Proyecto de Sistemas Informáticos

Curso Académico 2008 / 2009

Facultad de Informática

Universidad Complutense de Madrid

DBCASE

"Una herramienta para el diseño de Bases de Datos."

Autores *Rodrigo Denís Cepeda Mateos*

Cristina Marco De Francisco

Tello Serrano Gordillo

Profesor director *Fernando Sáenz Pérez*

Agradecimientos.

A Fernando por su dedicación, y sacar todas las semanas tiempo para revisar nuestro trabajo.

A nuestros padres, hermanos, amigos y todos aquellos que habéis hecho posible que este día llegue, gracias por creer siempre en nosotros, por no dejar que nos desanimáramos ante las dificultades, por vuestra ayuda, comprensión y cariño.

Queremos mencionar de forma especial a: Lourdes, Héctor, Valle, Adri, Adolfo, Vicky, Diego, Jose, Pascu, porque sin vosotros no habríamos llegado hasta aquí.

1. Resumen.

Español

DBCASE es una herramienta que permite diseñar bases de datos relacionales mediante una interfaz gráfica. El objetivo de la misma es facilitar la creación de las bases de datos mediante los diagramas entidad-relación, permitiendo comprobar posteriormente su corrección y generar el código asociado a las tablas creadas.

La herramienta está orientada principalmente al entorno académico, aunque también es útil en otros ámbitos. Tiene una implementación multiplataforma lo que permite utilizarla tanto en sistemas Windows como en distribuciones GNU/Linux (Debian y Ubuntu), y MAC OS X.

La interfaz gráfica se ha diseñado de manera que siga los estándares utilizados en los diagramas entidad-relación de los principales libros de la bibliografía de diseño de bases de datos, de forma que la notación utilizada resulte familiar al usuario.

La aplicación, además de permitir diseñar y comprobar la corrección de un esquema, realiza la tarea de generar y ejecutar un script en lenguaje SQL adaptado a tres gestores de bases de datos distintos: MySQL, ORACLE *y* ACCESS. Dicho proceso es automático y simplifica tanto el tiempo como la complejidad de un diseño teórico llevado a la práctica.

La finalidad principal de la herramienta es facilitar la tarea tanto a alumnos que estén aprendiendo la teoría de bases de datos, como a sus profesores y profesionales, facilitando el proceso de diseño de bases de datos en sus tres fases principales y obtener automáticamente el código asociado.

English

DBCASE is a tool for graphical design of relational databases. Its purpose is making the process of designing databases easier, using the entity-relationship diagram, checking its correctness and creating the SQL code associated to the tables.

This tool is targeted to academic purposes, but it can also be useful in other environments. It has a multi-platform implementation, so it can used in Windows systems, GNU/Linux distributions (already tested in Debian and Ubuntu), and MAC OS X.

The graphical user interface has been designed following the standards of entity-relationship diagrams described in the most commonly used database design literature. This makes the used notation more user-friendly.

Besides making possible to design and test a scheme, the application performs the task of generating and executing a SQL script adapted to three different data base management systems: MySQL, Oracle, and ACCESS. This process is automatic and simplifies both time and complexity of a theoretical design implemented in practice.

The main purpose of the tool is to help students of database theory in the design of databases, and also to help both teachers and professionals to simplify the database design process in its three main stages and automatically obtaining the physical schema as SQL code.

2. Índice.

3. Objetivos del proyecto.

Nuestro trabajo comienza a partir del proyecto DBDT (DataBase Design Tool*)* realizado durante el curso 2007/2008, por Alberto Milán, Miguel Martínez y Fco. Javier Cáceres. Este proyecto fue dirigido por Yolanda García.

Nuestra primera tarea fue acercarnos a la herramienta como usuarios y buscar todas las posibles mejoras que ampliasen o facilitasen su uso.

Uno de los principales objetivos marcados era extender al máximo las posibilidades que ofrece el diseño del Modelo Entidad-Relación, añadiendo la posibilidad de establecer múltiples participaciones de una entidad en una relación a través de roles o ofreciendo la posibilidad de establecer dominios e incluir restricciones como la de unicidad y la de elemento no nulo.

También era un requisito fundamental ampliar la herramienta para que fuese compatible con los gestores de bases de datos más comunes.

Otro de los objetivos era la globalización de la herramienta, antes sólo disponible en castellano, para ampliar el conjunto de posibles usuarios de la aplicación.

Todas estas mejoras debían realizarse sin perder la facilidad de uso. Con el objetivo de obtener un resultado intuitivo y manejable cambiamos todo lo relacionado con la gestión de proyectos. En DBDT cada proyecto consistía en un directorio con varios ficheros, ahora toda la información se guarda en un solo fichero XML que el usuario nombra y gestiona de manera equivalente a la de cualquier editor de textos.

También nos hemos concentrado en facilitar el uso haciendo ventanas sencillas e intuitivas, añadiendo múltiples eventos de teclado y evitando las tareas que ralentizaban el uso de la herramienta. Ejemplos de novedades que agilizan el manejo de la aplicación son el borrado de múltiples nodos del diseño o la colocación ordenada de los atributos en su creación.

4. Especificación de requisitos.

4.1. Introducción.

La descripción de los servicios que ofrece DBCASE y las restricciones de los mismos conforman los requisitos del sistema. Mediante el proceso de ingeniería de requisitos hemos identificado los que tiene nuestra herramienta.

Inicialmente, establecimos los requisitos de usuario y posteriormente los formalizamos como requisitos funcionales:

- Los requisitos de usuario son frases en lenguaje natural que describen lo que debe permitir hacer nuestro sistema y bajo qué condiciones se tienen que producir.
- Los requisitos funcionales determinan los servicios del sistema de forma unívoca y establecen las restricciones de los mismos con todo detalle.

El proceso de Ingeniería de Requisitos es un proceso largo y tedioso: el objetivo principal es determinar qué hay que hacer para cada uno de los requisitos del sistema, sin decir el cómo hay que hacerlo.

Su utilidad se basa en reducir el esfuerzo a la hora de la implementación, proporcionar una guía para la validación y la verificación del sistema y servir como base para mejoras posteriores. Se trata de una especie de contrato en el que se especifica de forma unívoca el comportamiento deseado para cada uno de los requisitos funcionales de nuestro sistema: todo lo que no esté especificado, no se implementa.

Aunque este proceso es muy estricto en cuanto las formas, a la hora de implementar nuestro sistema hemos realizado pequeñas modificaciones en la especificación (olvidos, pequeños errores...) que a grandes rasgos no influyen en el comportamiento inicial deseado.

La Especificación de Requisitos Software (SRS - Software Requirements Specification) es el documento que formaliza todos los requisitos funcionales de un sistema.

En nuestro caso, hemos creado una SRS reducida: seguimos el estándar IEEE Std. 830-1998 [Std_830], pero solamente nos hemos centrado en el apartado 3.2 correspondiente a los Requisitos Funcionales Específicos, ya que es la parte que realmente nos ha resultado útil.

Los requisitos funcionales de DBCASE están divididos en cuatro módulos:

1. Módulo Entidades.
2. Módulo Atributos
3. Módulo Relaciones.
4. Módulo Sistema.

Cada módulo agrupa los requisitos relacionados con cada uno de los objetos principales de nuestra aplicación.

En secciones posteriores especificaremos todos y cada uno de los requisitos de nuestro sistema exhaustivamente. Además, con fines ilustrativos, mostraremos el comportamiento deseado para algunos de ellos mediante diagramas de secuencia.

4.2. Requisitos Software específicos.

Como ya se ha dicho, utilizamos el estándar IEEE Std. 830-1998 para la especificación de los requisitos funcionales específicos de nuestro sistema. De acuerdo a esto, tenemos las siguientes especificaciones para cada uno de los módulos:

4.2.1. Módulo Entidades.

El diagrama de casos de uso para las entidades es el siguiente:

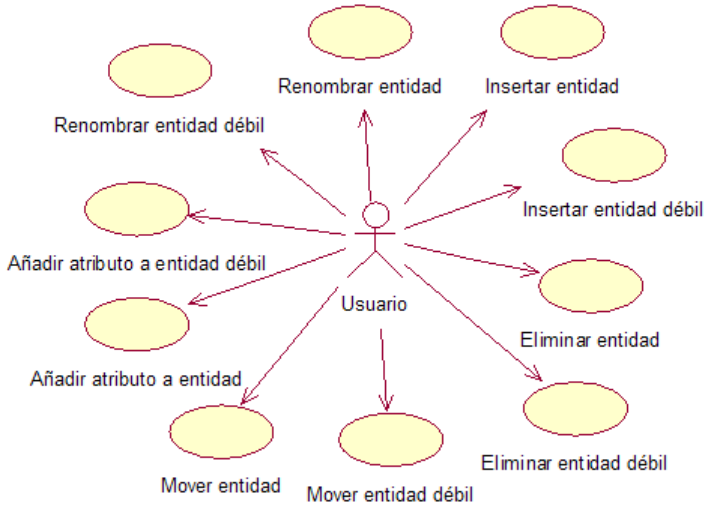


Figura 4. 1

Los requisitos funcionales correspondientes al módulo de entidades son los siguientes:

Función	Insertar una entidad.
Descripción	Añadir una nueva entidad al proyecto en curso.
Entrada	TranferEntidad que contiene el nombre para la nueva entidad que se quiere añadir y la posición en el panel donde se ha pinchado.
Salida	TransferEntidad que contiene el nombre de la nueva entidad, la posición en el panel y el identificador interno que le asigna el sistema.
Origen	GUI_InsertarEntidad.
Destino	Sistema.
Necesita	DAOEntidades.
Acción	Crea una nueva entidad en el proyecto en curso con el nombre que desee el usuario.
Precondición	El nombre para la nueva entidad no puede ser vacío. No puede existir ninguna otra entidad, ni relación, ni atributo en el proyecto en curso cuyo nombre coincida con el que se desea asignar a la nueva entidad.

Postcondición	En caso de éxito, información sobre la inserción de la nueva entidad. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Insertar una entidad débil.
Descripción	Añadir una nueva entidad débil al proyecto en curso.
Entrada	TranferEntidad que contiene el nombre para la nueva entidad débil que se quiere añadir y la posición en el panel donde se ha pinchado. TransferRelacion que contiene la posición y el nombre de la relación débil que se creará para asociar la entidad débil con una entidad fuerte.
Salida	TransferEntidad que contiene el nombre de la nueva entidad débil, la posición en el panel y el identificador interno que le asigna el sistema. TransferRelacion que contiene el nombre de la nueva relación débil, la posición en el panel y el identificador interno que le asigna el sistema.
Origen	GUI_InsertarEntidad.
Destino	Sistema.
Necesita	DAOEntidades, DAORelaciones.
Acción	Crea en el proyecto en curso una nueva entidad débil asociada a una entidad fuerte (ya existente) a través de una nueva relación débil; con los nombres que desee el usuario.
Precondición	Ni el nombre para la nueva entidad débil, ni el de la relación débil pueden ser vacíos. No puede existir ninguna otra entidad, ni relación, ni atributo en el proyecto en curso cuyo nombre coincida con el que se desea asignar a la nueva entidad débil, ni a la relación débil. Debe existir una entidad fuerte a la que poder asociar la entidad débil.
Postcondición	En caso de éxito, información sobre la inserción de la nueva entidad débil, y la nueva relación débil. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Se crea una relación débil.

Función	Renombrar una entidad/entidad débil.
Descripción	Cambiar el nombre a una entidad/entidad débil existente en el proyecto en curso.
Entrada	Vector con dos elementos: El primero de ellos es un TransferEntidad que contiene toda la información de la entidad/entidad débil que se quiere renombrar. El segundo es una cadena con el nuevo nombre que se pretende asignar a la entidad/entidad débil.
Salida	Vector con tres elementos: El primero es un TransferEntidad con el nombre de la entidad/entidad débil cambiado El segundo es una cadena con el nuevo nombre que se ha asignado El tercero es una cadena con el antiguo nombre que tenía la entidad/entidad débil.
Origen	GUI_RenombrarEntidad.
Destino	Sistema.
Necesita	DAOEntidades.
Acción	Cambia el nombre a la entidad/entidad débil seleccionada por el usuario en el proyecto en curso con el nuevo nombre que desee el usuario.
Precondición	El nuevo nombre que se desea asignar a la entidad/entidad débil no puede ser vacío. No puede existir ninguna otra entidad/entidad débil, ni atributo, ni relación en el proyecto en curso cuyo nombre coincida con el que se desea asignar a la entidad a renombrar.
Postcondición	En caso de éxito, información sobre el renombramiento de la entidad/entidad débil. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Añadir un atributo a una entidad/entidad débil
Descripción	Añade un nuevo atributo a una entidad/entidad débil existente en el proyecto en curso.
Entrada	Vector de dos o tres elementos: El primero de ellos es un TransferEntidad que contiene toda la información de la entidad/entidad débil a la que se desea añadir el nuevo atributo. El segundo es un TransferAtributo que contiene toda la información introducida por el usuario para ese atributo (nombre, carácter compuesto, carácter multivalorado y dominio). Si el vector tiene un tercer elemento éste es el tamaño del dominio del atributo para aquellos dominios en los que es necesario especificar su tamaño.
Salida	Vector de dos elementos: El primero de ellos es el TransferEntidad de entrada, con el identificador del nuevo atributo añadido a su lista de atributos. El segundo de ellos es un TransferAtributo igual que el de entrada al que el sistema le ha asignado un identificador único.
Origen	GUI_AnadirAtributoEntidad
Destino	Sistema.
Necesita	DAOEntidades, DAOAtributos.
Acción	Añade a la entidad/entidad débil seleccionada un nuevo atributo con las características que desee para éste el usuario.
Precondición	El nombre que se proporcione para el nuevo atributo no puede ser vacío. Si el dominio del nuevo atributo exige un tamaño, éste debe tener el formato adecuado, esto es, debe ser un valor entero positivo.
Postcondición	En caso de éxito, informar sobre la inserción del nuevo atributo y sobre la modificación de la entidad/entidad débil a la que se le ha insertado. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Eliminar una entidad.
Descripción	Elimina la entidad seleccionada por el usuario del proyecto en curso.
Entrada	TransferEntidad con toda la información de la entidad que se desea eliminar del sistema.
Salida	Vector de dos elementos: El primero es un TransferEntidad con la entidad que ha sido eliminada. El segundo es un Vector de TransferRelacion: cada uno de estos transfers se corresponden con una relación existente en el sistema que ha sido modificada por la eliminación de la entidad anterior.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAOEntidades, DAOAtributos y DAORelaciones
Acción	Elimina la entidad seleccionada del sistema. Si la entidad interviene en alguna relación, quita las referencias a la entidad en dichas relaciones.
Precondición	Ninguna
Postcondición	En caso de éxito, informar sobre la eliminación de la entidad y sobre la modificación de las relaciones en las que intervenía. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Si la entidad a eliminar tiene atributos también hay que eliminarlos del sistema.

Función	Eliminar una entidad débil.
Descripción	Elimina la entidad débil seleccionada por el usuario del proyecto en curso.
Entrada	TransferEntidad con toda la información de la entidad que se desea eliminar del sistema. TransferRelacion que contiene la relación débil asociada que también desaparecerá del sistema.
Salida	Vector de dos elementos: El primero es un TransferEntidad con la entidad que ha sido eliminada. El segundo es un Vector de TransferRelacion: cada uno de estos transfers se corresponden con una relación existente en el sistema que ha sido modificada por la eliminación de la entidad anterior, entre ellas la relación

	débil que será también eliminada.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAOEntidades, DAOAtributos y DAORelaciones
Acción	Elimina la entidad seleccionada del sistema. Si la entidad interviene en alguna relación, quita las referencias a la entidad en dichas relaciones.
Precondición	Ninguna
Postcondición	En caso de éxito, informar sobre la eliminación de la entidad y relación débil y sobre la modificación de las relaciones en las que intervenía. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Si la entidad débil a eliminar tiene atributos también serán eliminados del sistema, así como la relación débil a la que está asociada.

Función	Mover la posición de una entidad/entidad débil
Descripción	Mueve la entidad/entidad débil seleccionada de una posición a otra en el diagrama E/R.
Entrada	TransferEntidad con toda la información de la entidad/entidad débil a la que se desea cambiar su posición en el diagrama. La nueva posición ya está establecida en el TransferEntidad.
Salida	TransferEntidad a la que se le ha modificado su atributo posición.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAOEntidades
Acción	Cambia la entidad/entidad débil seleccionada a la nueva posición.
Precondición	Ninguna
Postcondición	En caso de éxito, informar sobre el cambio de posición de la entidad/entidad débil. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

4.2.2. Módulo Atributos.

El diagrama de casos de uso para los atributos es el siguiente:

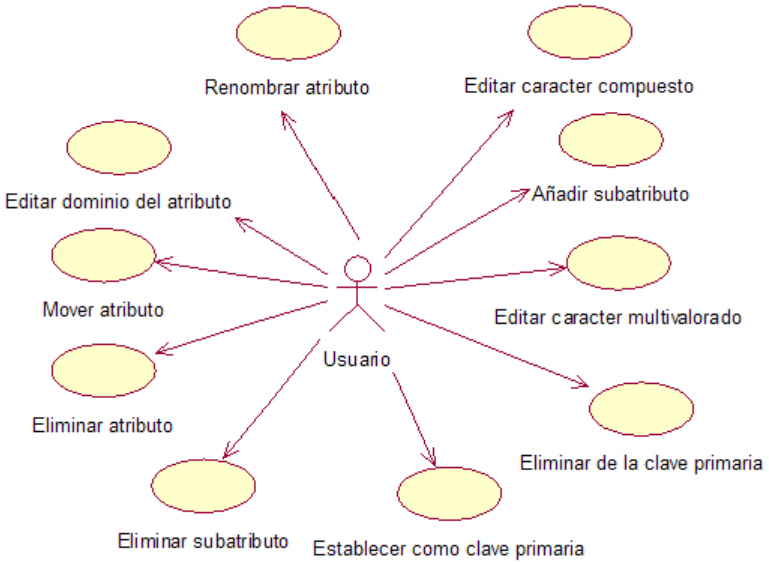


Figura 4. 2

Los requisitos funcionales correspondientes al módulo de atributos son los siguientes:

Función	Renombrar un atributo
Descripción	Cambiar el nombre a un atributo
Entrada	Vector con dos elementos: El primero de ellos es un TransferAtributo que contiene toda la información del atributo que se quiere renombrar. El segundo es una cadena con el nuevo nombre que se quiere asignar al atributo.
Salida	Vector con dos elementos: El primero es el TransferAtributo de entrada con el nuevo nombre ya asignado. El segundo es una cadena con el antiguo nombre que tenía el atributo.
Origen	GUI_RenombrarAtributo.
Destino	Sistema
Necesita	DAOAtributos.
Acción	Cambia el nombre del atributo seleccionado por el nuevo nombre que desee el usuario.
Precondición	El nuevo nombre que se desee asignar al atributo no puede ser vacío.
Postcondición	En caso de éxito, información sobre el renombrado del atributo.

	Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Editar el dominio de un atributo
Descripción	Cambiar el dominio actual de un atributo.
Entrada	Vector con dos o tres elementos: El primero de ellos es un TransferAtributo que contiene toda la información del atributo al que se quiere editar el dominio. El segundo de ellos es una cadena con el nuevo dominio del atributo. Si tiene un tercer elemento, éste es el tamaño del dominio para aquellos dominios en los que es necesario especificar su tamaño.
Salida	TransferAtributo con el nuevo dominio asignado.
Origen	GUI_EditarDominioAtributo.
Destino	Sistema.
Necesita	DAOAtributos.
Acción	Cambia el dominio al atributo seleccionado por el nuevo dominio elegido.
Precondición	Si el dominio del nuevo atributo exige un tamaño, éste debe tener el formato adecuado, esto es, debe ser un valor entero positivo.
Postcondición	En caso de éxito, información sobre el cambio en el dominio del atributo. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Editar el carácter compuesto de un atributo.
Descripción	Cambia el carácter compuesto de un atributo.
Entrada	TransferAtributo que contiene toda la información del atributo al que se quiere cambiar el carácter de compuesto.
Salida	TransferAtributo con el carácter de compuesto modificado.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAOAtributos.

Acción	Cambia el carácter de compuesto del atributo: Si se trata de un atributo simple, se pone como compuesto y su dominio se pone a nulo. Si es un atributo compuesto, se pone como simple.
Precondición	Ninguna.
Postcondición	En caso de éxito, información sobre el cambio en el carácter de compuesto del atributo. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Si se trata de cambiar un atributo compuesto a simple y el atributo tiene subatributos, éstos se eliminarán también del sistema.

Función	Añadir subatributo.
Descripción	Añade un nuevo subatributo a un atributo.
Entrada	Vector de 2 o 3 elementos: El primero de ellos es un TransferAtributo con toda la información del atributo (padre) al que se le va a añadir un nuevo subatributo. El segundo es un TransferAtributo (hijo) con los datos introducidos por el usuario (nombre, carácter compuesto, carácter multivalorado y dominio). Es el nuevo subatributo. Si el vector tiene un tercer elemento, éste es el tamaño del dominio del subatributo para aquellos dominios en los que es necesario especificar su tamaño
Salida	Vector con 2 elementos: El primero es un TransferAtributo (padre) al que le ha sido modificada su lista de componentes. El segundo es un TransferAtributo (hijo) que ha sido añadido al sistema y establecido como componente del padre.
Origen	GUI_AnadirSubatributoAtributo.
Destino	Sistema.
Necesita	DAOAtributos.
Acción	Añade un nuevo subatributo a un atributo del sistema.
Precondición	El atributo padre tiene que ser un atributo compuesto. El nombre del nuevo subatributo no puede ser vacío. Si el dominio del nuevo atributo exige un tamaño, éste debe tener el formato adecuado, esto es, debe ser un valor entero positivo.

