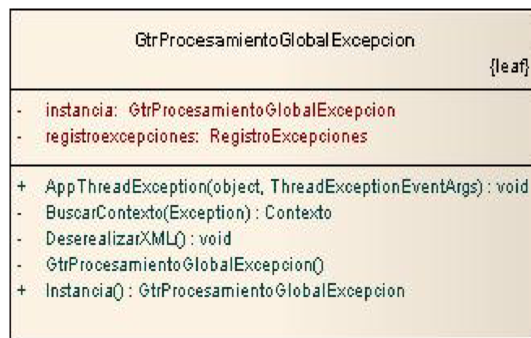


Figura 10 Clase gestor GtrProcesamientoGlobalExcepcion.

Entidades.

Las entidades son las clases que representan objetos o conceptos del negocio y mayormente se identifican por su nombre (en la Figura 11 se muestra un ejemplo de una clase entidad). Estas se identifican con un nombre sugerente a su representación.

Además de contener toda la información del sistema, las entidades realizan algunos cálculos y operaciones sobre los datos que poseen, siempre evitando que sean muy complejos, pues en ese caso sería tarea de una clase gestora desempeñar dicha función.

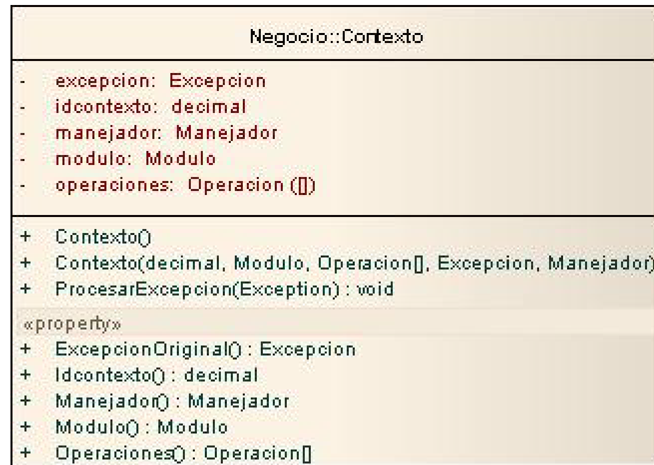


Figura 11 Clase entidad Contexto.

Fachadas.

Las clases que sirven de fachada son un punto intermedio entre los objetos del negocio de una funcionalidad completa y otros objetos o sistema que pretenden utilizar las funcionalidades que brindan estos grupos de clases. Por convenio se agrega al principio del nombre de estas clases la palabra “Fachada”.

Se construyó en la mayoría de los casos una clase fachada para cada grupo de entidades que forman un proceso del negocio. A continuación se muestra en la Figura 12 un ejemplo de una clase de este tipo.

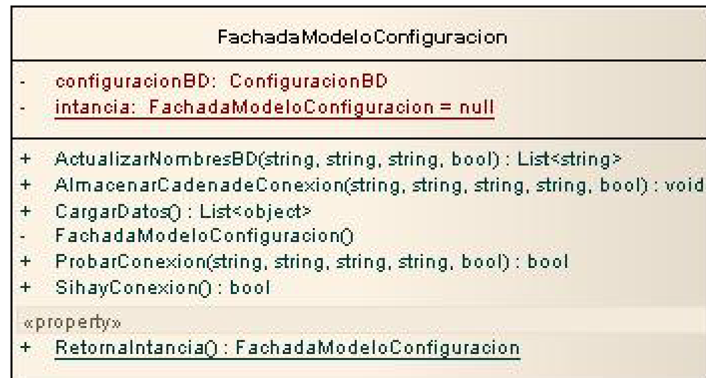


Figura 12 Clase fachada FachadaModeloConfiguracion.

Acceso a datos.

Las clases de Acceso a Datos son instancias que encapsulan todas las funcionalidades destinadas a operaciones con Base de Datos y Ficheros que requiere la aplicación. Dígase inserción, actualización, eliminación de elementos, así como llamadas procedimientos almacenados en el caso de base de dato y carga o creación de ficheros XML en el caso de archivos. En la Figura 13 se muestra un ejemplo de una clase de este tipo.

Estos objetos se encuentran en la capa de Acceso a Datos donde abstraen todo el negocio de la lógica de persistencia del resto de las capas, proporcionándoles sólo un servicio, sin importar como éstos están implementados.

Figura 13 Clase de acceso a datos Conexion.

2.3.2. Patrones de Diseño

La utilización de patrones en una arquitectura resulta una decisión acertada y eficiente, debido a que constituyen buenas prácticas y soluciones a problemas muy comunes en el desarrollo de cualquier aplicación. En la solución se implementan varios patrones como son el Singleton (Solitario), Facade (Fachada) y el Factory (Fábrica).

- **Solitario** es un Patrón de Creación cuyo propósito es garantizar que una clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma y que el acceso a ésta esté más controlado. Se usa generalmente en las clases gestoras de negocio, ya que es necesario tenerlas una sola vez instanciadas, y no pueden ser estáticas porque ellas cumplen otras necesidades o responsabilidades con otros objetos.

Permite que se reduzca el espacio de nombres (namespace) frente al uso de variables globales y refinamientos en las operaciones y en la representación mediante la especialización por herencia. Es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable).

- **Fachada** es un Patrón de Estructura, con el objetivo de proporcionar una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La "Fachada" satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas.

Separa al cliente de los componentes del subsistema, reduciendo el número de objetos con los que el cliente interactúa, facilitando entonces el uso del subsistema. También se promueve un acoplamiento débil entre éste y sus clientes, eliminándose o reduciéndose las dependencias. No existen obstáculos para que las aplicaciones usen las clases del subsistema que necesiten, de esta forma se puede elegir entre facilidad de uso y generalidad.

- **Fábrica** se usa generalmente para la creación de los objetos. Dentro de ella también se utiliza el patrón Solitario.

2.3.3. Principales Casos de Uso del Sistema

Los Casos de Usos del sistema son:

Cada forma en que los actores usan el sistema se representa con un caso de uso. Los casos de uso son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. De manera más precisa, un caso de uso especifica una secuencia de acciones que el sistema puede llevar a cabo interactuando con sus actores, incluyendo alternativas dentro de la secuencia. (Jacobson, y otros, 2002)

El módulo Configurador cuenta con tres casos de usos del sistema que son los siguientes: Gestionar XML Configurador, Adicionar Contexto y Gestionar Módulos.

Y el intérprete de excepciones presenta un caso de uso que se llama Manejar Excepción.

Se abordarán cuatro casos de uso en general, para una total descripción del la solución. También se exponen los Actores del Sistema[15] y una breve descripción de los mismos en la siguiente tabla:

Actor del Sistema	Descripción
Sistema	Es el sistema que utiliza el intérprete de excepciones para su tratamiento de excepciones.
Configurador de Contexto[16].	Es la persona encargada de adicionar al XML de configuración un nuevo contexto encontrado; generalmente son los programadores del sistema que usa el intérprete de excepciones propuesto.

Tabla 1 Descripción de los Actores del Sistema.

- **Intérprete de Excepciones.**

Es iniciado por el sistema que utiliza el manejador al lanzar una excepción y ser capturada por el hilo de ejecución ThreadException del sistema, tiene un solo caso de uso que se muestra a continuación.

Manejar Excepción: Consiste en dar el tratamiento; que se configuró con anterioridad, a la excepción capturada por el hilo ThreadException del sistema. Se identifica el contexto en que ocurre la excepción, ejecutando las acciones correspondientes a ese ámbito. El contexto está compuesto por una lista de Operaciones que se hacen para reconocer que es el contexto correcto, el módulo de la aplicación al que pertenece el error identificado, la excepción que lo define y por último un manejador que es el que contiene las acciones a realizar cuando se encuentra el contexto. En la Figura 14 se muestra el diagrama de clases “Manejar Excepción”.

[15] Constituyen una idealización de una persona externa, de un proceso, o de una cosa que interactúa con un sistema, un subsistema, o una clase. Puede ser un ser humano, otro sistema informático, o un cierto proceso ejecutable.

[16] Ámbito que conceptualiza la ocurrencia de una excepción en un sistema que contiene también que acciones realizar cuando esta se presenta además de las operaciones a realizar para identificar correctamente que es ese el contexto.

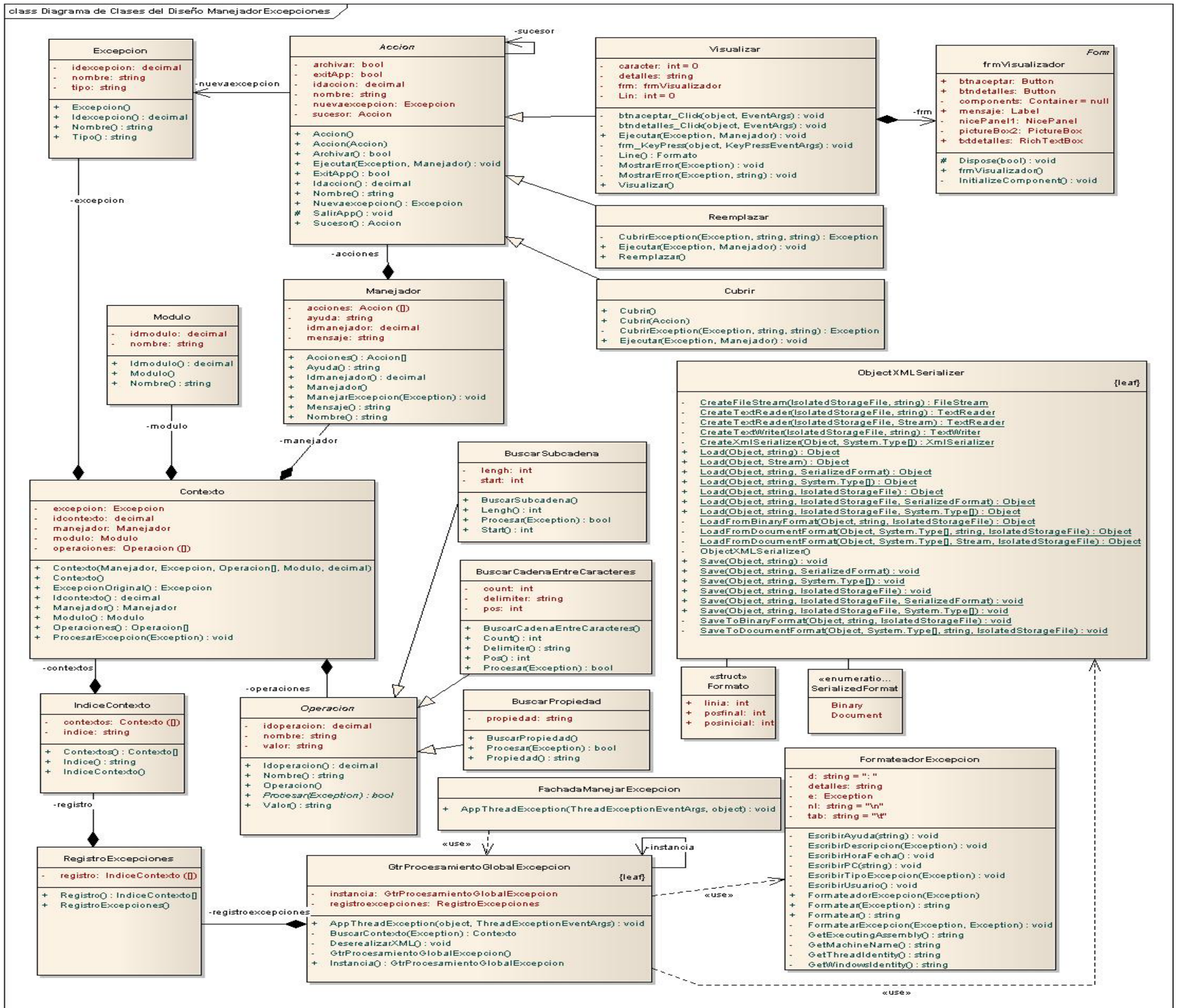


Figura 14 Diagrama de Clases "Manejar Excepción".

El caso de uso **Manejar Excepción** cuenta con un escenario de secuencia para su realización; cubrir, reemplazar y visualizar excepción. En la Figura 15 se muestra el diagrama para el escenario cubrir excepción los dos restantes se encuentran en los anexos 4 y 5 respectivamente.

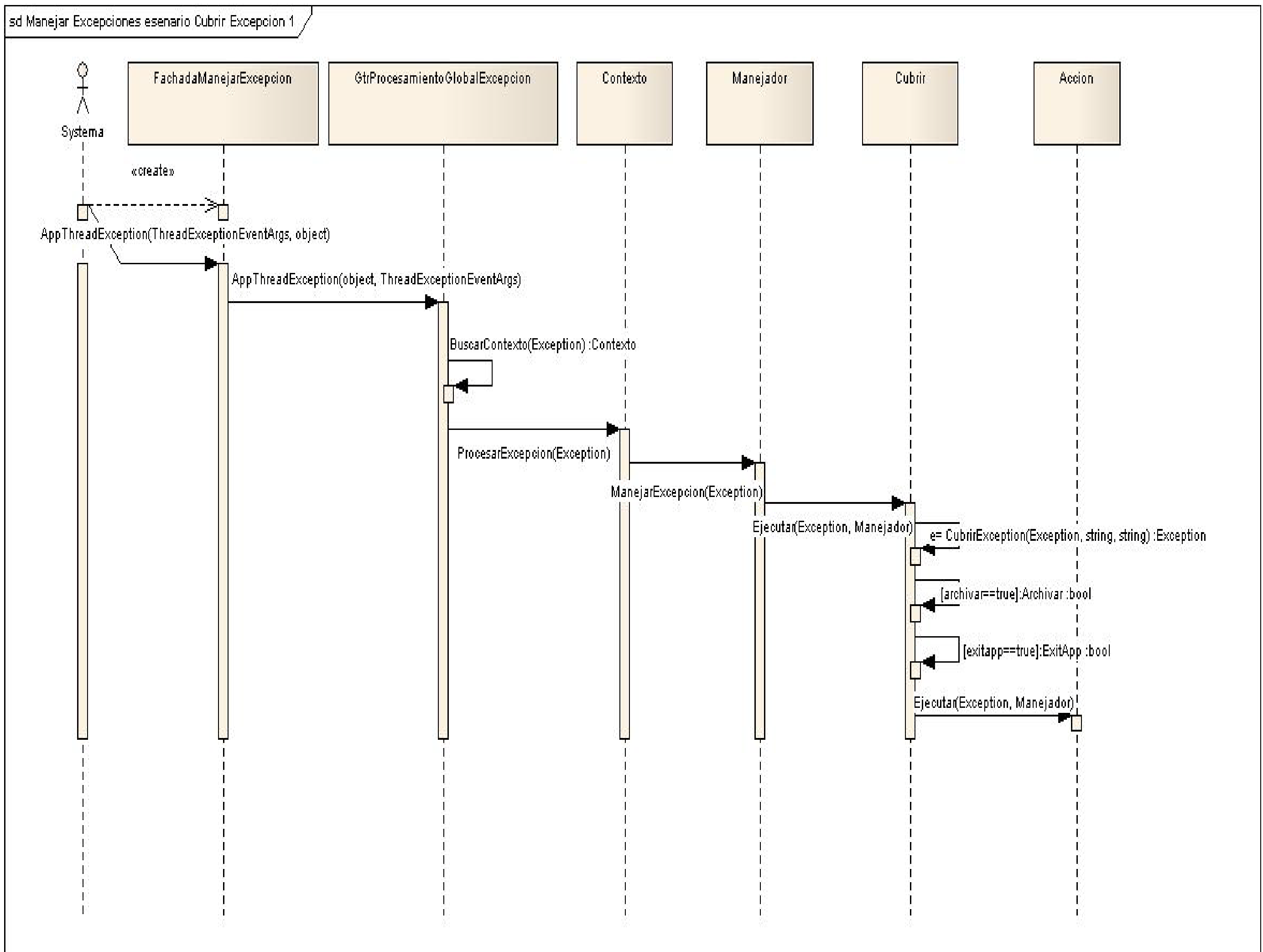


Figura 15 Diagrama de Secuencia “Manejar Excepción” escenario Cubrir Excepción.

- **Configurador Excepciones.**

Es iniciado por el Configurador de Contexto que puede ser cualquier persona perteneciente al quipo de desarrollo del sistema que usa el intérprete de excepciones. Consiste en configurar qué hacer con una excepción determinada, exportar el XML configurador para asociárselo al sistema en desarrollo o incluso cargar la Base de Datos de contextos con otros generados por otra Base de Datos o un fichero de configuración viejo. Contiene tres casos de usos, los cuales se muestran a continuación.

- **Gestionar XML Configurador:** Este caso de uso consiste en generar o guardar un XML en el que van a estar las configuraciones de los diferentes contextos identificados y almacenados previamente. En la Figura 16 se muestra el diagrama de clases y en la Figura 17 el diagrama de secuencia correspondiente al escenario Generar XML, el diagrama correspondiente a escenario Cargar XML se encuentra en el Anexo 6.

