TESIS
II 2007
L8
v.1

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

"HERRAMIENTA DE GENERACIÓN DE CÓDIGO PARA PERSISTIR DATOS CON NHIBERNATE"

TRABAJO ESPECIAL DE GRADO

presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

como parte de los requisitos para optar al título de

INGENIERO INFORMÁTICA

REALIZADO POR Kristal Lucero.

PROFESOR GUIA Lic. William Torrealba.

FECHA Febrero 2007.

Tabla de Contenidos

Introducción 1

Capítulo 1
Planteamiento del problema 3
1.1 Objetivo General 6
1.2 Objetivos Específicos 6
1.3 Limitaciones y Alcance 7
1.4 Justificación 8

Capítulo 2
Marco teórico 10
2.1 Proceso de desarrollo 10
2.2 Ingeniería de software asistida por computadora 12
2.2.1 Herramientas de generación de código 14
2.3 Programación Orientada a Objetos 16
2.3.1 Objeto 17
2.3.2 Clase 17
2.3.3 Encapsulamiento 17
2.3.4 Herencia 17
2.3.5 Polimorfismo 18
2.4 Modelo orientado a objetos 18
2.5 Modelo relacional 20
2.6 Object – Relational Mapping 21
2.7 NHibernate 23
2.8 Reflexión 24

Capítulo 3
Marco metodológico 26

Capítulo 4
Desarrollo..... 35
4.1 Análisis preliminar y especificación de requisitos.....35
4.2 Diseño global.....42
4.3 Diseño detallado.....45
4.4 Prototipos.....50
4.5 Pruebas.....54
4.6 Producto final.....57

Resultados..... 59

Conclusiones..... 61

Recomendaciones..... 63

Bibliografía..... 65

Glosario..... 69

Apéndice A
Casos de uso..... 76

Apéndice B
Diagramas de casos de uso..... 85

Apéndice C
Diagrama de actividades..... 88

Apéndice D
Diagramas de clases..... 93

Apéndice E
Diagramas de secuencia..... 96

Apéndice F
Herramientas de programación..... 101

Apéndice G
Estrategias de mapeo..... 102

Apéndice H
Estándares de codificación utilizados..... 118

Apéndice I
Manual del usuario..... 121

Apéndice J

Pruebas..... 142

Apéndice K
Diagramas de clases usados para las pruebas y sus resultados..... 148

Apéndice L
Resultados de las pruebas de evaluación del usuario..... 165

Apéndice M
Estándares de codificación de los archivos DDL..... 167

Apéndice N
Collaboration Responsability Cards..... 170

Apéndice Ñ
Introducción a los CRC..... 182

Apéndice O
Propiedades..... 187

Apéndice P
Reflexión..... 190

Apéndice Q
Atributos..... 194

Apéndice R
Documentación de la herramienta..... 198

Índice de tablas

Tabla 1: Esquema modelo metodológico adaptado. 34
Tabla 2: Caso de uso #1: Realizar mapeo de las clases.....46
Tabla 3: Listado de atributos.....47
Tabla 4: Tipos de datos de la plataforma .NET vs, Tipos de datos de Microsoft SQL.....48
Tabla 5: Tipos de datos de la plataforma .NET vs, Tipos de datos de NHibernate. 49
Tabla 6: Perfiles de los usuarios para la evaluación del producto final..... 58
Tabla 7: Fases de la prueba para la evaluación del producto final.....58
Tabla 8: Dialectos SQL soportados por NHibernate 1.0.2. 71
Tabla 9: Resultados de la prueba fase 1 al usuario 1.....165
Tabla 10: Resultados de la prueba fase 2 al usuario 1.....166
Tabla 11: Resultados de la prueba fase 1 al usuario 2.....166
Tabla 12: Resultados de la prueba fase 2 al usuario 2.....166

Índice de figuras

Figura 2.1: Proceso de desarrollo de software 10
Figura 2.2: Arquitectura de NHibernate. 23
Figura 2.3: Modelo jerárquico de la reflexión para la plataforma de .NET. 24
Figura 3.1: Metodología Prototipado Evolutivo. 27
Figura 4.1: Mapeo de un objeto compuesto o contenido....37
Figura 4.2: Mapeo de una asociación simple (1..1)....37
Figura 4.3: Mapeo de una colección básica (1..n) y una referencia simple (n..1).38
Figura 4.4: Mapeo de una relación múltiple (n..m)....38
Figura 4.5: Ejemplo - herencia....39
Figura 4.6: Mapeo de la herencia - "una tabla por clase jerárquica"....39
Figura 4.7: Mapeo de la herencia - "una tabla por subclase"....40
Figura 4.8: Mapeo de la herencia - "una tabla por clase concreta"....40
Figura 4.9: Diagrama de casos de uso de la herramienta....41
Figura 4.10: Modelo conceptual de la solución,....42
Figura 4.11: Proceso de generación de código para persistir datos con NHibernate. 43
Figura 4.12: Arquitectura del sistema. 43
Figura 4.13: Estructura interna de la capa NHBuilder. 45
Figura 4.14: Capas implementadas en la primera versión. 51
Figura 4.15: Capas implementadas en la segunda, tercera y cuarta versión. 51
Figura 4.16: Capas implementadas en la quinta versión. 53
Figura B.1: Diagrama de casos de uso. 85
Figura B.2: Caso de uso: Obtener estructura de clases. 86
Figura B.3: Caso de uso: Transformar estructura de clases a un modelo relacional. 86
Figura B.4: Caso de uso: Generar archivos de mapeo....87
Figura C.1: Diagrama de actividades - Realizar mapeo de las clases. 89
Figura C.2: Diagrama de actividades - Obtener estructura de clases....90
Figura C.3: Diagrama de actividades - Transformar estructura de clases a un modelo relacional. 91
Figura C.4: Diagrama de actividades - Generar archivos de mapeo. 92
Figura D.1: Diagrama de clases – MappingAttribute. 94
Figura D.2: Diagrama de clases – NHBuilder. 95
Figura E.1: Diagramas de secuencia - Realizar mapeo de las clases....97
Figura E.2: Diagramas de secuencia - Obtener estructura de clases. 98
Figura E.3: Diagramas de secuencia - Transformar estructura de clases a un modelo relacional. 99
Figura E.4: Diagramas de secuencia - Generar archivos de mapeo. 100
Figura L.1: Diagrama de clases 1 - prueba. 149
Figura L.2: Diagrama de clases 2 - prueba. 153
Figura L.3: Diagrama de clases 2 modificado - prueba. 161

Resumen

El presente trabajo, titulado "*Herramienta de generación de código para persistir datos con NHibernate*", consiste en el desarrollo de una herramienta que facilita la implantación de mecanismos de mapeo objeto-relacional para las aplicaciones orientadas a objetos que necesitan almacenar su información en bases de datos relacionales.

A través de esta herramienta el usuario puede generar los archivos necesarios para la incorporación de NHibernate a su sistema, y generar adicionalmente la estructura física de la base de datos a partir del código de las clases persistentes.

Para llevar a cabo el desarrollo de la herramienta se utilizó la metodología prototipado evolutivo, la cual recorre sucesivamente el ciclo de desarrollo de software para conseguir un refinamiento progresivo del proyecto y poder ofrecer servicio desde el primer momento.

El resultado es una herramienta capaz de reconocer la estructura de clases a partir del código, transformarlo a un modelo relacional para crear la estructura física de la base de datos usando una serie de estrategias, y generar los archivos de mapeo para las clases persistentes.

La herramienta de generación de código para persistir datos con NHibernate busca que los usuarios se concentren solamente en un buen diseño orientado a objetos.

Introducción

Hace ya más de 40 años que los Sistemas Manejadores de Bases de Datos surgieron como una necesidad para que las grandes empresas almacenaran sus datos, ya que en aquel tiempo sólo las aplicaciones más sofisticadas hacían uso de estos sistemas.

Después de muchos años los Sistemas Manejadores de Bases de Datos evolucionaron a Sistemas Manejadores de Bases de Datos Relacionales.

Los lenguajes de programación también han evolucionado a lo que se conocen actualmente como lenguajes de programación orientada a objetos.

El desarrollo orientado a objetos, apoyado en la existencia de lenguajes de programación orientado a objetos, se ha impuesto claramente y ha traído enormes ventajas en la programación.

Sin embargo, existe un eslabón débil en la orientación a objetos, ¿cómo comunicarse bien con la base de datos? Durante cierto tiempo se experimentó bastante con los llamados Sistemas Manejadores de Bases de Datos orientadas a

objetos, pero no se obtuvieron resultados adecuados.

Hoy es claro para todos que los sistemas manejadores de bases de datos relacionales dominan el mundo de los datos y lo seguirán haciendo en un futuro previsible y se trabaja mucho en la creación de mecanismos de convivencia entre programas orientados a objetos y bases de datos relacionales como tal es el caso de los llamados ORM (Object Relational Mapping).

Si bien es cierto que los ORM son mecanismos que en efecto eliminan la brecha entre el mundo relacional y el orientado a objetos, se debe pagar un precio muy costoso, en términos de tiempo, para implantarlo y poder disfrutar de sus beneficios.

Los procedimientos a seguir para implantar los ORM añaden un nivel más de dificultad en el desarrollo de programas porque exigen el cumplimiento de ciertos criterios para que exista compatibilidad entre la aplicación y la base de datos.

Capítulo 1

Planteamiento del problema

La mayoría de las aplicaciones necesitan una lógica de persistencia de los datos, es decir almacenar su información de forma que perdure en el tiempo, es por ello que se necesita ante todo una buena estrategia que permita grabar y recuperar la información. Por lo general esta información es almacenada en una base de datos relacional.

Las aplicaciones orientadas a objetos, que hacen uso de las bases de datos relacionales para persistir sus datos, sufren de algo llamado el *diferencial de impedancia*. Éste término fue acuñado en la ciencias computacionales a mediados de los años ochenta y se refiere a un problema electrónico cuando no se puede conectar dos componentes con diferente impedancia. Esto mismo sucede cuando se intenta hacer sinergia entre la aplicación, y el sistema manejador de base de datos relacional.

Una estrategia típica para la persistencia de datos, es que las clases se

encarguen de almacenar y recuperar la información, estas clases son llamadas *clases del negocio*. La mayoría de estas clases contienen instrucciones SQL escritas explícitamente.

Pero esta estrategia tiene fuertes limitaciones, primero porque no se puede evaluar la instrucción SQL sino hasta en tiempo de ejecución, y resulta muy difícil asegurar que toda la instrucción SQL sea siempre válida y correcta. Segundo, el código está muy ligado a lo que es el dialecto SQL del sistema manejador de base de datos, lo que lo hace muy dependiente.

Pero el verdadero problema denominado diferencial de impedancia, consiste en el hecho de que no existe una traducción exacta de la estructura de clases a un esquema relacional.

Al hacerse evidente este problema, surgieron soluciones como el patrón de diseño DAO, el cual encapsula el código de las transacciones en clases especializadas para ello y lo separa de las clases encargadas del procesamiento y control del sistema, permitiendo un mayor desacoplamiento. Pero este patrón de diseño no soluciona el problema, debido a que DAO son clases que dependen de igual forma del sistema manejador de base de datos.

Hoy en día existen los Object Relational Mapping (ORM), que ayudan a reducir el llamado diferencial de impedancia. Los ORM eliminan el requerimiento de escribir instrucciones SQL para cargar y persistir el estado de los objetos. Aunque aún se tiene que escribir consultas, una buena herramienta ORM debería hacerlo

más sencillo.

Un ORM permite crear un modelo de dominio apropiado sin muchos de los problemas asociados con el modelo relacional. Se puede pensar en términos de objetos, en vez de filas y columnas.

Los ORM mejoran la portabilidad de la aplicación en términos de bases de datos reduciendo la dependencia de los dialectos SQL. Con la implantación de un ORM el desarrollador logra abstraerse del manejo de los diferentes dialectos.

Un ORM conocido para aplicaciones desarrolladas en la plataforma de .NET es NHibernate. Los pasos siguiente explican en modo general cómo debe utilizarse NHibernate:

- Implementar las clases persistentes en .NET
- Crear las tablas asociadas a cada clase.
- Crear el archivo de mapeo de cada clase.
- Crear el archivo de configuración de NHibernate.
- Utilizar la API de NHibernate para persistir y recuperar los objetos.