Postcondición	En caso de éxito, información sobre la inserción del nuevo atributo y de la modificación del atributo padre. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Editar el carácter multivalorado de un atributo.
Descripción	Cambia el carácter multivalorado de un atributo.
Entrada	TransferAtributo que contiene toda la información del atributo al que se quiere cambiar el carácter de multivalorado.
Salida	TransferAtributo con el carácter de multivalorado modificado.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAOAtributos.
Acción	Cambia el carácter de multivalorado del atributo: Si se trata de un atributo multivalorado, se pone como monovalorado. Si es un atributo monovalorado, se pone como multivalorado.
Precondición	Ninguna.
Postcondición	En caso de éxito, información sobre el cambio en el carácter de multivalorado del atributo. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Establecer/quitar atributo como clave primaria.
Descripción	Establece o quita el atributo seleccionado como clave primaria de la entidad a la que pertenece.
Entrada	Vector de dos elementos: El primero de ellos es un TransferEntidad con toda la información de la entidad a la que pertenece el atributo. El segundo es un TransferAtributo con el atributo que se quiere establecer como clave primaria.
Salida	Vector de 2 elementos:

	El primero es un TransferEntidad con la entidad a la que se ha añadido/quitado el atributo de su lista de claves primarias. El segundo es un TransferAtributo al que se le ha cambiado el atributo de si es o no clave primaria.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAOEntidades.
Acción	Añade o quita de la lista de claves primarias de la entidad el atributo que ha sido seleccionado.
Precondición	El atributo que se pretende establecer/quitar como clave primaria debe ser un atributo directo de una entidad, es decir, no puede ser un atributo de una relación o subatributo de otro atributo.
Postcondición	En caso de éxito, información sobre el cambio realizado en la lista de claves primarias de la entidad. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Ninguno.

Función	Eliminar un atributo/subatributo
Descripción	Elimina del sistema el atributo/subatributo seleccionado.
Entrada	TransferAtributo con toda la información del atributo/subatributo que se desea eliminar del sistema.
Salida	Vector de 2 elementos: El primero es un TransferAtributo con el atributo/subatributo que ha sido eliminado del sistema. El segundo elemento es un Transfer que contiene el elemento que ha sido modificado al eliminar el atributo, quitándole de su lista de atributos: este Transfer puede ser un TransferEntidad, un TransferRelacion o un TransferAtributo dependiendo del elemento que lo contenga en su lista de atributos (entidades y relaciones) o en su lista de componentes (atributos compuestos).
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAOEntidades, DAOAtributos y DAORelaciones
Acción	Elimina el atributo/subatributo seleccionado del sistema.

	Elimina la referencia al atributo del elemento que lo contenga.
Precondición	Ninguna
Postcondición	En caso de éxito, informar sobre la eliminación del atributo/subatributo y sobre la modificación del elemento que lo contenía. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Si se trata de un atributo/subatributo compuesto, hay que eliminar también todos sus subatributos.

Función	Mover la posición de un atributo
Descripción	Mueve el atributo de una posición a otra en el diagrama E/R.
Entrada	TransferAtributo con toda la información del atributo al que se desea cambiar su posición en el diagrama. La nueva posición ya está establecida en el TransferAtributo.
Salida	TransferAtributo al que se le ha modificado su atributo posición.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAOAtributos.
Acción	Cambia el atributo posición del atributo a la nueva posición.
Precondición	Ninguna
Postcondición	En caso de éxito, informar sobre el cambio de posición del atributo. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Ninguno.

4.2.3. Módulo Relaciones.

DBCASE trabaja con dos tipos de relaciones: relaciones normales y relaciones de herencia (IsA). A las relaciones normales las denominaremos solamente como relaciones, mientras que a las relaciones de herencia las llamaremos relaciones IsA.

Por ello expondremos los requisitos funcionales de acuerdo a esta clasificación.

4.2.3.1. Relaciones normales.

El diagrama de casos de uso asociado a las relaciones es:

DBCASE

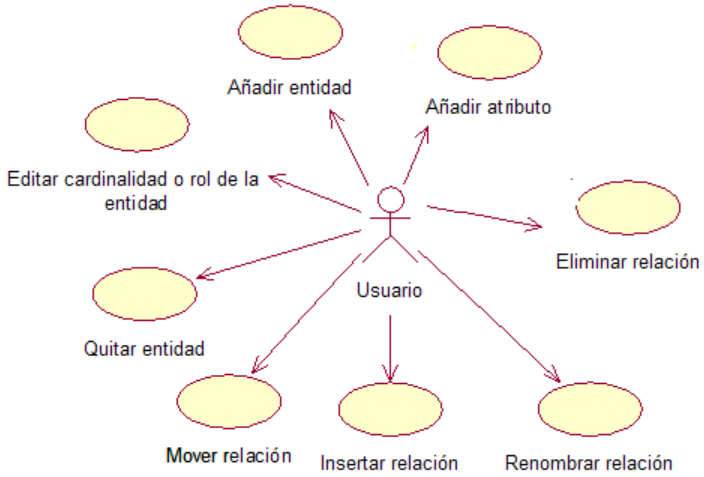


Figura 4. 3

Los requisitos funcionales correspondientes a esta parte son los siguientes:

Función	Insertar una relación.
Descripción	Añadir una nueva relación al proyecto en curso.
Entrada	TranferRelacion que contiene el nombre para la nueva relación que se quiere añadir y la posición en el panel donde se ha pinchado.
Salida	TransferRelacion que contiene el nombre de la nueva relación, la posición en el panel y el identificador interno que le asigna el sistema.
Origen	GUI_InsertarRelacion.
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Crea una nueva relación en el proyecto en curso con el nombre que desee el usuario.
Precondición	El nombre para la nueva relación no puede ser vacío. No puede existir ninguna otra relación en el proyecto en curso cuyo nombre coincida con el que se desea asignar a la nueva relación.
Postcondición	En caso de éxito, información sobre la inserción de la nueva relación. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Ninguno.

Función	Renombrar una relación.
Descripción	Cambiar el nombre a una relación existente en el proyecto en curso.
Entrada	Vector con 2 elementos: El primero de ellos es un TransferRelacion que contiene toda la información de la relación que se quiere renombrar. El segundo es una cadena con el nuevo nombre que se pretende asignar a la relación.
Salida	Vector con tres elementos: El primero es un TransferRelacion con el nombre de la relación cambiado. El segundo es una cadena con el nuevo nombre que se ha asignado. El tercero es una cadena con el antiguo nombre que tenía la relación.
Origen	GUI_RenombrarRelacion.
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Cambia el nombre a la relación seleccionada en el proyecto en curso con el nuevo nombre que desee el usuario.
Precondición	La relación a renombrar no puede ser una relación IsA. El nuevo nombre que se desea asignar a la relación no puede ser vacío. No puede existir ninguna otra relación en el proyecto en curso cuyo nombre coincida con el que se desea asignar a la relación a renombrar.
Postcondición	En caso de éxito, información sobre el renombramiento de la relación. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Añadir un atributo a una relación.
Descripción	Añade un nuevo atributo a una relación existente en el proyecto en curso.
Entrada	Vector de 2 o 3 elementos: El primero de ellos es un TransferRelacion que contiene toda la información de la relación a la que se desea añadir el nuevo atributo. El segundo es un TransferAtributo que contiene toda la información introducida por el usuario para ese atributo (nombre, carácter compuesto, carácter multivalorado y dominio). Si el vector tiene un tercer elemento éste es el tamaño del dominio del

	atributo para aquellos dominios en los que es necesario especificar su tamaño.
Salida	Vector de 2 elementos: El primero de ellos es el TransferRelacion de entrada, con el identificador del nuevo atributo añadido a su lista de atributos. El segundo de ellos es un TransferAtributo igual que el de entrada al que el sistema le ha asignado un identificador único.
Origen	GUI_AnadirAtributoRelacion.
Destino	Sistema.
Necesita	DAORelaciones, DAOAtributos.
Acción	Añade a la relación seleccionada un nuevo atributo con las características que desee para éste el usuario.
Precondición	La relación a la que se le quiere añadir un atributo no puede ser de tipo IsA. El nombre que se proporcione para el nuevo atributo no puede ser vacío. Si el dominio del nuevo atributo exige un tamaño, éste debe tener el formato adecuado, esto es, debe ser un valor entero positivo.
Postcondición	En caso de éxito, informar sobre la inserción del nuevo atributo y sobre la modificación de la relación a la que se le ha insertado. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Añadir una entidad a una relación.
Descripción	Añade una entidad como participante en una relación.
Entrada	Vector de 4 elementos: El primero es un TransferRelacion que contiene toda la información de la relación. El segundo es un TransferEntidad con toda la información de la entidad que se quiere añadir a la relación. El tercero es una cadena que representa el inicio del rango de la cardinalidad de la entidad en la relación. El cuarto es una cadena que representa el final del rango de la cardinalidad de la entidad en la relación.
Salida	Vector de 4 elementos: es el mismo que el de entrada salvo en el primer elemento (TransferRelacion) al que se ha añadido la entidad con la

	cardinalidad (inicio de rango, final de rango).
Origen	GUI_AnadirEntidadARelacion.
Destino	Sistema
Necesita	DAORelaciones.
Acción	Añade una entidad, con una cardinalidad de (inicio, final) a la relación seleccionada.
Precondición	La relación no puede ser de tipo IsA. La cardinalidad de la entidad en la relación debe tener el formato correcto (ambos enteros positivos y el inicio menor o igual que el final de rango).
Postcondición	En caso de éxito, informar sobre la adición de la entidad a la relación y con la cardinalidad que lo hace. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Quitar una entidad de una relación.
Descripción	Quita una entidad participante en una relación.
Entrada	Vector de dos elementos: El primero es un TransferRelacion con toda la información de la relación. El segundo es un TransferEntidad con toda la información de la entidad que se quiere quitar como participante de relación.
Salida	Vector de 2 elementos: El primero de ellos es un TransferRelacion con la relación IsA de entrada en la que se ha quitado la entidad de entrada como entidad participante. El segundo es un TransferEntidad. Es el mismo que el de entrada.
Origen	GUI_QuitarEntidadARelacion.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Quita la entidad como participante de la relación seleccionada.
Precondición	La relación no debe ser de tipo IsA. La relación debe tener establecida como participante la entidad que se desea quitar.

Postcondición	En caso de éxito, informar sobre el la modificación de la relación, quitando la entidad deseada de las entidades participantes. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Editar la cardinalidad de una entidad en una relación.
Descripción	Modifica la cardinalidad de la entidad en la relación.
Entrada	Vector de 4 elementos: El primero es un TransferRelacion que contiene toda la información de la relación. El segundo es un TransferEntidad con toda la información de la entidad que se quiere añadir a la relación. El tercero es una cadena que representa el nuevo valor para el inicio del rango de la cardinalidad de la entidad en la relación. El cuarto es una cadena que representa el nuevo valor para el final del rango de la cardinalidad de la entidad en la relación.
Salida	Vector de 4 elementos: es el mismo que el de entrada salvo en el primer elemento (TransferRelacion) al que se ha editado la cardinalidad (nuevo inicio de rango, nuevo final de rango) de la entidad, situada en segundo elemento del vector.
Origen	GUI_EditarCardinalidadEntidad
Destino	Sistema
Necesita	DAORelaciones.
Acción	Edita la cardinalidad de la entidad deseada (estableciendo los nuevos valores) a la relación seleccionada.
Precondición	La relación no puede ser de tipo IsA, ni débil. La cardinalidad de la entidad en la relación debe tener el formato correcto (ambos enteros positivos y el inicio menor o igual que el final de rango).
Postcondición	En caso de éxito, informar sobre la modificación de la cardinalidad de la entidad en la relación y con la cardinalidad que lo hace. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Ninguno.

Función	Eliminar una relación.
Descripción	Eliminar una relación existente en el sistema.
Entrada	TransferRelacion que contiene la relación que se desea eliminar.
Salida	TransferRelacion con la relación que ha sido eliminada.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Elimina una relación del sistema y quita las referencias a las entidades que intervienen en ella.
Precondición	Ninguna
Postcondición	En caso de éxito, información sobre la eliminación de la relación. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Si la relación a eliminar tiene atributos, éstos serán también eliminados del sistema.

Función	Mover la posición de una relación
Descripción	Mueve la relación de una posición a otra en el diagrama E/R.
Entrada	TransferRelacion con toda la información de la relación que se desea cambiar su posición en el diagrama. La nueva posición ya está establecida en el TransferRelacion.
Salida	TransferRelacion al que se le ha modificado su atributo posición.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Cambia el atributo posición de la relación por el valor de la nueva posición.
Precondición	Ninguna

Postcondición	En caso de éxito, informar sobre el cambio de posición de la relación. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Ninguno.

4.2.3.2 Relaciones de herencia IsA.

Los casos de uso para las relaciones IsA son:

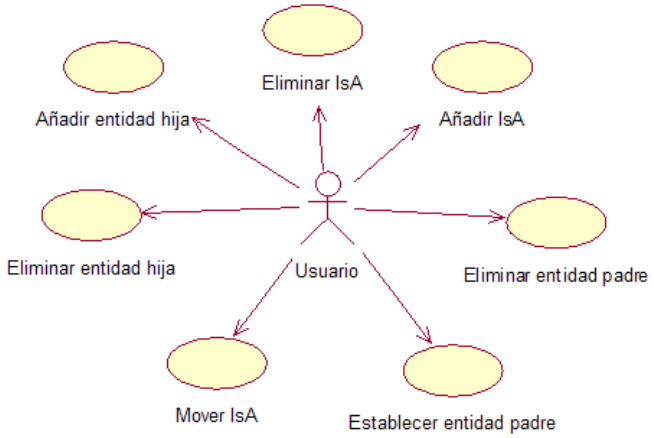


Figura 4. 4

Los requisitos funcionales correspondientes a este tipo de relaciones de herencia son los siguientes:

Función	Insertar una relación IsA.
Descripción	Añadir una nueva relación IsA al proyecto en curso.
Entrada	TransferRelacion la posición en el panel donde se ha pinchado.
Salida	TransferRelacion que contiene la nueva relación IsA que se ha añadido al sistema.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Crea una nueva relación de herencia IsA en el proyecto en curso.
Precondición	Ninguna

Postcondición	En caso de éxito, información sobre la inserción de la nueva relación IsA. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Establecer la entidad padre de una relación IsA
Descripción	Establece la entidad padre en una relación de herencia.
Entrada	Vector de 2 elementos: El primero de ellos es un TransferRelacion con toda la información de la relación IsA. El segundo es un TransferEntidad con toda la información de la entidad que se quiere establecer como entidad padre.
Salida	Vector de 2 elementos: El primero de ellos es un TransferRelacion con la relación IsA de entrada en la que se ha definido la entidad de entrada como entidad padre. El segundo es un TransferEntidad. Es el mismo que el de entrada.
Origen	GUI_EstablecerEntidadPadre.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Establece para la relación seleccionada la entidad deseada como entidad padre de la relación.
Precondición	La relación debe ser de tipo IsA. La relación no debe tener establecida previamente ninguna entidad padre. Debe existir alguna entidad en el sistema.
Postcondición	En caso de éxito, informar sobre el establecimiento de la entidad como entidad padre de la relación IsA. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Quitar la entidad padre de una relación IsA
Descripción	Quita la entidad padre de una relación de herencia.
Entrada	TransferRelacion con toda la información de la relación IsA.

Salida	TransferRelacion con la relación IsA de entrada en la que se ha quitado la entidad que tenía como entidad padre.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Quita para la relación seleccionada la entidad que tenía definida como entidad padre.
Precondición	La relación debe ser de tipo IsA. La relación debe tener establecida previamente una entidad padre.
Postcondición	En caso de éxito, informar sobre el la modificación de la relación IsA. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Al quitar la entidad padre de una relación IsA también serán quitadas las referencias a las posibles entidades hijas que tenga.

Función	Añadir una entidad hija a una relación IsA
Descripción	Añade una entidad como entidad hija en una relación de herencia.
Entrada	Vector de dos elementos: El primero de ellos es un TransferRelacion con toda la información de la relación IsA. El segundo es un TransferEntidad con toda la información de la entidad que se quiere establecer como entidad hija.
Salida	Vector de 2 elementos: El primero de ellos es un TransferRelacion con la relación IsA de entrada en la que se ha añadido la entidad de entrada como entidad hija. El segundo es un TransferEntidad. Es el mismo que el de entrada.
Origen	GUI_AnadirEntidadHija.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Añade una entidad como entidad hija en una relación de herencia IsA.
Precondición	La relación debe tener establecida previamente la entidad padre. Debe existir alguna otra entidad en el sistema, sin contar la entidad padre.

Postcondición	En caso de éxito, informar sobre el establecimiento de la entidad como entidad hija de la relación IsA. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Quitar una entidad hija de una relación IsA
Descripción	Quita una entidad hija de una relación de herencia.
Entrada	Vector de dos elementos: El primero es un TransferRelacion con toda la información de la relación IsA. El segundo es un TransferEntidad con toda la información de la entidad que se quiere quitar como entidad hija de la relación.
Salida	Vector de dos elementos: El primero de ellos es un TransferRelacion con la relación IsA de entrada en la que se ha quitado la entidad de entrada como entidad hija. El segundo es un TransferEntidad. Es el mismo que el de entrada.
Origen	GUI_QuitarEntidadHija.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Quita la entidad que se desee como hija de la relación de herencia seleccionada.
Precondición	La relación debe ser de tipo IsA. La relación debe tener establecida previamente una entidad padre. La entidad que se desea quitar debe ser una entidad hija de la relación IsA.
Postcondición	En caso de éxito, informar sobre el la modificación de la relación IsA al quitando la entidad deseada de las entidades hijas. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

Función	Eliminar una relación IsA.
Descripción	Eliminar una relación IsA existente en el sistema.
Entrada	TransferRelacion que contiene la relación IsA que se desea eliminar.
Salida	TransferRelacion con la relación IsA que ha sido eliminada.
Origen	GUI_Principal
Destino	Sistema.
Necesita	DAORelaciones.
Acción	Elimina una relación IsA del sistema y quita las referencias a las entidades que intervienen en ella.
Precondición	Ninguna
Postcondición	En caso de éxito, información sobre la eliminación de la relación IsA. Si no se ha podido realizar, información explicativa sobre la razón.
Efectos laterales	Ninguno.

Función	Mover la posición de una relación IsA.
Descripción	Mueve la relación seleccionada de una posición a otra en el diagrama E/R.
Entrada	TransferRelacion con toda la información de la relación a la que se desea cambiar su posición en el diagrama. La nueva posición ya está establecida en el TransferRelacion.
Salida	TransferRelacion a la que se le ha modificado su atributo posición.
Origen	GUI_Principal.
Destino	Sistema.
Necesita	DAORelaciones
Acción	Cambia el atributo posición de la relación seleccionada a la nueva posición.
Precondición	Ninguna

Postcondición	En caso de éxito, informar sobre el cambio de posición de la entidad. Si no se ha podido realizar, información explicativa sobre el motivo.
Efectos laterales	Ninguno.

4.2.4. Módulo Sistema.

El diagrama de casos de uso del módulo sistema es:

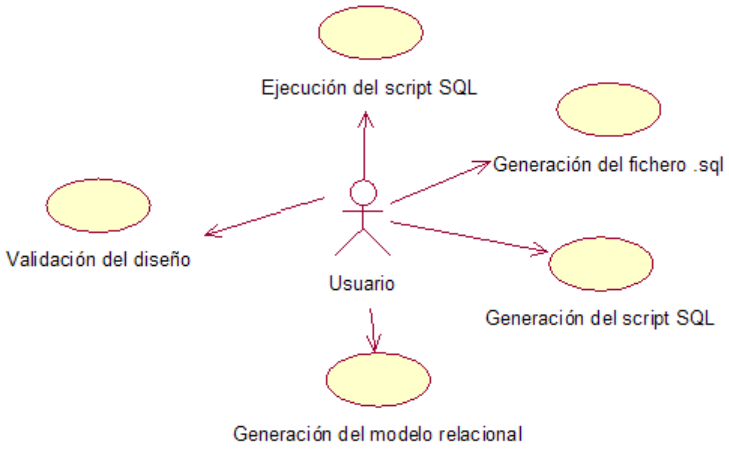


Figura 4. 5

Los requisitos funcionales correspondientes al sistema son los siguientes:

Función	Validación del diseño
Descripción	Valida el diseño E/R de acuerdo a las reglas de diseño de BD.
Entrada	Ninguna.
Salida	Secuencia de cadenas informativas sobre el proceso de validación.
Origen	GUI_Principal
Destino	GUI_Principal
Necesita	DAOEntidades, DAOAtributos y DAORelaciones.
Acción	Realiza la validación del diseño realizado e informa de los resultados al usuario.
Precondición	Ninguna.
Postcondición	Se muestra una secuencia de mensajes correspondientes a los resultados de la validación.
Efectos laterales	Ninguna.

DBCASE

Función	Generación del Modelo Relacional
Descripción	Genera el Modelo Relacional derivado del diseño realizado siguiendo las restricciones especificadas en éste.
Entrada	Ninguna.
Salida	Modelo Relacional en formato texto.
Origen	GUI_Principal
Destino	GUI_Principal
Necesita	DAOEntidades, DAOAtributos y DAORelaciones.
Acción	Genera el modelo relacional derivado del diseño.
Precondición	Diseño previamente validado de forma satisfactoria.
Postcondición	Se muestra al usuario el modelo relacional generado.
Efectos laterales	Ninguno.