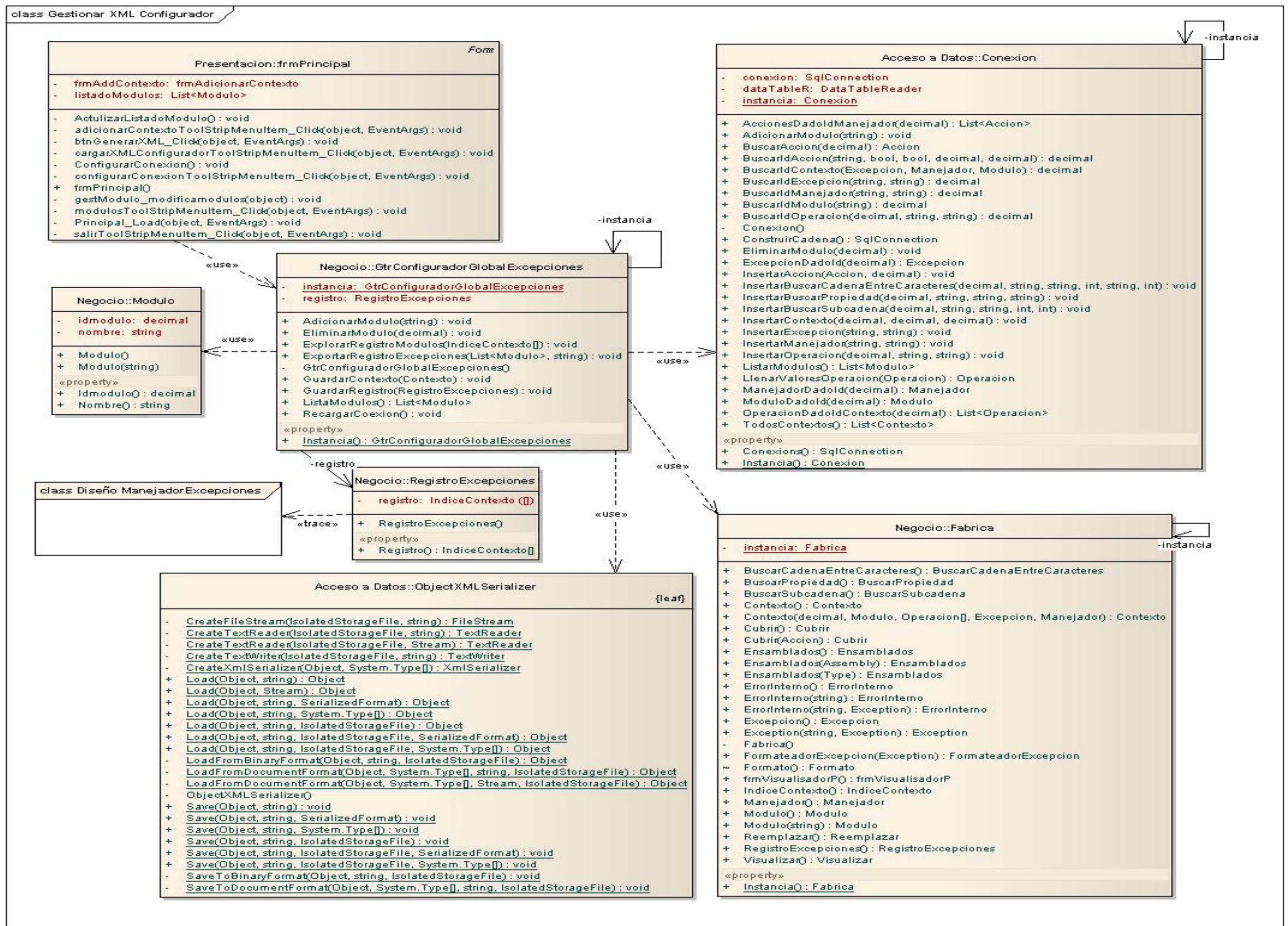


Figura 16 Diagrama de clases “Gestionar XML Configurador”.

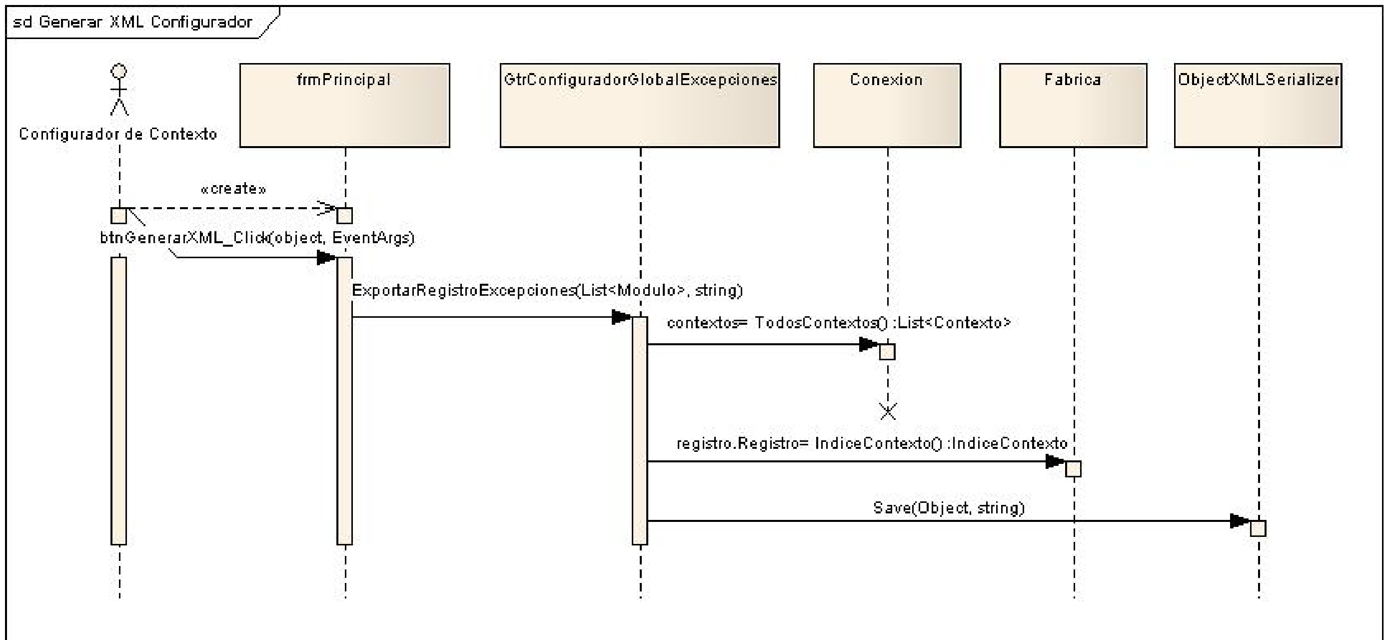


Figura 17 Diagrama de secuencia “Gestionar XML Configurador” escenario Generar XML.

- **Adicionar Contexto:** Este caso de uso se inicia cuando el actor Configurador de Contexto identifica una excepción que se desea manejar definiendo un contexto, los componentes que los conforma, excepción que lo produce, lista de operaciones para identificar la excepción y el manejador con sus acciones correspondientes a realizar. En la Figura 18 se muestra el diagrama de clases y en el Anexo 7 se muestra el diagrama se secuencia de la realización del caso de uso.

- **Gestionar Módulo:** Este caso de uso se inicia cuando el actor Configurador de Contexto identifica un nuevo módulo del sistema en el cual podrían ocurrir excepciones que se desean manejar y lo agrega al sistema configurador para luego poder adicionar nuevos contextos de excepciones asociados a ese módulo. O cuando identifica un módulo que se ha agregado y no se desea controlar los contextos asociados al mismo, entonces lo elimina. En la Figura 19 se muestra el diagrama de clases del caso de uso, en la Figura 20 el diagrama de secuencia de la realización del caso de uso escenario Eliminar Módulo y el diagrama de secuencia del escenario Adicionar Módulo se puede consultar en el Anexo 8.

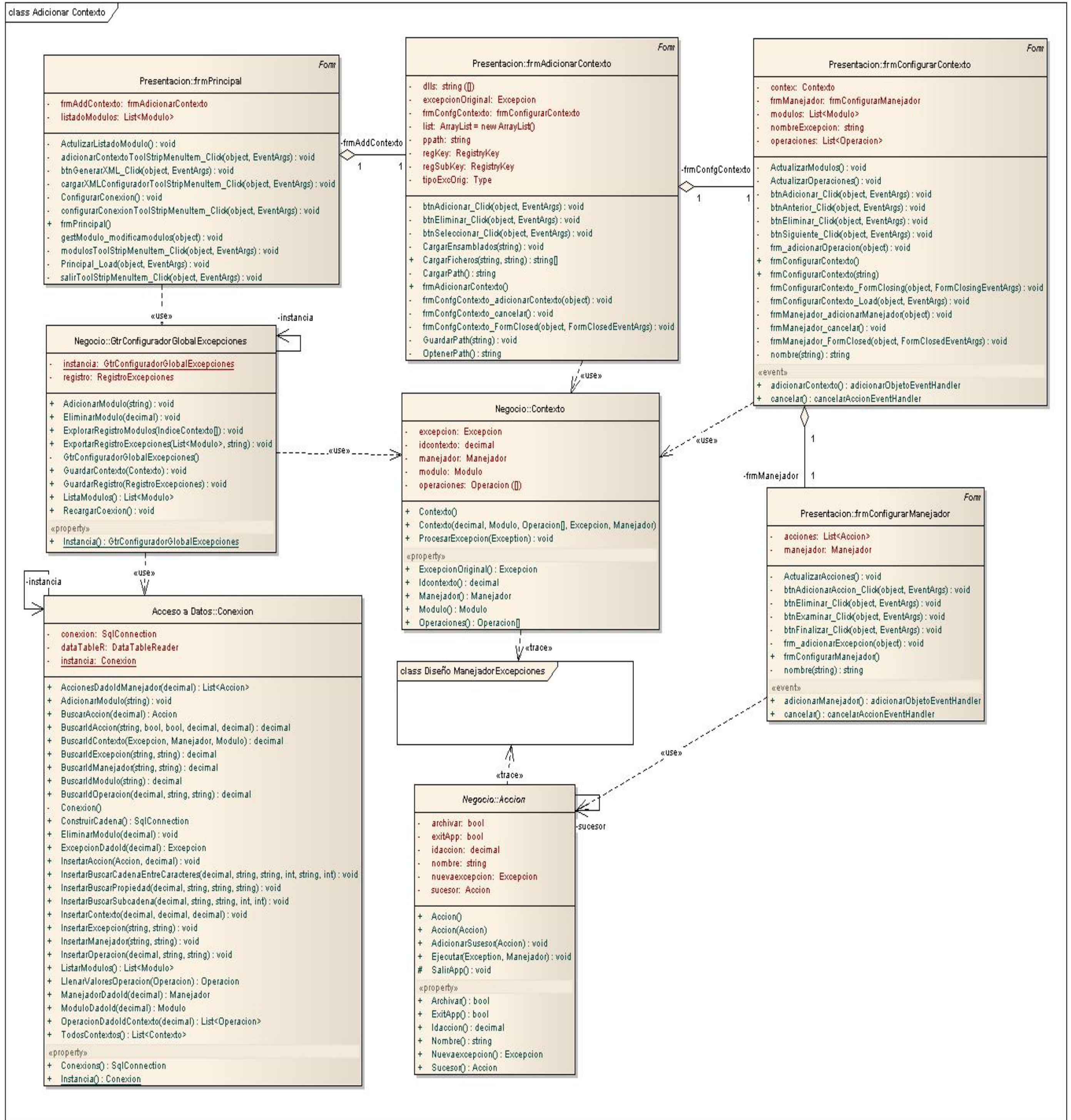


Figura 18 Diagrama de clases “Adicionar Contexto”.

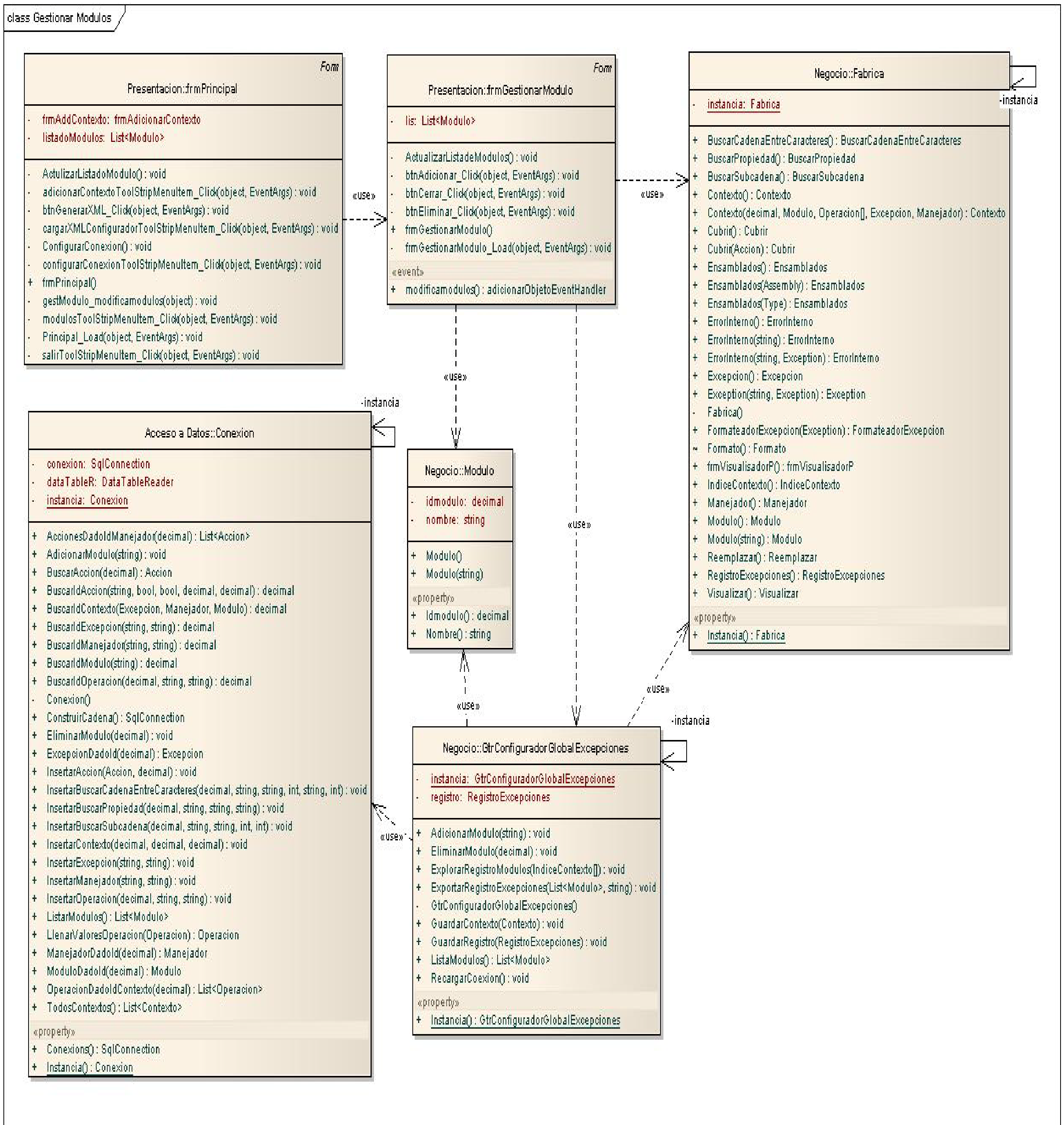


Figura 19 Diagrama de clases "Gestionar Módulo".

