UNIVERSIDAD AUSTRAL DE CHILE
CAMPUS PUERTO MONTT
ESCUELA DE INGENIERIA EN COMPUTACION

HERRAMIENTA PARA EL CONTROL DE INTEGRIDAD DE DATOS EN DB4O

Seminario de Titulación
para optar
al título de Ingeniero en Computación

PROFESOR PATROCINANTE:
Sra. CLAUDIA G. ZIL BONTES

ALEX ARTURO SEGOVIA MOLINA

PUERTO MONTT – CHILE
2007

Universidad Austral de Chile
Escuela de Ingeniería en Computación

Los Pinos s/n, Balneario Pelluco
Campus Puerto Montt
Puerto Montt · Chile
Casilla 1327 · Fono: 56 65 260990
Fax: 56 65 277156
Email: ecomputa@uach.cl
www.uach.cl

Puerto Montt, 21 de marzo de 2007

COMUNICACIÓN INTERNA N° 046

DE : Sra. Sandra Ruiz Aguilar
DIRECTORA ESCUELA DE INGENIERIA EN COMPUTACION

A : Dr. Renato Westermeier H. – **DIRECTOR CAMPUS PUERTO MONTT**
Sra. Cristina Barriga – **REGISTRO ACADEMICO**
Sra. Alba Vásquez - **ENCARGADA DE TITULACIÓN CAMPUS PUERTO MONTT**

C.c : Sr. Alex Segovia Molina
Sra. Claudia Zil Bontes
Sra. Mónica Gallardo Vargas
Sr. Moisés Coronado Delgado

MOTIVO:

Informar a usted, las calificaciones obtenidas por el alumno de Ingeniería en Computación **Sr. Alex Arturo Segovia Molina** Rut 12.004.879-1, en su informe de Titulación "***Herramientas para el Control de Integridad de datos en DB40***"

Prof. Claudia Zil Bontes	6.5
Prof. Mónica Gallardo Vargas	6.7
Prof. Moisés Coronado Delgado	6.5
Promedio Seminario	**6.56**

Sin otro particular, le saluda atentamente.

UNIVERSIDAD AUSTRAL DE CHILE
ESCUELA DE INGENIERIA EN COMPUTACION
CAMPUS PUERTO MONTT

SANDRA RUIZ AGUILAR
DIRECTORA

SRA/mva

PUERTO MONTT, 21 de marzo del 2007

De : Sra. Claudia Zil Bontes
PROFESORA COPATROCINANTE

A : Sra. Sandra Ruiz Aguilar
DIRECTORA ESCUELA INGENIERÍA EN COMPUTACIÓN

MOTIVO:

Informar a Usted la calificación obtenida por el alumno ***ALEX ARTURO SEGOVIA MOLINA*** en su Seminario de Titulación **"Herramientas para el Control de Integridad de Datos en DB4O":**

NOTA: 6,5

JUSTIFICACION:

Proceso bien documentado y de clara comprensión. Herramienta novedosa que es muy útil para desarrolladores y permitir una mejor interacción con una BDD.

OTRAS OBSERVACIONES:

Ver al interior del documento.

CLAUDIA ZIL BONTES
PROFESORA PATROCINANTE

PUERTO MONTT, 21 de Marzo de 2007

De : Sra. Mónica Gallardo Vargas
PROFESORA INFORMANTE

A : Sra. Sandra Ruiz Aguilar
DIRECTORA ESCUELA INGENIERÍA EN COMPUTACIÓN

MOTIVO:

Informar a Usted la calificación obtenida por el alumno ***ALEX ARTURO SEGOVIA MOLINA*** en su Seminario de Titulación "**Herramientas para el Control de Integridad de Datos en DB4O**":

NOTA: 6,7

JUSTIFICACION:

Proyecto innovador y actual.
Aun cuando es un prototipo, falto aplicar más pruebas que apoyen los objetivos específicos 4 y 5.-

OTRAS OBSERVACIONES:

MONICA GALLARDO VARGAS
PROFESORA INFORMANTE

PUERTO MONTT, 21 - mar - 2007

De : Sr. Moisés Coronado Delgado
PROFESOR INFORMANTE

A : Sra. Sandra Ruiz Aguilar
DIRECTORA ESCUELA INGENIERÍA EN COMPUTACIÓN

MOTIVO:

Informar a Usted la calificación obtenida por el alumno ***ALEX ARTURO SEGOVIA MOLINA*** en su Seminario de Titulación "**Herramientas para el Control de Integridad de Datos en DB4O**":

NOTA: 6,5

JUSTIFICACION:

- tema de fondo: completamente presentado,
- tema de presentación: sugiero mejorar como por ejemplo el escalado de imagenes, problema en las tablas Pag 22

OTRAS OBSERVACIONES:

MOISES CORONADO DELGADO
PROFESOR INFORMANTE

A mis padres, hermanas, tíos y en especial a Claudia por todo el apoyo brindado para alcanzar este importante logro. Con Amor, para ustedes.

INDICE

Tablas

Figuras

Síntesis

El trabajo planteado a continuación tiene por objetivo el acercar a db4o a los tradicionales gestores de bases de datos relacionales para facilitar el trabajo del desarrollador, en busca de una mayor productividad, proveyéndole un servidor y un administrador de base de datos para db4o que además cuenta con las ventajas de permitir diseñar las bases de datos y generar código C# para estas.

El desarrollo estará enfocado a generar una herramienta distribuida bajo licencia GNU GPL y se regirá por la metodología AUP (Agile Unified Proccess), aplicándose la técnica Test Driven Development (TDD) en pro de la continuidad y calidad del proyecto.

Como resultado se obtiene dicha herramienta que cumple a cabalidad con los objetivos planteados, quedando preparada para su distribución en sitios Web relacionados con db4o.

Abstract

The next raised work has as objetives to approach db4o to the traditional relational database managers to facilitate the work of the developer searching for a greater productivity, providing to him a db4o database server and manager system that in addition counts on the advantages to allow to design the data bases and to generate C# code for these.

The development will be focused to generate a tool distributed under GNU GPL license and it will be governed by the AUP methodology (Agile Unified Proccess), being applied the Test Driven Development (TDD) technique for the continuity and quality of the project.

As result obtains this tool that totally fulfills the raised objectives, being prepared for its distribution in Web sites related to db4o.

1. Introducción

A lo largo de la historia el área de Desarrollo de Software ha debido afrontar cambios constantes debido a la evolución de la Tecnología. A la fecha se ha producido uno bastante importante; la predominancia de la Programación Orientada a Objetos (POO u OOP por sus siglas en inglés – Object Oriented Languajes), lo que ha llevado a la masificación de sus lenguajes de programación.

Pero se ha llegado a un punto donde se encuentran dos realidades totalmente distintas y antagónicas; por un lado la programación se está realizando de manera Orientada a Objetos (OO), y la parte importante de las aplicaciones informáticas –que son los datos- se elabora, modela y desarrolla bajo el modelo Entidad-Relación, lo que deja una sola posibilidad, de momento, a la hora de la elección del Sistema Gestor de Base de Datos (SGBD o DBMS, Data Base Management System) y que son aquellos Relacionales (SGBDR o RDBMS, Relational DBMS).

Hasta ahora la reducción de tiempos y simplificación de procesos de desarrollo ha sido considerable, comparados a los de antaño, pero el problema mencionado anteriormente, conocido como "Desajuste o Diferencia de Impedancia" de objetos (Object-Relational Impedance Mismatch, [Ambler2003]),

está generando un cuello de botella importante al implementar software OO y permitir la persistencia de sus objetos.

Es por ello que la aparición de db4o (Data Base For Objects), un gestor de bases de datos OO, de libre acceso, de gran rendimiento y que elimina el problema de "Impedancia", ha sido importante al permitir ahorrar tiempo valioso y eliminar el trabajo tedioso de realizar un mapeo entre la orientación a objetos y las bases de datos relacionales.

Db4o es el punto de partida de la investigación desarrollada en el presente proyecto, pues aunque tiene todo el potencial necesario como para ser utilizada en reemplazo de un gestor de bases de datos relacionales, no posee todas las características disponibles en estas últimas.

La persistencia de objetos no es "todo" lo que el mundo empresarial necesita, y es que si de Software Informático se habla, el mayor punto bajo observación serán los datos.

En cuanto a esto, db4o ya maneja transparentemente la integridad relacional de los objetos, sin embargo no considera la integridad de las propiedades de dichos objetos, bajo restricciones básicas, lo que demandará tiempo extra de trabajo durante la implementación del Software.

Por otra parte, db4o aún no cuenta con mayores implementaciones al punto de una herramienta de administración de Bases de Datos OO, lo que repercute en los desarrolladores mayormente familiarizados con los RDBMS.

Es por ello que se pretende dar un acercamiento a lo que son los Administradores de Bases de Datos Relacionales, facilitando una herramienta para la administración de las Bases de Datos OO y que ayude en la definición de sus respectivas restricciones de datos.

Esta herramienta se destinará a ser el punto de partida de un trabajo más acabado, lo que implica la reducción de la complejidad de su desarrollo. Este es el punto importante para la elección de la metodología a utilizar para la construcción del software, por lo que se necesita una metodología adaptable, como Agile Unified Process (AUP) que será la utilizada por su simplicidad y adaptación al desarrollo de esta aplicación.

La composición del documento está dada de la siguiente manera:

En el capítulo 2 se exponen los objetivos del proyecto, tanto el objetivo general como los específicos.

El capítulo 3 detalla el planteamiento del problema a abordar, incluyendo su definición, los esfuerzos anteriores y la solución propuesta. También se describe la justificación del proyecto junto a la delimitación que se le hizo.

Para continuar, en el capítulo 4 se describe la metodología utilizada para el desarrollo del proyecto.

El capítulo 5 contiene el detalle de los recursos utilizados en el proyecto, tanto el hardware como software, junto a una descripción de su utilidad.

En los siguientes capítulos de describe el proceso de desarrollo del proyecto, es así como en el capítulo 6 se expone lo realizado en la etapa inicial del desarrollo. Aquí se podrá encontrar información sobre la búsqueda y selección de información relevante al problema a abordar, también sobre la definición del proyecto y lo referente a la preparación del desarrollo.

El capítulo 7 detalla el tema de la elaboración de diseños, además de la preparación del primer prototipo de la aplicación y el plan de pruebas.

En el capítulo 8 se describe la etapa de construcción del software, en las que se encuentra la generación del código de la base de datos, el de la aplicación y las pruebas realizadas al sistema.

El capítulo 9 describe el proceso de publicación y difusión que se realizó para la herramienta.

Los siguientes capítulos contienen las conclusiones extraídas del trabajo realizado junto a la bibliografía utilizada para tal efecto.

2. Objetivos

2.1. Objetivo general

El objetivo general del desarrollo es el diseñar y construir una herramienta para desarrolladores de software, basada en db4o, que otorgue la capacidad de Administración de un Gestor de Base de Datos Orientadas a Objetos, como así también la capacidad de permitir el control de la integridad de los datos.

2.2. Objetivos específicos

El objetivo principal, se puede descomponer en los siguientes objetivos específicos:

- Poner a disposición de los desarrolladores una herramienta de Administración del SGBDOO db4o.

- Permitir conectar concurrentemente a usuarios a una base de datos oo, bajo redes de computadoras.

- Permitir diseñar y visualizar gráficamente el modelo de la base de datos y otorgar la capacidad de definir las restricciones sobre los atributos de manera de controlar la integridad de los datos.

- Dar la capacidad de reducir el tiempo de desarrollo de una aplicación.

- Permitir la reducción en los costos de desarrollo de software.

3. Planteamiento del problema

3.1. Antecedentes

3.1.1 Definición del problema

Junto con la evolución de las computadoras, los lenguajes de programación y los SGBD se debieron adaptar a los cambios, o más bien, debieron sacar provecho de ellos.

En el plano de los SGBD, la predominancia desde la década de los 80's hasta la fecha ha sido de los SGBD Relacionales, dejando atrás en su momento a los SGBD Jerárquicos y en Red, como se puede ver en el artículo de [Marcos2000].

Sin embargo en la misma época ya se notaba la necesidad de herramientas con mayor capacidad, acordes a la POO y con soporte para tipos de datos más complejos, y aunque surgieron algunas, no lograban estar a la altura de lo que se esperaba o no lograron satisfacer lo que quedó descrito más tarde en [Atkinson1989] de lo que debía ser un SGBDOO.

Aún así, la misma descripción presentada en [Atkinson1989] fue criticada más tarde por retractores que preferían buscar mejoras y nuevas características a los SGBD Relacionales, manifestado en [Stonebraker1990].

Pero actualmente la masificación de la POO gracias a las tecnologías .NET de Microsoft, y también la propuesta libre Mono y la popularidad de los SGBD Relacionales han provocado que la implementación de Software Informáticos se vea mermada, más que por la falta de tipos soportados por el SGBD, por lo expuesto en [Ambler2003], debido al tiempo que lleva realizar un mapeo entre ambos, y distintos, enfoques.

Debido a este “Problema de Impedancia”, surgieron también soluciones como “Hibernate”, que es uno de los frameworks más conocidos para realizar la persistencia de objetos de una aplicación OO en bases de datos relacionales. Este tipo de frameworks tiene la desventaja de producir una baja en el rendimiento de la aplicación debido a la labor de mapeo que realizan, lo que a la larga puede provocar problemas si la solución requiere ciertas prestaciones.

Hasta hace poco, no se conocía mayormente acerca de algún SGBD Orientado a Objetos, aunque existían, pero recientemente se ha podido disponer libremente de db4o, un SGBDOO que ha logrado demostrar que su rendimiento es tanto o mejor que los frameworks como “Hibernate” y que está comenzando a

ser utilizado gracias a la poca complejidad que significa realizar la persistencia de los objetos. Más aún, recientemente se le han añadido mejoras que demuestran que el rendimiento de db4o está igualando el desempeño de los SGBDR.

Pero se evidencia que aún faltan cosas por hacer para que en algún momento los sistemas de información dejen de estar ligados a los tradicionales SGBDR. Esas “cosas por hacer” pueden llegar a ser significativas no sólo para los desarrolladores, sino también para que el producto de su trabajo satisfaga las necesidades de los clientes. Por nombrar algunas características no disponibles hasta el momento, se puede mencionar: que no existe una implementación de herramienta administradora del SGBDOO db4o, db4o no maneja integridad de datos, no existe alguna herramienta para una fácil implementación de consultas, entre otras que pudieran surgir.

Si bien en cuanto a consultas los SGBDR ofrecen mayores prestaciones, por el momento db4o tiene la capacidad de replicar sus datos en bases de datos Oracle, que puede ser útil al implementar aplicaciones de Data Warehouse.

3.1.2 Esfuerzos Anteriores

Hasta el momento no se ha desarrollado nada para satisfacer las necesidades anteriormente descritas, db4o está siendo aceptada recientemente y su desarrollo ha estado basado en sus mejoras como DBMSOO más que en facilitar la labor de los desarrolladores de software.

Sólo algunas entidades privadas han implementado sus propios Servidores con db4o para realizar conexiones a través de redes o para el acceso concurrente, pero no se conoce de ellas pues no están a disposición de los demás desarrolladores.

En cuanto al control de integridad de los datos de los objetos, sólo están disponibles las herramientas para realizarlo, tanto a nivel de lógica de negocios (Codificación en el lenguaje OO), como en db4o (“Callbacks”, que son métodos que se ejecutan de manera similar a los “Triggers” en bases de datos relacionales, aunque deben ser declarados en la definición de cada clase). Pero no se ha desarrollado herramienta alguna que facilite esta labor.

3.1.3 Solución propuesta

Esto llevó a la búsqueda de una solución para que los desarrolladores dispongan, de manera libre, de un Administrador del SGBDOO, que le permita realizar labores básicas de administración, definir los dominios o restricciones que puedan tener las Bases de Datos de una aplicación, y así también facilitar el acceso concurrente a Bases de Datos OO sobre una red de computadoras.

La distribución libre de esta herramienta tiene por finalidad hacer participe a otros desarrolladores para mejorar la herramienta de acuerdo a sus conocimientos.

Es por ello que la solución desarrollada es una herramienta de libre acceso que se conforma de la siguiente manera;

- Un servidor de base de datos; está basado en un servicio de Windows e implementado con db4o más el apoyo de información propuesta en [Edlich2006].

- Una interfaz gráfica de administración; que se conecta al servidor mencionado en el punto anterior y que permite llevar a cabo tareas simples

de administración, modelado de la base de datos (de acuerdo a [Kurniawan2003]), la definición de restricciones en los datos y generar el código para el control de las mismas.

3.2. Justificación

- **Situación sin proyecto:**

  La situación sin proyecto implicaría que los desarrolladores continúen trabajando con db4o de la manera habitual. Por un lado, la conexión a la base de datos continuaría o bien embebida en la aplicación, accediendo directamente al archivo de la base de datos sin posibilidad de concurrencia de usuarios, o teniendo que implementar un servidor por cuenta propia para poder disponer de estas características. Por otra parte la administración de las bases de datos, que significa la creación de nuevas bases de datos, eliminación o la asociación de los permisos a ciertos usuarios, quedaría como labor del desarrollador dentro de su aplicación.

  Además, para conseguir un producto de calidad, que asegure la integridad de los datos de forma mínima, cada desarrollador debería

implementar el código necesario para ello, incurriendo en un gasto de tiempo adicional para obtener los mismos resultados.

Sin duda, la situación sin proyecto significa mayor trabajo para los desarrolladores, por consiguiente, mayor tiempo de desarrollo para obtener un producto de calidad.

- **Situación con proyecto:**

Con la disponibilidad libre de esta herramienta los desarrolladores podrán ahorrar costos y tiempo en varias tareas y actividades que comprende un ciclo de vida del desarrollo de software.

Los desarrolladores dispondrán de un administrador del servidor de base de datos orientado a objetos, con una interfaz apropiada para realizar tareas administrativas, como también de una forma más estándar de realizar la conexión con las mismas.

Además al momento de diseñar la base de datos orientada a objetos, podrá luego definir las restricciones para cada atributo de un objeto, logrando evitar la codificación extra que demanda programar ese tipo de

control. Se suma a este ahorro de trabajo, la inevitable mejora en la calidad del producto software sin mayor esfuerzo.