En la realidad este proceso se puede convertir en algo engorroso, si la aplicación es de gran tamaño. Además este proceso exige dominar las estrategias de transformación o mapeo de las clases a un esquema relacional.

NHibernate utiliza el lenguaje XML para los llamados archivos de mapeo, que contienen la traducción o correspondencia de las clases y de cada uno de los campos a sus respectivas tablas y columnas. Posee una serie de especificaciones a seguir

para representar las clases de dominio en los archivos de mapeo.

Además debe existir un archivo de configuración, también en XML, donde se le debe indicar a NHibernate el sistema manejador de base de datos para determinar el dialecto SQL a utilizar.

Se debe invertir un tiempo considerable para escribir correctamente los archivos de mapeo, siguiendo cada una de las especificaciones de NHibernate, por cada clase persistente, por lo tanto el desarrollador puede reconsiderar y desistir de la implantación de dicha tecnología.

1.1 Objetivo General

Crear una herramienta que analice la definición de dominio y genere para NHibernate, la definición de las tablas asociadas a las clases de persistencia en la base de datos con su correspondiente archivo de mapeo.

1.2 Objetivos Específicos

- Definir un mecanismo automatizado de mapeo de las tablas en lenguaje SQL y las clases.
- Definir el Data Definition Language (DDL o Lenguaje de definición de datos) de las tablas.
- Crear automáticamente el documento en formato XML para hacer el mapeo entre las clases con sus tablas correspondientes.

- Crear una herramienta capaz de explorar la estructura de clases, y generar los archivos de mapeo y el archivo con el DDL, que permita la incorporación, en la generación de estos archivos, de diferentes dialectos SQL.

1.3 Limitaciones y Alcance

Esta herramienta de generación de código esta enfocada en la tecnología NHibernate, que es una capa de persistencia de datos. Esta capa permite la interconexión de las clases encargadas de manejar los datos con la base de datos.

Es importante resaltar que los archivos a generar por esta herramienta no implementan el patrón de diseño DAO, sino que, más bien gracias a la implementación de NHibernate, este patrón queda suprimido. Los archivos a generar son únicamente el archivo con el Data Definition Language y los archivos de mapeo, requisitos necesarios para NHibernate.

El mapeo parte de un modelo orientado a objetos por lo tanto se deben tomar en cuenta los casos especiales como lo son la herencia y las relaciones entre objetos.

El mecanismo permitirá el mapeo de la herencia considerando que NHibernate tiene varias formas de mapearla:

- Una tabla por clase jerárquica
- Una tabla por subclase
- Una tabla por cada clase concreta, usando polimorfismo implícito.

Y también permitirá el mapeo de las colecciones y las asociaciones:

- Uno a Uno (one-to-one)
- Uno a Muchos (one-to-many)
- Muchos a Uno (many-to-one)
- Muchos a muchos (many-to-many)

El estándar de C# que será utilizado será ECMA 334 3ra edición, el dialecto de SQL será el correspondiente al manejador Microsoft SQL Server, la versión del manejador será MS SQLServer 2005 y la versión de NHibernate será la 1.0.2 o superior.

1.4 Justificación

NHibernate es una herramienta que ofrece muchas ventajas porque permite pensar en las aplicaciones íntegramente desde el modelo de objetos y no desde la perspectiva del modelo de datos. Además de reducir considerablemente el número de líneas de código necesarias para mantener la persistencia de los datos, separa la base de datos del modelo de negocios.

El problema de usar herramientas ORM como NHibernate, es que hacen falta una serie de requisitos adicionales para el buen funcionamiento, lo cual puede tomar un tiempo considerable, que muchas veces no está contemplado dentro del tiempo estipulado para la etapa de construcción e implantación en el proceso de desarrollo.

El propósito central del presente Trabajo Especial de Grado, es la de apoyar al desarrollador en el proceso de desarrollo para incorporar NHibernate a su aplicación.

De esta forma la aplicación podrá disfrutar de los beneficios que brinda un ORM, como lo son: portabilidad con respecto a los sistemas manejadores de bases de datos, un código fácil de mantener, debido a que se eliminan, en la mayoría de los casos, las instrucciones SQL, y compatibilidad de la estructura de clases y el esquema relacional.

Existen actualmente herramientas que parten del modelo relacional para generar las clases y los archivos de mapeo. Pero esta estrategia tiene ciertas limitaciones, y es que las clases generadas pierden toda la esencia de la programación orientada a objetos, ya que no es posible determinar la herencia a partir de tablas, y mucho menos identificar cierto tipo de relaciones entre las clases.

Es por esto que es importante destacar que la herramienta de generación de código para persistir datos con NHibernate, obtiene la estructura de objetos a partir del código de las clases para generar la estructura de la base de datos y los archivos de mapeo.

Además, los ORM están especialmente diseñados para que la base de datos se adapte y soporte el modelo orientado a objetos, no viceversa.

Capítulo 2

Marco teórico

Para el desarrollo del presente Trabajo Especial de Grado es necesario revisar los planteamientos, conceptos y/o teorías de diversos autores con la intención de ampliar los conocimientos que servirán de sustento.

2.1 Proceso de desarrollo

La figura 2.1 ilustra el proceso de desarrollo del software, que se explica a continuación.

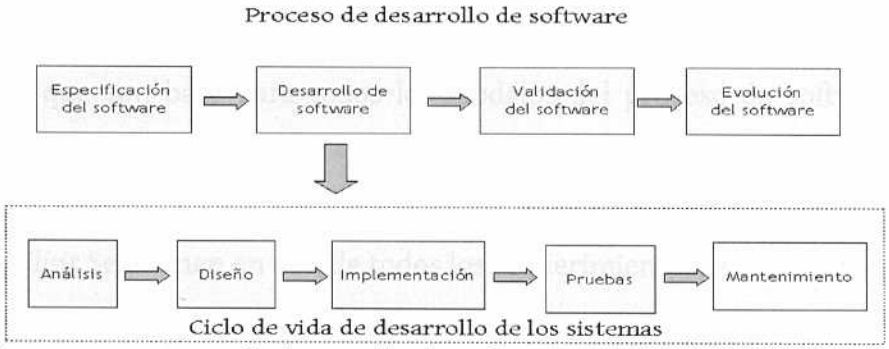

Figura 2.1: Proceso de desarrollo de software
Diseño: Lucero, K (2007)

[SOMM2002] El proceso de desarrollo es un conjunto de actividades y resultados asociados que generan un producto de software. Existen cuatro actividades fundamentales que son comunes para todos los procesos de software. Estas actividades son:

Especificación del software La funcionalidad del software y las restricciones sobre su operación deben quedar definidas.

Desarrollo de software Debe producirse software que cumpla con la especificación.

Validación del software El software debe validarse para asegurar qué es lo que el cliente requiere.

Evolución del software El software debe evolucionar para cumplir con los cambios requeridos por el cliente.

Distintos procesos del software organizan estas actividades de diferentes formas y las describen con diferente nivel de detalle. El tiempo de cada actividad varía, así como los resultados.

En la actividad de desarrollo de software, esta se subdivide en otras actividades que son base para todos los modelos del proceso de software, estas actividades son:

1. *Análisis* Se definen en detalle todos los requerimientos.

2. *Diseño* Identifica y describe las abstracciones fundamentales del sistema de

software y sus relaciones.

3. *Implementación* El diseño del software se lleva a cabo.

4. *Prueba* Se hacen las pruebas necesarias para verificar que el software cumpla con los requerimientos.

5. *Mantenimiento* Mejora la implementación de las unidades del sistema.

2.2 Ingeniería de software asistida por computadora

[CAL1999]Ingeniería asistida por computadora (CASE) es el nombre que se le da al sistema que se utiliza para ayudar a las actividades del proceso de desarrollo como análisis, diseño, la implementación de programas y las pruebas.

La tecnología CASE ayuda al proceso de desarrollo de sistemas, automatizando algunas de sus actividades.

La ingeniería de sistemas asistida por computador es la aplicación de tecnología informática a las actividades, las técnicas y las metodologías propias de desarrollo. Tiene como fin acelerar el proceso para el que han sido diseñadas, en el caso de CASE para automatizar o apoyar una o más fases del ciclo de vida del desarrollo de sistemas.

Objetivos de la ingeniería de sistemas asistida por computador [WIES]

1. Mejorar la productividad en el desarrollo y mantenimiento del software.

2. Aumentar la calidad del software.
3. Mejorar el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
4. Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
5. Automatizar, desarrollo del software, documentación, generación de código, pruebas de errores y gestión del proyecto.
6. Ayuda a la reutilización del software, portabilidad y estandarización de la documentación.
7. Gestión global en todas las fases de desarrollo de software con una misma herramienta.
8. Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

Clasificación de la ingeniería de sistemas asistida por computador

[WIES] Aunque no es fácil y no existe una forma única de clasificar las herramientas CASE, se pueden clasificar en base a los parámetros siguientes:

- Las plataformas que soportan.
- Las fases del ciclo de vida del desarrollo de sistemas que cubren.
- La arquitectura de las aplicaciones que producen.
- Su funcionalidad.

La siguiente clasificación es la más habitual basada en las fases del ciclo de desarrollo que cubren:

- Upper CASE, herramientas que ayudan en las fases de planificación, análisis de requisistos y estrategia del desarrollo, usando, entre otros diagramas UML.
- Middle CASE, herramientas para automatizar tareas en el análisis y diseño de la aplicación.
- Lower CASE, herramientas que semiautomatizan la generación de código, crean programas de detección de errores, soportan la depuración de programas y pruebas. Además automatizan la documentación completa de la aplicación. Aquí pueden incluirse las herramientas de desarrollo rápido de aplicaciones.

Por funcionalidad podríamos diferenciar algunas como:

- Herramientas de generación semiautomática de código.
- Editores UML.
- Herramientas de Refactorización de código.
- Herramientas de mantenimiento como los sistemas de control de versiones

2.2.1 Herramientas de generación de código

[VIE2006]La generación de código, data desde la existencia de los primeros compiladores, hasta la aparición de los primeros generadores de código comerciales u orientados a "usuarios finales".

En tiempos más recientes la generación de código, gracias al avance de la

ingeniería del software, se ha llevado a un nivel diferente; donde se encuentran programas generadores de pantallas, reportes y consultas, entre otros.

Para la creación de generadores de código se deben considerar los siguientes aspectos:

- La arquitectura de software para la cual se va ha desarrollar el generador.
- Las características especificas del lenguaje de programación para el cual se hará el generador.
- El lenguaje con el que se desarrollará el propio generador: Responder los interrogantes: ¿La generación de código se realizará a partir de modelos como UML? ¿La generación de código se hará a partir de las tablas de una base de datos?, ¿Se realizará un generador de código que su resultado sea fragmentos de código que son de uso más frecuente en el software?, ¿Se creará un generador de código para diferentes lenguajes?.

Para crear un generador de código se deben hacer muchas de las tareas que realizan los compiladores; algunas de estas tareas son: la búsqueda de patrones, la escritura de código, el análisis sintáctico, el análisis léxico y la optimización de código.

Estas tareas las realiza el desarrollador una vez para una arquitectura especifica.

2.3 Programación Orientada a Objetos

El presente Trabajo Especial de Grado toma como base conceptos de la programación orientada a objetos, que resulta necesario definir.

[WIES]La programación orientada a objetos (POO) es un paradigma que define los sistemas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos), comportamiento (procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

De esta forma, un objeto contiene toda la información, (los denominados atributos) que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases (e incluso entre objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos). A su vez, dispone de mecanismos de interacción (los llamados métodos) que favorecen la comunicación entre objetos (de una misma clase o de distintas), y en consecuencia, el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan (ni deben separarse) información (datos) y procesamiento (métodos).

Esto difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el

procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada se escriben funciones y después les pasan datos.

2.3.1 Objeto

[JAZA2002] Son una serie de datos con una relación lógica entre ellos, a los que se denominan variables de instancia, con las rutinas necesarias para manipularlos (métodos).

2.3.2 Clase

[JAZA2002] Es una plantilla para construir varios objetos con características similares. Los objetos se crean cuando se define una variable de una clase.

2.3.3 Encapsulamiento

[JAZA2002] Consiste en la combinación de datos y las operaciones que se pueden ejecutar sobre esos datos en un objeto, impidiendo usos indebidos al forzar que el acceso a los datos se efectúe siempre a través de los métodos del objeto.

