UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

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

APÉNDICES

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

Índice de tablas

Índice de figuras

Apéndice A

Casos de uso

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	
Curso normal de los eventos	Acción del actor	Respuesta del sistema
	1. El actor especifica los atributos para identificar las clases y las propiedades a persistir	
	2. El actor ejecuta el programa especificando la ruta del ensamblado, el dialecto SQL a utilizar, la estrategia para el manejo de la herencia, y las rutas donde se desea que se generen los archivos.	
		3. Se carga el ensamblado desde la ruta especificada.
		4. Explorar la estructura del ensamblado en busca de las clases que se desean persistir con NHibernate.
		5. Una vez obtenida la estructura de clases a persistir, procede a transformar dicha estructura a un esquema relacional, y al mismo tiempo traduciéndolo al lenguaje SQL
		6. Se procede a generar los archivos de mapeo según las

		especificaciones de NHibernate, siguiendo una correspondencia entre las clases y las tablas generadas para el DDL
	7. El actor obtiene los archivos de mapeo y el archivo de DDL.	
Curso alterno	4.1 Los atributos para especificar una relación no están definidos	
	4.2 Las clases relacionadas no están definidas como clases persistentes	
	5.1 Las clases no tienen una propiedad Id	
	5.2 Algunas propiedades del tipo de dato string no tienen definido el tamaño	
	5.3 La estrategia para el manejo de la herencia no existe	
	5.4 El dialecto seleccionado no existe	
Casos de uso relacionados	#1.1, # 1.2 y #1.3	

Caso de uso #1.1	Obtener estructura de clases
Actor	Desarrollador
Propósito	Obtener la especificación (propiedades, nombre, etc) de las clases que se desean persistir con NHibernate
Resumen	Este proceso consiste en explorar el ensamblado que contiene las clases persistentes, y almacenarlas en una estructura (lista) de forma que mantenga las relaciones con las otras clases (asociación, composición, herencia, etc), e identificar cuales son los campos que se desean persistir en el modelo relacional.
Precondiciones	El actor antes de ejecutar NHBuilder, previamente debe especificar los atributos para identificar las clases persistentes, y los atributos para cada uno de las propiedades de las clases según sea el caso.
Curso normal de los eventos	Sistema
	1. Cargar el ensamblado desde la ruta especificada
	2. Explora la estructura del ensamblado en busca de las clases que se desean persistir con NHibernate
	3. Se buscan las propiedades de las clases, y se obtienen los atributos de cada propiedad que determinan el comportamiento de dicha propiedad en el modelo relacional
	4. Se buscan las propiedades cuya característica es que son relaciones con otras clases.
	5. Se verifica que la relación tenga el atributo que especifica el tipo de relación (si es asociación, referencia, composición, N-aria)
	6. Se verifica que este explícitamente el nombre de la clase con la que se relaciona en el caso de ser colecciones.
	7. Cada una de las clases persistentes y toda la información concerniente es almacenada.
	8. Todas las clases persistentes con su información son almacenadas en una lista para posterior análisis
	9. Se verifica que las clases relacionadas estén en la lista, incluyendo las superclases
Curso alterno	5.1 El atributo no esta especificado.
	6. La clase con la que se relaciona no esta especificada.
Casos de uso relacionados	#1.1.1, #1.1.2 y #1.1.3

Caso de uso #1.1.1	Obtener información de las clases
Descripción	Consiste en buscar la información con respecto a nombre, espacio de al que pertenece, si es una subclase buscar cual es su superclase.

Caso de uso #1.1.2	Obtener las propiedades de las clases
Descripción	Consiste en para cada una de las clases obtener sus propiedades junto con los atributos definidos.

Caso de uso #1.1.3	Obtener las relaciones de las clases
Descripción	Consiste en obtener cada una de las relaciones de las clases, con la información necesaria, como por ejemplo, clase con la que se relaciona, tipo de relación, tipo de colección.

Caso de uso #1.2	Transformar estructura de clases a un modelo relacional
Actor	Desarrollador
Propósito	Representar la estructura de clases en un esquema relacional y generar el archivo DDL.
Resumen	Este proceso consiste en recorrer la lista que contiene la estructura de las clases persistentes y según el criterio de mapeo de la herencia seleccionado por el actor transformar la estructura en un esquema relacional y generar el archivo DDL
Precondiciones	El actor al llamar a la función de generar debe especificar un dialecto de un manejador soportado por NHBuilder y una estrategia para el manejo de la herencia.
Curso normal de los eventos	Sistema
	1. Se obtiene la lista con la estructura de clases.
	2. Se obtiene el dialecto SQL y la estrategia para el manejo de la herencia.
	3. Se recorre la lista con la estructura de clases y dependiendo de la estrategia seleccionada para el manejo de la herencia, se transforma a un modelo relacional
	4. Se transforman, dependiendo de la estrategia, las relaciones entre las clases, a las denominadas claves foráneas y tablas de intersección según sea el caso.
	5. Una vez finalizada la transformación de cada clase, las tablas son exploradas para crear una sentencia SQL "CREATE TABLE..." según el dialecto seleccionado.
	6. Estas sentencias son escritas en un archivo con extensión .sql.
Curso alterno	2.1 El dialecto seleccionado no existe
	2.2 La estrategia para el manejo de la herencia no existe
	5.1 Las clases no tienen una propiedad Id
	5.2 Las propiedades del tipo de dato string no tienen definido el tamaño
Casos de uso relacionados	#1.2.1, #1.2.2 y #1.2.3
Postcondiciones	El actor debe crear la base de datos en el manejador de su elección (correspondiente al dialecto SQL seleccionado) y ejecutar el archivo DDL para crear las tablas.

Caso de uso #1.2.1	Transformar estructura de clases usando la estrategia "una tabla por clase jerárquica"
Descripción	La transformación de la estructura de clases se guía por el siguiente criterio: Las superclases adoptan los campos y relaciones de sus subclases. Las subclases no son persistidas.

Caso de uso #1.2.2	Transformar estructura de clases usando la estrategia "una tabla por subclase"
Descripción	La transformación de la estructura de clases se guía por el siguiente criterio: Una tabla por subclase: La superclase establece una relación entre ella y sus subclases. Todas las clases son persistidas, pero existe una relación entre ellas.

Caso de uso #1.2.3	Transformar estructura de clases usando la estrategia "una tabla por cada clase concreta"
Descripción	La transformación de la estructura de clases se guía por el siguiente criterio: Una tabla por cada clase concreta: Las subclases adoptan los campos y relaciones de su superclase. La superclase no es persistida.

Caso de uso #1.3	Generar archivos de mapeo para NHibernate
Actor	Desarrollador
Propósito	Generar los archivos de mapeo según las especificaciones de NHibernate.
Resumen	Este proceso consiste en recorrer la lista que contiene la estructura de las clases persistentes y según el criterio de mapeo de la herencia seleccionado por el actor y realizar el mapeo de las clases.
Precondiciones	El actor al llamar a la función de generar debe especificar una estrategia para el manejo de la herencia.
Curso normal de los eventos	Sistema
	1. Se obtiene la lista con la estructura de clases.
	2. Se obtiene la estrategia para el manejo de la herencia.
	3. Se recorre la lista con la estructura de clases y dependiendo de la estrategia seleccionada para el manejo de la herencia, se tiene un mapeo específico de las superclases y subclases.
	4. Se mapean cada una de las propiedades y relaciones de las clases independientemente de la estrategia.
	5. Por cada clase persistente se genera un archivo .hbm.xml.
Curso alterno	3.1 La estrategia para el manejo de la herencia no existe
	4. Las clases no tienen una propiedad Id
	5.1 Las clases no tienen una propiedad Id
	5.2 Las propiedades del tipo de dato string no tienen definido el tamaño
Casos de uso relacionados	#1.3.1, #1.3.2 y #1.3.3
Postcondiciones	El actor debe incluir en el proyecto donde se encuentran sus clases persistentes los archivos de mapeo como "Recurso incrustado"

Caso de uso #1.3.1	Generar los archivos de mapeo usando la estrategia "una tabla por clase jerárquica"
Descripción	Se mapean las clases siguiendo las especificaciones de NHibernate para esta estrategia y manteniendo coherencia entre la estructura de clases y el modelo relacional generado.

Caso de uso #1.3.2	Generar los archivos de mapeo usando la estrategia "una tabla por subclase"
Descripción	Se mapean las clases siguiendo las especificaciones de NHibernate para esta estrategia y manteniendo coherencia entre la estructura de clases y el modelo relacional generado.

Caso de uso #1.3.3	Generar los archivos de mapeo usando la estrategia "una tabla por cada clase concreta"
Descripción	Se mapean las clases siguiendo las especificaciones de NHibernate para esta estrategia y manteniendo coherencia entre la estructura de clases y el modelo relacional generado.

Apéndice B

Diagramas de casos de uso

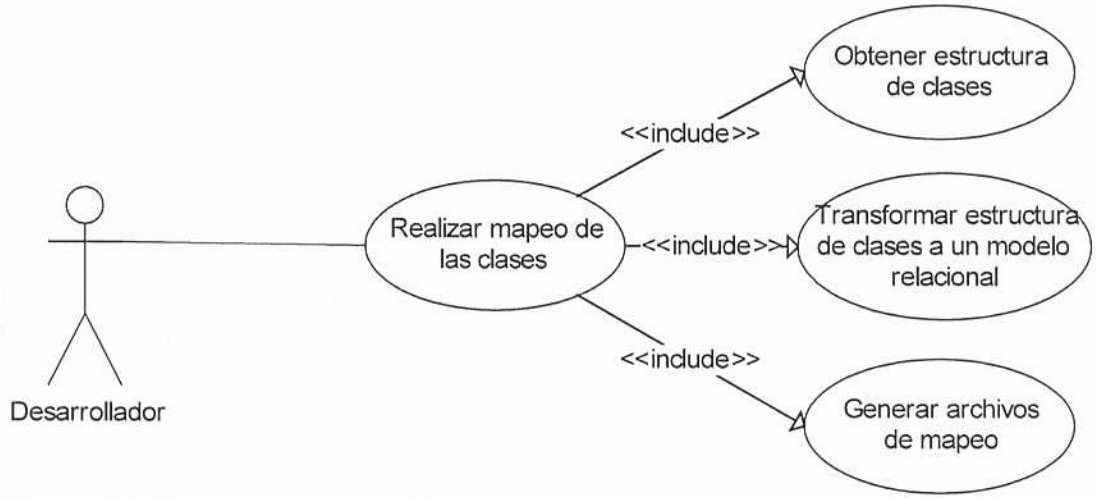


Figura B.1: Diagrama de casos de uso.
Diseño: Lucero, K. (2007)

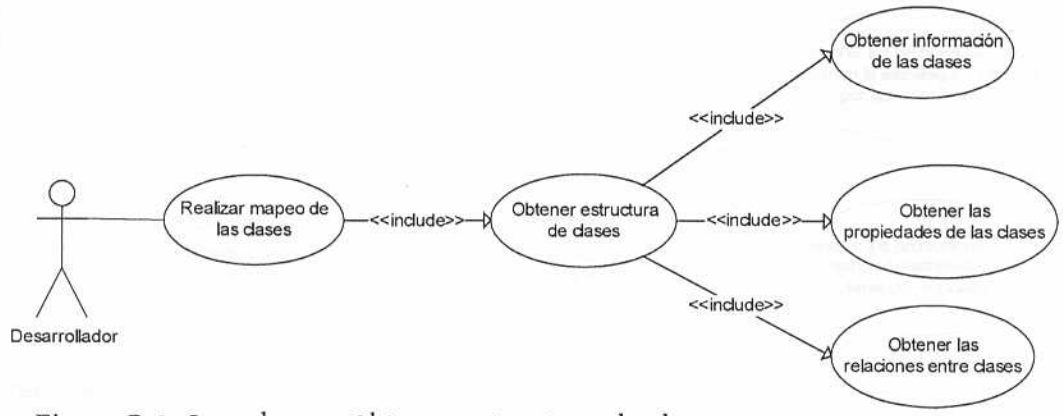


Figura B.2: Caso de uso: Obtener estructura de clases.
Diseño: Lucero, K. (2007)

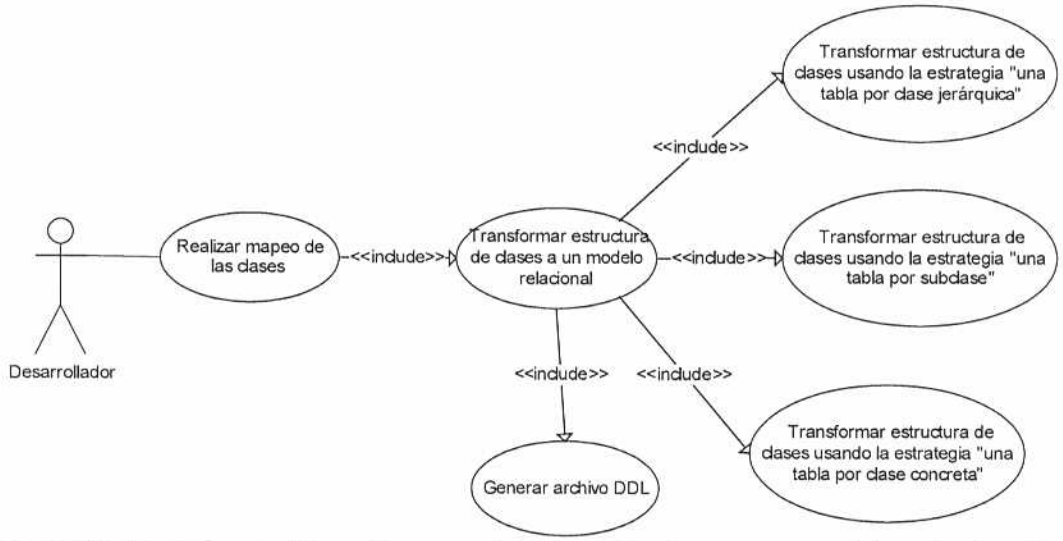


Figura B.3: Caso de uso: Transformar estructura de clases a un modelo relacional.
Diseño: Lucero, K. (2007)

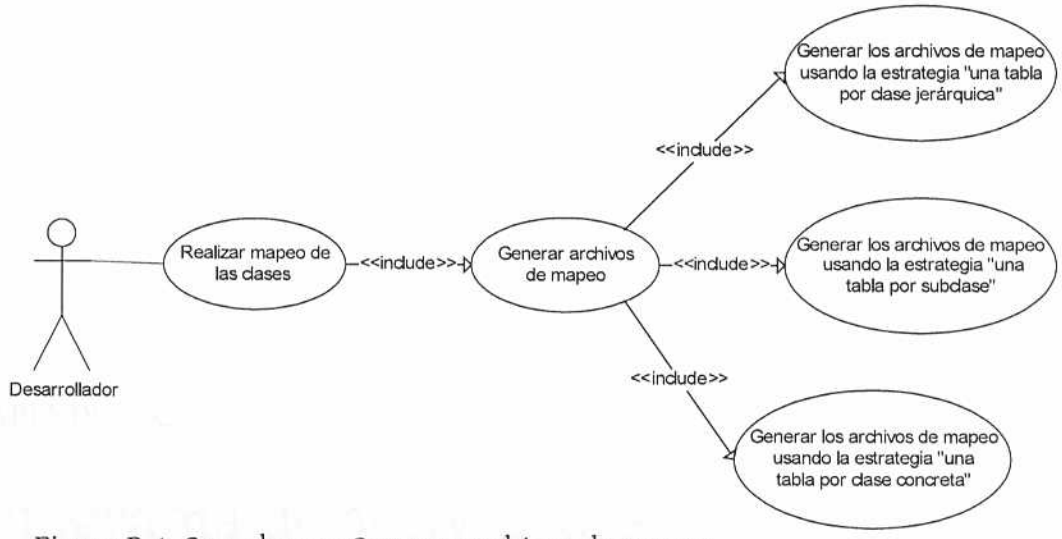


Figura B.4: Caso de uso: Generar archivos de mapeo.
Diseño: Lucero, K. (2007)

Apéndice C

Diagrama de actividades

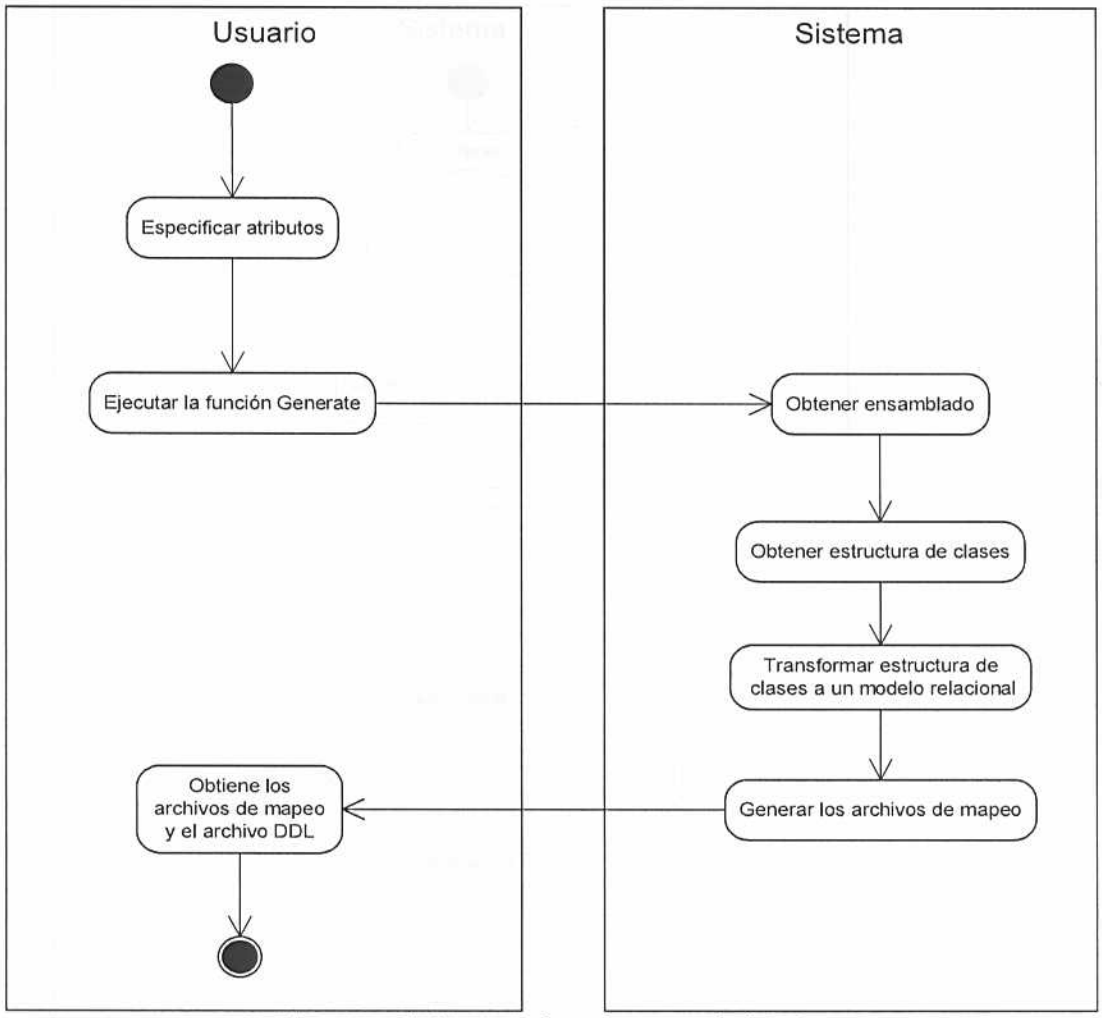


Figura C.1: Diagrama de actividades - Realizar mapeo de las clases.
Diseño: Lucero, K. (2007)

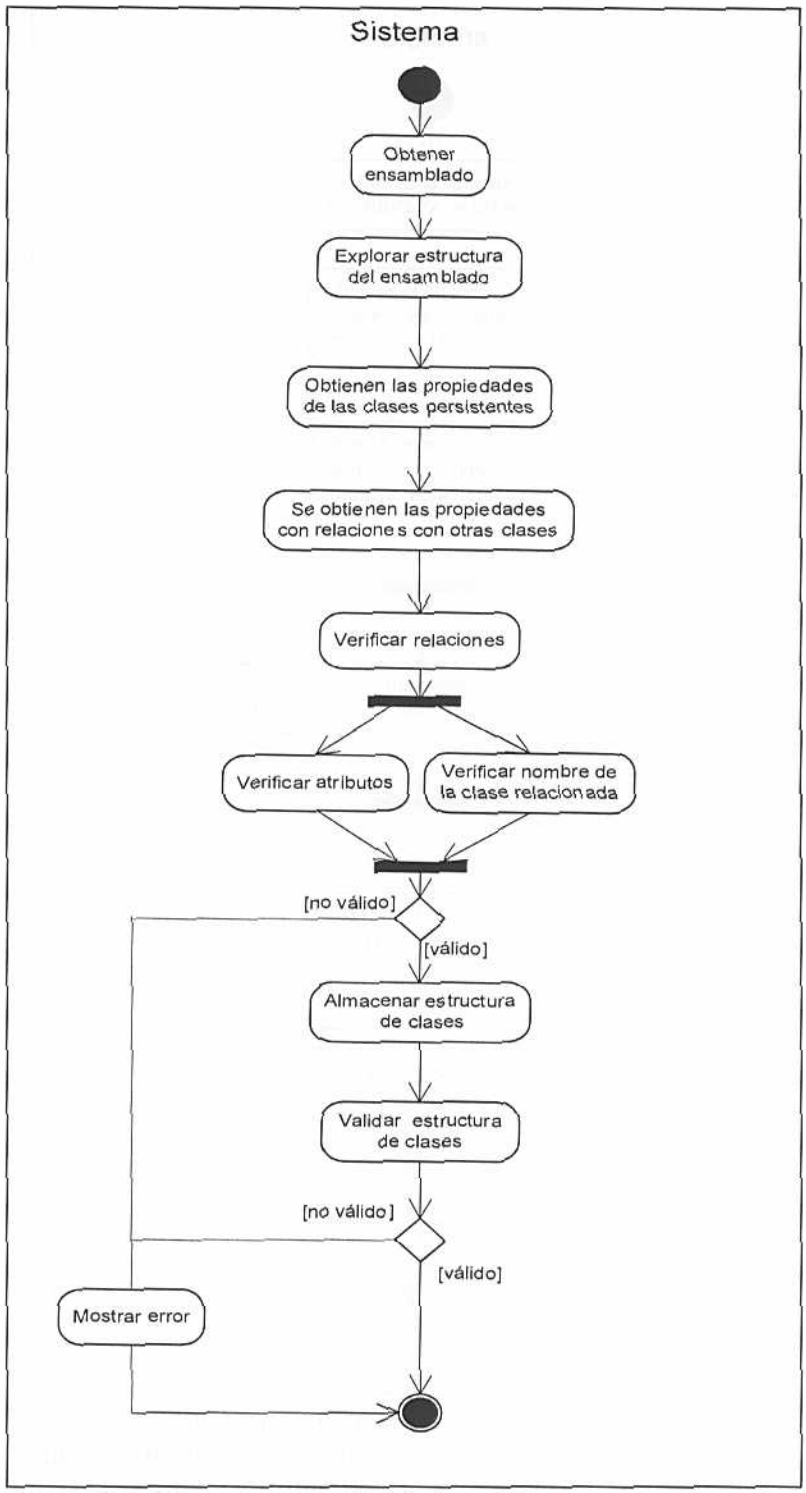


Figura C.2: Diagrama de actividades - Obtener estructura de clases.
Diseño: Lucero, K. (2007)