3.3. Delimitación

Las limitantes de la solución que se ha planteado se describen a continuación:

- Debido a lo complejo de lo que significa crear software bajo patrones de usabilidad, la herramienta se diseñó, en cuanto a la interfaz administrativa, lo suficientemente amigable para desarrolladores.

- La herramienta permite diagramar el modelo de la Base de Datos OO, pero sólo se dispone de la habilidad de crear Clases, Asociaciones y Generalizaciones.

- La definición de las Clases estará basada en la asignación del nombre y la creación de sus atributos, no permitiéndose al finalizar el proyecto la especificación de los métodos. Esto pues por el momento se privilegió el logro de los objetivos que apuntan a la manipulación de los atributos principalmente.

- La definición del diagrama no permite detallar información sobre las asociaciones establecidas entre las clases, como el nombre o la multiplicidad por ejemplo.

- La herramienta genera código C# para el control de la integridad de datos, el cual se debe mantener en el código de la aplicación en desarrollo. Esto debido a que los eventos que se pueden controlar al insertar, modificar o eliminar objetos desde la base de datos, deben ser desarrollados en las definiciones de cada clase, pues así ha sido establecido por db4o para su funcionamiento.

- El control de integridad de datos está destinado a atributos de una clase en particular, no permitiéndose la definición de acuerdo a un grupo de ellas.

- El control de la integridad de datos está basada en la definición de Atributos de tipo: único, requerido, además del control de valores.

- La herramienta está desarrollada para su funcionamiento bajo ambiente Windows de momento.

4. Metodología

La metodología utilizada para el desarrollo de la herramienta es “Agile Unified Proccess” (AUP), la cual fue creada y dada a conocer por Scott W. Ambler en Septiembre del 2005, y que se puede revisar en [Ambler2005]. Esta metodología está basada en la conocida “Rational Unified Process” (RUP), sin embargo el autor realizó modificaciones sobre la original para darle un sentido de metodología ágil, induciendo a seguir los valores y principios del Manifiesto Ágil ([Ambler2003]). Ciertamente, AUP es una metodología más liviana que RUP.

La principal razón por la cual se optó por esta metodología, es el hecho de que el software se va generando de forma incremental (Desarrollo de Software Evolucionario). Además posee gran adaptabilidad a los requisitos de cada proyecto; en cada actividad se puede definir qué tareas realizar y cuáles no, según lo requiera cada proyecto en desarrollo.

Otra razón no menos importante que llevó a la elección de esta metodología es el hecho que implica seguir los principios del Manifiesto Ágil, de los cuales el principal es que el desarrollo se guiará más en lograr resultados funcionales que generar una enorme cantidad de documentación.

La metodología se ejecutó en un solo ciclo, es decir, las fases que implica esta metodología se ejecutaron una sola vez, debido a que el proyecto se destinó a ser el punto de partida del desarrollo de una herramienta más acabada y con más características por lo que un segundo ciclo debería ejecutarse una vez que esta herramienta reciba feedback de sus usuarios desarrolladores.

En cuanto a las iteraciones dentro de las fases, éstas se realizaron sobre la Construcción, para permitir el refinamiento de los requisitos y funcionalidades.

Como se aprecia en la figura 1, las principales etapas de la metodología son similares a las de RUP.

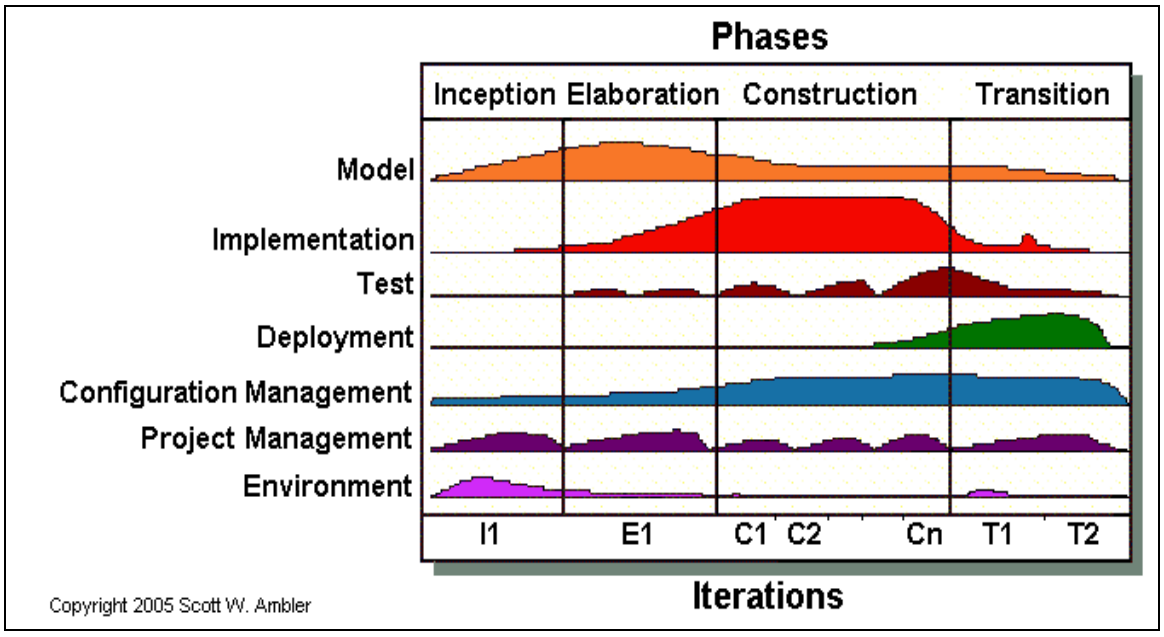

Figura 1. Ciclo de vida de AUP

5. Recursos

5.1. Hardware

El hardware utilizado en la implementación de la herramienta se destinó tanto al desarrollo como a pruebas de la misma.

Tabla Nº 1. Descripción hardware utilizado.

Equipo	Características	Rol
Workstation	Pentium IV modelo 630, 1 GB RAM DDR2, 160 GB HDD. Windows XP Pro.	Desarrollo de herramienta, pruebas, máquina virtual y control de configuración.
IBM ThinkPad T42	Pentium M 1.8 GHZ, 1 GB RAM DDR2, 80 GB HDD. Windows XP Pro.	Pruebas conexión remota y concurrencia a servidor db4o.

El desarrollo no implicó mayores requerimientos, en cuanto a hardware, para justificar la elección de equipos más avanzados. El propósito de la herramienta hará que sus usuarios sean los que deban afrontar la elección correcta de un servidor de acuerdo a requerimientos de rendimiento de sus propios desarrollos.

5.2. Software

La elección del software se basó en el conocimiento propio sobre herramientas de desarrollo utilizadas con anterioridad, además se prefirió el uso software libre o de código abierto (Open Source).

Sin embargo se debió utilizar la herramienta licenciada “VMware Workstation”, para la creación de una máquina virtual “Linux”, donde se configuró la herramienta “Subversion” para el Control de Configuración que implica el desarrollo del software según AUP. “Subversion” es una herramienta que preferentemente se instala en máquinas con Sistemas Operativos Unix o Linux. La utilidad “Cliente” de Subversion será “SmartSvn 2”, que permite conectarse a repositorios existentes en el servidor Subversion y realizar tareas típicas. Además se utilizó la herramienta MagicDraw UML para la diagramación de los modelos de casos de uso y de clases del proyecto.

El “Framework” o la “Librería” db4o, por así denominarla, se consideró como software empleado en el desarrollo del proyecto.

Además, para el desarrollo de la interfaz de usuario, y en específico del modelador de Diagramas de Clases para la Base de Datos OO, se utilizó código de un ejemplo desarrollado en [Kurniawan2003] y que fue adaptado para la aplicación. También se utilizó, como punto de partida, la implementación del Servidor db4o descrito en [Edlich2006].

Finalmente, en el desarrollo de la herramienta se aplicará una de las prácticas que sugiere AUP, y que son las Pruebas de Unidad. Para poder efectuar ésto se necesita de un framework que permita realizar dichas pruebas, y en este caso se eligió NUnit, uno de los primeros en surgir.

Tabla Nº 2. Descripción software utilizado.

Software	Rol
Microsoft Visual C# 2005 Express.	Programación de la herramienta.
db4o	DBMSOO y Facilitación de implementación de nuevas características.
NUnit	Permitir el desarrollo de pruebas de unidad.
Subversion	Permitir el Control de Configuración de la herramienta.
SmartSvn 2	Permitir la conexión a Subversion y la administración de archivos de la herramienta.
VM Ware	Facilitar una máquina virtual donde se almacene Subversion.
MagicDraw UML	Permitir el modelado UML de la aplicación.
Ejemplos: Diagrama de Clases y Servidor db4o	Minimiza la labor de desarrollo, reutilizándose su código.

6. Inicio del proyecto

El proyecto desarrollado surgió debido a la necesidad personal de buscar solución al problema de la “impedancia de objetos”. Este problema está inserto en una gran cantidad de empresas que desarrollan software con tecnologías nuevas como .Net y donde la predominancia del conocimiento sobre el modelo Entidad-Relación lleva a la búsqueda automática de SGBS relacionales tales como MySQL o el popular SQL Server de Microsoft.

6.1. Investigación previa

6.1.1. El problema de la “Impedancia de Objetos”

El problema de la “impedancia de objetos” lleva a las empresas a tener que desarrollar y mantener una cantidad considerable de código fuente que no aporta al objetivo final del desarrollo en curso que es satisfacer la necesidad del cliente. Por ello el trabajo que implica desarrollar tal código podría ser mejor aprovechado.

6.1.2. Soluciones al problema

Una de las soluciones más conocidas es Hibernate, que es considerado el servicio de persistencia de objetos en bases de datos relacionales de mejor rendimiento. Sin embargo, debe considerarse que el hecho de tener que realizar el mismo trabajo que efectúan los desarrolladores, que hacen el mapeo entre sus objetos y las bases de datos relacionales, consume recursos y produce una baja en el rendimiento, comparado a trabajar directamente sobre la base de datos.

Otra solución que ha existido desde hace un tiempo son los SGBDOO, de los cuales poco se conoce pues la mayoría son de costos elevados, por lo tanto de no tan fácil acceso, pero que en general han tenido la poca acogida por su bajo rendimiento.

6.1.3. db4o

El surgimiento de db4o ha constituido una de las soluciones más atractivas del momento. Db4o se creó como un “Framework” de persistencia de objetos y poco a poco ha ido siendo desarrollado y mejorado al punto de convertirse en un auténtico SGBDOO.

En cuanto a su rendimiento, las comparaciones indican que ha sido por mucho tiempo superior a Hibernate y recientemente está igualando el desempeño de los SGBDR. Suena bien, pero una desventaja de los SGBDOO es que en cuanto a consultas no se pueden comparar a los SGBDR, estos últimos son, hasta ahora, muy superiores en cuanto a la variedad y complejidad de consultas que se pueden crear, por lo que pensar en “Data Warehousing” con db4o no resulta tan sencillo; quizá en algún momento sí.

Pero sin duda, es atractivo pensar en la utilización de db4o para el desarrollo de sistemas de información no tan complejos, ya que la utilización de este SGBDOO reduce significativamente el esfuerzo en el desarrollo y produce mejores resultados en el rendimiento final del producto software. Es por esto que está siendo considerado por muchas empresas ante las ventajas que significa su utilización.

6.1.4. Necesidades de db4o

Aunque constituye la mejor opción ante el problema de “impedancia de objetos”, db4o debe afrontar la gran extensión y aceptación que tienen los SGBDR en el mercado, por lo que muchas empresas prefieren continuar trabajando con estos gestores en vez de realizar un cambio.

De aquí surge la idea del proyecto desarrollado en este seminario de titulación, pues db4o aún necesita ayuda para, por un lado, mejorar su aceptación por las empresas, y es que el hecho de ver un archivo o librería “dll” no dice mucho de db4o, y por otro, mejorar la producción de los desarrolladores, facilitando su labor por medio de la automatización y ayuda en tareas simples.

6.2. Definiendo el proyecto y sus requerimientos

Al momento de ahondar en db4o surge la necesidad de acercarlo a los tradicionales SGBDR y sus administradores para así comparar sus diferencias. Pero tal utilidad no existe en db4o; no se cuenta con algún administrador de los usuarios y las bases de datos a las que pueden acceder, ni algún tipo de diagramador para desarrollar el modelo de las bases de datos, entre otras cualidades que poseen los administradores como el “Enterprise manager” de SQL Server.

Una parte de este proyecto fue concebida para mitigar esta necesidad, con lo cual se favorece tanto la difusión de db4o como el trabajo de los desarrolladores. Esa parte se denominó “db4o Manager” y se desarrolló pensando en varios aspectos que son importantes de incluir a comparación de los administradores tradicionales:

- Se requiere que permita administrar bases de datos OO; crearlas, modificarlas y eliminarlas.

- Se requiere poder administrar usuarios y sus respectivos permisos para acceder a las bases de datos.

- Se requiere poder diagramar el modelo de clases, que en este caso representa el esquema de la base de datos.

- Es deseable que a partir del diagrama se pueda generar código fuente que se compare a los populares scripts SQL.

Esta parte del proyecto necesita también de algún servidor que mantenga en línea a las bases de datos y al cual poder conectarse para realizar las labores antes descritas. Esta parte se denominó “db4o Server” y se concibió como un servicio de Windows que realizará sus tareas en “background”.

Al indagar más sobre db4o y sus prestaciones que lo convierten en un SGBDOO, se encuentra el importante tema del manejo de los datos. Una característica típica es la “integridad referencial”; esto es manejado

transparentemente por db4o por lo que su comparación con los SGBDR no tiene mucho sentido, ambos funcionan.

Sin embargo al hablar del manejo de los datos se debe pensar también en la fiabilidad de los datos almacenados, más conocido como “integridad de datos”, expuesto en [Evaltech2000].

Más que importancia para los desarrolladores, esto es vital para los usuarios y clientes (Empresas), por lo que se constituye en un punto importante de calidad del software. El control que se realiza para asegurar la calidad de los datos no es realizado de forma nativa en los SGBDR, pues utilizan los conocidos Triggers o Desencadenadores para realizar las comprobaciones correspondientes a las reglas de negocio (dominio de la aplicación), que se permiten definir en las utilidades de diseño de la base de datos. Igualmente db4o no maneja esto de forma nativa, y que en el caso de los objetos se podría denominar “integridad de objetos”.

Lo que sí provee db4o son los denominados “Callbacks”, que son métodos que se ejecutan ante eventos específicos, al igual que los Triggers. Esto permite que se implemente una característica deseable para este SGBDOO; el control de integridad de datos (Objetos en este caso).

Tabla Nº 3. Descripción de callbacks de db4o.

Nombre	Descripción
ObjectCanActivate	Se ejecuta antes de activar (seleccionar o instanciar desde la base de datos) un objeto.
ObjectCanDeactivate	Se ejecuta antes de desactivar un objeto.
ObjectCanDelete	Se ejecuta antes de la eliminación de un objeto.
ObjectCanNew	Se ejecuta antes de guardar un nuevo objeto.
ObjectCanUpdate	Se ejecuta antes de actualizar un objeto de la base de datos.
ObjectOnActivate	Se ejecuta luego de activar un objeto.
ObjectOnDeactivate	Se ejecuta después de desactivar un objeto.
ObjectOnDelete	Se ejecuta después de eliminar un objeto de la base de datos.
ObjectOnNew	Se ejecuta luego de guardar un nuevo objeto en la base de datos.
ObjectOnUpdate	Se ejecuta luego de actualizar un objeto de la base de datos.

Estos métodos, descritos en la tabla 3, son ejecutados mediante la técnica de "Reflection" ([Shankar2001]), por lo que sólo se debe incluir, en la definición de la clase del objeto, el desarrollo de este método con el mismo nombre.

Con todo esto en mente se puede generar una solución que cumpla con los objetivos planteados en este proyecto, considerando las limitantes detalladas en el capítulo 3.

6.2.1. Consideraciones para el desarrollo

Algunas consideraciones que se deben aclarar y tomar en cuenta sobre db4o y las bases de datos OO en general son las siguientes:

- El modelo de clases constituye el esquema de las bases de datos.

- El código que se genera a partir del modelo o esquema de la base de datos se debe mantener dentro del proyecto de la aplicación en desarrollo.

- Para cada base de datos manejada con db4o se crea un servidor que recibe peticiones en un puerto TCP específico.

Además se debe considerar que la aplicación se desarrollará ante los requisitos propios, descritos anteriormente, establecidos como desarrollador de software. Es por ello que el resultado del seminario está destinado a ser el inicio de una herramienta más acabada, difundida bajo licencia GNU GPL con el fin de buscar desarrolladores interesados en mejorar la aplicación de acuerdo a sus conocimientos y requisitos.

6.3. Preparación del entorno de desarrollo

Con los antecedentes expuestos se realizó la tarea de preparar el ambiente del software, es decir, buscar, seleccionar y mantener disponibles las herramientas y utilidades necesarias para complementar el desarrollo.

Uno de los puntos críticos fue encontrar utilidades para realizar el modelado de clases de la base de datos, se probó el Framework “Piccolo.net”, pero debido a la complejidad de su utilización y a la necesidad de un mejor entrenamiento para su uso, se optó por lo expuesto en [Kurniawan2003], que describe detalladamente la creación de un modelador de clases.

El ejemplo desarrollado en este libro se sometió a pruebas y posteriormente a la adaptación del código para su reutilización. Las pruebas se basaron en evaluar el funcionamiento del software y la utilidad que prestaba, mientras la adaptación

del código se basó en la conversión del código, desde su lenguaje nativo, al lenguaje utilizado en este seminario (c#).

Otro punto clave fue la “Administración de configuración” (Configuration Manager) que expone la metodología utilizada. Uno de los aspectos importantes de este control es la mantención del código fuente y su documentación a lo largo del ciclo de desarrollo. Para ello existen alternativas automatizadas como Subversion, que se implementó en una máquina virtual Linux, en donde se creó un repositorio llamado “Tesis” y al cual se accedió gracias al cliente SmartSVN 2 para efectuar tareas típicas.