2.3.4 Herencia

[JAZA2002] Es la capacidad para crear nuevas clases (descendientes) que se

construyen sobre otras existentes, permitiendo que éstas les transmitan sus propiedades. En programación orientada a objetos, la reutilización de código se efectúa creando una subclase que constituye una restricción o extensión de la clase base, de la cual hereda sus propiedades.

2.3.5 Polimorfismo

[JAZA2002] Consigue que un mismo mensaje puede actuar sobre diferentes tipos de objetos y comportarse de modo distinto. El polimorfismo adquiere su máxima expresión en la derivación o extensión de clases, es decir, cuando se obtienen nuevas clases a partir de una ya existente mediante la propiedad de derivación de clases o herencia.

2.4 Modelo orientado a objetos

El modelo orientado a objetos se basa en la programación orientada a objetos (POO). Por lo tanto [JAZA2002] el Modelo de objetos utiliza un Diagrama de Configuración de Clases para definir y mostrar la estructura y comportamiento de todas las clases identificadas en el dominio del problema, así como sus relaciones.

Este modelo expresa las clases del sistema con sus relaciones estructurales y herencia, atributos y operaciones.

El modelo orientado a objetos utiliza una serie de mecanismos de abstracción:

- Herencia (Generalización / Especialización)

Indica que una subclase hereda los métodos y atributos especificados por una Súper Clase, por ende la Subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la Súper Clase.

- Asociación

La relación entre clases conocida como Asociación, permite asociar objetos que colaboran entre si. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.

- Dependencia o Instanciación (uso)

Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/clase).

- Agregación

Para modelar objetos complejos, no bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres. Cuando se requiere componer objetos que son instancias de clases definidas por el desarrollador de la aplicación, tenemos dos posibilidades:

Composición: Es un tipo de relación estática, en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comúnmente llamada Composición (el Objeto base se construye a partir del objeto incluido, es decir, es "parte/todo").

Agregación: Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es

comúnmente llamada Agregación (el objeto base utiliza al incluido para su funcionamiento).

2.5 Modelo relacional

La herramienta de generación de código para persistir datos con Nhibernte realiza un proceso de mapeo de la estructura de clases a un modelo relacional, por lo tanto, resulta útil definirlo.

[ERSN1997] El modelo relacional representa la base de datos como una colección de relaciones. En términos informales, cada relación semeja una tabla o, hasta cierto punto un archivo simple. Al visualizar una relación como una tabla de valores, cada fila de la tabla representa una colección de valores de datos relacionados entre sí. Dichos valores se pueden interpretar como hechos que describen una entidad o un vínculo entre entidades del mundo real.

La estructura de datos relacional es la siguiente:

- Entidades

Una entidad es una tabla de valores, cada fila de la tabla representa una colección de valores de datos relacionados entre sí. Cada columna representa un atributo de la relación. En terminología del modelo relacional, una fila se denomina una tupla, una cabecera de columna es un atributo. El tipo de datos que describe los tipos de valores que pueden aparecer en cada columna se llama dominio.

- Relaciones

Son las reglas para referenciar dos conjuntos, los tipos de relaciones son: Uno a uno, uno a muchos, muchos a muchos, estas relaciones tienen adicionalmente una restricción de participación parcial (opcionalidad) o total (mandatoriedad).

- Restricciones

Los identificadores, o claves primarias, son un atributo único o combinación de atributos irrepetible y mandatorio. La restricción de integridad de entidades establece que ningún valor de clave primaria puede ser nulo. La restricción de clave integridad referencial, se especifica entre dos entidades para mantener la consistencia entre tuplas. Establece que una tupla en una entidad que haga referencia a otra entidad, deberá referirse a una tupla existente en esa entidad.

2.6 Object – Relational Mapping

Para entender cabalmente NHibernate es necesario conocer el funcionamiento en general de los ORM.

[CAB2005] Es una herramienta que permite persistir objetos en una base de datos relacional. Ayuda la transición de un modelo orientado a objetos a un modelo relacional. El ORM permite definir la forma en que serán transformadas las clases a las tablas de la base de datos.

Con un buen ORM se pueden tomar los objetos de una aplicación e indicarle al ORM que los persista. Este generará automáticamente todo el código SQL necesario para almacenar el objeto.

Un ORM permite cargar los objetos fácilmente y debe aportar un buen lenguaje de consultas. Una plataforma ORM ofrece entre otras las siguientes ventajas:

1. Persistencia transparante: Los objetos del dominio se abstraen de la base de datos donde son persistidos, la plataforma lo resuelve en forma automática utilizando archivos de mapeo expresados en XML.

2. Soporte de polimorfismo: Se pueden cargar jerarquías de objetos en forma polimórfica.

3. Soporte de los 3 niveles de mapeo de herencia: Se puede mapear toda una jerarquía de clases a una sola tabla, crear una tabla por cada clase concreta o crear una tabla por cada escalón de la jerarquía.

4. Soporte completo de asociaciones: Las plataformas ORM soportan el mapeo de todos los tipos de relaciones que pueden existir en un modelo de objetos del dominio (asociaciones 1..1, 1...N, N..M, unidireccionales y bidireccionales).

5. Soporte de carga de objetos Proxy: Se pueden cargar objetos que solo contengan la clave del objeto completo.

6. Soporte de caching: En el contexto de una transacción, se puede disminuir la cantidad de veces que se accede a la base de datos cacheando en memoria los objetos que son accedidos varias veces.

Soporte de múltiples dialectos SQL: La aplicación se hace independiente

NHibernate soporta herencia, polimorfismo, relaciones uno a muchos, muchos a uno y muchos a muchos, claves compuestas y colecciones de datos.

2.8 Reflexión

La solución planteada del presente Trabajo Especial de Grado, hace uso de un mecanismo para la obtención de la estructura de clases persistentes del usuario, dicho mecanismo es denominado reflexión.

La reflexión es un proceso que consiste en obtener información (la metadata) de los elementos que conforman un programa desde otro o desde si mismo.

[MSDN3]La Interfaz de Programación de Aplicaciones (API) de reflexión de la plataforma .NET está basada en un modelo jerárquico donde cada elemento de un nivel está compuesto de uno o más elementos de nivel inferior. En la figura 2.2 se ilustra el modelo jerárquico.

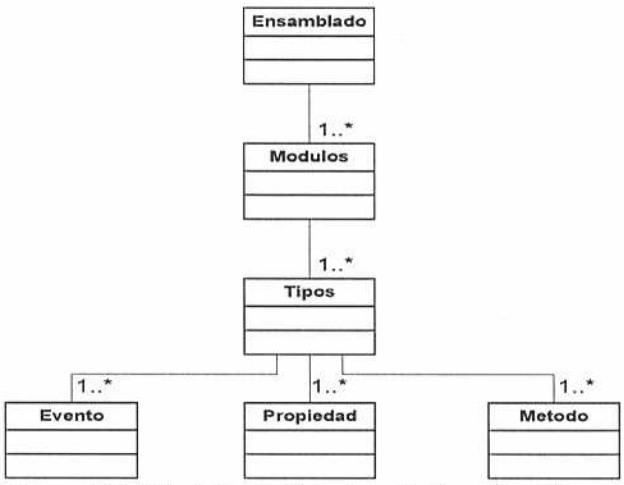

Figura 2.3: Modelo jerárquico de la reflexión para la plataforma de .NET.
Fuente: Som, G. (2004) Adaptación: Lucero, K (2007)

Un ensamblado generalmente está compuesto de uno o varios módulos, a su vez un modulo esta compuesto de uno o varios tipos, estos pueden ser clases, delegados y enumeraciones. Cada tipo suele estar compuesto de métodos, propiedades, eventos, campos, etc.

Los atributos constituyen un medio apropiado para asociar información declarativa en código C# (tipos, métodos, propiedades, etc.). Una vez asociado a una entidad del programa, el atributo se puede consultar en tiempo de ejecución.

Capítulo 3

Marco metodológico

El prototipado evolutivo es un modelo de ciclo de vida en el que se desarrolla el concepto del sistema a medida que avanza el proyecto. Normalmente se comienza desarrollando los aspectos más visibles del sistema.

La metodología de desarrollo es precisamente la de un "prototipo evolutivo". Es decir, se recorre sucesivamente el ciclo de desarrollo de software para conseguir un refinamiento progresivo del proyecto y poder ofrecer servicio desde el primer momento.

[RAN2003]Esta metodología reconoce las siguientes fases que conforman el Ciclo de Vida de desarrollo de un sistema (ver figura 3.1):

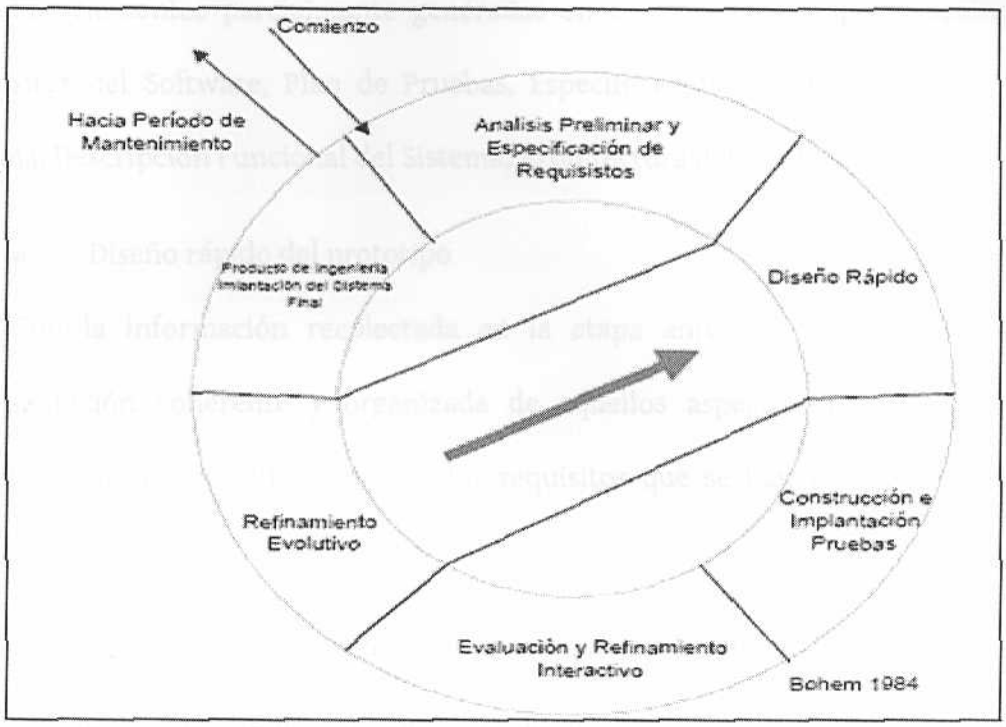

Figura 3.1: Metodología Prototipado Evolutivo.
Fuente: Rancán, J. (2003)

- Análisis preliminar y especificación de requisitos

Se identifican las necesidades, se formulan las soluciones potenciales y se evalúa la viabilidad de cada una de las soluciones para seleccionar aquella que parezca más eficiente a nivel de sistema.

Una vez establecidos los límites de la aplicación y considerando sólo aquellos requisitos globales del sistema, se analizan con detalle y refinan aquellos requerimientos que serán tratados mediante el sistema.

También dejan claramente establecidas las funciones del hardware, del software y de las interfaces, permitiendo de esa manera, desarrollar la arquitectura básica del prototipo a implementar.

Los elementos parcialmente generados en esta fase son: Especificación de Requisitos del Software, Plan de Pruebas, Especificaciones de las Interfaces del Sistema, Descripción Funcional del Sistema, Arquitectura del Sistema.

- Diseño rápido del prototipo

Con la información recolectada en la etapa anterior, se desarrolla una representación coherente y organizada de aquellos aspectos del sistema que cumplen con las especificaciones de los requisitos que se han podido recoger y refinar.

Dado que esta fase requiere la obtención de un diseño rápido para ser posteriormente implantado, las etapas de diseño de alto nivel y diseño detallado vistas en el ciclo de vida tradicional, se funden en una sola etapa de diseño.

En ella, se focaliza con la misma profundidad en las funciones y estructuras de los componentes que conforman el sistema, como en la definición detallada de los flujos datos y control.

También deben refinarse las representaciones algorítmicas que se utilizan en cada componente modular y las interfaces que requiere el prototipo a desarrollar, simulando, de resultar necesario, la presencia de otros módulos o dispositivos que a esta altura del proyecto aún no han sido desarrollados o se duda de su necesidad de implementación.