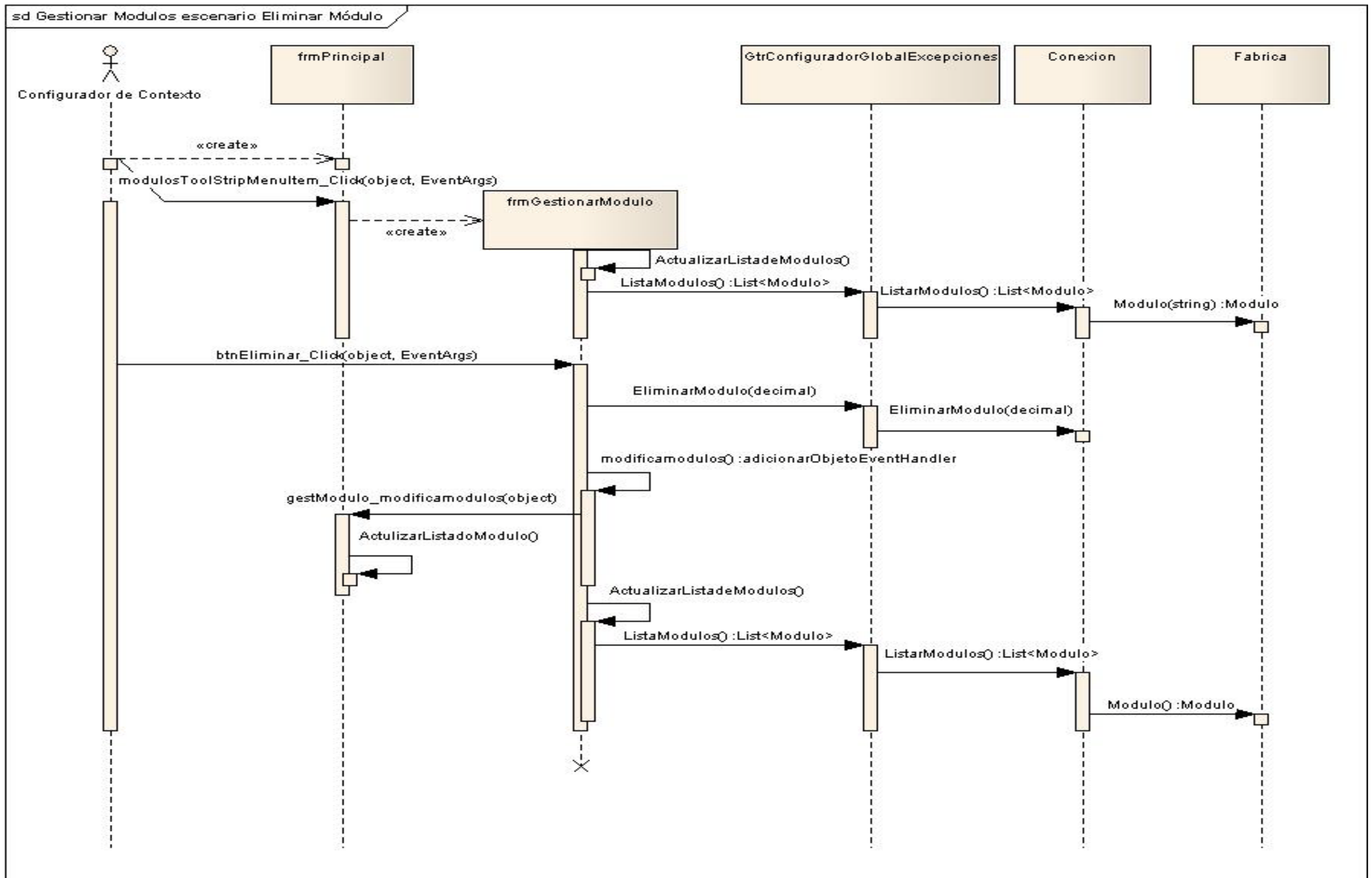


Figura 20 Diagrama de secuencia “Gestionar Módulo” escenario Eliminar Módulo.

2.3.4. Modelo de Datos

El modelo de datos describe la representación lógica y física de los datos persistentes usados por la aplicación. Es usado para describir la lógica y física de la información persistente manejada por el sistema. Puede ser inicialmente creado a través de Ingeniería Inversa de un almacenamiento de datos persistentes que ya exista (base de datos) o puede ser inicialmente creado a partir de un conjunto de clases del diseño persistentes en el modelo de diseño.

Se necesita dondequiera que el mecanismo de almacenamiento persistente sea basado en alguna tecnología no orientada a objetos. Específicamente se requiere donde la estructura de la información persistente no puede ser derivada automática y mecánicamente de la estructura de las clases persistentes

en el modelo de diseño. Es usado para definir el mapeo entre las clases persistentes del diseño y las estructuras de datos persistentes, así como para definir las estructuras de datos persistentes.

La solución propuesta no cuenta con persistencia en base de datos para el intérprete de excepciones. El módulo Configurador si cuenta con uso de persistencia en base datos, solucionando de esta forma los problemas que puedan surgir cuando en un equipo de desarrollo, varios programadores puedan ir adicionando las excepciones que quieren tratar y que van encontrando, permitiendo de esta forma tener una Base Datos de Contextos única para todo el equipo de desarrollo.

El modelo de datos desarrollado para el módulo configurador cuenta con nueve tablas, treinta y cinco atributos y nueve llaves. El la Figura 21 se muestra el modelo de datos obtenido.

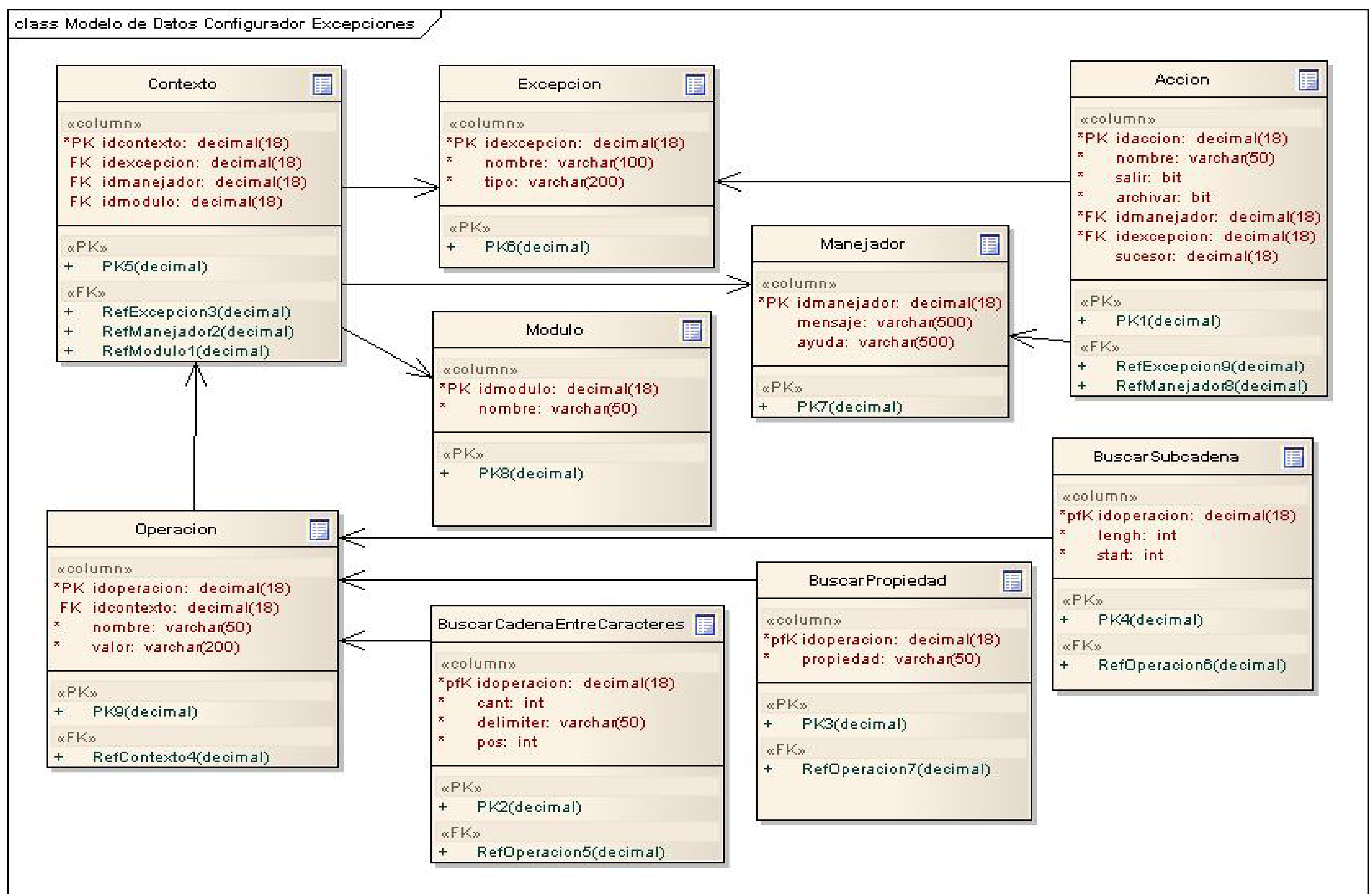


Figura 21 Modelo de datos del Módulo Configurador de Excepciones.

2.3.5. Modelo de despliegue.

El Diagrama de Despliegue constituye un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos (representan recursos de cómputo, normalmente un procesador o un dispositivo hardware similar). Se utiliza como entrada fundamental en las actividades de Diseño e Implementación ya que la distribución del sistema final ejerce una gran influencia en su diseño.

En el sistema propuesto; para el intérprete de excepciones no es necesario mostrar un diagrama de despliegue, al ser una librería de clases que servirá como herramienta para que los desarolladores manejen sus excepciones, estará ubicada en el ordenador en que se encuentre la aplicación que lo utilice como uno de sus componentes.

El módulo Configurador de Excepciones cuenta con una pequeña estructura de despliegue que se muestra a continuación en la Figura 22.

En el nodo PC se requiere para la utilización del módulo Configurador: *Windows XP* como sistema operativo, el *Framework .NET versión 2.0* instalado y en el servidor de la base de datos tiene que estar instalado el Sistema Gestor de Base Datos *Microsoft SQL Server 2000* o superior para montar el script de la base datos de Contextos.

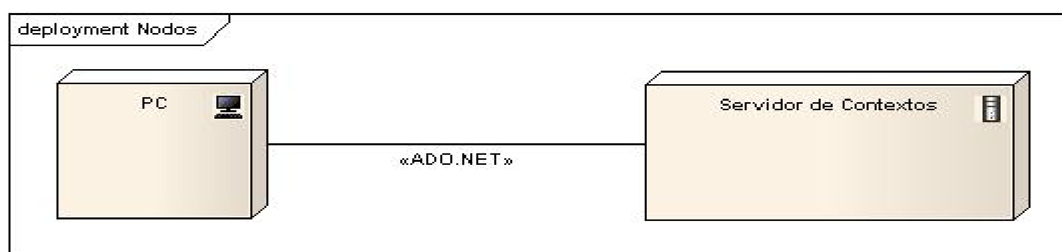


Figura 22 Diagrama de despliegue del módulo Configurador de Excepciones.

2.4. Estándar de Codificación.

Los Estándares de Codificación son reglas específicas a un lenguaje que reducen perceptiblemente el riesgo de que los desarrolladores introduzcan errores. Durante el desarrollo de un software, estos estándares ayudan a los ingenieros a producir un código de alta calidad y a entender y a utilizar el código

de sus colegas. Pero también realzan considerablemente la capacidad de mantenimiento y reusabilidad a largo plazo del producto final. (SYNSPACE AG, 2005)

2.4.1. Reglas a seguir

- Los nombres de las clases deben ser sustantivos o frases de sustantivos y no se deben usar prefijos.

- Para las clases interfaces, utilizar sustantivos, frases en sustantivos o adjetivos que describan comportamiento.

- Todas las clases deben tener prefijos que las identifiquen (como se muestran en la siguiente tabla) y seguidos de una letra mayúscula que sería la primera del nombre de la clase.

Clases	Prefijos
Formularios	Frm
Acciones	Acc
Interfaces	I
Entidades	E
Gestores	Gtr

Tabla 2 Prefijo de las clases.

- En los enumerados se recomienda no poner prefijos(sufijos) a los valores ni al identificador, el cual tiene que estar en singular.

- Para los campos de solo lectura y constantes se deben utilizar sustantivos o frases de éstos.

- Para los parámetros y campos no constantes se deben usar nombres descriptivos que deben ser lo suficientemente explícitos para determinar el significado de la variable y su tipo, preferentemente su significado.

- Utilizar i, j, k, l, m, n para contadores en ciclos triviales, en caso contrario usar nombres de variables más descriptivos.

- Los métodos deben ser nombrados con verbos o frases verbales.

- Las propiedades deben ser nombradas utilizando sustantivos o frases en sustantivo considerando siempre que sean nombradas con el mismo nombre de su tipo.

- Nombrar los manejadores de eventos con el sufijo EventHandler y las clases que sean para pasar argumentos con EventArgs. Denominar los eventos que utilizan el concepto de presente y pasado utilizando el tiempo en que ocurren y usar dos parámetros nombrados “sender” y “e”.

- No hacer público ninguna instancia o campo de una clase, deben ser privados.

- Excepto el código relacionado con la interfaz gráfica de la aplicación, todos los nombres de variables y campos que contengan elementos de interfaz como botones, etiquetas, etc., deben tener al principio la abreviatura del tipo como se muestra en la siguiente tabla:

Tipo	Abreviatura
System.Windows.Forms.Button	btn
System.Windows.Forms.TextBox	txt
System.Windows.Forms.ComboBox	cbx
System.Windows.Forms.ListBox	lbx
System.Windows.Forms.ListView	lvw
System.Windows.Forms.DateTimePicker	dtp
System.Windows.Forms.Label	lbl

Tabla 3 Prefijos de las clases de interfaz de Usuario.

- Todas las declaraciones se deben hacer en idioma Español para un mejor entendimiento y utilizando el estilo Pascal[17]; excepto los campos protegidos y privados, las variables locales y los parámetros, que deben ser en estilo Camello[18].

2.4.2. Pautas de Diseño de la Base de Datos

- **Modelo de Datos**

[17] Capitaliza la primera letra de cada palabra, ejemplo: AccProyecto.

[18] Capitaliza la primera letra de cada palabra, excepto para la primera palabra, ejemplo: objProyecto.

Modelo lógico se utiliza la notación de camello.

Modelo físico se eleva todo a mayúsculas para obtener un estándar.

- **Tablas, consultas o vistas**

Seguir las siguientes reglas en el momento de escoger el nombre:

Utilizar sustantivos o frases en sustantivo en singular.

Utilizar prefijos:

  - d: para las tablas de datos.
  - n: para las tablas de nomencladores.
  - v: para las vistas.

Utilizar estilo de Pascal.

- **Campos**

Para los identificadores tener en cuenta:

Utilizar sustantivos o frases en sustantivo en singular.

En minúsculas el nombre completo.

- **Generales**

Todos los identificadores en idioma español.

No utilizar underscore, solo en tablas temporales, nombres de las llaves primarias, foráneas, nombre de los tablespace y objetos secuence.

Definir tipos de datos, identificador en minúsculas completo utilizando el prefijo ‘t’.

Para todos los identificadores una longitud menor de 30 caracteres.

Identificador para cada tabla.

Prefijo ‘id’ mas el nombre de la tabla.

Llaves subrogadas para todas las tablas.

Prefijo ‘pk’ mas nombre completo del campo (identificador), utilizando el estilo de Pascal.

Campos únicos y/o llaves obvias se especifican como llaves alternas.

Prefijo ‘ak’ mas nombre completo del campo, utilizando el estilo de Pascal.

Descripción para cada tabla y para cada campo de cada tabla.

Especificar índices para las llaves primarias y para los campos por los cuales se van a realizar búsquedas.

Prefijo ‘idx_’ más nombre completo del campo, utilizando el estilo de Pascal.

Nombre descriptivo para todas las relaciones.

Nombres de los tablespaces y secuencias en mayúsculas el identificador completo. Utilizar sustantivos.

Prefijos ‘S’ para las secuencias y ‘TS’ para los tablespaces.

2.5. Implementación del sistema.

Según RUP, la Implementación es el flujo de trabajo de más peso durante la Fase de Construcción, aunque también se lleva a cabo durante la Fase de Elaboración para crear la línea base ejecutable de la arquitectura y durante la de Transición para tratar defectos que se hayan encontrado en ese momento, en caso de que existan.

El resultado del Diseño constituye la entrada fundamental de la Implementación, comenzándose a implementar el sistema en términos de componentes, es decir, ficheros de código fuente o binario, scripts, ejecutables o similares. La mayor parte de la arquitectura es capturada durante el Diseño, siendo el propósito fundamental de la Implementación desarrollar la arquitectura y el sistema como un todo. Aquí se implementan las clases elaboradas durante el Diseño como componentes de fichero que contienen código fuente.

2.5.1. Componente desarrollado.

Se contó con un componente desarrollado para la configuración de la conexión genérica a una instancia de base dato Microsoft SQL Server el cual se integró con éxito al módulo Configurador. El componente que se llama ConfiguradorConexion, el cual brinda un UserControl[19] o control de usuario de Visual Studio 2005 con todas las opciones necesarias para configurar una nueva instancia de conexión a un servidor Microsoft SQL Server y la guarda en el registro de Windows del ordenador. Además tiene una fachada que tiene las funcionalidades necesarias para la recuperar los datos de conexión guardados en el registro.

2.5.2. Diagrama de Componentes

El uso más importante que tiene este tipo de diagrama es mostrar la estructura de alto nivel del Modelo de Implementación, específicamente los Subsistemas de Implementación y sus dependencias de importación, además, que éstos estén organizados en capas.

También se usan para mostrar ficheros de código fuente y sus dependencias de compilación; ficheros de aplicación y sus dependencias en tiempo de ejecución; relaciones derivadas entre ficheros de código fuente y ficheros resultantes de la compilación; dependencias de implementación entre Elementos de Implementación y Elementos de Diseño que ellos implementan. (Rational Software Corporation, 2003)

En las Figura 23 y Figura 24, se muestran los diagramas de componente del intérprete de excepciones y el módulo Configurador respectivamente.

[19] Un control de usuario es similar a cualquier otra clase, pero con la posibilidad agregada de poder colocarlo en el Cuadro de herramientas y mostrarlo en un formulario. Donde un módulo de clase tiene sólo código, un módulo de control de usuario tiene código y un diseñador.

