Departamento de Informática

PROYECTO FIN DE CARRERA

Simulador de Redes Dinámicas almacenadas en Bases de Datos

Autor: Katherin Salazar Sepúlveda

Tutor: Jessica Rivero Espinosa

Leganés, Octubre de 2011

Título: Simulador en Redes Dinámicas almacenadas en Bases de Datos.
Autor: Salazar Sepulveda, Katherin.
Director: Espinosa Rivero, Jessica.

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

En primer lugar quiero agradecer a mi tutora, Jessica Rivero, por ayudarme en cada paso que di para la realización de este proyecto, por su sabiduría y su forma de ser tan paciente y alegre, que me ayudaron en todo momento a sentirme cómoda y a disfrutar mientras realizaba la "práctica" más importante de mi carrera.

A mis padres, Luis Carlos y Adriana, a quien va dedicado este proyecto, sin su esfuerzo y empeño por sacarnos adelante a mí y a mis hermanas yo no estaría aquí, a ellos, que día a día me inculcaron lo importante que era estudiar, ser humilde y buena persona. Espero que estén orgullosos de mí.

A mis hermanas, para que nunca se rindan en este largo camino llamado "estudiar", un camino lleno de esfuerzo y nuevas experiencias, que resultan muy gratificantes cuando alcanzas cada meta que te propones.

Y como no, a mi novio, amigo, compañero, Samuel, sin su apoyo incondicional, sus charlas constructivas hacia mí, pero sobre todo, al amor que me da, no me sentiría tan feliz en este momento.

También quería agradecer a mis compañeros de universidad que me han acompañado en este largo camino, de risas, estrés, preocupaciones, satisfacciones cuando conseguíamos terminar alguna práctica o aprobábamos un examen… nunca los olvidaré chicos. Y por último, a un amigo, profesor, Jose, que en la mayoría de veces cuando no encontraba una salida, su sabiduría me resultó de gran ayuda.

Gracias.

Índice General

Índice de figuras

Capítulo 1. Introducción y Objetivos

1.1 Introducción

Hoy en día en el mundo hay presentes millones de redes que representan distintos tipos de información, entendiendo por redes, un grafo en el que se interconectan distintos componentes o nodos mediante enlaces o conexiones.

Como ejemplos de redes tenemos redes informáticas en las que se pueden representar las conexiones entre un conjunto de ordenadores; una red de metro, que representa las conexiones entre las distintas paradas de metro; la representación con una red de las conexiones que tiene una empresa con cada lugar del planeta; o la información de las redes sociales que representan contactos a nivel mundial. Además, muchas de las redes que existen en la actualidad están en continuo cambio. Por ejemplo, en las redes de carreteras, una de las redes más frecuentemente empleadas por todo el mundo, puede haber un accidente que haga inadecuado el uso de un enlace, o tramos nuevos auxiliares debido a rehabilitaciones o mantenimientos de tramos de autopistas. El visualizar ese cambio continuo podría resultar muy interesante, ya que, podría servir para saber, en una red de carreteras, que enlaces están colapsados para escoger el mejor camino para ir de un lado a otro.

Por otro lado, debido a que es necesario tener un soporte de almacenamiento capaz de tratar con la gran cantidad de información que contienen este tipo de aplicaciones, los *SGBD* (*los sistemas gestores de bases de datos*) han experimentado una evolución que ha llevado a que hoy en día se pueda trabajar con datos espaciales presentes en los sistemas de información espacial que trabajan, por ejemplo, con aplicaciones para *GPS*, *Google Maps*, u *Oracle Maps* donde la representación geográfica es su principal objetivo.

Uniendo ambos campos, en este proyecto se tiene como objetivo el crear una aplicación para visualizar el continuo cambio que ejerce una red dinámica y visualizar su localización en un espacio geográfico empleando un SGBD para almacenar la red, sus componentes y como se conectan entre ellos, así como toda la información espacial asociada, para no limitar el tamaño de la misma. Dicha aplicación contendrá una interfaz gráfica con un panel de dibujo y en él se dibujará una red y los continuos cambios producidos en ella para poder tener una fácil visualización de toda esta información. Para ello es necesario investigar lenguajes de programación que diseñen interfaces gráficas y que además soporten la conexión con bases de datos espaciales.

Una vez expuestos los motivos que ha llevado al desarrollo del presente se proyecto se presentan a continuación en el siguiente apartado los principales objetivos.

1.2 Objetivos

El principal objetivo del desarrollo de este proyecto es visualizar y simular el comportamiento de una red dinámica para que pueda servir en un futuro para el diseño de nuevos algoritmos que calculen los caminos más eficientes para ir de un nodo a otro teniendo en cuenta el continuo cambio de la red.

En cuanto al diseño visual se tiene como objetivos:

- Diseñar una interfaz gráfica que sea intuitiva para el correcto uso de la misma.
- Que la interfaz gráfica contenga un panel de dibujo donde se pinte la red que se pretende representar.
- Una panel con una serie de botones que ejecuten las distintas funcionalidades de pintado o borrado de componentes de la red.
- Un panel con cuadros de texto que permitan que el usuario pueda introducir la información deseada correspondiente a los componentes de la red para poder introducir cambios en ellos.

En lo referente al desarrollo software, se pretende:

- Implementar una interfaz gráfica que permita la representación de una red y sus continuos cambios mediante un mecanismo manual o mediante un mecanismo automático.
- Diseñar una base de datos que almacene todos los datos referidos a los componentes de la red (espaciales o no).
- Conectar la interfaz gráfica con la base de datos para que lo que se dibuje en la interfaz gráfica se almacene en la base de datos y viceversa (que la información que esté almacenada en la base de datos se represente en la interfaz gráfica) de forma automática.

Por último, se persigue cubrir la siguiente funcionalidad:

- Pintar una red entera.
- Borrar la red por completo.
- Pintar un nodo en unas coordenadas específicas.
- Borrar un nodo existente.
- Pintar un enlace que conecte dos nodos.
- Borrar un enlace específico.

- Poder cambiar al coste a un determinado enlace.

1.3 Plan de trabajo

Para la elaboración del proyecto, éste se ha dividido en etapas para facilitar el desarrollo del mismo. A continuación se especifica cada etapa elemental del desarrollo de este proyecto:

- **Investigación sobre lenguajes de programación que permiten la conexión con bases de datos e implementen interfaces graficas.** Con el fin de tener una visión global de los distintos tipos de lenguajes de programación que facilitan la conexión con una base de datos y la implementación de interfaces gráficas en la actualidad, se decide investigar sobre ellos para obtener una comparativa de cual sería una buena opción como lenguaje de programación para el desarrollo de este proyecto.

- **Estudio del lenguaje de programación *Java*.** Puesto que este fue el lenguaje de programación elegido, y ante el desconocimiento total de él y del desarrollo de interfaces gráficas, se decidió estudiar los conceptos básicos de la programación orientada a objetos y el diseño de interfaces gráficas con *Java*, debido a que este lenguaje tiene una gran biblioteca de clases implementadas para ello. Por otro lado, se estudió también *JDBC* por ser elemental para la conexión de una aplicación *Java* con una base de datos.

- **Investigación sobre las bases de datos, los sistemas de información geográfica, así, como de las distintas bases de datos espaciales existentes en el mercado.** Con el fin de tener una visión global del significado y representación que tienen en la actualidad los sistemas de información espacial y de los distintos tipos de bases de datos espaciales existentes, se decidió investigar sobre ellos obteniendo además una comparativa de cuál sería la mejor base de datos espacial para el desarrollo de este proyecto.

- **Estudio de la base de datos espacial de *Oracle, Oracle Spatial* (por ser la base de datos elegida en el estudio anterior).** Estudio de los conceptos básicos de esta base de datos, así como las ventajas y funcionalidades que tiene *Oracle Spatial* para manipular datos espaciales.

- **Diseño y análisis de la aplicación.** Se procede al diseño de la base de datos así como de los requisitos y funcionalidades de la interfaz gráfica para satisfacer las necesidades del sistema.

- **Implementación de la interfaz gráfica.** Se procede a la implementación de la interfaz gráfica con el lenguaje *Java* y el entorno de desarrollo integrado *NetBeans*, creando las clases y sus procedimientos necesarios para la ejecución de las funcionalidades.

- **Implementación de la base de datos espacial.** Se crean los Scripts de la base de datos, como los de la creación de tablas, procedimiento, disparadores etc.

- **Implementación de la conexión entre la interfaz gráfica y la base de datos.** Se establece la conexión entre ambas y se implementan los procedimientos para acceder a la base de datos desde la interfaz gráfica y viceversa.

- **Elaboración de la memoria.** al final del desarrollo de la aplicación se procede a la realización de la memoria para presentar la descripción del proyecto, dividiéndola en distintos apartados, cada uno con la información específica de cada etapa del proyecto.

1.4 Estructura del documento

El contenido del documento se divide en capítulos, donde cada uno tiene una serie de apartados para detallar más el contenido de la información. A continuación se detalla la estructura del documento:

- **Capítulo1. Introducción.** Descripción introductoria de los motivos que llevaron al desarrollo de este proyecto, así como los objetivos impuestos, etapas de desarrollo del proyecto y resumen de la estructura del documento.

- **Capítulo 2. Estado del Arte.** Se analizan las tecnologías necesarias para el desarrollo del proyecto, dividido en dos apartados. Por un lado todo lo referente al desarrollo de interfaces gráficas, así como el lenguaje de programación utilizado, y por otro lado, se especifica todo lo referente a las bases de datos espaciales, así como la base de datos espacial elegida para el cumplimiento de los objetivos.

- **Capítulo 3. Diseño.** Se define la presentación del problema, los requisitos de la aplicación, y se realiza el diseño de la base de datos y el diseño visual de la aplicación.

- **Capítulo 4. Implementación.** Partiendo del diseño, se implementa la base de datos y se muestra el diagrama de clases de la aplicación para posteriormente explicar las funcionalidades de la misma implementadas.