Los elementos parcialmente generados en esta fase son: Descripción de Diseño del Software, Descripción de la Arquitectura, Descripción de las Bases de Datos,

Descripción de Interfaces, Descripción de Algoritmos, Referencias cruzadas con los Requisitos, Previsiones de Pruebas.

- Construcción e implantación - Pruebas

En el modelo de Construcción de Prototipos, en esta fase se unifican las fases de Construcción y Pruebas del modelo Clásico.

En lo que respecta a construcción, al igual que en el modelo Clásico, en esta fase se traduce el diseño a un lenguaje de programación.

El código fuente debe estar acompañado de la documentación correspondiente de acuerdo a los estándares y metodologías adoptados para el proyecto.

Para el caso que el prototipo esté conformado por componentes Hardware y Software, en esta fase también se debe planificar y ejecutar la integración de ambos componentes.

Esta fase contempla también la etapa de pruebas, que constituye la revisión de las especificaciones, del diseño y de la codificación.

Para las pruebas, se ejecutará el prototipo con datos de entrada prefijados para comprobar si los resultados que el Sistema desarrollado produce son similares a los que, teóricamente y según las especificaciones, el Sistema debería producir detectando así posibles errores.

A la finalización de esta fase se obtienen los siguientes elementos: Listados y

Archivo del Código Fuente, Listados y Archivo de las Bases de Datos Generadas, Archivo Objeto Generado, Documentación del Prototipo, Documentación del Usuario (parcialmente), Plan de Integración (parcialmente), Sistema Software Integrado (parcialmente), Documentación relacionada con las herramientas (características de las máquinas empleadas para el desarrollo, compiladores, enlazadores, depuradores) utilizadas para generar el producto, Datos para la Fase de Pruebas, Especificación de las Pruebas del Prototipo, Informe Resumen de Pruebas del Prototipo.

- Evaluación y refinamiento interactivo del prototipo

Es en esta fase donde el usuario toma contacto con el prototipo para, evaluar su utilidad con datos reales, aunque sólo sea en forma parcial, para resolver el problema.

Se produce aquí un proceso interactivo entre el usuario y el desarrollador, en el cuál, a partir de la experiencia en la utilización del prototipo, se depuren conceptos, surjan nuevas ideas o se detecten fallos de interpretación en requisitos o alcances de determinados servicios que debe prestar el futuro sistema.

A la finalización de esta fase pueden suceder dos cosas:

1) Que aún no se haya llegado a satisfacer los requisitos del cliente y se disponga de recursos para continuar con el proyecto, procediéndose, entonces, a la generación de un nuevo prototipo, creando los siguientes elementos y pasando a la fase Refinamiento de las especificaciones del prototipo:

Correcciones a aplicar en: Informe preliminar de necesidades, Especificación de requisitos funcionales del Sistema, Flujo de Información, Especificación de Requisitos del Software, Especificaciones de las Interfaces del Sistema, Especificación de Nuevos Requisitos Funcionales del Sistema.

2) Que ya se haya hecho más de una iteración y se haya comprobado que el prototipo desarrollado satisface los requisitos del cliente (o se han agotado los recursos destinados al proyecto), se podrá pasar directamente a la fase Producto de Ingeniería - Implantación del Sistema final.

- Refinamiento de las especificaciones del prototipo

Con los elementos generados en la fase anterior, se realiza un refinamiento de las especificaciones que servirán como base de una nueva fase de diseño rápido, con el objeto de generar un nuevo prototipo que contemplará mayores prestaciones que el anterior.

Los elementos parcialmente generados en esta fase son: Informe preliminar de necesidades, Especificación de requisitos funcionales del Sistema, Descripción del Flujo de Información, Especificación de Requisitos del Software, Especificaciones de las Interfaces del Sistema, Especificaciones de las Interfaces del Software con otros componentes, Descripción Funcional del Sistema, Arquitectura del Sistema.

Con estos elementos se vuelve a la fase Diseño Rápido, para generar un nuevo prototipo con mayores prestaciones.

- Producto de Ingeniería - Implantación del Sistema final.

Esta fase constituye la etapa final del desarrollo del modelo de prototipado evolutivo.

Comprende las tareas de consolidación técnica del producto, pruebas, revisiones finales, instalación y operación, debiéndose adoptar las previsiones necesarias para actividades de soporte y remoción del Producto.

Esta fase se continúa hasta que el Sistema es retirado de operación para ser reemplazado por un nuevo Sistema.

Durante esta fase se producen los siguientes elementos: Documentación del Sistema, Informe Resumen de Pruebas, Documentación del Usuario, Plan de Integración, Documentación relacionada con las herramientas (características de las máquinas empleadas para el desarrollo, compiladores, enlazadores, depuradores) utilizados para generar el producto, Sistema Software Integrado, Plan de Mantenimiento, Histórico de pedidos de mantenimiento, Ordenes de Mantenimiento, Documentación relacionada con la Gestión de Configuración, Recomendaciones de Mantenimiento, Plan de retiro.

Los productos parciales de la fase Construcción e Implantación - Pruebas. Y Evaluación y refinamiento interactivo del prototipo pasan a ser finales.

Para el desarrollo de la solución del presente Trabajo Especial de Grado se seleccionó la metodología de prototipado evolutivo para tener versiones funcionales del sistema, de esta forma se puede comprobar rápidamente si se están cumpliendo o no los requisitos, para reaccionar a tiempo y corregir los errores ya

sea del diseño o de las funcionalidades.

La tabla 1 muestra el modelo metodológico implementado para el desarrollo de la solución, siguiendo las fases del prototipado evolutivo y sus actividades, junto con sus entregables.

Etapa	Actividades	Entregables
Análisis		- Diagramas de actividades - Modelo de dominio - Diagramas de casos de uso - Casos de uso
Diseño	Diseño global	- Arquitectura del sistema, Diagramas de paquetes
	Diseño detallado	- CRC (Collaboration Responsabillity Cards) - Diagramas de clases - Diagramas de secuencia
Versión 1	Construcción	Prototipo: Realizar ingeniería inversa sobre las clases persistentes y obtener la estructura de clases.
	Pruebas	Datos para la fase de pruebas, Especificación de las pruebas del prototipo
	Refinamiento	Se depuran conceptos, nuevas ideas o se detectan fallos de interpretación en requisitos
Versión 2	Construcción	Prototipo: Transformar la estructura de clases a un esquema relacional, a través de la estrategia "una tabla por clase jerárquica"
	Pruebas	Datos para la fase de pruebas, Especificación de las pruebas del prototipo
	Refinamiento	Se depuran conceptos, nuevas ideas o se detectan fallos de interpretación en requisitos
Versión 3	Construcción	Prototipo: Transformar la estructura de clases a un esquema relacional, a través de la estrategia "una tabla por clase jerárquica"
	Pruebas	Datos para la fase de pruebas, Especificación de las pruebas del prototipo
	Refinamiento	Se depuran conceptos, nuevas ideas o se

Etapa	Actividades	Entregables
		detectan fallos de interpretación en requisitos
Versión 4	Construcción	Prototipo: Transformar la estructura de clases a un esquema relacional, a través de la estrategia "una tabla por clase jerárquica"
	Pruebas	Datos para la fase de pruebas, Especificación de las pruebas del prototipo
	Refinamiento	Se depuran conceptos, nuevas ideas o se detectan fallos de interpretación en requisitos
Versión 5	Construcción	Prototipo: Transformar la estructura de clases a un esquema relacional, a través de la estrategia "una tabla por clase jerárquica"
	Pruebas	Datos para la fase de pruebas, Especificación de las pruebas del prototipo
	Refinamiento	Se depuran conceptos, nuevas ideas o se detectan fallos de interpretación en requisitos
Producto final	Comprende las tareas de consolidación técnica del producto, pruebas, revisiones finales.	Sistema de software. Documentación del Sistema, Informe resumen de pruebas, Documentación del usuario, Documentación relacionada con las herramientas utilizados para generar el producto.

Tabla 1: Esquema modelo metodológico adaptado.
Diseño: Lucero, K (2007).

La fase de diseño rápido del prototipo fue adaptada aun diseño detallado del sistema en general, por la simple razón de tener una visión más amplia de los versiones de los prototipos a construir.

La fase de Evaluación y refinamiento interactivo del prototipo fue suprimida por cuestiones de tiempo y disponibilidad de usuarios. Aunque para el producto final se llevó a cabo una prueba con un grupo de desarrolladores con un perfil específico para la evaluación.

CAPÍTULO 4

Desarrollo

El presente capítulo muestra el proceso de desarrollo de la solución, que se llevó a cabo siguiendo el modelo metodológico especificado en el capítulo anterior.

4.1 Análisis preliminar y especificación de requisitos

Antes de especificar los requisitos del sistema, es necesario aclarar que la "*Herramienta de generación de código para persistir datos con NHibernate*" esta orientada a desarrolladores de software en la plataforma .Net.

Existen actualmente varias herramientas que a partir del modelo relacional generan los archivos de mapeo de las clases e inclusive generan las clases persistentes, es decir, van del modelo relacional al modelo orientado a objetos.

Este tipo de sistemas generan un modelo que no aprovecha el potencial que ofrece un modelo orientado a objetos. Es por esta razón que la "*Herramienta de*

generación de código para persistir datos con NHibernate" parte del modelo orientado a objetos al modelo relacional.

La solución necesita una manera de conocer la estructura de las clases persistentes del usuario, debe ser una herramienta que implemente un mecanismo de ingeniería inversa sobre las clases del usuario y obtener su estructura.

También es necesario que el usuario pueda elegir una estrategia de mapeo de la herencia, e inclusive cuál dialecto SQL desea para la estructura de la base de datos, de esta manera existe la flexibilidad de escoger la estrategia y el dialecto que más se adapte a sus necesidades.

Ahora bien, la herramienta debe soportar los siguientes requisitos para mapear un objeto a un esquema de base de datos relacional:

1. Los campos de las clases de tipo primitivo (número, cadena, booleano, otros), se mapean en las columnas.
2. Mapeo de las relaciones objeto compuesto o contenido. En la figura 4.1, se observa que la clase Empleado posee una relación con la clase Nombre. Dicha relación consiste en que un objeto del tipo Nombre está contenido dentro del objeto del tipo Empleado. NHibernate se refiere a este tipo de relación como componente. Este tipo de relaciones son persistidas en la misma tabla correspondiente a la clase Empleado.

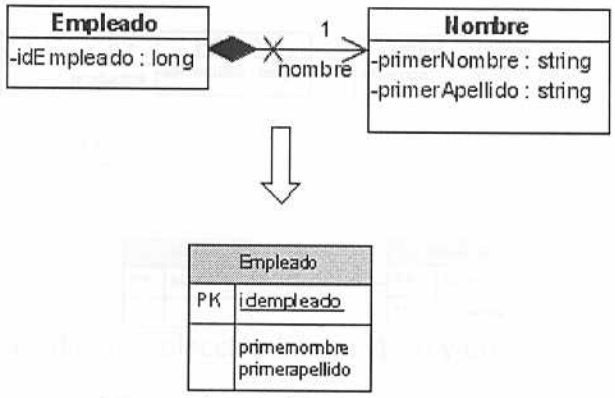

Figura 4.1: Mapeo de un objeto compuesto o contenido

3. Mapeo de las relaciones de tipo asociación simple (1..1). Las relaciones uno a uno entre objetos no es posible transformarlas al modelo relacional sin alguna modificación. El modelo equivalente es una relación 0..1, como se puede ver en la figura 4.2.

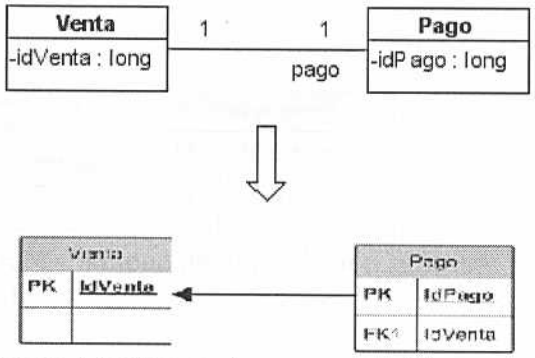

Figura 4.2: Mapeo de una asociación simple (1..1)

4. Mapeo de las relaciones referencia simple (N..1) y colección básica (1..N). Este tipo de relaciones se transforman de la misma manera al modelo relacional, y consiste en colocar una clave foránea del identificador en la

tabla cuya cardinalidad sea n. Ver figura 4.3.