Función	Generación del Script SQL
Descripción	Genera el código SQL asociado al diseño realizado.
Entrada	Ninguna.
Salida	Código SQL en formato texto.
Origen	GUI_Principal
Destino	GUI_Principal
Necesita	DAOEntidades, DAOAtributos y DAORelaciones.
Acción	Genera el código SQL asociado al diseño realizado, orientado al gestor de bases de datos eleccionado.
Precondición	Diseño previamente validado de forma satisfactoria.
Postcondición	Se muestra al usuario el código SQL generado.
Efectos laterales	Ninguno.

Función	Generación del fichero SQL
Descripción	Vuelca el código SQL generado a un fichero .sql.
Entrada	Ninguna.
Salida	Fichero .sql con el script del diseño.
Origen	GUI_Principal
Destino	GUI_Principal
Necesita	Nada.
Acción	Guarda en un fichero externo .sql el código SQL generado.
Precondición	Diseño previamente validado de forma satisfactoria y script SQL ya generado.
Postcondición	Fichero .sql correctamente creado.
Efectos laterales	Ninguno.

Función	Ejecución script SQL
Descripción	Se conecta con el gestor elegido y ejecuta el código generado.
Entrada	Ninguna.
Salida	Fichero .sql con el script del diseño.
Origen	GUI_Principal
Destino	GUI_Principal
Necesita	Gestor seleccionado.
Acción	Ejecuta el código SQL generado, creando las tablas correspondientes
Precondición	Diseño previamente validado de forma satisfactoria y script SQL ya generado. Se recomienda ejecutarlo en el mismo gestor de bases de datos para el que fue generado, pero no es obligatorio.
Postcondición	Tablas generadas en el gestor indicado.
Efectos laterales	Ninguno.

5. Arquitectura del sistema.

5.1. Arquitectura multicapa.

DBCASE tiene una estructura multicapa que permite tener dividida la aplicación en diferentes unidades funcionales totalmente independientes. Esto asegura una división clara de responsabilidades y hace que el sistema sea más mantenible y extensible.

Se distinguen tres capas fundamentales: presentación, negocio e integración:

- La capa de presentación encapsula toda la lógica de presentación necesaria para dar servicios a los usuarios que utilizan el sistema.
- La capa de negocio proporciona los servicios del sistema.
- La capa de integración es la responsable de la comunicación con los recursos y sistemas externos.

Una representación gráfica de una arquitectura multicapa de es la siguiente:

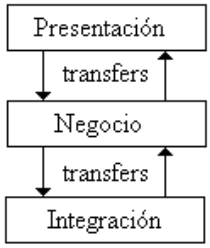


Figura 5. 1

En realidad se trata de una arquitectura de cinco capas, ya que incluye las capas de clientes y de recursos:

- La capa de clientes representa a todos los usuarios que acceden al sistema. Se encuentra situada por encima de la capa de presentación.
- La capa de recursos contiene todos los datos del negocio y los recursos externos. Está por debajo de la capa de integración.

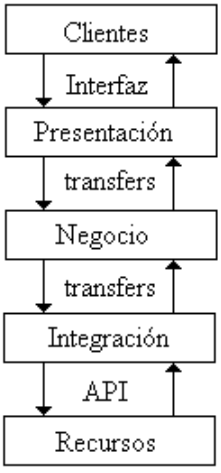


Figura 5. 2

El uso de una arquitectura multicapa ofrece múltiples ventajas frente a una arquitectura de una sola capa o de dos capas. Las principales ventajas son la modularidad y la adaptabilidad.

Cada una de las capas se comporta como una caja negra: ofrece una interfaz a la capa inferior y se comunica con la capa superior a través de la que ésta le proporciona. Debido a esto, se pueden modificar cada una de las capas por separado sin afectar a las demás.

Por el contrario, tiene el inconveniente de tener una mayor complejidad arquitectónica. Sin embargo, esta desventaja se ve superada con creces por las ventajas que proporciona.

Para la implementación de la arquitectura multicapa descrita anteriormente, nos han sido especialmente útiles los siguientes patrones arquitectónicos:

1. Modelo Vista Controlador (MVC).
2. Patrón Transferencia (Transfer).
3. Patrón Objeto de Acceso a Datos (DAO)

A continuación describiremos cada uno de estos patrones y explicaremos la motivación que nos ha llevado a usarlos.

5.2. Patrones de diseño.

5.2.1. Modelo Vista Controlador (MVC)

El modelo Vista Controlador (MVC) es un patrón de arquitectura software que divide una aplicación interactiva en tres capas fundamentales:

1. El **modelo** contiene la funcionalidad básica. En nuestro caso, contiene los servicios de aplicación y la persistencia de los datos:
    1. Los servicios de aplicación implementan todos los requisitos software del sistema.
    2. La persistencia se encarga de la gestión explicita de los datos de nuestra aplicación.
2. Las **vistas** tienen una doble funcionalidad: muestran y recogen información al/del usuario.
3. El **controlador** media entre las vistas y el modelo.

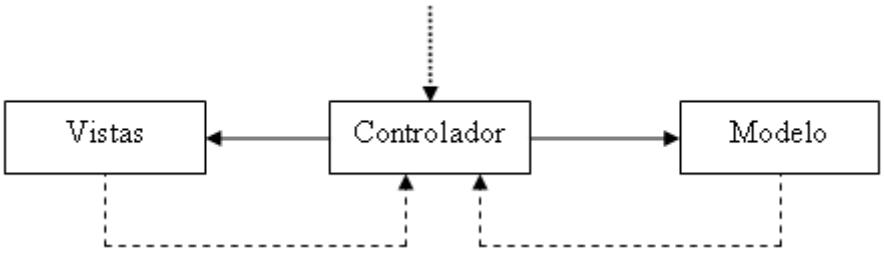


Figura 5. 3

De las dos variantes que tiene el MVC, DBCASE usa la versión con controlador **activo**: cualquier cambio en el modelo es notificado por el controlador a las vistas.

Toda la comunicación multicapa del sistema pasa por el controlador, es decir, la única forma de comunicar las vistas y el modelo es a través de éste.

De forma esquemática, el flujo de control que sigue este modelo es el siguiente:

2. El usuario interactúa con la interfaz de usuario de alguna forma, por ejemplo, pulsando uno de los botones.
3. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega.

4. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario.
5. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo (pasando previamente por el controlador) para generar la interfaz apropiada para el usuario, donde se reflejan los cambios realizados en el modelo. El modelo no tiene conocimiento directo sobre la vista.
6. En nuestra implementación, la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.
7. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

Este modelo ofrece múltiples ventajas. Entre ellas podemos destacar las siguientes:

- Posibilita tener un modelo independiente de la representación de la salida y del comportamiento de la entrada,
- Permite tener simultáneamente múltiples vistas para un mismo modelo.

Su interés de uso principal radica en la independencia de los cambios: se pueden hacer las modificaciones que se crean oportunas en cada una de las capas de la aplicación y no afectar al resto del modelo.

5.2.2. Patrón Transferencia (Transfer).

El patrón transferencia, más conocido como transfer, es un patrón de arquitectura software cuyo propósito es independizar el intercambio de datos entre capas.

Dado que nuestro objetivo es independizar cada una de las capas de nuestro sistema de las demás, no es posible que una capa tenga conocimiento de la representación de las entidades dentro de las otras.

Su utilidad radica en el hecho de homogeneizar el intercambio de datos entre las capas, de modo que en cada una de las transferencias de información solamente se intercambian objetos serializables: objetos transfer, conjuntos de transfer y otros objetos serializables (enteros, cadenas, caracteres...).

DBCASE usa cinco tipos de transfer, uno para cada una de las cuatro entidades con las que trabaja nuestro sistema y uno para las conexiones con las bases de datos, estas son:

- TransferEntidad.
- TransferAtributo.
- TransferRelacion.
- TransferDominio
- TransferConexion

El acceso a cada uno de los objetos transfer se realiza por medio de *getters* y de *setters*. De modo ilustrativo, mostramos el contenido de uno de los transfers:

```
public class TransferEntidad extends Transfer{
    private int idEntidad;
    private String nombre;
    private boolean debil;
    private Vector listaAtributos;
    private Vector listaClavesPrimarias;
    private Vector listaRestricciones;
    private Vector listaUniques;
    private Point2D posicion;
```

Figura 5. 4

Si se desea conocer la implementación concreta de cada uno de los transfers, consultar la sección de implementación de esta memoria.

La ventaja de trabajar con objetos transfers es su propia definición: independizar el intercambio de datos entre capas. Con ellos promovemos la modularidad del sistema y la independencia entre capas del mismo.

5.2.3. Objeto de Acceso a Datos (DAO)

El patrón DAO, se puede incluir en un diseño de una aplicación multicapa en la capa de persistencia.

El objetivo básico del patrón es facilitar y homogeneizar el acceso a los datos de la aplicación. Estos datos pueden tener una estructura concreta que se refleja en un sistema determinado de representación. En nuestro sistema, se trata de una colección de ficheros XML.

La idea de controlar los datos nos obliga a adquirir ciertos conocimientos sobre el sistema que los contendrá. Para conocer más sobre el acceso a los ficheros XML puede consultarse la sección de la documentación encargada de la implementación de la capa de persistencia.

La independencia del resto de capas que supone el uso del patrón DAO permite modificar precisamente el modo de almacenaje de los datos, pudiendo variar de un fichero a una base de datos o cualquier otra estructura. Esto permite optimizaciones en el diseño de cualquier aplicación. El único inconveniente que se le puede encontrar al patrón DAO puede ser que aumenta el número de objetos del sistema.

Las operaciones esenciales de un DAO son la escritura, la modificación, la consulta y la eliminación de datos. DBCASE, además de implementar estas cuatro operaciones en cada uno de los DAOs, incluye la posibilidad de obtener un listado con todos los objetos de información.

Cabe destacar que los objetos que se manejan en cada uno de nuestros DAOs son objetos Transfer o conjuntos de ellos.

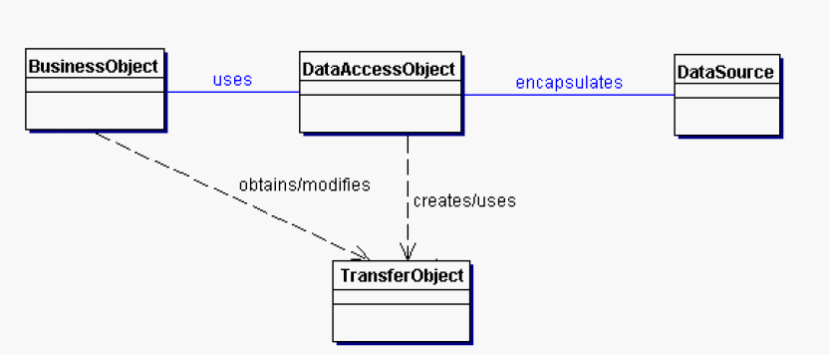


Figura 5. 5

5.2.4. Patrón de factoría

Para incluir los distintos gestores de bases de datos a utilizar en la aplicación, usamos una variante del patrón de factoría.

5.2.4.1. Patrón original

Este patrón de diseño consiste en utilizar una clase constructora abstracta con unos cuantos métodos definidos y otros abstractos: los dedicados a la construcción de objetos de un subtipo de un tipo determinado.

Las clases principales en este patrón son el creador y el producto. El creador necesita crear instancias de productos, pero el tipo concreto de producto no debe ser forzado en las subclases del creador, porque entonces las posibles subclases del creador deben poder especificar subclases del producto para utilizar.

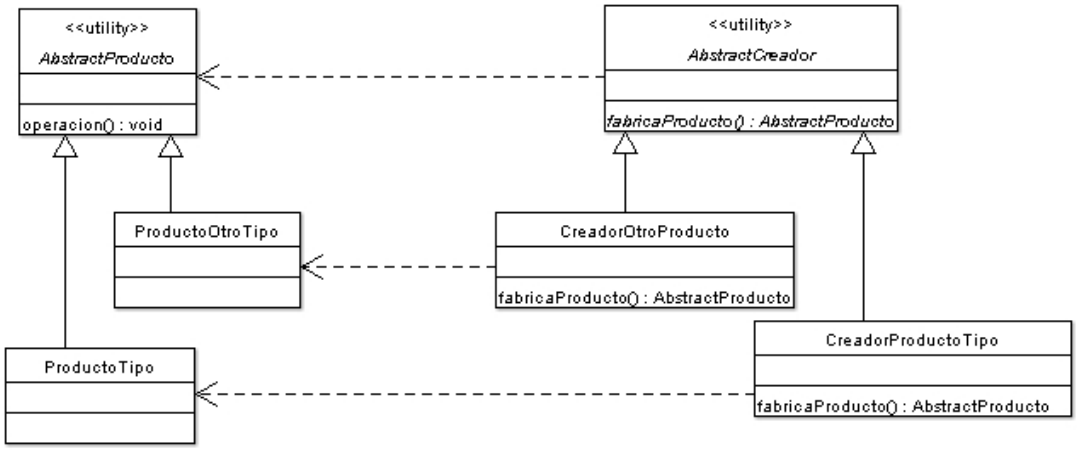


Figura 5. 6

5.2.4.2. Variación

En nuestro caso, modificamos en creador para que, además de poder crear instancias de todos los productos implementados, sea capaz de devolver una lista con los productos que es capaz de generar. De esta manera, el usuario de la factoría conoce qué productos puede solicitar a la factoría.

Además, la factoría deja de ser abstracta, y es ella misma la que se encarga de generar cualquier producto. Para distinguir qué producto se solicita, se utiliza un parámetro de entrada en la función fabricaProducto. Los posibles valores para este parámetro se obtienen invocando al método obtenerTiposDeProductos.

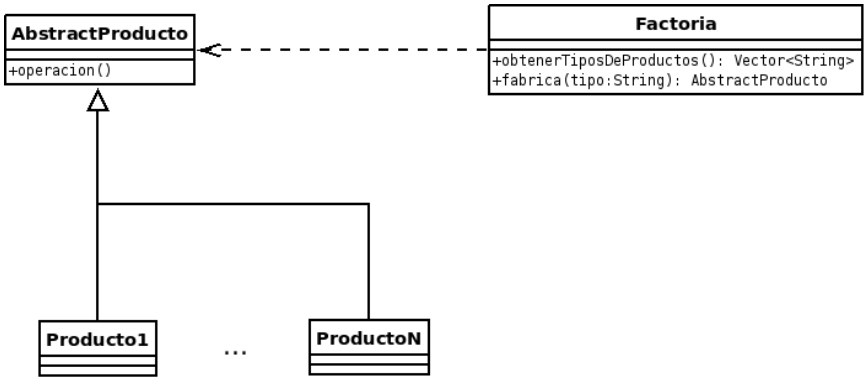


Figura 5. 7

6. Implementación del sistema.

6.1. Introducción.

Para la implementación final de la aplicación se ha escogido el lenguaje de programación **Java**.

Bien podría haberse escogido otro lenguaje de mayor rendimiento como C++, a priori, pero, al ser uno de los objetivos prioritarios la aplicación docente de la herramienta, se ha considerado como mejor alternativa el lenguaje definido originalmente por Sun Microsystems.

Es cierto que en torno al 90% de los sistemas de escritorio poseen un sistema operativo Windows, de la empresa Microsoft, pero en entornos académicos, y especialmente aquellos a los que se ha considerado más práctica la herramienta, dichos porcentajes varían considerablemente. En este motivo principal se basa la elección del lenguaje Java, y su correcto funcionamiento ha sido comprobado en sistemas Windows y distribuciones GNU/Linux (Debian y Ubuntu) MAC OS X.

El diseño de la aplicación trata de ser, en todo lo posible, independiente de la arquitectura y el sistema operativo utilizado, para que cualquier usuario tenga la capacidad de utilizarlo para el fin que estime oportuno.

La máquina virtual de Java utilizada en todos los casos ha sido la ofrecida por Sun Microsystems versión 1.6.0 (o posteriores). La elección de dicha máquina está motivada por la mejora técnica de la misma frente a versiones previas y funciones que la misma posee de forma única.

6.2. Organización de paquetes.

La implementación de la herramienta se estructura en una serie de paquetes, en los que se modularizan todos los comportamientos y acciones de la misma, para una fácil reutilización de código y simplicidad de comprensión del mismo.

A continuación se presenta una breve descripción de los mismos:

- **Controlador**. En este paquete se describen tanto el comportamiento del módulo Controlador como los mensajes que el mismo es capaz de enviar y recibir. Además, se contiene la clase encargada de configurar inicialmente la vista y la lógica de negocio.
- **LogicaNegocio**. Este paquete contiene dos subpaquetes internos en los que especifica, de forma más concreta la lógica de la aplicación:
    - **Servicios**. Proporciona a la aplicación el interfaz y ejecución de los servicios lógicos ofrecidos por la misma. Dichos servicios serán abstractos para el resto de módulos de la herramienta y realizarán, de forma atómica, las acciones necesarias para su correcto funcionamiento.
    - **Transfers**. Define las clases que modelarán los objetos en el flujo de mensajes de la aplicación. Los objetos Transfer son aquellos que se transmiten de módulo en módulo, a partir de mensajes, y en los que se almacena cualquier cambio o acción que ha de ser tratada.
- **Persistencia**. En este paquete se definen las clases que trabajan a más bajo nivel. DBCASE es una aplicación que trabaja con datos persistentes en todo momento y las clases DAOS, que este paquete contiene, son las encargadas de su correcta sincronización con el fichero temporal en disco.

- **Presentación**. Este paquete contiene el conjunto de GUI (Interfaz Gráfica de Usuario) que el usuario de la aplicación visualizará en su ejecución. Es importante mencionar que las interfaces únicamente tienen la tarea de mostrar datos y recibir acciones por parte del usuario, y nunca realizar gestiones con los mismos. El paquete Presentación posee dos subpaquetes, para mejor modularidad, que contienen de forma separada la representación gráfica del esquema y la clase lenguaje:
  - **Grafo**. En este paquete se describen las clases necesarias para la visualización del grafo relacional presentado por la aplicación. Para dicha visualización se ha hecho uso de la librería JUNG (Java Universal Network/Graph Framework), habiendo sido necesario una redefinición de muchas de las características que dicha librería ofrece.
  - **Lenguajes**. La única clase de este paquete es la encargada de la globalización, se encuentra incluida en la presentación ya que todas las ventanas utilizan elementos de esta clase para mostrar cadenas en función del lenguaje seleccionado
- **Utilidades**. Este último paquete constituye aquellas partes de la herramienta que han sido utilizadas como elementos externos o no-propios del diseño. Dichas partes, entre otras, son: un lanzador de ayuda mediante el explorador Web propio del sistema nativo, rutas de las imágenes presentadas en la aplicación, utilidades sobre código HTML...

6.3. Presentación.

La Presentación de la aplicación engloba todas las interfaces de usuario existentes, además del diseño que representará el esquema relacional.

El conjunto de interfaces está constituido por un frame, o marco principal, un diálogo de cambio de espacio de trabajo y dieciséis interfaces de acción/decisión. El marco de diseño gráfico, contenido en el frame, será expuesto de forma singular, al ser su implementación la más crítica dentro de esta capa.

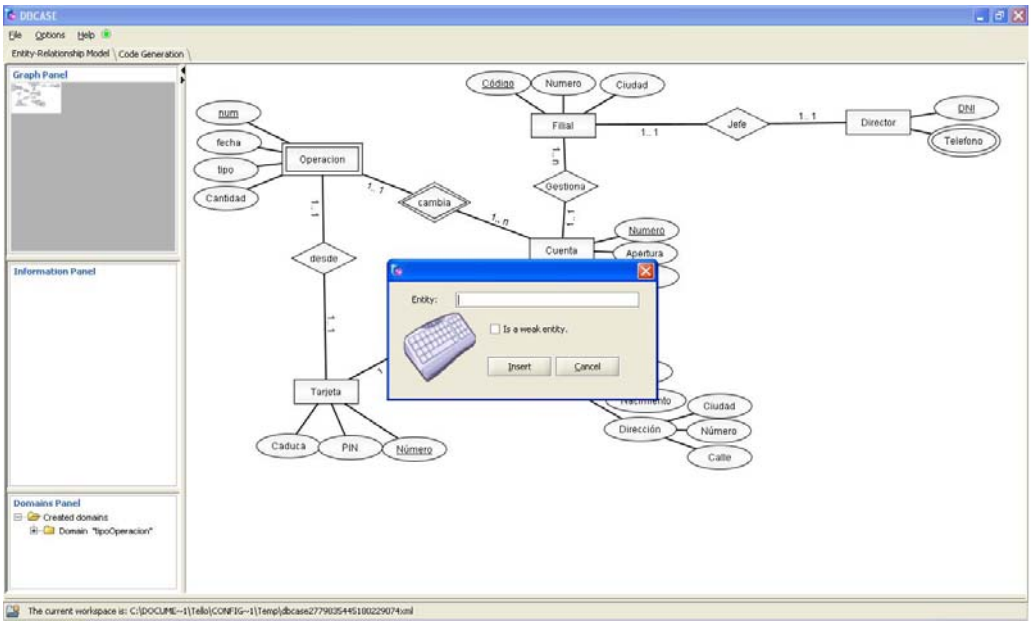


Figura 6. 1

El objetivo principal, y único, de todas las interfaces de usuarios es ser el nexo entre el usuario y los servicios ofrecidos por el sistema. De esta forma, la interfaz no manipula ni realiza cambio alguno en ningún dato, sino que registra las variaciones emitidas por el usuario de la aplicación y, si es necesario, actualiza la vista de los datos con la información obtenida de los demás módulos.