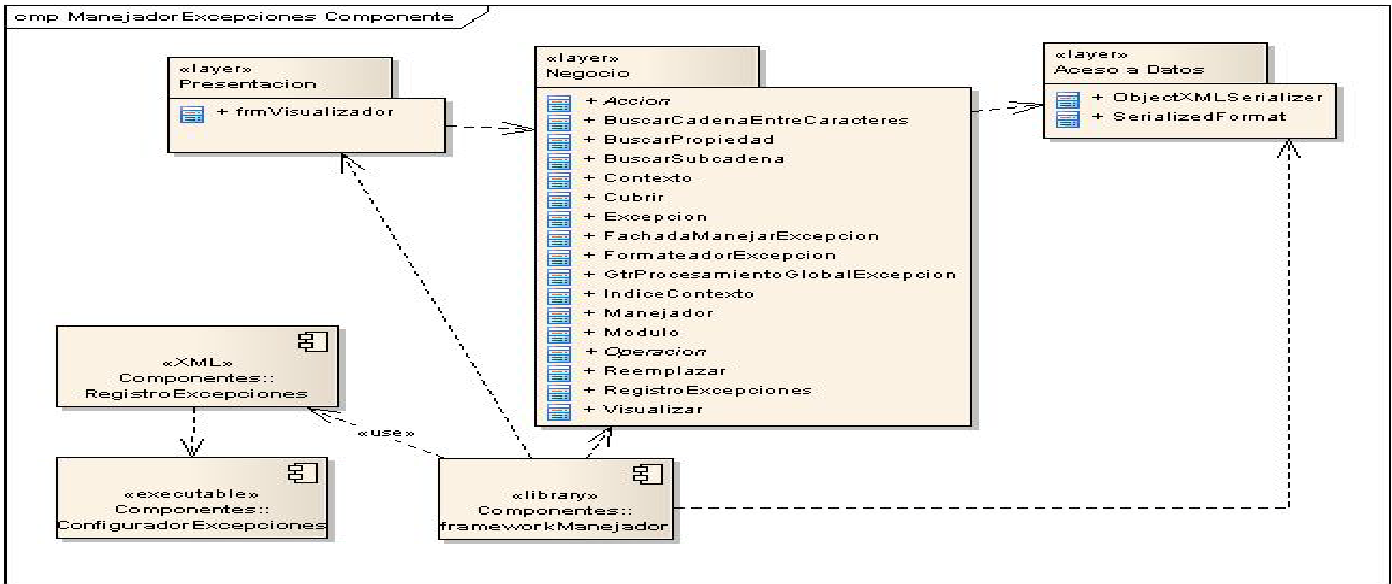


Figura 23 Diagrama de Componente del intérprete de excepciones.

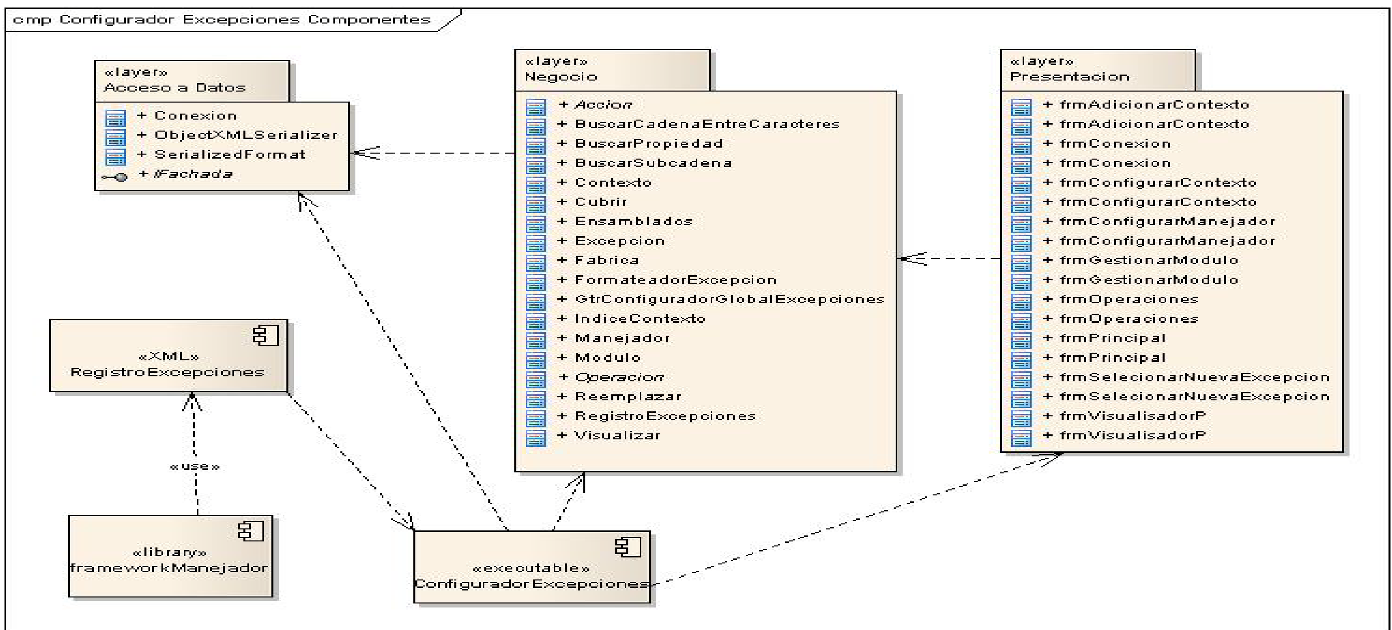


Figura 24 Diagrama de Componente del Módulo Configurador de Excepciones.

2.6. Conclusiones

A lo largo del capítulo se abordaron temas de interés correspondientes a las etapas de diseño e implementación de un intérprete de excepciones y un módulo asociado para dar configuración a las

excepciones que se desean manejar. Ilustrado el proceso con los principales artefactos que propone la metodología RUP para estos flujos de trabajo.

Se expusieron los Diagramas de Interacción realizados para los casos de uso: Manejar Excepción, Gestionar XML Configurador, Adicionar Contexto y Gestionar Módulos. El análisis de estos diagramas arrojó como resultado el Modelo de Clases del Diseño compuesto por las clases de diseño y sus relaciones. El Modelo de Datos del módulo Configurador también quedó definido como sostén a la Base de Datos del mismo.

La implementación del sistema terminando el diseño resultó en el Modelo de Implementación del sistema y como colofón del capítulo se presentó el Modelo del Despliegue.

Se abordaron temas de interés para la mejor comprensión de la base de los sistemas. Como resultado quedó definida la arquitectura de los mismos, sin obviar las diferentes capas por las que está compuesta.

Los objetivos planteados para la realización del presente capítulo han sido cumplidos, ya que resulta en el diseño e implementación de un intérprete de excepciones para sistemas con arquitectura .NET y un módulo complementario para configurar las excepciones que se desean manejar.

CAPÍTULO 3: ANÁLISIS DE RESULTADOS.

En el siguiente capítulo se abordarán las diferentes técnicas de validación, pruebas y sus resultados que se aplicaron al intérprete y el módulo configurador.

En el siguiente capítulo con el objetivo de encontrar errores durante el proceso de desarrollo y ejecución del programa, se realizan una serie de pruebas al sistema informático que se propone., las mismas se efectúan en circunstancias previamente especificadas, cuyos resultados se observan y se registran, con el propósito de obtener una evaluación de diferentes aspectos. Además se aplican diferentes técnicas de validación a la solución propuesta, comprobando de esta forma la calidad y eficiencia del resultado alcanzado.

3.1. Métricas Aplicadas

Con el objetivo de determinar el grado de calidad y fiabilidad del diseño propuesto se aplican algunas métricas de diseño basadas en clases, para medir categorías tales como tamaño, herencia, valores internos y valores externos, para aspectos orientados al código, a la cohesión, al acoplamiento y la reutilización.

3.1.1. Métrica Tamaño de clase (TC)

El tamaño general de una clase se puede determinar empleando medidas para saber el **número total de operaciones** (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase, así como encontrando el **número de atributos** (tanto atributos heredados como atributos privados de la Instancia) que están encapsulados en la clase. Si existen valores grandes de TC, éstos mostrarán que una clase puede tener demasiada responsabilidad, lo cual reducirá la reusabilidad de la clase y complicará la implementación y la comprobación, por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente.

La mayor cantidad de clases que involucran procesos presentes en el intérprete y en el módulo Configurador se encuentran en la Capa de Negocios, fue a esta capa a la que se le aplicó la métrica del Tamaño de Clase (TC). Para el análisis de las clases del Intérprete Manejador se elaboro la Tabla 4

Clases de la capa de Negocio Intérprete y para el análisis del módulo Configurador la Tabla 5 Clases de la capa de Negocio Módulo Configurador..

	Intérprete			
	Clase	Nro. Atributos	Nro. Operaciones	Total
1	Accion	6	3	9
2	Cubrir	6	4	10
3	Reemplazar	6	4	10
4	Visualizar	10	10	20
5	Operacion	3	1	4
6	BuscarCadenaEntreCaracteres	6	1	7
7	BuscarPropiedad	4	1	5
8	BuscarSubcadena	5	1	6
9	Contexto	5	1	6
10	Excepcion	3	0	3
11	IndiceContexto	2	0	2
12	Manejador	5	1	6
13	Módulo	2	0	2
14	grtProcesamientoGlobalExcepcion	2	4	6
15	RegistroExcepciones	1	0	1
	Promedio	4,4	2,066666667	

Tabla 4 Clases de la capa de Negocio Intérprete.

	Módulo Configurador			
	Clase	Nro. Atributos	Nro. Operaciones	Total
1	Accion	6	3	9
2	Cubrir	6	4	10
3	Reemplazar	6	4	10
4	Visualizar	10	10	20
5	Operacion	3	1	4
6	BuscarCadenaEntreCaracteres	6	1	7
7	BuscarPropiedad	4	1	5
8	BuscarSubcadena	5	1	6
9	grtConfiguradorGlobalExcepciones	1	8	9
10	Contexto	5	1	6
11	Ensamblados	2	0	2
12	Excepcion	3	0	3
13	Fabrica	1	25	26
14	FormateadorExcepcion	2	11	13
15	IndiceContexto	2	0	2
16	Manejador	5	1	6
17	Módulo	2	0	2
18	RegistroExcepciones	1	0	1
	Promedio	3,888888889	3,944444444	

Tabla 5 Clases de la capa de Negocio Módulo Configurador.

Resultados Obtenidos

Teniendo en cuenta las medidas o umbrales de referencia en lo que respecta al número de operaciones y/o atributos de las clases, se establece que un tamaño de clase pequeño es aquel que tiene un valor menor o igual que 20. Un tamaño de clase medio es aquel cuyos valores exceden a 20 y son menores o incluyen a 30 y un tamaño de clase grande es aquel que es mayor que este último valor (30). Así se concluye que la Capa de Negocio del intérprete tiene 15 clases, para un promedio de 4,4 atributos

y 2,07 cantidades de operaciones. Y la Capa de Negocio del módulo Configurador tiene 18 clases, para un promedio de 3,89 atributos y 3, 94 cantidad de operaciones. Los Valores de Tamaño quedan distribuidos de la siguiente manera reflejados en la Tabla 6 Resultados métrica Tamaño de Clases..

Umbral	Tamaño	Cantidad de Clases intérprete	Cantidad de Clases configurador
<=20	Pequeño	15	17
> 20 y <= 30	Mediano	0	1
> 30	Grande	0	0
Total de clases		15	18

Tabla 6 Resultados métrica Tamaño de Clases.

Finalmente quedando ilustrado los resultados en el Gráfico 1 Resultado del Intérprete y Gráfico 2 Resultado del módulo Configurador.

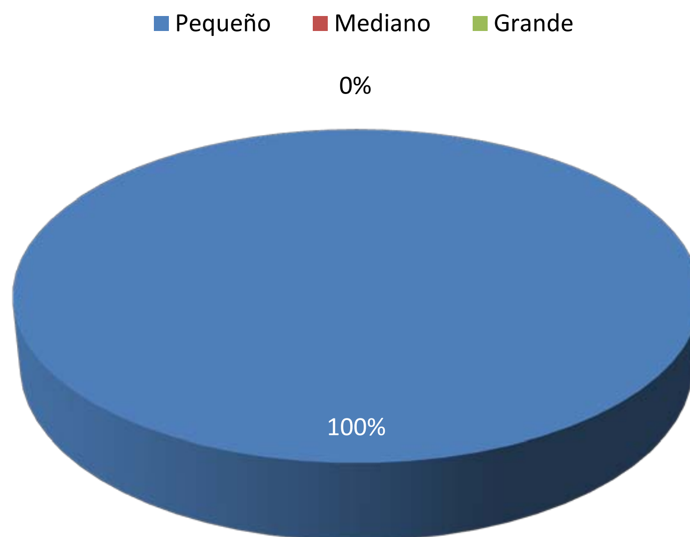


Gráfico 1 Resultado del Intérprete

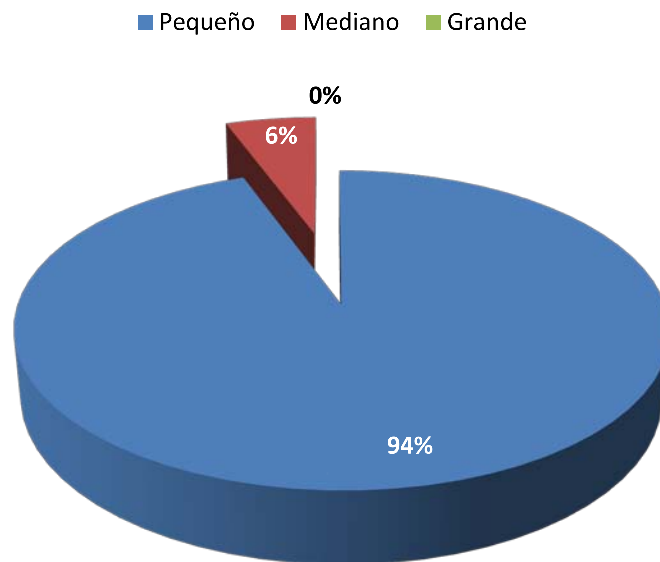


Gráfico 2 Resultado del módulo Configurador

La Tabla 6, los Gráfico 1 y Gráfico 2 respectivamente muestran que la mayoría de las clases son pequeñas por lo que se cumplen los parámetros establecidos por la métrica de tamaño de clase.

3.1.2. Árbol de Profundidad de Herencia

Esta métrica se define como la longitud máxima desde el nodo hasta la raíz del árbol (ver Figura 25) (Pressman, 2002), donde el valor de APH para la jerarquía de clases mostrada es 3. A medida que crece el APH, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (con un valor grande de APH) lleva también a una mayor complejidad de diseño. Por el lado positivo, los valores grandes de APH implican que se pueden reutilizar muchos métodos.

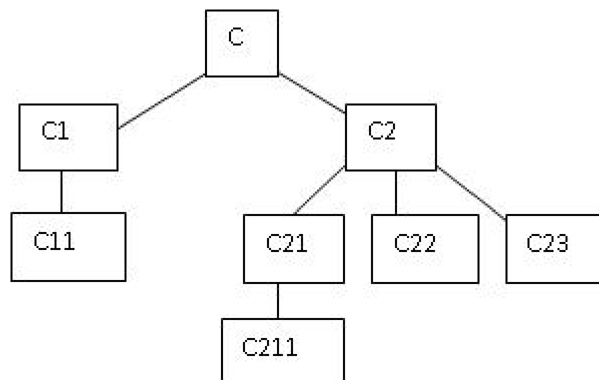


Figura 25 Una jerarquía de Clases.

Resultados Obtenidos

El Intérprete y el módulo Configurador presentan como máximo nivel de descendencia 1, obteniéndose un nivel de reutilización bajo, pero sin verse afectada la abstracción representada por la clase predecesora, quedando el diseño entre los umbrales de calidad.

3.1.3. Métrica de la Complejidad Ciclomática.

La Complejidad Ciclomática es una métrica que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cuando se usa en el contexto de la técnica del Camino Básico, este valor calculado como Complejidad Ciclomática indica el número de "caminos independientes[20]" del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuten al menos una vez cada sentencia.

Esta métrica es útil para predecir los módulos que son más propensos a error y puede ser usada para planificar pruebas así como para diseñar casos de prueba.

La complejidad se puede calcular de tres formas diferentes:

- El número de regiones del grafo de flujo es igual a la Complejidad Ciclomática.

[20] Cualquier camino del programa que introduce al menos un nuevo conjunto de sentencias o una nueva condición. En términos de grafo de flujo, está constituido por lo menos por una arista que no haya sido recorrida anteriormente a la definición del camino.

- Se define a la Complejidad Ciclomática V(G), de un grafo G, como:

  $$V(G) = A - N + 2$$

  Donde A es el número de aristas y N el número de nodos del grafo de flujo.

- También se define a la Complejidad ciclomática V(G), de un grafo G, como:

  $$V(G) = P + 1$$

  Donde P es el número de nodos predicado del grafo de flujo.