6.4. Modelado inicial

El modelado efectuado en la etapa inicial correspondió al diagrama de clases de la base de datos OO que utilizaría la herramienta y al diagrama de casos de usos que representaría los requerimientos de la aplicación. Estos diagramas fueron modificados y mantenidos durante el proceso de elaboración y construcción, el detalle se encuentra en el capítulo siguiente.

7. Elaboración

Esta etapa comprende principalmente la tarea de diseño del modelo de clases de la base de datos OO del sistema y el diseño del diagrama de casos de usos de la aplicación. También se realiza la generación de código respectivo al diagrama de clases y la incorporación del código fuente a reutilizar para establecer un prototipo inicial de la arquitectura del sistema.

7.1. Diseño de la base de datos OO

Como se mencionó con anterioridad, el diseño de la base de datos está dado por su diagrama de clases. El modelo diseñado representa una base de datos OO, manteniendo información acerca de sus usuarios, clases, atributos y reglas.

A diferencia del modelado Entidad-Relación de las bases de datos relacionales, el diagrama de una base de datos OO no necesita pasar por los modelos conceptual, lógico y físico, lo que representa una ventaja a la hora de la mantención del modelo, puesto que significa menor esfuerzo. Esto también significa que no se deben realizar operaciones especiales para representar relaciones N:N o recursivas.

Lo que sí implica es el desarrollo de lógica destinada a controlar reglas de negocio, pues por ejemplo no se definen claves primarias.

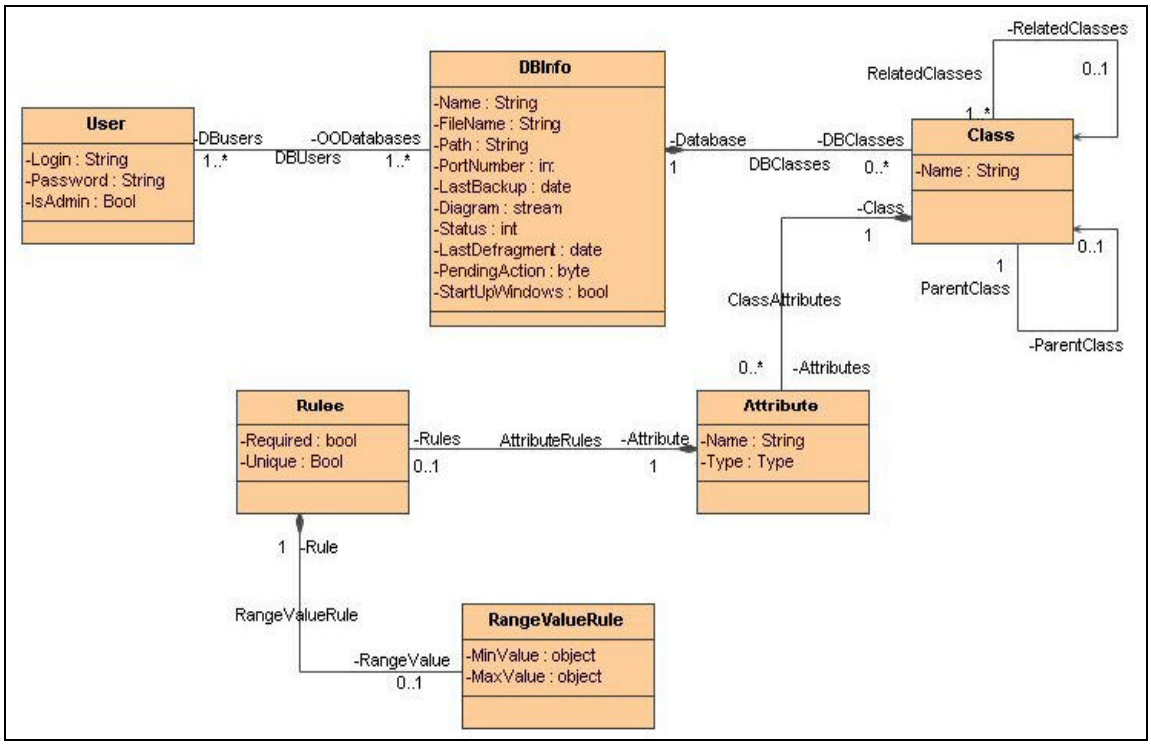

Figura 2. Diseño base de datos del sistema.

Al modelo se le aplicó la técnica de la “Normalización de clases”, definido en [Ambler2003], que tiene por finalidad mejorar la calidad del modelo. Esta normalización es similar a la “Normalización de datos”, y se denominan “Primera forma normal del objeto” (1ONF), “Segunda forma normal del objeto” (2ONF) y “Tercera forma normal del objeto” (3ONF).

Figura 3. Estado anterior de la clase Rules.

La figura 3 muestra el estado de la clase Rules antes de transformarlo a 3ONF. Se está en tercera forma cuando la clase encapsula un grupo de comportamiento cohesionado. En este caso la clase Rules posee atributos que representan un rango de valores que bien podría ser manipulado de forma

distinta al comportamiento de la clase, como obtener la diferencia entre los valores.

7.1.1. Especificación de clases y relaciones

En las siguientes tablas se detallan las clases que componen el modelo de la base de datos y las relaciones que existen entre ellas, sean asociaciones o composiciones.

Tabla Nº 4. Detalle de clases.

Nombre clase	Descripción
Attribute	Son los atributos que contiene un objeto de tipo Class.
Class	Son las clases que contiene un objeto de tipo DBInfo.
DBInfo	Representa una base de datos OO.
RangeValueRule	Representa una regla correspondiente a un rango de números. Esta clase complementa a la clase Rules.
Rules	Representa las reglas que puede contener un objeto de tipo Attribute.
User	Representa usuarios que acceden a objetos DBInfo.

Tabla Nº 5. Detalle de asociaciones.

Nombre asociación	*Descripción*	*Multiplicidad*
DBUsers	Indica que una instancia de DBInfo posee instancias User y viceversa.	*..*
ParentClass	Indica que una instancia de Class puede poseer una instancia Class padre.	0..1
RelatedClasses	Indica que una instancia de Class puede tener relacionado varias instancias Class.	0..*

Tabla Nº 6. Detalle de composiciones.

Nombre composición	*Descripción*	*Multiplicidad*
AttributeRules	Indica que una instancia de Attribute puede estar compuesta de una instancia Rules.	0..1
ClassAttributes	Indica que una instancia de Class puede estar compuesta de varias instancias Attribute.	0..*
DBClasses	Indica que una instancia de DBInfo puede estar compuesta de varias instancias Class.	0..*
RuleComponent	Indica que una instancia de Rules está compuesta también de una instancia RangeValueRule.	1..1

7.1.2. Especificación de atributos

A continuación de describen las propiedades o atributos de las clases pertenecientes al modelo de la base de datos OO del sistema.

Tabla Nº 7. Atributos de la clase “Attribute”.

Nombre atributo	*Tipo*	*Descripción*
Name	Alfanumérico	Nombre que tendrá el atributo.
Type	Tipo	Corresponde al tipo asociado al atributo

Tabla Nº 8. Atributos de la clase “Class”.

Nombre atributo	*Tipo*	*Descripción*
Name	Alfanumérico	Nombre que tendrá la clase.

Tabla Nº 9. Atributos de la clase "DBInfo".

Nombre atributo	Tipo	Descripción
Name	Alfanumérico	Nombre que tendrá la base de datos.
FileName	Alfanumérico	Nombre del archivo de la base de datos.
Path	Alfanumérico	Directorio donde se almacenará el archivo de la base de datos.
PortNumber	Entero	Puerto TCP en que el servidor de la base de datos aceptará peticiones.
LastBackup	Fecha	Fecha del último respaldo hecho.
Diagram	Arreglo de Bytes	Diagrama (Modelo de clases) de la base de datos.
Status	Entero	Estado en que está el servidor de la base de datos (0: Offline; 1: Online).
LastDefragment	Fecha	Fecha de la última defragmentación realizada sobre la base de datos.
PendingAction	Numérico	Acción pendiente a realizar sobre el servidor de la base de datos (0:None, 1:Start; 2: Stop).

StartUpWindows	Booleano	Indica si el servidor de la base de datos debe ser inicializado cuando el Servidor DB4O se inicie automáticamente al arrancar Windows.

Tabla Nº 10. Atributos de la clase "RangeValueRule".

Nombre atributo	*Tipo*	*Descripción*
MinValue	objeto	Corresponde al valor mínimo que se establece como regla para un atributo.
MaxValue	objeto	Corresponde al valor máximo que se establece como regla para un atributo.

Tabla Nº 11. Atributos de la clase “Rules”.

Nombre atributo	Tipo	Descripción
Required	Booleano	Indica que el atributo debe ser requerido, es decir, con valores no null.
Unique	Booleano	Indica que el atributo es único, por lo que no se debe repetir un mismo valor.

Tabla Nº 12. Atributos de la clase “User”.

Nombre atributo	Tipo	Descripción
Login	Alfanumérico	Nombre de usuario.
Password	Alfanumérico	Contraseña del usuario.
IsAdmin	Booleano	Indica si el usuario es Administrador.

7.2. Diseño del diagrama de casos de uso de la aplicación

Este diseño representa los requerimientos que se elaboraron para dotar a la herramienta de funcionalidad.

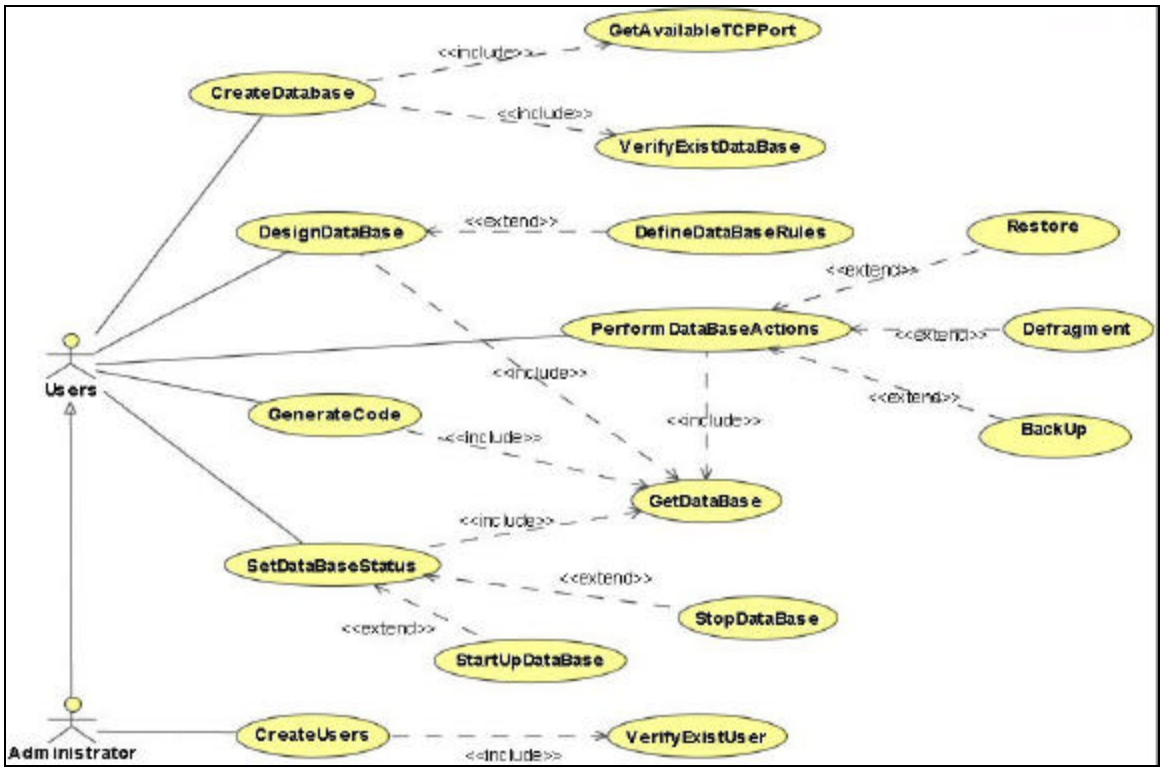

Figura 4. Casos de usos de la aplicación

7.3. Diseño de la aplicación

A partir de los casos de usos diseñados, expuestos en la figura 4, se realizó el diseño de la aplicación. Este diseño se realizó directamente en Visual Studio por medio de la interfaz de diseño de controles.

7.3.1. Diseño general de pantallas

Las pantallas que contendrá la aplicación seguirán el patrón que se muestra en la figura 5.

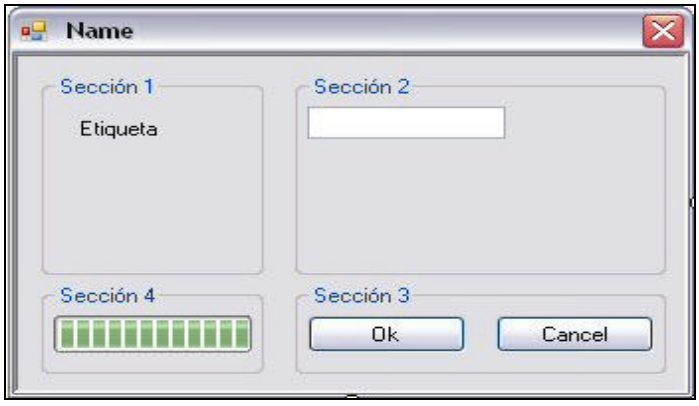

Figura 5. Patrón de diseño de pantallas

Tabla Nº 13. Descripción de secciones del patrón.

Nombre sección	*Descripción*
Sección 1	Contiene etiquetas informativas de los controles desplegados en la Sección 2.
Sección 2	Contiene controles de entrada de datos que el usuario debe ingresar. Existirán N controles de acuerdo a los atributos necesarios a ingresar para los objetos relacionados con la pantalla.
Sección 3	Contiene los botones necesarios para ejecutar una acción en la base de datos y para cancelar. El botón "Ok" cambiará de texto dependiendo de la pantalla. Algunos textos que se encontrarán son "Create", "Save", "Generate", entre otros.
Sección 4	Esta sección es visible en algunas pantallas donde la ejecución del botón "Ok" puede generar un tiempo de espera.
Name	Esta sección contendrá información referente al nombre de la pantalla, el cual da a conocer la acción que se está llevando a cabo.

En algunos casos se utilizó controles de grillas (Grids), las que fueron situadas utilizando las secciones 1 y 2.

7.3.2. Diseño de pantallas

En las siguientes figuras se muestra el diseño de algunas pantallas importantes de la aplicación denominada “DB4O Manager”.

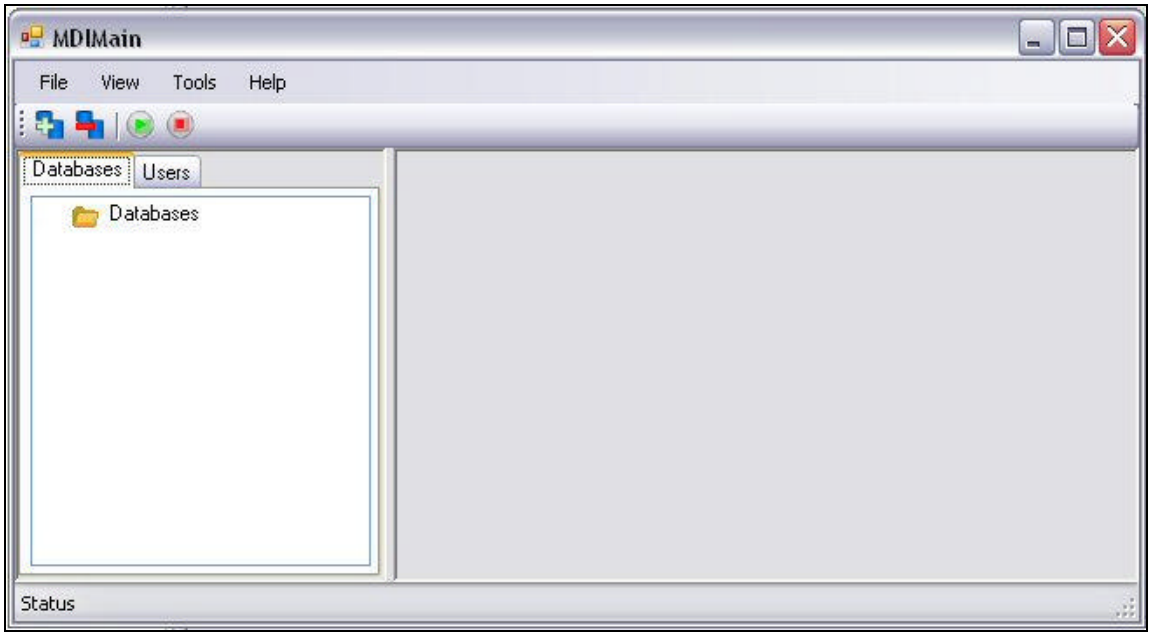

Figura 6. Pantalla principal del Administrador.

En la figura 6 se muestra la pantalla principal de la aplicación. A partir de esta pantalla se generan las demás pantallas destinadas a proveer las distintas funcionalidades.

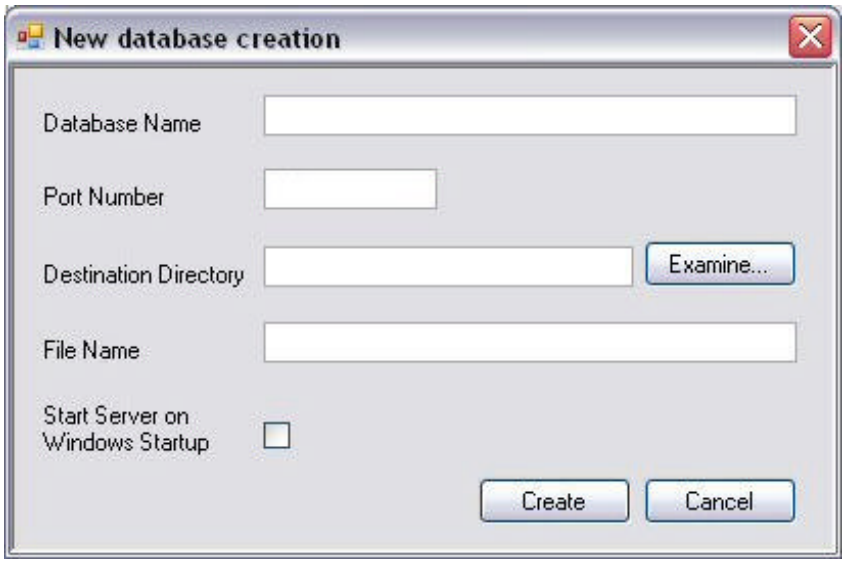

Figura 7. Pantalla de creación de nuevas bases de datos.