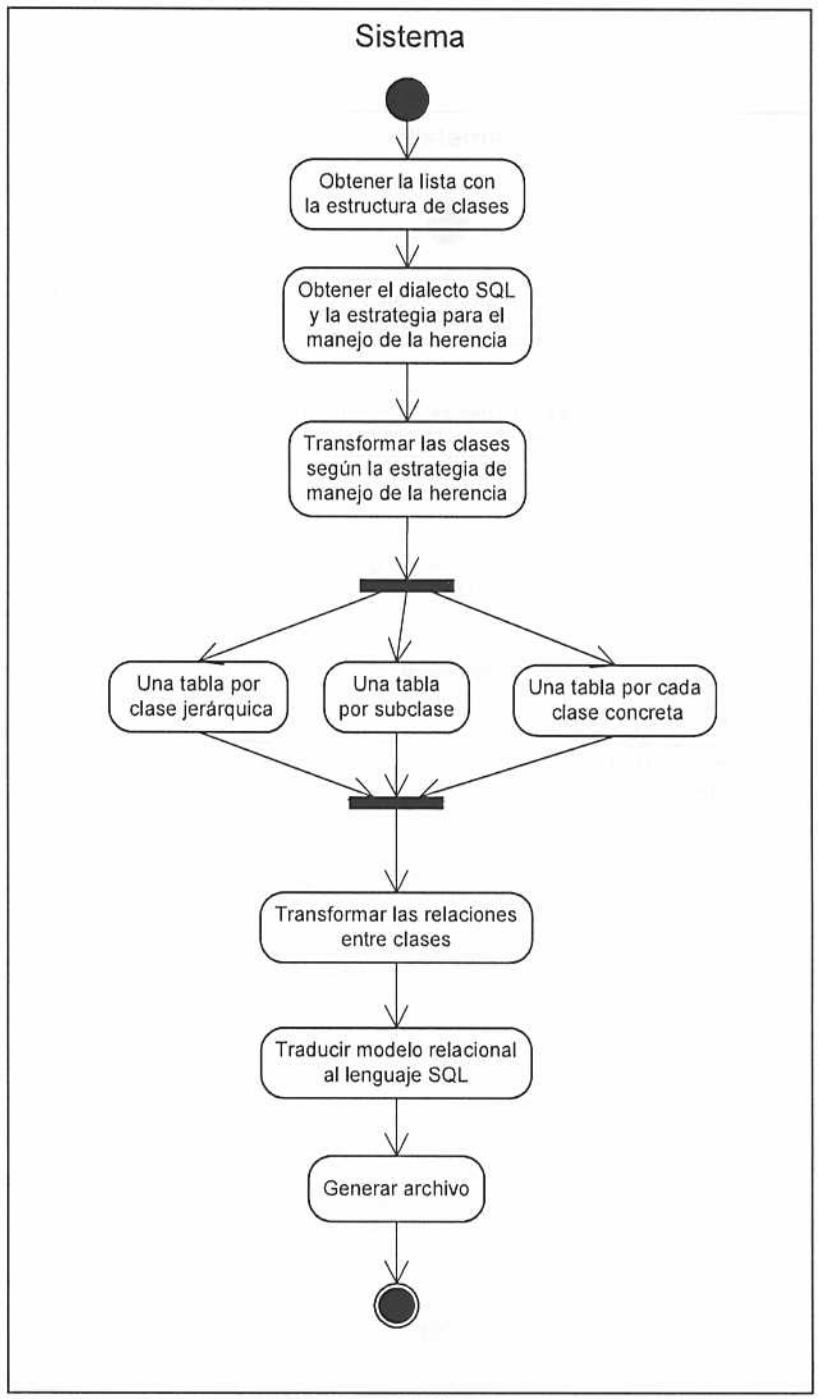


Figura C.3: Diagrama de actividades - Transformar estructura de clases a un modelo relacional.
Diseño: Lucero, K. (2007)

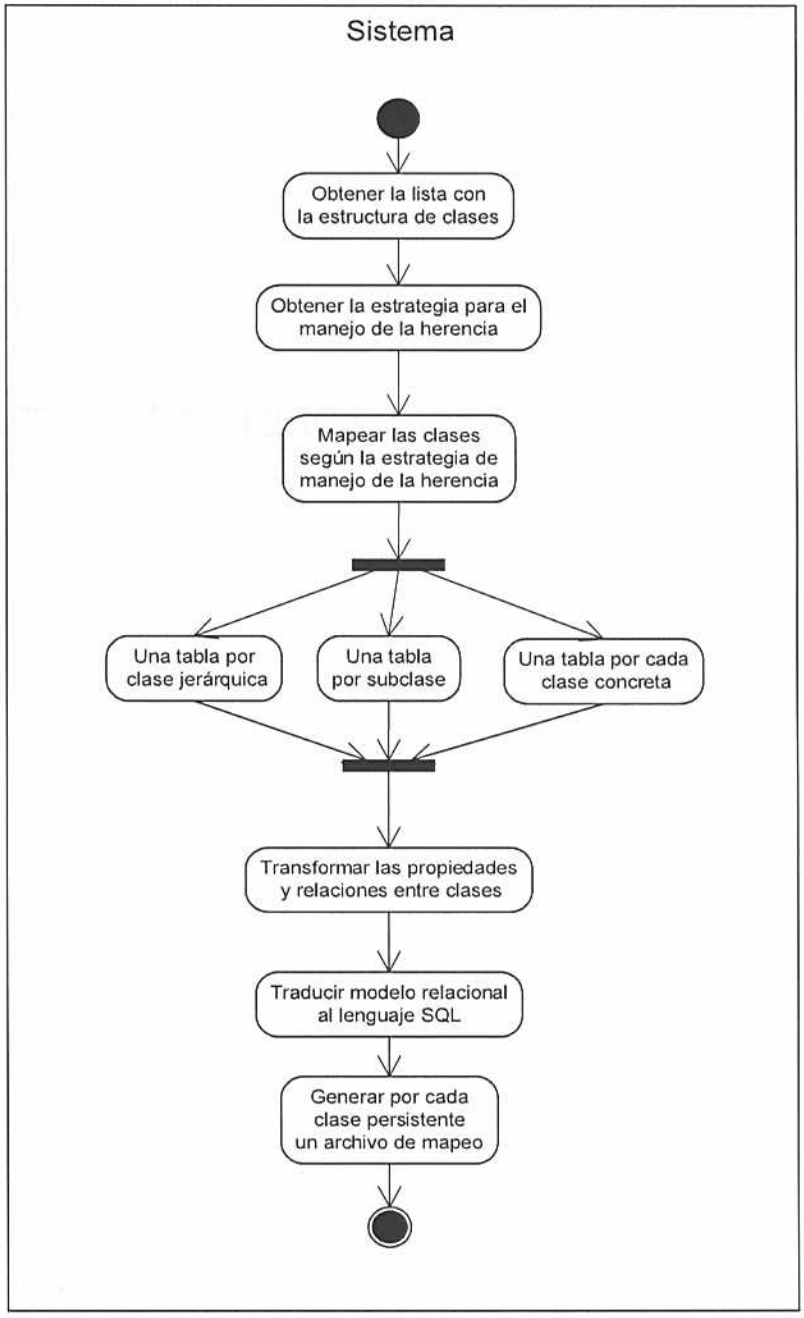


Figura C.4: Diagrama de actividades - Generar archivos de mapeo.
Diseño: Lucero, K. (2007)

Apéndice D

Diagramas de clases

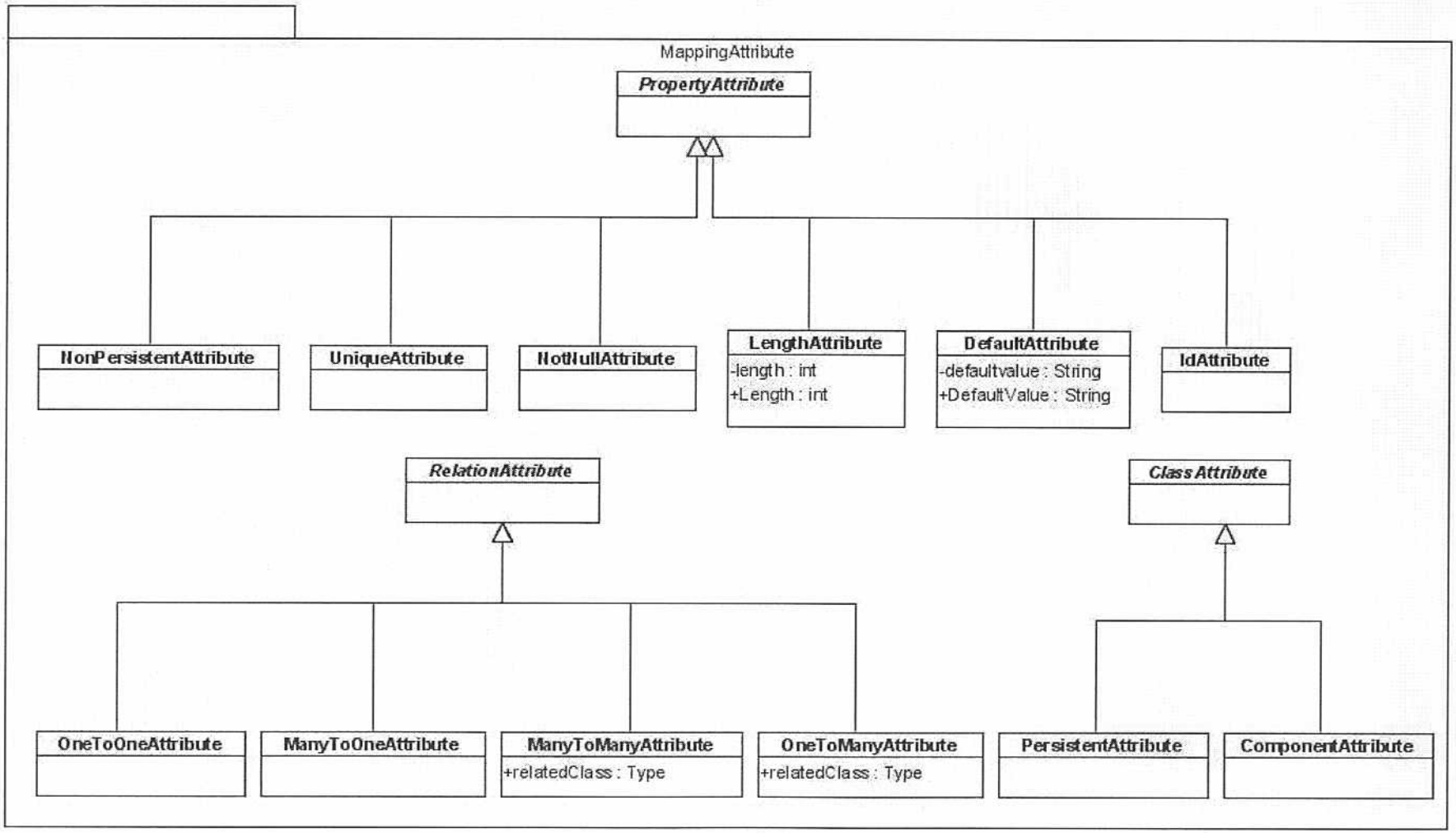


Figura D.1: Diagrama de clases – MappingAttribute.
Diseño: Lucero, K. (2007)

Apéndice E

Diagramas de secuencia

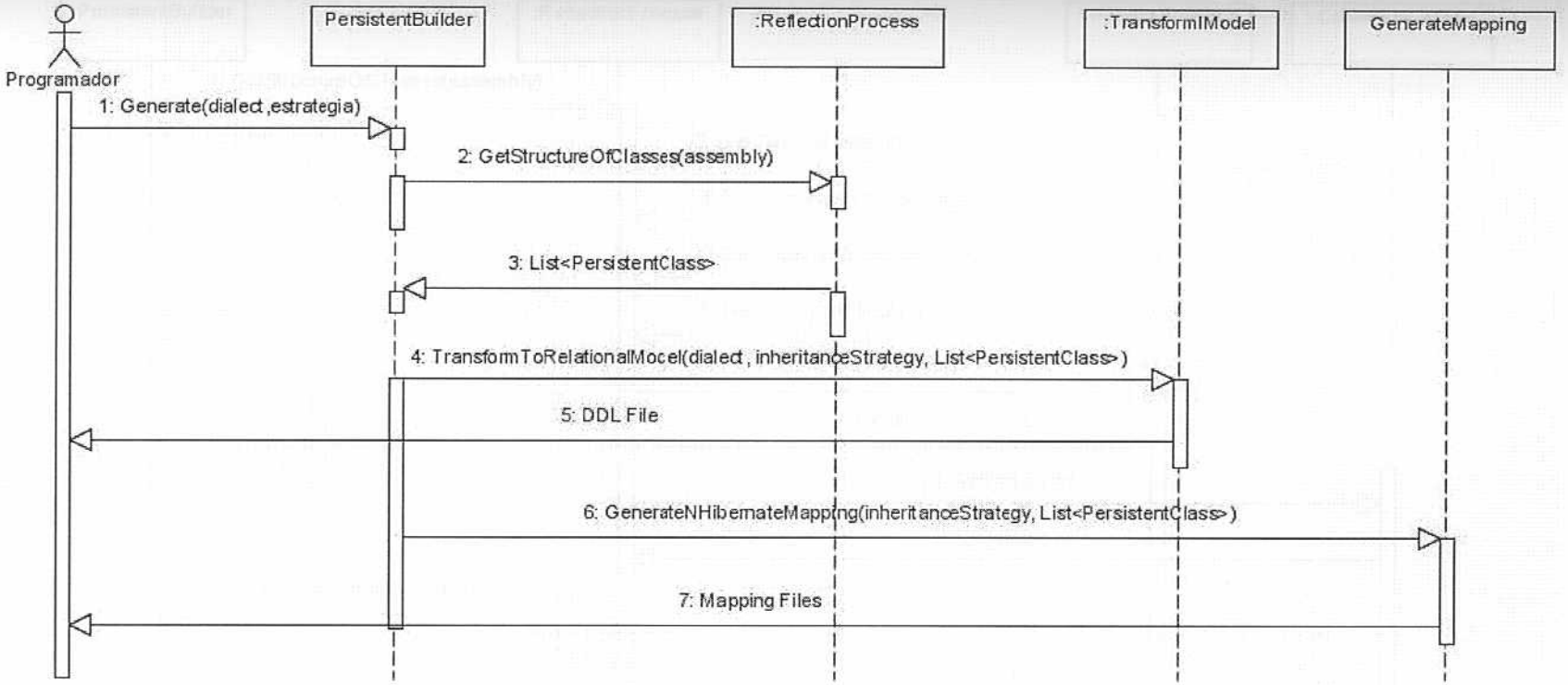


Figura E.1: Diagramas de secuencia - Realizar mapeo de las clases.
Diseño: Lucero, K. (2007)

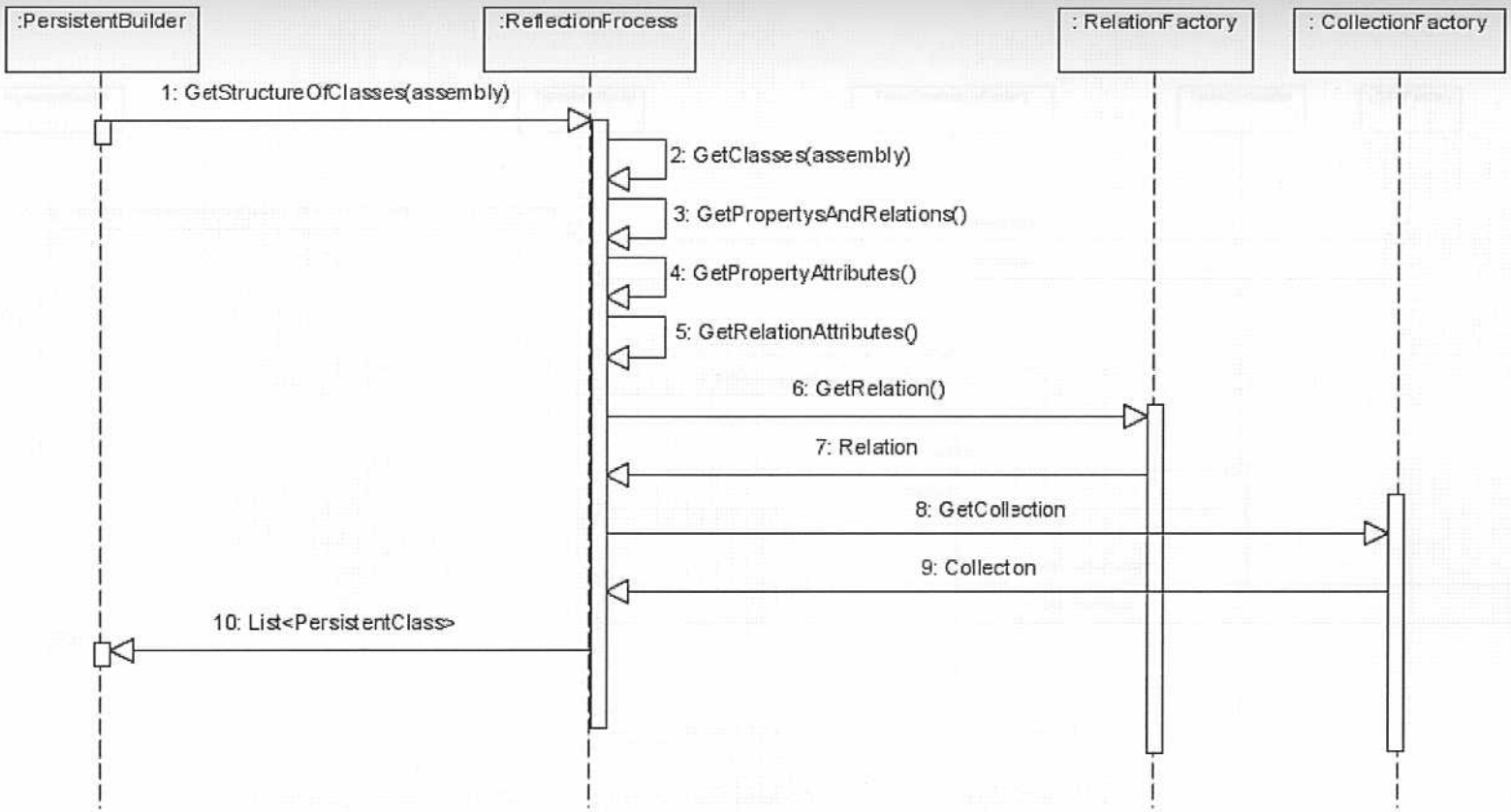


Figura E.2: Diagramas de secuencia - Obtener estructura de clases.
Diseño: Lucero, K. (2007)

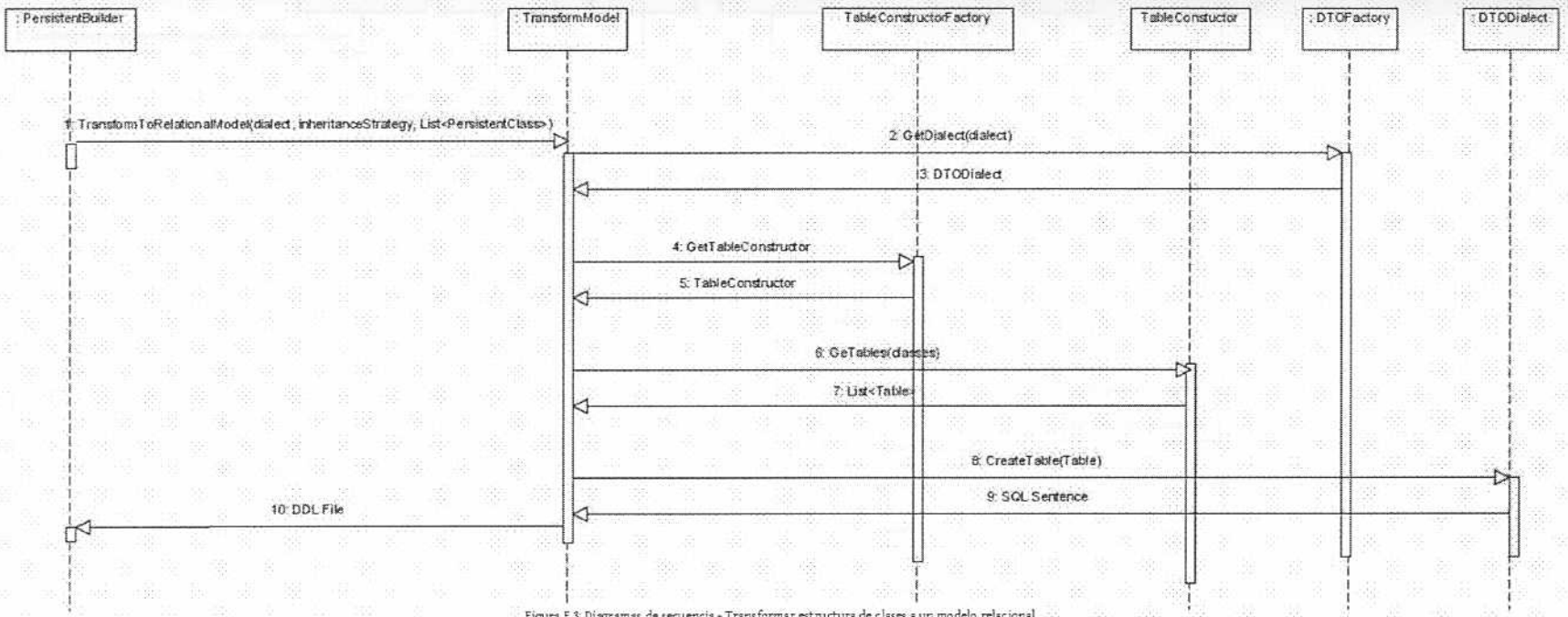


Figura E.3: Diagramas de secuencia - Transformar estructura de clases a un modelo relacional.
Diseño: Lucero, K. (2007)

Figura E.3: Diagramas de secuencia - Transformar estructura de clases a un modelo relacional.
Diseño: Lucero, K. (2007)

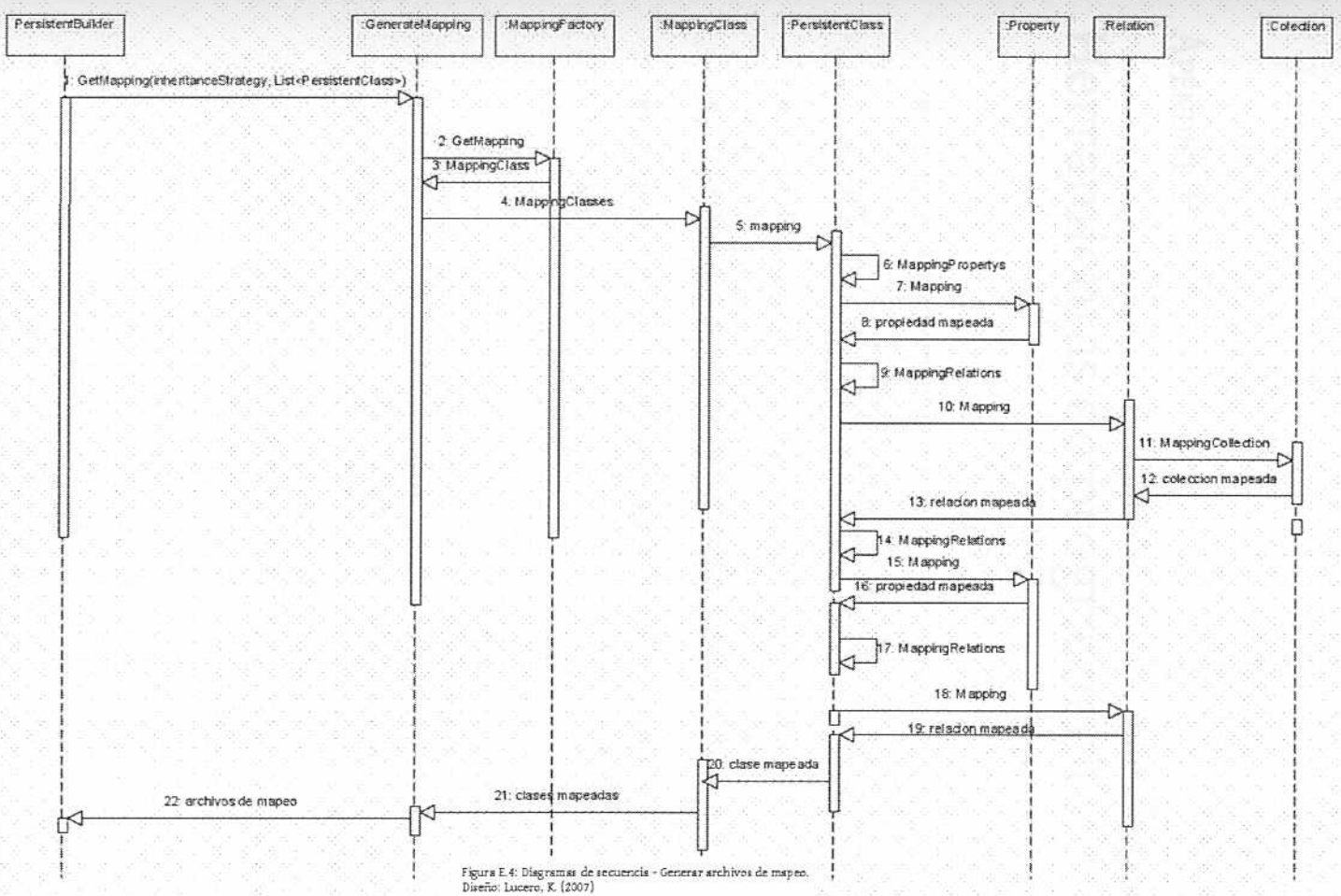


Figura E.4: Diagramas de secuencia - Generar archivos de mapeo.
Diseño: Lucero, K. (2007)

Apéndice F

Herramientas de programación

- Para la implementación de la herramienta se utilizó la plataforma de .NET 2.0.
- El IDE utilizado fue Visual C# 2005 Express Edition.
- Para ejecutar las pruebas unitarias se utilizó NUnit-Net-2.0 2.2.8.
- NDoc 1.3.1 fue usado para generar la documentación de la herramienta (API).