Por lo tanto, el interfaz de todas las GUIs (Graphic User Interface) consiste básicamente en recopilar eventos de acción del usuario, ya sea mediante pulsaciones de botones, activaciones de campos en menús desplegables o el mero hecho de usar la rueda del ratón, por ejemplo.

Todas las interfaces son generadas de forma estática al comienzo de la aplicación. Por este motivo, todas ellas, poseen los métodos:

```
public void setActiva()
public void setInactiva()
```

Figura 6.2

El nombre de los métodos es, explícitamente, lo que realiza su implementación. Si se activa un interfaz se muestra, mientras que si se desactiva sencillamente se oculta, no se libera su memoria. Este funcionamiento se expondrá de forma más concisa en el desarrollo de la implementación del Controlador.

Para simplificar el desarrollo únicamente se expondrá el esqueleto de una interfaz de acción/decisión, dado que, a excepción de ciertos contenidos gráficos como cajas de texto o combos de opciones, todas comparten el mismo esquema.

6.3.1 Interfaces de acción/decisión.

Estas interfaces se mostrarán cuando el usuario realice una acción en la que se necesite completar ciertos parámetros adicionales o, ya bien, solicitar una confirmación por parte del mismo.

Acciones pueden ser, entre otras: insertar una entidad en el diagrama, insertar una relación, añadir un atributo a una relación, alterar los parámetros de un atributo...

Por lo tanto, el esquema de todo interfaz acción/decisión será el de ofrecer ciertas propiedades susceptibles de cambio y solicitar al usuario la confirmación por medio de dos botones: Aceptar y Cancelar.

Si el usuario selecciona el botón de Cancelar, la interfaz se ocultará y no se realizará modificación alguna en los datos. No existe ninguna otra acción asociada a dicho botón, pues no es necesaria.

Por el contrario, si el usuario selecciona el botón de Aceptar, la interfaz recogerá todos los datos contenidos en la misma, que el usuario habrá completado o rellenado, y enviará un mensaje al Controlador notificando el tipo de acción que se ha llevado a cabo y almacenando en un objeto serializable (Transfer, conjunto de Transfers, cadena, entero…) la información asociada al cambio en los datos.

Se adjunta un ejemplo, en el caso de inserción de una relación en el esquema, como ejemplo de implementación. Al poseer el mismo esquema todas las interfaces, no será necesario exponer casos excepcionales.

```
private void botonInsertarActionPerformed(java.awt.event.ActionEvent evt) {
    // Generamos el transfer que mandaremos al controlador
    TransferRelacion tr = new TransferRelacion();
    tr.setPosicion(this.getPosicionRelacion());
    tr.setNombre(this.cajaNombre.getText());
    tr.setListaAtributos(new Vector());
    tr.setListaEntidadesYAridades(new Vector());
    tr.setListaRestricciones(new Vector());
    tr.setListaUniques(new Vector());
    tr.setTipo("Normal");
    // Mandamos mensaje + datos al controlador
    this.getControlador().mensajeDesde_GUI(TC.GUIInsertarRelacion_Click_BotonInsertar, tr);
}
```

Figura 6. 3

Al separar responsabilidades por módulos, o capas, no es necesario un mayor desarrollo en las acciones de usuario. Tras el envío de este mensaje, será el Controlador el encargado de ocultar la interfaz y completar los datos en el frame principal.

6.3.2. Diálogo de cambio gestión de proyectos

Esta interfaz se expone de forma separada al no compartir la funcionalidad de todas las demás.

La tarea de la misma es simple: permitir al usuario abrir y guardar proyectos. El funcionamiento de este servicio ha sido completamente modificado respecto a la versión anterior del proyecto, que gestionaba cada proyecto en un directorio de trabajo diferente. Así hemos hecho esta funcionalidad más simple y potente, ya que ahora tener diferentes versiones de un proyecto es tan sencillo como tener diferentes ficheros.

Un proyecto será un archivo XML en el que se almacena toda la información a cerca del Diagrama Entidad-Relación.

Se ha creado el menú Fichero en la barra de menús para gestionar los proyectos de manera cómoda e intuitiva.

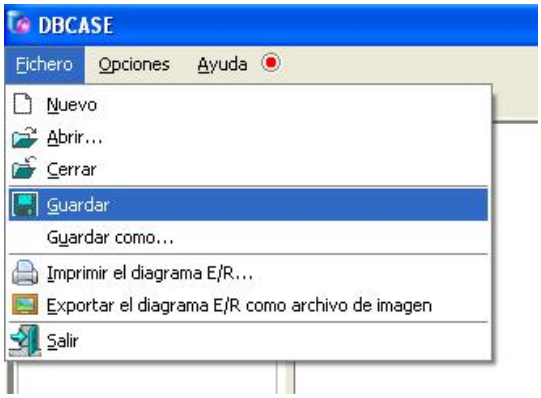


Figura 6.4

El usuario puede, a través de este menú, crear un proyecto o cerrar, abrir y guardar, proyectos ya creados. Para abrir un proyecto creado anteriormente se hace uso del cuadro de diálogo abrir donde se elige el directorio y el nombre del proyecto.

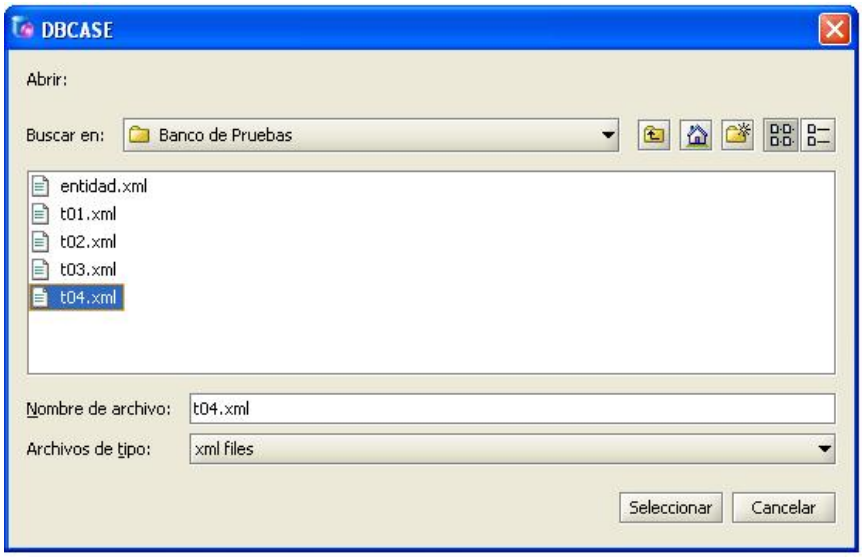


Figura 6.5

Para guardar, existen las dos opciones típicas Guardar y Guardar como, la primera guardará los últimos cambios en el fichero desde donde se abrió o guardó la última vez el proyecto, y en caso de ser un proyecto nuevo tendrá el mismo efecto que Guardar como, que abre un diálogo equivalente al de la opción Abrir.

Como ayuda visual para recordar el estado del proyecto, hemos añadido un icono que toma color verde o rojo según si el proyecto actual ha sido modificado desde el último cambio.

6.3.3. Frame principal.

Esta interfaz es la GUI principal a la que tendrá acceso el usuario.

Presenta diversas opciones, tanto en menús de título, como en acciones dentro del marco de diseño, y su implementación es tanto más compleja, como más amplia, que el resto de interfaces.

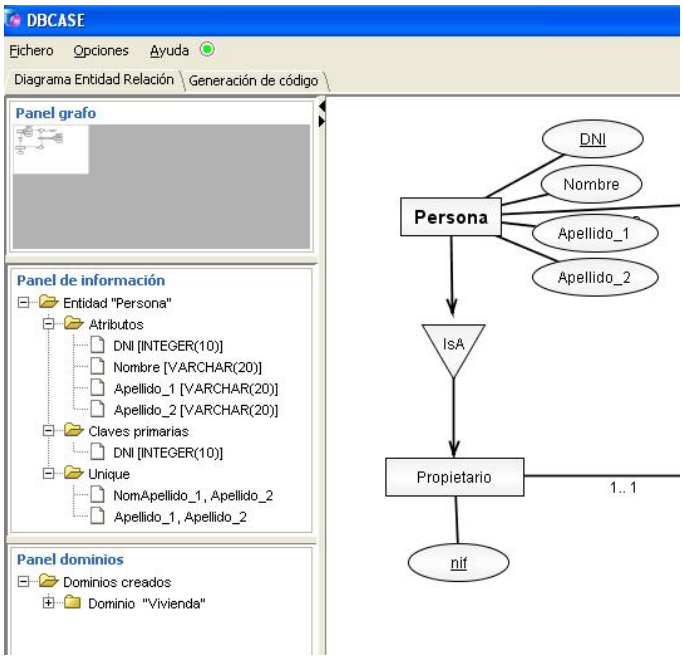


Figura 6. 6

Al seguir el patrón de diseño modelo vista-controlador la descripción de la implementación de esta interfaz se dividirá en dos secciones: las acciones directas y las acciones indirectas.

Se considerará una acción indirecta aquella que llega desde componentes internos de la aplicación, desde el Controlador en este caso, y acciones directas aquellas que se invocan por el usuario y realizan peticiones indirectas al resto de la herramienta.

NOTA: Los paneles de diseño serán expuestos en una sección separada, dada su complejidad.

A continuación se describen las dos secciones del frame principal:

- **Acciones directas**. Estas acciones son las provocadas por eventos del usuario. Principalmente, y a excepción de las relacionadas con los paneles de diseño, serán llevadas a cabo a través del menú de Opciones, del menú Fichero o a través de los botones situados en la pestaña Generación de código.
    - **Menú de opciones**. El menú de opciones consta de tres menús básicos.
        1. **Fichero**. El menú fichero, que no existía en la versión anterior, posee las opciones de:

           Nuevo, Abrir, Cerrar, Guardar y Guardar Como, para la gestión de proyectos, estas selecciones limitan su implementación a notificar al Controlador de la acción deseada por el usuario.

           Así mismo el menú fichero posee las selecciones de Exportar el diagrama a un fichero gráfico e Imprimir el diagrama. Por la estructura, y el diseño, de ambas opciones y dado que no es necesaria la intervención de datos externos se ha elegido implementar la funcionalidad de ambas opciones dentro de la clase del panel de diseño. Ello conlleva que ningún otro módulo es consciente de la acción de exportar o imprimir llevada a cabo, y su funcionamiento se expondrá en el desarrollo del panel de diseño.

           Por último, la opción Salir informa al controlador de que el usuario quiere cerrar la aplicación.

        2. **Opciones**. El menu Opciones posee las selecciones de:

           El Gestor de Base de Datos actual permite seleccionar el tipo de gestor para el que se quieren generar los *scripts*, esta funcionalidad se implementa dentro de la misma clase, ya que el diseño es independiente del gestor elegido. Esta opción es nueva.

           Lenguaje da al usuario la opción de elegir el idioma en el que desea trabajar. Informa al controlador de la opción elegida, ya que éste debe crear de nuevo el frame principal para mostrarlo en el idioma elegido. Es otra de las novedades de esta versión del proyecto.

           Ver panel de sucesos muestra u oculta el panel de sucesos, esta opción también se implementa localmente, por ser completamente independiente del resto de funcionalidades. Es también una opción nueva.

        3. **Ayuda**. El menú de Ayuda contiene las opciones de:

           Contenidos y Acerca de DBCASE. Ambas acciones se implementan de manera atómica en el frame principal. Contenidos, usando una utilidad externa, abre el Manual de Usuario en formato HTML en el

explorador Web por defecto del sistema y Acerca de DBCASE se limita a mostrar un diálogo con la información característica de la aplicación.

- **Botones de Generación de Código**. Existen seis botones de Generación de Código disponibles en la interfaz principal. A excepción del botón Limpiar pantalla de texto, todos los demás se implementan como hilos concurrentes dentro de la aplicación. La motivación de dicha implementación viene derivada del refresco deseado de información en el panel de texto y la no congelación de dicho refresco hasta la finalización de la acción deseada.
    1. Limpiar pantalla de texto. Este botón elimina el contenido, en formato texto, que el área de texto de generación posee.
    2. Validar modelo relacional. Dicho botón activa el chequeo del esquema diseñado por el usuario. Realiza el envío de un mensaje al módulo Controlador para la comprobación de dicha validación.
    3. Representación del modelo relacional. Este botón solicita la representación del modelo relacional por parte de los servicios de la herramienta. Su único interfaz es el envío de un mensaje al módulo Controlador para que se realice dicha acción. Para poder llevar a cabo esta acción, es necesario haber validado el diseño previamente.
    4. Generación del script SQL. Este botón solicita la generación del script SQL en el panel de texto para su posible uso en una base de datos comercial. Su único interfaz es el envío de un mensaje al módulo Controlador para que se realice dicha acción. El script será generado específicamente para el gestor de bases de datos que esté seleccionado. Para poder llevar a cabo esta acción, es necesario haber validado el diseño previamente.
    5. Exportar script a fichero de texto. Este botón solicita la exportación del script SQL a un fichero de texto ASCII con el objetivo de su futuro uso en una base de datos. Su único interfaz es el envío de un mensaje al módulo Controlador para que se realice dicha acción.
    6. Ejecutar script SQL en un DBMS. Este botón muestra un gestor de conexiones existentes, en el cual podemos crear, modificar o eliminar configuraciones de conexión. Una vez elegida una conexión, se envía un mensaje al Controlador para que ejecute el script generado en el gestor indicado. Para que esto sea posible, se debe haber generado un script previamente.

- **Acciones indirectas**. Las acciones indirectas son aquellas en las que se solicita un cambio de contenidos por parte de módulos internos de la aplicación. Como todos los mensajes son centralizados, por parte del módulo Controlador, se ha de seleccionar, en el caso adecuado, la acción que se ha de realizar. Todos los mensajes provenientes de acciones internas modifican, de algún modo, componentes gráficos y, en esencia, suelen modificar los paneles de diseño de la herramienta. Por lo tanto, el esquema de implementación patrón para todos los casos es la recepción del mensaje por parte del objeto Controlador, la extracción de los datos necesarios y la actualización en los objetos gráficos de su contenido. Como ejemplo de acción entrante señalamos otra pequeña novedad incluida en la herramienta: el indicador de cambios salvados, que informa al usuario de si los últimos cambios has sido guardados.

Figura 6. 7

6.3.4. Marcos de diseño.

Los marcos de diseño, como tal, forman parte del frame principal que la aplicación posee, pero se ha separado su exposición debido a la criticidad de los mismos y su notable diferente implementación con respecto a las demás.

Existen cuatro marcos de diseño en la aplicación DBCASE:

1. El Panel de Diseño.
2. El Panel de Pre-Visualización (o thumbnail).
3. El Panel de Información.
4. El Panel de Dominios

6.3.4.1. Panel de diseño.

El Panel de Diseño es el módulo de la herramienta más complejo, a nivel lógico y funcional. En el mismo se representa el esquema relacional diseñado, de forma gráfica, e interactúa en multitud de maneras con el usuario.

Para la implementación de este módulo se ha hecho uso de la librería JUNG (Java Universal Networks/Graphs FrameWork), aunque sólo se ha requerido su aspecto visual para los fines de la herramienta.

El Panel de Diseño tiene dos funcionalidades internas que requieren especial mención. Ambas funcionalidades son la impresión en papel del esquema diseñado y la exportación a fichero gráfico del mismo. Las dos características utilizan herramientas provistas por Java para facilitar ambas tareas, por lo que no ha sido necesaria una implementación especial dedicada para dichas acciones.

Se han redefinido y sobrecargado métodos y clases que la librería ofrece con el fin de adaptar el diseño al modelo deseado. Cabe señalar la implementación de etiquetadores de nodos y aristas para la correcta visualización de la información del esquema, así como representadores gráficos para cada tipo de nodo particular.

Así pues, el panel de diseño posee dos facetas destacables:

1. Información del esquema relacional diseñado. El Panel de Diseño posee todos los datos del esquema que se está desarrollando de forma exclusiva y los organiza en tablas dispersas (hash-tables) para un óptimo acceso a todos los datos necesarios. Estos datos son exactamente los mismos que poseen todos los demás módulos del sistema, ya que los datos almacenados son referencias a los mismos, y la única modificación que se posee es en la estructura de contenido de los mismos.
2. Representación y captura de eventos en el mismo por parte del usuario. El Panel de Diseño se encarga tanto del mantenimiento gráfico del esquema como del entorno de eventos que el usuario realiza sobre el mismo. Esto significa que cualquier acción sobre el grafo será capturada por el Panel de Diseño y se realizará el protocolo habitual de mensajes. El módulo de Panel de Diseño es un componente interno del

frame principal, pero todas las acciones que el usuario realice sobre el mismo son notificadas directamente al módulo Controlador, ya que podría considerarse más una extensión del mismo que un módulo interno. El Panel de Diseño maneja, en esencia, tres tipos de acciones:

1. Pulsaciones sobre el panel. Existen dos tipos de pulsaciones posibles sobre el panel, que dependerán del botón del ratón utilizado en cada caso. Si se pulsa el botón primario se asocia dicha acción a la petición de información sobre un elemento. Esto significa que el panel captura la acción del usuario, selecciona dónde ha pulsado, si en un nodo del esquema o un espacio vacío, y envía al Controlador dicha información para que se pueda actualizar el Panel de Información.
   Si, por el contrario, se pulsa el botón secundario, el panel capturará el evento del usuario y le mostrará, en un menú desplegable, las posibles opciones de modificación del esquema que son posibles en dicho espacio. Es decir, si la pulsación se realiza sobre un nodo se mostrará las posibles opciones de modificación sobre ese nodo, y si es sobre un espacio vacío, el panel mostrará un menú con las posibles opciones de inserción en dicho espacio vacío. Una vez capturado el evento de acción del usuario, y seleccionada la acción deseada por parte del mismo, se procede al funcionamiento estándar de paso de mensajes al Controlador.
2. Movimiento de nodos visuales. Si se pulsa con el botón izquierdo sobre el ratón y se realiza un movimiento del mismo sin la relajación del botón presionado, se realiza un movimiento sobre el nodo del esquema seleccionado. Ello significa que se capturará el nodo movido y la posición final del mismo para notificar al Controlador de dicha acción. Es posible la selección de varios nodos simultáneos y su movimiento en conjunto. El funcionamiento de esta característica está íntimamente ligada, y viene proporcionada gracias, a la implementación de la librería JUNG.
3. Zoom sobre la vista del panel. Si se utiliza la rueda del ratón sobre el Panel se capturará dicha acción y se almacenará el sentido del movimiento. El objetivo de esta acción será la de ampliar o disminuir el zoom que afecta al Panel de Diseño hasta, en principio, un límite no establecido. Este comportamiento no altera ningún dato del esquema diseñado y, por consiguiente, no existe ningún intercambio de mensajes con ningún otro módulo del sistema.

6.3.4.2. Panel de pre-visualización (o thumbnail).

Este panel no permite una modificación del esquema por parte del usuario, pero fue concebido para simplificar la búsqueda sobre el Panel de Diseño del esquema representado.

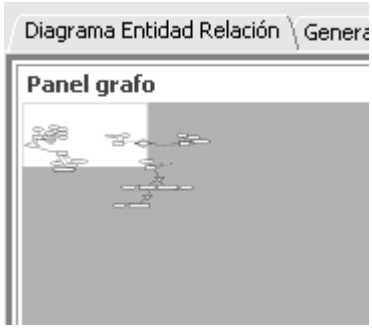


Figura 6. 8

El Panel thumbnail muestra una mera representación alejada de todo el espacio en el que se puede desarrollar el esquema relacional. La representación es la misma, pues se pueden observar los mismos objetos gráficos en ambos paneles y, de hecho, comparten la misma información. Ello significa que un movimiento o cambio en el Panel de Diseño modificará el contenido del Panel de Pre-Visualización sin la necesidad de realizar ningún cambio en el mismo.

Éste panel sólo permite un tipo de acción por parte del usuario, y es desplazar la parte visible, coloreada en blanco, por su contenido. Al estar enlazados ambos paneles, la actualización de la vista en el Panel de Diseño es automática, sin la necesidad, de nuevo, de realizar ningún tipo de procedimiento extra.

Es decir, el panel de Pre-Visualización no posee ningún interfaz con el resto de módulos del sistema, ya que está vinculado a los necesarios desde su construcción.

Aunque este panel ya existía en la versión anterior su funcionamiento era incorrecto. Un desplazamiento en este panel cambiaba la vista en el panel de diseño pero no la posición relativa donde el usuario pulsaba. Esto hacía que tras un movimiento de la vista, cualquier *click* se capturaba en un punto diferente a donde había sido realizado, haciendo la herramienta inutilizable.

6.3.4.3. Panel de información.

El Panel de Información muestra la información que se desee exponer de forma explícita. Ello se consigue con la pulsación del ratón sobre un nodo del esquema, por ejemplo.

La implementación del Panel de Información consta de un elemento `JTree` (o árbol gráfico) en que se muestran las características o propiedades del elemento deseado.

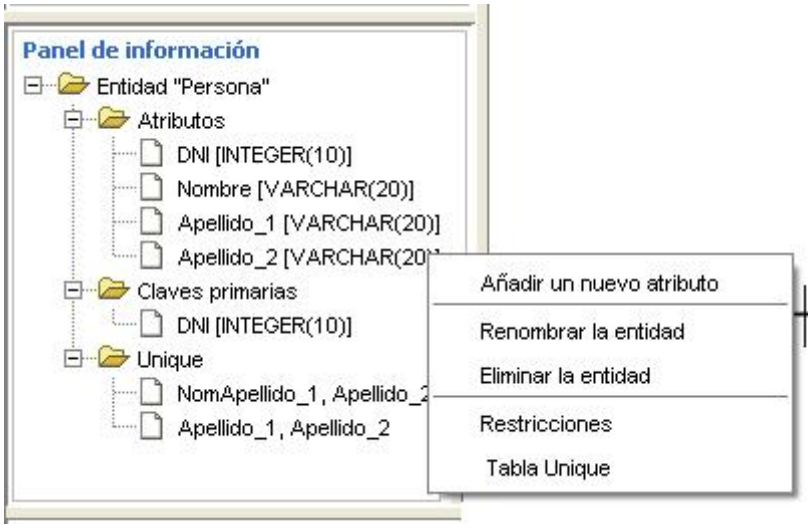


Figura 6. 9

Su interfaz consiste en la recepción de un mensaje por parte del Controlador, dicho mensaje ya poseerá el objeto árbol necesario para su representación. Además le hemos añadido la captura de la pulsación del botón secundario que muestra el menú desplegable con las acciones de modificación del nodo.