Se realizó el cálculo de la Complejidad Ciclomática en todos los bloques de código dentro del sistema, lo que ayudó a reflejar una medida de la complejidad del código escrito. A continuación se muestran algunos fragmentos de código junto a sus grafos de flujo correspondientes y el cálculo de la complejidad ciclomática.

I. Este grafo representa un método que adiciona un nuevo módulo a la Base de Datos del subsistema Configurador de Excepciones. El código correspondiente se muestra en el Anexo 9

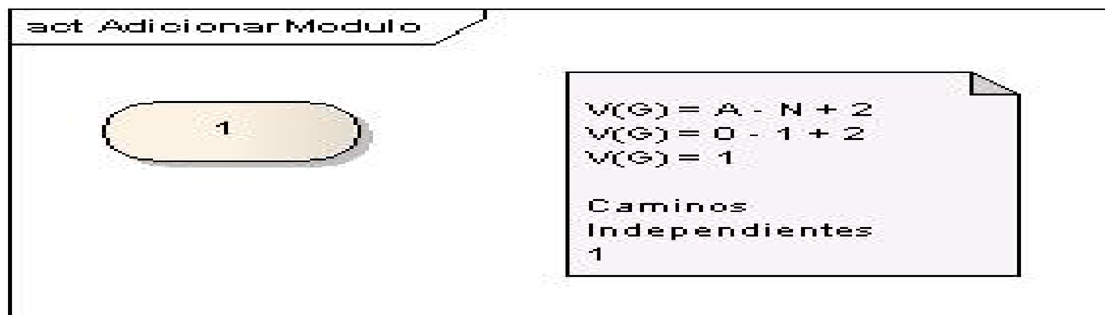


Figura 26 Grafo método Adicionar Módulo.

II. Este grafo representa al método AppThreadException de la clase gestora del intérprete, el mismo se encarga de identificar el contexto en que ocurrió la excepción y ejecutar las acciones correspondientes. El código correspondiente se encuentra en el Anexo 10.

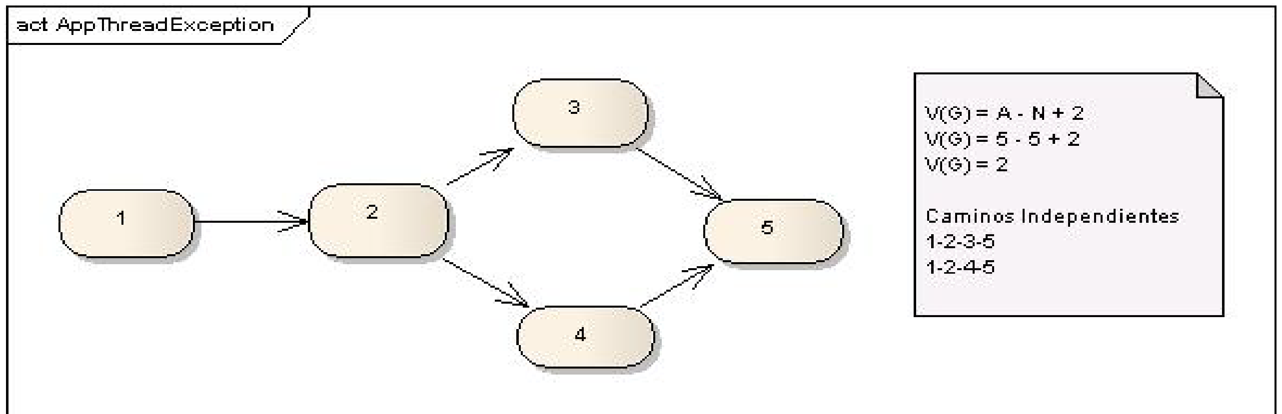


Figura 27 Grafo método AppThreadException.

III. Este grafo representa al evento que se lanza al oprimir el botón Generar XML del la interfaz principal del módulo Configurador, para generar un XML con la configuración de los contextos identificados y guardados en la base de datos. El código correspondiente se encuentra en el anexo 11.

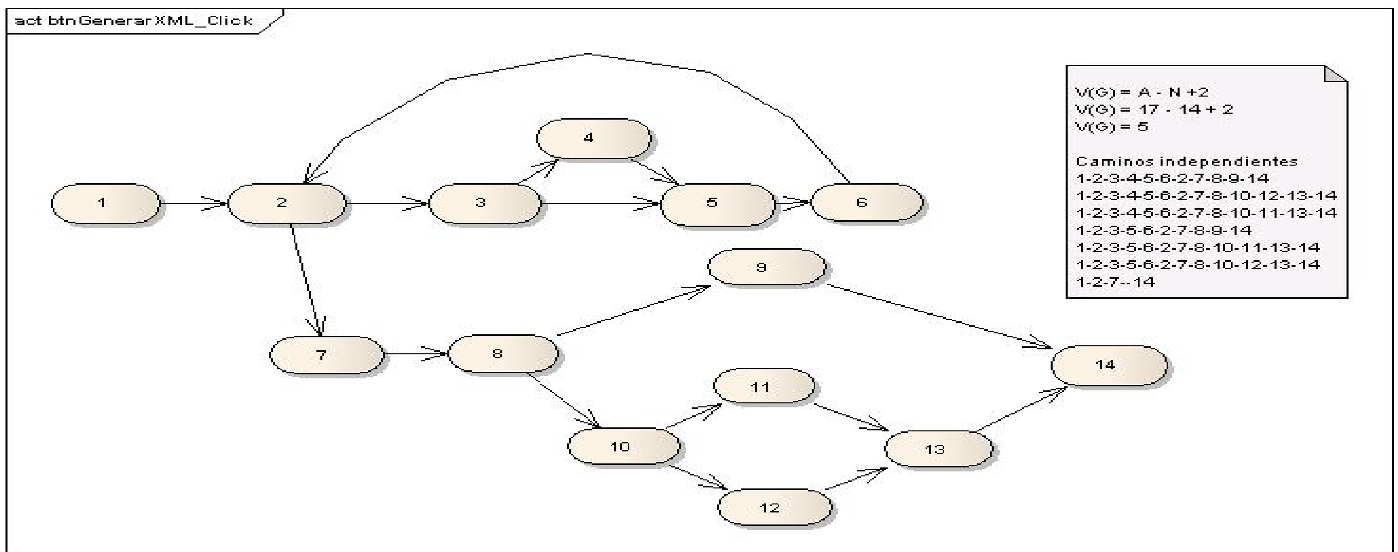


Figura 28 Grafo método btnGenerarXML_Click

IV. Este grafo representa al evento que se lanza al oprimir en el menú principal la opción Cargar XML Configurador del la interfaz principal del módulo Configurador; para realizar la operación con el

mismo nombre, teniendo guardado con anterioridad la configuración de los contextos en la base de datos. El código correspondiente se encuentra en el anexo 12.

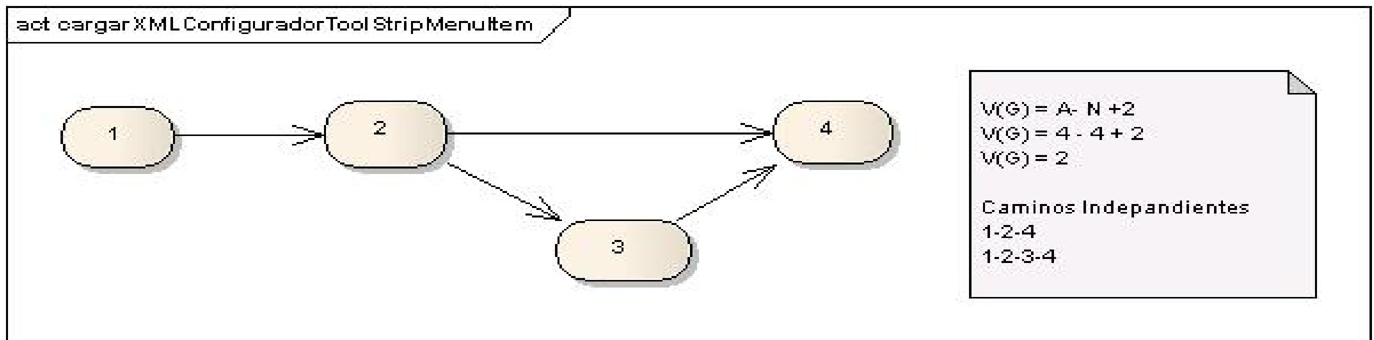


Figura 29 Grafo método cargarXMLConfiguradorToolStripMenuItem.

V. Este grafo representa el método Ejecutar de la clase Cubrir que es invocado cuando se identifica un contexto que tiene entre sus acciones a realizar la de cubrir la excepción. Lo que realiza es cambiar la excepción original por una nueva y la original queda como excepción interna de esta nueva. El código correspondiente se encuentra en el anexo 13.

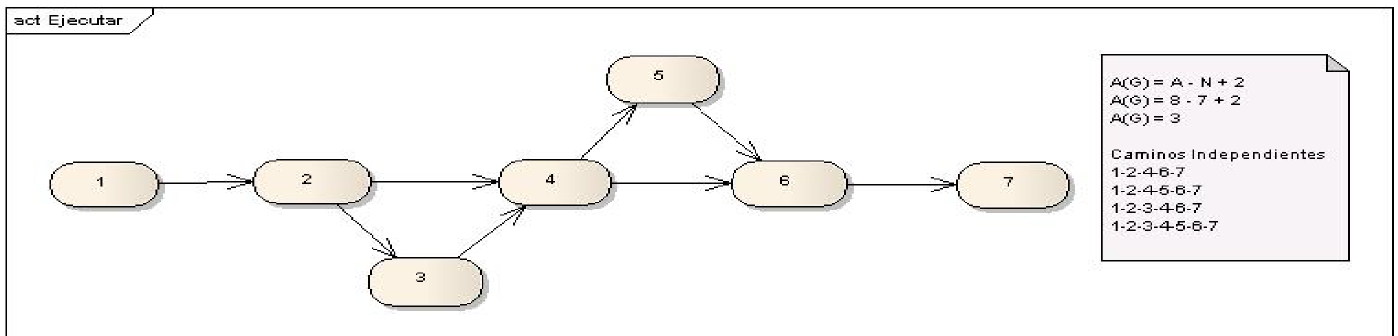


Figura 30 Grafo método Ejecutar.

VI. Este grafo representa al método Guardar Contexto de la clase Gestora de la Capa de Negocio, que salva un contexto identificado. El código correspondiente se encuentra en el anexo 14.

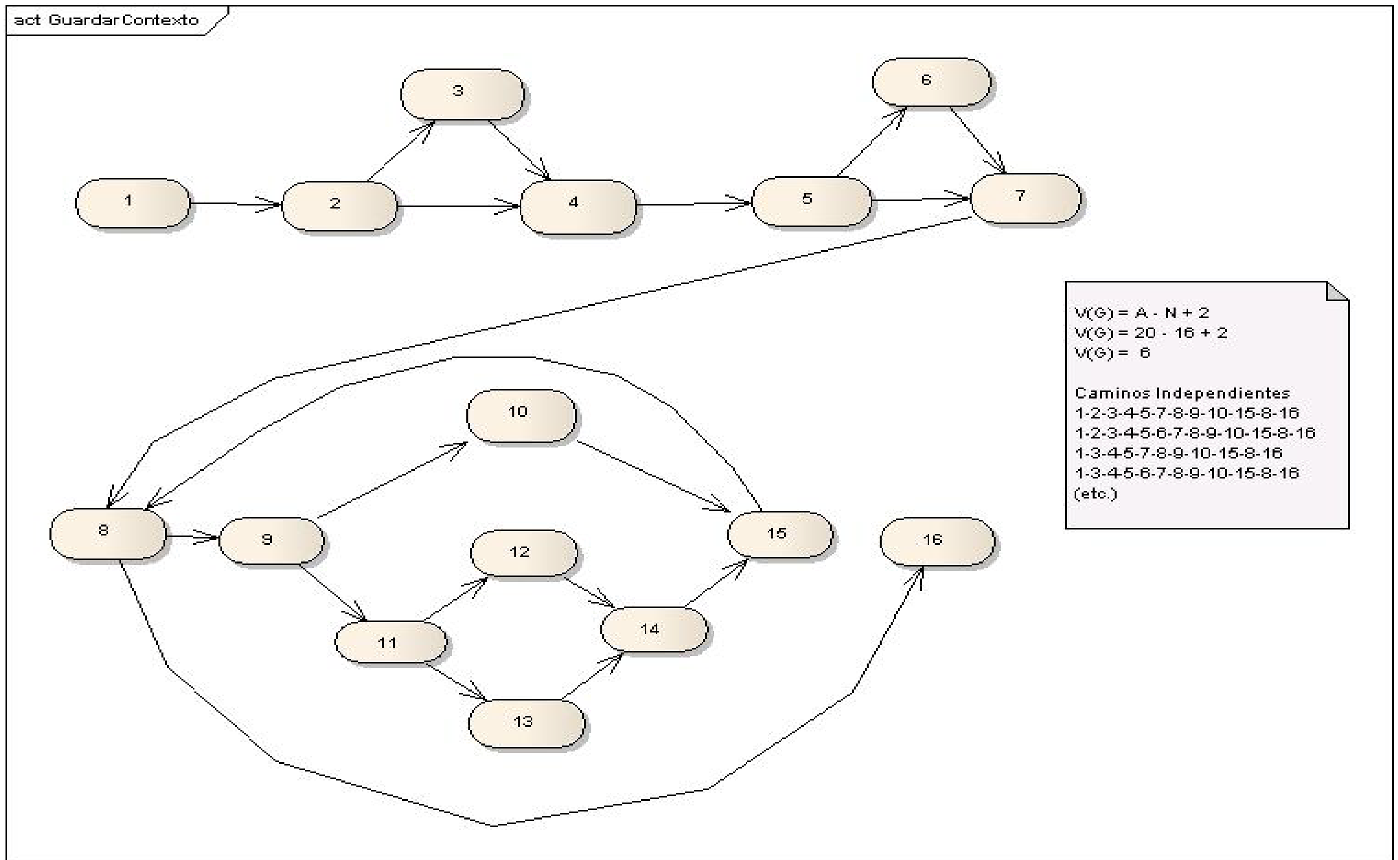


Figura 31 Grafo método GuardarContexto

Se han medido un gran número de programas a modo de establecer rangos de complejidad que ayuden al Ingeniero de Software a determinar la estabilidad y riesgo de un programa. La medida resultante puede ser utilizada en el desarrollo, mantenimiento y reingeniería para estimar el riesgo, costo y estabilidad del sistema. Algunos estudios experimentales indican la existencia de distintas relaciones entre esta métrica y el número de errores existentes en el código fuente, así como el tiempo requerido para encontrar y corregir esos errores. Se suele comparar la Complejidad Ciclomática obtenida contra un conjunto de valores límites como se observa en la siguiente tabla:

Complejidad Ciclomática	Evaluación del Riesgo
1 - 10	Programa Simple sin mucho riesgo
11 – 20	Más complejo, riesgo moderado
21 – 50	Complejo, programa de alto riesgo
50 +	Programa no testeable, muy alto riesgo

Tabla 7 Complejidad Ciclomática vs Evaluación del Riesgo

(Pressman, 2002) (Rizzi)

El cálculo de complejidad más alto que presenta esta propuesta de solución es de 6, el cual se encuentra en el rango de 1 - 10 por lo que se puede llegar a la conclusión de que se cuenta con un programa simple y sin mucho riesgo. Se demostró que las funciones del software son operativas, que las entradas se aceptan de forma adecuada produciendo una salida correcta, manteniendo la integridad de la información externa.

3.2. Pruebas de unidad.

3.2.1. Pruebas de Caja Negra.

Una vez identificadas las clases válidas e inválidas, se definieron los Casos de Prueba, los cuales se encuentran organizados en cuatro unidades diferentes: Gestionar Módulo, Gestionar XML, Adicionar Contexto y Manejar Excepción.

3.2.1.1. Gestionar Módulo

A este caso de uso se le realizaron las pruebas de adicionar y eliminar módulo.

CPR1. Adicionar Módulo.

Flujo Central.

1. El configurador de contexto identifica un nuevo módulo
2. El sistema muestra la interfaz principal.
3. El configurador de contexto ordena gestionar módulo mediante el menú con el mismo nombre.
4. El sistema muestra la interfaz Gestionar Módulo.
5. El configurador de contexto escribe el nombre del nuevo módulo.
6. El configurador de contexto pulsa en el botón ”+”.
7. El sistema adiciona un módulo con el nombre seleccionado.

8. El configurador de contexto cierra la ventana presionando el botón Cerrar.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El botón + está activado		El sistema permite adicionar	Satisfactorio	
El campo nombre permite cualquier cadena		El sistema permite todo tipo de caracteres en el campo nombre	Satisfactorio	
	El campo nombre está en blanco	El sistema no permite adicionar un nombre en blanco	Satisfactorio	
La interfaz Gestionar Módulo se cierra al oprimir el botón Cerrar		El sistema cierra la interfaz	Satisfactorio	

Tabla 8 Caso de prueba 1 Adicionar Módulo.

CPR2. Eliminar Módulo.

Flujo Central.