Apéndice G

Estrategias de mapeo

Objeto dependiente o contenido Para la representación de este tipo de relación se tiene el siguiente ejemplo:

```
C#
[Persistent]
    public class Empleado
    {
        long idempleado;
        Nombre nombre;
        [Id]
        public long IdEmpleado
        {
            get { return idempleado; }
            set { idempleado = value; }
        }

        public Nombre Nombre
        {
            get { return nombre; }
            set { nombre = value; }
        }
    }
    [Component]
    public class Nombre
    {
        string primerNombre;
        string primerApellido;
```

```
        [Length(25)]
        public string PrimerNomnre
        {
            get { return primerNombre; }
            set { primerNombre = value; }
        }
        [Length(25)]
        public string PrimerApellido
        {
            get { return primerApellido; }
            set { primerApellido = value; }
        }
    }
```

El siguiente código escrito en lenguaje SQL para Microsoft SQL Server, muestra como sería el esquema relacional para las clases definidas anteriormente.

Microsoft SQL Server 2005

```
CREATE TABLE [empleado] (
      [empleado_idempelado] [bigint] IDENTITY(1,1) NOT NULL,
      [nombre_primernombre] [nvarchar] (25) NULL,
      [nombre_primerapellido] [nvarchar] (25) NULL,
      PRIMARY KEY (empleado_idempleado)
)
```

```
Empleado.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Empleado" table="empleado">
  <id name="IdEmpleado" column="empleado_idempleado" type="Int64">
   <generator class="identity"/>
  </id>
  <component name="Nombre" class="Nombre">
   <property name="PrimerNombre" column="nombre_primernombre"
            type="String"/>
   <property name="PrimerapAllido" column="nombre_primerapellido"
            type="String"/>
  </component>
 </class>
</hibernate-mapping>
```

- Asociación simple (1 .. 1): Para la representación de una asociación simple se tiene el siguiente código en C# con los atributos definidos previamente para la identificación de las clases persistentes y las características de las propiedades

```
C#
[Persistent]
public class Venta
{
  long idventa;
  Pago tipopago;
  [Id]
  public long IdVenta
  {
    get { return idventa;}
    set { idventa = value;}
  }
  [OneToOne]
  public Pago TipoPago
  {
    get { return tipopago;}
    set { tipopago = value;}
  }
}

[Persistent]
public class Pago
{
   long idpago;
  [Id]
  public long IdPago
  {
    get { return idpago;}
    set { idpago = value;}
  }
}
```

El siguiente código escrito en lenguaje SQL para Microsoft SQL Server, muestra como sería el esquema relacional para las clases definidas anteriormente.

```
Microsoft SQL Server 2005

CREATE TABLE [venta] (
      [venta_idventa] [bigint] IDENTITY(1,1) NOT NULL,
      PRIMARY KEY (venta_idventa)
)
CREATE TABLE [pago] (
      [pago_idpago] [bigint] IDENTITY(1,1) NOT NULL,
      PRIMARY KEY (pago_idpago)
)


Venta.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Venta" table="venta">
  <id name="IdVenta" column="venta_idventa" type="Int64">
   <generator class="identity"/>
  </id>
  <one-to-one name="TipoPago" class="Pago"/>
 </class>
</hibernate-mapping>

Pago.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Pago" table="pago">
  <id name="IdPago" column="pago_idpago" type="Int64">
   <generator class="identity"/>
  </id>
 </class>
</hibernate-mapping>
```

- Referencia simple (N .. 1) : Para la representación de una referencia simple se tiene el siguiente código en C# con los atributos definidos previamente para la identificación de las clases persistentes y las características de las propiedades

```
C#
[Persistent]
public class Tienda
{
   long idtienda;
   [Id]
   public long IdTienda
   {
      get { return idtienda;}
      set { idtienda = value;}
   }
}

[Persistent]
public class Producto
{
   long idproducto;
   Tienda owner;
   [Id]
   public long IdProducto
   {
      get { return idproducto;}
      set { idproducto = value;}
   }
   [ManyToOne]
   public Tienda Owner
   {
        get { return owner;}
        set { owner = value;}
   }
}
```

El siguiente código escrito en lenguaje SQL para Microsoft SQL Server, muestra como sería el esquema relacional para las clases definidas anteriormente.

```
Microsoft SQL Server 2005
CREATE TABLE [tienda] (
      [tienda_idtienda] [bigint] IDENTITY(1,1) NOT NULL,
      PRIMARY KEY (tienda_idtienda)
)
CREATE TABLE [producto] (
      [producto_idproducto] [bigint] IDENTITY(1,1) NOT NULL,
      [fk_tienda] [bigint] NOT NULL,
      PRIMARY KEY (producto_idproducto)
      FOREIGN KEY (fk_tienda) REFERENCES tienda(tienda_idtienda)
)

Tienda.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Tienda" table="tienda">
  <id name="IdTienda" column="tienda_idtienda" type="Int64">
   <generator class="identity"/>
  </id>
 </class>
</hibernate-mapping>

Producto.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Producto" table="producto">
  <id name="IdProducto" column="producto_idproducto" type="Int64">
   <generator class="identity"/>
  </id>
  <many-to-one name="Owner" class="Tienda" column="fk_tienda"/>
 </class>
</hibernate-mapping>
```

- Colección básica (1 .. N) : Para la representación de una colección básica se tiene el siguiente código en C# con los atributos definidos previamente para la identificación de las clases persistentes y las características de las propiedades

```
C#
[Persistent]
public class Tienda
{
   long idtienda;
   IList productos;
   [Id]
   public long IdTienda
   {
      get { return idtienda;}
      set { idtienda = value;}
   }
   [OneToMany(relatedClass=typeof(Producto))]
   public IList Productos
   {
       get { return productos;}
       set { productos = value;}
   }
}

[Persistent]
public class Producto
{
       long idproducto;
      [Id]
      public long IdProducto
      {
          get { return idproducto;}
          set { idproducto = value;}
      }
}
```

El siguiente código escrito en lenguaje SQL para Microsoft SQL Server, muestra como sería el esquema relacional para las clases definidas anteriormente

```
Microsoft SQL Server 2005
CREATE TABLE [tienda] (
      [tienda_idtienda] [bigint] IDENTITY(1,1) NOT NULL,
      PRIMARY KEY (tienda_idtienda)
)
CREATE TABLE [producto] (
      [producto_idproducto] [bigint] IDENTITY(1,1) NOT NULL,
      [fk_tienda] [bigint] NOT NULL,
      PRIMARY KEY (producto_idproducto)
      FOREIGN KEY (fk_tienda) REFERENCES tienda(tienda_idtienda)
)

Tienda.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Tienda" table="tienda">
  <id name="IdTienda" column="tienda_idtienda" type="Int64">
   <generator class="identity"/>
  </id>
  <list name="Productos" table="tienda">
   <key column="fk_tienda" />
   <index column="producto_idproducto" />
   <one-to-many class="Producto" />
  </list>
 </class>
</hibernate-mapping>

Producto.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Producto" table="producto">
  <id name="IdProducto" column="producto_idproducto" type="Int64">
   <generator class="identity"/>
  </id>
 </class>
</hibernate-mapping>
```

- Relación N..M: Para la representación de una relación N..M se tiene el siguiente código en C# con los atributos definidos para la identificación de las clases persistentes y las características de las propiedades.

C#

```
[Persistent]
    public class Factura
    {
        long idfactura;
        IList productos;
        [Id]
        public long IdFactura
        {
            get { return idfactura; }
            set { idfactura = value; }
        }
        [ManyToMany(relatedClass=typeof(Producto))]
        public IList Productos
        {
            get { return productos; }
            set { productos = value; }
        }
    }

    [Persistent]
    public class Producto
    {
        long idproducto;
        [Id]
        public long IdProducto
        {
            get { return idproducto; }
            set { idproducto = value; }
        }
    }
```

El siguiente código escrito en lenguaje SQL para Microsoft SQL Server, muestra como sería el esquema relacional para las clases definidas anteriormente

```
Microsoft SQL Server 2005
CREATE TABLE [factura] (
      [factura_idfactura] [bigint] IDENTITY(1,1) NOT NULL,
      PRIMARY KEY (factura_idfactura)
)
CREATE TABLE [producto] (
      [producto_idproducto] [bigint] IDENTITY(1,1) NOT NULL,
      PRIMARY KEY (producto_idproducto)
)
CREATE TABLE Factura_Producto(
      [idfactura_producto] [bigint] IDENTITY(1,1) NOT NULL,
      [fk_factura] [bigint] NOT NULL,
      [fk_producto] [bigint] NOT NULL,
      PRIMARY KEY (idfactura_producto)
      FOREIGN KEY (fk_factura) REFERENCES factura(factura_idfactura)
      FOREIGN KEY (fk_producto) REFERENCES producto(producto_idproducto)
)
```

```
Factura.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Factura" table="factura">
  <id name="IdFactura" column="factura_idfactura" type="Int64">
   <generator class="identity"/>
  </id>
  <list name="Productos" table=" producto">
   <key column="fk_factura" />
   <index column="id_factura_producto " type="Int64" />
   <many-to-many column="fk_producto" class="Producto" />
  </list>
 </class>
</hibernate-mapping>
```

```
Producto.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Producto" table="producto">
  <id name="IdProducto" column="producto_idproducto" type="Int64">
   <generator class="identity"/>
  </id>
 </class>
</hibernate-mapping>
```

- Herencia: Para la representación de la herencia se tiene el siguiente código en C# con los atributos definidos previamente para la identificación de las clases persistentes y las características de las propiedades.

Supóngase que se tiene una clase Pago, y las siguientes subclases Credito, Efectivo, Cheque.

```
C#
[Persistent]
public class Pago
{
   long idpago;
   long monto;
   [Id]
   public long IdPago
   {
      get { return idpago;}
      set { idpago = value;}
   }
   [NotNull]
   public long Monto
   {
       get { return monto;}
       set { monto = value;}
   }
}
[Persistent]
public class Credito : Pago{ /**…*/ }
[Persistent]
public class Efectivo: Pago { /**…*/ }
[Persistent]
public class Cheque: Pago{ /**…*/ }
```

Una tabla por clase jerárquica

Sólo se requiere una tabla. Esta estrategia tiene una gran limitación: Las columnas de las subclases no deben tener la restricción NOT NULL

```
Microsoft SQL Server 2005
CREATE TABLE Pago(
        [pago_idpago] [bigint] IDENTITY(1,1) NOT NULL,
        [pago_monto] [int] NOT NULL,
        [pago_tipo] [nvarchar] (100) NOT NULL,
        PRIMARY KEY (pago_idpago)
)
```

```
Pago.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Pago" table="pago">
  <id name="IdPago" type="Int64" column="pago_idpago">
   <generator class="identity"/>
  </id>
  <discriminator column="pago_tipo" type="String"/>
  <property name="Monto" column="pago_monto" not-null="true"/>

  <subclass name="Credito" discriminator-value="CREDIT">
      ...
  </subclass>

  <subclass name="Efectivo" discriminator-value="CASH">
      ...
  </subclass>

  <subclass name="Cheque" discriminator-value="CHEQUE">
      ...
  </subclass>
 </class>
</hibernate-mapping>
```

Una tabla por subclase

Se requieren cuatro tablas. Las tres tablas de las subclases tienen una clave primaria asociada a la tabla de la superclase (entonces la relación en este caso sería uno a uno)

```
Microsoft SQL Server 2005
CREATE TABLE Pago(
      [pago_idpago] [bigint] IDENTITY(1,1) NOT NULL,
      [monto] [int] NOT NULL,
      [tipo_pago] [nvarchar] (100) NOT NULL,
      PRIMARY KEY (pago_idpago)
)
CREATE TABLE Credito(
      ...
      [pago_idpago] [bigint] NOT NULL,
      FOREIGN KEY (pago_idpago) REFERENCES pago(pago_idpago)
 )
CREATE TABLE Efectivo(
      ...
      [pago_idpago] [bigint] NOT NULL,
      FOREIGN KEY (pago_idpago) REFERENCES pago(pago_idpago)
 )
CREATE TABLE Cheque(
       ...
      [pago_idpago] [bigint] NOT NULL,
      FOREIGN KEY (pago_idpago) REFERENCES pago(pago_idpago)
 )
```

```
Pago.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Pago" table="pago">
  <id name="IdPago" type="Int64" column="pago_idpago">
   <generator class="identity"/>
  </id>
  <property name="Monto" column="pago_monto" not-null="true"/>

  <joined-subclass name="Credito" table="credito">
   <key column="pago_idpago"/>
      ...
  </joined-subclass>

  <joined-subclass name="Efectivo" table="efectivo">
   <key column="pago_idpago"/>
      ...
  </joined-subclass>

  <joined-subclass name="Cheque" table="cheque">
   <key column="pago_idpago"/>
      ...
  </joined-subclass>
 </class>
</hibernate-mapping>
```

Una tabla por cada clase concreta

Sólo se requieren tres tablas. Nótese que no se menciona en ningún momento de forma explicita la interfaz Pago, en vez de eso NHibernate usa el polimorfismo implícito. Nótese también que las propiedades (las columnas) de Pago han sido mapeadas en cada una de las subclases.

```
Microsoft SQL Server 2005
CREATE TABLE [credito] (
        [pago_idpago] [bigint] IDENTITY(1,1) NOT NULL,
        [pago_monto] [int] NOT NULL,
      PRIMARY KEY (pago_idpago)
)
CREATE TABLE [efectivo] (
        [pago_idpago] [bigint] IDENTITY(1,1) NOT NULL,
        [pago_monto] [int] NOT NULL,
      PRIMARY KEY (pago_idpago)
)
CREATE TABLE [cheque] (
        [pago_idpago] [bigint] IDENTITY(1,1) NOT NULL,
        [pago_monto] [int] NOT NULL,
      PRIMARY KEY (pago_idpago)
)
```

```
Credito.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Credito" table="credito">
   <id name="IdPago" type="Int64" column="pago_idpago">
    <generator class="identity"/>
   </id>
   <property name="Monto" column="pago_monto" not-null="true"/>
      ...
 </class>
</hibernate-mapping>

Efectivo.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Efectivo" table="efectivo">
  <id name="IdPago" type="Int64" column="pago_idpago">
   <generator class="identity"/>
  </id>
  <property name="Monto" column="pago_monto" not-null="true"/>
      ...
 </class>
</hibernate-mapping>

Cheque.hbm.xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
 <class name="Cheque" table="cheque">
  <id name="IdPago" type="Int64" column="pago_idpago">
   <generator class="identity"/>
  </id>
  <property name="Monto" column="pago_monto" not-null="true"/>
      ...
 </class>
</hibernate-mapping>
```

Apéndice H

Estándares de codificación utilizados

El estándar de codificación de la herramienta es la siguiente:

Las clases, espacio de nombres, métodos, enumeraciones y propiedades utilizan la siguiente nomenclatura: La primera letra del identificador y la primera letra de las siguientes palabras concatenadas están en mayúsculas.

Ejemplo: MappingAttribute	Espacio de Nombres
NHibernateMapping	Clase
TransformToRelationalModel	Método
Name	Propiedad

Para los campos de las clases, parámetros de los métodos, y variables utilizan la siguiente nomenclatura: La primera letra del identificador está en minúscula y la primera letra de las siguientes palabras concatenadas en mayúscula.

nombreVariable

Para nombrar las clases, espacios de nombres, propiedades, métodos se utilizo el alfabeto inglés.

Para el nombrado de los espacios de nombres se siguió la siguiente regla: usar primero el nombre del proyecto y a continuación la división interna de los componentes del proyecto.

NHBuilder.Reflection

NHBuilder.RelationalModel

NHBuilder.Relations

Para el nombrado en general de las clases se utilizaron sustantivos que signifiquen algo concreto.

Property

Table

Column

Las clases de atributos poseen el sufijo Attribute

PersistentAttribute

OneToOneAttribute

Para las excepciones se usaron las mismas reglas paras las clases,

agregandoles el sufijo Exception.

IdNotFoundException

InvalidRelationException

Para las interfaces se usaron sustantivos y adjetivos con el prefijo I. Aunque existe la excepción de DTODialect.

Las enumeraciones y sus valores poseen nombres en singular:

RelationType { OneToOne, OneToMany, ...

InheritanceStrategy

Apéndice I

Manual del usuario

Author - Microsoft Visual C# 2005 Express

Archivo Edición Ver Refactorizar Proyecto Generar Depurar Datos Herramientas Ventana Comunidad Ayuda

Work.cs | Song.cs | Book.cs | Author.cs

Author.Author | id

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Text;
5  using MappingAttribute;
6
7  namespace Author
8  {
9      [Persistent]
10     public class Author
11     {
12         long id;
13         string alias;
14         Person person;
15         IList works;
16
17         [Id]
18         public long Id
19         {
20             get { return id; }
21             set { id = value; }
22         }
23         [Length(20)]
24         public string Alias
25         {
26             get { return alias; }
27             set { alias = value; }
28         }
29
30         [OneToOne]
31         public Person Person
32         {
33             get { return person; }
34             set { person = value; }
```

Paso 1: Agregar referencia a MappingAttribute

Explorador de soluciones - Solución 'Author' ...

- Solución 'Author' (2 proyectos)
  - Author
    - Properties
    - References
      - MappingAttribute
      - System
      - System.Data
      - System.Xml
    - App.config
    - Author.cs
    - Book.cs
    - Person.cs
    - Song.cs
    - Work.cs
  - AuthorApp
    - Properties
    - References
    - Program.cs

Propiedades

Lista de errores

Cuadro de herramientas

Author - Microsoft Visual C# 2005 Express

Archivo Edición Ver Refactorizar Proyecto Generar Depurar Datos Herramientas Ventana Comunidad Ayuda

Work.cs | Song.cs | Book.cs | **Author.cs**

Author.Author | id

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Text;
5  using MappingAttribute;
6
7  namespace Author
8  {
9      [Persistent]
10     public class Author
11     {
12         long id;
13         string alias;
14         Person person;
15         IList works;
16
17         [Id]
18         public long Id
19         {
20             get { return id; }
21             set { id = value; }
22         }
23         [Length(20)]
24         public string Alias
25         {
26             get { return alias; }
27             set { alias = value; }
28         }
29
30         [OneToOne]
31         public Person Person
32         {
33             get { return person; }
34             set { person = value; }
```

Paso 2: Incluir el espacio de nombres MappingAttribute

Explorador de soluciones - Solución 'Author' ...

- Solución 'Author' (2 proyectos)
  - **Author**
    - Properties
    - References
      - MappingAttribute
      - System
      - System.Data
      - System.Xml
    - App.config
    - Author.cs
    - Book.cs
    - Person.cs
    - Song.cs
    - Work.cs
  - AuthorApp
    - Properties
    - References
    - Program.cs

Propiedades

Lista de errores

Cuadro de herramientas

Author - Microsoft Visual C# 2005 Express

Archivo Edición Ver Refactorizar Proyecto Generar Depurar Datos Herramientas Ventana Comunidad Ayuda

Work.cs | Song.cs | Book.cs | **Author.cs**

Author.Author | id

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Text;
5  using MappingAttribute;
6
7  namespace Author
8  {
9      [Persistent]
10     public class Author
11     {
12         long id;
13         string alias;
14         Person person;
15         IList works;
16
17         [Id]
18         public long Id
19         {
20             get { return id; }
21             set { id = value; }
22         }
23         [Length(20)]
24         public string Alias
25         {
26             get { return alias; }
27             set { alias = value; }
28         }
29
30         [OneToOne]
31         public Person Person
32         {
33             get { return person; }
34             set { person = value; }
```

Paso 3: Colocar el atributo [Persistent] a las clases persistentes.

Explorador de soluciones - Solución 'Author' ...

- Solución 'Author' (2 proyectos)
  - **Author**
    - Properties
    - References
      - MappingAttribute
      - System
      - System.Data
      - System.Xml
    - App.config
    - Author.cs
    - Book.cs
    - Person.cs
    - Song.cs
    - Work.cs
  - AuthorApp
    - Properties
    - References
    - Program.cs

Propiedades

Lista de errores

Cuadro de herramientas

Author - Microsoft Visual C# 2005 Express

Archivo Edición Ver Refactorizar Proyecto Generar Depurar Datos Herramientas Ventana Comunidad Ayuda

Work.cs | Song.cs | Book.cs | **Author.cs**

Author.Author | Id

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Text;
5  using MappingAttribute;
6
7  namespace Author
8  {
9      [Persistent]
10     public class Author
11     {
12         long id;
13         string alias;
14         Person person;
15         IList works;
16
17         [Id]
18         public long Id
19         {
20             get { return id; }
21             set { id = value; }
22         }
23         [Length(20)]
24         public string Alias
25         {
26             get { return alias; }
27             set { alias = value; }
28         }
29
30         [OneToOne]
31         public Person Person
32         {
33             get { return person; }
34             set { person = value; }
```

Paso 4: Colocar los atributos a las propiedades de la clases

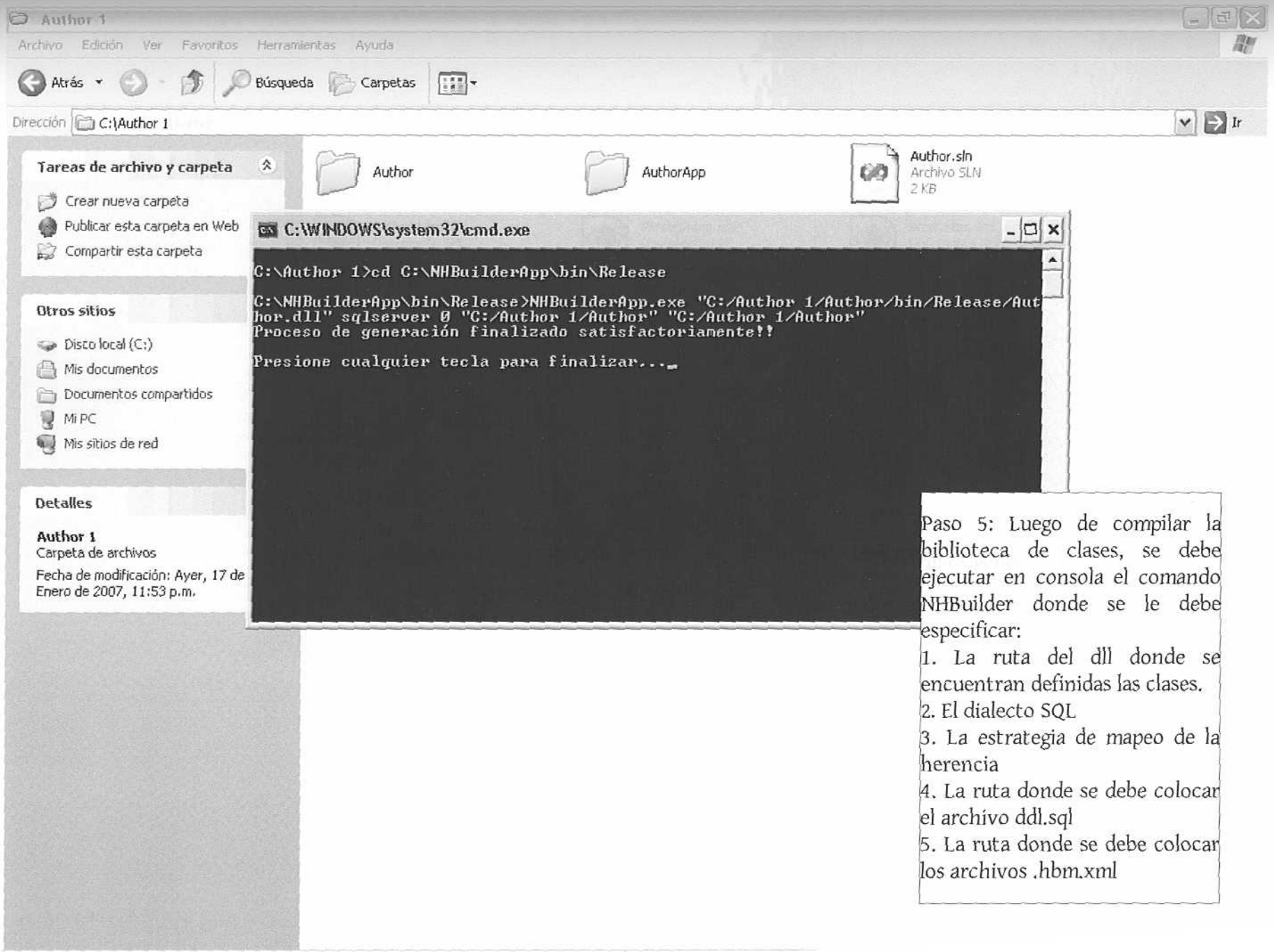
Explorador de soluciones - Solución 'Author' ...