Figura 8. Pantalla de creación de nuevos usuarios.

En la figura 9, que se muestra a continuación, se realiza la asignación de acceso a bases de datos para un usuario. En la pantalla no se requiere de la selección del usuario, pues este se envía al formulario al momento de crearlo, según la selección hecha en la pestaña correspondiente a los usuarios existentes en el sistema.

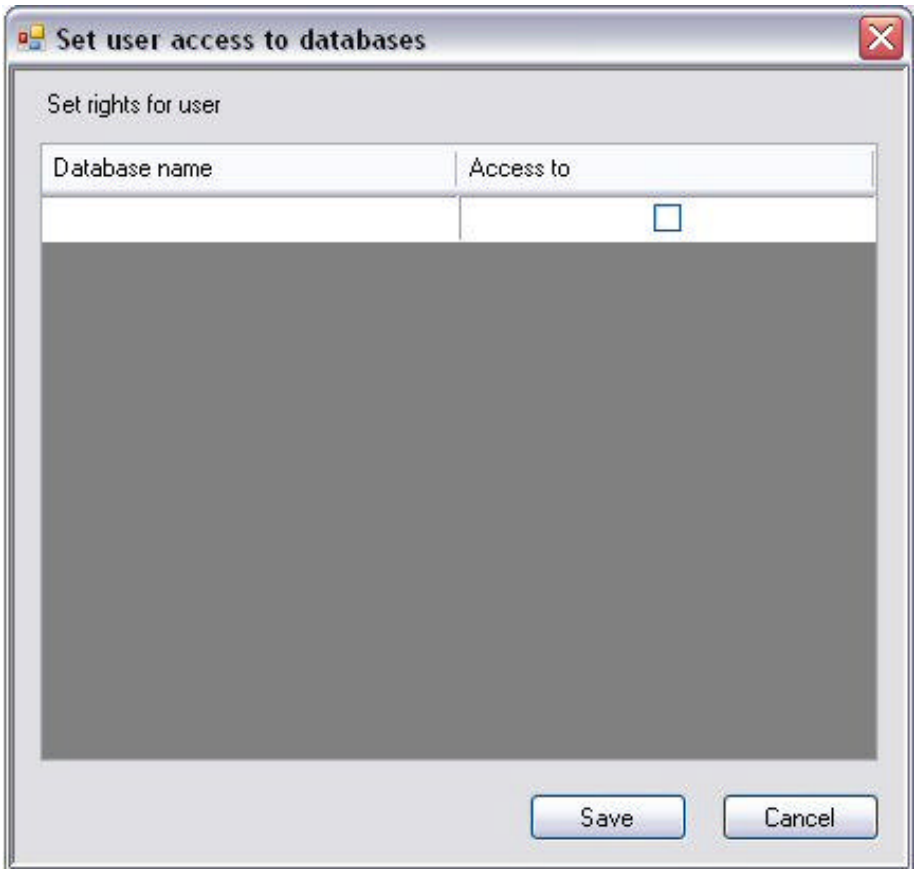

Figura 9. Pantalla de asignación de acceso a base de datos.

La pantalla mostrada en la figura 10 permite la modificación de sólo algunos atributos de la base de datos seleccionada para modificar.

Figura 10. Pantalla de despliegue y modificación de atributos de una base de datos existente.

La figura 11 muestra el diseño para la definición de las clases pertenecientes a una base de datos. Esta pantalla es desplegada a partir del diagramador de clases. Al producirse el evento “Doble clic” sobre una figura “Clase” la pantalla se despliega para ingresar los datos respectivos.

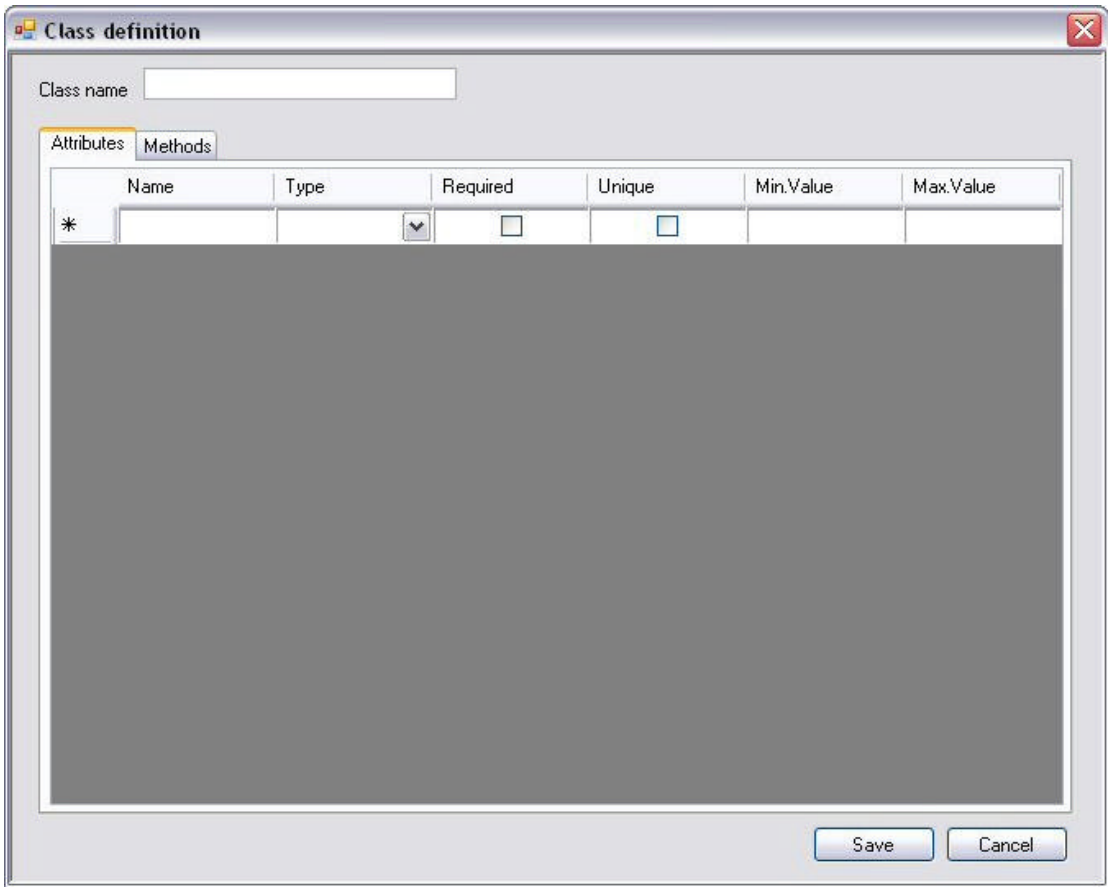

Figura 11. Pantalla de definición de clases.

Las siguientes pantallas se diseñaron para la realización de tareas como el respaldo de una base de datos y la automatización de la creación de código correspondiente al esquema o diagrama de una base de datos.

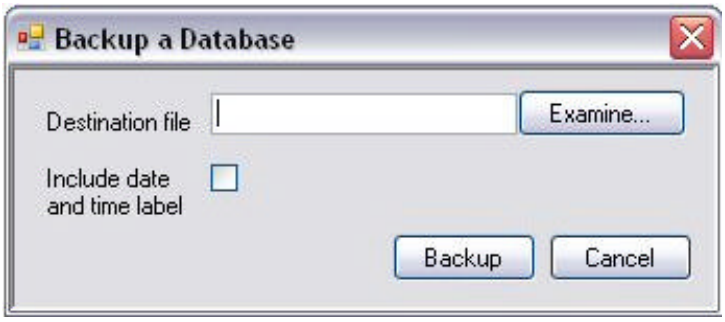

Figura 12. Pantalla para el respaldo de una base de datos.

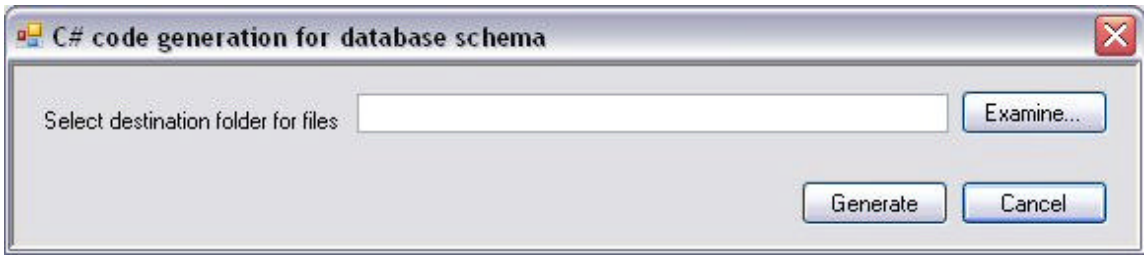

Figura 13. Pantalla de generación de código.

7.3.2.1. Criterios de usabilidad de los diseños

Los criterios de usabilidad utilizados en el diseño de las interfaces están relacionados con el usuario objetivo de la aplicación, que son los desarrolladores y que cuentan con un grado de conocimiento de aplicaciones similares.

Entre los criterios tomados en cuenta están:

- La interfaz se diseñó para que las pantallas del sistema sean accesibles en a lo más 3 eventos clic.

- La navegación entre la pantalla principal y las restantes es sencilla; la estructura de organización es de 1 nivel.

- La interfaz es familiar debido a su similitud con otras herramientas de desarrollo, como el “Enterprise manager” de SQL Server.

- Los colores utilizados son estándares; el ingreso de textos se realiza con color de texto negro sobre fondo blanco y los campos no disponibles para edición tienen un fondo gris.

- Los íconos de la barra de herramienta cuentan con un texto informativo sobre la función que cumplen.

- Las pantallas son consistentes; se lanzan sobre la pantalla MDI principal y no se puede realizar otra acción hasta culminar la que se encuentra en curso, no permiten modificación de tamaño, sólo cuentan con el botón cerrar en la parte superior y los botones de acción son siempre 2.

- En pantallas que la acción a realizar pueda tomar un tiempo de espera para la ejecución total se despliega una barra de progreso para mantener informado al usuario.

- Los niveles de pantalla son para todas iguales; 1. Cada pantalla realiza una función independiente.

- Los controles de pantalla se configuraron para permitir el paso de un control al siguiente mediante la tecla “TAB”. Luego del último control en pantalla, el siguiente siempre es el botón “Ok” (Save, Create, Generate).

- Ante errores se generan cuadros de mensaje con un contenido informativo sobre qué hacer. Además el servidor denominado “DB4O

Server” mantiene un registro en un archivo de “Log” con las acciones que se han realizado en él para mantener trazabilidad y detectar posibles problemas.

7.3.3. Estructura de organización de la aplicación

La forma de acceder desde la pantalla principal de la aplicación se puede apreciar en la figura 14 expuesta a continuación. Se puede distinguir la simplicidad del diseño para facilitar el acceso a las distintas funcionalidades.

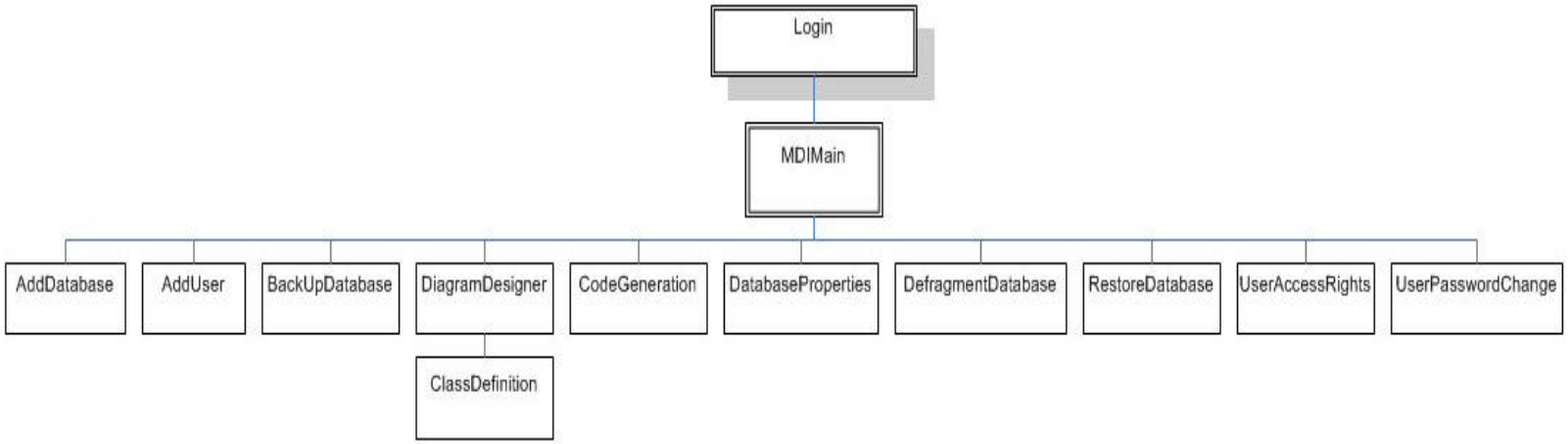

Figura 14. Estructura de organización de la aplicación.

7.4. Generación del código fuente de la base de datos

Mediante la herramienta UML Draw se generó código a partir del diseño de clases de la base de datos del sistema. Este código fue mejorado para seguir ciertos estándares de codificación; la utilización de lowerCamelCase para definir nombres de variables, UpperCamelCase para las propiedades y preceder el nombre de atributos privados con "_", son algunos.

7.5. Incorporación de código reutilizable

Luego de las pruebas y conversión del código fuente del diagramador de clases, realizado en la etapa de inicio, se incorporaron dichos archivos de código al proyecto en desarrollo para luego generar una primera solución sin errores de compilación o ejecución ni advertencias.

Con ello se generó el primer prototipo de la arquitectura del sistema, tan sólo el caparazón del "DB4O Manager" y el primer acercamiento al servicio de Windows que implementaría el mencionado "DB4O Server".

7.6. Generación del plan de pruebas de la aplicación

Como se mencionó, las pruebas de la herramienta serán realizadas mediante UnitTest (Pruebas de Unidad). Estas pruebas tienen la doble finalidad de, por una parte simular un proceso completo que pueda ocurrir sobre una base de datos, ante situaciones normales de ingreso y por otro lado ayudar a detectar errores ante cambios en el código de la aplicación. Para llevar a cabo esto, se deberá implementar el código necesario, en la etapa de construcción, para representar lo definido en el siguiente plan de pruebas:

7.6.1. Plan de pruebas mediante pruebas de unidad

- Crear nueva base de datos llamada uTestDB.
- Inicializar base de datos uTestDB.
- Probar conexión a uTestDB.
- Modificar atributos de base de datos uTestDB.
- Probar conexión a uTestDB.
- Crear usuario uTestUser.
- Asignar permiso a uTestUser a la base de datos uTestDB.
- Importar diagrama previo generado a la base de datos uTestDB.
- Generar código de acuerdo al diagrama de uTestDB.

- Comprobar código nuevo generado con código previamente generado.
- Defragmentar base de datos uTestDB.
- Respaldar base de datos uTestDB.
- Eliminar usuario uTestUser.
- Eliminar base de datos uTestDB.

8. Construcción

Una vez generado el prototipo inicial se continuó con el desarrollo de la aplicación de la manera en que se describe a continuación.