1. El configurador de contexto identifica un módulo a eliminar.
2. El sistema muestra la interfaz principal.
3. El configurador de contexto ordena gestionar módulo mediante el menú con el mismo nombre.
4. El sistema muestra la interfaz Gestionar Módulo con la lista de módulos.
5. El configurador de contexto selecciona un nombre en la lista de módulos.
6. El configurador de contexto oprime el botón “-”.
7. El sistema elimina el módulo seleccionado.
8. El configurador de contexto cierra la ventana por el botón Cerrar.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El botón Eliminar (-) permanece está habilitado.		Seleccionar el módulo en la lista antes de oprimir el botón eliminar	Satisfactorio	
Al oprimir el botón "- " el sistema pide confirmación		El sistema muestra mensaje de confirmación y elimina el módulo	Satisfactorio	
	No está seleccionado un módulo para eliminar	El sistema no permite eliminar módulo y se muestra una alerta de error	Satisfactorio	

Tabla 9 Caso de prueba 2 Eliminar módulo.

3.2.1.2. Gestionar XML

A este caso de uso se le realizaron las pruebas de Cargar XML y Generar XML.

CPR1. Cargar XML

Flujo Central.

1. El configurador de contexto tiene un XML configurador generado mediante otra base de datos y desea adicionar esos contextos a la de él.
2. El sistema muestra la interfaz principal.
3. El configurador de contexto ordena Cargar XML configurador mediante la opción en el menú principal con el mismo nombre.
4. El sistema muestra una interfaz para buscar y seleccionar el XML que se desea cargar.
5. El configurador de contexto selecciona el XML.
6. El sistema adiciona los contextos nuevos.
7. El sistema muestra un mensaje de información confirmando que la operación se realizó con éxito.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El usuario selecciona la opción Carga XML del menú principal.		Se muestra una interfaz para seleccionar el XML deseado.	Satisfactorio	
El usuario selecciona el XML y presiona el botón aceptar		El sistema adiciona los nuevos contextos a la Base de Datos y muestra un mensaje de confirmación de la operación.	Satisfactorio	
El usuario selecciona el XML y presiona el botón Cancelar		El sistema no carga los contextos	Satisfactorio	
	El usuario selecciona un XML con formato incorrecto	El sistema arroja un error informando que el formato no está correcto	Satisfactorio	

Tabla 10 Caso de prueba 1 Cargar XML.

CPR2. Generar XML.

Flujo Central.

1. El configurador de contexto desea generar un XML configurador con los contextos identificados.
2. El sistema muestra la interfaz principal con el listado de módulos.
3. El configurador de contexto selecciona el o los módulos a los que desea generarle el XML.
4. El configurador de contexto ordena Guardar XML configurador mediante el botón en el con el mismo nombre en la interfaz principal.
5. El sistema muestra una interfaz para seleccionar el nombre y el destino de XML que va a generar.
6. El configurador de contexto selecciona el nombre y el destino y presiona el botón aceptar.
7. El sistema genera el XML y muestra un mensaje de confirmación de generación.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El sistema muestra la lista de módulos disponibles		El sistema muestra el listado de módulos disponibles al cargar la interfaz principal	Satisfactorio	
	El usuario no selecciona ningún módulo a generar y presiona el botón Generar XML	El sistema muestra una alerta de error al lado del botón con un mensaje que debe seleccionar un módulo.	Satisfactorio	
El usuario selecciona el o los módulos con los que desea generar el XML y presiona el botón Generar XML		El sistema muestra una interfaz para que el usuario seleccione el nombre y el destino del XML	Satisfactorio	
El usuario selecciona el nombre y el destino del XML y presiona el botón Aceptar		El sistema genera el XML con el nombre y el destino que usuario selecciono	Satisfactorio	

Tabla 11 Caso de prueba 2 Generar XML

3.2.1.3. Adicionar Contexto

A este caso de uso se le aplicaron dos casos de pruebas Adicionar Contexto con una operación y una acción, adicionar contexto con múltiples operaciones y múltiples acciones.

CPR1. Adicionar contexto con una operación y una acción.

Flujo Central

1. El configurador de contexto identifica un nuevo contexto que desea adicionar.
2. El sistema muestra la interfaz principal.
3. El configurador de contexto ordena Adicionar Contexto o Excepción en el menú principal de la interfaz principal.

4. El sistema muestra la interfaz Seleccionar Excepción con la lista de las excepciones que se encuentran en los ensamblados del proyecto.
5. El configurador de contexto selecciona la excepción que origina el contexto y presiona el botón Siguiente.
6. El sistema muestra la interfaz Configurar Manejador.
7. El configurador de contexto selecciona el módulo al que pertenece el contexto.
8. El configurador de contexto presiona el botón “+” para adicionar una operación.
9. El sistema muestra la interfaz Operación.
10. El configurador de contexto selecciona el tipo de operación y los datos referentes a ese tipo y presiona el botón Adicionar.
11. El configurador de contexto cierra la interfaz Operaciones.
12. El configurador de contexto presiona el botón Siguiente.
13. El sistema muestra la interfaz Configurar Manejador.
14. El configurador de contexto selecciona el tipo de Acción y los atributos de la misma.
15. El configurador de contexto presiona el botón “…” para adicionar la nueva excepción que va a envolver a la original.
16. El sistema muestra la interfaz Seleccionar Excepción.
17. El configurador de contexto selecciona y presiona el botón Seleccionar.
18. El configurador de contexto presiona el botón “Adicionar Acción”.
19. El sistema adiciona la acción a la lista.
20. El configurador de contexto escribe el mensaje y la ayuda del nuevo manejador.
21. El configurador de contexto presiona el botón “Finalizar”.
22. El sistema adiciona el nuevo contexto con la configuración seleccionada.
23. El sistema muestra un mensaje que confirma que se adiciono el contexto.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El configurador de contexto ordena adicionar contexto o Excepción		El sistema muestra la interfaz seleccionar excepción.	Satisfactorio	
	El configurador de contexto oprime el botón Siguiente sin seleccionar la excepción	El sistema muestra una alerta al lado del botón, informando que debe seleccionar la excepción	Satisfactorio	
El configurador de contexto selecciona la excepción deseada y presiona el botón Siguiente.		El sistema muestra la interfaz Configurar Contexto.	Satisfactorio	
El configurador de contexto selecciona el módulo al que pertenece el nuevo contexto		El sistema muestra una lista de módulos disponibles y permite seleccionar un módulo.	Satisfactorio	
El configurador de contexto oprime el botón "+"		El sistema muestra la interfaz Operación	Satisfactorio	
El configurador de contexto selecciona el tipo de operación a adicionar, escribe los datos necesario de la operación y oprime el		El sistema adiciona una nueva operación al listado de operaciones del contexto.	Satisfactorio	

botón "Adicionar".				
El configurador de contexto cierra la interfaz Operaciones		La interfaz Operaciones se cierra y se activa la interfaz Configurar Contexto.	Satisfactorio	
El configurador de contexto oprime el botón "Siguiente"		El sistema muestra la interfaz Configurar Manejador.	Satisfactorio	
	El configurador de contexto oprime el botón Finalizar sin seleccionar ni poner ningún dato adicional	El sistema muestra advertencias en todos los campos que debe llenar.	Satisfactorio	
El configurador de contexto selecciona un tipo de Acción		El sistema permite seleccionar un tipo de acción	Satisfactorio	
El configurador de contexto oprime el botón "…"		El sistema muestra la interfaz Seleccionar Excepción	Satisfactorio	
El configurador de contexto selecciona la excepción deseada y oprime el botón Seleccionar		El sistema cierra la interfaz Seleccionar Excepción y muestra la excepción seleccionada en el campo con el mismo nombre	Satisfactorio	
El configurador de contexto Oprime el botón "Adicionar Acción"		El sistema adiciona a la lista la nueva acción.	Satisfactorio	

El configurador de contexto escribe el mensaje del manejador, la ayuda y oprime el botón Finalizar adicionando el contexto configurado en todo el proceso		El sistema adiciona el nuevo contexto configurado, y muestra un mensaje indicando que finalizó la operación correctamente	Satisfactorio	

Tabla 12 Caso de prueba 1 Adicionar contexto con una operación y una acción.

CPR2. Adicionar contexto con dos operaciones y dos acciones.

Flujo Central

- El flujo central de este caso de prueba coincide con el caso de prueba anterior, diferenciándose en la adición de más de una operación en el paso 10 y más de una acción en el paso 18.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El configurador de contexto adiciona un nuevo contexto que tiene dos operaciones y dos acciones		El sistema adiciona el nuevo contexto configurado y muestra un mensaje de confirmación	Satisfactorio	Este caso de prueba se realizó para verificar los caminos del proceso que utilizaban ciclos.

Tabla 13 Caso de prueba 2 Adicionar contexto con dos operaciones y dos acciones.

3.2.1.4. Manejar excepción.

Se realizaron varias pruebas al presente caso de uso, con el propósito de evaluar los diferentes tipos de casos que se pueden producir.

CPR1. Excepción controlada que se configuró para que sea cubierta con una nueva.

Flujo Central

1. El actor sistema produce una excepción.
2. El intérprete procesa la excepción producida.

Clases Válidas	Clases Inválidas	Resultado Esperado	Resultado de la Prueba	Observaciones
El sistema produce una excepción ya configurada anteriormente.		El interprete busca el contexto de la excepción y ejecuta el manejador	Satisfactorio	
El sistema produce una excepción ya configurada anteriormente con una acción reemplazar y un atributo archivar.		El interprete ejecuta la acción reemplazar encontrada y es archivada.	Satisfactorio	
El sistema produce una excepción ya configurada anteriormente con una acción cubrir con un atributo archivar, y otro salir de la aplicación.		El interprete ejecuta la acción cubrir encontrada archiva y finaliza la aplicación.	Satisfactoria	
El sistema produce una excepción ya configurada anteriormente con una acción cubrir y otra visualizar con un atributo archivar.		El interprete ejecuta la acción cubrir encontrada archiva y ejecuta la acción visualizar y visualiza el mensaje con la ayuda del manejador asociado.	Satisfactorio	
	El sistema produce una excepción no controlada y	El sistema no encuentra un contexto para la excepción y	Satisfactorio	

	configurada anteriormente.	visualiza un mensaje informando que se produjo una excepción no controlada.		

Tabla 14 Caso de prueba 1 Excepción controlada que se configuró para que sea cubierta con una nueva.

3.3. Conclusiones

En consecuencia al trabajo realizado se ejercitaron completamente los requisitos funcionales del sistema, demostrándose que las funciones del software son totalmente operativas. En efecto, fueron derivadas un conjunto de reglas en soporte a las técnicas aplicadas de acuerdo a los métodos establecidos, encaminadas a obtener los casos de pruebas asociados a los casos de usos del módulo Configurador y el intérprete de excepciones. Se realizó el estudio y análisis de la complejidad lógica del sistema a través de diferentes herramientas, arribando a resultados concretos y dando validez a la calidad del sistema.

CONCLUSIONES

La realización de la presente investigación deviene en favorables resultados que pueden concluirse de la siguiente forma:

- Constituyó un punto de partida el estudio del proceso de tratamiento de excepciones actual en la plataforma .NET, dirigido en primera instancia a un mejor entendimiento del negocio a diseñar e implementar.
- Se realizó un análisis completo de la problemática relacionada con la gestión y manipulación de excepciones que oscurece el código de un sistema desarrollado sobre arquitectura .NET que contribuyó a un mejor entendimiento del problema a resolver.
- Se obtuvo el Modelo de Diseño y de Implementación, concebidos convenientemente de acuerdo a las especificaciones de los casos de uso del sistema, los requisitos asociados a éstos y las reglas generales del negocio, resultado que constituyó el punto de entrada a la implementación de las clases elaboradas.
- La validación del diagrama de clases y los elementos que lo componen permitió verificar la obtención del modelo de diseño con la calidad requerida para un desarrollo de la fase de implementación con mínimos riesgos.
- Las pruebas de unidad permitieron la verificación del sistema desarrollado asegurando que el mismo tuviera la calidad requerida.

Finalmente como conclusión general se logró independizar el tratamiento asegurado de excepciones del código de los programadores; facilitando la claridad y más información de las mismas.

RECOMENDACIONES

- Integrar el intérprete de excepciones a los proyectos sobre arquitectura .NET para aprovechar las ventajas que ofrece y poder probar su desempeño en grandes aplicaciones.
- Documentar el intérprete y el configurador para uso de otros desarrolladores.
- Continuar el desarrollo del intérprete para lograr su transformación en un framework manejador de excepciones incrementando sus funcionalidades y ventajas de uso.
- Continuar el curso de la investigación científica para el aporte de resultados novedosos sobre la base del presente trabajo.

BIBLIOGRAFÍA

A field guide to Boxology: Preliminary classification of architectural styles for software systems. **Shaw, Mary y Clements, Paul. Abril de 1996.** Computer Science Department and Software Engineering Institute, Carnegie Mellon University, Abril de 1996., Pennsylvania, Estados Unidos de América : s.n., Abril de 1996.

Almenares, Kiosmy y de León, Rubén. 2007. *Diseño e implementación del proceso de inscripción del Módulo de Mercantil en las Oficinas Registrales de la República Bolivariana de Venezuela.* Caracas, Venezuela : s.n., 2007.

Ávila, Rodolfo Pérez. 2007. Análisis, Diseño e Implementación de los Servicios en Línea del Módulo Mercantil del Proyecto de Informatización de los Registros y Notarías. [Trabajo de Diploma para optar por el título de Ingeniero Informático]. Caracas, Venezuela : s.n., Mayo de 2007.

Charte Ojeda, Francisco. 2002. *Visual C# .Net.* Madrid : Anaya Multimedia, 2002.

Cuevas Guerrero, Daniel. Universidad de Málaga. *Departamento de Lenguajes y Ciencias de la Computación.* [En línea] [Citado el: 7 de Diciembre de 2007.] http://www.lcc.uma.es/~pastrana/EP/trabajos/45.pdf.

Dodero, Juan Manuel y Fernández Llamas, Camino. 2002-2003. Universidad Carlos III de Madrid. *Laboratorio DEI.* [En línea] 2002-2003. [Citado el: 5 de Marzo de 2008.] http://peterpan.uc3m.es/docencia/p_s_ciclo/tdp/curso0203/apuntes/factory.pdf.

Ferguson, Jeff, Patterson, Brian y Beres, Jason. 2003. *La Biblia de C#.* Madrid : Anaya Multimedia, 2003.

Gamma, Erich, y otros. 1998. *Design Patterns CD.* s.l., Massachusetts : Addison Wesley Longman Inc., 1998. Design Patterns:Elements of Reusable Object-Oriented Software.

Guía de referencia básica de Ada. [En línea] http://www.gedlc.ulpgc.es/docencia/NGA/excepciones.html.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2002. *El Proceso Unificado de Desarrollo de Software.* s.l. : Addison Wesley, 2002.

Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2004. *El Proceso Unificado de Desarrollo de Software.* Ciudad de la Habana : Félix Varela, 2004. Vol. 1.

Lanzarini, Laura. Instituto de Investigación en Informática LIDI. *Facultad de Informática.* [En línea] [Citado el: 21 de Febrero de 2008.] http://weblidi.info.unlp.edu.ar/catedras/seminariob/Introduccion.doc.

Martin Torres, Daniel Javier y Palacios Mellado, Daniel. 2002. *C#: Clases, Polimorfismo y Excepciones.* Salamanca : s.n., 2002.

Martínez, Natividad. http://www.it.uc3m.es/tsirda/material/Tema10.pdf. [En línea] http://www.it.uc3m.es/tsirda/material/Tema10.pdf.

Microsoft. 2008. MSDN. [En línea] 2008. http://msdn.microsoft.com/es-es/library/e80y5yhx(VS.80).aspx.

Navarro, Juan José Moreno y Collado, M. Universidad Politécnica de Madrid. *Facultad de Informática, Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software.* [En línea] [Citado el: 22 de Febrero de 2008.] http://lml.ls.fi.upm.es/~jjmoreno/sbc/intro_pbc.ppt.

Orellana, Marco. 2006. Universidad del Azuay. *Facultad de Ciencias de la Administración, Ingeniería de Sistemas.* [En línea] Enero de 2006. [Citado el: 21 de Febrero de 2008.] http://uazuay.edu.ec/estudios/sistemas/eventos/cuaderno_docente.doc.

Pérez, Isidro Pablo. http://www.mitecnologico.com/Main/ExcepcionesDefinicion. [En línea] http://www.mitecnologico.com/Main/ExcepcionesDefinicion.

Pressman, Roger S. 2002. *Ingeniería de Software, un enfoque práctico.* Quinta Edición. s.l. : Mc Graw Hill, 2002.

Rational Software Corporation. 2003. *Rational Unified Process.* 2003. Vol. 2003.06.00.65.

Rizzi, Francisco Marcelo. Instituto Tecnológico de Buenos Aires (ITBA). [En línea] [Citado el: 8 de Mayo de 2008.] http://www.itba.edu.ar/capis/rtis/articulosdeloscuadernosetapaprevia/RIZZI-COMPLEJIDAD.pdf.

Romero, Eduardo Álvarez y Pueyo, Daniel. 2004-2005. *Integration Definition For Funcion Modeling.* 2004-2005.