- Solución 'Author' (2 proyectos)
  - **Author**
    - Properties
    - References
      - MappingAttribute
      - System
      - System.Data
      - System.Xml
    - App.config
    - Author.cs
    - Book.cs
    - Person.cs
    - Song.cs
    - Work.cs
  - AuthorApp
    - Properties
    - References
    - Program.cs

Propiedades

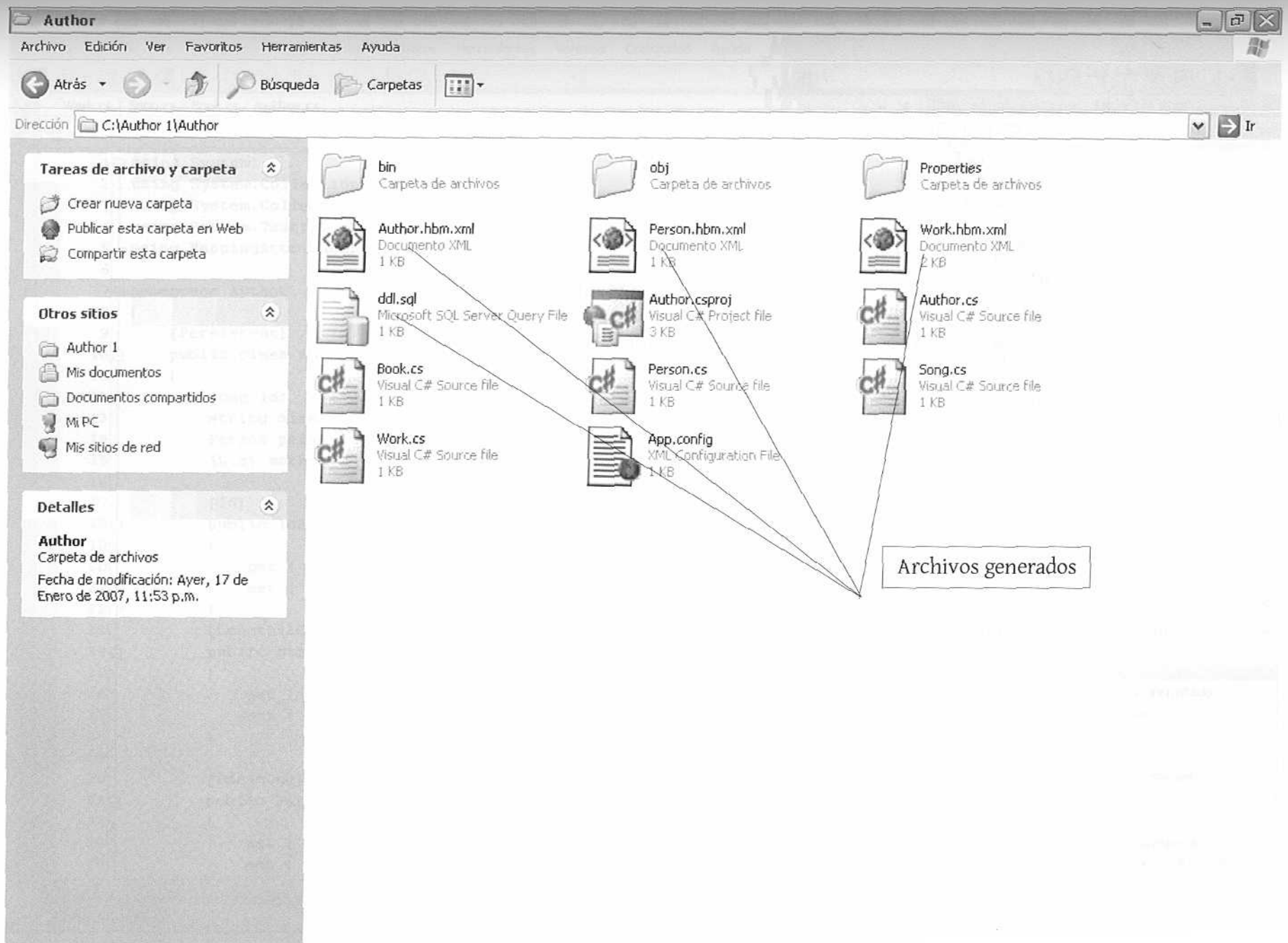
Lista de errores

Cuadro de herramientas



Paso 5: Luego de compilar la biblioteca de clases, se debe ejecutar en consola el comando NHBuilder donde se le debe especificar:

1. La ruta del dll donde se encuentran definidas las clases.
2. El dialecto SQL
3. La estrategia de mapeo de la herencia
4. La ruta donde se debe colocar el archivo ddl.sql
5. La ruta donde se debe colocar los archivos .hbm.xml


Author
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás
Búsqueda
Carpetas
Dirección C:\Author 1\Author
Ir
Tareas de archivo y carpeta
Crear nueva carpeta
Publicar esta carpeta en Web
Compartir esta carpeta
Otros sitios
Author 1
Mis documentos
Documentos compartidos
Mi PC
Mis sitios de red
Detalles
Author
Carpeta de archivos
Fecha de modificación: Ayer, 17 de Enero de 2007, 11:53 p.m.
bin
Carpeta de archivos
obj
Carpeta de archivos
Properties
Carpeta de archivos
Author.hbm.xml
Documento XML
1 KB
Person.hbm.xml
Documento XML
1 KB
Work.hbm.xml
Documento XML
2 KB
ddl.sql
Microsoft SQL Server Query File
1 KB
Author.csproj
Visual C# Project file
3 KB
Author.cs
Visual C# Source file
1 KB
Book.cs
Visual C# Source file
1 KB
Person.cs
Visual C# Source file
1 KB
Song.cs
Visual C# Source file
1 KB
Work.cs
Visual C# Source file
1 KB
App.config
XML Configuration File
1 KB
Archivos generados

Author - Microsoft Visual C# 2005 Express

Archivo Edición Ver Proyecto Generar Depurar Datos Herramientas Ventana Comunidad Ayuda

Work.cs | Song.cs | Book.cs | **Author.cs**

Author.Author | Id

```
1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Text;
5  using MappingAttribute;
6
7  namespace Author
8  {
9      [Persistent]
10     public class Author
11     {
12         long id;
13         string alias;
14         Person person;
15         IList works;
16
17         [Id]
18         public long Id
19         {
20             get { return id; }
21             set { id = value; }
22         }
23         [Length(20)]
24         public string Alias
25         {
26             get { return alias; }
27             set { alias = value; }
28         }
29
30         [OneToOne]
31         public Person Person
32         {
33             get { return person; }
34             set { person = value; }
```

Incluir los archivos como recursos incrustados al proyecto, y crear la base de datos con el archivo ddl.sql

Explorador de soluciones - Solución 'Author' ...

- Solución 'Author' (2 proyectos)
  - **Author**
    - Properties
    - References
    - App.config
    - Author.cs
    - Author.hbm.xml
    - Book.cs
    - Person.cs
    - Person.hbm.xml
    - Song.cs
    - Work.cs
    - Work.hbm.xml
  - AuthorApp
    - Properties
    - References
    - Program.cs

Propiedades

Author.hbm.xml Propiedades del archivo

Acción de generación	Recurso incrustado
Copiar en el directorio de	No copiar
Espacio de nombres de la	
Herramienta personalizad	
Nombre de archivo	Author.hbm.xml
Ruta de acceso completa	D:\Kristal\Tesis\TEG Project\te

Espacio de nombres de la herramienta

Espacio de nombres donde se sitúa el resultado de la herramienta personalizada.

Lista de errores

Listo

Dialectos SQL

Manejador de base de datos	Dialecto
Microsoft SQL Server 2005	SQLServer

Estrategias para el mapeo de la herencia

Estrategia	Valor
Tabla por clase jerárquica	0
Tabla por subclase	1
Tabla por cada clase concreta	2

Atributos

Persistent

```
[Persistent]
public class Author
{
    long id;
    string alias;
    Person person;
    IList works;

    [Id]
    public long Id
    {
        get { return id; }
        set { id = value; }
    }
    [Length(20)]
    public string Alias
    {
        get { return alias; }
        set { alias = value; }
    }

    [OneToOne]
    public Person Person
    {
        get { return person; }
        set { person = value; }
    }

    [ManyToMany(typeof(Work))]
    public IList Works
    {
        get { return works; }
        set { works = value; }
    }
}
```

Para indicar si una clase es persistente, se debe colocar [Persistent] antes de la declaración de la clase

Component

```
[Component]
class Nombre
{
    string primerNombre;
    string iniciales;
    string primerApellido;

    [Length(20)]
    public string PrimerNombre
    {
        get { return primerNombre; }
        set { primerNombre = value; }
    }

    [Length(4)]
    public string Iniciales
    {
        get { return iniciales; }
        set { iniciales = value; }
    }

    [Length(20)]
    public string PrimerApellido
    {
        get { return primerApellido; }
        set { primerApellido = value; }
    }
}
```

Para indicar si una clase es componente de otra, en el sentido de composición, se debe colocar [Component] antes de la declaración de la clase

Nota: Las clases de este tipo no necesitan tener una propiedad del tipo id

Id

```
[Persistent]
public class Author
{
    long id;
    string alias;
    Person person;
    IList works;

    [Id]
    public long Id
    {
        get { return id; }
        set { id = value; }
    }
    [Length(20)]
    public string Alias
    {
        get { return alias; }
        set { alias = value; }
    }

    [OneToOne]
    public Person Person
    {
        get { return person; }
        set { person = value; }
    }

    [ManyToMany(typeof(Work))]
    public IList Works
    {
        get { return works; }
        set { works = value; }
    }
}
```

Para indicar si una propiedad es el identificador, se debe colocar [Id] antes de la declaración de la propiedad

NotNull

```
[Persistent]
class Pago
{
    long idPago;
    int monto;
    bool flag;

    [NotNull]
    public int Monto
    {
        get { return monto; }
        set { monto = value; }
    }

    [Id]
    public long IdPago
    {
        get { return idPago; }
        set { idPago = value; }
    }

    [NonPersistent]
    public bool Flag
    {
        get { return flag; }
        set { flag = value; }
    }
}
```

Para indicar si una propiedad no debe ser null, se debe colocar [NotNull] antes de la declaración de la propiedad, si no se coloca este atributo, se asume que la propiedad puede ser NULL

Lenght

```
[Persistent]
public class Author
{
    long id;
    string alias;
    Person person;
    IList works;

    [Id]
    public long Id
    {
        get { return id; }
        set { id = value; }
    }
    [Length(20)]
    public string Alias
    {
        get { return alias; }
        set { alias = value; }
    }

    [OneToOne]
    public Person Person
    {
        get { return person; }
        set { person = value; }
    }

    [ManyToMany(typeof(Work))]
    public IList Works
    {
        get { return works; }
        set { works = value; }
    }
}
```

Para indicar si la longitud de un string se debe colocar [Lenght(<longitud>)] antes de la declaración de la propiedad

Default

```
[Persistent]
class Persona
{
    long idPersona;
    Nombre nombre;
    DateTime fechaNacimiento;

    [Id]
    public long IdPersona
    {
        get { return idPersona; }
        set { idPersona = value; }
    }

    [Component]
    public Nombre Nombre
    {
        get { return nombre; }
        set { nombre = value; }
    }

    [Default("01/01/1900")]
    public DateTime FechaNacimiento
    {
        get { return fechaNacimiento; }
        set { fechaNacimiento = value; }
    }
}
```

Para indicar el valor por defecto de una propiedad, se debe colocar [Default("<valor>")] antes de la declaración de la propiedad

NonPersistent

```
[Persistent]
class Pago
{
    long idPago;
    int monto;
    bool flag;

    [NotNull]
    public int Monto
    {
        get { return monto; }
        set { monto = value; }
    }

    [Id]
    public long IdPago
    {
        get { return idPago; }
        set { idPago = value; }
    }

    [NonPersistent]
    public bool Flag
    {
        get { return flag; }
        set { flag = value; }
    }
}
```

Para indicar si una propiedad no debe ser persistida, se debe colocar [NonPersistent] antes de la declaración de la propiedad

OneToOne

```
[Persistent]
public class Author
{
    long id;
    string alias;
    Person person;
    IList works;

    [Id]
    public long Id
    {
        get { return id; }
        set { id = value; }
    }
    [Length(20)]
    public string Alias
    {
        get { return alias; }
        set { alias = value; }
    }

    [OneToOne]
    public Person Person
    {
        get { return person; }
        set { person = value; }
    }

    [ManyToMany(typeof(Work))]
    public IList Works
    {
        get { return works; }
        set { works = value; }
    }
}
```

Para indicar una relación del tipo uno a uno, se debe colocar [OneToOne] antes de la declaración de la propiedad

ManyToOne

```
[Persistent]
class Employment
{
    long id;
    DateTime startDate;
    DateTime endDate;
    MonetaryAmount hourlyRate;
    Employee employee;
    Employer employer;

    [Id]
    public long Id
    {
        get { return id; }
        set { id = value; }
    }
    public DateTime StartDate
    {
        get { return startDate; }
        set { startDate = value; }
    }
    public DateTime EndDate
    {
        get { return endDate; }
        set { endDate = value; }
    }
    public MonetaryAmount HourlyRate
    {
        get { return hourlyRate; }
        set { hourlyRate = value; }
    }
    [ManyToOne]
    public Employee Employee
    {
        get { return employee; }
        set { employee = value; }
    }
    [ManyToOne]
    public Employer Employer
    {
        get { return employer; }
        set { employer = value; }
    }
}
```

Para indicar una relación del tipo muchos a uno, se debe colocar [ManyToOne] antes de la declaración de la propiedad

OneToMany

```
[Persistent]
public class Empresa
{
    long id;
    string nombre;
    string rif;
    string direccion;
    int telefono;
    IList clientes;
    IList empleados;

    [Id]
    public long Id...
    [Length(100)]
    public string Nombre...
    [Length(15)]
    public string Rif...
    [Length(255)]
    public string Direccion...
    public int Telefono...

    [OneToMany(relatedClass = typeof(Cliente))]
    public IList Clientes
    {
        get { return clientes; }
        set { clientes = value; }
    }
    [OneToMany(relatedClass = typeof(Empleado))]
    public IList Empleados
    {
        get { return empleados; }
        set { empleados = value; }
    }
}
```

Para indicar una relación del tipo uno a muchos, se debe colocar [OneToMany] antes de la declaración de la propiedad Para las colecciones no genéricas es necesario indicar la clase con la que se relaciona [OneToMany(typeof(<clase>)]

ManyToMany

```
[Persistent]
public class Author
{
    long id;
    string alias;
    Person person;
    IList works;

    [Id]
    public long Id
    {
        get { return id; }
        set { id = value; }
    }
    [Length(20)]
    public string Alias
    {
        get { return alias; }
        set { alias = value; }
    }

    [OneToOne]
    public Person Person
    {
        get { return person; }
        set { person = value; }
    }

    [ManyToMany(typeof(Work))]
    public IList Works
    {
        get { return works; }
        set { works = value; }
    }
}
```

Para indicar una relación del tipo muchos a muchos, se debe colocar [ManyToMany] antes de la declaración de la propiedad
Para las colecciones no genéricas es necesario indicar la clase con la que se relaciona [ManyToMany(typeof(<clase>)]

Component

```
[Persistent]

  class Persona
  {
      long idPersona;
      Nombre nombre;
      DateTime fechaNacimiento;

      [Id]
      public long IdPersona
      {
          get { return idPersona; }
          set { idPersona = value; }
      }

      public Nombre Nombre
      {
          get { return nombre; }
          set { nombre = value; }
      }

      [Default("01/01/1900")]
      public DateTime FechaNacimiento
      {
          get { return fechaNacimiento; }
          set { fechaNacimiento = value; }
      }
  }
```

La clase Nombre esta como [Component], Si a la relación no se le coloca ningún atributo por defecto asume que la relación es de tipo component

APÉNDICE J

Pruebas

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, 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.

Prueba #1.1

Descripción: Consiste en realizar la prueba #1.1 usando la estrategia de mapeo *una tabla por clase jerárquica.*

Prueba #1.2

Descripción: Consiste en realizar la prueba #1.1 usando la estrategia de mapeo *una tabla por subclase.*

Prueba #1.3

Descripción: Consiste en realizar la prueba #1.1 usando la estrategia de mapeo *una tabla por cada clase concreta.*

Prueba #2 Curso alterno

Descripción: Dada una jerarquía de clases, la propiedad identificada Id no se encuentra definida en la superclase.

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: Debe salir la excepción IdNotFoundException.

Prueba #3 Curso alterno

Descripción: Dada una clase que no pertenece a ninguna jerarquía, la propiedad identificada Id no se encuentra definida.

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: Debe salir la excepción IdNotFoundException.

Prueba #4 Curso alterno

Descripción: Dada una clase que posee una relación component (objeto compuesto), la clase que viene siendo el componente no esta definida como component o no es válida. Este caso se puede dar también si el usuario no coloca los atributos a las relaciones del tipo [OneToOne], [ManyToOne], [OneToMany], [ManyToMany]. Cuando el usuario no indica ningún atributo en una propiedad que no es del tipo de dato primitivo, se asume que es una relación Component.

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: Debe salir la excepción InvalidRelatedClassException.

Prueba #5 Curso alterno

Descripción: Dada una propiedad de una clase, del tipo string, y no está definido el atributo [Lenght(<longitud>)]

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: Debe salir la excepción LenghtNotFoundException.

Prueba #6 Curso alterno

Descripción:: La estrategia para el mapeo de la herencia no es válido.

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: Debe salir la excepción InvalidInheritanceStrategyException.

Prueba #7 Curso alterno

Descripción:: El dialecto SQL no es válido, o no es soportado aún por la herramienta.

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: Debe salir la excepción DialectNotFoundException.

Prueba #8 Curso alterno

Descripción:: Dada una propiedad con tipo de dato primitivo, no soportado aún por la herramienta.

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: Debe salir la excepción InvalidDataTypeException.

Prueba #8 Curso alterno

Descripción:: Dada una librería de clases persistentes, que se desea generar los archivos usando la estrategia de mapeo *una tabla por clase concreta*, pero existen clases que se relacionan con las clases más altas de la jerarquía, el polimorfismo no es soportado para ciertas relaciones cuando se utiliza la estrategia una tabla por clase concreta.

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: Debe salir la excepción RelationNotAllowedException.

Prueba #9 Curso alterno

Descripción:: Dada una clase que no posea Id (lo hereda de su superclase), y no posea una sola propiedad de tipo primitivo.

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: Debe salir la excepción InvalidTableException.

Prueba #10 Curso alterno

Descripción:: Dada una relación del tipo [OneToOne] o [ManyToOne], y la clase relacionada no es una clase persistente válida, es decir, es una clase del tipo [Component].

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: Debe salir la excepción InvalidRelatedClassException.

Apéndice K

Diagramas de clases usados para las pruebas y sus resultados

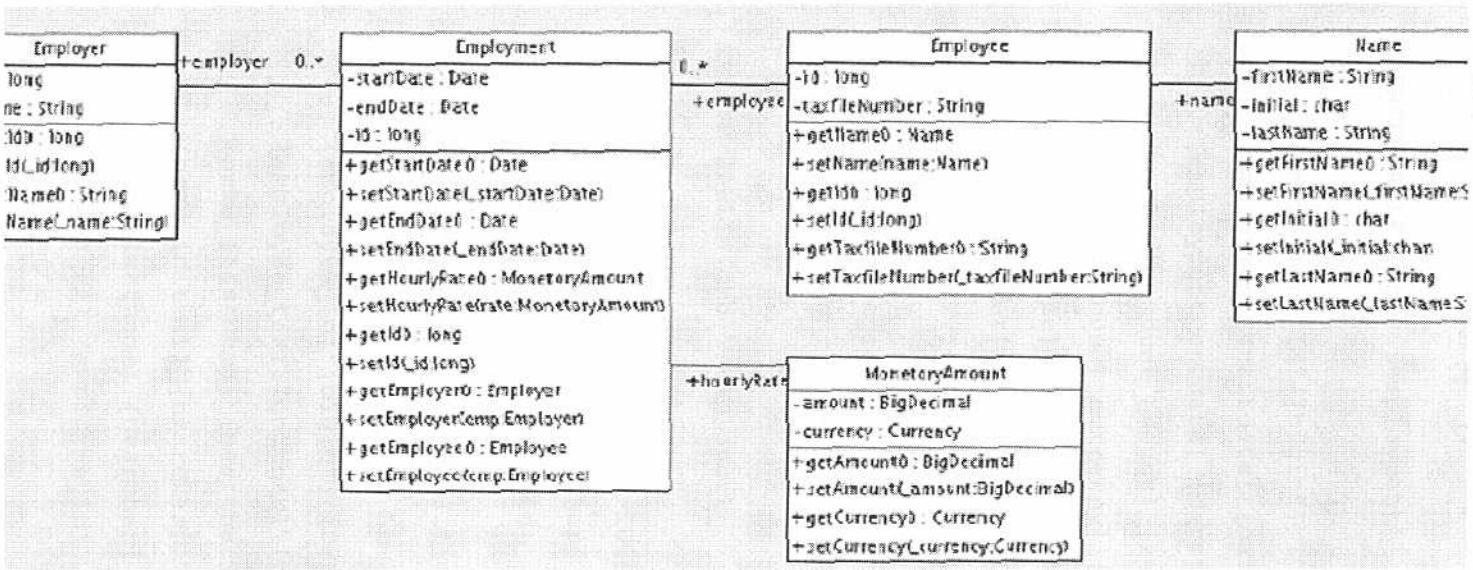


Figura L.1: Diagrama de clases 1 - prueba.
Fuente: NHiberbnate Reference.