- **Capítulo 5. Pruebas.** Se realizan una serie de pruebas para comprobar que se han cumplido los objetivos impuestos y que el funcionamiento de la aplicación es correcto.

- **Capítulo 6. Conclusiones y líneas futuras.** Se obtienen las conclusiones de la realización del proyecto. Además, se exponen las posibles mejoras que se pueden realizar sobre este proyecto.

- **Anexos.** Este apartado incluye una lista de los acrónimos utilizados en este documento, la bibliografía utilizada para el desarrollo del proyecto, así como el manual de usuario, la planificación del proyecto y el presupuesto del mismo.

Capítulo 2. Estado del Arte

2.1 Introducción

El estado del arte se divide en dos grandes apartados debido a que la implementación del sistema de este proyecto necesitará de un lenguaje de programación capaz de crear una interfaz gráfica, y de una base de datos espacial para la parte de almacenamiento de la información relativa a la red.

Por tanto en el primer apartado se explicará a grandes rasgos qué es un lenguaje de programación, pasando a enumerar algunos lenguajes de programación que permiten conectar con bases de datos y lenguajes de programación que permiten desarrollar interfaces gráficas para, posteriormente, hacer un análisis de todos estos tipos de lenguajes de programación y presentar las conclusiones que llevarán a elegir el lenguaje de programación más apropiado para desarrollar la interfaz gráfica.

Una vez explicados los puntos anteriores, es necesario describir el lenguaje de programación elegido, su historia, características más importantes, el desarrollo de interfaces gráficas y, dentro de este apartado, las características más destacadas para entender cómo se desarrolla y qué herramientas se utilizan para crear una interfaz gráfica.

El segundo apartado contempla información referida a las bases de datos, su historia y evolución, y, por tratarse de bases de datos espaciales, explicar los aspectos relevantes de los sistemas de información geográfica. Al igual que en el apartado anterior, en este caso también es importante enumerar tipos bases de datos espaciales para obtener una pequeña descripción de cada una de ellas, incluyendo una comparativa de las bases de datos enumeradas y la posterior explicación detallada de la que se ha considerado mejor en dicha comparación.

2.2 Lenguaje de programación

En base a [18] la programación es el proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático, el cual está definido bajo un lenguaje de programación.

Para entender que es un lenguaje de programación se define lo siguiente: es un lenguaje artificial que intenta estar relativamente próximo al lenguaje humano o natural, y que es diseñado para describir un conjunto de tareas que un ordenador debe ejecutar. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen la estructura de cada lenguaje y el significado de sus elementos y expresiones.

Además, según [19] los lenguajes de programación se pueden clasificar según el paradigma que utilizan, el cual determina la visión y métodos de un programador en la implementación de un programa. Los lenguajes de programación más relevantes según su paradigma se pueden clasificar en lenguajes de programación estructurados, por ejemplo *Pascal, C, ADA*, etc., donde la programación se divide en bloques (procedimientos y funciones) y la ejecución se controla con secuencia, selección e iteración. Por otro lado los lenguajes de programación orientados a objetos, por ejemplo *Java, PHP*, etc., donde la idea es encapsular el estado y operaciones en objetos y la programación se resuelve comunicando estos objetos a través de mensajes.

Es importante destacar que los lenguajes orientados a objetos tienen grandes ventajas como la reutilización y extensión del código, el permitir crear sistemas más complejos relacionados con el mundo real, intentando representar este mundo tan fielmente como sea posible, o la facilidad para la creación de programas con interfaces gráficas entre otras.

2.3 Los lenguajes de programación y las bases de datos

Para tener una visión global de muchos de los lenguajes de programación que permiten conectar con una base de datos, se enumeran algunos lenguajes de programación con esta característica y la manera en que permiten ejecutar la conexión con una base de datos con el fin de compararlos.

2.3.1 Java

Como se indica en [1] para conectar bases de datos Relacionales con programas *Java* es necesario utilizar un *API (Application Programming Interface)* de *Java* llamada *JDBC (Java Database Connectivity),* esta es proporcionada por un conjunto de clases, que permite ejecutar instrucciones *SQL* para trabajar y gestionar la información almacenada en la base de datos.

Este conjunto de instrucciones *SQL* están definidas como su propio nombre indica en lenguaje *SQL,* el cual es un lenguaje estándar que nació de la necesidad de simplificar el uso de programas de bases de datos Relacionales, pues suplanta la necesidad de aprender diferentes lenguajes de bases de datos para cada formato de base de datos como *Java* DB o la base de datos incluida en *Java* 6, que tiene soporte para *SQL*.

Antes de especificar los puntos más relevantes para trabajar con una base de datos desde *Java* es importante tener claro que la *API JDBC* se proporciona en dos paquetes, el paquete *Java.sql* y *Javax.sql* donde el primero contiene las interfaces y las clases fundamentales de *JDBC*. Cabe destacar que cualquier aplicación *Java* utilizará casi siempre las siguientes clases:

- *DriverManager*: permite gestionar todos los controladores instalados en la máquina virtual, además es utilizada para crear el objeto *Connection.*
- *Connection:* representa una conexión con una base de datos, además es utilizado para crear un objeto *Statement.*
- *Statement:* permite ejecutar sentencias *SQL*, además es utilizado para crear el objeto *Resultset.*
- *ResultSet:* almacena el conjunto de resultados en filas al ejecutar cualquier sentencia *SELECT.*

Una vez que están claras las principales clases de *JDBC*, se especifican los pasos más relevantes para trabajar con una base de datos desde *Java:*

- **Conectar con la base de datos**. Para que una aplicación *Java* pueda hacer operaciones sobre una base de datos, previamente se debe establecer una conexión con ella mediante un controlador (*driver*) que permite la conexión directa con la base de datos, por lo que es más rápido, mediante un conjunto de información como son la *url*, el usuario y la contraseña asociados con la conexión que se quiere establecer.

- **Recuperar datos de la base de datos**. Para trabajar sobre la base de datos, se obtiene del objeto *Connection* un objeto *Statement* para ejecutar su método *executeQuery(sentencia SQL)* al cual se le pasa por parámetro la sentencia *SQL* encargada de realizar la consulta, para que devuelva los resultados en un objeto *ResultSet*.

  Además de realizar consultas, el objeto *Statement* cuenta con el método *executeUpdate(sentencia SQL)* para realizar actualizaciones, borrados o inserciones sobre los datos almacenados en la base de datos.

- **Ver las consultas resultantes.** El resultado de estas consultas se almacena como un conjunto de filas encapsuladas en un objeto de tipo *ResultSet.* En definitiva, este resultado es una tabla que utiliza un cursor para indicar la fila sobre la que se realizará una determinada operación. Este cursor puede ser movido gracias a distintos métodos como *next(), first()…*etc. Y gracias al conjunto de métodos como *getString() o getInt()*se pueden obtener los datos de cada fila.

  En el momento que no se utilice más la base de datos se debe cerrar la conexión con el método *close()* del objeto *Connection* y aunque esta acción implica cerrar también los objetos *ResultSet,* y *Statement,* es conveniente cerrar cada uno por separado en el momento que no se usen, aunque la conexión no haya finalizado.

- **Lanzar procedimientos de la Base de datos desde la aplicación *Java.*** Esto es posible gracias a un objeto de tipo *CallableStatement* que prepara la llamada al procedimiento.

2.3.2 Visual Basic.Net

Como se indica en [2], a la hora de trabajar con bases de datos desde Visual Studio se usa el proveedor *ADO.NET* para trabajar con Microsoft SQL Server. *ADO.NET* es una tecnología de acceso a datos, potente y fácil de utilizar con ventajas como:

- No depende de conexiones continuamente activas, es decir, las aplicaciones se conectan a las bases de datos sólo durante el tiempo necesario para extraer la información.
- Las interacciones con la base de datos se realizan mediante objetos que encapsulan las sentencias *SQL* de acceso a los datos.
- Los datos requeridos, normalmente se almacenan en memoria caché, lo que permite trabajar sin conexión sobre una copia temporal de los datos obtenidos.
- El formato de transferencia de datos es *XML* lo que permite enviar la información a través de cualquier protocolo como, por ejemplo, *HTTP*.

Para entender el funcionamiento de *ADO.NET* cabe destacar que es un conjunto de clases pertenecientes al espacio de nombres *System.Data* donde los componentes que proporciona están diseñados para separar el acceso a los datos de la manipulación de los mismos, por tanto los componentes principales son el *DataSet* que representa el conjunto de datos y *.Net Framework,* un proveedor de datos que sirve como puente entre la aplicación y la base de datos.

Los componentes principales de *.Net Framework* son:

- El objeto *Connection,* que establece la conexión al origen de datos especificado. La función de este objeto es presentar atributos y métodos para permitir establecer y modificar las propiedades de la conexión como el nombre de usuario y contraseña entre otras.

- El objeto *Command,* que ejecuta sentencias *SQL* y devuelve resultados de un origen de datos después de establecer la conexión.

- El objeto *DataReader,* que lee una secuencia de datos cuando estos sólo se necesitan para lectura, es decir no van a ser actualizados, por tanto este objeto obtiene los datos de la base de datos y los pasa directamente a la aplicación.

- El objeto *DataAdapter,* intercambia datos entre el origen y el conjunto de datos (*DataSet*) para realizar las acciones necesarias de modificación en el origen de datos.

  Para llevar a cabo estas acciones, el objeto *DataAdapter* cuenta con las propiedades *SelectCommand, InsertCommand, DeleteCommand, UpdateCommand,* que como sus nombres indican *SelectCommand* hace referencia a una orden que recupera filas de la base de datos, es decir, realiza consultas, *InsertCommand* hace referencia a una orden para insertar filas, *DeleteCommand* hace referencia a una orden para borrar filas y

UpdateCommand hace referencia a la orden de modificar filas del origen de datos.