8.1. Consideración de la licencia de software

Al difundir el software bajo licencia GNU GPL se debe tener en consideración que cada archivo de código que se genere en la aplicación deberá contener la siguiente cláusula en el encabezado:

```
/*  This file is part of DB4O Server & Manager.

    DB4O Server & Manager is free software; you can redistribute it and/or
modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    DB4O Server & Manager is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with DB4O Server & Manager; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/
```

8.2. Implementación de la base de datos

Luego de generados los archivos de código iniciales, que representan el esquema de la base de datos OO del sistema, se mejoró el código resultando:

8.2.1. Clase “Attribute”

```
public class Attribute
{
        //Atributos privados
        private string _name;
        private Type _type;
        private Rules _rules;

        //Propiedades públicas que permiten interactuar con los
        //atributos privados

        public string Name
        {
           get
           {
              return _name;
           }
           set
           {
              _name = value;
           }
        }

        public Type Type
        {
           get
           {
              return _type;
           }
           set
```

```
            {
                _type = value;
            }
        }

        public Rules Rules
        {
            get
            {
                return _rules;
            }
        }

        //Constructores de la clase
        public Attribute() { }

        public Attribute(string name, Type type)
        {
            Name = name;
            Type = type;
        }

        public Attribute(string name, Type type, bool isRequired, bool isUnique,
object minValue, object maxValue)
        {
            Name = name;
            Type = type;
            _rules = new Rules(isRequired, isUnique, minValue, maxValue);
        }

        public Attribute(string name, Type type, bool isRequired, bool isUnique, int
maxLenght)
        {
            Name = name;
            Type = type;
            _rules = new Rules(isRequired, isUnique, maxLenght);
        }

        //Implementación reglas de negocio de inserción (requeridos, únicos y
        //valores)
        public Boolean objectCanNew(IObjectContainer container)
        {
```

```
            //Comprobación de valores requeridos
            if ((Name == null) || (Type == null))
                return false;

            return true;
        }

        //Implementación reglas de negocio de actualización
        public Boolean objectCanUpdate(IObjectContainer container)
        {
            //Comprobación de valores requeridos
            if (Type == null)
                return false;

            return true;
        }
}
```

8.2.2. Clase “Class”

```
public class Class
{
        //Atributos privados
        private string _name;
        private Attribute[] _attributes;
        private Class _parentClass;
        private Class[] _relatedClasses;

        //Propiedades públicas
        public string Name
        {
            get
            {
                return _name;
            }
            set
            {
                _name = value;
            }
```

```
}

public Attribute[] Attributes
{
   get
   {
      return _attributes;
   }
}

public Class ParentClass
{
   get
   {
      return _parentClass;
   }
   set
   {
      _parentClass = value;
   }
}

public Class[] RelatedClasses
{
   get
   {
      return _relatedClasses;
   }
}


//Constructores
public Class() { }

public Class(string className)
{
   Name = className;
}

//Implementación reglas de negocio de inserción
public Boolean objectCanNew(IObjectContainer container)
{
```

```
            //Comprobación de valores requeridos
            if (Name == null)
                return false;

            return true;
        }

}
```

8.2.3. Clase “DBInfo”

```
public class DBInfo
{
        private User[] _dBUsers;
        private Class[] _dBClasses;
        private string _name;
        private int _portNumber;
        private string _fileName;
        private string _path;
        private DateTime _lastBackup;
        private DateTime _lastDefragment;
        private Stream _diagram;
        private byte _status;
        private byte _pendingAction;
        private bool _startUpWindows;

        public User[] DBUsers
        {
            get
            {
                return _dBUsers;
            }
        }

        public Class[] DBClasses
        {
            get
            {
```

```
            return _dBClasses;
        }
    }

    public string Name
    {
        get
        {
            return _name;
        }
        set
        {
            _name = value;
        }
    }

    public int PortNumber
    {
        get
        {
            return _portNumber;
        }
        set
        {
            _portNumber = value;
        }
    }

    public string FileName
    {
        get
        {
            return _fileName;
        }
        set
        {
            _fileName = value;
        }
    }

    public string Path
    {
        get
```

```
    {
        return _path;
    }
    set
    {
        _path = value;
    }
}

public string EntirePath
{
    get
    {
        return @Path + @"\" + @FileName;
    }
}

public DateTime LastBackup
{
    get
    {
        return _lastBackup;
    }
    set
    {
        _lastBackup = value;
    }
}

public DateTime LastDefragment
{
    get
    {
        return _lastDefragment;
    }
    set
    {
        _lastDefragment = value;
    }
}

public Stream Diagram
{
```

```
    get
    {
        return _diagram;
    }
    set
    {
        _diagram = value;
    }
}

public DBStatus Status
{
    get
    {
        return (DBStatus)_status;
    }
    set
    {
        _status = Convert.ToByte(value);
    }
}

public DBPendingAction PendingAction
{
    get
    {
        return (DBPendingAction)_pendingAction;
    }
    set
    {
        _pendingAction = Convert.ToByte(value);
    }
}

public bool StartUpWindows
{
    get
    {
        return _startUpWindows;
    }
    set
    {
        _startUpWindows = value;
```

```
    }
}

public DBInfo() { }

public DBInfo(string name, int portnumber, string fileName, string path)
{
    Name = name;
    PortNumber = portnumber;
    FileName = fileName;
    Path = path;
}

//Implementación reglas de negocio de inserción
public Boolean objectCanNew(IObjectContainer container)
{
    //Comprobar valor único de _name
    IQuery nameUniqueQuery = container.Query();
    nameUniqueQuery.Constrain(typeof(DBInfo));
    nameUniqueQuery.Descend("_name").Constrain(this.Name);
    IObjectSet nameUniqueResult = nameUniqueQuery.Execute();
    if (nameUniqueResult.Count != 0)
        return false;

    //Comprobar valor único de _path y _fileName
    IQuery entirePathUniqueQuery = container.Query();
    entirePathUniqueQuery.Constrain(typeof(DBInfo));
    entirePathUniqueQuery.Descend("_path").Constrain(this.Path);
    entirePathUniqueQuery.Descend("_fileName").Constrain(this.FileName);
    IObjectSet entirePathUniqueResult = entirePathUniqueQuery.Execute();
    if (entirePathUniqueResult.Count != 0)
        return false;

    //Comprobar valor único de _portNumber
    IQuery portUniqueQuery = container.Query();
    portUniqueQuery.Constrain(typeof(DBInfo));
    portUniqueQuery.Descend("_portNumber").Constrain(this.PortNumber);
    IObjectSet portUniqueResult = portUniqueQuery.Execute();
    if (portUniqueResult.Count != 0)
        return false;

    //Comprobación de valores requeridos
    if ((Name == null) || (Path == null) || (FileName == null))
```

```
            return false;

        //Comprobar rango de valores para PortNumber
        if ((PortNumber < 1025) || (PortNumber > 65536) || (PortNumber ==
8731))
            return false;


        return true;
    }

    //Implementación reglas de negocio de actualización
    public Boolean objectCanUpdate(IObjectContainer container)
    {
        //Comprobar valor único de _path y _file Name
        IQuery entirePathUniqueQuery = container.Query();
        entirePathUniqueQuery.Constrain(typeof(DBInfo));
        entirePathUniqueQuery.Descend("_name").Constrain(this.Name).Not();
        entirePathUniqueQuery.Descend("_path").Constrain(this.Path);
        entirePathUniqueQuery.Descend("_fileName").Constrain(this.FileName);
        IObjectSet entirePathUniqueResult = entirePathUniqueQuery.Execute();
        if (entirePathUniqueResult.Count != 0)
            return false;

        //Comprobar valor único de _portNumber
        IQuery portUniqueQuery = container.Query();
        portUniqueQuery.Constrain(typeof(DBInfo));
        portUniqueQuery.Descend("_name").Constrain(this.Name).Not();
        portUniqueQuery.Descend("_portNumber").Constrain(this.PortNumber);
        IObjectSet portUniqueResult = portUniqueQuery.Execute();
        if (portUniqueResult.Count != 0)
            return false;
        //Comprobar valores requeridos
        if ((Path == null) || (FileName == null))
            return false;
        //Comprobar rango de valores para PortNumber
        if ((PortNumber < 1025) || (PortNumber > 65536) || _
                (PortNumber == 8731))
            return false;

        return true;
    }
}
```

8.2.4. Clase “RangeValueRule”

```
public class RangeValueRule
{
        private object _minValue;
        private object _maxValue;

        public object MinValue
        {
           get
           {
              return _minValue;
           }
           set
           {
              _minValue = value;
           }
        }

        public object MaxValue
        {
           get
           {
              return _maxValue;
           }
           set
           {
              _maxValue = value;
           }
        }

        public RangeValueRule(){ }


        public RangeValueRule(object minValue, object maxValue)
        {
           MinValue = minValue;
           MaxValue = maxValue;
        }
}
```

8.2.5. Clase “Rules”

```
public class Rules
{
    private RangeValueRule _rangeValue;
    private bool _required;
    private bool _unique;

    public RangeValueRule RangeValue
    {
      get
      {
        return _rangeValue;
      }
    }

    public bool Required
    {
      get
      {
        return _required;
      }
      set
      {
        _required = value;
      }
    }

    public bool Unique
    {
      get
      {
        return _unique;
      }
      set
      {
        _unique = value;
      }
    }

    public Rules() { }
```

```
        public Rules(bool isRequired, bool isUnique)
        {
            Required = isRequired;
            Unique = isUnique;
        }

        public Rules(bool isRequired, bool isUnique, int
maxLenght):this(isRequired, isUnique, (object)0, maxLenght)
        {
        }

        public Rules(bool isRequired, bool isUnique, object minValue, object
maxValue)
        {
            Required = isRequired;
            Unique = isUnique;
            _rangeValue = new RangeValueRule(minValue, maxValue);
        }
}
```

8.2.6. Clase “User”

```
public class User
{
        private string _login;
        private string _password;
        private DBInfo[] _oODatabases;
        private bool _isAdmin;

        public string Login
        {
            get
            {
                return _login;
            }

            set
            {
                _login = value;
```

```
    }
}

public string Password
{
    get
    {
        return _password;
    }
    set
    {
        _password = value;
    }
}

public DBInfo[] OODataBases
{
    get
    {
        return _oODatabases;
    }
}

public bool IsAdmin
{
    get
    {
        return _isAdmin;
    }
    set
    {
        _isAdmin = value;
    }
}

public User() { }

public User(string userName, string password)
{
    Login = userName;
    Password = password;
    IsAdmin = false;
}
```

```
        public User(string userName, string password, bool isAdmin)
        {
            Login = userName;
            Password = password;
            IsAdmin = isAdmin;
        }

        //Implementación reglas de negocio de inserción
        public Boolean objectCanNew(IObjectContainer container)
        {
            //Comprobar valores únicos
            IQuery uniqueQuery = container.Query();
            uniqueQuery.Constrain(typeof(User));
            uniqueQuery.Descend("_login").Constrain(this.Login);
            IObjectSet uniqueResult = uniqueQuery.Execute();
            if (uniqueResult.Count != 0)
                return false;

            //Comprobar valores requeridos
            if ((Login == null) || (Password == null))
                return false;

            return true;
        }

        //Implementación reglas de negocio de actualización
        public Boolean objectCanUpdate(IObjectContainer container)
        {
            //Comprobar valores requeridos
            if ((Login == null) || (Password == null))
                return false;

            return true;
        }
    }
```

8.3. Implementación de DB4O Server (servicio Windows)

El servicio de Windows implementado se conforma principalmente por 2 clases importantes; una que representa el servicio como tal y que debe contener ciertos métodos con nombres específicos para poder funcionar y otra que representa un servidor para bases de datos OO en db4o (Cuya base fue extraída de[Edlich2006]).

A continuación de describe cada una de estas clases mencionadas respectivamente.

8.3.1. Clase “DB4OServerService”

```
public class DB4OServerService : ServiceBase
{
	//Declaración de constantes y atributos privados
	public const string MyServiceName = "DB4OServer";
	private Hashtable _onLineServers;
	private LogFile _logMessages;

	//Declaración de propiedades privadas
	private String SystemDataPath
	{
	   get
	   {
	      return ConfigurationManager.AppSettings["SystemDataPath"];
	   }
	}

	private String SystemServiceLogPath
```

```
{
   get
   {
      return ConfigurationManager.AppSettings["SystemServiceLogPath"];
   }
}

public DB4OServerService()
{
       InitializeComponent();
}

//Métodos propios de un objeto Service
private void InitializeComponent()
{
       this.ServiceName = MyServiceName;
       _onLineServers = new Hashtable();
       _logMessages = new LogFile(SystemServiceLogPath);
 }



protected override void Dispose(bool disposing)
{
       base.Dispose(disposing);
}

//Método del servicio ejecutado cuando se inicia
protected override void OnStart(string[] args)
{

        DBInfo systemDB = new DBInfo("DB4OServer", 8731, _
                        "DB4OServer.oodb", SystemDataPath);


        systemDB.AddUser(SystemDatabase.Common.Utilities. _
                        SystemDBUtilities.GetAdminUser());

        //Inicialización de la base de datos del sistema
        StartDBServer(systemDB);

        IObjectContainer miDb = _
                SystemDBClient.GetInstance().GetDbClient();
```

```
            SystemDatabase.SysDBClasses.User admin = new _
                SystemDatabase.SysDBClasses.User("admin", "admin", true);

            IObjectSet users = miDb.Get(admin);
            if (users.Count == 0)
            {
                miDb.Set(admin);
                miDb.Commit();
            }

            miDb.Close();

            IObjectContainer sysDatabaseClient = _
                                    SystemDBClient.GetInstance().GetDbClient();

            //Consulta por todas las bases de datos creadas
            IObjectSet dataBases = sysDatabaseClient.Get(new DBInfo());

            _logMessages.Log();
            _logMessages.Log("Starting the service");

            //Inicialización de todas las bases de datos que tienen la
            //propiedad StartUpWindows en true
            foreach (DBInfo DB in dataBases)
            {
                if (DB.StartUpWindows)
                {
                    _logMessages.Log(DB.Name + " database initialized");
                    StartDBServer(DB);
                    DB.Status = DBStatus.OnLine;
                    sysDatabaseClient.Set(DB);
                }
                else
                {
                    _logMessages.Log(DB.Name + " database not initialized port:"_
                                            + DB.PortNumber.ToString() + ", status:" _
                                            + DB.Status.ToString() + ")");
                }
            }
            SystemDBClient.GetInstance().CloseDbClient();
        }
```

```
//Método del servicio ejecutado cuando se detiene
protected override void OnStop()
{
   _logMessages.Log();
   _logMessages.Log("Stoping the service");

   SystemDatabase.SysDBClasses.User admin = _
                                        SystemDBUtilities.GetAdminUser();

   //Se detienen todos los servidores que están en ejcución
   foreach (DictionaryEntry serverEntry in _onLineServers)
   {
      Server stopServer = (Server)serverEntry.Value;

      IObjectContainer stoperClient = Db4oFactory.OpenClient("localhost",
         stopServer.DataBase.PortNumber, admin.Login, admin.Password);
      IMessageSender messageSender = _
         stoperClient.Ext().Configure().ClientServer().GetMessageSender();
      messageSender.Send(new StopServerDatabase("Database server _
         stoped by service"));
      _logMessages.Log(stopServer.DataBase.Name + " database _
         stoped");
   }
   _onLineServers.Clear();
}

//Método del servicio ejecutado cuando se envía la solicitud de
//ejecutar un comando
protected override void OnCustomCommand(int command)
{
   switch ((ServiceCommand)command)
   {
      case ServiceCommand.StartDBServer: StartDataBase();
         break;
      case ServiceCommand.StopDBServer: StopDataBase();
         break;
   }
}



//Método que se encarga de inicializar un servidor cuando se
//envía una solicitud desde el administrador
```

```
private void StartDataBase()
{
    IObjectContainer sysDatabaseClient = _
        SystemDBClient.GetInstance().GetDbClient();

    IQuery query = sysDatabaseClient.Query();
    query.Constrain(typeof(DBInfo));
    query.Descend("_pendingAction").Constrain(_
        (byte)DBPendingAction.Start);
    query.Descend("_status").Constrain((byte)DBStatus.OffLine);

    IObjectSet queryResult = query.Execute();

    _logMessages.Log();
    _logMessages.Log("Starting a new database server");
    try
    {
        //Manejo de errores y “log” de ellos
        if (queryResult.Count == 1)
        {
            DBInfo foundDB = (DBInfo)queryResult.Next();
            StartDBServer(foundDB);
            foundDB.PendingAction = DBPendingAction.None;
            foundDB.Status = DBStatus.OnLine;
            sysDatabaseClient.Set(foundDB);
            sysDatabaseClient.Commit();

            _logMessages.Log(foundDB.Name + " database started _
                (port:" + foundDB.PortNumber.ToString() + ")");
        }
        else if (queryResult.Count > 1)
        {
            throw new Exception("Only one database server can be started_
                at a time");
        }
        else
        {
            throw new Exception("The database server is already running");
        }
    }
    catch (Exception ex)
    {
        _logMessages.Log(ex.ToString());
```

```
    }
    SystemDBClient.GetInstance().CloseDbClient();
}

//Método que se encarga de detener un servidor cuando se
//envía una solicitud desde el administrador
private void StopDataBase()
{
    ObjectContainer sysDatabaseClient = _
        SystemDBClient.GetInstance().GetDbClient();

    IQuery query = sysDatabaseClient.Query();
    query.Constrain(typeof(DBInfo));
    query.Descend("_pendingAction").Constrain(_
        (byte)DBPendingAction.Stop);
    query.Descend("_status").Constrain((byte)DBStatus.OnLine);

    IObjectSet queryResult = query.Execute();

    _logMessages.Log();
    _logMessages.Log("Stoping a database server");


    try
    {
        //Manejo y Log de errores
        if (queryResult.Count == 1)
        {
            DBInfo foundDB = (DBInfo)queryResult.Next();
            SystemDatabase.SysDBClasses.User admin = _
                SystemDBUtilities.GetAdminUser();

            //Se envía un mensaje de Stop al servidor para su detención
            IObjectContainer stoperClient = Db4oFactory.OpenClient(_
                "localhost", foundDB.PortNumber, admin.Login, _
                admin.Password);

            IMessageSender messageSender=stoperClient.Ext().Configure()._
                ClientServer().GetMessageSender();

            messageSender.Send(new StopServerDatabase(_
                "Database server stoped by user"));
```

```
                _onLineServers.Remove(foundDB.Name);

                foundDB.PendingAction = DBPendingAction.None;
                foundDB.Status = DBStatus.OffLine;
                sysDatabaseClient.Set(foundDB);
                sysDatabaseClient.Commit();

                _logMessages.Log(foundDB.Name + " database stoped (port:" _
                        + foundDB.PortNumber.ToString() + ")");
            }
            else if (queryResult.Count > 1)
            {
                throw new Exception("Only one database server can be _
                        stopped at a time");
            }
            else
            {
                throw new Exception("The database server is already stopped");
            }
        }
        catch
        {
            _logMessages.Log(ex.ToString());
        }

        SystemDBClient.GetInstance().CloseDbClient();
    }

    //Método que crea una instancia de la clase “Server”
    //que representa al servidor db4o de la base de datos
    private void StartDBServer(DBInfo db)
    {
        Server newServer = new Server(db);

        _onLineServers.Add(newServer.DataBase.Name, newServer);

        //Creación de un hilo independiente para el servidor
        Thread serverThread = new Thread(new ThreadStart(newServer.Run));


        serverThread.Start();
    }
}
```

8.3.2. Clase “Server”

```
public class Server:Db4objects.Db4o.Messaging.IMessageRecipient
{
	//Atributos privados de la clase
	private DBInfo _dataBase;
	private IObjectServer _dbServer;

	//Propiedades públicas de la clase
	public DBInfo DataBase
	{
	   get
	   {
	      return _dataBase;
	   }
	}

	public IObjectServer DataBaseServer
	{
	   get
	   {
	      return _dbServer;
	   }
	}

	//Constructor de la clase
	public Server(DBInfo dataBase)
	{
	   _dataBase = dataBase;
	}

	//Método ejecutado al iniciar el hilo del servidor
	public void Run()
	{
	   lock (this)
	   {
	      _dbServer = Db4oFactory.OpenServer(@_dataBase.EntirePath,_
	                        _dataBase.PortNumber);

	       //Se configura el servidor para que acepte mensajes enviados
	       _dbServer.Ext().Configure().ClientServer(). _
```

```
            SetMessageRecipient(this);

        if (_dataBase.DBUsers != null)
        {
            //Se asignan los permisos para los usuarios asociados
            foreach (SystemDatabase.SysDBClasses.User user in _
                        _dataBase.DBUsers)
                _dbServer.Ext().GrantAccess(user.Login, user.Password);
        }
    }
}

//Método ejecutado cuando el objeto recibe un mensaje
public void ProcessMessage(IObjectContainer con, Object message)
{
    DB4OServer.Common.Utilities.LogFile logFile = new _
        DB4OServer.Common.Utilities.LogFile((string) _
        ConfigurationManager.AppSettings["SystemServiceLogPath"]);
    lock (this)
    {
        if (message is StopServerDatabase)
        {
            try
            {
                //Si se recibe un mensaje Stop se detiene el servidor
                _dbServer.Close();
                logFile.Log(message.ToString());
            }
            catch (Exception ex)
            {
                logFile.Log(ex.ToString());
            }

        }
        else if (message is BackupServerDatabase)
        {

            BackupServerDatabase backUpThis = _
                                        (BackupServerDatabase)message;


            try
            {
```

```
                    //Si se recibe un mensaje BackUp se respalda la base de datos
                    //sin necesidad de detener el servidor
                    _dbServer.Ext().Backup(@backUpThis.DestinationFile);
                    logFile.Log(_dataBase.Name + " backup:" + _
                        message.ToString());

                }
                catch (Exception ex)
                {
                    logFile.Log(ex.ToString());
                }
            }
        }
    }
}
```

Los servidores creados se manejan en múltiples hilos con el fin de independizar a cada uno respecto al consumo de recursos. Así se evita que si alguno de los servidores se “cuelga” todos se vean afectados.