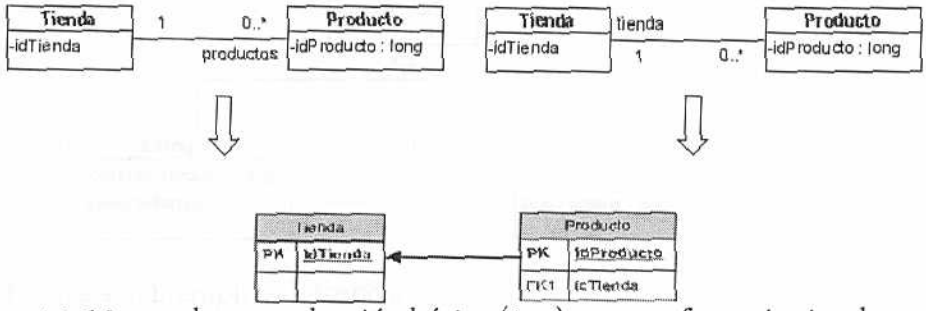

Figura 4.3: Mapeo de una colección básica (1..n) y una referencia simple (n..1)

5. Mapeo de las relaciones múltiples (N..M). Las relaciones muchos a muchos implican que se debe crear una tabla asociativa que registre todos los identificadores que guardan relación. Ver figura 4.4.

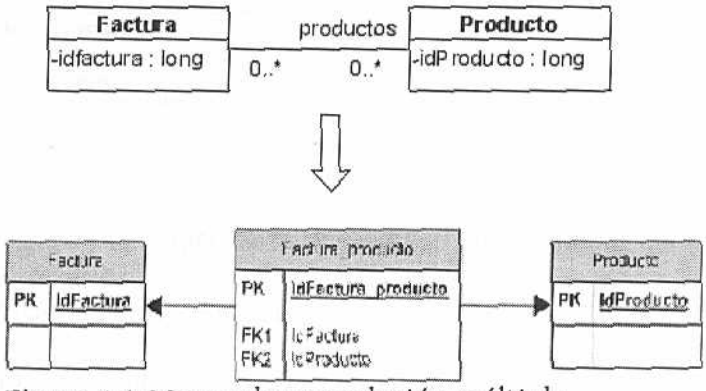

Figura 4.4: Mapeo de una relación múltiple (n..m)

6. Mapeo de la herencia. Dado un modelo de clases, ver figura 4.5. La herencia tiene varias formas de representarse en un esquema relacional.

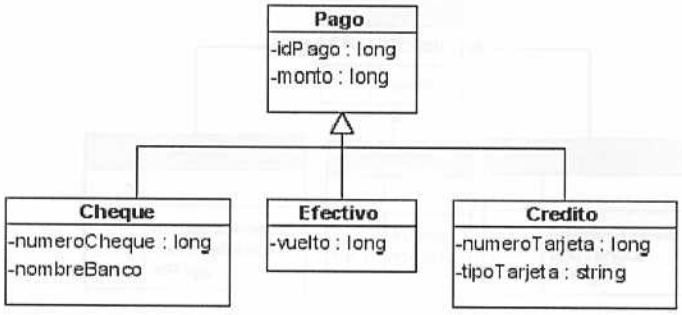

Figura 4.5: Ejemplo - herencia
Diseño: Lucero, K. (2007)

Para manejar la herencia existen tres estrategias:

6.2. Una tabla por clase jerárquica. La tabla resultante de la transformación, posee los campos y relaciones propios de la clase con la más alta jerarquía y los campos y relaciones de las subclases. Ver figura 4.6.

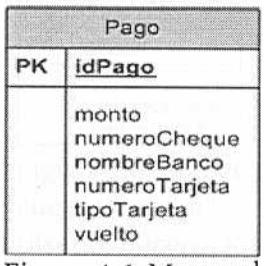

Figura 4.6: Mapeo de la herencia - "una tabla por clase jerárquica"
Diseño: Lucero, K. (2007)

6.3. Una tabla por subclase. Cada clase posee su propia tabla, pero si se desea consultar un objeto, por ejemplo del tipo Cheque, se debe hacer la unión entre la tabla perteneciente a su superclase y su propia tabla. Ver figura 4.7.

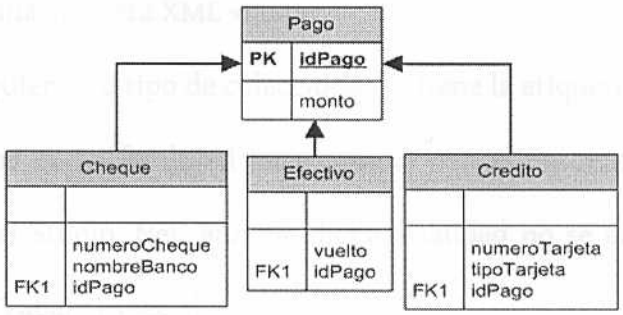

Figura 4.7: Mapeo de la herencia - "una tabla por subclase"
Diseño: Lucero, K. (2007)

6.4. Una tabla por cada clase concreta. Sólo las clases de la más baja jerarquía son las que serán persistidas en tablas. Dichas tablas poseen los campos y relaciones propios más los campos y relaciones de su superclase. Ver figura 4.8.

Figura 4.8: Mapeo de la herencia - "una tabla por clase concreta"
Diseño: Lucero, K (2007)

Otro requerimiento de la solución es la de reconocer el tipo de colección utilizada en las relaciones 1..N y M..N para poder indicarlo en los archivos de mapeo. La razón es porque NHibernate posee ciertas especificaciones para cada tipo de colección:

- Para las colecciones del tipo IList o List<> (plataforma .NET) se usa la etiqueta XML <list>
- Para las colecciones de tipo IDictionary o Dictionary<> (plataforma .NET)

se define una etiqueta XML <map>

- Para cualquier otro tipo de colecciones se tiene la etiqueta XML <bag>.

La solución tiene como finalidad incorporarse con el Entorno Integrado de Desarrollo (IDE) Visual Studio .Net, aunque dicha finalidad no se encuentra en el alcance del presente Trabajo Especial de Grado.

Como la herramienta se convertirá en un add-in de Visual Studio .Net, la misma debe consistir en una aplicación en consola, que pueda ejecutarse por medio de un comando, donde se le especifique la ruta del ensamblado, la estrategia de mapeo de la herencia, el dialecto SQL y las rutas de salida para los archivos generados.

Los archivos generados deben estar escritos en:

- Lenguaje SQL (en un dialecto específico), para la estructura de la base de datos.
- Lenguaje XML para los archivos de mapeo.

Para visualizar mejor los procesos o requerimientos del sistema se tienen los diagramas de casos de uso, en la figura 4.9 se puede ver un ejemplo.

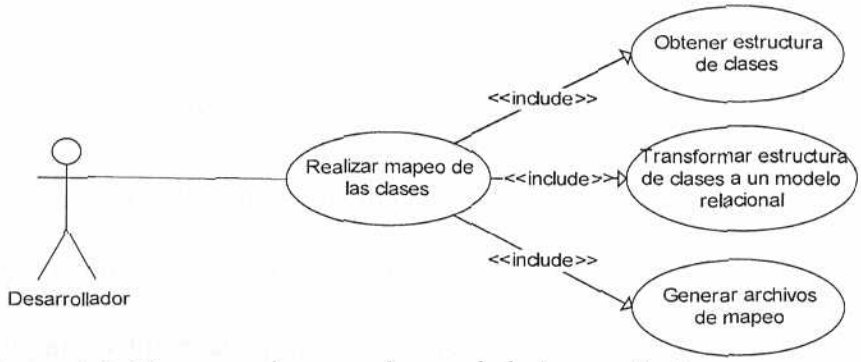

Figura 4.9: Diagrama de casos de uso de la herramienta.
Diseño: Lucero, K. (2007)

El resto de los diagramas de casos de uso se encuentran en el apéndice B

En la figura 4.10 se observa el modelo conceptual o de dominio de la herramienta donde se pueden identificar los elementos esenciales para el proceso de generación de los archivos.

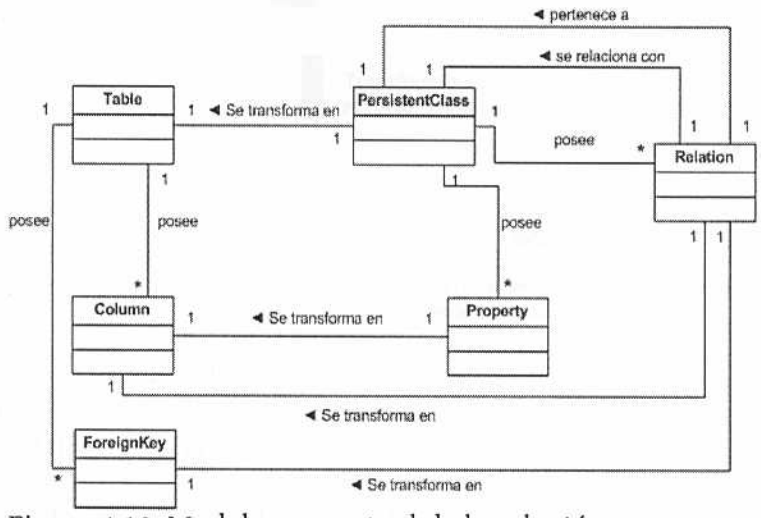

Figura 4.10: Modelo conceptual de la solución,
Diseño: Lucero, K. (2007)

4.2 Diseño global

La figura 4.11 ilustra, a modo general, el proceso de generación de código para persistir datos con NHibernate.

La solución recibe como entrada las clases persistentes del usuario, el proceso se encarga de obtener la estructura de clases y a partir de allí generar los archivos de mapeo para NHibernate, junto con la transformación de la estructura de clases al

modelo relacional para generar la estructura de la base de datos.

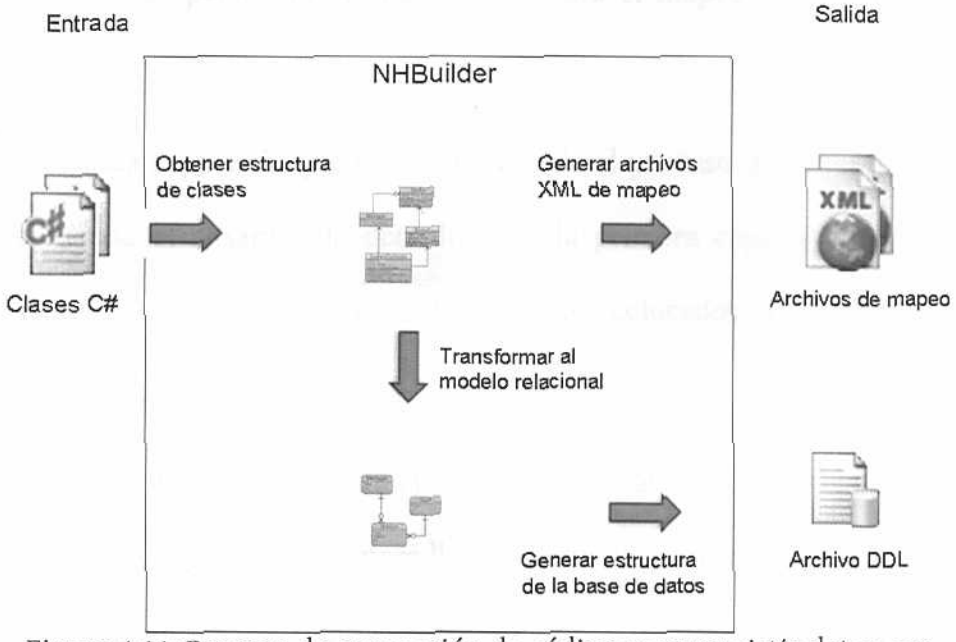

Figura 4.11: Proceso de generación de código para persistir datos con NHibernate.
Diseño: Lucero, K. (2007)

Para visualizar el flujo de control entre las actividades del proceso de generación de código, se realizaron los diagramas de actividades, ver apéndice C.

La arquitectura del sistema, para llevar a cabo el proceso de generación de código, posee tres capas como se puede ver en la figura 4.12.