2.3.3 PHP

PHP es un lenguaje de programación orientado al desarrollo de páginas web que como se indica en [4] [3] también permite conectar con bases de datos usando el gestor *MySQL* o a través del estándar *ODBC (Open Data Base Connectivity).*

La forma de conectar con motores de bases de datos que trabajan bajo *ODBC*, en realidad, no es el modo de trabajo más eficiente posible, ya que al ser necesario interponer el *ODBC* entre el script y la base de datos se reduce el rendimiento del sistema. Además, las funciones de gestión de base de datos que aporta *ODBC* no son nativas de *PHP*, por lo que habrá más limitaciones a la hora de trabajar con la información almacenada en una base de datos. Por tanto, para ahorrarse el hecho de interponer *ODBC* entre *PHP* y el motor de la base de datos, es más conveniente hacer uso del gestor *MySQL*.

Existen numerosas funciones de *PHP* que funcionan específicamente con *MySQL*, por lo que hablar de *PHP* va ligado a hablar de *MySQL*. *E*ntre las funciones más destacadas se encuentran:

- *Mysql_connect*(“nombredelhost”,”usuario”,”contraseña”). Función utilizada para conectar con el servidor de *MySQL* por medio de las variables de conexión.

- *Mysql_query*(“consulta”). Función utilizada para enviar cualquier tipo de comando de consulta a la base de datos.

- *Mysql_fetch_rows*(“variable de resultados de la consulta”). Función que se utiliza para devolver una fila de resultados de una consulta a la base de datos.

- *Mysql_fetch_array*(“variable de resultados de la consulta”). Función que se utiliza para devolver varias filas de resultados de una consulta a la base de datos.

2.4 Lenguajes de programación y las interfaces gráficas

En este apartado se analizarán lenguajes de programación que implementan interfaces gráficas como el lenguaje de programación *Java* y el lenguaje de programación VisualBasic.Net.

El motivo de elección de estos dos lenguajes para hacer el análisis es debido a que estos dos lenguajes hoy en día son la mejor opción para los principales desarrolladores de software, ya que ambos, con el paso del tiempo, han alcanzado posiciones muy dominantes en el mercado. Si antes el lenguaje PHP era parte de la comparativa entre

lenguajes que conectan con bases de datos, en este apartado no será incluido debido a que es un lenguaje orientado al desarrollo de páginas Web.

A continuación se detalla de manera genérica las características que tiene cada lenguaje para implementar interfaces gráficas.

2.4.1 Java

Para implementar una interfaz gráfica con este lenguaje se han de tener en cuenta las siguientes características [8]:

- **Entorno de desarrollo integrado (*IDE*).** Las operaciones de diseño de interfaces gráficas como la edición, compilación, ejecución y depuración son más fáciles cuando se trabaja con un entorno de desarrollo integrado, como es el caso del *IDE NetBeans* de *Sun Microsystems* que permite el diseño de interfaces, escribiendo poco código fuente: basta con seleccionar uno a uno los componentes de una paleta de componentes y añadirlos a la ventana, además, facilita la conexión entre ellos.

- **Bibliotecas de clases.** Para diseñar interfaces gráficas con componentes como ventanas, cajas de texto, botones, etc. *Java* proporciona una biblioteca de clases *JFC* (*Java Foundation Classes*) agrupadas en las siguientes *APIs* (interfaces para programación de aplicaciones):

  - *AWT* (kit de herramientas de ventanas abstractas). Grupo de componentes para diseñar Interfaces gráficas común a todas las plataformas, pero escritos para cada plataforma en código nativo, es decir no están escritos en *Java*.

  - *SWING*. Conjunto de componentes escritos en *Java* que se ejecutan en cualquier plataforma nativa que soporta la máquina virtual de *Java*. Los componentes *SWING* han sustituido en gran medida a los componentes *AWT*, por lo que muchos de los componentes *SWING* heredan de sus correspondientes componentes *AWT*.

- **Conceptos básicos.** En una aplicación que contenga una interfaz gráfica su principal componente es un formulario, también llamado ventana, sobre la que se dibujan otros componentes, también llamados controles, como etiquetas, cajas de texto, barras de desplazamiento, menús, etc.
  Pueden existir distintos tipos de formulario dependiendo de la funcionalidad que se le atribuya, pueden visualizar información textual o una imagen.

  Una vez finalizado el diseño visual es necesario agregarle funcionalidad a cada componente mediante la edición de código fuente asociado a cada uno, ya que cada componente debe estar ligado a un trozo de código fuente que determine su funcionalidad.

Ese componente estará esperando a que sea activado mediante un evento producido por una acción sobre él, es decir, cuando ocurre un evento, el sistema enviará un mensaje al componente que deba responder a ese evento, y la respuesta a ese evento consistirá en la ejecución del método asociado a ese componente.

2.4.2 Visual Basic .Net

Para implementar una interfaz gráfica con este lenguaje se ha de tener en cuenta las siguientes características [2]:

- **Entorno de desarrollo integrado (*IDE*).** Los entornos de desarrollo integrado siempre facilitan la construcción de aplicaciones, en este caso, con el lenguaje *Visual Basic .Net,* el *IDE Visual Basic 2005 Express* permite diseñar la interfaz gráfica de una manera visual, donde se utiliza el ratón para arrastrar y colocar los objetos prefabricados en el lugar deseado dentro del formulario, para luego darle una funcionalidad asociada.

- **Bibliotecas de clases.** Las bibliotecas de clases contienen todos los componentes y funciones necesarias para la implementación de una interfaz gráfica. En *Visual Basic .Net* están organizadas en espacios de nombres llamados *System, System.Windows.Form,* y *System.Drawing.*

- **Conceptos básicos.** Una aplicación para *Windows* diseñada con *Visual Basic .Net* para interaccionar con el usuario está formada por dos tipos de objetos, que serán fundamentalmente ventanas, también llamadas formularios, y controles, también llamados botones, cuadros de texto, listas, etc.

  El procedimiento de implementación es muy sencillo: primero con estos objetos se diseña el aspecto visual de la interfaz, se crea una ventana y sobre esa ventana se añaden los controles. Además, para que estos objetos produzcan una funcionalidad, es necesario añadir código fuente relacionado con la función que tiene cada objeto. Así, en el momento que el usuario interaccione con alguno de esos objetos, se producirá un evento que hará que se ejecute el código fuente escrito para la funcionalidad de ese objeto. Se dice entonces que esta programación es conducida por eventos y orientada a objetos.

  Por tanto la característica conducida por eventos, define que cuando una acción sobre un objeto produce un evento, entendiendo por evento, un mensaje que envía un objeto a otro objeto, se espera que suceda algo. Ese algo es el código fuente que hay que programar.

2.5 Conclusión

Este apartado describe una pequeña conclusión de la comparativa que se ha obtenido de los lenguajes de programación *Java, Visual Basic .NET, PHP* y su forma de conectar con una base de datos. Por otro lado, también se ha tenido en cuenta la forma de los lenguajes de programación *Java* y *Visual Basic .NET* y su relación con aplicaciones que contienen interfaces gráficas.

Referido a los lenguajes de programación y su forma de conectar con bases de datos se concluye que la manera de conectar con una base de datos de los lenguajes es muy similar, cada uno posee su propia biblioteca de clases con las funciones necesarias para establecer una conexión con una base de datos y trabajar con ella, y aunque se diferencian en el nombre de las funciones, todas vienen a ejercer la misma función. Sin embargo, las bibliotecas de clases están más desarrolladas en *Visual Basic .Net* y *Java* y por tanto tienen la ventaja de conectar con más sistemas gestores de bases de datos. Además, las funciones de conexión y trabajo con una base de datos de las bibliotecas de de estos lenguajes, están más desarrolladas en comparación con las bibliotecas de clases de *PHP* que son relativamente más nuevas.

Referido a los lenguajes que desarrollan aplicaciones con interfaces gráficas se concluye que al comparar el modo de procedimiento de *Java* y *Visual Basic .Net* para implementar interfaces gráficas es el mismo, cada uno posee su propia biblioteca de clases que contiene las funciones y componentes necesarios para diseñar una interfaz gráfica, y prácticamente todos los componentes son los mismo como cajas de texto, botones, listas, formularios, etc. También cada uno se complementa con su propio *IDE* para que la implementación de interfaces gráficas sea muy sencilla, pero lo más importante es que los dos lenguajes tienen el mismo concepto básico a tener en cuenta a la hora de implementar una interfaz gráfica, y es que las interfaces gráficas que se implementen con estos dos lenguajes serán orientadas a objetos y conducidas por eventos.

Por tanto, la conexión y trabajo con una base de datos y la implementación de interfaces gráficas con los lenguajes de programación evaluados es muy similar, sin embargo hay que tener en cuenta para quien se va a desarrollar la aplicación. Para ello, es importante indicar, antes de proseguir con la comparativa, que *Visual Basic .Net* fue creado para ser usado en plataformas *Windows* y que *Java* es un lenguaje multiplataforma y está presente en *Windows, Linux, Mac OS*, y aplicaciones para móviles.

En conclusión antes de elegir un lenguaje de programación, tanto para implementar aplicaciones con interfaces gráficas como para establecer conexiones con bases de datos y trabajar con ellas, es importante definir en qué plataforma va a ser implantado, porque en caso que se desee implantar sólo en una plataforma Windows y exista la certeza que no se va a migrar a ninguna otra plataforma, la mejor elección es el lenguaje *Visual Basic .Net*, en cambio si la aplicación va a ser implantada en una determinada plataforma pero con posibilidades de migrar, por ejemplo de Windows a Linux o mover la aplicación de un sistema gestor de base de datos *Oracle* a *MySQL* la mejor opción sería *Java*.

2.6 Elección de *Java*

Una vez analizadas las distintas características de otros lenguajes, se decide analizar en más profundidad el lenguaje de programación *Java* en lo que a implementación de aplicaciones con interfaces gráficas se refiere.

Una de las razones de esta elección se refiere a que es un lenguaje bastante demandado a nivel laboral y muy útil a la hora de realizar aplicaciones, debido en gran medida a la ayuda de *IDEs* como *NetBeans* que facilita mucho el desarrollo de interfaces gráficas.