8.4. Implementación de DB4O Manager (aplicación Windows)

Una vez que los diseños de pantallas estuvieron generados, se continuó con el proceso de dotarlos de funcionalidad. De ellas se describen en las secciones siguientes algunas más importantes. El resto está disponible para su revisión en el Anexo 1.

8.4.1. Clase “SystemDBClient”

Esta clase es importante comentar. Corresponde a una clase que se encarga de que en el sistema exista sólo una instancia de la clase misma. Por ello, esta clase contiene un objeto IObjectContainer, que representa a la base de datos OO del sistema. El IObjectContainer permite la interacción con la base de datos (representa a la vez la conexión a la misma), para realizar acciones de inserción, actualización, eliminación, transacciones (Commit y Rollback).

La clase fue diseñada siguiendo el patrón de diseño “Singleton” (Singleton Pattern), y está basada en la definición de [Rasheed2003]. Además se realizaron algunas modificaciones para lograr el comportamiento deseado.

```
public class SystemDBClient
{
        //Atributos privados de la clase
        private static volatile SystemDBClient _instance;
        private IObjectContainer _sysDBClient;


        //Constructor no accesible. Sólo se permite el acceso a los métodos
        //públicos GetInstance, GetDbClient y CloseDbClient
        private SystemDBClient()
        {

        }

        //Método que asegura la existencia de sólo una instancia
        public static SystemDBClient GetInstance()
        {
           if (_instance == null)
           {
              lock (typeof(SystemDBClient))
              {
                 if (_instance == null)
                 {
                    _instance = new SystemDBClient();
                    _instance._sysDBClient = OpenSystemDBClient();
                 }
              }
           }
           return _instance;
        }

        //Método que devuelve el objeto de la base de datos
        //manteniendo la conexión abierta
        public IObjectContainer GetDbClient()
        {
           if (_sysDBClient.Ext().IsClosed())
           {
              _instance._sysDBClient = OpenSystemDBClient();
           }

           return _sysDBClient;
        }
```

```
        //Método que cierra la conexión y establece la instancia en null
        public void CloseDbClient()
        {
            if (!_sysDBClient.Ext().IsClosed())
                _sysDBClient.Close();
            _sysDBClient.Dispose();
            _instance = null;
        }

        //Método privado usado para abrir la conexión a la base de datos
        //del sistema
        private static IObjectContainer OpenSystemDBClient()
        {
            SystemDatabase.SysDBClasses.User admin = _
              SystemDatabase.Common.Utilities.SystemDBUtilities.GetAdminUser();

            return Db4oFactory.OpenClient("localhost", 8731, admin.Login, _
               admin.Password);
        }
    }
```

8.4.2. Pantalla “AddDatabase”

En esta sección se lleva a cabo la creación de una nueva base de datos. El objeto generado contará con sus atributos básicos, pues la definición de sus clases y atributos se realiza mediante el diagramador de clases.

```
public partial class AddDatabase : Form
{
        private SystemDatabase.SysDBClasses.User _user;

        public AddDatabase(SystemDatabase.SysDBClasses.User user)
        {
            InitializeComponent();
            _user = user;
```

```
}

private void AddDatabase_Load(object sender, EventArgs e)
{
    IObjectContainer sysDatabaseClient = _
        SystemDBClient.GetInstance().GetDbClient();

    IObjectSet dataBases = sysDatabaseClient.Get(new DBInfo());

    //Al cargarse la pantalla se busca un valor por defecto para el atributo
    //PortNumber. Este valor corresponde a un puerto disponible según la
    //aplicación.

    int maxPortNumber = 0;

    if (dataBases != null)
    {
        foreach (DBInfo DB in dataBases)
        {
            if (DB.PortNumber>maxPortNumber)
                maxPortNumber = DB.PortNumber;
        }
        if (maxPortNumber == 0)
            maxPortNumber = 8732;
        else
            maxPortNumber++;
    }
    txtPortNumber.Text = maxPortNumber.ToString();
}

//Evento que abre un examinador de carpetas para la selección
//del destino del archivo de la base de datos
private void btnExamine_Click(object sender, EventArgs e)
{
    dDirectoryBrowser.ShowDialog();
    txtDestinationDirectory.Text = dDirectoryBrowser.SelectedPath;
}

//Evento generado al crear la base de datos
private void btnCreate_Click(object sender, EventArgs e)
{
```

```
    //Se instancia un nuevo objeto de base de datos
    DBInfo newOODB = new DBInfo(txtName.Text, _
        Convert.ToInt32(txtPortNumber.Text), txtFileName.Text, _
        @txtDestinationDirectory.Text);

    newOODB.StartUpWindows = chkStartUpWindows.Checked;

    try
    {
        //Se procede a la creación
        CreateDatabase(newOODB);

        MessageBox.Show("New database created successfully");

        this.Dispose();
        this.Close();
    }
    catch
    {
        SystemDBClient.GetInstance().GetDbClient().Rollback();
        MessageBox.Show("Save error; check input data and try again");
    }
}

public void CreateDatabase(DBInfo database)
{
    //Se obtiene la base de datos del sistema
    IObjectContainer sysDatabaseClient = _
        SystemDBClient.GetInstance().GetDbClient();

    //Se asignan los permisos correspondientes
    database.AddUser(_user);
    _user.AddOODatabase(database);


    SystemDatabase.SysDBClasses.User adminUser = new _
        SystemDatabase.SysDBClasses.User();
    adminUser.IsAdmin = true;

    IObjectSet adminUsers = sysDatabaseClient.Get(adminUser);
```

```
            if (adminUsers != null)
            {
                foreach (SystemDatabase.SysDBClasses.User user in adminUsers)
                {
                    if (_user != user)
                    {
                        database.AddUser(user);
                        user.AddOODatabase(database);
                        sysDatabaseClient.Set(user);
                    }
                }
            }

            //Se guarda la base de datos
            sysDatabaseClient.Set(database);

            //Se guarda los cambios al usuario creador de la base de datos
            sysDatabaseClient.Set(_user);

            //Se realiza un commit de la transacción
            //Toda acción en la base de datos es una transacción, sólo basta
            //completarla o deshacerla
            sysDatabaseClient.Commit();

        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            this.Dispose();
            this.Close();
        }
    }
```

8.4.3. Clase “DatabaseServerUtilities”

Esta clase provee las utilidades referentes a los servidores de bases de datos creados.

Las utilidades son las de iniciar un servidor o detener uno en ejecución, para lo cual se realiza una interacción con el servicio “DB4O Server” en la que se le solicita la ejecución de uno de sus comandos definidos.

```
public static class DatabaseServerUtilities
{
        //Método encargado de inicializar un servidor
        public static void StartDatabaseServer(DBInfo database)
        {
            System.ServiceProcess.ServiceController sControl = new _
                System.ServiceProcess.ServiceController("DB4OServer");

            //Se verifica que el servicio está ejecutándose
            if (sControl.Status == _
                System.ServiceProcess.ServiceControllerStatus.Running)
            {
                if (database != null)
                {
                    IObjectContainer sysDatabaseClient = _
                        SystemDBClient.GetInstance().GetDbClient();

                    //Se modifican parámetros destinados a identificar la base de
                    //datos que requiere la inicialización de un servidor
                    database.PendingAction = DBPendingAction.Start;
                    sysDatabaseClient.Set(database);
                    sysDatabaseClient.Commit();

                    sControl.Refresh();

                    //Se realiza la petición de ejecución del comando correspondiente
                    sControl.ExecuteCommand((int)ServiceCommand.StartDBServer);

                    sysDatabaseClient.Ext().Refresh(database, 10);

                    do
                    {
                        //wait to start db server
                    } while (DatabaseClient.GetDatabaseClient(database) == null);
                }
```

```
        else
        {
            throw new Exception("No database found for name '" + _
                  database.Name + "'");
        }
    }
}

//Método encargado de detener un servidor en ejecución
public static void StopDatabaseServer(DBInfo database)
{
    System.ServiceProcess.ServiceController sControl = new _
        System.ServiceProcess.ServiceController("DB4OServer");

    if (sControl.Status == _
         System.ServiceProcess.ServiceControllerStatus.Running)
    {
        if (database != null)
        {
            IObjectContainer sysDatabaseClient = _
                SystemDBClient.GetInstance().GetDbClient();

            //Se modifican parámetros destinados a identificar la base de
            //datos que requiere la detención de su servidor
            database.PendingAction = DBPendingAction.Stop;
            sysDatabaseClient.Set(database);
            sysDatabaseClient.Commit();

            sControl.Refresh();

            //Se realiza la petición de ejecución del comando correspondiente
            sControl.ExecuteCommand((int)ServiceCommand.StopDBServer);

            sysDatabaseClient.Ext().Refresh(database, 10);

            do
            {
                //wait to stop db server
            } while (DatabaseClient.GetDatabaseClient(database) != null);
        }
        else
        {
```

```
                throw new Exception("No database found for name '" + _
                    database.Name + "'");
            }
        }
    }
```

8.4.4. Pantalla “ClassDefinition”

Esta pantalla se despliega dentro del diagramador de clases al momento de realizar un evento “Doble clic” sobre uno de los objetos representativos de una clase.

```
public partial class ClassDefinition : Form
  {
    private Class _newClass;

    public Class NewClass
    {
      get
      {
        return _newClass;
      }
    }

    //Constructor de la clase.
    public ClassDefinition(Class theClass)
    {
      InitializeComponent();
      _newClass = theClass;
    }
```

```
//Evento generado al cargarse la pantalla
private void ClassDefinition_Load(object sender, EventArgs e)
{
    //Se genera el contenido de los tipos disponibles para su asignación a
    //los atributos que se definan

    this.type.DataSource = _
        DB4OManager.Common.Enumerations.DataTypes.GetTypes();

   //Si la clase definida ya existía, se despliegan sus valores
   if (_newClass != null)
   {
      txtClassName.Text = _newClass.Name;
      if (_newClass.Attributes != null)
      {
         foreach (SystemDatabase.SysDBClasses.Attribute attribute _
               in _newClass.Attributes)
         {
            string aName = attribute.Name;
            string aType = attribute.Type.Name;
            bool aRequired = attribute.Rules.Required;
            bool aUnique = attribute.Rules.Unique;
            object aMinVal = attribute.Rules.RangeValue.MinValue;
            object aMaxVal = attribute.Rules.RangeValue.MaxValue;

            dgvAttributes.Rows.Add(aName, aType, aRequired, aUnique,_
                aMinVal, aMaxVal);
         }
      }
   }
}

//Evento generado al solicitar el salvado de los datos
private void btnSave_Click(object sender, EventArgs e)
{
   try
   {
      //Se crea la clase y su definición de atributos incluyendo reglas
      //Los objetos generados se guardan en la base de datos sólo hasta
      //cuando el usuario solocita el guardado del esquema
      _newClass = new Class(txtClassName.Text);
```

```
        foreach (DataGridViewRow row in dgvAttributes.Rows)
        {
            if (row.Index < dgvAttributes.Rows.Count - 1)
            {
                string name = row.Cells["name"].Value.ToString();

                Type attType = Type.GetType("System." + _
                    row.Cells["type"].Value.ToString(), false, true);

                bool isRequired = false;
                if (row.Cells["required"].Value != null)
                    isRequired = (bool)row.Cells["required"].Value;

                bool isUnique = false;
                if (row.Cells["unique"].Value != null)
                    isUnique = (bool)row.Cells["unique"].Value;

                object minValue = (object)row.Cells["minValue"].Value;

                object maxValue = (object)row.Cells["maxValue"].Value;

                SystemDatabase.SysDBClasses.Attribute newAttribute = new _
                    SystemDatabase.SysDBClasses.Attribute(name, attType, _
                    isRequired, isUnique, minValue, maxValue);

                _newClass.AddAttribute(newAttribute);
            }
        }

        this.DialogResult = DialogResult.OK;
        this.Close();
    }
    catch
    {
        this.DialogResult = DialogResult.No;
        MessageBox.Show("Class definition error; check input data _
            and try again");
    }
}

private void btnCancel_Click(object sender, EventArgs e)
{
    this.DialogResult = DialogResult.No;
```

```
            this.Close();
        }

    }
```

8.4.5. Pantalla “CodeGeneration”

Una vez que se tiene un esquema definido para una base de datos, se puede realizar la generación del código fuente C# que representa a cada clase creada. La generación se realizó por medio de CodeDOM (Code Document Object Model), que es una librería de clases que permite la generación de código de forma dinámica.

El código generado va a depender de la manera en que se realizó el diseño de la base de datos. Tomando en cuenta las limitaciones de la aplicación al término del proyecto, se debe considerar que; la herencia se realiza trazando una línea que va desde la clase hija a la clase padre y las asociaciones se generan trazando una línea desde la clase contenedora a la que es contenida.

Otro punto importante es que el código tendiente a realizar el control de la integridad de los objetos se generará comentado para que el usuario revise y pueda mejorar o cambiar código según corresponda. La generación busca ser