Resultados:

ddl.sql

```
CREATE TABLE [dbo].[employment]
(
      [employment_id] [bigint] IDENTITY(1,1) NOT NULL,
      [employment_startdate] [datetime] NULL ,
      [employment_enddate] [datetime] NULL ,
      [monetaryamount_amount] [real] NULL ,
      [monetaryamount_currency] [real] NULL ,
      [fk_employee] [bigint] NULL,
      [fk_employer] [bigint] NULL,
      PRIMARY KEY (employment_id) ,
      FOREIGN KEY (fk_employee) REFERENCES
employee(employee_id),
      FOREIGN KEY (fk_employer) REFERENCES employer(employer_id)
)
CREATE TABLE [dbo].[employer]
(
      [employer_id] [bigint] IDENTITY(1,1) NOT NULL,
      [employer_name] [nvarchar] (100) NULL ,
      PRIMARY KEY (employer_id)
)
CREATE TABLE [dbo].[employee]
(
      [employee_id] [bigint] IDENTITY(1,1) NOT NULL,
      [employee_taxfilenumber] [nvarchar] (100) NULL ,
      [name_firstname] [nvarchar] (20) NULL ,
      [name_initial] [nvarchar] (1) NULL ,
      [name_lastname] [nvarchar] (20) NULL ,
      PRIMARY KEY (employee_id)
)
```

Employer.hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Employee.Employer, Employee" table="employer">
    - <id name="Id" column="employer_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="Name" column="employer_name" type="String" />
    </class>
  </hibernate-mapping>
```

Employee.hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Employee.Employee, Employee" table="employee">
    - <id name="Id" column="employee_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="TaxfileNumber" column="employee_taxfilenumber" type="String" />
    - <component name="Name" class="Employee.Name, Employee">
        <property name="FirstName" column="name_firstname" type="String" />
        <property name="Initial" column="name_initial" type="String" />
        <property name="LastName" column="name_lastname" type="String" />
      </component>
    </class>
  </hibernate-mapping>
```

Employment.hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Employee.Employment, Employee" table="employment">
    - <id name="Id" column="employment_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="StartDate" column="employment_startdate" type="datetime" />
      <property name="EndDate" column="employment_enddate" type="datetime" />
    - <component name="HourlyRate" class="Employee.MonetaryAmount, Employee">
        <property name="Amount" column="monetaryamount_amount" type="Single" />
        <property name="Currency" column="monetaryamount_currency" type="Single" />
      </component>
      <many-to-one name="Employee" class="Employee.Employee,Employee" column="fk_employee" />
      <many-to-one name="Employer" class="Employee.Employer,Employee" column="fk_employer" />
    </class>
  </hibernate-mapping>
```

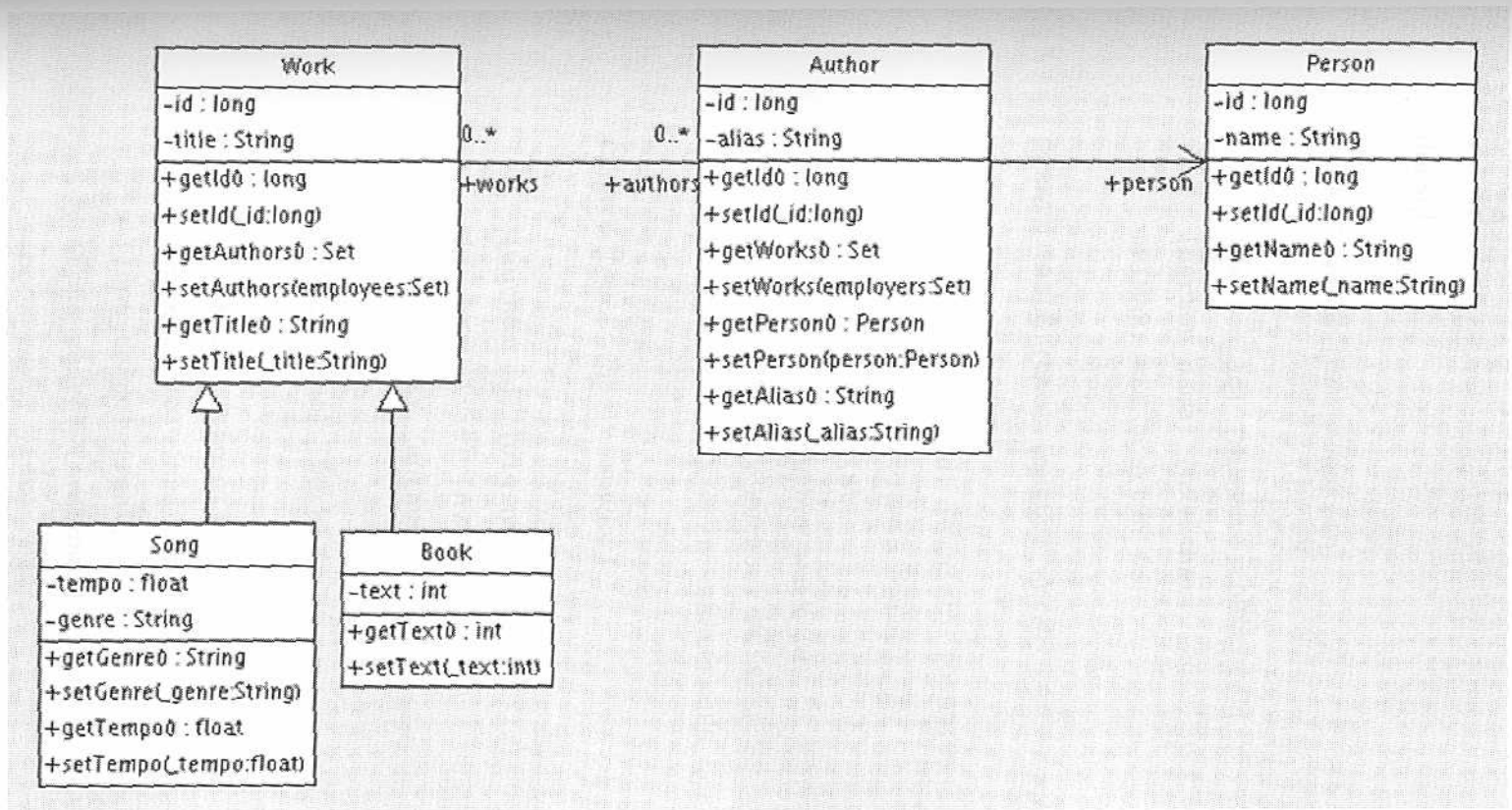


Figura L.2: Diagrama de clases 2 - prueba.
Fuente: NHiberbnate Reference.

Resultados

Tabla por clase jerárquica

ddl.sql

```
CREATE TABLE [dbo].[person]
(
      [person_id] [bigint] IDENTITY(1,1) NOT NULL,
      [person_name] [nvarchar] (20) NULL ,
      PRIMARY KEY (person_id)
)
CREATE TABLE [dbo].[author_work]
(
      [author_work_id] [bigint] IDENTITY(1,1) NOT NULL,
      [fk_author] [bigint] NULL,
      [fk_work] [bigint] NULL,
      PRIMARY KEY (author_work_id) ,
      FOREIGN KEY (fk_author) REFERENCES author(author_id),
      FOREIGN KEY (fk_work) REFERENCES work(work_id)
)
CREATE TABLE [dbo].[author]
(
      [author_id] [bigint] IDENTITY(1,1) NOT NULL,
      [author_alias] [nvarchar] (20) NULL ,
      PRIMARY KEY (author_id)
)
CREATE TABLE [dbo].[work]
(
      [work_id] [bigint] IDENTITY(1,1) NOT NULL,
      [work_title] [nvarchar] (100) NULL ,
      [song_tempo] [real] NULL ,
      [song_genre] [nvarchar] (100) NULL ,
      [book_text] [int] NULL ,
```

```
        [work_type] [nvarchar] (100)  NOT NULL ,
        PRIMARY KEY (work_id)
```

Person.hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Author.Person, Author" table="person">
    - <id name="Id" column="person_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="Name" column="person_name" type="String" />
    </class>
  </hibernate-mapping>
```

Autor.hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Author.Author, Author" table="author">
    - <id name="Id" column="author_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="Alias" column="author_alias" type="String" />
      <one-to-one name="Person" class="Author.Person,Author" />
    - <list name="Works" table="author_work">
        <key column="fk_author" />
        <index column="author_work_id" type="Int64" />
        <many-to-many column="fk_work" class="Author.Work,Author" />
      </list>
    </class>
  </hibernate-mapping>
```

Work.hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Author.Work, Author" table="work">
    - <id name="Id" column="work_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <discriminator column="work_type" type="String" />
      <property name="Title" column="work_title" type="String" />
    - <list name="Authors" table="author_work">
        <key column="fk_work" />
        <index column="author_work_id" type="Int64" />
        <many-to-many column="fk_author" class="Author.Author,Author" />
      </list>
    - <subclass name="Author.Song, Author" discriminator-value="SONG">
        <property name="Tempo" column="song_tempo" type="Single" />
        <property name="Genre" column="song_genre" type="String" />
      </subclass>
    - <subclass name="Author.Book, Author" discriminator-value="BOOK">
        <property name="Text" column="book_text" type="Int32" />
      </subclass>
    </class>
  </hibernate-mapping>
```

Tabla por subclase

ddl.sql

```
CREATE TABLE [dbo].[work_author]
(
      [work_author_id] [bigint] IDENTITY(1,1) NOT NULL,
      [fk_work] [bigint] NULL,
      [fk_author] [bigint] NULL,
      PRIMARY KEY (work_author_id) ,
      FOREIGN KEY (fk_work) REFERENCES work(work_id),
      FOREIGN KEY (fk_author) REFERENCES author(author_id)
)
CREATE TABLE [dbo].[work]
(
      [work_id] [bigint] IDENTITY(1,1) NOT NULL,
      [work_title] [nvarchar] (100) NULL ,
      PRIMARY KEY (work_id)
)
CREATE TABLE [dbo].[book]
(
      [book_text] [int] NULL ,
      [work_id] [bigint] NULL,
      FOREIGN KEY (work_id) REFERENCES work(work_id)
)
CREATE TABLE [dbo].[person]
(
      [person_id] [bigint] IDENTITY(1,1) NOT NULL,
      [person_name] [nvarchar] (20) NULL ,
      PRIMARY KEY (person_id)
)
CREATE TABLE [dbo].[author]
(
      [author_id] [bigint] IDENTITY(1,1) NOT NULL,
```

```
        [author_alias] [nvarchar] (20) NULL ,
        PRIMARY KEY (author_id)
    )
    CREATE TABLE [dbo].[song]
    (
        [song_tempo] [real] NULL ,
        [song_genre] [nvarchar] (100) NULL ,
        [work_id] [bigint] NULL,
        FOREIGN KEY (work_id) REFERENCES work(work_id)
    )
```

Person.hbm.xml

```
  <?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Author.Person, Author" table="person">
    - <id name="Id" column="person_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="Name" column="person_name" type="String" />
    </class>
  </hibernate-mapping>
```

Autor.hbm.xml

```
  <?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Author.Author, Author" table="author">
    - <id name="Id" column="author_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="Alias" column="author_alias" type="String" />
      <one-to-one name="Person" class="Author.Person,Author" />
    - <list name="Works" table="work_author">
        <key column="fk_author" />
        <index column="work_author_id" type="Int64" />
        <many-to-many column="fk_work" class="Author.Work,Author" />
      </list>
    </class>
  </hibernate-mapping>
```

Work.hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Author.Work, Author" table="work">
    - <id name="Id" column="work_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="Title" column="work_title" type="String" />
    - <list name="Authors" table="work_author">
        <key column="fk_work" />
        <index column="work_author_id" type="Int64" />
        <many-to-many column="fk_author" class="Author.Author,Author" />
      </list>
    - <joined-subclass name="Author.Book, Author" table="book">
        <key column="work_id" />
        <property name="Text" column="book_text" type="Int32" />
      </joined-subclass>
    - <joined-subclass name="Author.Song, Author" table="song">
        <key column="work_id" />
        <property name="Tempo" column="song_tempo" type="Single" />
        <property name="Genre" column="song_genre" type="String" />
      </joined-subclass>
    </class>
  </hibernate-mapping>
```

Tabla por cada clase concreta

Se hicieron modificaciones al diagrama para poder generar los archivos con esta estrategia

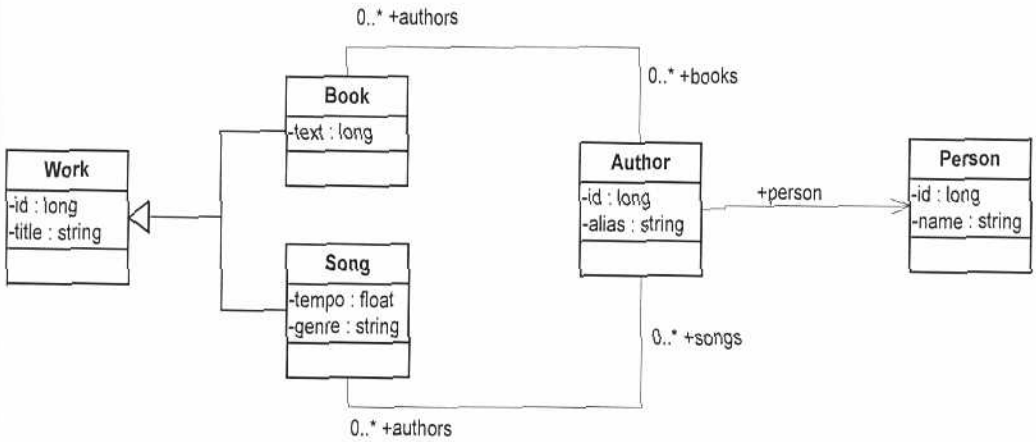


Figura L.3: Diagrama de clases 2 modificado - prueba.
Diseño: Lucero, K. (2007)

ddl.sql

```
CREATE TABLE [dbo].[song_author]
{
        [song_author_id] [bigint] IDENTITY(1,1) NOT NULL,
        [fk_song] [bigint] NULL,
        [fk_author] [bigint] NULL,
        PRIMARY KEY (song_author_id) ,
        FOREIGN KEY (fk_song) REFERENCES song(work_id),
        FOREIGN KEY (fk_author) REFERENCES author(author_id)
}
CREATE TABLE [dbo].[song]
{
        [work_id] [bigint] IDENTITY(1,1) NOT NULL,
```

```
        [song_tempo] [real] NULL ,
        [song_genre] [nvarchar] (100) NULL ,
        [work_title] [nvarchar] (100) NULL ,
        PRIMARY KEY (work_id)
)
CREATE TABLE [dbo].[book_author]
(
        [book_author_id] [bigint] IDENTITY(1,1) NOT NULL,
        [fk_book] [bigint] NULL,
        [fk_author] [bigint] NULL,
        PRIMARY KEY (book_author_id) ,
        FOREIGN KEY (fk_book) REFERENCES book(work_id),
        FOREIGN KEY (fk_author) REFERENCES author(author_id)
)
CREATE TABLE [dbo].[book]
(
        [work_id] [bigint] IDENTITY(1,1) NOT NULL,
        [book_text] [int] NULL ,
        [work_title] [nvarchar] (100) NULL ,
        PRIMARY KEY (work_id)
)
CREATE TABLE [dbo].[person]
(
        [person_id] [bigint] IDENTITY(1,1) NOT NULL,
        [person_name] [nvarchar] (20) NULL ,
        PRIMARY KEY (person_id)
)
CREATE TABLE [dbo].[author]
(
        [author_id] [bigint] IDENTITY(1,1) NOT NULL,
        [author_alias] [nvarchar] (20) NULL ,
        PRIMARY KEY (author_id)
)
```

Person.hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Author.Person, Author" table="person">
    - <id name="Id" column="person_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="Name" column="person_name" type="String" />
    </class>
  </hibernate-mapping>
```

Autor.hbm.xml

```
<?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Author.Author, Author" table="author">
    - <id name="Id" column="author_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="Alias" column="author_alias" type="String" />
      <one-to-one name="Person" class="Author.Person,Author" />
    - <list name="Songs" table="song_author">
        <key column="fk_author" />
        <index column="song_author_id" type="Int64" />
        <many-to-many column="fk_song" class="Author.Song,Author" />
      </list>
    - <list name="Books" table="book_author">
        <key column="fk_author" />
        <index column="book_author_id" type="Int64" />
        <many-to-many column="fk_book" class="Author.Book,Author" />
      </list>
    </class>
  </hibernate-mapping>
```

Book.hbm.xml

```
  <?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Author.Book, Author" table="book">
    - <id name="Id" column="work_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="Text" column="book_text" type="Int32" />
      <property name="Title" column="work_title" type="String" />
    - <list name="Authors" table="book_author">
        <key column="fk_book" />
        <index column="book_author_id" type="Int64" />
        <many-to-many column="fk_author" class="Author.Author,Author" />
      </list>
    </class>
  </hibernate-mapping>
```

Song.hbm.xml

```
  <?xml version="1.0" encoding="utf-8" ?>
- <hibernate-mapping xmlns="urn:nhibernate-mapping-2.0">
  - <class name="Author.Song, Author" table="song">
    - <id name="Id" column="work_id" type="Int64" unsaved-value="0">
        <generator class="identity" />
      </id>
      <property name="Tempo" column="song_tempo" type="Single" />
      <property name="Genre" column="song_genre" type="String" />
      <property name="Title" column="work_title" type="String" />
    - <list name="Authors" table="song_author">
        <key column="fk_song" />
        <index column="song_author_id" type="Int64" />
        <many-to-many column="fk_author" class="Author.Author,Author" />
      </list>
    </class>
  </hibernate-mapping>
```

Apéndice L

Resultados de las pruebas de evaluación del usuario

Resultados usuario 1:

Fase 1	Tiempo
Tiempo entrenamiento NHibernate	27 minutos
Tiempo creando la base de datos	24 minutos
Tiempo creando los archivos XML de mapeo	40 minutos
Total	**91 minutos** --> 1h 31 minutos

Tabla 9: Resultados de la prueba fase 1 al usuario 1.

Fuente: Lucero, K. (2007)

Fase 2	Tiempo
Tiempo entrenamiento NHBuilder	5 minutos
Tiempo escribiendo los atributos	8 minutos
Tiempo ejecutando la herramienta	2 minutos
Total	**15 minutos**

Tabla 10: Resultados de la prueba fase 2 al usuario 1.

Fuente: Lucero, K. (2007)

Resultados usuario 2:

Fase 1	Tiempo
Tiempo creando la base de datos	12 minutos
Tiempo creando los archivos XML de mapeo	16 minutos
Total	**28 minutos**

Tabla 11: Resultados de la prueba fase 1 al usuario 2.

Fuente: Lucero, K. (2007)

Fase 2	Tiempo
Tiempo entrenamiento NHBuilder	5 minutos
Tiempo escribiendo los atributos	2 minutos
Tiempo ejecutando la herramienta	40 segundos
Total	**7 minutos 40 segundos**

Tabla 12: Resultados de la prueba fase 2 al usuario 2.

Fuente: Lucero, K. (2007)

Apéndice M

Estándares de codificación de los archivos DDL

La estructura de la base de datos posee el siguiente estándar:

El nombre de la tablas es el mismo nombre de la clase pero en minúculas

Ejemplo:

Clase: Persona

Tabla: persona

Las propiedades son mapedas como columnas, y el nombre de la columna posee el nombre de la clase a la cual pertenece seguido del caracter "_" y el nombre de la propiedad, todo en minúsculas.

Ejemplo:

Clase: Persona

Propiedad: Nombre

Columna: persona_nombre

Las tablas de intersección como resultado de una relación N..M, tienen como nombre el nombre de alguna de las dos clases seguido del caracter "_" y luego el nombre de la otra clase, todo en minúsculas.

Ejemplo:

Clase 1: Factura

Clase 2: Producto

Tabla intersección: factura_producto, o producto_factura

Las columnas llamadas discriminadores, utilizados cuando se utiliza la estrategia de mapeo dela herencia "una tabla por clase jerárquica", llevan como nombre la clase jerárquica seguido de "_type", en minúsculas.

Ejemplo:

Clase: Persona

Discriminador: persona_type

Las claves foráneas llevan como nombre el prefijo "fk_" seguido del nombre de la clase a la que se le hace referencia, todo en minúsculas.

Ejemplo

Clase: Empleado

Clase a la que se le hace referencia: Empresa

Clave foránea: fk_empresa

Este mismo estándar se mantiene en los archivos de mapeo, al momento de especificar el nombre de la tabla, de las columnas, etc.

Apéndice N

Collaboration Responsability Cards

Clase: PersistentClass	
Descripción:: Representa las clases persistentes del usuario. Sus campos almacenan la información necesaria para la generación de los archivos de mapeo y la estructura de la base de datos. Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo, ya sea como clase, subclase o componente.	
Responsabilidades	**Colaboradores**
IsSuperClass MappingPropertys MappingRelations MappingComponent MappingSubClass MappingJoinedSubClass	Property Relation

Clase: Property	
Descripción:: Representa las propiedades de las clases persistentes del usuario. Sus campos almacenan la información necesaria para la generación de los archivos de mapeo y la estructura de la base de datos. Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo, ya sea como id de la clase o como una simple propiedad.	
Responsabilidades	**Colaboradores**
Mapping	

Clase: NHibernateMapping	
Descripción:: Se encarga de transformar los tipos de datos de .Net a los tipos de datos manejados por NHibernate	
Responsabilidades	**Colaboradores**
GetNHibernateType	

Clase: Relation	
Descripción:: Representa a las relaciones entre las clases persistentes del usuario. Sus campos almacenan toda la información necesaria para la generación de los archivos de mapeo y la estructura de la base de datos (las claves foráneas). Todas las clases que heredan de esta clase Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo, ya sea como id de la clase o como una simple propiedad.	
Responsabilidades	**Colaboradores**
Mapping	

Clase: RelationFactory	
Descripción:: Fábrica de relaciones entre clases	
Responsabilidades	**Colaboradores**
GetRelation	Relation OneToOne ManyToOne OneToMany ManyToMany Component

Clase: OneToOne	
Descripción:: Representa a las relaciones uno a uno entre las clases. Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo.	
Responsabilidades	**Colaboradores**
Mapping	

Clase: ManyToOne	
Descripción:: Representa a las relaciones muchos a uno entre las clases Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo.	
Responsabilidades	**Colaboradores**
Mapping	

Clase: OneToMany	
Descripción:: Representa a las relaciones uno a muchos entre las clases. Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo.	
Responsabilidades	**Colaboradores**
Mapping	Collection

Clase: ManyToMany	
Descripción:: Representa a las relaciones muchos a muchos entre las clases Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo.	
Responsabilidades	**Colaboradores**
Mapping	Collection

Clase: Component	
Descripción:: Representa a las relaciones de objetos compuestos contenidos en una clase. Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo.	
Responsabilidades	**Colaboradores**
Mapping	

Clase: Collection	
Descripción:: Representa a las colecciones de objetos para las relaciones de tipo uno a muchos y muchos a muchos.	
Responsabilidades	**Colaboradores**
MappingOneToMany MappingManyToMany	

Clase: CollectionFactory	
Descripción:: Fábrica de colecciones	
Responsabilidades	**Colaboradores**
GetCollection	Collection Bag Set List Map

Clase: Bag	
Descripción:: Representa cualquier colección de .Net a la terminología de NHibernate Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo.	
Responsabilidades	**Colaboradores**
MappingOneToMany MappingManyToMany	

Clase: Set	
Descripción:: Representa a la colecciones Iesi.Collections.ISet (coleccion definida por NHibernate.org) Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo.	
Responsabilidades	**Colaboradores**
MappingOneToMany MappingManyToMany	

Clase: List	
Descripción:: Representa a la colecciones System.Collections.IList y System.Collection.Generic.List de .Net. Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo.	
Responsabilidades	**Colaboradores**
MappingOneToMany MappingManyToMany	

Clase: Map	
Descripción:: Representa a la colecciones System.Collections.IDictionary y System.Collection.Generic.Dictionary de .Net. Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo.	
Responsabilidades	**Colaboradores**
MappingOneToMany MappingManyToMany	

Clase: ReflectionProcess	
Descripción:: Lleva a cabo el proceso de reflexión sobre el ensamblado con las clases persistentes del usuario, para obtener la estructura de clases con sus propiedades y relaciones.	
Responsabilidades	**Colaboradores**
GetStructureOfClasses GetClasses GetPropertysAndRelations GetPropertyAttributes GetRelationAttributes	Persistent PersistentClass Property Relation RelationFactory CollectionFactory

Clase: Table	
Descripción:: Representa un tabla en el modelo relacional. Almacena la información pertinente para crear la tabla dentro de la estructura de la base de datos.	
Responsabilidades	**Colaboradores**
ContainsFK AddColumns	Column ForeignKey PersistentClass Property

Clase: Column	
Descripción:: Representa la columna de una tabla en el modelo relacional.	
Responsabilidades	**Colaboradores**

Clase: ForeignKey	
Descripción:: Representa las claves foráneas de una tabla en el modelo relacional, que son el resultado de las relaciones entre las clases persistentes.	
Responsabilidades	**Colaboradores**

Clase: TransformModel	
Descripción:: Lleva a cabo el proceso de conversión de la estructura de clases a un modelo relacional.	
Responsabilidades	**Colaboradores**
TransformToRelationalModel	Table PersistentClass TableConstructor TableConstructorFactory DTODialect DTOFactory