Por la simplicidad de implementación del panel se decidió implementar dentro del módulo frame principal.

6.3.4.4. Panel de dominios.

El Panel de Dominios consiste en un árbol en el cual se visualiza toda la información relativa a los dominios creados por el usuario. Debido a que los dominios no son representados en el

diagrama entidad relación, este panel es la única interfaz para visualizar y modificar un dominio. Este panel no existía en la versión anterior.

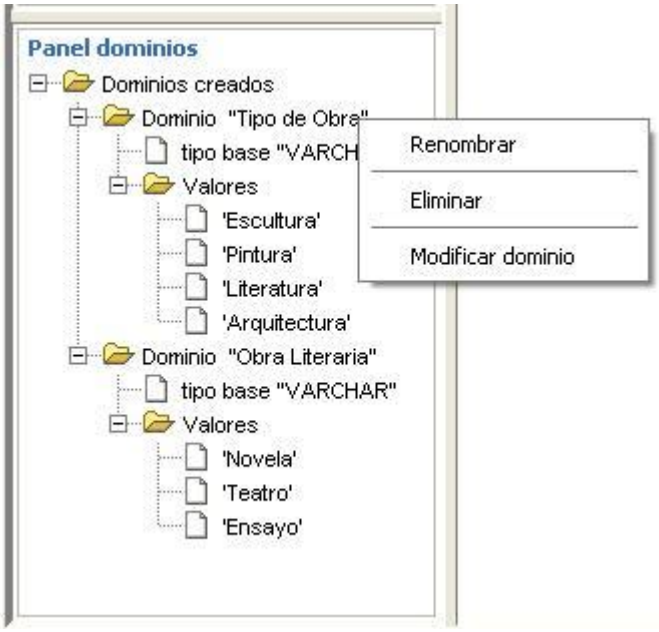


Figura 6. 10

La información de cada dominio contiene el tipo base, que es el tipo al que pertenecen los elementos del enumerado, y un nodo Valores con todos los que componen el dominio.

Las modificaciones se hacen a través de menús desplegables que están asociados a las pulsaciones del botón secundario del ratón. Cada tipo de nodo del árbol despliega un menú distinto con las opciones relativas a ese nodo:

1. Menú nodo raíz: Permite añadir nuevos dominios, mostrando la ventana Insertar un nuevo dominio.
2. Menú nodo principal de un dominio: Está formado por las opciones Renombrar, Eliminar y Modificar dominio.
3. Menú nodo tipo base: Permite modificar el dominio a través de la ventana Editar el dominio, en la que se puede cambiar el tipo base y los valores que lo componen.
4. Menú nodo valor: Además de permitir modificar el dominio al igual que el menú anterior permite ordenar los valores de manera lexicográfica.

6.4. Controlador.

El controlador es el núcleo principal de la herramienta.

Se podría considerar su función realizando el símil con la estructura de un Sistema Operativo modelo cliente-servidor, también llamado micronúcleo, en el caso del *kernel* de dicho sistema.

El controlador tiene dos funciones principales:

1. Inicializar todos los componentes de la aplicación.
2. Controlar el paso de mensajes y coordinación de los mismos.

Todos los componentes, o módulos, de la aplicación son inicializados y mantenidos en memoria desde el arranque de la misma. La motivación de ese comportamiento viene derivada de una

estabilidad general por parte de la aplicación, ya que una vez arrancada se tiene total seguridad de su fiabilidad y comportamiento.

Es el Controlador el que realiza la acción de arranque, y el módulo al que se llama desde el arranque de la herramienta y, por lo tanto, tiene la función definida de poner en marcha toda la maquinaria del sistema.

Todos los mensajes del sistema pasan por el módulo Controlador, según el esquema Vista-Controlador expuesto con anterioridad. Ello significa que el Controlador es capaz de enrutar cualquier petición entrante y realizar el desarrollo que de ella deriva.

Aunque el código de este módulo pudiera ser simplificado, o eliminado, en una implementación directa entre las demás capas, esta decisión mejora enormemente la escalabilidad del software diseñado y limita las interfaces y funcionalidades del resto de módulos de una forma tajante.

Para la selección de mensajes posibles ha sido declarada una clase enumerada TC que almacena el tipo de mensaje, tanto entrante como saliente, que puede manejar el módulo Controlador.

No se adjunta ninguna pieza de código de la clase expuesta debido a la extensa cantidad de información que maneja. Si se desea puede consultar su implementación concreta en la clase **Controlador.java**.

6.5. Servicios de aplicación.

Los servicios de aplicación se encargan de la implementación de todos los requisitos funcionales anteriormente descritos. DBCASE tiene 4 tipos de servicios, agrupando en cada uno de ellos la funcionalidad de cada uno de los módulos de la aplicación.

Estos servicios son los siguientes:

1. Servicios de entidades
2. Servicios de atributos.
3. Servicios de relaciones.
4. Servicios de dominios.
5. Servicios de sistema.

Los servicios de aplicación son solicitados por el controlador y hacen uso de la capa de persistencia.

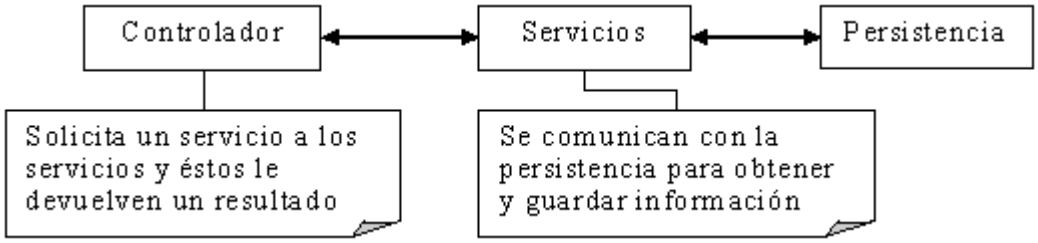


Figura 6. 11

Es importante resaltar el hecho de que tener realizada la especificación de los requisitos funcionales del sistema previamente a la implementación, ha hecho que muchos de los servicios implementados en cada uno de los módulos sean una mera traducción a JAVA de los mismos.

A continuación explicamos los servicios de dominios y del sistema. El primero por ser el implementado por nosotros y similar a los de entidades, atributos y relaciones. El segundo por su complejidad y por haber sufrido muchas ampliaciones.

6.5.1. Servicios de dominios.

Los servicios de dominios se centran en la implementación de los requisitos funcionales relacionados con éstos. La implementación se encuentra en el fichero **ServiciosDominios.java**.

A la hora de implementar los servicios de dominios hemos hecho una traducción casi literal de la especificación de los requisitos funcionales del módulo dominios. Son los siguientes:

```
public void anadirDominio(TransferDominio td)
public void renombrarDominio(Vector v)
public void eliminarDominio(TransferDominio td)
public void modificarDominio(Vector<Object> v)
public void modificarElementosDominio(Vector<Object> v)
```

Figura 6. 12

Si se desea conocer la implementación concreta de estos métodos, consultar el fichero fuente.

Otros métodos que tienen los servicios de dominios es:

```
public void ListaDeDominios()
```

Figura 6. 13

Este método accede a la persistencia mediante el DAODominios y devuelve al controlador una lista con todos los dominios del proyecto actual.

6.5.2. Servicios de sistema.

Los servicios de sistema, tal y como se han especificado anteriormente, se centran en:

- La validación del diseño realizado por el usuario.
- La generación del modelo relacional extraído del diseño.
- La generación de código en lenguaje SQL. Debido a las diferencias entre distintos sistemas gestores de bases de datos, permite generar scripts adaptados especialmente para MySQL, Oracle y Java.
- La ejecución de dichos scripts en los sistemas gestores de bases de datos indicados.

La implementación de estos servicios se encuentra en el fichero **ServiciosSistema.java**.

Cabe destacar que se han seguido ciertos criterios de validación, teniendo en cuenta optimizaciones del diseño y sobre todo las restricciones básicas para un buen funcionamiento de la base de datos resultante.

Los servicios del sistema, como se ha podido comprobar en la especificación, acceden a los datos del proyecto en curso a través de objetos DAO (patrón arquitectónico ya comentado en otra sección de este documento).

6.5.2.1. Validación del diseño.

La validación del diagrama Entidad Relación, se realiza mediante el recorrido directo del fichero de datos. La idea principal es ir comprobando uno a uno los objetos del modelo, comenzado por los dominios, siguiendo con atributos y entidades y terminando con las relaciones.

Gracias a esta estructuración, los servicios de sistema devuelven secuencialmente mensajes al controlador con el resultado de cada uno de los pasos de validación.

Se han implementado tres tipos de mensajes resultantes al comprobar las características del diseño. Además estos mensajes van acompañados de un comentario informativo que especifica la situación concreta que ha tenido lugar. Son los siguientes:

- **Éxito**: El resultado del paso de validación es satisfactorio.
- **Error**: El resultado del paso de validación no cumple el criterio tratado. Interrumpe inmediatamente el proceso de validación.
- **Aviso**: El resultado del paso de validación cumple los criterios pero se detecta alguna posible debilidad del diseño.

Ejemplos de estos tres tipos de mensajes son ilustrados por las siguientes imágenes:

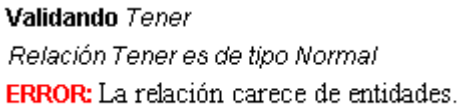


Figura 6. 14

Al validar la entidad Paciente, el sistema detecta que la entidad no tiene definida ninguna clave primaria.

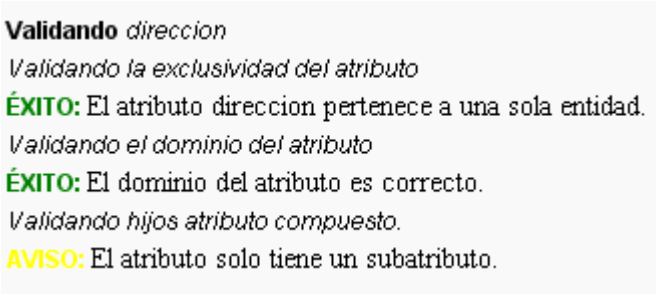


Figura 6. 15

El mensaje de aviso se corresponde con un atributo que está definido como compuesto y únicamente tiene un componente.

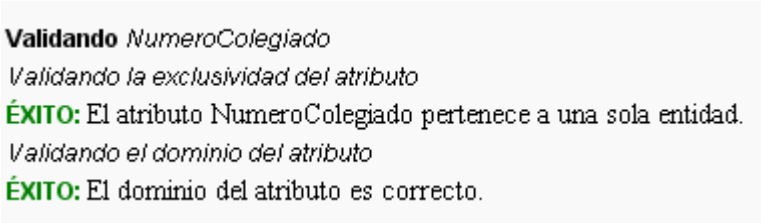


Figura 6. 16

En esta imagen se observa que la validación del atributo es satisfactoria en todos los casos.

6.5.2.1.1. Validación de dominios

Los primeros elementos en ser validados son los dominios creados por el usuario, si los hay. Para cada uno de ellos se comprueba que no está repetido, que su conjunto de valores posibles no es vacío, y si es utilizado en el resto del diseño (no cumplir esta última condición no provoca un error de validación, pero sí un aviso).

6.5.2.1.2. Validación de atributos.

Los atributos del diseño han de cumplir determinadas condiciones para el futuro funcionamiento del sistema. Se han implementado funciones específicas para cada una de dichas condiciones.

```
private boolean validaDominio(TransferAtributo ta)
private boolean validaFidelidadAtributo(TransferAtributo ta)
private boolean validaCompuesto(TransferAtributo ta)
```

Figura 6. 17

Los posibles roles de un atributo en un modelo E/R son variados. Puede ser un atributo multivalorado y/o compuesto, puede pertenecer a una entidad, a una relación o ser un subatributo de otro. Todas estas posibilidades se han tenido en cuenta a la hora de comprobar la corrección del diseño.

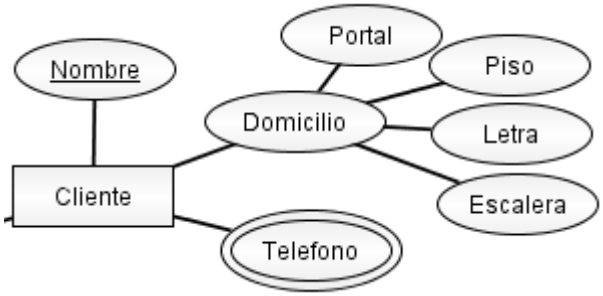


Figura 6. 18

Cabe destacar, que ciertas características que podrían considerarse propias de los atributos, como el carácter de clave de una entidad por ejemplo, son comprobadas en la fase de validación de entidades dado que se ha considerado que resulta mas cómodo y fiable debido a la estrategia seguida para el almacenamiento de los datos.

Como primer parámetro a controlar está lo que hemos denominado fidelidad del atributo. El objetivo de esta comprobación es asegurarse de que un atributo concreto no esta repetido en varias entidades o relaciones. Hemos de precisar que al hablar de repeticiones no nos referimos a atributos que tengan el mismo nombre, sino a una posible variación a nivel de acceso a los datos a través de los cuales se organiza el sistema.

La segunda comprobación que tiene lugar en la validación de atributos se centra en controlar el dominio de éstos. La idea básica es asegurarse de que los dominios introducidos por el usuario existen en el sistema. Además, cuando se trata un atributo compuesto, por convenio en el diseño de la aplicación se considera que el atributo padre deberá tener dominio nulo comprobándose en consecuencia los tipos de los hijos.

Precisamente sobre este tipo de atributos, los compuestos, se centra el último de los criterios de validación de esta sección.

El objetivo de estas comprobaciones es controlar el número de hijos que posee dicho atributo. Al ser compuesto, ha de tener al menos un hijo, con lo que si se detecta que esto no sucede, la validación lo considerará un error. Si el número de hijos es exactamente uno, se considerara correcto, pero el servicio de sistema enviara un mensaje de aviso indicando una situación potencialmente incómoda.

6.5.2.1.3. Validación de entidades.

Análogamente a los atributos, las entidades pueden cumplir ciertos roles en modelo ER. Pueden ser débiles o fuertes y esto influye en la validación de cada una de ellas.

```
private boolean validaKey(TransferEntidad te)
private boolean compruebaClaveCompuesto(Vector clavesEntidad,TransferAtributo ta)
private boolean validaNombresAtributosEntidad(TransferEntidad te)
private void validaFidelidadEntidadEnIsA(TransferEntidad te)
```

Figura 6. 19

La primera comprobación hecha sobre una entidad es si tiene atributos. En caso de no tener ninguno, se producirá un error de validación.

La segunda validación que se realiza está centrada en las claves de la entidad. Una entidad fuerte que esté vinculada a una relación esta obligada por diseño a tener un atributo clave. Salvo ciertas excepciones, como sería pertenecer a una relación de herencia, el sistema dará un error en el modelo E/R si no se cumple esta condición. En caso de que la entidad no tenga clave primaria, pero no esté vinculada a ninguna relación, se dará un aviso de la situación.

Surgen situaciones variadas para asociar una clave a una entidad. Por ejemplo, si se da el caso de que un atributo compuesto es clave, se consideran claves todos sus hijos. Otro caso particular es la posibilidad de haber declarado como clave un atributo multivalorado. Esto el sistema lo considera inaceptable, ya que no es un atributo que distinga inequívocamente una entidad de otra.

La segunda validación que se realiza en las entidades se trata de controlar los nombres de sus atributos. Devuelve un error si encuentra atributos con el mismo nombre dentro de una misma entidad.

Como tercera validación, comprueba la intervención de la entidad en relaciones de herencia. Se procura controlar que dicha entidad sea padre solo de una relación de herencia, pero en caso contrario, solo da un aviso dado que no constituye un error en si.

6.5.2.1.4. Validación de relaciones.

Sin lugar a dudas, las comprobaciones más complejas se realizan con las relaciones. Hemos tenido que controlar el papel de cada una de las entidades que intervienen dependiendo del tipo de relación implicada.

Al igual que las entidades y los atributos, las relaciones pueden actuar con distintos roles: débiles, normales o IsA (relaciones de herencia). Para cada uno de estos papeles hay un método en los servicios de sistema que evalúa su corrección.

Es de destacar que, como se explica en la implementación de la capa de persistencia de datos, se recurre a una clase llamada **EntidadYAridad.java** que almacena cada grupo de entidades que participan en una relación y la aridad con la que intervienen.

```
private boolean validaComponentesRelacionIsA(TransferRelacion tr)
private boolean validaComponentesRelacionNormal(TransferRelacion tr)
private boolean validaComponentesRelacionDebil(TransferRelacion tr)
```

Figura 6. 20

6.5.2.1.5 Validación de relaciones de herencia.

Las comprobaciones básicas que se tienen en cuenta para este tipo de relaciones están basadas en los papeles de las entidades que intervienen.

Se comprueba el numero de hijos que posee la relación y obviamente la existencia de un padre.

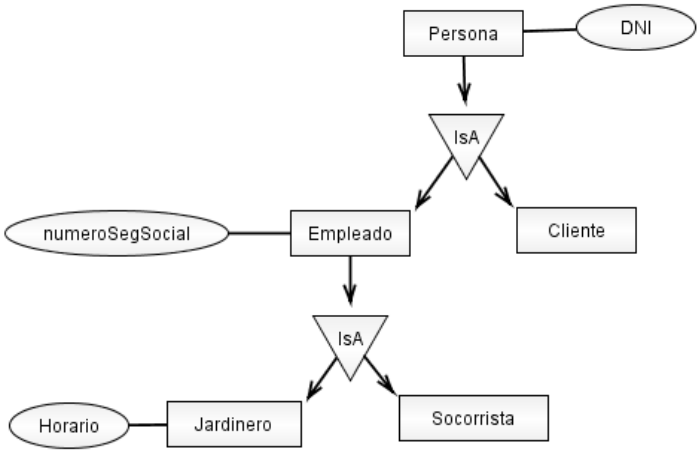


Figura 6. 21

6.5.2.1.6. Validación relaciones normales.

Esta es la validación más sencilla. Para este tipo de relaciones, la aplicación comprueba que existan al menos dos entidades relacionadas (puede ser la misma relación dos veces, con distintos roles).

6.5.2.1.7. Validación de relaciones débiles.

Las relaciones débiles deben unir una única entidad débil con una única entidad fuerte. La cardinalidad es siempre de 1 a n para la entidad fuerte, y de 1 a 1 para la entidad débil. El usuario no puede modificar estos valores, por lo que no sería necesario validarlo, pero se comprueba por si el fichero XML ha sido modificado sin tener en cuenta esta restricción.

Figura 6. 22

6.5.2.2. Generación del modelo relacional

Esta segunda parte de los servicios del sistema se centra en la generación del Modelo Relacional asociado.

El objetivo es determinar qué tablas serán necesarias para implementar el modelo entidad relación. Para poder desarrollar esta implementación se ha utilizado la clase **Tabla.java** que almacena los valores de las claves (primarias y foráneas), atributos etc.

6.5.2.2.1. Clase Tabla.java.

Cada objeto de esta clase tiene como atributos: el nombre de la tabla, la lista de atributos, la de claves primarias y la de foráneas.

```
private String nombreTabla;
private Vector<String[]> atributos;
private Vector<String[]> primaries;
private Vector<String[]> foreigns;
```

Figura 6. 23

Cada una de estas listas esta implementada como un vector de arrays de String. Dentro del código estos arrays tienen un tamaño definido de 3 componentes, el nombre del atributo, su dominio y el nombre de la tabla de donde procede. Así, para la creación de tablas que requieran claves foráneas, se puede acceder a esa información fácilmente.

Ejemplo:

```
Atributos[0] = DNI;
Atributos[1] = Char(10);
Atributos[2] = Cliente;
```

Figura 6. 24

Alguna de esta información es irrelevante para el modelo relacional, pero sí es útil para la generación de código SQL.

Para obtener el Modelo Relacional se ha implementado otro método que utiliza la estructura de la tabla.

```
public String modeloRelacionalDeTabla()
```

Figura 6. 25

Esta clase detecta y solventa las situaciones con tablas con nombres de atributos comunes y la corrección de nombres con guiones medios y espacios en blanco. Este último detalle es bastante importante debido a que los clientes SQL más habituales no aceptan este tipo de escritura.

Una vez explicada la clase tabla, podemos comentar la implementación de la sección de los servicios de sistema.

Los servicios poseen cuatro tablas hash que almacenan cada objeto tabla resultante. Como clave de cada una de estas tablas hash se utiliza el identificador exclusivo en el sistema del elemento implicado.

```
private Hashtable<Integer,Tabla> tablasEntidades=new Hashtable<Integer,Tabla>();
private Hashtable<Integer,Tabla> tablasRelaciones=new Hashtable<Integer,Tabla>();
private Vector<Tabla> tablasMultivalorados=new Vector<Tabla>();
private Hashtable<Integer,Enumerado> tiposEnumerados = new Hashtable<Integer,Enumerado>();
```

Figura 6. 26

6.5.2.2.2. Tablas de dominios

Algunos gestores de bases de datos facilitan herramientas para establecer dominios, pero otros no. Por ello, se ha buscado un forma de representarlos lo más genérica posible. Para ello, se crea una tabla con el nombre del tipo en cuestión. Esta tabla tiene una sola columna, de tipo varchar, llamada value_list, y que es clave primaria. En ella, se insertan todos los valores disponibles en el tipo enumerado.