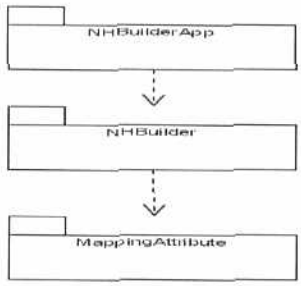

Figura 4.12: Arquitectura del sistema.
Diseño: Lucero, K. (2007)

La primera es la interfaz por el cual el usuario especifica el ensamblado que contiene las clases persistentes, la estrategia para el mapeo de la herencia y el dialecto SQL.

La segunda capa es la que lleva a cabo todo el proceso de generación de los archivos usando el ensamblado obtenido por la primera capa, y depende de la tercera capa para la identificación de los atributos colocados por el usuario en su ensamblado.

La tercera capa posee la definición de todos los atributos que el usuario debe colocar en las clases persistentes para llevar a cabo el proceso de generación de los archivos.

Con esta estructura se separa la interfaz del procesamiento, pero adicionalmente permite que la librería MappingAttribute pueda ser incluida dentro de la aplicación del usuario, para que las otras dos capas no interfieran

La segunda capa posee la siguiente estructura interna:

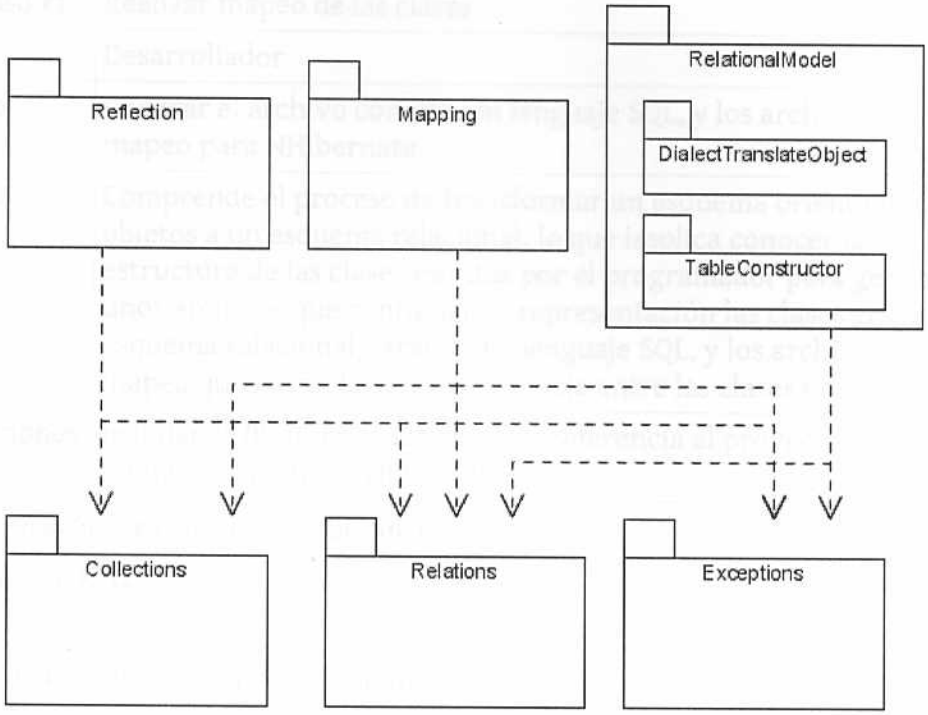

Figura 4.13: Estructura interna de la capa NHBuilder.
Diseño: Lucero, K. (2007)

4.3 Diseño detallado

Para describir la secuencia de eventos del actor y el sistema, no como actividades, sino de una forma más detallada, se tiene como ejemplo la tabla 2 que contiene la descripción del caso de uso: Realizar mapeo de las clases

Caso de uso #1	Realizar mapeo de las clases
Actor	Desarrollador
Propósito	Generar el archivo con DDL en lenguaje SQL, y los archivos de mapeo para NHibernate.
Resumen	Comprende el proceso de transformar un esquema orientado objetos a un esquema relacional, lo que implica conocer la estructura de las clases escritas por el programador para generar unos archivos que contienen la representación las clases en un esquema relacional a través del lenguaje SQL, y los archivos de mapeo que hacen la correspondencia entre las clases y las tablas.
Precondiciones	Instalar NHBuilder, y agregar una referencia al proyecto del archivo MappingAttribute.dll

Tabla 2: Caso de uso #1: Realizar mapeo de las clases.
Diseño: Lucero, K. (2007)

Para ver el resto de la descripción de este y el resto de los casos de uso ir al apéndice A.

El proceso de generación de código para persistir datos con NHibernate está compuesto principalmente por los siguientes casos de uso:

Obtener la estructura de clases consiste en un proceso de reconocimiento de las clases de persistencia y cualquier información relevante de las propiedades de dichas clases, haciendo uso de la reflexión de la plataforma .Net.

Para ello se deben definir los atributos que servirán para identificar toda la información necesaria para el proceso de generación de código, ver tabla 3.

Para mayor información en cuanto a la reflexión y los atributos ver apéndices P y Q. Y para conocer mejor acerca de las propiedades de una clase ver apéndice O.

Atributo	Valor	Interpretación
Persistent		Indica si una clase o estructura es persistente.
Component		Indica si una clase o estructura debe ser persistida como un objeto compuesto de otra clase.
Id		Indica si una propiedad de la clase es el identificador de la tabla.
NotNull		Indica si una propiedad tiene la restricción Not Null, si no se indica se asume que el campo puede ser Null.
Length	número	Indica el tamaño de una propiedad del tipo string.
Default	string	Indica si una propiedad posee un valor por defecto.
Unique		Indica si una propiedad posee la restricción Unique.
NonPersistent		Indica si una propiedad de la clase no se desea persistir.
OneToOne		Sirve para indicar una relación uno a uno.
ManyToOne		Sirve para indicar una relación muchos a uno.
OneToMany	Type (opcional)	Sirve para indicar una relación uno a muchos. Si la colección no es genérica debe indicarse la clase con la que se relaciona.
ManyToMany	Type (opcional)	Sirve para indicar una relación muchos a muchos. Si la colección no es genérica debe indicarse la clase con la que se relaciona.

Tabla 3: Listado de atributos.

Diseño: Lucero, K. (2007)

Transformar la estructura de clases a un modelo relacional, implementa las estrategias para el mapeo de las clases con sus campos y relaciones con otras clases.

Para el mapeo de los tipos de datos soportados por la plataforma .NET es necesario conocer su correspondiente tipo de dato tanto en el manejador

seleccionado (en este caso Microsoft SQL Server 2005). Para ello se tiene la tabla 4.

.Net Type	MS SQL Server Type
Boolean	bit
Byte	tinyint
DateTime	datetime
Decimal	decimal
Double	float
Int16	smallint
Int32	int
Int64	bigint
String	nvarchar
Single	real

Tabla 4: Tipos de datos de la plataforma .NET vs, Tipos de datos de Microsoft SQL
Fuente: Server Data Types and Their .NET Framework Equivalents

Para ver detalladamente las estrategias de transformación de las clases al esquema relacional, ir al apéndice G.

Generar los archivos de mapeo, consiste en tomar la estructura de clases y generar los archivos en formato XML, según las especificaciones de NHibernate, para cada una de las distintas relaciones entre clases incluyendo las distintas estrategias para el manejo de la herencia, y los tipos de datos.

La siguiente tabla ilustra los tipos de datos soportados por la plataforma .Net y su correspondiente tipo de dato de mapeo para NHibernate.

.Net Type	NHibernate Type
Boolean	boolean
Byte	byte
DateTime	datetime
Decimal	decimal
Double	float
Int16	int16
Int32	int32
Int64	int64
String	string
Single	single

Tabla 5: Tipos de datos de la plataforma .NET vs, Tipos de datos de NHibernate.
Fuente: NHiberbnate Reference 1.0.2.

Para ver detalladamente los formatos de los archivos de mapeo según la estrategia y las especificaciones de NHibernate, ir al apéndice G.

Para el diseño de la solución en lo que respecta a las clases, se procedió a la realización de una técnica llamada Collaboration Responsability Cards (CRC) de esta forma las clases se filtran y se refinan en base a sus responsabilidades con respecto al sistema, y las clases con las que necesitan colaborar para completar sus responsabilidades. Ir al apéndice N.

Una vez definidas las clases y sus responsabilidades se diseñó la estructura de clases del sistema. Los diagramas de clases se encuentran en el apéndice D.

Luego se elaboraron los diagramas de secuencia para poder observar la interacción de los objetos que intervienen en cada caso de uso. Ver apéndice E.

4.4 Prototipos

Esta fase contempla una serie de iteraciones, divididas en función de los casos de uso. Cada iteración tuvo como producto un prototipo funcional y la refinación del diseño del producto final.

Versión 1: Consistió en desarrollar un prototipo que lograra hacer reflexión sobre el ensamblado que contiene las clases persistentes, para obtener la información de cada una de ellas.

La forma de identificar a las clases persistentes es haciendo uso de los atributos descritos en el diseño detallado. Así como también existe información relevante de las propiedades de las clases persistentes que se obtiene cuando el usuario coloca los atributos correspondientes.

El refinamiento de las especificaciones del prototipo, consistió en hacer que el proceso de reflexión tomara en cuenta las propiedades de la clase y no los campos, como fue previsto antes de desarrollar el prototipo.

En esta versión se implementaron las siguientes capas de la arquitectura del sistema, ver figura 4.14:

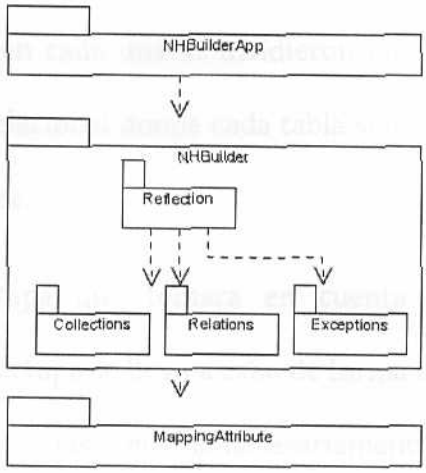

Figura 4.14: Capas implementadas en la primera versión.
Diseño: Lucero, K. (2007)

Versión 2, 3 y 4: Se implementó la transformación de la estructura de clases al modelo relacional, específicamente la construcción estuvo enfocada en la capa NHBuilder (ver figura 4.15).

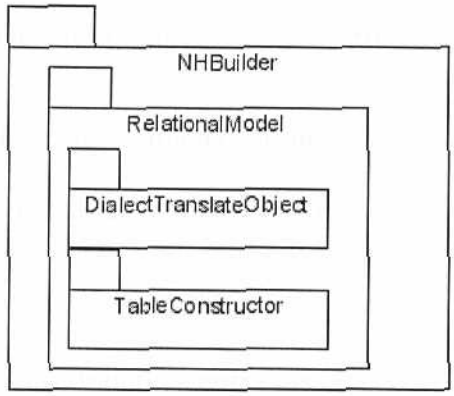

Figura 4.15: Capas implementadas en la segunda, tercera y cuarta versión.
Diseño: Lucero, K. (2007)

Se realizó una versión del prototipo por cada una de las estrategias de mapeo de herencia, donde en cada una se dividieron en etapas, la primera consistió en generar el modelo relacional donde cada tabla solo debía contener las propiedades de tipo base de la clase.

Luego otra etapa que tomara en cuenta las relaciones de las clases persistentes, en esta etapa se llevo a cabo de forma muy meticulosa, ya que pueden existir relaciones entre clases que no necesariamente poseen una tabla, sino que las relaciones deben ser modificadas para que referencien a la tabla de su superclase o subclase (según la estrategia).

La última etapa consistió en transformar todas las tablas a su correspondiente sentencia SQL según el dialecto seleccionado, y finalmente generar el archivo donde deben ser escritas dichas sentencias.

El refinamiento para cada versión consistió en la corrección de errores de sintaxis de los archivos SQL generados por los prototipos.

Versión 5: Se implementó el mapeo de las clases con sus propiedades y las relaciones, a los archivos de mapeo de NHibernate.

En este punto no es necesario transformar nada al modelo relacional, pero si se debe mantener una correspondencia entre las clases y el modelo relacional.

El prototipo genera por cada clase (según la estrategia) su correspondiente archivo de mapeo de NHibernate.

Las capas implementadas fueron las que se encentran en la figura 4.16.