Otra razón importante es que *Java* es un lenguaje independiente de la plataforma, puede implantarse en cualquier plataforma como *Windows, Linux, Mac OS,* o teléfonos móviles. También es sencillo de aprender para programadores que hayan trabajado previamente con lenguajes orientados a objetos debido a su sintaxis sencilla.

Por otro lado, es importante conocer que, como se explica en [5], *Java* es un lenguaje orientado a objetos y tiene una extensa biblioteca de funciones en continuo crecimiento, tanto por las ampliaciones que realiza *Sun* (los creadores de *Java*), como por desarrollos de otras empresas, organismos, o particulares que liberan su desarrollo. Además, al ser un lenguaje tan extendido se cuenta con una amplia gama de documentación de apoyo para la implementación de cualquier aplicación con *Java*.

2.6.1 Historia

En base a [21] los inicios de *Java* se remontan a los años 90. En un principio se creó bajo la dirección de James Gosling como una herramienta de programación para ser usada en un proyecto de un dispositivo encargado de la recepción y opcionalmente decodificación de señal de televisión analógica. En un principio el lenguaje se denomino *Oak*, después paso a llamarse *Green* y finalmente se renombró como *Java*.

En cuanto a su desarrollo, *Java* nació de la necesidad de implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a *C++* y ya entre junio y junio de 1994, tras largos días de trabajo entre John Gaga, James Gosling, Joy Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. De esta manera nació por fin *Java* 1.0, pero hasta el 23 de mayo de 1995 *Java* y su navegador Web *HotJava* en las conferencias de *SunWorld,* no vieron la luz pública. Este acontecimiento fue anunciado por el director científico de *Sun Microsystems*, acto que estuvo acompañado con el anuncio de que *Java* sería soportado en los navegadores de *Netscape*.

En 1996 se fundó el grupo empresarial *Java*Soft para que se encargara del desarrollo tecnológico de *Java* y se publicó su primera versión.

Según [6][1][8] *Java* fue inicialmente ofrecido como tecnología para realizar sitios Web con programas que se ejecutaban en navegadores Web. Hoy en día, tiende más a encontrarse en servidores, dirigiendo aplicaciones Web dinámicas respaldadas por bases de datos relacionales en algunos de los portales más grandes de la Web. Según va madurando, este lenguaje se convierte en una herramienta capaz de desarrollar

aplicaciones de propósito general para medios que son distintos a los navegadores Web, incluso ha madurado para llegar a convertirse en un importante competidor para otros lenguajes de desarrollo de propósito general como *C++*, *Perl, Python, Ruby y Visual Basic*.

En la actualidad acaba de ser lanzada la nueva plataforma *Java* SE 7. Esta nueva versión de la colección de *APIs* del lenguaje de programación *Java* útiles para muchos programas de la Plataforma *Java*, simplifica la sintaxis y soporta Unicode 6 y lenguajes como *Ruby, Python y JavaScript*.

Java SE **7** es la primera versión bajo la dirección de *Oracle*, viene además con el nuevo *framework Fork/Join* que permitirá descomponer problemas más fácilmente y ejecutar tareas en paralelo.

2.6.2 ¿Qué es *Java*?

En base a [9] *Java e*s un lenguaje de programación orientado a objetos con gran peso en Internet gracias a su plataforma *J2EE*. No obstante, *Java* no se queda ahí, ya que en la industria de la programación para teléfonos móviles tiene gran peso y también está disponible para desarrollar aplicaciones de uso general, de uso personal, para puestos de trabajo, aplicaciones de bases de datos y además es en cierta manera el heredero legítimo de *C++* pues preserva gran parte de su sintaxis, aunque presenta cambios sustanciales en las características como lenguaje orientado a objetos. Cambios como la eliminación de la aritmética de los punteros (un puntero es una variable que apunta a una dirección de memoria) puesto que ahora es la propia máquina virtual la encargada de las referencias y gestión de la memoria, aparece también el concepto de *interfaz* que engloba un conjunto de especificaciones de métodos y de atributos constantes, aparece también el concepto *paquete*, el cual es una entidad organizativa que permite agrupar clases, interfaces y excepciones y el concepto de *herencia simple* que indica que ahora una clase sólo heredará comportamientos y métodos de una sola superclase.

Por otro lado la expansión de este lenguaje entre la comunidad de programadores ha sido vertiginosa convirtiéndose en uno de los lenguajes de programación orientados a objetos más importantes. Por ejemplo, en el entorno académico e investigador, *Java* está reemplazando lenguajes de programación estructurada como *Pascal* e incluso *C* que siempre fueron considerados lenguajes de elección para la introducción a la programación.

Las características más importantes [6] son:

- ***ES UN LENGUAJE MULTIPLATAFORMA***

  El código binario producido por un compilador de *Java* es universal, por lo que un programa escrito en *Java* funcionará correctamente en las principales plataformas del mercado como *Unix, Linux, OS/390, Windows,* ó *HP-UX* entre otros sistemas operativos para ordenadores personales o estaciones de trabajo, y *Palm OS* ó *EPOC* entre otros sistemas operativos para dispositivos de telefonía móvil.

- ***ES UN LENGUAJE ORIENTADO A OBJETOS***

  Java es un lenguaje orientado a objetos debido a que los módulos fundamentales de programación son las clases, y no existen funciones independientes, es decir, toda variable y todo método utilizado pertenece a una clase. Además, *Java* ofrece las características del paradigma orientado a objetos como la **herencia** que permite reutilizar el comportamiento de una clase en la definición de otra nueva [9], el **polimorfismo** asociado a las funciones, de modo que una función polimorfa es aquella que es capaz de ser aplicada de igual manera a varios objetos [9], o las **interfaces** que viene a ser una colección de métodos abstractos y propiedades donde se indica que se debe hacer pero no su implementación, etc. [20].

- ***POSEE UNA EXCELENTE BIBLIOTECA DE CLASES***

  El entorno de ejecución de *Java* está complementado con una extensa colección de clases agrupadas en subdirectorios. De esta manera muchas de las funciones más comunes ya vienen implementadas en estas clases y solo haría falta implementar la parte correspondiente a la funcionalidad que cada uno quiera darle.

2.6.3 Diseño de interfaces gráficas con *Java*

Como se indica en [8] el desarrollo de aplicaciones que utilizan interfaces gráficas a través de ventanas es una realidad, pues tiene como ventaja que todas las ventanas se comportan de manera idéntica en cualquier plataforma (*Windows, Linux*…) y en muchas ocasiones utilizan los mismos componentes básicos para introducir órdenes como el botón de cerrar ventana, maximizar ventana o minimizarla.

Además, como podemos ver en [8], la parte fundamental de una interfaz gráfica es el formulario que, dependiendo de la utilidad, puede servir para visualizar un gráfico, información o aceptar datos, además sobre él se dibujan otros objetos llamados componentes o controles: cajas de texto, botones, etiquetas, barras de desplazamiento, etc.

Una vez diseñada la interfaz gráfica, como indica [9], cada objeto del formulario debe ir ligado a un código que dicta su propio comportamiento, es decir, cada vez que ocurre un evento sobre un componente, se ejecuta el comportamiento que se le ha implementado, es decir, una aplicación con una interfaz gráfica trabaja mediante una comunicación entre los objetos a través de mensajes, y éstos a su vez son producidos por eventos.

Cuando ocurre un suceso sobre un objeto, éste notifica mediante un evento tal hecho. Hay varios tipos de eventos: evento producido por un usuario, por ejemplo, cuando se pulsa una tecla; evento producido por el sistema, por ejemplo, cuando transcurre un determinado tiempo; o un evento producido indirectamente por el código, por ejemplo, cuando el código carga una ventana. La respuesta a un evento es la ejecución de un

método del objeto sobre el cual se ha producido el evento, ya que cada objeto puede responder a un conjunto de métodos predefinidos.

En resumen, para programar una aplicación basada en ventanas, hay que escribir código para cada objeto, en general por separado, quedando la aplicación dividida en pequeños métodos cada uno correspondiente a un evento. Un ejemplo de esto es un evento de ratón, como el evento *MouseClicked*, que cuando se produce sobre un botón se debe escribir código que responda al momento en el que se dio clic al botón, es decir, si se da clic al botón *Pintar Círculo*, se pintará un círculo y ese "pintar círculo" estará implementado dentro de un método que se ejecuta al dar clic al botón.

A continuación se explica brevemente, gracias a la información sustraída de [6].[8], las herramientas más importantes que aporta *Java* de cara a desarrollar una interfaz gráfica:

1. *SWING*

   Java proporciona una biblioteca de clases *JFC* (*Java Foundation Classes*) agrupadas en las *APIs AWT* y *SWING*. La primera oferta de *Java* en lo referido a interfaces gráficas de usuario que contienen los componentes gráficos y sus funciones se denomina *AWT*. El inconveniente es que las interfaces gráficas que se construyen mediante *AWT* distan mucho de una interfaz de usuario que ofrezca excelentes prestaciones al usuario. Ante esto, en *Java* apareció una segunda biblioteca de clases cuya base es *AWT* pero con la diferencia de que su conjunto de componentes están escritos en *Java* y se ejecutan en cualquier plataforma nativa que soporte la máquina virtual de *Java*. Esta segunda biblioteca se denomina *SWING*.

   La funcionalidad de *SWING* es ofrecer todas las herramientas posibles para poder crear una interfaz gráfica de usuario en la que además se puedan recoger los datos de entrada mediante el ratón, teclado y otros dispositivos de entrada, siendo todo esto posible gracias a los componentes que ofrece como, por ejemplo, marcos, contenedores, botones, etiquetas, campos de texto, o áreas de texto.

2. Componentes

   Todos los componentes son objetos derivados de la clase *JComponent*, que a su vez se deriva de la clase ***Java.AWT*.Component,** lo que pone de manifiesto que *SWING* deriva de *AWT*.