Rumbaugh, James y Jacobson, Ivar and Grady Booch. 2000. *El Lenguaje Unificado de Modelado, Manual de Referencia.* 2000.

Sparx Systems. 2007. Sparx Systems. *Enterprise Architect - Herramienta de diseño UML.* [En línea] 2007. [Citado el: 10 de Marzo de 2008.] http://www.sparxsystems.com.ar/products/ea.html.

SYNSPACE AG. 2005. SYNSPACE AG. [En línea] 17 de Agosto de 2005. [Citado el: 28 de Marzo de 2008.] http://www.synspace.com/ES/Services/tcc.html.

Teso, Leandro del. El Paradigma orientado a objetos. [En línea] [Citado el: 5 de Diciembre de 2007.] http://www.monografias.com/trabajos14/paradigma/paradigma.shtml.

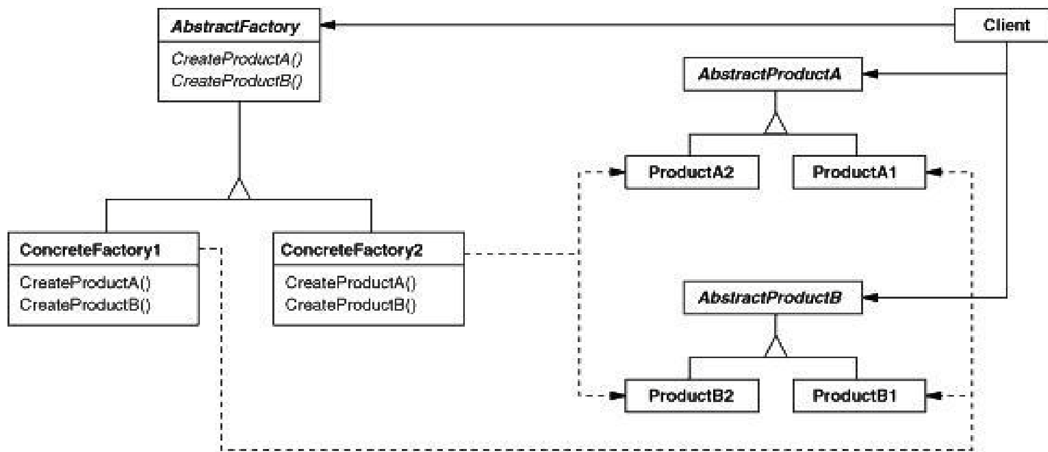
Trowbridge, David. 2003. *Enterprise Solution Patterns using Microsoft .NET. s.l.* s.l. : Microsoft Corporation, 2003., 2003.

Universidad de Burgos. 1999. Introducción a la Programacion Orientada a Objetos. [En línea] Octubre de 1999. [Citado el: 5 de Diciembre de 2007.] http://pisuerga.inf.ubu.es/lsi/Invest/Java/Tuto/I_1.htm.

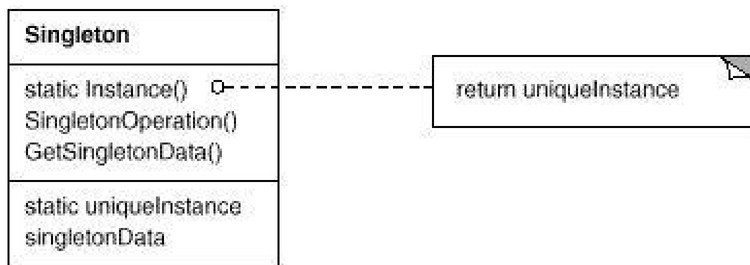
Universidad de Jaén. 2008. Departamanto de Informática. [En línea] 15 de Febrero de 2008. [Citado el: 22 de Febrero de 2008.] http://wwwdi.ujaen.es/asignaturas/progav/progav-tema4.pdf.

ANEXOS

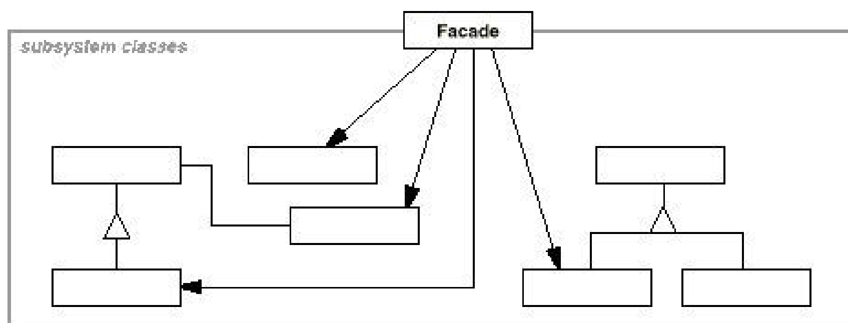
Anexo 1: Estructura del Patrón Abstract Factory (Fábrica Abstracta).



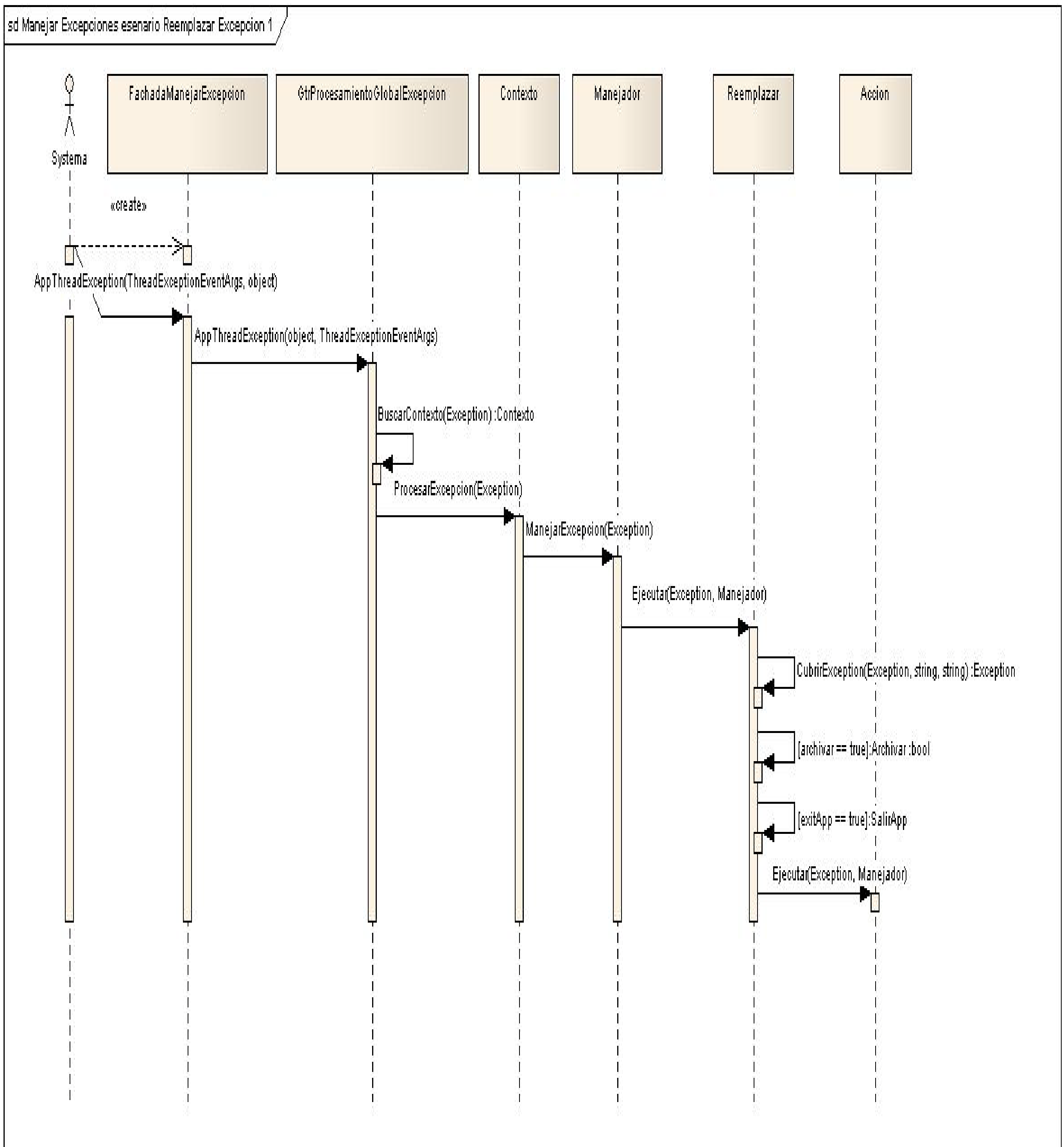
Anexo 2: Estructura del Patrón Singleton.



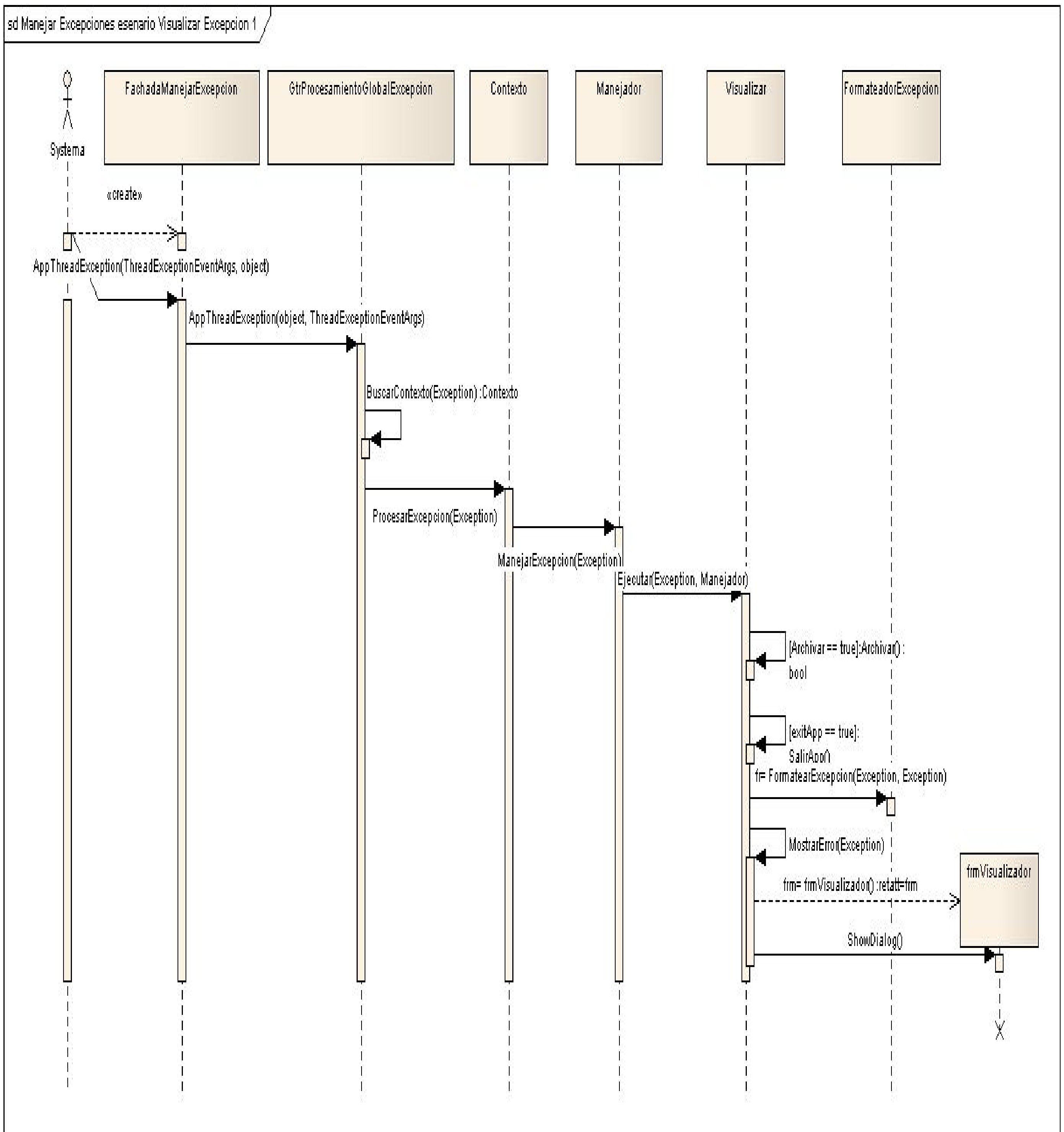
Anexo 3: Estructura del Patrón Facade.



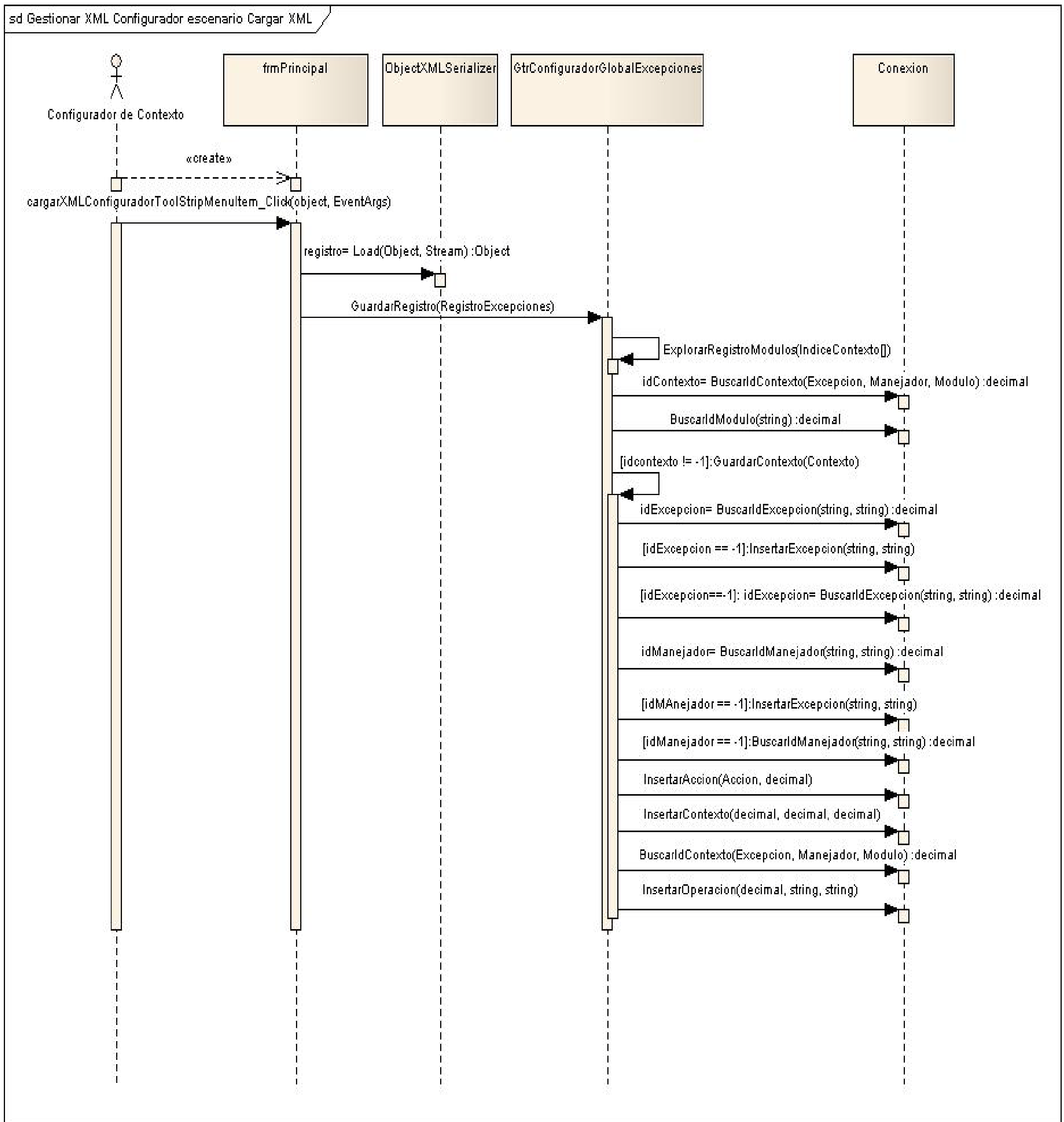
Anexo 4: Diagrama de secuencia "Manejar Excepción" escenario Reemplazar Excepción.