Posteriormente, cada atributo declarado del tipo enumerado, será marcado como varchar, y se le establecerá como clave foránea de la tabla creada anteriormente. En la versión anterior existía la limitación de que una clave foránea debía llamarse igual que la variable a la que referenciaba, ya que antiguamente sólo se utilizaba esta restricción a la hora de vincular entidades a relaciones. Con la llegada de los dominios, es necesario modificar este apartado, ya que si mantenemos la condición perdemos poder semántico.

6.5.2.2.3. Tablas de entidades

En este caso, la traducción a tabla es trivial. Para cada atributo, es necesario indicar su tipo (o dominio) y si es un valor not null.

También deben establecerse las restricciones que afectan a conjuntos de atributos: clave primaria, clave secundaria y grupos de atributos únicos.

6.5.2.2.4. Tablas de relaciones

En las relaciones normales, la cardinalidad establece las claves primarias. Para ello, se toma la siguiente política:

Se considera que sólo hay 2 tipos de cardinalidad:

- 0..1
- 0..n

El resto de tipos se considera de la siguiente manera:

- 1..1 ~ 0..1
- a..b ~ 0..n; siendo b =/= n

Para decidir la cardinalidad se toman las siguientes decisiones:

- Si la entidad participa en la relación con una cardinalidad de 0..n, los miembros de la clave primaria de la entidad serán también clave primaria de la relación.
- Si la entidad participa con una cardinalidad de 0..1, los miembros de su clave no serán clave primaria en la relación.
- Caso particular: Varias entidades participan con cardinalidad 0..1. Cada conjunto de atributos formando la clave primaria de una entidad debería ser clave primaria de la relación, independientemente del resto. Como esto no es posible, se establece que la clave sea un conjunto al azar (el primero), y el resto no sean claves, pero su combinación de valores debe establecerse como unique.

En la generación de tablas para relaciones IsA, para que un objeto hijo de una relación hija tenga sentido, es necesario que haya una entrada en la tabla padre, y otra más en la tabla hija, que haga referencia a la tabla padre. Por ello, en la tabla hija se deben añadir los campos que formen la clave primaria de la tabla padre, y ser clave primaria en ésta también. Además, para guardar la integridad referencial deben ser clave foránea de la tabla padre. Si el vínculo entre una entidad y un rol está nombrado (posee un rol), se debe reflejar en la tabla resultante. Para ello, la clave primaria de la entidad se añade a la lista de atributos de la relación precedida por el nombre del rol.
Este hecho provoca que haya atributos que estén vinculados a otros en otras tablas que no tienen el mismo nombre. Hasta ahora las referencias foráneas siempre cumplían esta condición, por lo que sólo era necesario almacenar a qué tabla hacía referencia el atributo. Para corregir esto, es necesario almacenar tanto la tabla como el atributo al que hace referencia.

6.5.2.3. Generación y ejecución de código SQL

Debido a que una de las mayores modificaciones del proyecto ha sido la generación de código SQL adaptado a distintos sistemas gestores de bases de datos, toda la lógica encargada de la generación de código ha sido replanteada.

En primer lugar, la clase Tabla ha dejado de ser la encargada de generar su propio código SQL para dar paso a la clase ConectorDBMS, la cual conoce la sintaxis admitida por el gestor al que se conecta. Y es capaz de generar scripts a partir de la tabla dada.

Se ofrecen distintas conexiones a los gestores de bases de datos más significativos. La gestión de las mismas está hecha siguiendo el patrón de factoría, de manera que sea sencillo añadir nuevos gestores de bases de datos, ya que el código referente a estas tareas permanece transparente al resto de la aplicación. Todas las clases referentes a este apartado están contenidas en el paquete

DBCASE

Utilidades.ConectorDBMS.
En este patrón hay dos elementos especialmente significativos: la factoría de conectores, y el interfaz que debe seguir cada conector.

6.5.2.3.1. Factoría de conectores

La clase FactoriaConectores tiene dos funciones principales:

- Facilitar la lista de conexiones disponibles.
- Dado el identificador de una conexión, obtener su conector.

6.5.2.3.2. Interfaz de conector

El interfaz ConectorDBMS determina el comportamiento que deben seguir todas las conexiones contempladas por la aplicación. Los tipos de funciones contenidos en esta interfaz se dividen en dos tipos:

Conexión a la base de datos

- **abrirConexion (String cadenaDeConexion, String user, String password)**: Establece una conexión con el gestor determinado en cadenaDeConexion, con el usuario user y la contraseña password.
- **usarDatabase(String nombre)**: En la conexión ya abierta, destruye la tabla con el nombre dado (si existe), y crea una nueva vacía, lista para ser rellenada con la ejecución del script
- **cerrarConexion()**: Libera los recursos empleados para efectuar la conexión. El método abrirConexion debe haber sido ejecutado con éxito previamente.
- **ejecutarOrden (String ordenSQL)**: Ejecuta un comando de actualización en el gestor de base de datos. Estos comandos pueden ser:
    - CREATE
    - DROP
    - INSERT
    - UPDATE

La secuencia a seguir a la hora de realizar una conexión es la siguiente:

- abrirConexion (_, _, _)
- usarDatabase (_)
- Para cada instrucción del script, ejecutarOrden(inst)
- cerrarConexion()

Las instrucciones se envían una a una al sistema gestor de bases de datos. Los comentarios y las líneas en blanco son eliminados previamente, para evitar posibles conflictos al ejecutarlos en el gestor.

Generación de código

- **obtenerCodigoCreacionTabla (Tabla t)**: Genera el código necesario para crear la tabla t en el gestor de bases de datos.
- **obtenerCodigoCreacionTablaHTML (Tabla t)**: Igual al anterior, pero el texto generado está coloreado.
- **obtenerCodigoClavesTabla (Tabla t)**: Genera el código necesario para incluir en la tabla t sus restricciones (clave primaria, claves foráneas y conjuntos Unique).
- **obtenerCodigoClavesTablaHTML (Tabla t)**: Igual al anterior, pero el texto generado

está coloreado.

- **obtenerCodigoEnumerado (Enumerado e)**: Genera el código necesario para implementar el tipo enumerado dado.
- **obtenerCodigoEnumeradoHTML (Tabla t)**: Igual al anterior, pero el texto generado está coloreado.

6.5.2.3.3. Conectores implementados

Los sistemas gestores de bases de datos implementados actualmente para la aplicación son los siguientes:

SGBD	Conector usado	Fichero .jar importado
MySQL	MySQL Connector/J 5.1 driver	mysql-connector-java-5.1.6-bin.jar
Oracle	Oracle Database 11*g* Release 1 (11.1.0.7.0) JDBC Drivers	ojdbc6.jar
MS Access (Con. ODBC)	Sun Jdbc Odbc driver	Nativo de Java 6.0
MS Access (Fichero MDB)	Sun Jdbc Odbc driver	Nativo de Java 6.0

Pese a que los sistemas gestores de bases de datos elegidos utilizan el estándar SQL, no todos los tipos son iguales. Por ello, elegimos los tipos más significativos. Es tarea del gestor de datos decidir si es necesario traducir esos tipos básicos a otros más adecuados.

Tipo genérico	MySQL	Oracle	MS Access
integer	integer	integer	integer
float	real	real	double
bit	bit	char(1)	yesno
date	date	date	datetime
time	time	time	datetime
datetime	datetime	timestamp	datetime
blob	blob	blob	image
char	char	char	char
varchar	varchar	varchar2	varchar
text	text	clob	memo
decimal	decimal	number	currency

6.6. Persistencia

La persistencia de los datos en DBCASE esta basada en el patrón DAO con acceso a ficheros XML. En esta sección explicaremos cómo se gestionan esos datos y cual es su estructura concreta en este fichero.

Además, en esta aplicación existe un identificador numérico exclusivo para cada una de las entidades, relaciones o atributos residentes en el sistema, garantizando así el acceso inequívoco éstos.

Para mas información sobre el funcionamiento del patrón DAO, se puede consultar la sección de esta documentación dedicada a los patrones utilizados.

6.6.1. XML (Extensible Markup Language).

XML es un metalenguaje estructurado que permite almacenar información de manera organizada. Esta información puede ser fácilmente compartida con cualquier software siempre que se tenga un analizador adecuado.

La estructura de un fichero XML es sencilla y jerárquica. Esta basada en etiquetas, similares a HTML, que se encuentran anidadas con el contenido correspondiente dentro de ellas. Para que un documento este bien formado y no de lugar a errores, todas las etiquetas deben de estar cerradas y además debe de existir un único elemento raíz del que todos los demás serán parte.

Es posible configurar la codificación de los datos que se van a almacenar para adaptarlos a distintos alfabetos o lenguajes.

A continuación, mostraremos como son los ficheros XML con los que trabaja DBCASE.

6.6.2. Fichero XML en DBCASE

El almacenamiento de la información en esta aplicación está organizado mediante un único fichero codificado con el estándar UTF-8. En la versión anterior de la herramienta un proyecto se almacenaba como tres ficheros XML codificados en el estándar ISO-8859-1. La inclusión de toda la información en un sólo fichero hace de la gestión de los proyectos una tarea más sencilla y cómoda. Un proyecto era antes un conjunto de ficheros incluidos en un directorio *área de trabajo* y ahora un sólo fichero.

El cambio de ISO-8859-1 a UTF-8 ha sido propiciado por la globalización de la herramienta. La codificación anterior no daba soporte a los caracteres especiales de cada idioma, lo que propiciaba errores al guardar elementos con caracteres propios de cada idioma, como las tildes y las eñes en castellano.

El fichero que almacena un proyecto está dividido en cuatro campos principales referentes a las listas de entidades, relaciones, atributos y dominios:

6.6.2.1. Lista de entidades

Es una lista de entidades cuyo atributo es el próximo identificador a asignar a la siguiente entidad que ingrese en el sistema (esta parte es análoga en las listas de relaciones, atributos y dominios). De cada entidad almacenamos:

- **Nombre**: El nombre representativo de la entidad.
- **Carácter de débil**: Indica si la entidad es débil o no.
- **Lista de atributos**: Una lista de identificadores. Cada uno indica inequívocamente a un atributo del mismo fichero.
- **Lista de claves primarias**: Una lista de identificadores. Cada uno indica inequívocamente a un atributo de la entidad.
- **Lista de restricciones:** Lista con las restricciones de la entidad.
- **Lista de únicos:** Lista de los atributos únicos de la entidad. Los campos Uniques formados por varios atributos representan que la restricción afecta al conjunto.
- **Posición:** Coordenadas que ocupa en nodo en el diagrama.

A continuación mostramos un ejemplo del campo EntityList de un posible fichero de DBCASE.

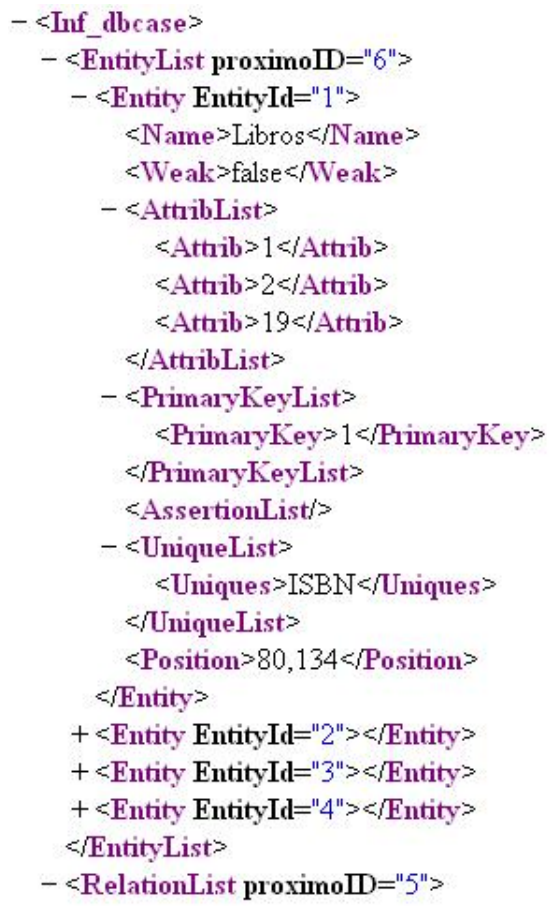


Figura 6. 27

6.6.2.2. Lista de relaciones

Es una lista de relaciones. El próximo identificador a asignar está almacenado en dicha lista y los campos son los siguientes:

- **Nombre**: El nombre representativo de la relación.
- **Tipo**: Almacena el tipo de relación del que se trata: débil, normal o de herencia (IsA).
- **Lista de entidades y aridades**: Almacena una lista con los identificadores de cada una de las entidades participantes en la relación y la aridad con la que se implican. Más adelante se mostrara el uso de la clase entidadYAridad.java.
- **Lista de atributos**: Una lista de identificadores de atributos.
- **Lista de restricciones:** Lista con las restricciones de la entidad.
- **Lista de únicos:** Lista de los atributos únicos de la relación. Los campos Uniques formados por varios atributos representan que la restricción afecta al conjunto.
- **Posición:** Coordenadas que ocupa en nodo en el diagrama.

A continuación mostramos un ejemplo del campo RelationList de un fichero de la aplicación.

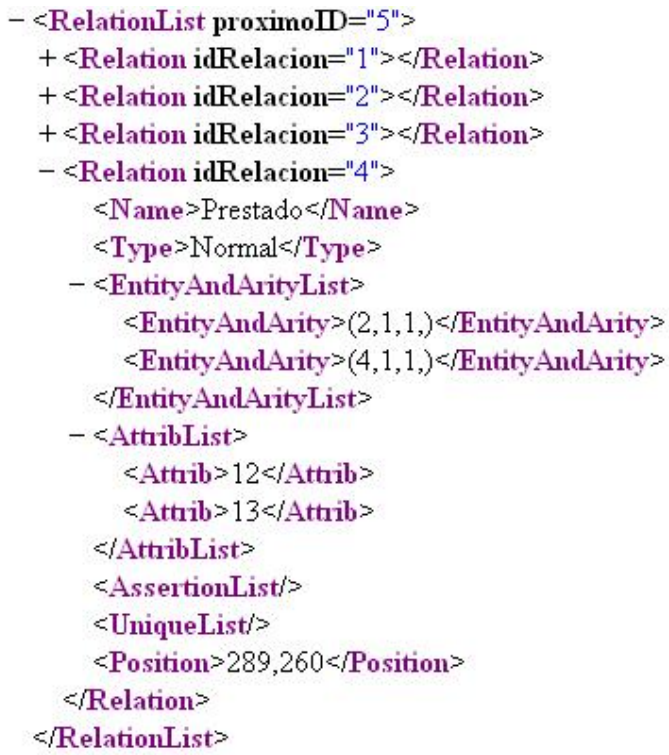


Figura 6. 28

6.6.2.3. Lista de atributos

Es una lista de atributos. Análogamente a las entidades y relaciones, lleva el control sobre el próximo identificador de atributo a aplicar. Los campos de cada atributo son los siguientes:

- **Nombre**: El nombre representativo del atributo.
- **Dominio**: El dominio del atributo
- **Carácter de compuesto**: Un valor booleano que indica si es un atributo compuesto o no.
- **No nulo**: Un valor booleano que indica si posee la propiedad Not null.
- **Único**: Valor booleano que indica si el atributo debe ser único.
- **Lista de Componentes**: En caso de que sea un atributo compuesto, aquí se almacenan los identificadores de los subatributos que tiene asociados.
- **Multivalorado**: Un valor booleano que indica si el atributo es multivalorado
- **Lista de restricciones:** Lista con las restricciones del atributo.
- **Posición:** Coordenadas que ocupa en nodo en el diagrama.

A continuación mostramos un extracto de un posible fichero XML con la información relativa a algunos atributos.

```
− <AttributeList proximoID="20">
  + <Attribute AttributeId="1"></Attribute>
  + <Attribute AttributeId="16"></Attribute>
  − <Attribute AttributeId="19">
      <Name>Tipo</Name>
      <Dom>Libro</Dom>
      <Composed>false</Composed>
      <NotNull>false</NotNull>
      <Unique>false</Unique>
      <ComponentList/>
      <Multivalued>false</Multivalued>
      <AssertionList/>
      <Position>189,95</Position>
    </Attribute>
</AttributeList>
```

Figura 6. 29

6.6.2.4. Lista de dominios

Es una lista que contiene los dominios creados por el usuario. Análogamente a las otras listas, lleva el control sobre el próximo identificador de dominio a aplicar. Los campos de cada atributo son los siguientes:

- **Nombre**: El nombre representativo del dominio.
- **Tipo Base**: El dominio al que pertenecen los valores enumerados, es siempre un dominio de entre los predefinidos.
- **Lista de Valores**: Lista de los valores que podrán tomar los atributos don este dominio.

A continuación mostramos un extracto de un posible fichero XML con la información relativa a un dominio.

```
− <DomainList proximoID="2">
  − <Domain DomainId="1">
      <Name>Libro</Name>
      <BaseType>VARCHAR</BaseType>
    − <ValueList>
        <Value>'Novela'</Value>
        <Value>'Teatro'</Value>
        <Value>'Ensayo'</Value>
        <Value>'Relatos'</Value>
        <Value>'Guía'</Value>
      </ValueList>
    </Domain>
  </DomainList>
</Inf_dbcase>
```

Figura 6. 30

6.6.3. Implementación de los DAOs

Como sabemos, el patrón DAO nos propone implementar métodos de acceso y modificación a los datos. En el caso de esta aplicación estos datos están almacenados, como comentábamos en el apartado anterior, en un fichero XML.

Existe una clase DAO para cada una de las listas de los ficheros del sistema y cada una de estas clases tiene cinco métodos comunes: añadir, consultar, modificar, borrar y listar.

Además existe una clase llamada entidadesYAridades.java que ayuda a organizar las entidades implicadas en una relación.

Para poder utilizar la información del archivo XML, cada uno de los DAOs posee dos métodos de apertura y cierre del documento. Estos métodos recurren a la clase DocumentBuilderFactory que proporciona un parser para el procesamiento del fichero.

```
private Document dameDoc()
private void guardaDoc() throws IOException
```

Figura 6. 31

La implementación de cada uno de estos métodos es la siguiente:

```
private Document dameDoc() {
    Document doc = null;
    DocumentBuilder parser = null;
    try {
        DocumentBuilderFactory factoria = DocumentBuilderFactory.newInstance();
        parser = factoria.newDocumentBuilder();
        doc = parser.parse(this.path);
    }
    catch (Exception e) {
        JOptionPane.showMessageDialog(
                null,
                Lenguaje.getMensaje(Lenguaje.ERROR)+":\n" +
                Lenguaje.getMensaje(Lenguaje.UNESPECTED_XML_ERROR)+" \""+ path+".xml\"",
                Lenguaje.getMensaje(Lenguaje.DBCASE),
                JOptionPane.ERROR_MESSAGE);

    }
    return doc;
}

private void guardaDoc() throws IOException {
    OutputFormat formato = new OutputFormat(doc, "utf-8", true);
    StringWriter s = new StringWriter();
    XMLSerializer ser = new XMLSerializer(s, formato);
    ser.serialize(doc);
    // El FileWriter necesita espacios en la ruta
    this.path = this.path.replace("%20"," ");
    FileWriter f = new FileWriter(this.path);
    this.path = this.path.replace(" ", "%20");
    ser = new XMLSerializer(f, formato);
    ser.serialize(doc);
}
```

Figura 6. 32

Después de obtener la variable doc, que nos permite la lectura y escritura del fichero XML, recurrimos a la clase Node, que nos proporciona cada uno de los elementos que se obtienen al procesar el documento. Lógicamente podemos obtener listas de nodos con la clase NodeList.

Existen una amplia gama de sentencias para la modificación, eliminación e inserción de información. Cabe destacar que toda la información que la aplicación introduce en el XML esta en formato texto y al recuperarla se realiza el casting adecuado para cada caso.

Todos los métodos principales de los DAOs, devuelven o requieren un objeto Transfer, exceptuando la función de listado que devuelve una lista de Transfer.

6.6.4. Clase EntidadYAridad.java

Esta clase sirve como ayuda auxiliar para almacenar la información sobre la participación de las entidades en las relaciones. Posee cuatro atributos: El identificador de la entidad, el principio del rango de la aridad y el final de éste y el rol con el que participa la entidad en la relación; este último atributo puede ser vacío.

```
int entidad;
int pRango;
int fRango;
String rol;
```

Figura 6. 33

Además los objetos de esta clase poseen métodos que permiten la interpretación de su información desde los ficheros XML. Así, cuando se introducen rangos totales (por ejemplo 1..n), el sistema es capaz de asimilarlos y almacenarlos correctamente.

6.7. Globalización.

6.7.1 Introducción.

El objetivo de la globalización es internacionalizar la aplicación, de forma que el idioma o los códigos de colores, no sea un impedimento para su uso por parte de personas de otras culturas.

La globalización de la herramienta fue desde el principio uno de nuestros objetivos principales. Ya que la primera versión estaba sólo disponible en castellano, el número de usuarios potenciales se limitaba considerablemente.

Se trataba no sólo de hacer que la aplicación estuviese disponible en varios idiomas, si no que además fuese fácil añadir nuevos lenguajes, sin necesidad de tocar el código fuente de la aplicación.