   Para combinar los componentes *SWING* formando una interfaz gráfica, existen los contenedores de nivel alto como *JWindow*, el cual es una ventana sin barra de título y sin botones que permite su manipulación, *JFrame* por el contrario es una ventana con barra de título y con botones para manipulación, *JDialog* permite visualizar una caja de dialogo y *JApplet* es un programa que visualiza una interfaz gráfica en el contexto de una página *Web*. Por ejemplo uno de los componentes más importantes de una aplicación es un *frame* de la clase contenedor *JFrame* que, como su nombre expone, es la clase que contiene a todos los demás componentes que hacen posible el funcionamiento de una aplicación. Otra clase importante es la clase *JPanel*, que representa un

panel donde se puede dibujar lo que se desee. Por último, paso a numerar el resto de componentes más relevantes que conforman una aplicación:

- Etiquetas: componentes *JLabel*. Son cajas de texto no modificables por el usuario. Su función es informar al usuario que tiene que hacer y cuál es la función de cada componente. Pueden ser texto o una imagen o los dos.

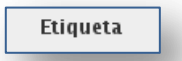


Figura 1. Etiqueta.

- Cajas de texto: componentes *JTextField*. Son cajas de texto de una sola línea y permite editar una única línea de texto.

Figura 2. Caja de texto.

- Botones: componentes *JButton*. Es un botón de pulsación que permite al usuario ejecutar una acción cuando sea preciso.

Figura 3. Botón.

- Cuadro de selección: componentes *JCheckBox*. Es una casilla de verificación que muestra gráficamente su estado, seleccionada o deseleccionada, al usuario.

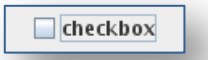


Figura 4. Cuadro de selección.

3. Gestor de distribución

Un gestor de distribución es útil para organizar los componentes de una interfaz de usuario en *Java*, ya que determinan como se ordenarán cuando se añadan a un contenedor, entendiendo por contenedor un componente SWING utilizado para ubicar otros componentes, por ejemplo una ventana es un contenedor. Existen distintos tipos de gestores según los intereses del diseñador de la interfaz de usuario, como por ejemplo:

- *FlowLayout.* Ubica los componentes en el contenedor de izquierda a derecha.
- *BoxLayout.* Ubica los componentes en el contenedor en una única fila o columna.
- *GridLayout.* Ubica los componentes en el contenedor en filas y columnas.
- *BorderLayout.* Divide al contenedor en cinco secciones (norte, sur, este, oeste y centro) y ubica los componentes en una sección u otra según decida el diseñador.
- *Absolute-Layout*: Permite ubicar los componentes en el contenedor exactamente donde se desee.

Cabe destacar que incluso se pueden mezclar distintos gestores de distribución en una misma interfaz gráfica de *Java*.

4. Eventos

Para que una interfaz de usuario responda a las acciones ejercidas por éste, es necesario implementar que la interfaz sea conducida por eventos, es decir, cuando se produce una acción sobre un objeto se produce un evento, entendiendo por evento, un mensaje que envía un objeto a otro objeto. Los eventos se manipulan gracias a un conjunto de interfaces llamadas escuchadores (*listeners),* la función de estos objetos es "escuchar" cuando ocurre un evento sobre el componente al que esté asociado para así ejecutar el método asociado para que realice todas aquellas acciones que se hayan considerado. Un componente tendrá tantos escuchadores como tipos de eventos tenga que manejar.

La figura 5 tomada de [8] explica gráficamente lo mencionado anteriormente, empleando un esquema donde en un componente se ha producido un evento, el escuchador lo procesa para mandar a ejecutar el método asociado a ese componente como respuesta al evento ocurrido.

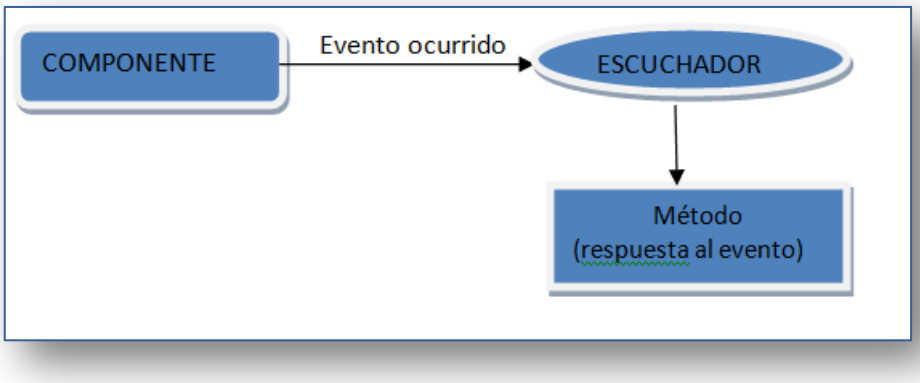


Figura 5. Proceso que describe cuando ocurre un evento [8].

En base a [8] algunos de los eventos más frecuentemente empleados son los descritos a continuación:

- **Eventos de acción**: el escuchador *ActionListener* permite a un componente responder a las acciones que ocurren sobre él, es decir, a eventos de tipo *ActionEvent* como, por ejemplo, un clic sobre un botón. La manera de implementarlo es la siguiente:

Se crea una clase que implemente la interfaz que tenga los métodos que permiten responder al tipo de eventos que se quieren escuchar, en este caso se trata de la interfaz *ActionListener*.

En la figura 6 se observa que la clase *ManejadorBoton* implementa a la clase *ActionListener* redefiniendo el método *actionPerformed* y dentro de este método se implementa la respuesta al evento producido.

```
Class ManejadorBoton implements java.awt.event.Actionlistener
{
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
                //responder al evento evt
        }
}
```

Figura 6. Clase Manejador Botón, ejemplo de clase con métodos que responden a un tipo de eventos.

Como se observa en la figura 7 se construye un objeto de la clase *ActionListener* que tendrá redefinido el método *ActionPerformed* con la implementación de la respuesta al evento ocurrido y por último este objeto se vincula al manejador de eventos del componente con la sintaxis *Jcomponente.addActionListener* (objeto de la clase *ActionListener*).

```
java.awt.event.Actionlistener al = new java.awt.event.Actionlistener()
{
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
                //responder al evento evt
        }
};
JBoton.addActionListener(al);
```

Figura 7. Ejemplo 2 de código manejador de eventos.

Por último, es importante mencionar que en la figura 7 se observa que el método *ActionPerformed* recibe como argumento un evento de tipo *ActionEvent*. Dicho argumento no es más que un objeto que proporciona información del componente que originó el evento.

- **Eventos de movimiento de ratón**: estos eventos suceden cuando hay un movimiento del ratón sobre un componente. Una clase debe implementar la interfaz *MouseMotionListener* para apoyarlos, por consiguiente en esta interfaz hay dos métodos *MouseDragged(MouseEvent)* y *MouseMoved(MouseEvent)*, como se puede observar estos métodos también utilizan los objetos *MouseEvent.*

- **Eventos de ratón**: los eventos de ratón se generan mediante un clic de ratón, entrando en un área de componente o saliendo de un área de componente. Los métodos correspondientes son *MouseClicked(MouseEvent), MouseEntered(MouseEvent), MouseExited(MouseEvent), MousePressed(MouseEvent), MouseReleased(MouseEvent).* Cualquier componente puede generar estos eventos que se implementan en una clase a través de la interfaz *MouseListener.*

  En la figura 8 se observa que cuando se desea implementar la respuesta a un evento de ratón, en este caso, el de presionar sobre un componente, se implementa la clase *mousePressed* como otra clase cualquiera, escribiendo el código de la respuesta a este evento dentro de la clase.

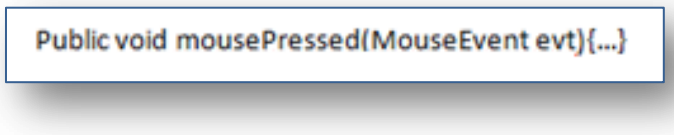


Figura 8. Evento de ratón.

Los siguientes métodos se pueden usar como objetos *MouseEvent* pasados por parámetro a otros métodos*:*

- *getClickCount(). D*evuelve el número de *clicks* de ratón sobre un componente.
- *getPoint().* Devuelve la posición en coordenadas *x* e *y* del componente donde se ha dado *click.*
- *getX().* Devuelve la coordenada *x* del componente donde se ha dado *click.*
- *getY().* Devuelve la coordenada *x* del componente donde se ha dado *click.*

5. Método *Paint*.

El método *Paint* se encarga de pintar los componentes, es decir, un componente (objeto de una clase de *JComponent*) se repinta desde el método *paint(Graphics g)* donde el parámetro *g* es el contexto gráfico asociado al componente.

Este método es invocado directamente con una llamada a *repaint()*, de manera que cuando esto ocurre, el método *Paint* se ejecuta y todo lo que está definido dentro de él se vuelve a pintar.

La manera de implementar gráficos dentro de este método es utilizando la clase *Graphics* y sus métodos. Algunos de ellos se muestran en la Figura 9, en ella se puede observar el método *drawString()* encargado de pintar texto en unas coordenadas determinadas, el método *drawLine()* encargado de pintar una línea, especificando las coordenados de inicio y de fin donde debe dibujarse la línea y por último el método *drawRect()*, encargado de pintar un rectángulo, especificando las coordenados junto con el ancho y alto del rectángulo:

```
void drawString(String str.int x. int y)
void drawLine(int x1, int y1,int x2, int y2)
void drawRect(int x, int y.int ancho.int alto)
```

Figura 9. Métodos Graphics.

En la figura 10 se especifica la manera de implementar el pintado sólo en un componente específico, es decir, si una interfaz gráfica está compuesta por un *frame* y un panel y se desea pintar sólo en el panel, se debe utilizar el objeto *Graphics* del objeto *PanelPrincipal* como se observa en la figura 13, para que posteriormente se utilice el método *drawImage* del objeto *PanelPrincipal*, de esta manera sólo se dibujará en las coordenadas del panel, la imagen y no en otra zona de la interfaz gráfica.

```
Graphics g2 = PanelPrincipal.getGraphics ();
g2.drawImage (nod, X, Y, 30, 30, this);
```

Figura 10. Componente Graphics del Panel Principal.