Clase: TableConstructor	
Descripción:: Transforma la estructura de clases a un modelo relacional dependiendo de la estrategia de mapeo de la herencia.	
Responsabilidades	**Colaboradores**
GetTables ContainsTable GetFK	

Clase: TableConstructorFactory	
Descripción:: Fábrica de las distintas formas de generar un modelo relacional dependiendo de la estrategia para el mapeo de la herencia	
Responsabilidades	**Colaboradores**
GetTableConstructor	TableConstructor TablePerClassHierarchy TablePerSubclass TablesPerConcreteClass

Clase: TablePerClassHierarchy	
Descripción:: Transforma la estructura de clases a un modelo relacional usando la estrategia "una tabla por clase jerárquica"	
Responsabilidades	**Colaboradores**
GetTables GetTable GetRelations GetRelationsManyToOne GetRelationsOneToMany GetRelationManyToMany	Table Column ForeignKey PersistentClass Relation

Clase: TablePerSubclass	
Descripción:: Transforma la estructura de clases a un modelo relacional usando la estrategia "una tabla por subclase"	
Responsabilidades	**Colaboradores**
GetTables GetTable GetRelations GetRelationsManyToOne GetRelationsOneToMany GetRelationManyToMany	Table Column ForeignKey PersistentClass Relation

Clase: TablesPerConcreteClass	
Descripción:: Transforma la estructura de clases a un modelo relacional usando la estrategia "una tabla por cada clase concreta"	
Responsabilidades	**Colaboradores**
GetTables GetTable GetRelations GetRelationsManyToOne GetRelationsOneToMany GetRelationManyToMany	Table Column ForeignKey PersistentClass Relation

Clase: DTODialect	
Descripción:: Representa a todos los dialectos SQL	
Responsabilidades	**Colaboradores**
GetDbType CreateTable	

Clase: DTOFactory	
Descripción:: Fábrica de los dialectos SQL	
Responsabilidades	**Colaboradores**
GetDialect	DTODialect SQLServerDTODialect

Clase: SQLServerDTODialect	
Descripción:: Realiza el proceso de generación del DDL para un modelo relacional usando el dialecto SQL para el manejador MS SQL Server	
Responsabilidades	**Colaboradores**
GetDbType CreateTable GetId GetColumns GetFKs PKConstraint FKConstraint	Table Column ForeignKey

Clase: GenerateMapping	
Descripción:: Lleva a cabo el proceso de generación de los archivos de mapeo para las clases persistentes dependiendo de la estrategia de mapeo de la herencia	
Responsabilidades	**Colaboradores**
GenerateNHibernateMapping	MappingClass MappingFactory PersistentClass

Clase: MappingClass	
Descripción:: Genera los archivos de mapeo a partir de la estructura de clases dependiendo de la estrategia de mapeo de la herencia.	
Responsabilidades	**Colaboradores**
MappingClasses	

Clase: MappingFactory	
Descripción:: Fábrica de las distintas formas de generar los archivos de mapeo dependiendo de la estrategia para el mapeo de la herencia	
Responsabilidades	**Colaboradores**
GetMapping	MappingClass MappingTablePerClassHierarchy MappingTablePerSubclass MappingTablesPerConcreteClass

Clase: MappingTablePerClassHierarchy	
Descripción:: Genera los archivos de mapeo para la estructura de clases usando la estrategia "una tabla por clase jerárquica"	
Responsabilidades	**Colaboradores**
MappingClases	PersistentClass

Clase: MappingTablePerSubclass	
Descripción:: Genera los archivos de mapeo para la estructura de clases usando la estrategia "una tabla por subclase"	
Responsabilidades	**Colaboradores**
MappingClases	PersistentClass

Clase: MappingTablesPerConcreteClass	
Descripción:: Genera los archivos de mapeo para la estructura de clases usando la estrategia "una tabla por cada clase concreta"	
Responsabilidades	**Colaboradores**
MappingClases	PersistentClass

Apéndice Ñ

Introducción a los CRC

[RUB1998] Es una técnica de "lluvia de ideas" o brainstorming, Es usada para la construcción inicial del modelo Orientado a Objetos, apoya la exploración de las distintas alternativas de diseño.

Las CRC son una colección de cartas, divididas en tres secciones: Clase, responsabilidad y colaborador.

Clase

Una Clase representa una colección de objetos similares. Los objetos son las cosas de interés al sistema siendo modelado. Pueden ser una persona, un lugar, una cosa, o cualquier otro concepto importante del sistema. El nombre de la Clase aparece a en la cima de la tarjeta CRC.

Responsabilidad

Una Responsabilidad es algo que la clase sabe (conoce) o hace. Estas

responsabilidades son cosas que la clase tiene el conocimiento sobre sí.

Por ejemplo, una clase persona podría tener el conocimiento (y la responsabilidad) sobre su nombre, dirección, y el número de teléfono. En otro ejemplo una clase de automóvil podría tener el conocimiento de su tamaño, su número de puertas, o podría ser capaz de hacer cosas como la parada e ir. Las Responsabilidades de una clase aparecen a lo largo del lado izquierdo de la tarjeta CRC.

Colaborador

Un Colaborador es otra clase que es usada para conseguir información, o para realizar acciones para la clase. Los Colaboradores de una clase aparecen a lo largo del lado derecho de la tarjeta CRC.

El equipo CRC

El tamaño total del equipo del CRC es importante, y los participantes activos deben ser limitados a 6 . Por lo general existen tres tipos de participantes activos en la sesión del CRC.

1. Los usuarios del dominio
2. Analista de diseño OO
3. Facilitador

Seleccionar un escenario

Se debe seleccionar el escenario al comenzar la sesión CRC. A menudo se suele dejar de lado el brainstorming inicial para identificar un escenario apropiado.

El escenario escogido bien debería ser documentado y muy específico. Cualquier escenario relacionado debería ser modelado separadamente. A menudo es deseable escoger un escenario fácil, para empezar.

Si ya existen los diagramas de casos de uso, un buen escenario para comenzar podría ser cualquier "curso normal" de alguno de estos casos de uso. La utilización solamente del "curso normal" asegura que la sesión se fija en un escenario particular, y no en el caso de uso entero.

Es importante acentuar la utilización sólo del "curso normal" del caso de uso, y la no concentración en cualquiera de los "curso alternativo". Esto le permitirá al facilitador asegurar que el grupo no diverge en caminos alternativos por un caso de uso.

También durante este tiempo de brainstorming un juego inicial de clases puede ser identificado.

Identificando las clases

Una Clase representa una colección de objetos similares. Los objetos son las cosas de interés al sistema siendo modelado. Ellos pueden ser una persona, un lugar, una cosa, o cualquier otro concepto importante del sistema.

Hay muchos modos de identificar clases. Uno del los más fáciles es comenzar con la extracción de los sustantivos. Se deben identificar los sustantivos en la declaración del problema y/o del escenario del caso de uso. Los sustantivos extraídos son clases de candidatas excelentes.

Otra forma de identificar clases es la de buscar los artículos que actúan recíprocamente con el sistema, o las cosas que son parte del sistema. Pregunte si hay un cliente del sistema, e identificar con qué el cliente actúa recíprocamente. Las pantallas e informes representan clases de interfaz, pero por una sesión CRC, una clase sola GUI puede representar estos.

Si una clase no puede ser llamada con menos de tres palabras, entonces esto no es probablemente una clase, pero sí una responsabilidad de otra clase.

Identificando las responsabilidades

Una Responsabilidad es algo que la clase sabe (conoce) o hace. Estas responsabilidades son cosas que la clase tiene el conocimiento sobre sí, o cosas que la clase puede hacer con el conocimiento que esto tiene.

Con la extracción de todos los verbos en una declaración de problema y/o el escenario de un caso de uso, se obtienen por lo general las acciones que deben ser realizadas por las clases del sistema.

Identificando los colaboradores

La colaboración ocurre cuando una clase necesita la información que no tiene.

Las clases saben (conocen) cosas específicas sobre si mismas. Muy a menudo para realizar una tarea una clase necesita la información que no tiene. A menudo es necesario conseguir esta información de otra clase, en forma de la colaboración.

La colaboración también puede ocurrir si una clase tiene que modificar la información que no tiene. Una característica (propiedad) de información de la cual una clase sabe sí mismo es la capacidad de poner al día la información. A menudo una clase querrá poner al día la información que no tiene. Cuando esto pasa, la clase a menudo preguntará otra clase, en forma de una colaboración, poner al día la información para ello.

Apéndice O

Propiedades

[MSDN7]Las propiedades son miembros que ofrecen un mecanismo flexible para leer, escribir o calcular los valores de campos privados. Se pueden utilizar las propiedades como si fuesen miembros de datos públicos, aunque en realidad son métodos especiales denominados descriptores de acceso. De este modo, se puede tener acceso a los datos con facilidad, a la vez que proporciona la seguridad y flexibilidad de los métodos.

En este ejemplo, la clase TimePeriod almacena un período de tiempo. Internamente, la clase almacena el tiempo en segundos, pero se proporciona una propiedad denominada Hours que permite que un cliente especifique el tiempo en horas. Los descriptores de acceso de la propiedad Hours realizan la conversión entre horas y segundos.

```
C#
class TimePeriod
{
    private double seconds;

    public double Hours
    {
        get { return seconds / 3600; }
        set { seconds = value * 3600; }
    }
}

class Program
{
    static void Main()
    {
        TimePeriod t = new TimePeriod();

        // Assigning the Hours property causes the 'set'
accessor to be called.
        t.Hours = 24;

        // Evaluating the Hours property causes the 'get'
accessor to be called.
        System.Console.WriteLine("Time in hours: " + t.Hours);
    }
}
```

Resultados

Time in hours: 24

Información general sobre propiedades

- Las propiedades permiten que una clase exponga una manera pública de obtener y establecer valores, ocultando el código de implementación o comprobación.
- El descriptor de acceso de una propiedad get se utiliza para devolver el

valor de la propiedad y el descriptor de acceso set se utiliza para asignar un nuevo valor. Estos descriptores de acceso pueden tener niveles de acceso diferentes. Para obtener más información, vea Accesibilidad del descriptor de acceso.

- La palabra clave value se utiliza para definir el valor asignado por el indizador set.
- Las propiedades que no implementan un método set son de sólo lectura.

Apéndice P

Reflexión

[MSDN3]Las clases del espacio de nombres System.Reflection, junto con System.Type, permiten obtener información acerca de los ensamblados cargados y los tipos definidos en los mismos, como clases, interfaces y tipos de valor. También se puede utilizar la reflexión para crear instancias de tipo en tiempo de ejecución, así como para invocarlas y obtener acceso a las mismas.

[MSDN4]El cargador de Common Language Runtime administra los dominios de aplicación. Dicha administración incluye la carga de cada ensamblado en el dominio de aplicación apropiado y el control del diseño de memoria correspondiente a la jerarquía de tipos de cada ensamblado.

Los ensamblados contienen módulos, los módulos contienen tipos y éstos últimos contienen miembros. La reflexión proporciona objetos que encapsulan ensamblados, módulos y tipos. Se puede utilizar la reflexión para crear dinámicamente una instancia de un tipo, enlazar el tipo a un objeto existente u

obtener el tipo a partir de un objeto existente. A continuación, se puede invocar a los métodos del tipo u obtener acceso a los campos y propiedades correspondientes. Normalmente, se usa la reflexión en los siguientes casos:

- Se utiliza Assembly para definir y cargar ensamblados, cargar módulos que se enumeran en el manifiesto del ensamblado, buscar un tipo de este ensamblado y crear una instancia del mismo.
- Se utiliza Module para descubrir información como el ensamblado que contiene el módulo y las clases en el módulo. También se pueden obtener todos los métodos globales u otros métodos específicos no globales que se definen en el módulo.
- Se utiliza ConstructorInfo para descubrir información como el nombre, los parámetros, los modificadores de acceso (como public o private) y los detalles de implementación (como abstract o virtual) de un constructor. Se utiliza el método GetConstructors o GetConstructor de un Type para invocar a un constructor específico.
- Se utiliza MethodInfo para descubrir información como el nombre, los parámetros, los modificadores de acceso (como public o private) y los detalles de implementación (como abstract o virtual) de un método. Se utiliza el método GetMethods o GetMethod de un Type para invocar a un método específico.
- Se utiliza FieldInfo para descubrir información como el nombre, los modificadores de acceso (como public o private) y los detalles de

implementación (como static) de un campo y para obtener o establecer valores de campo.

- Se utiliza EventInfo para descubrir información como el nombre, el tipo de datos del controlador de eventos, los atributos personalizados, el tipo que declara y el tipo reflejado de un evento, así como para agregar o quitar controladores de eventos.
- Se utiliza PropertyInfo para descubrir información como el nombre, el tipo de datos, el tipo que declara, el tipo reflejado y el estado de sólo lectura o de escritura de una propiedad, así como para obtener o establecer valores de propiedad.
- Se utiliza ParameterInfo para descubrir información como el nombre de un parámetro, el tipo de datos, si un parámetro es un parámetro de entrada o salida, así como la posición del parámetro en un prototipo de método.
- Se utiliza CustomAttributeData para descubrir información sobre atributos personalizados cuando se trabaja en el contexto de solo reflexión del dominio de una aplicación. CustomAttributeData permite examinar atributos sin crear instancias de ellos.

Las clases del espacio de nombres System.Reflection.Emit proporcionan una forma especializada de reflexión que permite generar tipos en tiempo de ejecución.

La reflexión puede utilizarse asimismo para crear aplicaciones denominadas exploradores de tipos, que permiten a los usuarios seleccionar tipos y, a

continuación, ver la información correspondiente.

La reflexión tiene también otras aplicaciones. Los compiladores de lenguajes como JScript utilizan la reflexión para generar tablas de símbolos. Las clases del espacio de nombres System.Runtime.Serialization utilizan la reflexión para obtener acceso a los datos y determinar los campos que se van a mantener. Las clases del espacio de nombres System.Runtime.Remoting utilizan la reflexión indirectamente a través de la serialización.

Apéndice Q

Atributos

[SCH2002] C# permite agregar información declarativa a un programa en forma de un *atributo.* Un atributo define la información adicional que se asocia con una clase, estructura, método, etc. Por ejemplo, puede definir un atributo que determine el tipo de botón que una clase mostrará. Los atributos se especifican entre corchetes, anteponiéndose al elemento al elemento al que se aplican. Por tanto, un atributo no es un miembro de una clase. Antes bien, un atributo especifica información adicional que se adjunta a un elemento.

Un atributo es compatible con una clase que herede System.Attribute. Así, todas las clases de atributos deben ser subclases de Attribute. Aunque Attribute define una serie notable de funciones no siempre son necesarias a la hora de trabajar con atributos. Por convención, las clases de atributos utilizan el sufijo Attribute. Por ejemplo, ErrorAttribute sería un nombre para una clase de atributo que describe un error.

[MSDN8]El hecho de poder definir atributos personalizados y colocarlos en el código fuente tendría un valor escaso si no se dispone de un método para recuperar la información y actuar sobre ella. C# tiene un sistema de reflexión que permite recuperar la información definida con atributos personalizados. El método clave es GetCustomAttributes, el cual devuelve una matriz de objetos que son los equivalentes en tiempo de ejecución de los atributos del código fuente. Este método tiene varias versiones sobrecargadas. Para obtener más información, vea Attribute.

Una especificación de atributos tal como:

```
C#
[Author("H. Ackerman", version = 1.1)]
class SampleClass
```

equivale conceptualmente a:

```
C#
Author anonymousAuthorObject = new Author("H. Ackerman");
anonymousAuthorObject.version = 1.1;
```

Sin embargo, el código sólo se ejecuta cuando se consultan los atributos de SampleClass. Una llamada a GetCustomAttributes en SampleClass hace que se construya un objeto Author y se inicialice como se indicó anteriormente. Si la clase cuenta con otros atributos, se crean de forma similar otros objetos de atributo. A continuación, GetCustomAttributes devuelve el objeto Author y cualquier otro objeto de atributo en una matriz. Es posible recorrer en iteración esta matriz y determinar qué atributos se aplicaron según el tipo de cada elemento de la matriz, así como extraer información de los objetos atributo.

Ejemplo

A continuación se muestra un ejemplo completo. Se define un atributo personalizado, que se aplica a varias entidades, y se recupera por medio del mecanismo de reflexión.

```
C#
[System.AttributeUsage(System.AttributeTargets.Class |
                       System.AttributeTargets.Struct,
                       AllowMultiple = true)  // multiuse
                                              //attribute
]
public class Author : System.Attribute
{
    string name;
    public double version;

    public Author(string name)
    {
        this.name = name;
        version = 1.0;  // Default value
    }

    public string GetName()
    {
        return name;
    }
}

[Author("H. Ackerman")]
private class FirstClass
{
    // ...
}

// No Author attribute
private class SecondClass
{
    // ...
}

[Author("H. Ackerman"), Author("M. Knott", version = 2.0)]
private class ThirdClass
{
```

```
    // ...
}

class TestAuthorAttribute
{
    static void Main()
    {
        PrintAuthorInfo(typeof(FirstClass));
        PrintAuthorInfo(typeof(SecondClass));
        PrintAuthorInfo(typeof(ThirdClass));
    }

    private static void PrintAuthorInfo(System.Type t)
    {
        System.Console.WriteLine("Author information for {0}",
t);
        System.Attribute[] attrs =
System.Attribute.GetCustomAttributes(t);  // reflection

        foreach (System.Attribute attr in attrs)
        {
            if (attr is Author)
            {
                Author a = (Author)attr;
                System.Console.WriteLine("   {0}, version
{1:f}", a.GetName(), a.version);
            }
        }
    }
}
```

Resultado

```
Author information for FirstClass
H. Ackerman, version 1.00
Author information for SecondClass
Author information for ThirdClass
H. Ackerman, version 1.00
M. Knott, version 2.00
```

APÉNDICE R

Documentación de la herramienta

Namespace List

The namespaces specified in this document are:

Namespace	Assembly
NHBuilder.Relations	NHBuilder
NHBuilder.RelationalModel.TableConstructor	NHBuilder
NHBuilder.Exceptions	NHBuilder
NHBuilder.Mapping	NHBuilder
NHBuilder.Collections	NHBuilder
NHBuilder.RelationalModel.DialectTranslateObject	NHBuilder
NHBuilder.RelationalModel	NHBuilder
NHBuilder	NHBuilder
NHBuilder.Reflection	NHBuilder
NHBuilder.MappingAttribute	MappingAttribute

Namespace : NHBuilder.RelationalModel

NHBuilder.RelationalModel Type List

Classes

Type	Summary
Column	Representa a las columnas de una tabla en el modelo relacional
ForeignKey	Representa a las claves foraneas de una tabla en el modelo relacional
Table	Clase que contiene las estructura de una entidad en el modelo relacional

NHBuilder.RelationalModel Classes

Table Class

Summary

```
notPublic class Table
```

Clase que contiene las estructura de una entidad en el modelo relacional

Constructor Members

Name	Access	Summary
Table()	public	constructor

Field Members

Name	Access	Summary
columns : *0, Culture=neutral, PublicKeyToken=null]]*	private	lista de las columnas de la tabla
foreignKeys : *0, Culture=neutral, PublicKeyToken=null]]*	private	lista de claves foraneas (FK) de la tabla
hasId : *Boolean*	private	Para verificar si la tabla debe poseer una columna primary key
idColumn : *Column*	private	columna correspondiente a la primary key
name : *String*	private	nombre de la tabla

Property Members

Name	Access	Summary
Columns : *0, Culture=neutral, PublicKeyToken=null]]*	public	lista de las columnas de la tabla
ForeignKeys : *0, Culture=neutral, PublicKeyToken=null]]*	public	lista de claves foraneas (FK) de la tabla
HasId : *Boolean*	public	Para verificar si la tabla debe poseer una columna primary key
IdColumn : *Column*	public	columna correspondiente a la primary key
Name : *String*	public	nombre de la tabla

Method Members

Name	Access	Summary
AddColumns() : *Void*	public	Agrega las columnas a la tabla a partir de las propiedades de la clase persistente
ContainsFK() : *Boolean*	public	Busca dentro del listado de claves foraneas si ya existe alguna clave foranea

ForeignKey Class

Summary

```
otPublic class ForeignKey
```

Representa a las claves foraneas de una tabla en el modelo relacional

Constructor Members

Name	Access	Summary
ForeignKey()	public	Initializes a new instance of the class.

Field Members

Name	Access	Summary
dataType : *Type*	private	Tipo de dato de la columna
idtablereference : *String*	private	nombre del id de la tabla a la cual hace referencia
isColumn : *Boolean*	private	Indica si la clave foranea es un columna y no es necesario crear la columna "fk" Sino que solo necesita definir la restricction de FOREIGN KEY en la tabla
name : *String*	private	nombre de la columna
nametablereference : *String*	private	nombre de la tabla a la cual hace referencia

Property Members

Name	Access	Summary
DataType : *Type*	public	Tipo de dato de la columna
IdTableReference : *String*	public	si la columna es FK se debe indicar el id de la tabla a la cual hace referencia
IsColumn : *Boolean*	public	Indica si la clave foranea es un columna y no es necesario crear la columna "fk" Sino que solo necesita definir la restricction de FOREIGN KEY en la tabla
Name : *String*	public	nombre de la columna
NameTableReference : *String*	public	si la columna es FK se debe indicar la tabla a la cual hace referencia

Method Members

Name	Access	Summary

Column Class

Summary

```
PublicPublic class Column
```

Representa a las columnas de una tabla en el modelo relacional

Constructor Members

Name	Access	Summary
Column()	public	Initializes a new instance of the class.