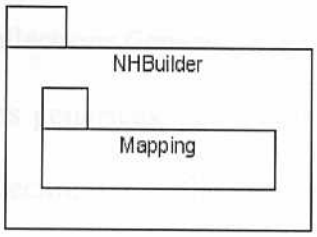

Figura 4.16: Capas implementadas en la quinta versión.
Diseño: Lucero, K. (2007)

El refinamiento de las especificaciones y los requisitos, tuvo como propósito la corrección de errores detectados en las pruebas del prototipo.

Las librerías de la plataforma .NET utilizadas en el desarrollo de la herramientas son:

- System, contiene clases fundamentales y clases base que definen tipos de datos de referencia y de valor de uso frecuente, eventos y controladores de eventos, interfaces, atributos y excepciones de procesamiento.
- System.Text, contiene clases que representan codificaciones de caracteres ASCII, Unicode, UTF-7, y UTF-8; clases base abstractas para la conversión de bloques de caracteres en bloques de bytes y viceversa; y una clase auxiliar que manipula y da formato a objetos String sin necesidad de crear instancias intermedias de String.
- System.Reflection, contiene clases e interfaces que proporcionan una

vista administrada de los campos, los métodos y los tipos cargados, con la posibilidad de crear e invocar tipos dinámicamente.

- System.Collections.Generic, contiene interfaces y clases que definen colecciones genéricas, permitiendo que los usuarios creen colecciones con establecimiento inflexible de tipos para proporcionar una mayor seguridad de tipos y un rendimiento mejor que los de las colecciones no genéricas con establecimiento inflexible de tipos.
- System.IO contiene tipos que permiten leer y escribir en los archivos y secuencias de datos, así como tipos que proporcionan compatibilidad básica con los archivos y directorios.
- System.XML, proporciona compatibilidad basada en estándares para procesar XML.

En cada iteración en el proceso de construcción e implementación se llevaron a cabo pruebas para comprobar el funcionamiento adecuado de los prototipos.

4.5 Pruebas

Las pruebas caja negra, fueron pruebas que implicaron el desarrollo de otros proyectos en .Net con estructuras de clases lo más cercanas a la realidad, que presentaran jerarquías de clases, distintas relaciones entre clases, y distintos tipos de datos. Por ejemplo:

Prueba #1 Curso normal

Descripción:: La prueba consiste en crear una librería de clases persistentes donde exista herencia entre clases, distintos tipos de relaciones y diferentes tipos de datos. Generar el dll de la librería de clases, y luego ejecutar la herramienta para generar los archivos. Una vez generados los archivos, crear la base de datos en el manejador (Microsoft SQL Server 2005 Express Edition), insertar varios registros, luego crear una pequeña aplicación donde se utilice NHibernate para consultar datos en la base de datos, usando la librería de clases y los archivos de mapeo generados.

Entrada: Ruta donde se encuentra el ensamblado (dll) con las clases persistentes, dialecto SQL (sqlserver), estrategia de mapeo de la herencia, ruta para el archivo ddl y la ruta para los archivos XML.

Salida: Para la herramienta de generación de código la salida son El archivo ddl y los archivos XML, para comprobar el buen funcionamiento de los archivos la aplicación para consultar los datos usando NHibernate debe mostrar los datos obtenidos de la base de datos.

Para ver detalladamente la descripción de las pruebas ver el Apéndice J.

Para ver los diagramas de clases para las pruebas y los resultados obtenidos, ver el apéndice K.

Para cada versión estas pruebas fueron adaptadas, según la funcionalidad esperada:

Versión 1: Las pruebas consistieron únicamente en obtener la estructura de clases para los distintos diagramas de clases de prueba.

Versión 2: Las pruebas consistieron en obtener la estructura de clases, para los distintos diagramas de clases de prueba, transformar la estructura al modelo relacional y generar su archivo DDL (Data Definition Language) usando la estrategia de mapeo de la herencia "una tabla por clase jerárquica".

Versión 3: Las pruebas consistieron en obtener la estructura de clases, para los distintos diagramas de clases de prueba, transformar la estructura al modelo relacional y generar su archivo DDL (Data Definition Language) usando la estrategia de mapeo de la herencia "una tabla por subclase".

Versión 4: Las pruebas consistieron en obtener la estructura de clases, para los distintos diagramas de clases de prueba, transformar la estructura al modelo relacional y generar su archivo DDL (Data Definition Language) usando la estrategia de mapeo de la herencia "una tabla por cada clase concreta".

Versión 5: Las pruebas consistieron en obtener la estructura de clases, para los distintos diagramas de clases de prueba, transformar la estructura al modelo relacional, generar su archivo DDL (Data Definition Language) y generar los archivos de mapeo. Para cada una de las diferentes estrategias de mapeo de la herencia.

Es importante resaltar que se hicieron pruebas unitarias en todo el proceso de construcción de los prototipos, que es una forma de probar individualmente el

correcto funcionamiento de cada una de las clases pertenecientes a la herramienta.

4.6 Producto final

El producto final es una herramienta Lower CASE, para la generación de código. Encargada de automatizar la generación de la estructura de la base de datos y la generación de los archivos XML de mapeo para NHibernate, dado el ensamblado con las clases persistentes.

Además de la solución, se tienen los siguientes entregables como:

- Documentación de las herramientas de programación utilizadas. Ver apéndice F.
- Documentación de los estándares de codificación utilizados. Ver apéndice H.
- Estándares de codificación usados para el archivo con el DDL. Ver apéndice M.
- Manual del usuario, ver apéndice I.
- Documentación de la herramienta, ver apéndice R.
- Pruebas realizadas, ver apéndice J.
- Diagramas de clases usados para las pruebas y sus resultados, ver apéndice L.

Ahora bien, la prueba de evaluación del usuario del producto final, consistió en:

Para dos usuarios cuyo perfiles son los siguientes:

	.Net y C#	MS SQL Server	SQL Server Managment Studio	Nhibernate
Usuario 1	✓	✓	-	+/-
Usuario 2	✓	✓	✓	✓

Tabla 6: Perfiles de los usuarios para la evaluación del producto final
Diseño: Lucero, K. (2007)

A ambos se les proporcionó una aplicación que contenía solo las clases con la misma estructura, ver figura K.2 del apéndice K.

La prueba se dividió en dos fases:

	Descripción
Fase 1	Sin hacer uso de la herramienta: Se creó la base de datos a través del ambiente SQL Server Managment Studio, y se escribieron los archivos de mapeo.
Fase 2	Haciendo uso de la herramienta: Se le colocaron los atributos a las clases persistentes y se ejecuto la herramienta.

Tabla 7: Fases de la prueba para la evaluación del producto final
Diseño: Lucero, K. (2007)

La estrategia de mapeo de la herencia fue "una tabla por subclase".

Los resultados de las pruebas se encuentran en el apéndice L.

Con estos resultados se puede concluir, que en efecto, la herramienta automatiza el procedimiento de generación de archivos para la implantación de NHibernate, ahorrando un tiempo considerable, además que los archivos de mapeo generados no dieron ningún conflicto de sintaxis con NHibernate, y para todos los casos se pudieron crear las bases de datos sin ningún conflicto.

Resultados

Los resultados obtenidos en el proceso de desarrollo de la herramienta, fueron los siguientes:

Se diseñó e implementó un mecanismo con para la transformación dada una estructura de clases a un esquema relacional

Dicho mecanismo, una vez obtenido el modelo relacional crea la estructura de la base de datos (DDL) usando el dialecto SQL elegido por el usuario.

La solución lleva a cabo un proceso de generación de archivos de mapeo partiendo de la estructura de clases. La herramienta le permite al usuario especificar las clases persistentes e indicar información relevante para la generación de los archivos. Además se le da la posibilidad al usuario de escoger la estrategia de mapeo de la herencia.

El proceso encargado de explorar la estructura de clases del usuario se implementó usando la reflexión de la plataforma de .Net. Este proceso reconoce las relaciones entre clases, y las manipula, según la estrategia de mapeo de la herencia,

para que exista coherencia entre las tablas y las columnas de la base de datos con sus correspondientes clases persistentes y sus propiedades.

En forma general, se obtuvo una herramienta CASE, como bien se sabe, de generación de código, pero no fue sólo eso, sino que se logró un sistema que aporta un valor adicional, y es el de automatizar un proceso complicado, convirtiéndolo en un proceso más llevadero y más factible de cumplir en el tiempo estipulado.

El resultado final es una herramienta que le proporciona al desarrollador el apoyo necesario para incorporar NHibernate a su aplicación de la manera más transparente y natural.

Conclusiones

La ingeniería del software evoluciona cada día, esta evolución conlleva a la creación de métodos mucho más cómodos, precisos, confiables y rápidos para la creación de software.

Los avances de la ingeniería de software, permiten que la construcción de aplicaciones pueda ser planificable y repetible, trayendo como consecuencia que los sistemas sean de mejor calidad.

La ingeniería de software busca siempre automatizar aquellas tareas repetitivas en el proceso de desarrollo, a través de las herramientas CASE. Quizás un pensamiento utópico es el de poder generar sistemas de software, o por lo menos la mayoría, a través de estas herramientas. Pero cada día se trabaja para llegar a ello, creando herramientas que agilicen aquellos procesos ineficientes pero que son necesarios en el proceso de desarrollo de software.

La herramienta desarrollada, es una herramienta CASE de generación de código, que facilita el proceso de implantación de NHibernate, a las aplicaciones

desarrolladas en la plataforma .Net.

Utilizar esta herramienta trae como beneficios: primero abstraerse del diseño de la base de datos, segundo no caer en complicaciones aprendiendo las especificaciones de NHibernate para los archivos de mapeo, que a la vez tiene como consecuencia, el ahorro de tiempo. Tercero: el desarrollador puede asegurase que la estructura de la base de datos fue diseñada de tal forma que cumpla con las reglas del modelo relacional y cuarto que los archivos están escritos correctamente, para el buen desempeño de NHibernate.

El proceso de implantación de una herramienta ORM para un sistema, es prácticamente repetible y manual. Con la herramienta desarrollada se logra la automatización de este proceso consiguiendo que el desarrollador que utilice la herramienta lo ayude a ser más eficiente.

La herramienta desarrollada aporta su grano de arena para que la ingeniería de software siga evolucionando en pro de la creación de software de la mejor calidad.

Recomendaciones

Existen un serie de recomendaciones para la herramienta de generación de código para persistir datos con Nhibernate, que de ser implementadas aportarían más valor a la solución:

- La herramienta fue concebida desde el principio como una aplicación de consola, pensando a futuro la posibilidad de integrar la herramienta a Visual Studio .Net.
- La incorporación de la misma cantidad de dialectos SQL soportados por NHibernate, para darle más libertad al usuario, además que este punto puede ser decisivo al momento de usar la herramienta.
- Las versiones posteriores de NHibernate pueden incorporar nuevas funcionalidades y mejoras, por lo tanto se debe estar atento a los posibles cambios, porque la herramienta podría quedar obsoleta.
- Existen funcionalidades y parámetros secundarios que proporciona NHibernate para los archivos de mapeo, agregar esto le aportaría mucho

valor a la herramienta.

- Para el archivo generado que contiene la estructura de la base de datos (DDL) sería recomendable ordenar las tablas según las restricciones de clave foránea.
- Las excepciones arrojadas por la herramienta podrían ser capturadas y almacenadas, para que el proceso siga adelante y detecte la mayor cantidad de errores, y mostrar al final el listado de excepciones.

Bibliografía

- [CAB2005] Cabrera, M.(2005). Introducción a NHibernate, [en línea]. Disponible en: http://mcabrera.datacenter1.com/articles/dotNET/nhibernate/
- [CAL1999] Calderi, Irivict y Moises Puyosa(1999). Herramientas CASE, [en línea]. Disponible en: http://www.geocities.com/SiliconValley/Bit/6238/index.htm
- [CAS2004] Castro Unai(2004). Emisión dinámica de ensamblados, [en línea]. Disponible en: http://www.elguille.info/colabora/puntoNET/unai_EmitAndReflecting.htm
- [ERSN1997] Elmasri, Ramez y Shamkant Navathe(1997). Sistemas de bases de datos.Pearson Educación .
- [GAR2003] García, A.(2003). Guía del autoestopista a Hibernate, [en línea]. Disponible en: http://www.javahispano.org/articles.article.action?id=80
- [GONDA2003] Gonda, Breogán(2003). ¿Desarrollo orientado a procesos u orientado a datos?, [en línea]. Disponible en: http://scholar.google.com/url?sa=U&q=http://200.40.205.140/download/files/7542462124_1163525534/desarrollo%2520orientado%2520a%2520procesos%2520u%2520orientado%2520a%2520datos.pdf
- [JAZA2002] Joyanes Aguilar y Luis Ignacio Zahonero Martínez(2002). Programación en Java2.Mc Graw Hill .