2.6.4 Hilos

Una de las principales ventajas del lenguaje de programación *Java* es la utilización de hilos para introducir la multitarea en una aplicación. En este apartado se explica de manera general el porqué es necesario escribir un programa con hilos.

Determinadas tareas intensivas de una aplicación pueden tomar bastante tiempo del procesador, sobre todo en programas con interfaz de usuario gráfica. Para resolver este problema se pueden implementar funciones que acaparen el procesador en una clase *Java* de modo que se ejecuten paralelamente del resto del programa. Esto es posible gracias a los hilos, los cuales se implementan en *Java* con la clase *Thread* en el paquete *Java.lang*.

Los hilos forman parte de un programa preparado para ejecutarse paralelamente al resto del programa, de esta manera poniendo la carga del programa en un hilo, se estará liberando el resto de programa para ejecutar otras tareas. Esto también se conoce como multitarea porque el programa puede manejar más de una tarea a la vez.

La figura 11 representa una aplicación *Java* que ejecuta varios hilos a la vez, por ejemplo, en una aplicación cualquiera, se puede dar el caso que se deban ejecutar tres tareas a la vez, tareas como, pintar a la vez que se busca una determinada información o a la vez que se reproduce determinado contenido multimedia por ejemplo.

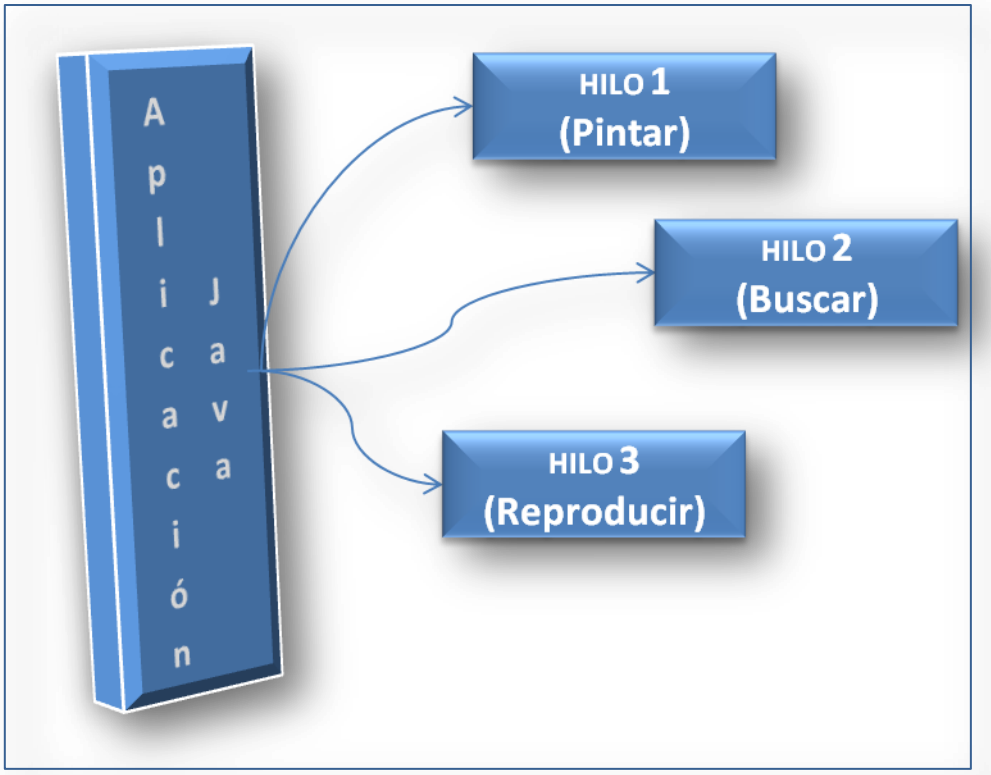


Figura 11. Esquema de Multitarea.

2.7 Bases de datos

En base a [10] hoy en día el uso de bases de datos y los sistemas de bases de datos son algo prácticamente imprescindible que se encuentran en cualquier rincón donde se quiera mirar, desde una simple base de datos donde se guardan los materiales que se venden en una tienda de electrodomésticos, o la información referida al personal de una empresa, comprar cualquier producto *online***,** o reservar un billete de avión. Por lo tanto, gran cantidad de acciones de la vida cotidiana de un gran número de personas implican cierta interacción con una base de datos. Este tipo de base de datos se pueden llamar *aplicaciones de bases de datos tradicionales* en la que la mayor parte de la información que se halla almacenada, es información de tipo numérica o textual.

Sin embargo, este tipo de bases de datos no tiene nada que ver con las nuevas bases de datos que han surgido como consecuencia de los constantes avances tecnológicos. Un cambio destacable es el tipo de información almacenada, la cual pasa de ser numérico o textual, a ser información de tipo imagen, audio o vídeo para una base de datos de contenido multimedia; información temporal que utilizan las tecnologías de tiempo real; bases de datos activas que se utilizan para controlar procesos industriales y de fabricación; y la información espacial, que se encuentra almacenada en bases de datos espaciales.

Como se puede observar, el mundo de las bases de datos es inagotable conforme surgen nuevas necesidades de almacenar nuevos tipos de información, en ocasiones sustraída de investigaciones de cualquier ámbito, sea científico, industrial, cultural, histórico, lingüístico...etc. Los sistemas gestores de bases de datos deben evolucionar, ya que son ellos los que gestionan las bases de datos.

2.7.1 ¿Qué es una Base de datos?

Una base de datos es una estructura que permite almacenar una gran cantidad de información relacionada entre sí para ser procesada de manera rápida y eficaz. Surge como consecuencia de la necesidad de controlar y manejar grandes flujos de datos.

En base a [10] una base de datos tiene las siguientes propiedades implícitas:

- Una base de datos representa algunos aspectos de la vida real, en algunos casos denominado *minimundo* o *Universo del discurso (UdD).* Los cambios en aquel *minimundo* se reflejan en la base de datos.
- Una base de datos es un conjunto coherente de datos, de manera que no tendría sentido tener, en una base de datos, información que no tuviera ninguna relación entre ella.
- Por último, una base de datos se diseña, se construye y se rellena de información por un objetivo específico. Está destinada a un grupo de usuarios.

2.7.2 ¿Qué es un SGBD?

Tal y como indica [11], un *SGBD (Sistema Gestor de Base de datos)* es el Software diseñado para colaborar con el mantenimiento y empleo de grandes conjuntos de datos. Aunque existen muchos tipos de *SGBD,* el tipo dominante hoy en día es el Sistema Gestor de Base de Datos Relacionales. La alternativa a un *SGBD* es almacenar la información en archivos y escribir código específico para una aplicación que lo gestione, algo que es, evidentemente, más complicado.

Como ventajas de un SGBD existen las siguientes:

- Independencia con respecto a los datos.
- Emplean gran variedad de técnicas sofisticadas para almacenar y recuperar los datos de manera eficiente.
- Integridad y seguridad de los datos.
- Cuando varios usuarios comparte los mismos datos, la centralización de la administración de estos, puede ofrecer mejoras significativas.
- Acceso concurrente y recuperación en caso de fallo.
- Reducción del tiempo de desarrollo de las aplicaciones.

En resumen, según explica el esquema de la figura 12 un sistema de base de datos lo componen unos usuarios o programadores que mediante una interfaz de usuario o aplicación acceden con unas consultas a la base de datos, estas consultas son procesadas por el *SGBD.*

Este *SGBD* está compuesto por el software para procesar las consultas que escribe el usuario en el programa de aplicación y por el software para acceder a los datos almacenados. Gracias a este software se obtiene información de la definición de la base de datos y se accede a la base de datos almacenada, para obtener de ella la información deseada.

En la figura 12 se refleja muy bien lo que significa un sistema de base de datos.

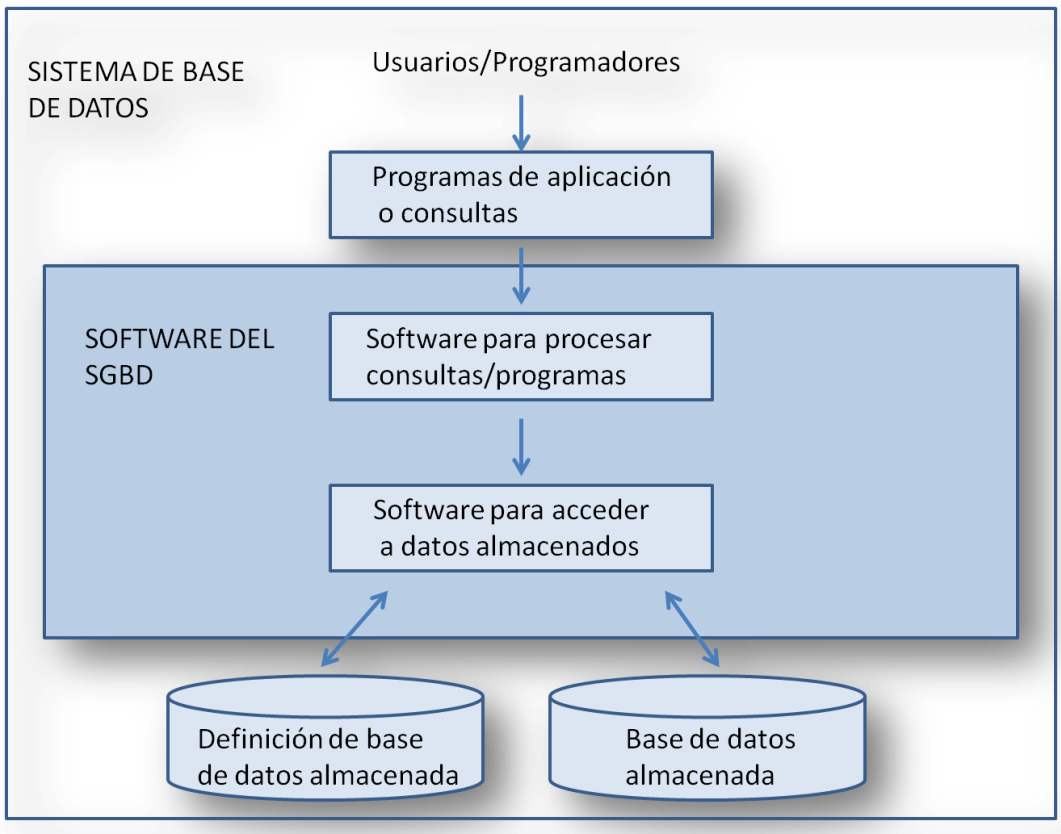


Figura 12. Sistema de base de datos *[11].*

2.7.3 Evolución de las bases de datos

En base a [12] las técnicas de almacenamiento y procesamiento de datos han evolucionado a lo largo de los años:

- Años 50 y principios de años 60:

    - El término bases de datos fue escuchado por primera vez en un simposio celebrado en California en 1963.
    - Se desarrollaron las cintas magnéticas para el almacenamiento de datos y el procesamiento de estos, que consistía en leer datos secuencialmente de una o varias cintas y escribir datos en una nueva cinta. El tamaño de los datos era mucho mayor que la memoria principal.