Field Members

Name	Access	Summary
_default : *String*	private	valor por defecto de la columna
dataType : *Type*	private	Tipo de dato de la columna
lenght : *Int32*	private	longitud de la columna si es una cadena de caracteres
name : *String*	private	nombre de la columna
notNull : *Boolean*	private	permite determinar si puede ser vacio o no
unique : *Boolean*	private	permite saber si es una columna del tipo unique

Property Members

Name	Access	Summary
DataType : *Type*	public	Tipo de dato de la columna
Default : *String*	public	permite saber si la columna es de tipo FK
Lenght : *Int32*	public	longitud de la columna si es una cadena de caracteres
Name : *String*	public	nombre de la columna
NotNull : *Boolean*	public	permite determinar si puede ser vacio o no
Unique : *Boolean*	public	permite saber si es una columna del tipo unique

Method Members

Name	Access	Summary

Namespace : NHBuilder.MappingAttribute

NHBuilder.MappingAttribute Type List

Classes

Type	Summary
ClassAttribute	Atributos que pueden ser utilizados para las clases
ComponentAttribute	Sirve para indicar una propiedad compuesta
DefaultAttribute	Sirve para indicar algun valor por defecto de una propiedad
IdAttribute	Sirve para indicar si una propiedad es id
LengthAttribute	Sirve para indicar la longitud de una propiedad del tipo string
ManyToManyAttribute	Sirve para indicar una relacion del tipo N..M
ManyToOneAttribute	Sirve para indicar una relacion del tipo N..1 (Referencia simple)
NonPersistentAttribute	Sirve para indicar si una propiedad no debe ser persistida
NotNullAttribute	Sirve para indicar si una propiedad puede ser no vacia
OneToManyAttribute	Sirve para indicar una relacion del tipo 1..N (Colección básica)
OneToOneAttribute	Sirve para indicar una relacion del tipo 1..1 (Asociación simple)
PersistentAttribute	Sirve para indicar si una clase o estuctura es persistente
PropertyAttribute	Atributos para las propiedades con tipos de datos base
RelationAttribute	Atributos para las propiedades que sirven para indicar una relacion
UniqueAttribute	Sirve para indicar si una propiedad es unique

NHBuilder.MappingAttribute Classes

RelationAttribute Class

Summary

```
ublic abstract class RelationAttribute : Attribute
```

ributos para las propiedades que sirven para indicar una relacion

Constructor Members

Name	Access	Summary
RelationAttribute()	protected	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

OneToOneAttribute Class

Summary

```
ublic class OneToOneAttribute : RelationAttribute
```

Sirve para indicar una relacion del tipo 1..1 (Asociación simple)

Constructor Members

Name	Access	Summary
OneToOneAttribute()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

PropertyAttribute Class

Summary

```
ublic abstract class PropertyAttribute : Attribute
```

ributos para las propiedades con tipos de datos base

Constructor Members

Name	Access	Summary
PropertyAttribute()	protected	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

NonPersistentAttribute Class

Summary

```
ublic class NonPersistentAttribute : PropertyAttribute
```

Sirve para indicar si una propiedad no debe ser persistida

Constructor Members

Name	Access	Summary
NonPersistentAttribute()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

ManyToManyAttribute Class

Summary

```
ublic class ManyToManyAttribute : RelationAttribute
```

ive para indicar una relacion del tipo N..M

Constructor Members

Name	Access	Summary
ManyToManyAttribute()	public	constructor por defecto

Field Members

Name	Access	Summary
elatedClass : *Type*	public	clase relacionada

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

ManyToOneAttribute Class

Summary

```
ublic class ManyToOneAttribute : RelationAttribute
```

ive para indicar una relacion del tipo N..1 (Referencia simple)

Constructor Members

Name	Access	Summary
ManyToOneAttribute()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

IdAttribute Class

Summary

```
ublic class IdAttribute : PropertyAttribute
```

Sirve para indicar si una propiedad es id

Constructor Members

Name	Access	Summary
IdAttribute()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

ClassAttribute Class

Summary

```
ublic abstract class ClassAttribute : Attribute
```

tributos que pueden ser utilizados para las clases

Constructor Members

Name	Access	Summary
ClassAttribute()	protected	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

PersistentAttribute Class

Summary

```
public class PersistentAttribute : ClassAttribute
```

Sirve para indicar si una clase o estuctura es persistente

Constructor Members

Name	Access	Summary
PersistentAttribute()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

LengthAttribute Class

Summary

```
public class LengthAttribute : PropertyAttribute
```

Sirve para indicar la longitud de una propiedad del tipo string

Constructor Members

Name	Access	Summary

LengthAttribute()	public	constructor

Field Members

Name	Access	Summary
length : *Int32*	private	longitud de la cadena

Property Members

Name	Access	Summary
Length : *Int32*	public	longitud de la cadena

Method Members

Name	Access	Summary

OneToManyAttribute Class

Summary

```
public class OneToManyAttribute : RelationAttribute
```

Sirve para indicar una relacion del tipo 1..N (Colección básica)

Constructor Members

Name	Access	Summary
OneToManyAttribute()	public	constuctor por defecto

Field Members

Name	Access	Summary
relatedClass : *Type*	public	clase relacionada

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

DefaultAttribute Class

Summary

```
ublic class DefaultAttribute : PropertyAttribute
```

Sirve para indicar algun valor por defecto de una propiedad

Constructor Members

Name	Access	Summary
DefaultAttribute()	public	constructor

Field Members

Name	Access	Summary
defaultvalue *: String*	private	valor por defecto

Property Members

Name	Access	Summary
DefaultValue *: String*	public	valor por defecto

Method Members

Name	Access	Summary

ComponentAttribute Class

Summary

```
ublic class ComponentAttribute : ClassAttribute
```

Sirve para indicar una propiedad compuesta

Constructor Members

Name	Access	Summary
ComponentAttribute()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

NotNullAttribute Class

Summary

```
public class NotNullAttribute : PropertyAttribute
```

Sirve para indicar si una propiedad puede ser no vacia

Constructor Members

Name	Access	Summary
NotNullAttribute()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

UniqueAttribute Class

Summary

```
public class UniqueAttribute : PropertyAttribute
```

Sirve para indicar si una propiedad es unique

Constructor Members

Name	Access	Summary
UniqueAttribute()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary

Namespace : NHBuilder.RelationalModel.TableConstructor

NHBuilder.RelationalModel.TableConstructor Type List

Classes

Type	Summary
TableConstructor	Transforma la estructura de clases a un modelo relacional dependiendo de la estrategia de mapeo de la herencia
TableConstructorFactory	Controla las estrategia para el mapeo de la herencia en la transformacion al modelo relacional
TablePerClassHierarchy	Transforma la estructura de clases a un modelo relacional usando la estrategia "una tabla por clase jerárquica"
TablePerConcreteClass	Transforma la estructura de clases a un modelo relacional usando la estrategia "una tabla por cada clase concreta"
TablePerSubclass	Transforma la estructura de clases a un modelo relacional usando la estrategia "una tabla por subclase"

TransformModel	Lleva a cabo el proceso de conversión de la estructura de clases a un modelo relacional

NHBuilder.RelationalModel.TableConstructor Classes

TableConstructor Class

Summary

```
otPublic abstract class TableConstructor
```

Transforma la estructura de clases a un modelo relacional dependiendo de la estrategia de mapeo de herencia

Constructor Members

Name	Access	Summary
TableConstructor()	public	Initializes a new instance of the class.

Method Members

Name	Access	Summary
ContainsTable() *: Boolean*	public	
GetFK() *: ForeignKey*	public	Dada una clase (A) que se relaciona con otra clase (B) La tabla A debe poseer la clave foranea que haga referencia a la clase B _class viene siendo la clase B
GetTables() *: 0, Culture=neutral, PublicKeyToken=null]]*	public	

TablePerSubclass Class

Summary

```
otPublic class TablePerSubclass : TableConstructor
```

Transforma la estructura de clases a un modelo relacional usando la estrategia "una tabla por subclase

Constructor Members

Name	Access	Summary

TablePerSubclass()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance : *TablePerSubclass*	private	
tables : *0, Culture=neutral, PublicKeyToken=null]]*	private	Listado de clases persistentes

Property Members

Name	Access	Summary
Instance : *TablePerSubclass*	public	Retorna la instancia de la clase

Method Members

Name	Access	Summary
ContainsTable() : *Boolean*	public	*(from NHBuilder.RelationalModel.TableConstructor.TableConstructor)*
GetFK() : *ForeignKey*	public	*(from NHBuilder.RelationalModel.TableConstructor.TableConstructor)* Dada una clase (A) que se relaciona con otra clase (B) La tabla A debe poseer la clave foranea que haga referencia a la clase B _class viene siendo la clase B
GetRelationManyToMany() : *Void*	private	Busca las relaciones [ManyToMany] de persistentClass Si no existe una relacion [ManyToMany] donde relatedClass sea persistentClass crea una tabla interseccion, sino no
GetRelations() : *Void*	private	
GetRelationsManyToOne() : *Void*	private	Busca todas las relaciones [ManyToOne]
GetRelationsOneToMany() : *Void*	private	
GetTable() : *Table*	private	
GetTables() : *0, Culture=neutral, PublicKeyToken=null]]*	public	

TransformModel Class

Summary

```
ublic class TransformModel
```

leva a cabo el proceso de conversión de la estructura de clases a un modelo relacional

Constructor Members

Name	Access	Summary
TransformModel()	public	Initializes a new instance of the class.

Field Members

Name	Access	Summary
dialect : *DTODialect*	private	dielecto sql
tableConstructor : *TableConstructor*	private	Manejo de la herencia

Method Members

Name	Access	Summary
TransformToRelationalModel() : *Void*	public	

TablePerClassHierarchy Class

Summary

```
otPublic class TablePerClassHierarchy : TableConstructor
```

Transforma la estructura de clases a un modelo relacional usando la estrategia "una tabla por clase erárquica"

Constructor Members

Name	Access	Summary
TablePerClassHierarchy()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance : *TablePerClassHierarchy*	private	

tables : *0, Culture=neutral, PublicKeyToken=null]]*	private	Listado de clases persistentes

Property Members

Name	Access	Summary
Instance : *TablePerClassHierarchy*	public	Retorna la instancia de la clase

Method Members

Name	Access	Summary
ContainsTable() : *Boolean*	public	*(from NHBuilder.RelationalModel.TableConstructor.TableConstructor)*
GetFK() : *ForeignKey*	public	*(from NHBuilder.RelationalModel.TableConstructor.TableConstructor)* Dada una clase (A) que se relaciona con otra clase (B) La tabla A debe poseer la clave foranea que haga referencia a la clase B _class viene siendo la clase B
GetRelationManyToMany() : *Void*	private	Busca las relaciones [ManyToMany] de persistentClass Si no existe una relacion [ManyToMany] donde relatedClass sea persistentClass crea una tabla interseccion, sino no
GetRelations() : *Void*	private	
GetRelationsManyToOne() : *Void*	private	Busca todas las relaciones [ManyToOne]
GetRelationsOneToMany() : *Void*	private	
GetTable() : *Table*	private	
GetTables() : *0, Culture=neutral, PublicKeyToken=null]]*	public	

TablePerConcreteClass Class

Summary

```
notPublic class TablePerConcreteClass : TableConstructor
```

Transforma la estructura de clases a un modelo relacional usando la estrategia "una tabla por cada clase concreta"

Constructor Members

Name	Access	Summary
TablePerConcreteClass()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance : *TablePerConcreteClass*	private	
tables : *0, Culture=neutral, PublicKeyToken=null]]*	private	Listado de clases persistentes

Property Members

Name	Access	Summary
Instance : *TablePerConcreteClass*	public	Retorna la instancia de la clase

Method Members

Name	Access	Summary
ContainsTable() : *Boolean*	public	*(from NHBuilder.RelationalModel.TableConstructor.TableConstructor)*
GetFK() : *ForeignKey*	public	*(from NHBuilder.RelationalModel.TableConstructor.TableConstructor)* Dada una clase (A) que se relaciona con otra clase (B) La tabla A debe poseer la clave foranea que haga referencia a la clase B _class viene siendo la clase B
GetFkToSubclases() : *Boolean*	private	
GetRelationManyToMany() : *Void*	private	
GetRelations() : *Void*	private	
GetRelationsManyToOne() : *Void*	private	
GetRelationsOneToMany() : *Void*	private	
GetTable() : *Table*	private	
GetTables() : *0, Culture=neutral, PublicKeyToken=null]]*	public	

ValidateRelation() : *Void*	private

TableConstructorFactory Class

Summary

```
notPublic abstract class TableConstructorFactory
```

Controla las estrategia para el mapeo de la herencia en la transformacion al modelo relacional

Constructor Members

Name	Access	Summary
TableConstructorFactory()	protected	Initializes a new instance of the class.

Method Members

Name	Access	Summary
GetTableConstructor() : *TableConstructor*	public	Retorna un objeto del tipo Inheritance para el mapeo dela herencia

Namespace : NHBuilder

NHBuilder Type List

Enumerations

Type	Summary
Persistent.CollectionTypes	Tipo de colecciones soportadas
Persistent.InheritanceStrategy	Estrategias para el manejo de la herencia
Persistent.RelationTypes	Tipo de relaciones entre clases

Classes

Type	Summary

NHibernateMapping	Contiene el metodo de mapear los tipos de datos de .Net a los tipos de datos manejados por NHibernate 1.0.2
Persistent	Contiene los elementos para el proceso de mapeo
PersistentClass	Representa las clases persistentes del usuario. Sus campos almacenan la información necesaria para la generación de los archivos de mapeo y la estructura de la base de datos. Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo, ya sea como clase, subclase o componente.
Property	Representa las propiedades de las clases persistentes del usuario. Sus campos almacenan la información necesaria para la generación de los archivos de mapeo y la estructura de la base de datos. Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo, ya sea como id de la clase o como una simple propiedad.

NHBuilder Enumerations

Persistent.RelationTypes Enumeration

Summary

nestedPublic enumeration Persistent.RelationTypes

Tipo de relaciones entre clases

Enumeration Members

Field	Summary
Component	Relacion objeto compuesto o contenido
ManyToMany	Relacion muchos a muchos
ManyToOne	Relacion muchos a uno
OneToMany	Relacion uno a muchos
OneToOne	Relacion uno a uno

Persistent.CollectionTypes Enumeration

Summary

nestedPublic enumeration Persistent.CollectionTypes

Tipo de colecciones soportadas

Enumeration Members

Field	Summary
BagCollection	Cualquier tipo de coleccion
GenericBagCollection	Cualquier tipo de coleccion genérica
GenericListCollection	System.Collection.Generic.List
GenericMapCollection	System.Collection.Generic.IDictionary
GenericSetCollection	Iesi.Collection.ISet
ListCollection	System.Collection.IList
MapCollection	System.Collection.IDictionary
SetCollection	Iesi.Collection.ISet

Persistent.InheritanceStrategy Enumeration

Summary

```
estedPublic enumeration Persistent.InheritanceStrategy
```

Estrategias para el manejo de la herencia

Enumeration Members

Field	Summary
tablePerClassHierarchy	Estrategia de mapeo "una tabla por clase jerarquica"
tablePerConcreteClass	Estrategia de mapeo "una tabla por cada clase concreta"
tablePerSubclass	Estrategia de mapeo "una tabla por subclase"

NHBuilder Classes

NHibernateMapping Class

Summary

```
otPublic abstract class NHibernateMapping
```

Contiene el metodo de mapear los tipos de datos de .Net a los tipos de datos manejados por NHibernate 1.0.2

Constructor Members

Name	Access	Summary

NHibernateMapping()	protected	Initializes a new instance of the class.

Method Members

Name	Access	Summary
GetNHibernateType() : *String*	public	Dado un tipo de dato de .Net Framework Retorna su correspondiente para NHibernate 1.0.2

PersistentClass Class

Summary

```
public class PersistentClass
```

Representa las clases persistentes del usuario. Sus campos almacenan la información necesaria para la generación de los archivos de mapeo y la estructura de la base de datos. Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo, ya sea como clase, subclase o componente.

Constructor Members

Name	Access	Summary
PersistentClass()	public	constructor

Field Members

Name	Access	Summary
_abstract : *Boolean*	private	Inheritance information si es una clase abstracta
_assembly : *String*	private	nombre del ensamblado
_namespace : *String*	private	namespace al que pertenece la clase
_component : *Boolean*	private	Indica si la clase es un componente y no una clase persistente como tal
_idProperty : *Property*	private	Propiedad id
_name : *String*	private	nombre de la clase
_propertys : *0, Culture=neutral, PublicKeyToken=null]]*	private	propedades de la clase
_relations : *0, Culture=neutral, PublicKeyToken=null]]*	private	relaciones de la clase con otras clases

superClass : *PersistentClass*	private	Inheritance information superclase
superClassName : *String*	private	Inheritance information nombre de la superclase si la tiene

Property Members

Name	Access	Summary
Abstract : *Boolean*	public	Inheritance information si es una clase abstracta
Assembly : *String*	public	nombre del ensamblado
Component : *Boolean*	public	Indica si la clase es un componente y no una clase persistente como tal
IdProperty : *Property*	public	Propiedad id
Name : *String*	public	nombre de la clase
Namespace : *String*	public	namespace al que pertenece la clase
Propertys : *0, Culture=neutral, PublicKeyToken=null]]*	public	propedades de la clase
Relations : *0, Culture=neutral, PublicKeyToken=null]]*	public	relaciones de la clase con otras clases
SuperClass : *PersistentClass*	public	Inheritance information superclase
SuperClassName : *String*	public	Inheritance information nombre de la superclase si la tiene

Method Members

Name	Access	Summary
Equals() : *Boolean*	public	Metodo Equals
IsSuperClass() : *Boolean*	public	
MappingComponent() : *Void*	public	Retorna una cadena de caracteres en formato xml correspondiente al mapeo de una clase como un componente de otra clase para NHibernate 1.0.2
MappingJoinedSubClass() : *Void*	public	Retorna una cadena de caracteres en formato xml correspondiente al mapeo de una clase como una subclase para la estrategia de manejo de la herencia TablePerSubClass para NHibernate 1.0.2
MappingPropertys() : *Void*	public	Retorna una cadena de caracteres en formato xml correspondiente al mapeo de todas las propiedades de la clase para NHibernate 1.0.2

MappingRelations() : *Void*	public	Retorna una cadena de caracteres en formato xml correspondiente al mapeo de todas las relaciones de la clase para NHibernate 1.0.2
MappingSubClass() : *Void*	public	Retorna una cadena de caracteres en formato xml correspondiente al mapeo de una clase como una subclase para la estrategia de manejo de la herencia TablePerClassHierarchy para NHibernate 1.0.2
ToString() : *String*	public	Metodo ToString

Property Class

Summary

```
public class Property
```

Representa las propiedades de las clases persistentes del usuario. Sus campos almacenan la información necesaria para la generación de los archivos de mapeo y la estructura de la base de datos Tiene la capacidad de generar su propio elemento XML dentro de los archivos de mapeo, ya sea como id de la clase o como una simple propiedad.

Constructor Members

Name	Access	Summary
Property()	public	Initializes a new instance of the class.

Field Members

Name	Access	Summary
_default : *String*	private	valor por defecto de la propiedad
dataType : *Type*	private	Tipo de dato de la propiedad
id : *Boolean*	private	permite saber si la propiedad es el identificador
lenght : *Int32*	private	longitud de la columna si es una cadena de caracteres
name : *String*	private	nombre de la propiedad
nonPersistent : *Boolean*	private	si la propiedad no se debe persistir
notNull : *Boolean*	private	permite determinar si puede ser vacio o no
ownerclassname : *String*	private	nombre de la clase
unique : *Boolean*	private	permite saber si es una columna del tipo unique

Property Members

Name	Access	Summary
DataType : *Type*	public	Tipo de dato de la propiedad
Default : *String*	public	valor por defecto de la propiedad
Id : *Boolean*	public	permite saber si la propiedad es el identificador
Lenght : *Int32*	public	longitud de la columna si es una cadena de caracteres
Name : *String*	public	nombre de la propiedad
NonPersistent : *Boolean*	public	si la propiedad no se debe persistir
NotNull : *Boolean*	public	permite determinar si puede ser vacio o no
OwnerClassName : *String*	public	nombre de la clase
Unique : *Boolean*	public	permite saber si es una columna del tipo unique

Method Members

Name	Access	Summary
Mapping() : *Void*	public	Retorna una cadena de caracteres en formato xml correspondiente al mapeo de una propiedad para NHibernate 1.0.2

Persistent Class

Summary

```
public abstract class Persistent
```

Contiene los elementos para el proceso de mapeo

Constructor Members

Name	Access	Summary
Persistent()	protected	Initializes a new instance of the class.

Method Members

Name	Access	Summary

Namespace : NHBuilder.RelationalModel.DialectTranslateObject

NHBuilder.RelationalModel.DialectTranslateObject Type Lis

Interfaces

Type	Summary
DTODialect	Interfaz DTO que representa a todos los dialectos SQL

Classes

Type	Summary
DTOFactory	Clase abstracta para las implementaciones DTODialect que devuelve el dialecto del manejador para llevar a cabo la transformación al modelo relacional
MySQLDialect	Realiza el proceso de generacion del DDL para un modelo relacional usando el dialecto para el manejador MySQL
OracleDialect	Realiza el proceso de generacion del DDL para un modelo relacional usando el dialecto para el manejador Oraacle
PostgreSQLDialect	Realiza el proceso de generacion del DDL para un modelo relacional usando el dialecto para el manejador PostgreSQL
SQLServerDialect	Realiza el proceso de generacion del DDL para un modelo relacional usando el dialecto para el manejador MS SQL Server

NHBuilder.RelationalModel.DialectTranslateObject Interfaces

DTODialect Interface

Summary

```
otPublic interface DTODialect
```

nterfaz DTO que representa a todos los dialectos SQL

Method Members

Name	Access	Summary
CreateTable() : *String*	public	Retorna una cadena de caracteres con la sentencia sql para crear una tabla

GetDbType() : *String*	public	Dado un tipo de dato de .Net Framework Retorna su correspondiente tipo de dato del dialecto sql

NHBuilder.RelationalModel.DialectTranslateObject Classes

OracleDialect Class

Summary

```
otPublic class OracleDialect :
HBuilder.RelationalModel.DialectTranslateObject.DTODialect
```

Realiza el proceso de generacion del DDL para un modelo relacional usando el dialecto para el manejador Oraacle

Constructor Members

Name	Access	Summary
OracleDialect()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance : *OracleDialect*	private	

Property Members

Name	Access	Summary
Instance : *OracleDialect*	public	

Method Members

Name	Access	Summary
CreateTable() : *String*	public	Retorna una cadena de caracteres con la sentencia sql para crear una tabla
GetDbType() : *String*	public	Dado un tipo de dato de .Net Framework Retorna su correspondiente para Oracle

PostgreSQLDialect Class

Summary

```
notPublic class PostgreSQLDialect :
NHBuilder.RelationalModel.DialectTranslateObject.DTODialect
```

Realiza el proceso de generacion del DDL para un modelo relacional usando el dialecto para el manejador PostgreSQL

Constructor Members

Name	Access	Summary
PostgreSQLDialect()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance : *PostgreSQLDialect*	private	

Property Members

Name	Access	Summary
Instance : *PostgreSQLDialect*	public	

Method Members

Name	Access	Summary
CreateTable() : *String*	public	Retorna una cadena de caracteres con la sentencia sql para crear una tabla
GetDbType() : *String*	public	Dado un tipo de dato de .Net Framework Retorna su correspondiente para PostgreSQL

MySQLDialect Class

Summary

```
otPublic class MySQLDialect :
HBuilder.RelationalModel.DialectTranslateObject.DTODialect
```

Realiza el proceso de generacion del DDL para un modelo relacional usando el dialecto para el manejador MySQL

Constructor Members

Name	Access	Summary
MySQLDialect()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance : *MySQLDialect*	private	

Property Members

Name	Access	Summary
Instance : *MySQLDialect*	public	

Method Members

Name	Access	Summary
CreateTable() : *String*	public	Retorna una cadena de caracteres con la sentencia sql para crear una tabla
GetDbType() : *String*	public	Dado un tipo de dato de .Net Framework Retorna su correspondiente para MySQL

SQLServerDialect Class

Summary

```
notPublic class SQLServerDialect :
WHBuilder.RelationalModel.DialectTranslateObject.DTODialect
```

Realiza el proceso de generacion del DDL para un modelo relacional usando el dialecto para el manejador MS SQL Server

Constructor Members

Name	Access	Summary
SQLServerDialect()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance : *SQLServerDialect*	private	

Property Members

Name	Access	Summary
Instance : *SQLServerDialect*	public	

Method Members

Name	Access	Summary
CreateTable() : *String*	public	Retorna una cadena de caracteres con la sentencia sql para crear una tabla
FKConstraint() : *String*	private	
GetColumns() : *String*	private	
GetDbType() : *String*	public	Dado un tipo de dato de .Net Framework Retorna su correspondiente para SQL Server 2000 y 2005
GetFKs() : *String*	private	
GetId() : *String*	private	Retorna la sintaxis para definir la columna id de una tabla
PKConstraint() : *String*	private	Retorna la restriccion de clave primaria

DTOFactory Class

Summary

```
notPublic abstract class DTOFactory
```

Clase abstracta para las implementaciones DTODialect que devuelve el dialecto del manejador para llevar a cabo la transformación al modelo relacional

Constructor Members

Name	Access	Summary
DTOFactory()	protected	Initializes a new instance of the class.