una ayuda, lo que no implica que el resultado cumpla totalmente con lo requerido.

```
public partial class CodeGeneration : Form
{
    private DBInfo _database;

    public CodeGeneration(DBInfo database)
    {
        InitializeComponent();
        _database = database;
    }

    //Evento generado al solicitar la creación de los archivos de código
    private void btnGenerate_Click(object sender, EventArgs e)
    {
        try
        {
            //Se realiza el manejo de los posibles errores
            if (_database.Diagram != null)
            {
                //Se realiza la generación de código en la carpeta de destino
                //seleccionada por el usuario
                GenerateCode(txtDestinationFolder.Text);
                this.Dispose();
                this.Close();
            }
            else
            {
                MessageBox.Show("Database diagram is required to generate_
                    the code");
            }
        }
        catch
        {
            MessageBox.Show("Code generation failed; try again or check _
                database schema");
        }
    }
```

```
//Método principal de la generación de código
public void GenerateCode(string destinationFolderPath)
{
   //Por cada clase contenida en el diagrama se crea un archivo
    foreach (Class gClass in _database.DBClasses)
    {

       //Se crea un objeto namespace con el nombre de la base de datos
       CodeNamespace newNamespace = new _
            CodeNamespace(_database.Name);

       //Se incorporan algunos “using” básicos
       newNamespace.Imports.Add(new CodeNamespaceImport("System"));
       newNamespace.Imports.Add(new _
            CodeNamespaceImport("System.Collections"));
       newNamespace.Imports.Add(new _
            CodeNamespaceImport("Db4objects.Db4o"));
       newNamespace.Imports.Add(new _
            CodeNamespaceImport("Db4objects.Db4o.Query"));

       //Se crea la clase
       CodeTypeDeclaration theClass = CreateCSharpClass(gClass, _
            newNamespace.Name);

       //Se incorporan los atributos a la clase
       theClass.Members.AddRange(_
            CreateClassAttributes(gClass.Attributes));

       if (gClass.RelatedClasses != null)
       {
          //En caso de que existan clases relacionadas, se crean los
          //atributos para representar la asociación
          theClass.Members.AddRange(_
               CreateClassAttributes(gClass.RelatedClasses, _
               newNamespace.Name));
       }

       //Se crea el métodos de la clase que implementa el callback
       //de control de inserción de objetos “objectCanNew”
       theClass.Members.Add(CreateCallback(gClass,  _
            newNamespace.Name + "." + theClass.Name));
```

```
            //Se asocia el objeto clase al objeto namespace generado
            newNamespace.Types.Add(theClass);

            //Se crean los archivos de código para el objeto namespace definido
            WriteCode(newNamespace, destinationFolderPath);
        }
    }

    private void btnCancel_Click(object sender, EventArgs e)
    {
        this.Dispose();
        this.Close();
    }

    //Método que crea el objeto Class para la generación de código
    private CodeTypeDeclaration CreateCSharpClass(Class theClass, string _
            namespaceName)
    {
        //Se establecen las propiedades según la clase que contiene el
        //diagrama diseñado
        CodeTypeDeclaration newClass = new CodeTypeDeclaration();
        newClass.IsClass = true;
        newClass.Name = theClass.Name;

        //En caso de una herencia, se define el tipo base (ParentClass)
        if (!(theClass.ParentClass == null))
        {
            newClass.BaseTypes.Add(new _
                CodeTypeReference(namespaceName + "." + _
                theClass.ParentClass.Name));
        }

        //Se definen 2 constructores, uno general y otro según los atributos de
        //la clase.
        CodeConstructor classConstructor = new CodeConstructor();
        classConstructor.Attributes = MemberAttributes.Public;

        newClass.Members.Add(classConstructor);

        classConstructor = new CodeConstructor();
        classConstructor.Attributes = MemberAttributes.Public;
```

```
    foreach (SystemDatabase.SysDBClasses.Attribute cAttribute in _
            theClass.Attributes)
    {
        classConstructor.Parameters.Add(new _
            CodeParameterDeclarationExpression(cAttribute.Type, "_" + _
            cAttribute.Name));

        classConstructor.Statements.Add(new _
            CodeSnippetExpression(cAttribute.Name + " = _" + _
            cAttribute.Name));
    }

    newClass.Members.Add(classConstructor);

    return newClass;
}

//Método que genera un arreglo de objetos de tipo atributo para la
//generación del código de acuerdo a los atributos definidos en la clase
//mediante el diseñador de clases.
private CodeMemberField[] CreateClassAttributes(_
     SystemDatabase.SysDBClasses.Attribute[] attributes)
{
    CodeMemberField[] classAttributes = new _
            CodeMemberField[attributes.Length];
    int c = 0;
    foreach (SystemDatabase.SysDBClasses.Attribute gAttribute in _
            attributes)
    {
        CodeMemberField codeAttribute = new _
            CodeMemberField(gAttribute.Type, gAttribute.Name);

         //Se generan los atributos de tipo público
        codeAttribute.Attributes = MemberAttributes.Public;
        classAttributes[c] = codeAttribute;
        c++;
    }
    return classAttributes;
}
```

```
//Método que genera un arreglo de objetos de tipo atributo para la
//generación del código de acuerdo a las clases relacionadas.
private CodeMemberField[] CreateClassAttributes(_
    Class[] relatedClasses, string namespaceName)
{
   CodeMemberField[] classAttributes = new _
          CodeMemberField[relatedClasses.Length];
   int c = 0;
   foreach (Class rClass in relatedClasses)
   {
      CodeMemberField codeAttribute = new CodeMemberField(new _
          CodeTypeReference(namespaceName + "." + rClass.Name), _
          "related" + rClass.Name);

       //Se generan los atributos de tipo público
      codeAttribute.Attributes = MemberAttributes.Public;
      classAttributes[c] = codeAttribute;
      c++;
   }
   return classAttributes;
}

//Método que genera el método que define el callback “objectOnNew”
//para el control de integridad de objetos.
private CodeMemberMethod CreateCallback(Class theClass, _
    string typeOfQuery)
{
   //Estos métodos siempre reciben un parámetro de tipo IObjectContainer
   //y devuelven un valor booleano
   string parameterName = "container";
   CodeMemberMethod FunctionMember = new CodeMemberMethod();
   FunctionMember.Name = "objectCanNew";
   FunctionMember.ReturnType = new CodeTypeReference(typeof(bool));
   FunctionMember.Parameters.Add(new _
          CodeParameterDeclarationExpression(new _
          CodeTypeReference("IObjectContainer"), parameterName));

   FunctionMember.Attributes = MemberAttributes.Public;
```

```
//En el caso de el método a generar esté en una clase hija
//se define el método como Override, es decir, sobreescribe la
//definición del método onObjectCanNew definido en la clase
//padre.
if (!(theClass.ParentClass == null))
{
   FunctionMember.Attributes = MemberAttributes.Public | _
       MemberAttributes.Override;
   FunctionMember.Statements.Add(new _
       CodeCommentStatement("//Check base values; paste _
       base.objectCanNew code here and change unique query _
       constraint type to " + theClass.Name));
}

//Se genera el código para comprobar valores únicos
CodeCommentStatement commentUniqueValues = new_
    CodeCommentStatement("//Check unique values");

foreach (SystemDatabase.SysDBClasses.Attribute gAttribute in _
       theClass.Attributes)
{
   if (gAttribute.Rules.Unique)
   {
      if (!(FunctionMember.Statements.Contains(_
             commentUniqueValues)))
      {
         FunctionMember.Statements.Add(commentUniqueValues);

         FunctionMember.Statements.Add(new _
             CodeCommentStatement("IQuery uniqueQuery = _
             container.Query();"));

         FunctionMember.Statements.Add(new _
             CodeCommentStatement("uniqueQuery.Constrain(typeof("_
             + typeOfQuery + "));"));
      }


      FunctionMember.Statements.Add(new _
          CodeCommentStatement("uniqueQuery.Descend(\"" +_
           gAttribute.Name + "\").Constrain(" + gAttribute.Name + ");"));
   }
}
```

```
if (FunctionMember.Statements.Contains(commentUniqueValues))
{
   FunctionMember.Statements.Add(new _
       CodeCommentStatement("IObjectSet uniqueResult = _
       uniqueQuery.Execute();"));

   FunctionMember.Statements.Add(new _
       CodeCommentStatement("if (uniqueResult.Count != 0)"));

   FunctionMember.Statements.Add(new _
       CodeCommentStatement("\t return false;"));
}


//Se genera el código para comprobar valores requeridos
//de los tipos de datos que pueden tener valores null
FunctionMember.Statements.Add(new _
       CodeCommentStatement("//Check required values"));
string requiredStatement = "";
foreach (SystemDatabase.SysDBClasses.Attribute gAttribute in _
       theClass.Attributes)
{
   if (gAttribute.Rules.Required)
      if ((gAttribute.Type == Type.GetType("System.Char")) || _
              (gAttribute.Type == Type.GetType("System.DateTime")) || _
              (gAttribute.Type == Type.GetType("System.Object")) || _
              (gAttribute.Type == Type.GetType("System.String")))

         requiredStatement += "||(" + gAttribute.Name + " == null)";

}

if (requiredStatement.Length > 1)
{
   requiredStatement = requiredStatement.Remove(0, 2);

   FunctionMember.Statements.Add(new _
       CodeCommentStatement("if (" + requiredStatement + ")"));

   FunctionMember.Statements.Add(new _
       CodeCommentStatement("\t return false;"));
}
```

```
//Se genera el código para comprobar rangos de valores
FunctionMember.Statements.Add(new _
        CodeCommentStatement("//Check range values"));

string rangeValueStatement = "";

foreach (SystemDatabase.SysDBClasses.Attribute gAttribute in _
        theClass.Attributes)
{
   if (!((gAttribute.Rules.RangeValue.MinValue == null) || _
        (gAttribute.Rules.RangeValue.MaxValue == null)))
   {
      string minValue = _
               gAttribute.Rules.RangeValue.MinValue.ToString();

      string maxValue = _
               gAttribute.Rules.RangeValue.MaxValue.ToString();

      if (gAttribute.Type == Type.GetType("System.String"))
         rangeValueStatement += "||( (" + gAttribute.Name + _
               ".Length < " + minValue + ") || (" + gAttribute.Name + _
               ".Length > " + maxValue + "))";

      else if (!((gAttribute.Type == Type.GetType("System.Boolean")) || _
               (gAttribute.Type == Type.GetType("System.Char")) || _
               (gAttribute.Type == Type.GetType("System.Object"))))

         rangeValueStatement += "||( ((" + gAttribute.Type + ")" + _
               gAttribute.Name + " < " + minValue + ") || ( (" + _
               gAttribute.Type + ")" + gAttribute.Name + " > " + _
               maxValue + "))";
   }
}
if (rangeValueStatement.Length > 1)
{
   rangeValueStatement = rangeValueStatement.Remove(0, 2);

   FunctionMember.Statements.Add(new _
        CodeCommentStatement("if (" + rangeValueStatement + ")"));

   FunctionMember.Statements.Add(new _
        CodeCommentStatement("\t return false;"));
}
```

```
            FunctionMember.Statements.Add(new _
                    CodeMethodReturnStatement(new _
                    CodeSnippetExpression("true")));

            return FunctionMember;

        }

        //Método destinado a crear los archivos en el directorio especificado
        private void WriteCode(CodeNamespace classNamespace, string_
                    destinationFolderPath)
        {
            string fileName = classNamespace.Types[0].Name + ".cs";

            //Se crean objetos de CodeDOM que permiten la generación de código
            CSharpCodeProvider codeProvider = new CSharpCodeProvider();
            ICodeGenerator generator;
            generator = codeProvider.CreateGenerator(_
                    @destinationFolderPath + @"\" + fileName);

            StringBuilder sbuilder = new StringBuilder();
            StringWriter sWriter = new StringWriter(sbuilder);

            //Se crea el archivo
            StreamWriter file = File.CreateText(@destinationFolderPath + @"\" +
fileName);

            //Se genera el código y se almacena en una variable
            generator.GenerateCodeFromNamespace(classNamespace, _
                    sWriter, null);

            //Se escribe el código en el archivo creado
            file.Write(sbuilder.ToString());
            file.Close();

            sWriter.Close();
        }
```

```
    //Evento generado al examinar carpetas para almacenar los archivos de
    //código.
    private void btnExamine_Click(object sender, EventArgs e)
    {
        destinationFolderSelector.ShowDialog();
        txtDestinationFolder.Text = destinationFolderSelector.SelectedPath;
    }
}
```

8.5. Pruebas mediante UnitTest

Siguiendo el plan de pruebas diseñado en la etapa de Elaboración, se generó el código correspondiente a través de una clase llamada “ManagerTest”. Para que la clase y las pruebas sean ejecutadas, deben señalarse ciertas etiquetas que define NUnit para poder reconocer el código como prueba.

8.5.1. Clase “ManagerTest”

```
//Se etiqueta la clase para ser utilizada como Test
[TestFixture]
public class ManagerTest
{

    //Crear nueva base de datos llamada uTestDB
    //La etiqueta Test indica que este método es una prueba de unidad
    [Test]
    public void T01CreateDatabase()
    {
        SystemDatabase.SysDBClasses.User adminUser = _
            SystemDBUtilities.GetUserbyName("admin");

        DB4OManager.Presentation.Controls.AddDatabase createDBWin = _
            new AddDatabase(adminUser);

        DBInfo  newDB = new DBInfo("uTestDB", 8888, "uTestDB.odb", @"c:\");
        newDB.StartUpWindows = false;

        createDBWin.CreateDatabase(newDB);

        DBInfo createdDB = _
            SystemDBUtilities.GetDatabasebyName("uTestDB");
```

```
    //Las cláusulas “Assert” sirven para comprobar resultados esperados
    //En este caso se espera que el objeto createdDB no sea null
    //de lo contrario se genera un error, se detiene el proceso de pruebas
    //y se envía el mensaje descrito.
    Assert.IsNotNull(createdDB, "uTestDB database not created");
}


//Inicializar base de datos uTestDB
[Test]
public void T02InitializeDatabase()
{
    DBInfo db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    //Se comprueba que se encontró la base de datos.
    Assert.IsNotNull(db, "uTestDB database not found");

    DatabaseServerUtilities.StartDatabaseServer(db);

    IObjectContainer dbConection = _
        SystemDatabase.DatabaseClient.GetDatabaseClient(db);

    //Se comprueba que el objeto dbConection no sea null.
    Assert.IsNotNull(dbConection, "uTestDB not initialized");
}

//Probar conexión a uTestDB
[Test]
public void T03DatabaseConectionTest()
{
    DBInfo db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    //Se comprueba que el se encontró la base de datos.
    Assert.IsNotNull(db, "uTestDB database not found");

    IObjectContainer dbConection = _
        SystemDatabase.DatabaseClient.GetDatabaseClient(db);

    //Se comprueba que se estableció conexión a la base de datos.
    Assert.IsNotNull(dbConection, "uTestDB conection failed");
```

```
    IObjectContainer dbConectionNonValid = _
          SystemDatabase.DatabaseClient.GetDatabaseClient(db, new _
          SystemDatabase.SysDBClasses.User("nn", "nn", false));

    //Se comprueba que no se puede establecer la conección con un
    //usuario que no tiene permisos. El objeto dbConectionNonValid
    //debe ser null
    Assert.IsNull(dbConectionNonValid, "uTestDB conection failed; _
          nn user has no rights");
}


//Modificar atributos de base de datos uTestDB
[Test]
public void T04ModifyDatabaseAttributes()
{
    DBInfo dbBefore = SystemDBUtilities.GetDatabasebyName("uTestDB");

    Assert.IsNotNull(dbBefore, "uTestDB database not found");

    //Se almacena el número de puerto que tiene la db antes de los cambios
    int previousPortNumber = dbBefore.PortNumber;

    DB4OManager.Presentation.Controls.DatabaseProperties _
          dbPropertiesWin = new DatabaseProperties(dbBefore);

    //Se simula el guardado por medio del control de pantalla destinado
    //a modificar la base de datos. El puerto se modifica sumándole 1
    dbPropertiesWin.SaveProperties(dbBefore.Path, dbBefore.FileName _
          , dbBefore.PortNumber + 1, dbBefore.StartUpWindows);

    //Se busca la base de datos modificada
    DBInfo dbAfter = SystemDBUtilities.GetDatabasebyName("uTestDB");

    //Se valida que el número de puerto anterior + 1 es igual al actual
    Assert.AreEqual(previousPortNumber + 1, dbAfter.PortNumber, _
          "Modifies to uTestDB database failed");
}
```

```
//Probar conexión a uTestDB modificado
[Test]
public void T05ModifiedDatabaseConectionTest()
{
    DBInfo db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    Assert.IsNotNull(db, "uTestDB database not found");

    IObjectContainer dbConection = _
        SystemDatabase.DatabaseClient.GetDatabaseClient(db);

    //Se valida que se establece conexión a la base de datos modificada
    Assert.IsNotNull(dbConection, "uTestDB modified conection failed");
}

//Crear usuario uTestUser
[Test]
public void T06CreateUser()
{
    SystemDatabase.SysDBClasses.User newUser = new _
        SystemDatabase.SysDBClasses.User("uTestUser", _
        "uTestUser", false);

    //Se simula la creación de un usuario por pantalla
    DB4OManager.Presentation.Controls.AddUser addUserWin = _
        new AddUser();

    addUserWin.CreateUser(newUser);

    SystemDatabase.SysDBClasses.User user = _
         SystemDBUtilities.GetUserbyName("uTestUser");

    //Se valida la creación
    Assert.IsNotNull(user, "uTestUser user not created");
}


//Asignar permiso a userUTest a la base de datos uTestDB
[Test]
public void T07SetUserRights()
{
    DBInfo db = SystemDBUtilities.GetDatabasebyName("uTestDB");
```

```
    Assert.IsNotNull(db, "uTestDB database not found");

    SystemDatabase.SysDBClasses.User user = _
        SystemDBUtilities.GetUserbyName("uTestUser");

    Assert.IsNotNull(user, "uTestUser user not found");

    //Se simula la asignación de acceso a la base de datos por pantalla
    DB4OManager.Presentation.Controls.UserAccessRights arWin = _
        new UserAccessRights(user);

    arWin.SaveDatabaseRights(db, true);
    arWin.SaveUserRights();

    IObjectContainer dbConectionByUser = _
        SystemDatabase.DatabaseClient.GetDatabaseClient(db, user);

    //Se valida el permiso asignado
    Assert.IsNotNull(dbConectionByUser, "uTestDB rights for _
        uTestUser not saved");
}

//Importar diagrama previo a la base de datos uTestDB
[Test]
public void T08ImportDatabaseSchema()
{
    DBInfo db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    Assert.IsNotNull(db, "uTestDB database not found");

    DB4OManager.Common.GUI.DiagramTool.DiagramViewer diagViewer _
        = new B4OManager.Common.GUI.DiagramTool.DiagramViewer(db);

    System.IO.Stream diagramFile = _
        File.OpenRead(@"../TestFiles/uTestDB.uml");

    //Se simula la generación del diagrama de la base de datos
    //importando un archivo previamente generado y se guarda la base de
    //datos con su nuevo diagrama.
    diagViewer.GenerateDiagram(diagramFile);
    diagViewer.Save();

    db = SystemDBUtilities.GetDatabasebyName("uTestDB");
```

```
    //Se valida que el diagrama fue cargado correctamente
    Assert.IsNotNull(db.Diagram, "uTestDB database diagram not loaded");
}

//Generar código de acuerdo al diagrama de uTestDB
[Test]
public void T09CodeGeneration()
{
    DBInfo db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    Assert.IsNotNull(db, "uTestDB database not found");

    //Se simula la generación de código por pantalla
    DB4OManager.Presentation.Controls.CodeGeneration cgWin = _
        new CodeGeneration(db);

    cgWin.GenerateCode(@"../TestFiles");

    //Se valida que los archivos fueron creados
    Assert.IsTrue(File.Exists(@"../TestFiles/Customer.cs"), _
        "Code for Customer not generated");
    Assert.IsTrue(File.Exists(@"../TestFiles/Employee.cs"), _
        "Code for Employee not generated");
    Assert.IsTrue(File.Exists(@"../TestFiles/Person.cs"), _
        "Code for Person not generated");
    Assert.IsTrue(File.Exists(@"../TestFiles/Title.cs"), _
        "Code for Title not generated");
}

//Comprobar código nuevo generado con código previamente generado
[Test]
public void T10CheckGeneratedCode()
{
    System.IO.Stream customerTestFile = _
        File.OpenRead(@"../TestFiles/CustomerTest.cs");
    System.IO.Stream employeeTestFile = _
        File.OpenRead(@"../TestFiles/EmployeeTest.cs");
    System.IO.Stream personTestFile = _
        File.OpenRead(@"../TestFiles/PersonTest.cs");
    System.IO.Stream titleTestFile = _
        File.OpenRead(@"../TestFiles/TitleTest.cs");
```

```
    System.IO.Stream customerGeneratedFile = _
        File.OpenRead(@"../TestFiles/Customer.cs");
    System.IO.Stream employeeGeneratedFile = _
        File.OpenRead(@"../TestFiles/Employee.cs");
    System.IO.Stream personGeneratedFile = _
        File.OpenRead(@"../TestFiles/Person.cs");
    System.IO.Stream titleGeneratedFile = _
        File.OpenRead(@"../TestFiles/Title.cs");

    //Se valida que el contenido de los archivos generados previamente
    //es igual al que se generó en la aplicación
    Assert.AreEqual(customerTestFile.ToString(), _
    customerGeneratedFile.ToString(), "Bad code generation for customer");

    Assert.AreEqual(employeeTestFile.ToString(), _
   employeeGeneratedFile.ToString(), "Bad code generation for employee");

    Assert.AreEqual(personTestFile.ToString(), _
    personGeneratedFile.ToString(), "Bad code generation for person");

    Assert.AreEqual(titleTestFile.ToString(), _
    titleGeneratedFile.ToString(), _"Bad code generation for title");

    customerTestFile.Close();
    customerGeneratedFile.Close();
    employeeTestFile.Close();
    employeeGeneratedFile.Close();
    personTestFile.Close();
    personGeneratedFile.Close();
    titleTestFile.Close();
    titleGeneratedFile.Close();
}


//Defragmentar base de datos uTestDB
[Test]
public void T11DatabaseDefragment()
{
    DBInfo db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    Assert.IsNotNull(db, "uTestDB database not found");
```

```
    //Se simula la defragmentación realizada por pantalla
    DB4OManager.Presentation.Controls.DefragmentDatabase defragWin=_
        new DefragmentDatabase(db);

    defragWin.PerformDefragment();
    db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    //Se valida que se creó el archivo de respaldo resultante de la
    //defragmentación, que indica que se realizó correctamente.
    Assert.IsTrue(File.Exists(db.EntirePath + @".defrag.backup"), _
        "Defragment file backup not generated");
}

//Respaldar base de datos uTestDB
[Test]
public void T12DatabaseBackup()
{
    DBInfo db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    Assert.IsNotNull(db, "uTestDB database not found");

    //Se simula el respaldo por pantalla
    DB4OManager.Presentation.Controls.BackUpDatabase backUpWin=_
        new BackUpDatabase(db);

    backUpWin.PerformBackup(db.EntirePath + ".backup", false);

    db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    //Se valida que se creó el respaldo correctamente
    Assert.IsTrue(File.Exists(db.EntirePath + ".backup"), _
        "Backup file not generated");
}

//Eliminar usuario uTestUser
[Test]
public void T13RemoveUser()
{
    DBInfo db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    Assert.IsNotNull(db, "uTestDB database not found");
```

```
    SystemDatabase.SysDBClasses.User user = _
        SystemDBUtilities.GetUserbyName("uTestUser");

    Assert.IsNotNull(user, "uTestUser user not found");

    //Se simula el eliminado por pantalla de usuarios
    //Primero se eliminan los permisos y luego el usuario
    DB4OManager.Presentation.Controls.UserAccessRights arWin = _
        new UserAccessRights(user);

    arWin.SaveDatabaseRights(db, false);
    arWin.SaveUserRights();

    SystemDBUtilities.DeleteUser(_
        SystemDBUtilities.GetUserbyName("uTestUser"));

    SystemDatabase.SysDBClasses.User deletedUser = _
        SystemDBUtilities.GetUserbyName("uTestUser");

    //Se valida que el usuario ya no existe
    Assert.IsNull(deletedUser, "uTestUser user not deleted");

}


//Eliminar base de datos uTestDB
[Test]
public void T14RemoveDatabase()
{
    DBInfo db = SystemDBUtilities.GetDatabasebyName("uTestDB");

    Assert.IsNotNull(db, "uTestDB database not found");

    //Se simula la eliminación  por pantalla
    //Se detiene el servidor de la base de datos si está en ejecución
    //y luego se elimina.

    if (db.Status == DBStatus.OnLine)
        DatabaseServerUtilities.StopDatabaseServer(db);
```

```
            SystemDBUtilities.DeleteDatabaseServer(_
                SystemDBUtilities.GetDatabasebyName(db.Name));

            DBInfo deletedDB = _
                SystemDBUtilities.GetDatabasebyName("uTestDB");

            //Se valida que la base de datos no exista
            Assert.IsNull(deletedDB, "uTestDB database not deleted");
        }
    }
```