- Finales de los años 60 y principios de los 70:

    - Se empezaron a usar los discos duros. La ventaja de ellos es que se eliminó la secuencialidad, ya que se podía tener acceso a cualquier parte del disco.
    - Se crearon las bases de datos de red y las bases de datos jerárquicas, que permitieron que estructuras de datos como las listas y los árboles pudieran almacenarse en disco, con lo que los programadores pudieron crear y manipular este tipo de estructuras.
    - Codd definió el modelo Relacional y por consiguiente nacieron las bases de datos relacionales.

- Años 80:

    - Primeros sistemas comerciales de bases de datos relacionales como *DB2*, *IBM*, *Oracle*, *Ingres* y *Rdb* de *DEC*.
    - Las bases de datos relacionales sustituyeron a las bases de datos de red y las bases de datos jerárquicas.
    - Se empezó a investigar en las bases de datos paralelas y distribuidas, así como en las bases de datos orientadas a objetos.

- Años 90:

    - El *lenguaje SQL* se diseñó fundamentalmente para las aplicaciones de ayuda a la toma de decisiones.
    - El uso del *SGBD* experimentó un gran crecimiento.

- Finales de los 90:

- Con el nacimiento de la *World Wide Web* los sistemas de bases de datos tenían que soportar tasas de procesamiento de transacciones muy elevadas, así como una fiabilidad muy alta y disponibilidad 24x7.
- Los sistemas de bases de datos tenían que soportar interfaces *Web* para los datos.

- Principios del siglo XXI:

  - Hoy en día este campo está siendo explotado gracias a proyectos interesantes como proyectos científicos, industriales, empresariales, sociales, etc.
  - Comercialmente los sistemas gestores de bases de datos representan uno de los mayores y más vigorosos segmentos del mercado, como ejemplo, la base de datos más avanzada ha sido *Oracle*, que hace años era la única que cumplía las exigencias de seguridad y escalabilidad del gobierno americano. No obstante, hoy en día el software de almacenamiento de información se encuentra ya muy igualado. Entre las bases de datos actuales se pueden distinguir las marcas comerciales: *Oracle*, *SQLServer*, *DB2*, etc. y las marcas de software libre: *MySQL*, *PostgreSQ*, etc.

2.7.4 Visión global de una Base de datos Relacional

Para diseñar una base de datos es importante tener claramente definidos los principales fundamentos del modelo Relacional como lo son las relaciones. En base a [11] cada relación consiste en un esquema de relación y un ejemplar de relación. El ejemplar de la relación es un conjunto de tuplas o registros que forman una tabla y el esquema de la relación son las cabeceras de las columnas de esa tabla, que indican el nombre de la relación, el de cada campo y el dominio de cada campo.

En la figura 13 se puede apreciar la estructura de una tabla.

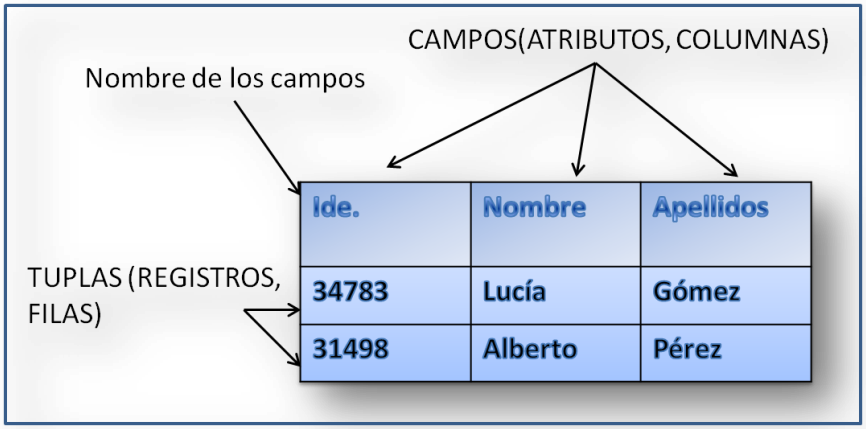


Figura 13. Ejemplar de una relación [11].

Por tanto un ejemplar de una base de datos Relacional es un conjunto de relaciones, uno por cada esquema de relación del esquema de la base de datos. Evidentemente cada ejemplar de relación debe cumplir las restricciones de dominio de su esquema.

El lenguaje de creación, eliminación, modificación de las tablas y que obliga a cumplir las restricciones de integridad de una base de datos es el Leguaje *SQL* ya que es el lenguaje estándar ANSI/ISO de definición, manipulación y control de bases de datos relacionales. Es un lenguaje declarativo: sólo hay que indicar qué se quiere hacer. Gracias a este lenguaje se puede especificar las restricciones de integridad como de *clave principal*, de *clave externa* y por otro lado controlar las inserciones o los borrados.

En el apartado de diseño de la base de datos, se especificará con más detalle el modelo Relacional de una base de datos acompañados de un ejemplo explicativo de cómo se diseña una base de datos.

2.7.5 GEODATABASES (Bases de Datos Espaciales)

Existen distintos tipos de bases de datos, como bases de datos tradicionales, bases de datos temporales, distribuidas, etc. Pero el tipo de base de datos elegida para este estudio, es una base de datos espacial por el tipo de información que se manejará en la aplicación.

Para entender que es una base de datos espacial primero es necesario saber **¿Qué es un GIS?**, puesto que este tipo de base de datos nace de la necesidad de analizar tipos de datos territoriales, tarea principal de un *GIS (Sistema de información Geográfica).*

Según [15] un *GIS*, es un sistema hardware, software y procedimientos diseñados para soportar la captura, administración, manipulación, análisis, modelado y representación grafica de datos u otros objetos referenciados espacialmente para resolver problemas complejos de planeación y administración.

Puesto que la información espacial solo toma valor de manera relativa a un sistema de referencia espacial, junto a la anterior definición también es necesario incluir una definición de lo que se entiende por ellos. Como se indica en [12] los sistemas de referencia espacial pueden ser de dos tipos: georrefenciados (aquellos que se establecen sobre la superficie terrestre) y no georreferenciados (son sistemas que tienen valor físico, pero que pueden ser útiles en determinadas situaciones).

Por tanto un *GIS* puede ser visto desde tres puntos de vista:

- Como una herramienta para recolectar, almacenar, recuperar, procesar y desplegar datos del mundo real.

- Como una base de datos capacitada para operar un conjunto de procedimientos que responden a consultas acerca de entidades en la base de datos.

- Como una organización para la toma de decisiones que involucra la integración de datos referenciados espacialmente con sus propiedades reales.

Como se indica en [15], en función del modelo de datos implementado en cada sistema, se pueden distinguir tres grandes grupos de *GIS:*

- *GIS Vectoriales:* permite representar información más precisa simplemente representando unos puntos en el plano.

- *GIS Raster:* no pueden representar información muy precisa. Se divide el plano en cuadrados y a cada uno se le asocia un número con una definición, por ejemplo: 1-agua, 2-bosque.

- *GIS con modelo de datos orientado a objetos:* a diferencia de los *GIS Vectoriales y Raster* que estructuran su información mediante capas, este sistema intenta organizar la información geográfica a partir del propio objeto

geográfico y sus relaciones con otros. Además, este tipo de sistemas añade dinamismo a la información incluida en el sistema, por eso este modelo es más aconsejable para situaciones en las que la naturaleza se somete a cambios tanto en el tiempo como en el espacio.

Su ámbito de utilización es muy amplio debido a que puede ser empleado en cualquier disciplina que necesite la combinación de planos cartográficos y bases de datos, por ejemplo, algunas de las ramas que utilizan este sistema son:

- Ingeniería Civil.
- Arquitectura.
- Geología.
- Recursos Naturales.
- Agricultura.
- Arqueología.
- Forestación.
- Servicios de emergencia.
- Planificación urbana en las áreas de catastros, energía, redes telefónicas.
- Estudios Sociales.

Todas estas ramas de la ciencia y otras muchas han encontrado en un *GIS* características importantes como:

- Integración de datos: tanto datos de localización como de caracterización se encuentran enlazados.

- Análisis topológico de la información, ya que mediante las relaciones de las entidades se pueden ejecutar consultas del tipo: ¿cuántos objetos *x* se encuentran en un perímetro de *n* Km?

- Actualización inmediata de los mapas digitales.

- Representación tridimensional de los fenómenos de estudio.

- Divide la información en capas temáticas

En la figura 14 se aprecia como un *GIS* divide la información terrestre en capas temáticas, de abajo hacia arriba, divide en capas de realidad, usos del suelo, altitud, parcelas, vías y por último productores.