- [JOH2005] Johnson, Rod y Jim Clark(2005). Persistencia y POJOs: La Unión de los Mundos de Objetos y Relacional, [en línea]. Disponible en: http://www.programacion.net/java/articulo/jap_j2eemaster_9/
- [LANDGR] Landeros Gómez, R.(2006). Herramientas CASE, [en línea]. Disponible en: http://www.monografias.com/trabajos14/herramicase/herramicase.shtml
- [LARMC2001] Larman, Craig(2001). UML y patrones.Pearson .
- [LIN2006] Grupo Linuxero(2006). El problema de la diferencia de impedancia, [en línea]. Disponible en: http://www.glib.org.mx/article.php?story=20060611151541892
- [LOP2005] Lopez, A. J.(2005). Material del Seminario Object Relational Mapping en.NET, [en línea]. Disponible en: http://www.ajlopez.net/ReferenciaVe.php?Id=56
- [MART2006] Martínez, R(2006). Mapeadores Objeto-Relacionales (ORMs) y su utilización en el desarrollo de aplicaciones Web, [en línea]. Disponible en: http://dei.uca.edu.sv/dei/html/download/ciclo_02-2006/alicia/practica_especialidad/08-Mapeador%20objeto%20relacional.doc
- [MSDN1] MSDN(2005). Server Data Types and Their .NET Framework Equivalents, [en línea]. Disponible en: http://msdn2.microsoft.com/enus/library/ms131092.aspx
- [MSDN2] MSDN(2005). Data Type Conversion (Database Engine), [en línea]. Disponible en: http://msdn2.microsoft.com/en-us/library/ms191530.aspx
- [MSDN3] MSDN(2005). Reflexión, [en línea]. Disponible en: http://msdn2.microsoft.com/es-es/library/cxz4wk15(VS.80).aspx
- [MSDN4] MSDN(2005). Información general sobre la reflexión, [en línea]. Disponible en: http://msdn2.microsoft.com/es-es/library/f7ykdhsy(VS.80).aspx
- [MSDN5] MSDN(2005). Conferencias Técnicas gratuitas MSDN 2005: Desde el día a día,algunos consejos de sistemas reales, [en línea]. Disponible en: http://www.microsoft.com/spanish/msdn/argentina/conferencias_tecnicas/

anteriores/2005_Julio.asp

- [MSDN6] MSDN(2005). Información general sobre la reflexión , [en línea]. Disponible en: http://msdn2.microsoft.com/es-es/library/f7ykdhsy(VS.80).aspx
- [MSDN7] MSDN(2005). Propiedades (Guía de programación de C#) , [en línea]. Disponible en: http://msdn2.microsoft.com/es-es/library/x9fsa0sw(VS.80).aspx
- [MSDN8] MSDN(2005). Acceso a atributos mediante reflexión (Guía de programación de C#) , [en línea]. Disponible en: http://msdn2.microsoft.com/es-es/library/z919e8tw(VS.80).aspx
- [NHDOC] NHibernate.org(2006). NHibernate Documentation 1.0.2.0, [en línea]. Disponible en: http://www.hibernate.org/hib_docs/nhibernate/html/
- [NHREF] NHiberbnate.org(2006). NHiberbnate Reference 1.0.2., [en línea]. Disponible en: http://www.hibernate.org/hib_docs/nhibernate/nhibernate_reference.pdf
- [RAE] Real Academia Española(2004). DICCIONARIO DE LA LENGUA ESPAÑOLA, [en línea]. Disponible en: http://www.rae.es/
- [RAN2003] Rancán, J. C.(2003). Gestión de Configuración de Productos Software en Etapa de Desarrollo, [en línea]. Disponible en: http://www.itba.edu.ar/capis/epg-tesis-y-tf/rancan-trabajofinaldeespecialidad.pdf
- [ROS2003] Rosés Albiol, F.(2003). Introducción a Hibernate, [en línea]. Disponible en: http://www.javahispano.org/articles.article.action?id=93
- [RUB1998]: Rubin, David M., Introduction to CRC Cards, 1998. Softstar Research, Inc.
- [SCH2002] Schildt, Herbert(2002). C# Manual de Referencia.Mc Graw hill .
- [SED2002] Sedge, Tom(2002). Index of Relationships, [en línea]. Disponible en: http://www.xylax.net/hibernate/index.html
- [SOM2004] Som, Guillermo.(2004). Enumerar los miembros de una clase (tipo), usandoReflection, [en línea]. Disponible en:

http://www.elguille.info/NET/dotnet/reflectionInfoTipo.htm

- [SOMG2004] Som, Guillermo.(2004). Enumerar los miembros de una clase (tipo), usandoReflection, [en línea]. Disponible en: http://www.elguille.info/NET/dotnet/reflectionInfoTipo.htm
- [SOMM2002] Sommerville, I.(2002). Ingeniería de Software.Addison-Wesley .
- [SUA2003] Suárez González, H.(2003). Manual Hibernate, [en línea]. Disponible en: http://www.javahispano.org/articles.article.action?id=82
- [SUNMI] Sun Microsystems(2002). Core J2EE Patterns - Data Access Object, [en línea]. Disponible en: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html
- [VIE2006] Viera, Victor(2006). Generadores de código, [en línea]. Disponible en: http://www.mailxmail.com/curso/informatica/generadores
- [WIES] wikipedia.org(). Wikipedia, la enciclopedia libre!!, [en línea]. Disponible en: http://es.wikipedia.org/wiki/Portada

Glosario

API, Interfaz de programación de aplicaciones.

Cache, es un conjunto de datos duplicados de otros originales, con la propiedad de que los datos originales son costosos de acceder, normalmente en tiempo, respecto a la copia en el caché. Cuando se accede por primera vez a un dato, se hace una copia en el caché; los accesos siguientes se realizan a dicha copia, haciendo que el tiempo de acceso medio al dato sea menor.

Cardinalidad, indica el número o cantidad de los elementos constitutivos de un conjunto.

Clase, declaración o abstracción de un objeto cuando se programa según el paradigma de orientación a objetos.

Clase persistente, o clases del negocio es cuando los objetos pertenecientes a esta clase son almacenados en una o varias tablas de una base de datos relacional.

DAO (Data Access Object) es un patrón de diseño para separar el mecanismo

de acceso a la fuente de datos.

El acceso al almacenamiento persistente, como una base de datos, varía en gran medida dependiendo del tipo de almacenamiento (bases de datos relacionales, bases de datos orientadas a objetos, archivos planos, etc.) y de la implementación del vendedor.

Los componentes necesitan ser transparentes al almacenamiento persistente real o la implementación de la fuente de datos para proporcionar una migración sencilla a diferentes productos, diferentes tipos de almacenamiento y diferentes tipos de fuentes de datos.

Al utilizar un Data Access Object (DAO) se puede abstraer y encapsular todos los accesos a la fuente de datos. DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.

DDL. Lenguaje de definición de datos, es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios de la misma llevar a cabo las tareas de definición de las estructuras que almacenarán los datos así como de los procedimientos o funciones que permitan consultarlos.

Dialecto, 1. m. Ling. Sistema lingüístico considerado con relación al grupo de los varios derivados de un tronco común. El español es uno de los dialectos nacidos del latín. 2. m. Ling. Sistema lingüístico derivado de otro, normalmente con una concreta limitación geográfica, pero sin diferenciación suficiente frente a otros de origen común. 3. m. Ling. Estructura lingüística, simultánea a otra, que no alcanza la

categoría social de lengua.

Dialecto SQL, dialecto derivado del lenguaje SQL, cada sistema manejador de base de datos(SMBD) posee su propio dialecto SQL.

La versión de Nhibernate 1.0.2 soporta los siguientes dialectos:

SMBD	Dialecto
DB2	`NHibernate.Dialect.DB2Dialect`
PostgreSQL	`NHibernate.Dialect.PostgreSQLDialect`
MySQL	`NHibernate.Dialect.MySQLDialect`
Oracle (cualquier versión)	`NHibernate.Dialect.OracleDialect`
Oracle 9/10g	`NHibernate.Dialect.Oracle9Dialect`
Sybase	`NHibernate.Dialect.SybaseDialect`
Microsoft SQL Server 2000	`NHibernate.Dialect.MsSql2000Dialect`
Microsoft SQL Server 7	`NHibernate.Dialect.MsSql7Dialect`
Firebird	`NHibernate.Dialect.FirebirdDialect`
SQLite	`NHibernate.Dialect.SQLiteDialect`

Tabla 8: Dialectos SQL soportados por NHibernate 1.0.2.

Fuente:NHibernate Documentation 1.0.2.0

Diferencial de impedancia, se refiere al conjunto de dificultades conceptuales y técnicas que ocurren frecuentemente cuando un programa está escrito usando la programación orientada a objetos y debe hacer uso de un sistema de gestión de base de datos relacional.

DLL, es el acrónimo de Dynamic Linking Library (Librería 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.

Mapeo, convertir objetos en tablas. Según la Real Academia Española, 1. tr. *Biol.* Localizar y representar gráficamente la distribución relativa de las partes de un todo; como los genes en los cromosomas. 2. tr. cult. *Chile.* Hacer mapas. 3. tr. cult. *Chile.* Trasladar a un mapa sistemas o estructuras conceptuales.

Modelo Relacional, para la gestión de una base de datos es un modelo de datos basado en la lógica de predicado y en la teoría de conjuntos. Este modelo considera la base de datos como una colección de relaciones. De manera simple, una relación representa una tabla, en que cada fila representa una colección de valores que describen una entidad del mundo real.

NHibernate, herramienta ORM para la integración con la plataforma .NET

Objeto, un objeto es cualquier cosa, real o abstracta, acerca de la cual almacenamos datos y los métodos que controlan dichos datos.

ORM (Object-Relational-Mapping) Es una herramienta que permite persistir objetos en una base de datos relacional. Ayuda la transición de un modelo orientado a objetos a un modelo relacional. El ORM permite definir la forma en que serán transformadas las clases a las tablas de la base de datos.

Patrones de diseño. El patrón es una descripción del problema y la esencia de

su solución de tal forma que ésta se pueda reutilizar en diferentes casos. Es una solución adecuada para un problema común. A este respecto, los patrones se utilizan durante el análisis para desarrollar modelos de sistemas, así como durante el proceso de diseño.

Los patrones resuelven problemas de diseño específicos y hacen el diseño orientado a objetos más flexible, elegante y en última instancia reusable. Ayudan a los diseñadores a reutilizar diseños exitosos basando el nuevo diseño en experiencias anteriores. Un diseñador familiarizado con tales patrones puede aplicarlos de manera inmediata a problemas de diseño sin tener que redescubrilos

Persistencia, (Del lat. Persistĕre). 1. intr. Mantenerse firme o constante en algo. 2. intr. Durar por largo tiempo.

Programación Orientada a Objetos, es un paradigma que define a los sistemas en términos de "clases de objetos", objetos que son entidades que combinan *estado* (es decir, datos), *comportamiento* (esto es, procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto). La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

Propiedad, se pueden utilizar tanto propiedades como campos para almacenar información en un objeto. Mientras que los campos son simplemente variables públicas, las propiedades utilizan procedimientos de propiedad para controlar cómo

se establecen y devuelven valores. Los procedimientos de propiedad son bloques de código declarados dentro de las definiciones de propiedad que puede ejecutar código cuando se establece o recupera el valor de una propiedad. Los procedimientos de propiedad son: **Get** para recuperar un valor de una propiedad y **Set** para asignar un valor a una propiedad.

Proxy, La palabra proxy se usa en muchas situaciones en donde tiene sentido un intermediario. También se puede traducir por delegado o apoderado.

SQL, Structured Query Language, lenguaje de consulta estructurado, es un lenguaje declarativo de acceso a bases de datos relacionales relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Aúna características del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar información de interés de una base de datos, de una forma sencilla.

UML, Lenguaje Unificado de Modelado, Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

XML, eXtensible Markup Language, lenguaje de marcas extensible, es un metalenguaje, extensible de etiquetas desarrollado por el World Wide Web

Consortium (W3C). Permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.