8.5.2. Resultados de las pruebas de unidad

En la figura 15 se detallan los resultados obtenidos de las pruebas de unidad. Para la ejecución de las pruebas se debe iniciar la interfaz de pruebas de NUnit, llamado NUnit-GUI y luego abrir el archivo “dll” generado en el proyecto de pruebas.

Una vez abierto el archivo se despliegan los UnitTest contenidos en la librería “dll”, dándose la posibilidad de ejecutar los seleccionados por el usuario.

Los nombres de los UnitTest comienzan con TXX, donde XX es el número del test. Esto debido a que la ejecución se realiza por orden alfabético y en este caso el orden de ejecución es importante para los resultados esperados.

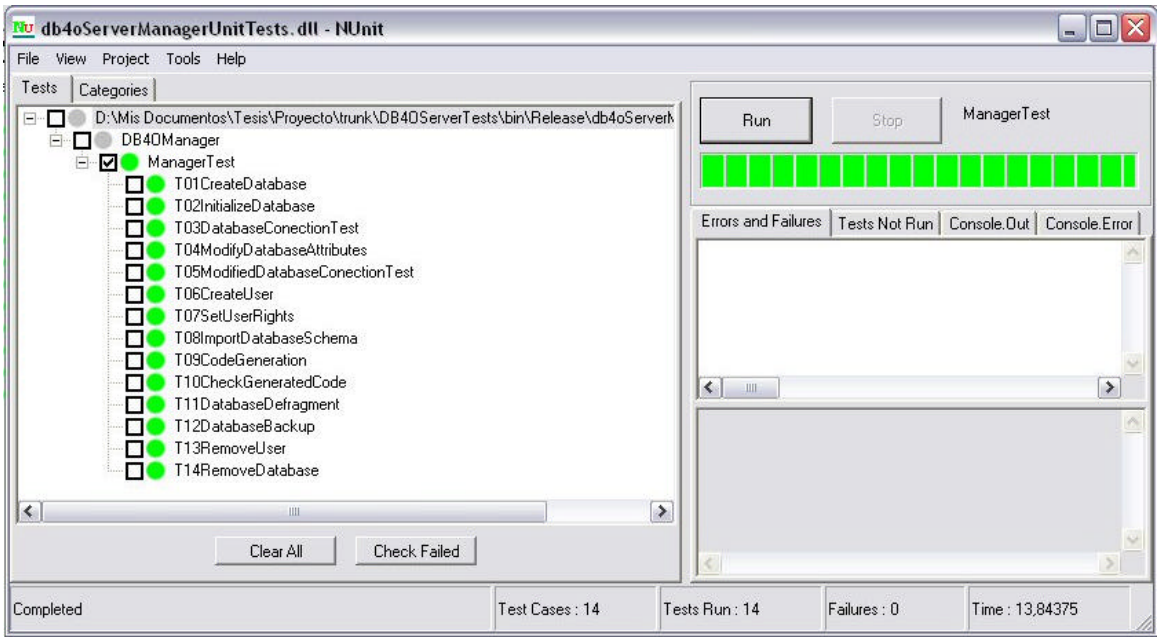

Figura 15. Resultados de UnitTest.

En la figura se aprecian los test ejecutados. El círculo verde que se ubica al lado del nombre del test indica que se completó con éxito. En caso de fallo el color se vuelve rojo y en caso de no ejecutarse el test, se vuelve amarillo.

La barra de progreso en verde asegura que la totalidad de tests se ejecutó satisfactoriamente.

8.6. Pruebas conexión remota

Esta prueba tiene la finalidad de comprobar que los servidores para las bases de datos están disponibles a través de una red de computadores.

En la figura 16 se muestra el esquema de red utilizado.

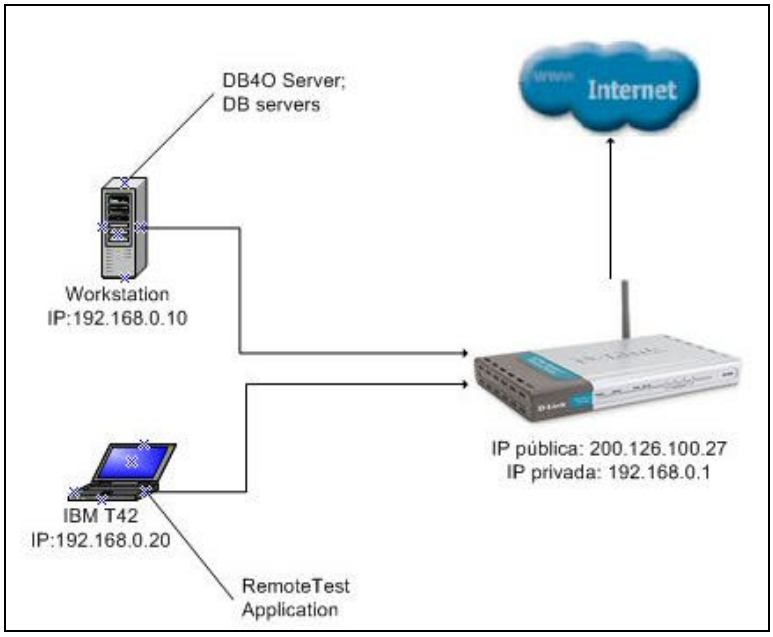

Figura 16. Esquema de red

El “Router” se configuró para que redirija las peticiones desde el puerto público 8889 hacia la dirección IP interna 192.168.0.10 puerto 8889.

Ante este esquema, se utilizó el mismo código generado por las pruebas de unidad, para generar una aplicación de prueba consistente en el siguiente código:

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

     private void button1_Click(object sender, EventArgs e)
     {
        dataGridView1.Rows.Clear();
        Db4objects.Db4o.IObjectContainer dbClient = _
              Db4objects.Db4o.Db4oFactory.OpenClient("200.126.100.27", _
              8889, "admin", "admin");

        uTestDB.Customer newCustomer = new _
              uTestDB.Customer(textBox3.Text, textBox4.Text);

        newCustomer.Name = textBox1.Text;
        newCustomer.Age = Convert.ToInt32(textBox2.Text);
        dbClient.Set(newCustomer);
        dbClient.Commit();

        dbClient.Close();

        dbClient=Db4objects.Db4o.Db4oFactory.OpenClient("200.126.100.27",_
              8889, "admin", "admin");

        Db4objects.Db4o.IObjectSet customers = dbClient.Get(new _
              uTestDB.Customer());
```

```
        foreach (uTestDB.Customer cust in customers)
        {
            this.dataGridView1.Rows.Add(cust.Name, cust.Age, cust.Phone, _
                cust.Address);
        }

        dataGridView1.Refresh();
        dbClient.Close();

    }
}
```

La aplicación se ejecutó en 2 ocasiones, en la segunda oportunidad se modificó la IP para la conexión a “192.168.0.10” obteniendo resultados satisfactorios en ambos casos.

En la figura 17 que se muestra a continuación, se aprecia que la conexión se estableció, pues se produjo la inserción de un objeto en la base de datos y luego se ejecutó una consulta.

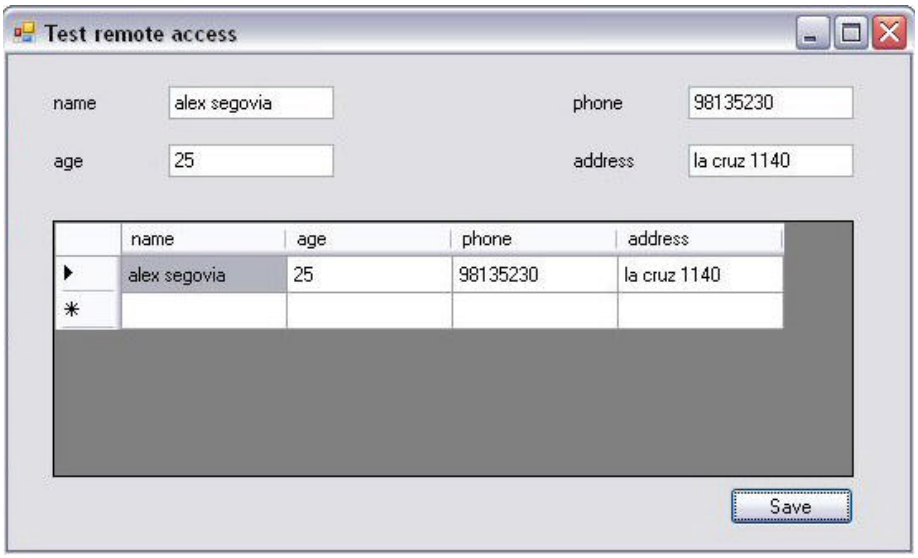

Figura 17. Resultado prueba de conexión remota.

9. Transición

La última etapa del desarrollo de la aplicación se centró principalmente en la preparación del proyecto para ser publicado y difundido.

9.1. Publicación de la Aplicación

Para la publicación se decidió solicitar la creación de un “Espacio de Proyecto” (Project Spaces) en el sitio “http://www.db4o.com/community/ “, lo que significó establecer un nombre apropiado a la herramienta; “db4o Server and Manager”.

Es así como se encuentra disponible una descripción del proyecto y los archivos, tanto ejecutables como fuentes en el espacio “http://developer.db4o.com/ProjectSpaces/view.aspx/Db4o_Server_and_Manager”.

9.2. Difusión del proyecto

En cuanto a la difusión, se crearon los llamados “Post” en el foro de la comunidad Española para que los usuarios accedan al espacio y puedan probar la herramienta. Más adelante se realizará difusión destinada a captar desarrolladores interesados en seguir desarrollando y mejorando el producto.

10. Conclusiones

El surgimiento de db4o ha traído consigo una notoria reducción de esfuerzo y tiempo en el desarrollo de software, lo cual anima a buscar complementos para este SGBDOO. Tal como la herramienta producida en el seminario, que constituye un paso importante para generar un acercamiento a lo que se conoce mayoritariamente, que son los administradores de bases de datos relacionales, además de ser un punto de referencia para continuar mejorando sus prestaciones.

Al observar las funcionalidades de esta aplicación y su distribución se puede apreciar el cumplimiento de gran parte de los objetivos planteados, pero se hace necesario describir la validación de objetivos que son independientes de la consecución del producto.

En primer lugar, el tema de la automatización de tareas en el área del desarrollo de software, es uno de los principales elementos que ayudan a reducir el esfuerzo que demanda la realización de una tarea y así también el tiempo que lleva ejecutarla. En el caso del producto elaborado en el seminario, la generación de código fuente representa el punto más importante para validar que la reducción de tiempo en el desarrollo de software es posible gracias a

esta herramienta, puesto que se facilita la creación de código fuente representativo de una base de datos OO.

Y en segundo, el conjunto que se genera entre el diseñador del modelo de clases de una base de datos OO y la generación de su código es una característica que predomina en herramientas comerciales, por lo que al distribuir esta utilidad bajo licencia pública se evita la adquisición de otro producto similar para el desarrollo de un software, favoreciendo la reducción de sus costos.

Por otra parte, en referencia al proceso de desarrollo del producto lo más importante, sin contar la elección de AUP como metodología, es la aplicación de la técnica “TDD” (Test Driven Development). Esta ayudará a que ante eventuales cambios futuros se pueda comprobar, rápidamente mediante las pruebas de unidad, si éstos no perjudicaron el funcionamiento normal de la aplicación, lo que se traduce en una mayor productividad y calidad en el software.

La recomendación para continuar con el desarrollo de la herramienta es seguir 2 aspectos importantes; uno es mejorar el diseñador de clases y la generación del código y el otro es otorgar mayor manejo de las bases de datos creadas, pues db4o cuenta con un sinnúmero de opciones para configurar un servidor.

11. Bibliografía

[Ambler2003] Ambler, Scott W. Agile DatabaseTechniques: Effective Strategies for the Agile Software Developer. Wiley Publishing, Inc. 2003

[Ambler2005] Ambler, Scott W. The Agile Unified Process (AUP). Disponible en http://www.ambysoft.com/unifiedprocess/agileUP.html, Septiembre 2005.

[Atkinson1989] Atkinson, Malcom – De Witt, David – Maier, David – Bancilhon, François – Dittrich, Klaus – Zdonik, Stanley. The Object-Oriented Database System Manifesto. Disponible en http://www.cs.brown.edu/courses/cs295-11/oo-manifesto.pdf#search=%22The%20Object-Oriented%20Database%20System%20Manifesto%20pdf%22, de 1989.

[Edlich2006] Edlich, Stefan – Hörning, Henrik – Hörning, Reidar – Paterson, Jim. The Definitive Guide to db4o. Apress. Junio 2006.

[Evaltech2000] Evaltech. Ensuring Data Integrity. Disponible en http://www.evaltech.com/wpapers/ensuringdataintegrity.htm, del 2000.

[Marcos2000] Marcos, Esperanza. Bases de Datos orientadas a Objetos. Disponible en http://kybele.escet.urjc.es/documentos/BD/T2-BDOO.pdf.

[Kurniawan2003] Kurniawan, Budi. Real World .NET Applications. Apress. 2003.

[Rasheed2003] Rasheed, Faraz. Implementing Design Patterns in C# - Singleton Pattern. Disponible en http://www.c-sharpcorner.com/UploadFile/faraz.rasheed/SingletonPattern12052005063955AM/SingletonPattern.aspx, Enero del 2003.

[Shankar2001] Shankar, Hari. Reflection in .NET. Disponible en http://www.c-sharpcorner.com/UploadFile/harishankar2005/Reflectionin.NET12032005045926AM/Reflectionin.NET.aspx, Febrero 2001.

[Stonebraker1990] Stonebraker, M. – Lindsay, B. – Gray, J. – Carey, M. – Brodie, M. – Bernstein, P - Beech, D. Third-Generation Database System Manifesto. Disponible en http://www.cl.cam.ac.uk/Teaching/2003/DBaseThy/or-manifesto.pdf#search=%22THIRD-GENERATION%20DATABASE%20SYSTEM%20MANIFESTO%20pdf%22, Septiembre 1990.

12. Anexos

A. Anexo Digital CD