6.7.2 Implementación.

Se ha incluido la clase Lenguaje.java que es la encargada de hacer el *parsing* de los documentos que contienen los diferentes lenguajes, así como de dar las funcionalidades de lenguaje a la aplicación. Para ello, implementa un diccionario en el cual almacena todos los mensajes de texto del idioma actual, pudiendo acceder a ellos a través del nombre de identificador. Para asegurar que los mensajes solicitados existen, se añade un enumerado con todos los posibles identificadores.

A continuación mostramos dos secciones del código de esta clase, la primera del enumerado que contiene los identificadores y la segunda de la función getMensaje, que devuelve la cadena de texto correspondiente al identificador, en el lenguaje seleccionado.

```
//--- --- --- MENSAJES GENERALES --- --- ---
public static final int SELECT = 11;
public static final int YES = 12;
public static final int NO = 13;
public static final int INFO = 14;
public static final int ERROR = 15;
public static final int WISH_CONTINUE =16;
```

Figura 6. 34

```
public static String getMensaje(int tipoMensaje) {
    String texto;

    switch (tipoMensaje){

    case SELECT: texto = _textos.get("select"); break;
    case YES: texto = _textos.get("yes"); break;
    case NO: texto = _textos.get("no"); break;
    case CANCEL: texto = _textos.get("cancel");break;
    case INSERT: texto = _textos.get("insert");break;
    case INFO: texto = _textos.get("info");break;
    case ERROR: texto = _textos.get("error");break;
    case SUCCESS: texto = _textos.get("success");break;
    case WARNING: texto = _textos.get("warning");break;
    case WISH_CONTINUE: texto = _textos.get("wishContinue");break;
```

Figura 6. 35

6.7.3. Incluir un nuevo lenguaje.

Para añadir un idioma a DBCASE, se debe añadir un nuevo fichero de texto, con el nombre del lenguaje y extensión .lng, al directorio del proyecto llamado languages, el cual contiene los siguientes ficheros:

- **index.txt**:

  Este fichero almacena una lista de todos los lenguajes disponibles. Cada una de sus líneas debe tener la forma:

  fichero.lng=Lenguaje del fichero

  Para incluir el nuevo lenguaje se debe añadir una entrada en index.txt con el formato indicado.

  Siempre hay al menos un fichero .lng en la lista, cuyo nombre es default.lng, y almacena el lenguaje cargado por defecto por la aplicación.

  Las líneas que comienzan con el carácter # son tomadas como comentarios, y las líneas en blanco son ignoradas.

  Las dos entradas que contiene el índice son:

  default.lng=English
  spanish.lng=Español

- **Ficheros .lng:**

  Estos ficheros almacenan la traducción a los distintos lenguajes. Todos los ficheros .lng descritos en index.txt deben estar contenidos y bien definidos dentro de la carpeta languages.

  El formato de estos ficheros es el siguiente:

  Nombre del lenguaje

  identificadorMensaje1=Mensaje de texto número 1

  identificadorMensaje2=Mensaje de texto número 2

  ...
  identificadorMensajeN=Mensaje de texto número N

  Al igual que en el fichero anterior, las líneas que comienzan con el carácter # son tomadas como comentarios, y las líneas en blanco son ignoradas. Hay una restricción importante: antes del nombre del lenguaje no debe haber ninguna línea en blanco, aunque puede haber comentarios.

  Mostramos a continuación parte del contenido de default.lng, en el que se puede apreciar el nombre del idioma y una pequeña parte de la lista de mensajes, correspondientes a los de las figuras 6.34 y 6.35.

```
# WARNING: There cannot be empty lines before this information
English

#
# General messages
yes=Yes
no=No
select=Select
cancel=Cancel
insert=Insert
info=INFO.
error=ERROR
success=SUCCESS
warning=WARNING
wishContinue=Do you want to continue?
```

6.8. Bibliotecas externas utilizadas

Para la implementación de la herramienta se ha hecho uso de bibliotecas externas, o no declaradas y definidas por el usuario. Dichas bibliotecas simplifican, en la medida de lo posible, una implementación partiendo de cero y han hecho posible la mejora de componentes inherentemente internos a la aplicación diseñada.

Las bibliotecas utilizadas son:

- **JRE System Library (v. 1.6.0).** Es la librería que proporciona la máquina virtual de *Java* y contiene estructuras de datos, optimizadas en muy alto grado, que resultan útiles en la implementación de código. Su utilización es libre y, por lo tanto, permite la distribución de cualquier proyecto derivado de su uso de forma no privativa.

- **Collections Generic (v. 4.01)**. Esta librería es utilizada para definir transformaciones internas en el diseño de los esquemas relacionales. Su uso es derivado de la librería *JUNG*, que requiere dicha funcionalidad. La licencia que posee es *Apache License Version 2.0* y, por consiguiente, es de libre distribución.
- **Colt (v. 1.2.0).** Esta librería es utilizada de forma interna por la librería principal JUNG y no ha sido necesaria referencia alguna desde la implementación propia. La licencia que posee es *The Artistic License* y, al igual que la librería anteriormente expuesta, es de libre distribución.
- **Concurrent (v. 1.3.4)**. Esta librería es utilizada de forma interna por la librería principal *JUNG* y no ha sido necesaria referencia alguna desde la implementación propia. Esta librería no posee licencia explícita, pero su creador, *Doug Lea,* permite su libre uso en la página Web oficial del proyecto: http://g.oswego.edu/dl/classes/EDU/oswego/cs/dl/util/concurrent/intro.html
- **JUNG - Java Universal Network/Graph Framework (v. 2.0).** Librería utilizada para la representación gráfica de los esquemas relacionales. Ha sido necesaria una reimplementación de mucha funcionalidad ya introducida en la librería, pero cabe destacar su completitud tanto en el tratamiento de grafos como en su representación visual. La licencia que posee es *Berkeley Software Distribution (BSD) license* y permite una libre distribución.
- **Looks (v. 2.1.4)**. Looks es parte de la librería gráfica *Jigloo*. Esta librería visual mejora la claridad del contenido y el acabado gráfico de la aplicación, además de su diseño. El paquete *Looks* se ha utilizado para mejorar los efectos y objetos gráficos ofrecidos por Swing, la librería gráfica de Java.La licencia de distribución que posee es *CloudGarden.com SOFTWARE USER AGREEMENT*, la cual permite su distribución, incluso en instituciones académicas, si los desarrolladores no han sido contratados por las mismas. Enlace oficial del proyecto es el siguiente: http://www.cloudgarden.com/jigloo/
- **MySQL Connector/J 5.1 driver.** Conector utilizado para el acceso a gestores de bases de datos MySQL. Es el driver JDBC oficial proporcionado por MySQL. La licencia de esta librería es GPL, por lo que su utilización es libre. Se puede obtener más información acerca de esta librería en: http://dev.mysql.com/downloads/connector/j/5.1.html
- **Oracle Database 11*g* Release 1 (11.1.0.6.0) JDBC Drivers.** Conector utilizado para el acceso a gestores de bases de datos Oracle. Es el driver JDBC oficial proporcionado por Oracle. La licencia de esta librería es OTN(Oracle Technology Network Development and Distribution License). Permite el uso no comercial de la aplicación, aunque restringe su distribución a países y personas que no cumplan los requisitos de explotación de la legislación estadounidense. Se puede obtener más información acerca de esta librería en la dirección que se indica a continuación: http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc_111060.html

Se ha intentado, en todo momento, el uso de bibliotecas de libre acceso y libre distribución para eliminar los posibles límites que ello podría acarrear. La licencia bajo la que la herramienta se acoge, por lo tanto, será acorde a la más restrictiva de las expuestas previamente.

7. Manual de usuario.

7.1. Introducción.

Este manual pretende explicar las nociones básicas de interacción entre el usuario y la aplicación Data Base CASE.

El objetivo es que el usuario se familiarice con el funcionamiento del programa de una forma sencilla y sepa afrontar las tareas básicas que se pueden desempeñar en el mismo.

Toda la interacción entre el usuario y la aplicación se realiza de forma primaria con el ratón, y de forma secundaria con el teclado. El uso de los dos botones principales del ratón es el estándar, y la rueda permite hacer zoom sobre el panel de diseño. Esta cualidad resulta útil para esquemas relativamente grandes donde se desea tener una visión más amplia que la que ofrece el interfaz de partida.

El uso del teclado además de para rellenar los campos permite la interacción con algunas de las ventanas de la forma que se describirá más a delante.

A lo largo de la explicación de las diversas opciones de la herramienta se adjuntarán imágenes explicativas del funcionamiento de las mismas, para la comodidad del lector.

7.2. Requisitos del sistema.

El único requisito que solicita Data Base CASE es la instalación de la máquina virtual de Java de Sun Microsystems (JRE v. 1.6.0 o posterior).

La herramienta trabaja de forma correcta, y ha sido verificada, en entornos Windows, distribuciones GNU/Linux (Ubuntu y Debian) y MAC OS X.

7.3. Instalación y ejecución de la aplicación.

Dado el objetivo de realizar una aplicación multiplataforma, utilizable en distintos sistemas, no se ha realizado un instalador de la misma para ninguna arquitectura específica.

La aplicación se proporciona de forma auto-ejecutable contenida en un fichero JAR (Java ARchives). La instalación, por tanto, se completaría con la descarga de dicho fichero y su localización en un lugar deseado a elección del usuario.

La ejecución de la aplicación es automática, siempre que la extensión de ficheros JAR esté asociada a la máquina virtual de *Java* en la máquina destino.

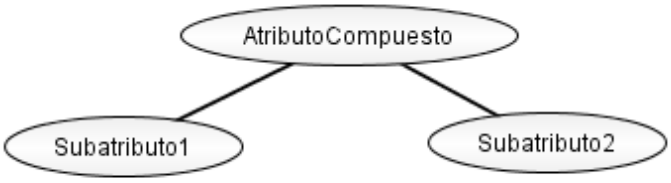
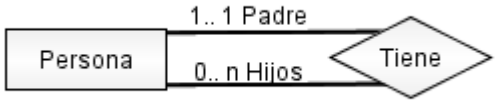
De no ser así probablemente se abra con una aplicación de tratamiento de ficheros comprimidos, pues es lo que es además de ser un fichero ejecutable. Habrá de asociarse la extensión JAR entonces a la aplicación JRE Java Runtime y se procederá a la ejecución de la misma realizando doble *click* sobre el fichero presentado.

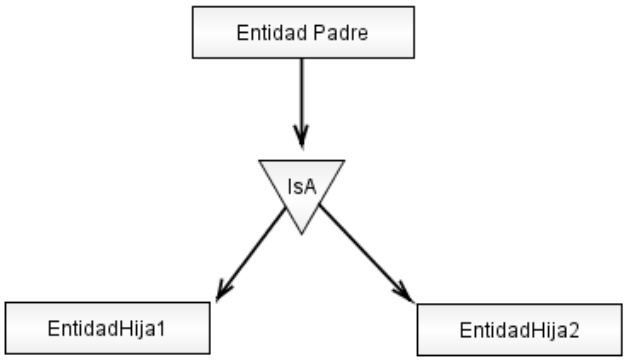
7.4. Glosario

A continuación se presenta la representación gráfica de los distintos nodos posibles en el interfaz de la aplicación:

Entidad fuerte	

DBCASE

Entidad débil	
Atributo simple	
Atributo clave primaria	
Atributo compuesto	
Atributo multivalorado	
Relación fuerte	
Relación débil	
Roles de una entidad	

Relación IsA	

7.5. Empezando a usar Data Base CASE

El primer paso será familiarizarse con la interfaz gráfica de la aplicación. En ésta podemos encontrar:

- **Barra de menús:** Está formado por tres menús principales:
  - **Menú Fichero:** A través de él podemos gestionar nuestros proyectos, guardando, abriendo y cerrando los mismos.
  - **Menú Opciones:** En él se permite puede configurar el gestor de bases de datos, el lenguaje y la visualización del panel de sucesos.
  - **Menú Ayuda:** Muestra información sobre la herramienta.
- **Pestaña Diagrama Entidad Relación:** Esta pestaña contiene los paneles a través de los cuales se diseña el diagrama Entidad-Relación.
- **Pestaña Generación de Código:** En ella encontramos los botones para validar el diagrama generado en la pestaña anterior, generar su modelo relacional y su código SQL. Además podemos guardar y ejecutar en un DBMS los scripts generados.

7.5.1. Añadir entidades, atributos y relaciones a un proyecto

El proceso de aprendizaje constará de la generación de una entidad, que posee un atributo, la generación de otra entidad y la relación existente entre ambas.

La inserción de entidades, atributos, relaciones y dominios, se realiza a través de la interfaz principal a través del botón secundario del ratón.

Añadir una entidad

Se ha de pulsar, dentro del panel de diseño, con el botón secundario sobre el lugar donde se desea añadir la nueva entidad.

Se expandirá el siguiente cuadro de selección.

DBCASE

Figura 7. 1

Se selecciona la opción de *Insertar* una nueva entidad y aparecerá la siguiente ventana:

Figura 7. 2

En la misma se completa el nombre de la entidad dentro del campo de texto, y se selecciona el *checkbox* si se desea que dicha entidad sea de carácter débil o no. En este ejemplo no lo seleccionaremos.

Cabe destacar la posibilidad de manejar todas las ventanas de la aplicación con el teclado, gracias a las teclas Escape, Enter y Tabulador. También se pueden pulsar los botones con las teclas correspondientes a la primera letra de cada uno de ellos, “I” y “C” en el ejemplo.

Tras presionar en Insertar se obtendrá un esquema similar al siguiente.

Figura 7. 3

Vemos la correcta inserción de la nueva Entidad en el Panel de Diseño.

Añadir un atributo

Para añadir un atributo ha de tenerse, previamente, una entidad o una relación sobre la que relacionar dicho atributo.

Para añadir un atributo basta con pulsar con el botón derecho del ratón dentro de una entidad, por ejemplo, y aparecerá el siguiente cuadro:

Figura 7. 4

Se selecciona, dentro del mismo, Añadir un nuevo atributo y aparecerá la interfaz de la siguiente página.

En ella se completa, en este orden, el nombre del atributo a generar, se marcan los *checkboxes* si el atributo es clave primaria, compuesto, no nulo, único o multivalorado y se especifica el tipo del atributo generado. Si el tipo requiere la especificación de un tamaño se activará el campo de texto, para rellenar el tamaño, en número, deseado para tal atributo.

Figura 7. 5

Si seleccionaremos el *ckeckbox* de clave primaria y se pulsa Insertar y se obtiene el siguiente esquema.

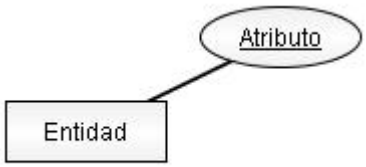


Figura 7. 6

Podemos mover ambos nodos a la posición del esquema que deseemos sin más que pulsar en un nodo y arrastrarlo al lugar deseado.

Añadir una relación

Ahora añadiremos una relación, que posteriormente relacionaremos con diversas entidades.

Para añadir una relación al sistema se pulsa con el botón derecho en un espacio libre del esquema y aparecerá el siguiente cuadro de selección.

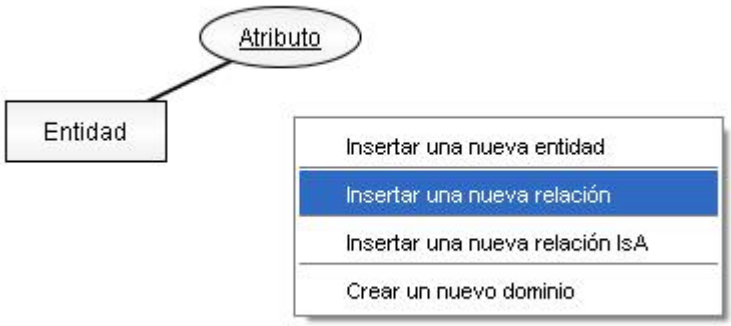


Figura 7. 7

Seleccionamos Insertar una nueva relación y aparecerá la siguiente interfaz:

Figura 7. 8

En ella hay que completar el nombre de la relación a generar y marcar el *checkbox* en caso de que se desee agregar una relación débil.

Por último, obtenemos el diseño final pulsando el botón Insertar.

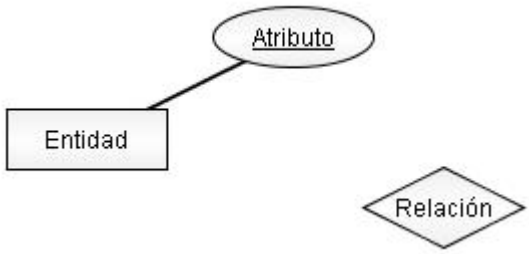


Figura 7. 9

7.5.2. Crear dominios y editar entidades, atributos y relaciones

A continuación se van a realizar modificaciones en el esquema actual. No se realizarán todas las posibles, pero sí las más relevantes, aunque todas compartan el mismo esquema de realización.

Primeramente, siguiendo los pasos anteriormente descritos, generar una nueva entidad denominada Vehículo con un nuevo atributo denominado Tipo, este atributo no será clave primaria. El esquema final quedaría como sigue.

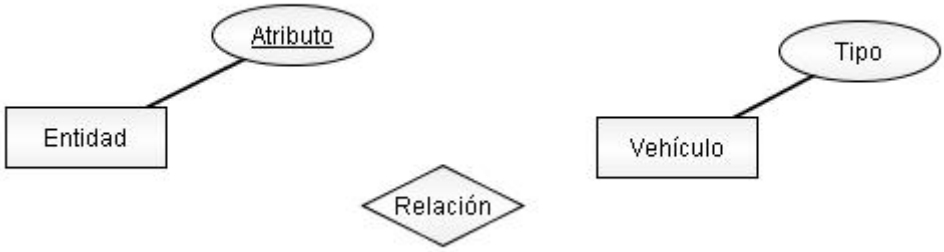


Figura 7. 10

A continuación se creará un dominio enumerado para el nuevo atributo.

Si se pulsa con el botón secundario sobre el panel principal y selecciona Crear nuevo Dominio. Dentro del interfaz seleccionar el nombre para el dominio, en el ejemplo Tipo de Vehículo, su tipo básico, VARCHAR y la lista de valores separados por comas y entre comillas simples.

Figura 7. 11

Queda ahora reflejado en el Panel de Dominios el nuevo enumerado, a través de este panel se podrá modificar o eliminar un dominio creado.

Figura 7. 12

A continuación se modificará el dominio del atributo Tipo de la entidad Vehículo.

Se pulsa con el botón secundario sobre el atributo que se quiera modificar y se elige Editar el dominio. Entre los dominios seleccionables se encontrará el creado en el paso anterior.

Figura 7. 13

Pulsando en Entidad y seleccionando Renombrar entidad, se modifica el nombre del nodo.

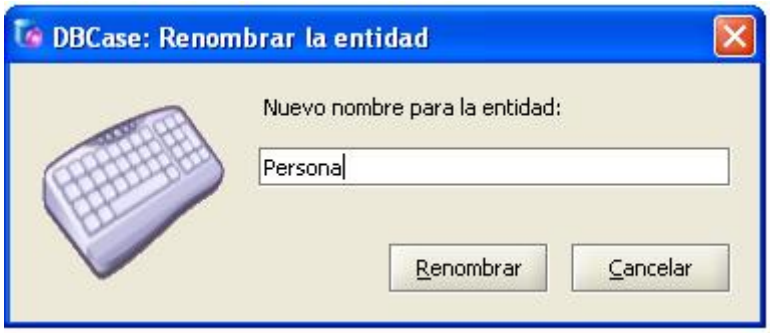


Figura 7. 14

De igual manera se renombra el nodo Atributo. Y se obtiene el siguiente esquema:

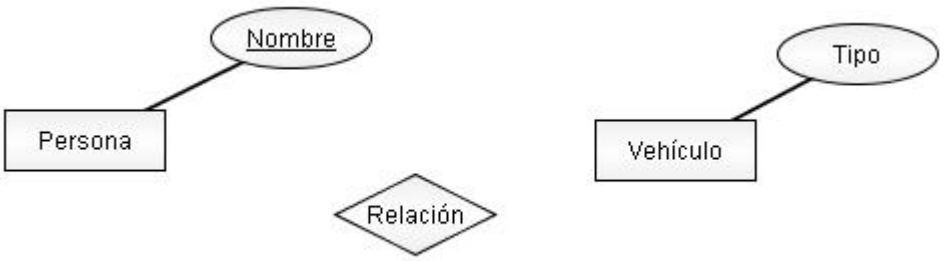


Figura 7. 15

El proceso finaliza con la edición de la relación, que relacionará ambas entidades. Se explicará el proceso para una entidad pues es idéntico en el segundo caso.

Pulsar con el botón derecho sobre la relación y seleccionar, en el cuadro de opciones, Añadir una entidad.

Figura 7. 16

En la interfaz seleccionar dentro del objeto combo la entidad que se quiere relacionar y elegir la cardinalidad de dicha relación, también se puede configurar un Rol, aunque sólo es obligatorio en implicaciones múltiples de la misma entidad en una relación. Una vez terminado pulsar Insertar.