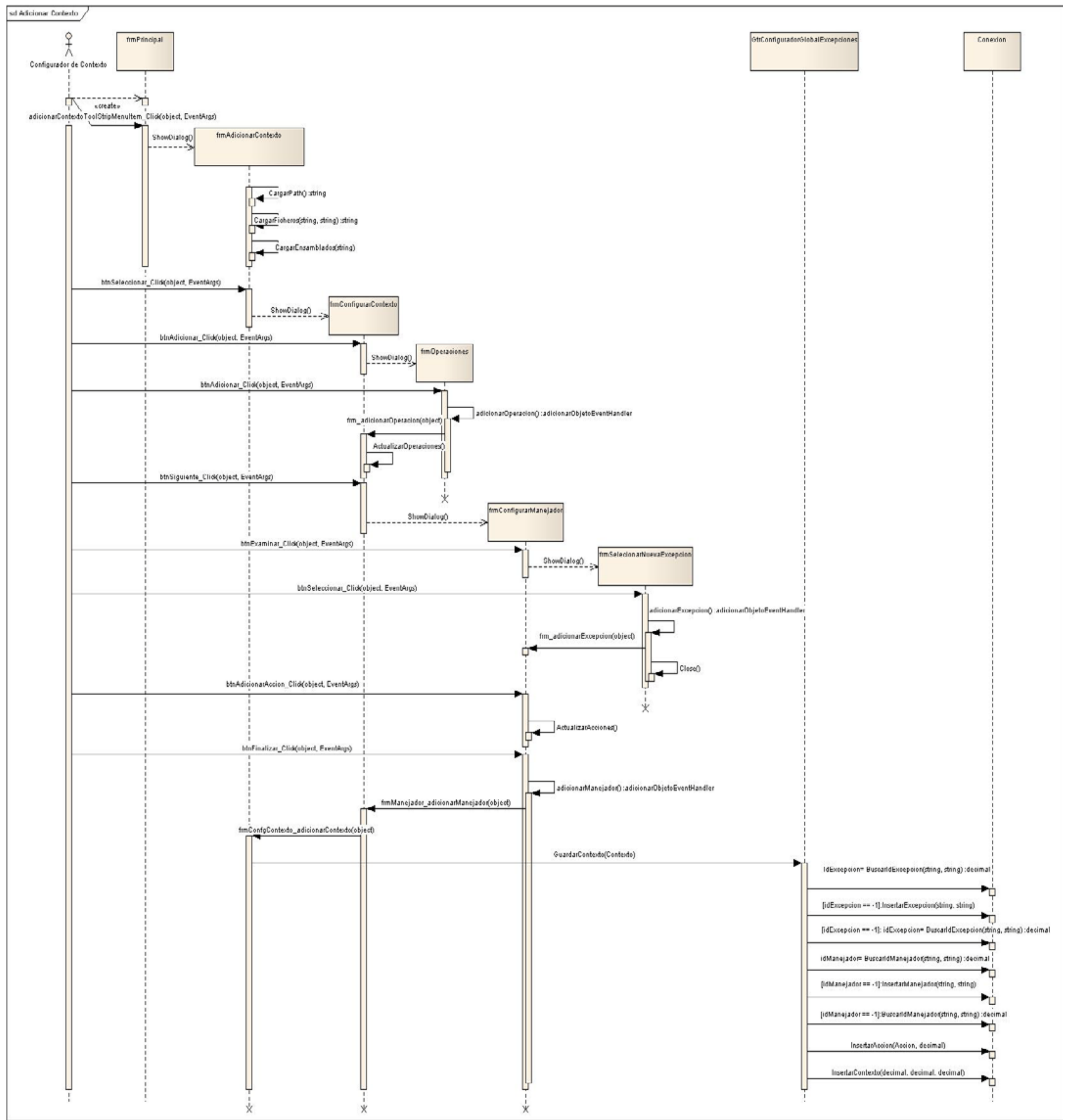
Anexo 5: Diagrama de secuencia "Manejar Excepción" escenario Visualizar Excepción.



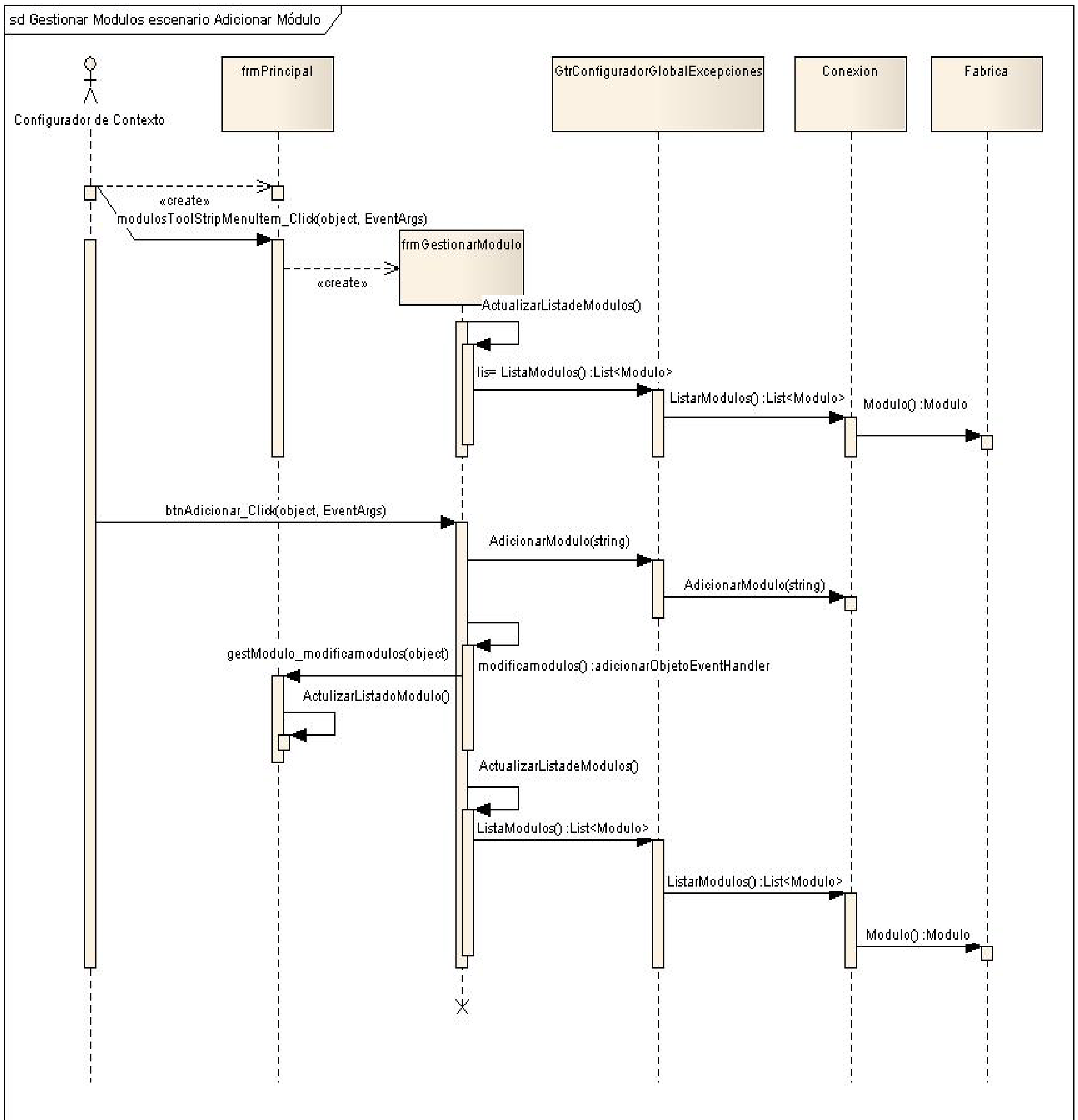
Anexo 6: Diagrama de secuencia "Gestionar XML Configurador" escenario Cargar XML.



Anexo 7: Diagrama de secuencia “Adicionar Contexto”.



Anexo 8: Diagrama de secuencia “Gestionar Módulo” escenario adicionar módulo.



Anexo 9. Metodo de la clase grtConfigurador GlobalExcepciones.

```
public void AdicionarModulo(string nombre)
{
    Conexion.Instancia.AdicionarModulo(nombre);
}
```

Anexo 10. Metodo de la clase gestora del intérprete AppThreadException.

```
public void AppThreadException(object source, ThreadExceptionEventArgs e)
{
      Contexto contexto = BuscarContexto(e.Exception);
      if(contexto != null)
            contexto.ProcesarExcepcion(e.Exception);
      else
      {
            Visualizar v = Fabrica.Instancia.Visualizar();
            Exception ex = Fabrica.Instancia.Exception("Ha ocurrido una excepción no
            controlada, consulte los detalles para más información.", e.Exception);
            v.Ejecutar(ex,null);
      }
}
```

Anexo 11. Metodo de la clase frmPrincipal btnGenerarXML_Click que implementa el evento click del boton btngenerarXML.

```
private void btnGenerarXML_Click(object sender, EventArgs e)
{
      epAyudaErrores.Clear();
      List<Modulo> modulosSeleccionados = new List<Modulo>();
      for (int i = 0; i < dgvListaModulos.Rows.Count; i++)
      {
            if ((bool)dgvListaModulos.Rows[i].Cells[0].Value == true)
            {
                  modulosSeleccionados.Add(listadoModulos[i]);
```

```
            }
        }
        if (modulosSeleccionados.Count != 0)
        {
            DialogResult resultadoOperacion;
            string direccion;
            SaveFileDialog sfdExportarXML = new SaveFileDialog();
            sfdExportarXML.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*";
            sfdExportarXML.DefaultExt = "xml files (*.xml)";
            resultadoOperacion = sfdExportarXML.ShowDialog();
            direccion = sfdExportarXML.FileName;
            if (resultadoOperacion == DialogResult.OK)
            {
                try
                {
                    ConfiguradorGlobalExcepciones.Instancia.ExportarRegistroExcepciones
                                                  (modulosSeleccionados,direccion);
                }
                catch (Exception ex)
                {
                        MessageBox.Show("La  operacion  n  se  pudo  completar  por  una
                        interrupcion: " + ex.Message);
                }
                }
            else
            {
                  epAyudaErrores.SetError(btnGenerarXML,  "Debe  selecionar  un  nombre
                  para el archivo a guardar");
            }
          }
          else  epAyudaErrores.SetError(btnGenerarXML,  "Debe  selecionar  al  menos  un
modulo");
}
```

Anexo 12. Metodo de la clase frmPrincipal cargarXMLConfiguradorToolStripMenuItem_Click que implementa el evento click del menú principal la opción Cargar XML Configurador.

```
private void cargarXMLConfiguradorToolStripMenuItem_Click(object sender, EventArgs e)
{
      RegistroExcepciones registro;
      OpenFileDialog OP = new OpenFileDialog();
      OP.Filter = "XML files (*.xml)|*.xml|All files (*.*)|*.*";
      DialogResult result = OP.ShowDialog();
      if (result == DialogResult.OK)
      {
          string direccion = OP.FileName;
          registro = ProcesamientoGlobalExcepcion.DeserealizarXML(direccion);
          ConfiguradorGlobalExcepciones.Instancia.GuardarRegistro(registro);
      }
}
```

Anexo 13. Metodo ejecutar de la clase Cubrir.

```
public override void Ejecutar(Exception ext,Manejador m)
{
      Exception e = CubrirException(ext,m.Mensaje,m.Ayuda);
      if (Archivar)
            ArchivarError(e);
      if(ExitApp)
            SalirApp();
      base.Ejecutar(e,m);
}
```

Anexo 14. Metodo GuardarContexto de la clase gestora del módulo configurador.

```
public void GuardarContexto(Contexto contex)
{
    decimal idExcepcion = Conexion.Instancia.BuscarIdExcepcion(
```

```
                contex.ExcepcionOriginal.Nombre,contex.ExcepcionOriginal.Tipo);
    if (idExcepcion == -1)
    {
            Conexion.Instancia.InsertarExcepcion(contex.ExcepcionOriginal.Nombre,
            contex.ExcepcionOriginal.Tipo);
            idExcepcion = Conexion.Instancia.BuscarIdExcepcion(
                          contex.ExcepcionOriginal.Nombre,
                contex.ExcepcionOriginal.Tipo);
    }
    contex.ExcepcionOriginal.Idexcepcion = idExcepcion;
    decimal                                idManejador                              =
Conexion.Instancia.BuscarIdManejador(contex.Manejador.Mensaje,
                                                               contex.Manejador.Ayud
                                                          a);
    if (idManejador == -1)
    {
      Conexion.Instancia.InsertarManejador(contex.Manejador.Mensaje,
contex.Manejador.Ayuda);
      idManejador = Conexion.Instancia.BuscarIdManejador(contex.Manejador.Mensaje,
                                                          contex.Manejador.Ayuda);
    }
    contex.Manejador.Idmanejador = idManejador;
    Conexion.Instancia.InsertarAccion(contex.Manejador.Acciones[0],
                                      contex.Manejador.Idmanejador);
    Conexion.Instancia.InsertarContexto(contex.ExcepcionOriginal.Idexcepcion,
                                        contex.Manejador.Idmanejador,
                                        contex.Modulo.Idmodulo);
   contex.Idcontexto = Conexion.Instancia.BuscarIdContexto(contex.ExcepcionOriginal,
                                                       contex.Manejador,
                                             contex.Modulo);
   int i = 0;
   while (i < contex.Operaciones.Length && contex.Operaciones[i] != null)
   {
```

```
        if (contex.Operaciones[i] is BuscarCadenaEntreCaracteres)
              Conexion.Instancia.InsertarBuscarCadenaEntreCaracteres(contex.Idcontexto,
                          contex.Operaciones[i].Nombre,      contex.Operaciones[i].Valor,
                    (contex.Operaciones[i]     as     BuscarCadenaEntreCaracteres).Count,
                    (contex.Operaciones[i]   as   BuscarCadenaEntreCaracteres).Delimiter,
                    (contex.Operaciones[i] as BuscarCadenaEntreCaracteres).Pos);
        else if (contex.Operaciones[i] is BuscarPropiedad)
              Conexion.Instancia.InsertarBuscarPropiedad(contex.Idcontexto,
                    contex.Operaciones[i].Nombre,           contex.Operaciones[i].Valor,
                    (contex.Operaciones[i] as BuscarPropiedad).Propiedad);
        else
             Conexion.Instancia.InsertarBuscarSubcadena(contex.Idcontexto,
                    contex.Operaciones[i].Nombre,           contex.Operaciones[i].Valor,
              (contex.Operaciones[i] as BuscarSubcadena).Lengh, (contex.Operaciones[i] as
              BuscarSubcadena).Start);
       i++;
    }
  }
```

GLOSARIO

A

Actores del Sistema

Constituyen una idealización de una persona externa, de un proceso, o de una cosa que interactúa con un sistema, un subsistema, o una clase. Puede ser un ser humano, otro sistema informático, o un cierto proceso ejecutable.

API

Del inglés (Application Programming Interface - Interfaz de Programación de Aplicaciones) es un conjunto de especificaciones de comunicación entre componentes software. Se trata del conjunto de llamadas al sistema que ofrecen acceso a los servicios del sistema desde los procesos y representa un método para conseguir abstracción en la programación, generalmente entre los niveles o capas inferiores y los superiores del software.

C

CLR

(Common Language Runtime) en español Lenguaje común en tiempo de ejecución.

Componente.

Es un objeto particular que puede ser reutilizado en diferentes contextos. Los componente de Software son todo aquel recurso desarrollado para un fin concreto y que puede formar solo o junto con otro/s, un entorno funcional requerido por cualquier proceso predefinido. Son independientes entre ellos, y tienen su propia estructura e implementación. Parte modular de un sistema, desplegable y reemplazable. Típicamente contiene clases y puede ser implementado por uno o más artefactos.

Contexto

Ámbito que conceptualiza la ocurrencia de una excepción en un sistema que contiene también que acciones realizar cuando esta se presenta además de las operaciones a realizar para identificar correctamente que es ese el contexto.

D

DataSet

Es una estructura de datos del Microsoft .Net framework que encapsula un conjunto de datos y soporta un modelo relacional de una o más tablas.

DLL

Es el acrónimo de Dynamic Linking Library (Bibliotecas de Enlace Dinámico), término con el que se refiere a los archivos con código ejecutable que se cargan bajo demanda del programa por parte del sistema operativo. Esta denominación se refiere a los sistemas operativos Windows siendo la extensión con la que se identifican los ficheros, aunque el concepto existe en prácticamente todos los sistemas operativos modernos

E

Evento

Es un hecho que se produce en un momento dado bajo ciertas condiciones y puede desencadenar reacciones. Constituye las acciones que pueda ejecutar un usuario sobre el programa que está utilizando en ese momento.

I

IEEE

Institute of Electrical and Electronics Engineers, Inc., en español Instituto de electrónica e ingenieros electrónicos es la asociación líder mundial de profesionales. Su propósito fundamental es adoptar innovaciones tecnológicas y la excelencia en beneficio de la humanidad. Reconocida mundialmente por las contribuciones tecnológicas.

IDE

Un entorno de desarrollo integrado o en inglés Integrated Development Environment ('IDE') es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios. Es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI.

P

Polimorfismo

En programación orientada a objetos se denomina polimorfismo a la capacidad que tienen objetos de diferentes clases de responder al mismo mensaje.

S

Servicio Web

En inglés (Web Service) es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

Sparx Systems

Es una empresa australiana líder en el Análisis y Diseño de Sistemas y cuenta con varios productos para realizar estas actividades que están basadas en el estándar UML 2.1 de la OMG.

W

Win32

Significa "Windows 32 bits". Hace referencia a todas las plataformas de 32 bits del sistema operativo Windows: Windows NT, Windows 95, Windows 98, Windows CE.