Method Members

Name	Access	Summary
GetDialect() : *DTODialect*	public	Retorna un objeto de tipo DTOFactory que contiene la implementación DTODialect para el dialecto del manejador para llevar a cabo la transformación al modelo relacional

Namespace : NHBuilder.Exceptions

NHBuilder.Exceptions Type List

Classes

Type	Summary
ClassNotFoundException	Esta excepcion es lanzada cuando la clase relacionada de una relacion no esta definida en el atributo de la relacion (al usar colecciones no genéricas)
DialectNotFoundException	Esta excepcion es lanzada cuando el dialecto SQL escogido no se encuentra dentro de los dialectos, ya sea porque no es soportado por NHBuilder, o es icorrecto
IdNotFoundException	Esta excepcion es lanzada cuando una clase que debe poseer una propiedad identificador, no tiene definido el atributo [Id]
InvalidCollectionException	Esta excepcion es lanzada cuando un tipo de colección no esta soportada por NHBuilder
InvalidDataTypeException	Este tipo de excepcion es lanzada cuando alguna propiedad de alguna clase persistente tiene como tipo de dato base un tipo de dato no soportado por NHBuilder
InvalidInheritanceStrategyException	Este tipo de excepcion es lanzada cuando la estrategia para el mapeo de la herencia escogida se sale del rango (0,1,2)
InvalidRelatedClassException	Esta excepcion es lanzada cuando la clase relacionada de una relacion no esta definida como clase persistente
InvalidRelationException	Este tipo de excepcion es lanzada cuando se define una relacion que no es soportada por NHBuilder
InvalidTableException	Esta excepcion ocurre cuando se intenta crear una tabla que no posee ninguna columna
LenghtNotFoundException	Esta excepcion ocurrre cuando no se define para una propiedad del tipo String, el atributo Lenght con la longitud
NHBuilderException	Excepción base para todas las excepciones que puede lanzar NHBuilder a lo largo del procesamiento

RelationException	Excepción base para todas las excepciones con respecto a las relaciones entre clases
RelationNonAllowedException	Esta excepcion ocurre si se utiliza la estrategia de mapeo "TablePerConcreteClass" y se maneja polimorfismo con cierto tipo de relaciones

NHBuilder.Exceptions Classes

NHBuilderException Class

Summary

```
notPublic class NHBuilderException : ApplicationException,
System.Runtime.Serialization.ISerializable
```

Excepción base para todas las excepciones que puede lanzar NHBuilder a lo largo del procesamiento

Constructor Members

Name	Access	Summary
NHBuilderException()	public	Initializes a new instance of the class.
NHBuilderException()	public	

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

InvalidDataTypeException Class

Summary

```
notPublic class InvalidDataTypeException : NHBuilderException,
System.Runtime.Serialization.ISerializable
```

Este tipo de excepcion es lanzada cuando alguna propiedad de alguna clase persistente tiene como tipo de dato base un tipo de dato no soportado por NHBuilder

Constructor Members

Name	Access	Summary
InvalidDataTypeException()	public	Initializes a new instance of the class.
InvalidDataTypeException()	public	

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

InvalidTableException Class

Summary

```
notPublic class InvalidTableException : NHBuilderException,
System.Runtime.Serialization.ISerializable
```

Esta excepcion ocurre cuando se intenta crear una tabla que no posee ninguna columna

Constructor Members

Name	Access	Summary
InvalidTableException()	public	Initializes a new instance of the class.

InvalidTableException()	public

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

InvalidInheritanceStrategyException Class

Summary

```
notPublic class InvalidInheritanceStrategyException : NHBuilderException,
System.Runtime.Serialization.ISerializable
```

Este tipo de excepcion es lanzada cuando la estrategia para el mapeo de la herencia escogida se sale del rango (0,1,2)

Constructor Members

Name	Access	Summary
InvalidInheritanceStrategyException()	public	Initializes a new instance of the class.
InvalidInheritanceStrategyException()	public	

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

RelationException Class

Summary

```
notPublic class RelationException : NHBuilderException,
System.Runtime.Serialization.ISerializable
```

Excepción base para todas las excepciones con respecto a las relaciones entre clases

Constructor Members

Name	Access	Summary
RelationException()	public	Initializes a new instance of the class.
RelationException()	public	

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

InvalidRelatedClassException Class

Summary

```
notPublic class InvalidRelatedClassException : RelationException,
System.Runtime.Serialization.ISerializable
```

Esta excepcion es lanzada cuando la clase relacionada de una relacion no esta definida como clase persistente

Constructor Members

Name	Access	Summary
InvalidRelatedClassException()	public	Initializes a new instance of the class.
InvalidRelatedClassException()	public	

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

LenghtNotFoundException Class

Summary

```
notPublic class LenghtNotFoundException : NHBuilderException,
System.Runtime.Serialization.ISerializable
```

Esta excepcion ocurrre cuando no se define para una propiedad del tipo String, el atributo Lenght con longitud

Constructor Members

Name	Access	Summary
LenghtNotFoundException()	public	Initializes a new instance of the class.

LenghtNotFoundException()	public

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

IdNotFoundException Class

Summary

```
notPublic class IdNotFoundException : NHBuilderException,
System.Runtime.Serialization.ISerializable
```

Esta excepcion es lanzada cuando una clase que debe poseer una propiedad identificador, no tiene definido el atributo [Id]

Constructor Members

Name	Access	Summary
IdNotFoundException()	public	Initializes a new instance of the class.
IdNotFoundException()	public	

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

InvalidCollectionException Class

Summary

```
notPublic class InvalidCollectionException : NHBuilderException,
System.Runtime.Serialization.ISerializable
```

Esta excepcion es lanzada cuando un tipo de colección no esta soportada por NHBuilder

Constructor Members

Name	Access	Summary
InvalidCollectionException()	public	Initializes a new instance of the class.
InvalidCollectionException()	public	

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

ClassNotFoundException Class

Summary

```
notPublic class ClassNotFoundException : RelationException,
System.Runtime.Serialization.ISerializable
```

Esta excepcion es lanzada cuando la clase relacionada de una relacion no esta definida en el atributo de la relacion (al usar colecciones no genéricas)

Constructor Members

Name	Access	Summary
ClassNotFoundException()	public	Initializes a new instance of the class.
ClassNotFoundException()	public	

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() *: String*	public	

InvalidRelationException Class

Summary

```
notPublic class InvalidRelationException : RelationException,
System.Runtime.Serialization.ISerializable
```

Este tipo de excepcion es lanzada cuando se define una relacion que no es soportada por NHBuilder

Constructor Members

Name	Access	Summary
InvalidRelationException()	public	Initializes a new instance of the class.

InvalidRelationException()	public

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

RelationNonAllowedException Class

Summary

```
notPublic class RelationNonAllowedException : RelationException,
System.Runtime.Serialization.ISerializable
```

Esta excepcion ocurre si se utiliza la estrategia de mapeo "TablePerConcreteClass" y se maneja polimorfismo con cierto tipo de relaciones

Constructor Members

Name	Access	Summary
RelationNonAllowedException()	public	Initializes a new instance of the class.
RelationNonAllowedException()	public	

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

DialectNotFoundException Class

Summary

```
notPublic class DialectNotFoundException : NHBuilderException,
System.Runtime.Serialization.ISerializable
```

Esta excepcion es lanzada cuando el dialecto SQL escogido no se encuentra dentro de los dialectos,) sea porque no es soportado por NHBuilder, o es icorrecto

Constructor Members

Name	Access	Summary
DialectNotFoundException()	public	Initializes a new instance of the class.
DialectNotFoundException()	public	

Field Members

Name	Access	Summary

Property Members

Name	Access	Summary

Method Members

Name	Access	Summary
ToString() : *String*	public	

Namespace : NHBuilder.Mapping

NHBuilder.Mapping Type List

Classes

Type	Summary
GenerateMapping	Lleva a cabo el proceso de generación de los archivos de mapeo para las clases persistentes
MappingClass	Genera los archivos de mapeo a partir de la estructura de clases dependiendo de la estrategia de mapeo de la herencia.
MappingFactory	Clase abstracta, que controla las estrategia para el mapeo de la herencia
MappingTablePerClassHierarchy	Genera los archivos de mapeo para la estructura de clases usando la estrategia "una tabla por clase jerárquica"
MappingTablePerConcreteClass	Genera los archivos de mapeo para la estructura de clases usando la estrategia "una tabla por cada clase concreta"
MappingTablePerSubclass	Genera los archivos de mapeo para la estructura de clases usando la estrategia "una tabla por subclase"

NHBuilder.Mapping Classes

MappingClass Class

Summary

```
notPublic abstract class MappingClass
```

Genera los archivos de mapeo a partir de la estructura de clases dependiendo de la estrategia de mapeo de la herencia.

Constructor Members

Name	Access	Summary
MappingClass()	protected	Initializes a new instance of the class.

Method Members

Name	Access	Summary
MappingClasses() : *Void*	public	

MappingTablePerSubclass Class

Summary

```
notPublic class MappingTablePerSubclass : MappingClass
```

Genera los archivos de mapeo para la estructura de clases usando la estrategia "una tabla por subclase"

Constructor Members

Name	Access	Summary
MappingTablePerSubclass()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance *: MappingTablePerSubclass*	private	

Property Members

Name	Access	Summary
Instance *: MappingTablePerSubclass*	public	Retorna la instancia de la clase

Method Members

Name	Access	Summary
MappingClasses() *: Void*	public	

MappingFactory Class

Summary

```
notPublic class MappingFactory
```

Clase abstracta, que controla las estrategia para el mapeo de la herencia

Constructor Members

Name	Access	Summary

MappingFactory()	public	Initializes a new instance of the class.

Method Members

Name	Access	Summary
GetMapping() : *MappingClass*	public	Retorna un objeto del tipo Inheritance para el mapeo dela herencia

GenerateMapping Class

Summary

```
public class GenerateMapping
```

Lleva a cabo el proceso de generación de los archivos de mapeo para las clases persistentes

Constructor Members

Name	Access	Summary
GenerateMapping()	public	Initializes a new instance of the class.

Field Members

Name	Access	Summary
mapping : *MappingClass*	private	Para relaizar el mapeo de las clases para NHibernate 1.0.2

Method Members

Name	Access	Summary
GenerateNHibernateMapping() : *Void*	public	

MappingTablePerClassHierarchy Class

Summary

```
notPublic class MappingTablePerClassHierarchy : MappingClass
```

Genera los archivos de mapeo para la estructura de clases usando la estrategia "una tabla por clase jerárquica"

Constructor Members

Name	Access	Summary
MappingTablePerClassHierarchy()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance *: MappingTablePerClassHierarchy*	private	

Property Members

Name	Access	Summary
Instance *: MappingTablePerClassHierarchy*	public	Retorna la instancia de la clase

Method Members

Name	Access	Summary
MappingClasses() *: Void*	public	

MappingTablePerConcreteClass Class

Summary

```
notPublic class MappingTablePerConcreteClass : MappingClass
```

Genera los archivos de mapeo para la estructura de clases usando la estrategia "una tabla por cada clase concreta"

Constructor Members

Name	Access	Summary
MappingTablePerConcreteClass()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary

instance : *MappingTablePerConcreteClass*	private

Property Members

Name	Access	Summary
Instance : *MappingTablePerConcreteClass*	public	Retorna la instancia de la clase

Method Members

Name	Access	Summary
MappingClasses() : *Void*	public	

Namespace : NHBuilder.Relations

NHBuilder.Relations Type List

Classes

Type	Summary
Component	Representa a las relaciones uno a uno entre las clases peristentes
ManyToMany	Representa a las relaciones uno a uno entre las clases peristentes
ManyToOne	Representa a las relaciones uno a uno entre las clases peristentes
OneToMany	Representa a las relaciones uno a uno entre las clases peristentes
OneToOne	Representa a las relaciones uno a uno entre las clases peristentes
Relation	Representa las relaciones entre clases persistentes del usuario.
RelationFactory	Clase abstracta para los distintos tipos de relaciones entre clases

NHBuilder.Relations Classes

Relation Class

Summary

```
public abstract class Relation
```

Representa las relaciones entre clases persistentes del usuario.

Constructor Members

Name	Access	Summary
Relation()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
className *: String*	private	nombre de la clase a la que pertenece la relacion
classPropertyName *: String*	private	nombre de la propiedad de la clase
collectionType *: Collection*	private	Tipo de coleccion para relaciones OneToMany y ManyToMany
ownerClass *: PersistentClass*	private	Clase a la que pertenece la relacion
relatedClass *: PersistentClass*	private	Clase con la que esta relacionada
relatedClassName *: String*	private	nombre de la clase con la que se relaciona
relationType *: Int32*	private	Tipo de relacion OneToOne, ManyToOne, OneToMany, ManyToMany
tablename *: String*	private	nombre de la tabla de interseccion para la relacion del tipo ManyToMany

Property Members

Name	Access	Summary
ClassName *: String*	public	nombre de la clase a la que pertenece la relacion
ClassPropertyName *: String*	public	nombre de la propiedad de la clase
CollectionType *: Collection*	public	Tipo de coleccion para relaciones OneToMany y ManyToMany
OwnerClass *: PersistentClass*	public	Clase a la que pertenece la relacion
RelatedClass *: PersistentClass*	public	Clase con la que esta relacionada
RelatedClassName *: String*	public	nombre de la clase con la que se relaciona
RelationType *: Int32*	public	Tipo de relacion OneToOne, ManyToOne, OneToMany, ManyToMany
TableNameManyToMany *: String*	public	nombre de la tabla de interseccion para la relacion del tipo ManyToMany

Method Members

Name	Access	Summary
Mapping() : *Void*	public	Realiza el proceso de mapeo para la relacion, retorna una cadena de caracteres en formato xml, segun las especificaciones de NHibernate

OneToOne Class

Summary

```
notPublic class OneToOne : Relation
```

Representa a las relaciones uno a uno entre las clases peristentes

Constructor Members

Name	Access	Summary
OneToOne()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
ClassName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la clase a la que pertenece la relacion
ClassPropertyName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la propiedad de la clase
CollectionType : *Collection*	public	*(from NHBuilder.Relations.Relation)* Tipo de coleccion para relaciones OneToMany y ManyToMany
OwnerClass : *PersistentClass*	public	*(from NHBuilder.Relations.Relation)* Clase a la que pertenece la relacion
RelatedClass : *PersistentClass*	public	*(from NHBuilder.Relations.Relation)* Clase con la que esta relacionada
RelatedClassName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la clase con la que se relaciona
RelationType : *Int32*	public	*(from NHBuilder.Relations.Relation)* Tipo de relacion OneToOne ManyToOne, OneToMany, ManyToMany
TableNameManyToMany : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la tabla de interseccion para la relacion del tipo ManyToMany

Method Members

Name	Access	Summary
Mapping() : *Void*	public	Realiza el proceso de mapeo para la relacion, retorna una cadena de caracteres en formato xml, segun las especificaciones de NHibernate

OneToMany Class

Summary

```
notPublic class OneToMany : Relation
```

Representa a las relaciones uno a uno entre las clases peristentes

Constructor Members

Name	Access	Summary
OneToMany()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary
ClassName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la clase a la que pertenece la relacion
ClassPropertyName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la propiedad de la clase
CollectionType : *Collection*	public	*(from NHBuilder.Relations.Relation)* Tipo de coleccion para relaciones OneToMany y ManyToMany
OwnerClass : *PersistentClass*	public	*(from NHBuilder.Relations.Relation)* Clase a la que pertenece la relacion
RelatedClass : *PersistentClass*	public	*(from NHBuilder.Relations.Relation)* Clase con la que esta relacionada
RelatedClassName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la clase con la que se relaciona
RelationType : *Int32*	public	*(from NHBuilder.Relations.Relation)* Tipo de relacion OneToOne ManyToOne, OneToMany, ManyToMany
TableNameManyToMany : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la tabla de interseccion para la relacion del tipo ManyToMany

Method Members

Name	Access	Summary
Mapping() : *Void*	public	Realiza el proceso de mapeo para la relacion, retorna una cadena de caracteres en formato xml, segun las especificaciones de NHibernate

RelationFactory Class

Summary

```
notPublic abstract class RelationFactory
```

Clase abstracta para los distintos tipos de relaciones entre clases

Constructor Members

Name	Access	Summary
RelationFactory()	protected	Initializes a new instance of the class.

Method Members

Name	Access	Summary
GetRelation() : *Relation*	public	Retorna un objeto del tipo Relation para el mapeo

ManyToMany Class

Summary

```
notPublic class ManyToMany : Relation
```

Representa a las relaciones uno a uno entre las clases peristentes

Constructor Members

Name	Access	Summary
ManyToMany()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

ClassName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la clase a la que pertenece la relacion
ClassPropertyName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la propiedad de la clase
CollectionType : *Collection*	public	*(from NHBuilder.Relations.Relation)* Tipo de coleccion para relaciones OneToMany y ManyToMany
OwnerClass : *PersistentClass*	public	*(from NHBuilder.Relations.Relation)* Clase a la que pertenece la relacion
RelatedClass : *PersistentClass*	public	*(from NHBuilder.Relations.Relation)* Clase con la que esta relacionada
RelatedClassName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la clase con la que se relaciona
RelationType : *Int32*	public	*(from NHBuilder.Relations.Relation)* Tipo de relacion OneToOne ManyToOne, OneToMany, ManyToMany
TableNameManyToMany : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la tabla de interseccion para la relacion del tipo ManyToMany

Method Members

Name	Access	Summary
Mapping() : *Void*	public	Realiza el proceso de mapeo para la relacion, retorna una cadena de caracteres en formato xml, segun las especificaciones de NHibernate

ManyToOne Class

Summary

```
notPublic class ManyToOne : Relation
```

Representa a las relaciones uno a uno entre las clases peristentes

Constructor Members

Name	Access	Summary
ManyToOne()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

ClassName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la clase a la que pertenece la relacion
ClassPropertyName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la propiedad de la clase
CollectionType : *Collection*	public	*(from NHBuilder.Relations.Relation)* Tipo de coleccion para relaciones OneToMany y ManyToMany
OwnerClass : *PersistentClass*	public	*(from NHBuilder.Relations.Relation)* Clase a la que pertenece la relacion
RelatedClass : *PersistentClass*	public	*(from NHBuilder.Relations.Relation)* Clase con la que esta relacionada
RelatedClassName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la clase con la que se relaciona
RelationType : *Int32*	public	*(from NHBuilder.Relations.Relation)* Tipo de relacion OneToOne ManyToOne, OneToMany, ManyToMany
TableNameManyToMany : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la tabla de interseccion para la relacion del tipo ManyToMany

Method Members

Name	Access	Summary
Mapping() : *Void*	public	Realiza el proceso de mapeo para la relacion, retorna una cadena de caracteres en formato xml, segun las especificaciones de NHibernate

Component Class

Summary

```
notPublic class Component : Relation
```

Representa a las relaciones uno a uno entre las clases peristentes

Constructor Members

Name	Access	Summary
Component()	public	Initializes a new instance of the class.

Property Members

Name	Access	Summary

ClassName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la clase a la que pertenece la relacion
ClassPropertyName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la propiedad de la clase
CollectionType : *Collection*	public	*(from NHBuilder.Relations.Relation)* Tipo de coleccion para relaciones OneToMany y ManyToMany
OwnerClass : *PersistentClass*	public	*(from NHBuilder.Relations.Relation)* Clase a la que pertenece la relacion
RelatedClass : *PersistentClass*	public	*(from NHBuilder.Relations.Relation)* Clase con la que esta relacionada
RelatedClassName : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la clase con la que se relaciona
RelationType : *Int32*	public	*(from NHBuilder.Relations.Relation)* Tipo de relacion OneToOne ManyToOne, OneToMany, ManyToMany
TableNameManyToMany : *String*	public	*(from NHBuilder.Relations.Relation)* nombre de la tabla de interseccion para la relacion del tipo ManyToMany

Method Members

Name	Access	Summary
Mapping() : *Void*	public	Realiza el proceso de mapeo para la relacion

Namespace : NHBuilder.Reflection

NHBuilder.Reflection Type List

Classes

Type	Summary
ReflectionProcess	Lleva a cabo el proceso de reflexion para obtener la estructura de clases persistentes con sus propiedades y relaciones

NHBuilder.Reflection Classes

ReflectionProcess Class

Summary

```
public class ReflectionProcess
```

Lleva a cabo el proceso de reflexion para obtener la estructura de clases persistentes con sus propiedades y relaciones

Constructor Members

Name	Access	Summary
ReflectionProcess()	public	Initializes a new instance of the class.

Method Members

Name	Access	Summary
GetClass() *: PersistentClass*	public	
GetClasses() *: 0, Culture=neutral, PublicKeyToken=null]]*	private	Obtiene las clases persistentes del ensamblado a traves de reflexion
GetCollection() *: Relation*	private	Obtiene el tipo de colección de la relación
GetPropertyAttributes() *: Property*	private	Se encargga de detectar cuales son los atributos de un campo de la clase
GetPropertysAndRelations() *: PersistentClass*	private	Obtiene todos los campos y relaciones de una clase
GetRelationAttributes() *: Relation*	private	Se encargga de detectar cuales son los atributos de una relaciion de la clase
GetStructureOfClasses() *: 0, Culture=neutral, PublicKeyToken=null]]*	public	Obtiene la estructura de clases persistentes del ensamblado

Namespace : NHBuilder.Collections

NHBuilder.Collections Type List

Classes

Type	Summary
Bag	Representa cualquier colección de .Net a la terminología de Nhibernate
Collection	Representa a las colecciones de objetos para las relaciones de tipo uno a muchos y muchos a muchos.

CollectionFactory	Clase abstracta, maneja todos los tipos de colecciones soportadas
List	Representa a la colecciones System.Collections.IList y System.Collection.Generic.List de .Net.
Map	Representa a la colecciones System.Collections.IDictionary y System.Collection.Generic.Dictionary de .Net.
Set	Representa a la colecciones Iesi.Collections.ISet (coleccion definida por Nhibernate.org)

NHBuilder.Collections Classes

Collection Class

Summary

```
public abstract class Collection
```

Representa a las colecciones de objetos para las relaciones de tipo uno a muchos y muchos a mucho:

Constructor Members

Name	Access	Summary
Collection()	protected	Initializes a new instance of the class.

Method Members

Name	Access	Summary
MappingManyToMany() : *Void*	public	Realiza el proceso de mapeo para las colecciones cuyo tipo de relacion sea ManyToMany
MappingOneToMany() : *Void*	public	Realiza el proceso de mapeo para las colecciones cuyo tipo de relacion sea OneToMany

Map Class

Summary

```
notPublic class Map : Collection
```

Representa a la colecciones System.Collections.IDictionary y System.Collection.Generic.Dictionary de .Net.

Constructor Members

Name	Access	Summary
Map()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance *: Map*	private	
isGeneric *: Boolean*	private	

Property Members

Name	Access	Summary
Instance *: Map*	public	Retorna la instancia de la clase
IsGeneric *: Boolean*	public	

Method Members

Name	Access	Summary
MappingManyToMany() *: Void*	public	Realiza el proceso de mapeo para las colecciones cuyo tipo de relacion sea ManyToMany
MappingOneToMany() *: Void*	public	Realiza el proceso de mapeo para las colecciones cuyo tipo de relacion sea OneToMany

Bag Class

Summary

```
notPublic class Bag : Collection
```

Representa cualquier colección de .Net a la terminología de Nhibernate

Constructor Members

Name	Access	Summary
Bag()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance : *Bag*	private	
isGeneric : *Boolean*	private	

Property Members

Name	Access	Summary
Instance : *Bag*	public	Retorna la instancia de la clase
IsGeneric : *Boolean*	public	

Method Members

Name	Access	Summary
MappingManyToMany() : *Void*	public	Realiza el proceso de mapeo para las colecciones cuyo tipo de relacion sea ManyToMany
MappingOneToMany() : *Void*	public	Realiza el proceso de mapeo para las colecciones cuyo tipo de relacion sea OneToMany

List Class

Summary

```
notPublic class List : Collection
```

Representa a la colecciones System.Collections.IList y System.Collection.Generic.List de .Net.

Constructor Members

Name	Access	Summary
List()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance : *List*	private	
isGeneric : *Boolean*	private	

Property Members

Name	Access	Summary
Instance *: List*	public	Retorna la instancia de la clase
IsGeneric *: Boolean*	public	

Method Members

Name	Access	Summary
MappingManyToMany() *: Void*	public	Realiza el proceso de mapeo para las colecciones cuyo tipo de relacion sea ManyToMany
MappingOneToMany() *: Void*	public	Realiza el proceso de mapeo para las colecciones cuyo tipo de relacion sea OneToMany

Set Class

Summary

```
notPublic class Set : Collection
```

Representa a la colecciones Iesi.Collections.ISet (coleccion definida por Nhibernate.org)

Constructor Members

Name	Access	Summary
Set()	protected	Initializes a new instance of the class.

Field Members

Name	Access	Summary
instance *: Set*	private	
isGeneric *: Boolean*	private	

Property Members

Name	Access	Summary
Instance *: Set*	public	Retorna la instancia de la clase

IsGeneric : *Boolean*	public

Method Members

Name	Access	Summary
MappingManyToMany() : *Void*	public	Realiza el proceso de mapeo para las colecciones cuyo tipo de relacion sea ManyToMany
MappingOneToMany() : *Void*	public	Realiza el proceso de mapeo para las colecciones cuyo tipo de relacion sea OneToMany

CollectionFactory Class

Summary

```
notPublic abstract class CollectionFactory
```

Clase abstracta, maneja todos los tipos de colecciones soportadas

Constructor Members

Name	Access	Summary
CollectionFactory()	protected	Initializes a new instance of the class.

Method Members

Name	Access	Summary
GetCollection() : *Collection*	public	Retorna un objeto del tipo Collection para relaizar el mapeo