Figura 7. 17

Repetir el proceso con la entidad Vehículo. El esquema final es el siguiente:

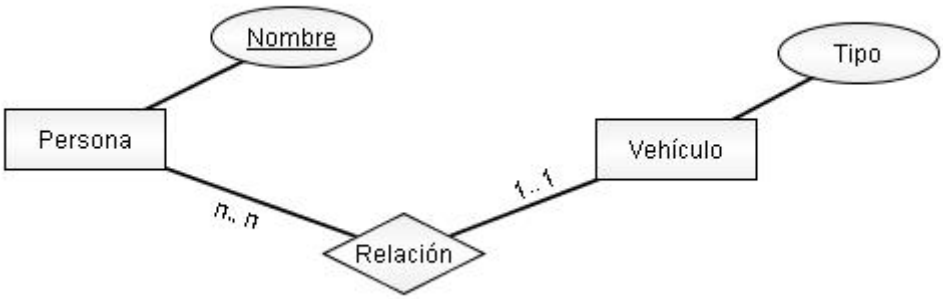


Figura 7. 18

7.5.3. Validar diseño del diagrama E/R

Para validar el diseño generado en la etapa previa hay que situar la ventana principal en su ficha Generación de código.

Ahí se ven seis botones situados en la franja izquierda de la ventana. Nombrados en orden descendente son los siguientes:

- Limpiar el área de texto de generación.
- Validación del diseño del esquema relacional.
- Generación del modelo relacional.
- Generación del script SQL en modo texto.
- Exportación del script SQL a fichero de texto.
- Ejecutar el script generado en un DBMS

En esquina superior derecha se puede seleccionar el gestor con el que se quiera trabajar.

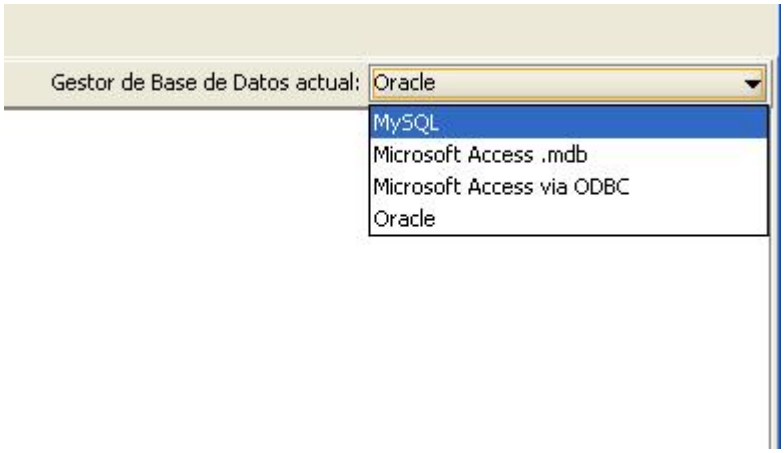


Figura 7. 19

Así pues, como se tiene un esquema ya diseñado, se procede pulsando el botón de Validación del diseño.

Obtenemos el siguiente resultado:

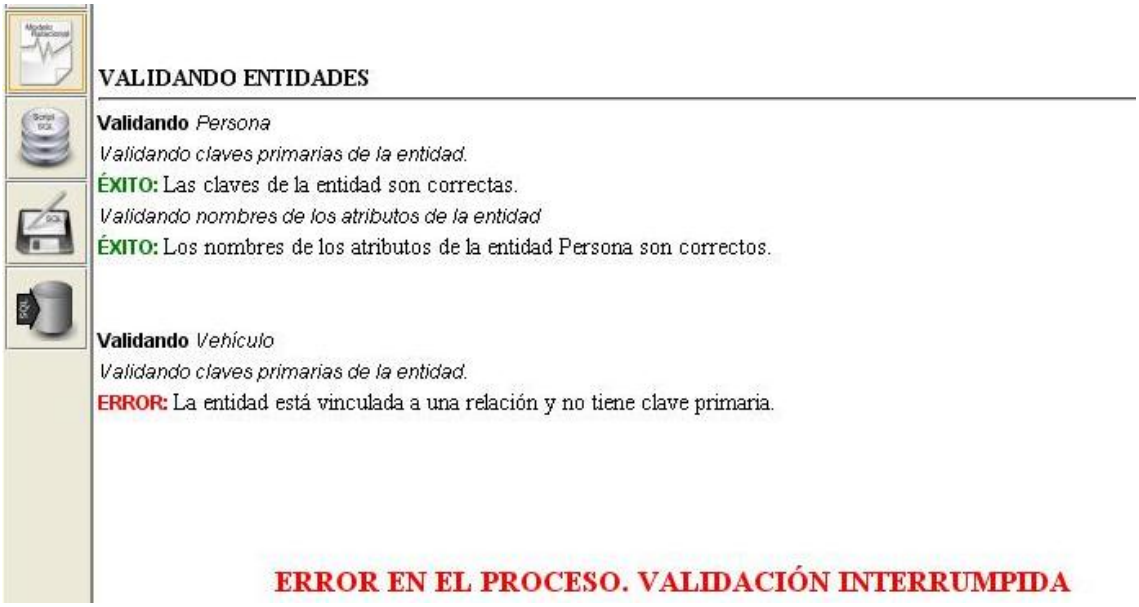


Figura 7. 20

Se nos informa de que el diseño no es correcto y se nos indica la razón.

Solucionamos el problema añadiendo un nuevo atributo Matricula. Podemos seleccionar el *checkbox* de clave primaria al crear el atributo, o si se nos olvida, podemos configurarlo pinchando con el botón derecho sobre el atributo.

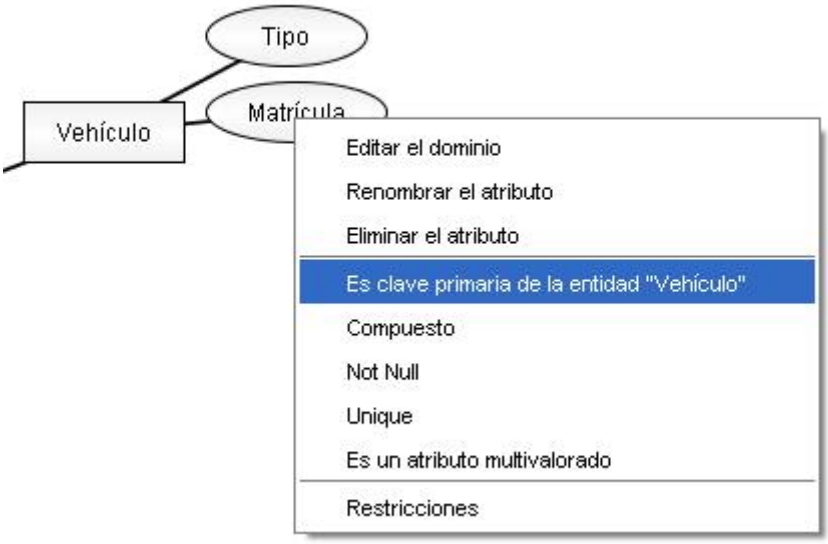


Figura 7. 21

Validamos de nuevo.

VALIDACIÓN REALIZADA CON ÉXITO

Figura 7. 22

Ahora ya si tenemos un diseño correcto.

7.5.4. Generación del Modelo Relacional.

La generación del modelo relacional se lanza con la pulsación sobre el tercer botón de la margen izquierda de la ventana. Si se desea, se puede pulsar de forma previa el primer botón para limpiar el área de texto de generación.

El resultado de la ejecución del modelo relacional es el siguiente.

MODELO RELACIONAL GENERADO :

Vehículo (Matrícula, Tipo)

Persona (Nombre)

Relación (Nombre, Matrícula)

Figura 7. 23

7.5.5. Generación del código SQL

La generación del script SQL se lanza con la pulsación sobre el cuarto botón de la margen izquierda de la ventana. Como en el caso anterior, si se desea se puede limpiar el contenido del área de texto de generación para una más fácil comprensión y legibilidad.

El código SQL generado para el esquema anterior y habiendo seleccionado como gestor MySQL es el siguiente:

CÓDIGO SQL DEL DISEÑO (MySQL)

SECCIÓN DE CREACIÓN DE TABLAS

```
DROP TABLE IF EXISTS Relación;
CREATE TABLE Relación(Nombre VARCHAR(10), Matrícula VARCHAR(8));

DROP TABLE IF EXISTS Vehículo;
CREATE TABLE Vehículo(Tipo VARCHAR(11), Matrícula VARCHAR(8));

DROP TABLE IF EXISTS Persona;
CREATE TABLE Persona(Nombre VARCHAR(10));
```

SECCIÓN DE CREACIÓN DE TIPOS ENUERADOS

```
DROP TABLE IF EXISTS Tipo de Vehículo;
CREATE TABLE Tipo de Vehículo(value_list VARCHAR(11)) ENGINE = InnoDB;
ALTER TABLE Tipo de Vehículo ADD PRIMARY KEY (value_list);
INSERT INTO Tipo de Vehículo VALUES ("Coche");
INSERT INTO Tipo de Vehículo VALUES ("Moto");
INSERT INTO Tipo de Vehículo VALUES ("Camión");
INSERT INTO Tipo de Vehículo VALUES ("Bicicleta");
```

SECCIÓN DE ESTABLECIMIENTO DE CLAVES

```
ALTER TABLE Vehículo ADD PRIMARY KEY (Matrícula);
ALTER TABLE Vehículo ADD FOREIGN KEY (Tipo) REFERENCES Tipo_de_Vehículo(value_list);
ALTER TABLE Persona ADD PRIMARY KEY (Nombre);
ALTER TABLE Relación ADD PRIMARY KEY (Nombre);
ALTER TABLE Relación ADD FOREIGN KEY (Nombre) REFERENCES Persona(Nombre);
ALTER TABLE Relación ADD FOREIGN KEY (Matrícula) REFERENCES Vehículo(Matrícula);
ALTER TABLE Relación ADD UNIQUE KEY (Matrícula);
```

Figura 7. 24

Pulsando el botón de exportación a fichero de texto plano una vez generado el código SQL, se solicitará la ubicación y el nombre del fichero en el que se guardará el script generado.

7.5.6. Ejecutar el script

Si es pulsado el botón de ejecutar el script generado tras haber generado el código aparece la siguiente ventana:

DBCASE

Figura 7. 25

En esta ventana se debe primero crear la conexión pulsando en Nueva, tras lo que aparecerá el siguiente formulario:

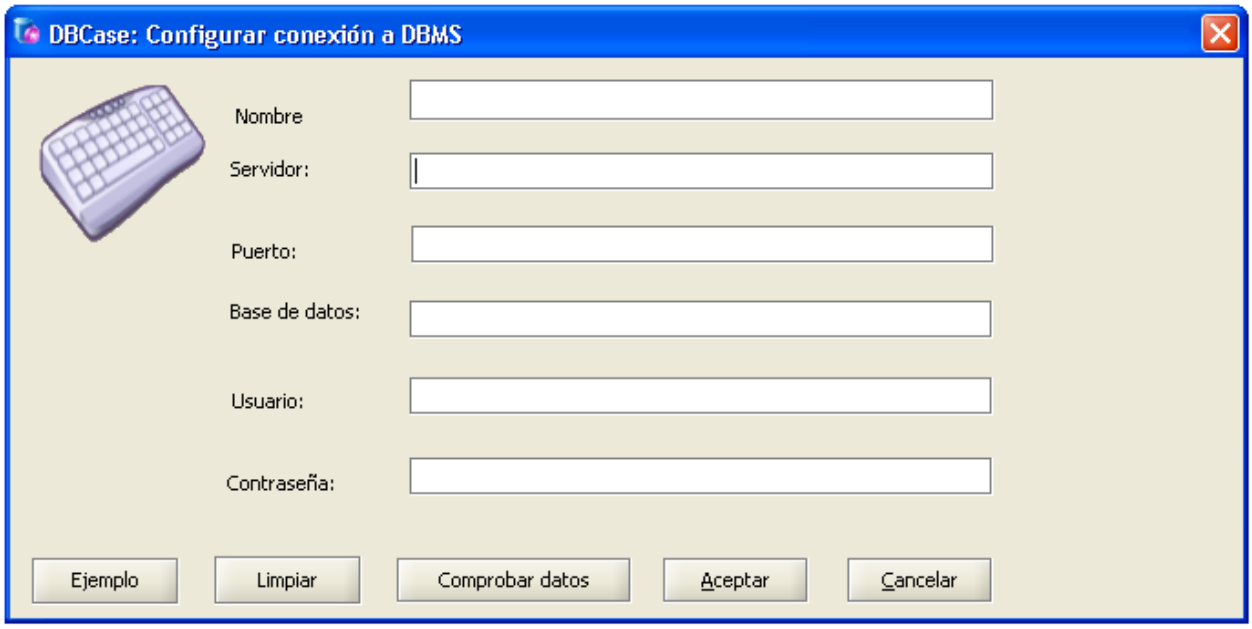


Figura 7.26

Donde deberá rellenar los campos teniendo en cuenta la siguiente información:

- Nombre: identificador que se le asignará a esta conexión.
- Servidor: al que desee conectarse.
- Puerto: si se deja en blanco tomará el puerto por defecto.
- Base de datos: nombre de la BBDD en la que se va a trabajar.
- Usuario: debe tener permisos para crear, acceder, modificar y borrar la base de datos.
- Contraseña: asociada al usuario.

Dependiendo del gestor elegido habrá algunos valores que no necesitarán ser completados.

Un ejemplo de como poder rellenarlos utilizando como gestor MySQL sería:

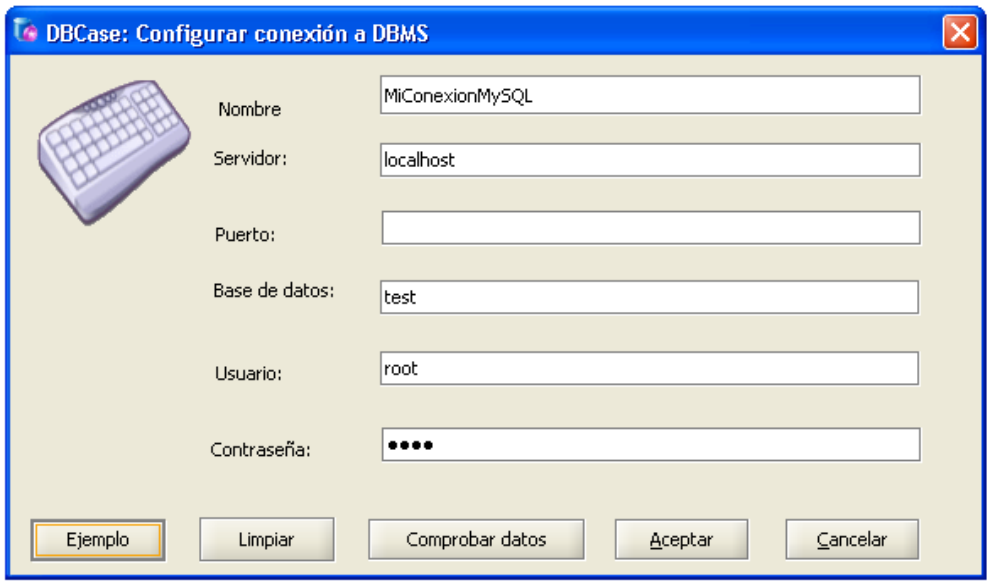


Figura 7.27

Se pulsará Aceptar; volviendo a la ventana anterior.

Se seleccionará la conexión y se pulsará Conectar

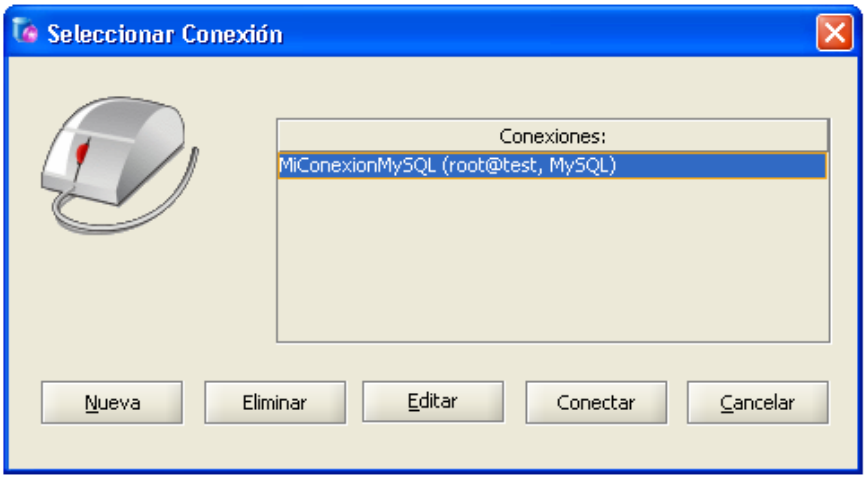


Figura 7.28

Si hay algún error aparecerá una ventana informando del motivo por el cual no ha podido realizarse la conexión, en caso de éxito se visualizará:

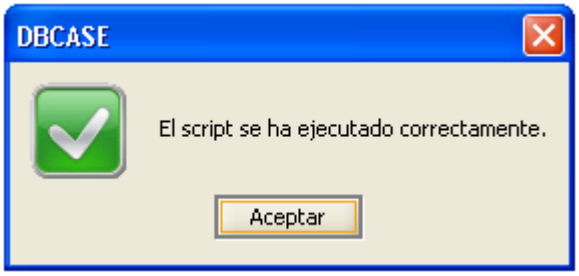


Figura 7.29

8. Conclusiones y trabajo futuro.

Con este proyecto se ha intentado, en la medida de lo posible corregir, ampliar y mejorar la funcionalidad de la aplicación DBDT de la cual partía nuestro trabajo, con el fin de obtener una herramienta globalizada que simplificase el diseño de las bases de datos a nivel conceptual, lógico y físico; que interactuara con tres de los principales gestores: MySQL, Access y Oracle.

Las primeras dificultades encontradas en el proceso de desarrollo fueron comprender el funcionamiento de DBDT, su estructura, cómo y dónde se llevaban a cabo cada una de las acciones, ya que la cantidad de código que teníamos para reutilizar era considerable.

Posteriormente también hemos tenido que solventar ciertos problemas con las bibliotecas usadas que no estaban bien documentadas y hacían compleja la tarea de utilizarlas.

A lo largo de la realización del proyecto han ido surgiendo ideas para ampliar y mejorar las utilidades de esta herramienta, que quedan pendientes para una siguiente versión de DBCASE, que podrían dividirse en dos líneas de trabajo:

Interfaz gráfica:

- Reorganización automática del grafo: recolocar los objetos gráficos de forma óptima para que el número de cruces entre aristas sea mínimo, situando contiguos los elementos que estén relacionados entre sí.
- Cuadrícula de diseño: darle la opción al usuario de visualizar una rejilla que le facilite la tarea de alinear los componentes de su diagrama, trazar líneas de referencia tanto verticales como horizontales, así como un cuadro de herramientas que permita girar los elementos.
- Permitir usar el botón secundario del ratón sobre los arcos para poder editar las aridades y los roles.
- Añadir a la barra de menú iconos para permitir realizar las mismas acciones que al interactuar con el ratón sobre el panel de diseño, así como incluir atajos de teclado como pueden ser Ctrl+C, Ctrl+Y, Ctrl+Z, o poder desplazar elementos usando los cursores.
- Especificar el nivel de zoom aplicado a cualquiera de los paneles de diseño

Funcionalidad:

- Normalización/desnormalización: permitir al usuario chequear el grado de normalización que posee su diseño: 1FN, 2FN, 3FN, Forma Normal Boyce-Codd; dándole las pautas a seguir para obtener cualquiera de los anteriores esquemas.
- Permitir agregaciones en el modelo Entidad-Relación.
- Incluir la posibilidad de añadir restricciones de participación total como alternativa a las cardinalidades, usando la notación gráfica adecuada.
- Dar la opción de guardar e imprimir el registro de validación
- Traducir la aplicación a otros idiomas; en la actualidad funciona en inglés y español.
- Incluir conectores a otros gestores de bases de datos diferentes a los ya existentes: MySQL, Access.mdb, Access vía ODBC, Oracle.

A la hora de validar el diseño permitir seleccionar si se desea ver la comprobación de cada uno de los elementos del diagrama, o simplemente el resultado

9. Bibliografía.

***Fundamentos de Bases de Datos*.**
Silberschaz, Koth y Sudarshan.
4ª Edición, Editorial McGrawHill.
Madrid 2002

***J2EE Design Patterns*.**
Jonathan Kaplan.
1ª Edición. Editorial O'Reilly & Associates

Fundamentos de Sistemas de Bases de Datos
Ramirez Elmasri y Shamkant B.Navathe.
Pearson Educación S.A.
Madrid 2007

Apuntes de Bases de Datos y Sistemas de la Información
Fernando Sáenz Pérez.

Apuntes de Bases de Datos y Sistemas de la Información
Luis Garmendia Salvador.

Apuntes de Bases de Datos y Sistemas de la Información
Héctor Gómez Gauchía.

API de Java
http://java.sun.com/javase/6/docs/api/

Documentación de MySQL (sintaxis y acceso)
http://dev.mysql.com/doc/

Documentación de Oracle (sintaxis y acceso)
http://www.oracle.com/technology/documentation/index.html

Sybase - Sybooks Online – Access Rules (Documentación de Microsoft Access)
http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sag1/html/sag1/sag1787.htm

Documentación y ejemplos de la biblioteca gráfica JUNG
http://jung.sourceforge.net/

Estándar IEEE Std. 830-1998 para la realización de la SRS.
http://standards.ieee.org/reading/ieee/std_public/description/se/830-1998_desc.html

10. Palabras clave.

Listado de palabras clave usadas en Data Base CASE:

- Bases de Datos Relacionales.
- Diagrama Entidad Relación.
- Validación.
- Modelo Relacional.
- MySQL.
- Microsoft Access.
- Oracle.
- Globalización.
- Persistencia.
- XML.

Autorizaciones de uso.

Se autoriza a la Universidad Complutense difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Firma de los autores:

Rodrigo Denís Cepeda Mateos
Cristina Marco De Francisco
Tello Serrano Gordillo