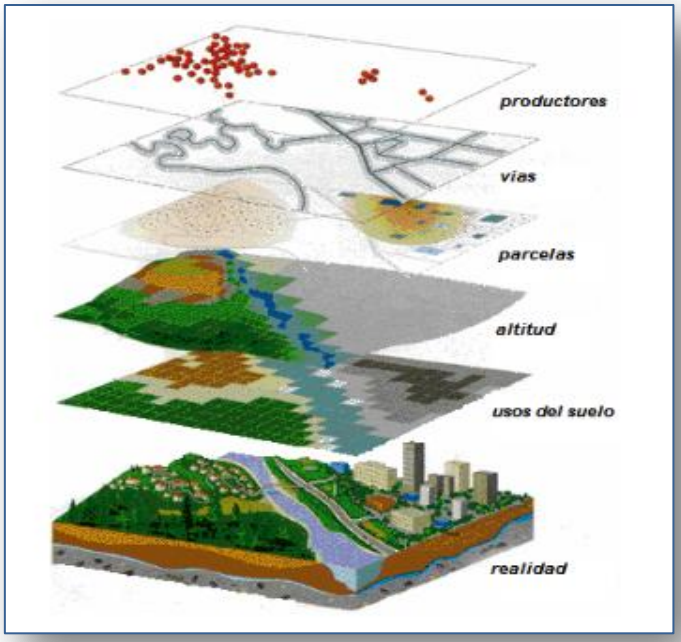


Figura 14. Superposicionamiento de capas de información [15].

2.7.5.1 Definición de GEODATABASE

En base a [15], una *GEODATABASE* es una colección de datos espaciales organizados de tal manera que sirva para una o varias aplicaciones *GIS*. Con respecto a los datos espaciales éstos representan la geografía como formas geométricas, redes, superficies, ubicaciones e imágenes, a los cuales se les asigna sus respectivos atributos que los definen.

Como indica [13] un dato espacial es una variable que hace referencia a una posición física en la Tierra mediante la representación de puntos, los cuales están determinados por las coordenadas terrestres medidas por latitud y longitud, líneas que cubren una distancia dada, uniendo Nodos o puntos por lo que también se pueden considerar arcos debido a la forma esférica de la Tierra, y polígonos que representan figuras planas conectadas por distintas líneas u objetos cerrados que cubren un área determinada.

Teniendo en cuenta el modo de aproximación del modelado de las ubicaciones geográficas se diferencian dos tipos de modelo. El primero de ellos es el *modelo Espacial Geodésico* que indica que las ubicaciones sobre la superficie del planeta son descritas en términos de latitud y longitud, las cuales son medidas en grados, minutos y segundos. Según [16] como modelos geodésicos existentes se menciona el elipsoide *Airy 1830* usado en el sistema geográfico de Ordenamiento de tierras de Gran Bretaña y el elipsoide *WGS84* utilizado a nivel mundial en soluciones de GPS. El segundo es el *modelo Espacial Planar*, el cual utiliza la superficie terrestre en forma de plano. En la figura 15 se representan los dos modelos espaciales, el *Geodésico* y el *Planar*.

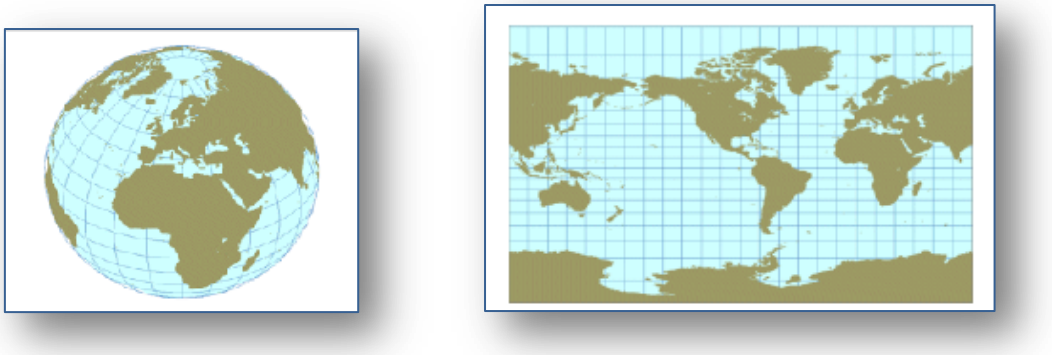


Figura 15. Modelo Geodésico y Modelo Planar [16].

Una vez definidos los tipos de datos que se almacenan y los sistemas de referencia existentes, se especifica que se pueden realizar los siguientes tipos de consulta:

- De rango: para consultas en las que se necesite información de dentro de un área.
- De vecinidad: para consultas en las que sólo se necesite la información más cercana un punto.

Como indica [14], además de las características mencionadas anteriormente, una *GEODATABASE* presenta también los siguientes aspectos:

- Cuando este tipo de base de datos reside en un sistema de administración de base de datos estándar (*Oracle, Microsoft SQL Server, IBM DB2, Informix y Microsoft Access*) permite aprovechar todo el potencial de las herramientas de estos sistemas además de completar la funcionalidad presente en la base de datos con funciones necesarias para el tratamiento de información espacial.

- Facilita la generación de una visión más completa de la realidad al almacenar también el comportamiento de los elementos geográficos que almacena.

- Su modelo de datos es escalable y en función de la necesidad de cada organización se puede diferenciar entre:

    - *GEODATABASE* basada en archivos
    - *GEODATABASE* personal, implementada por *Microsoft Access*
    - *GEODATABASE* Corporativa, implementada sobre *Oracle, Microsoft SQL Server, IBM DB2 o Informix.*

En base a [14] una *GEODATABASE* tiene su gran importancia con respecto a otros modelos de datos gracias a los numerosos beneficios que implica su uso como:

- **Administración de datos centralizada**: debido a que los datos de este tipo de base de datos son almacenados en sistemas de administración de bases de datos

comerciales o en sistemas de archivos, los cuales constituyen un repositorio común.

- **Edición multiusuario**: es posible realizar tareas multiusuario a través de los mecanismos de los sistemas de administración de las bases de datos.

- **Implementación de comportamiento**: se puede trabajar con elementos más intuitivos, ya que el implementar su comportamiento se acerca más a la realidad.

- **Tecnología *COM***: al ser desarrollado utilizando estándares *COM*, permite la integración con otros sistemas.

- **Acceso a las *GEODATABASES***: el acceso a este tipo de base de datos se puede realizar a través de los menús estándares como *ArcCatalog*, *ArcMap*, *ArcToolbox*, así mismo los programadores pueden utilizar *APIs* como *ArcObjects*, *OLE DB* y *SQL* incluidos con el software.

- **Replicación**: la información geográfica se permite distribuir en dos o más *GEODATABASES*.

- **Históricos**: gracias a que se almacenan todos los cambios producidos en este tipo de base de datos, es posible consultar una versión histórica del estado de la base de datos en un momento dado.

2.7.5.2 Distintas bases de datos espaciales

Según ha avanzado la tecnología ha cambiado la forma del tratamiento de los datos, una muestra de ello es como ha crecido el número de bases de datos que cuentan con la característica de poder almacenar también datos espaciales. Hoy en día se cuenta con las siguientes bases de datos espaciales:

- ***INFORMIX***
- ***Oracle Spatial***
- ***IBM DB2 Spatial extender***
- ***Postgre SQL/Postgis***
- ***MySQL Spatial Extensions***

Es interesante tener una visión global sobre cómo funcionan estas bases de datos espaciales existentes hoy en día, de cara a elegir la mejor opción posible. Por esta razón los siguientes apartados ofrecen un breve resumen de algunas de ellas, en base al documento [16].

2.7.5.3 SQL Server 2008 Spatial

Incluye un soporte para datos geográficos para poder trabajar con los mismos. Las características incorporadas son:

- Usa tipos de datos geográficos tanto para almacenar datos espaciales geodésicos como datos espaciales planos y realizar operaciones sobre estos.
- Incorpora un nuevo índice espacial basado en árbol B para mejorar las consultas.
- El acceso al resultado de consultas espaciales es rápido y fácil.
- A través de formatos definidos *WKT (Well Known text), WKB (Well Known Binary), GML* (*Geographic Markup Language*) y especificaciones espaciales permite la integración con otras aplicaciones y utilidades disponibles en el mercado para importar, exportar y manejar datos espaciales.

2.7.5.4 DB2 Spatial y Geodetic Extender

IBM DB2 utiliza diferentes tecnologías para gestionar el modelo Geodésico con el extensor *Geodetic Extender* y para el modelo Planar con el extensor *Spatial Extender*, el cual se utiliza principalmente con conjuntos de datos regionales y locales que se pueden representar en coordenadas proyectadas sobre un plano y para aplicaciones en las que no es muy necesario demasiada precisión en la ubicación.

Una de las características de esta base de datos es que se pueden obtener datos espaciales de otros como, por ejemplo, direcciones con *Geocoding*, es decir, a partir de datos de entrada se pueden generar datos espaciales.

Debido a la naturaleza multidimensional de las consultas el índice *B-tree* nativo de *DB2* es insuficiente, por tanto se emplean dos nuevos índices: *Indice Spatial Grid* e *Indice Geodetic Voronoi.* El primero utiliza indexación en cuadrícula, el cual está diseñado para indexar datos espaciales multidimensionales, y el segundo considera a la Tierra como una esfera continua, organizando el acceso a los datos geodésicos mediante el uso de un mosaico de *Voronoi* de la superficie de la Tierra.

La figura 16 representa un mosaico de *Voronoi* de la superficie de la Tierra.

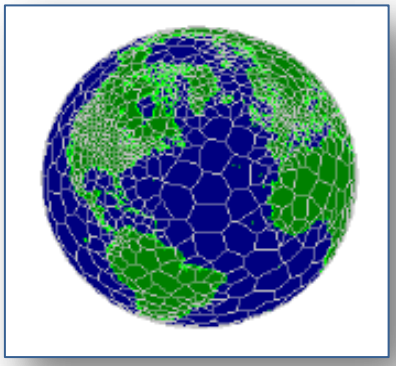


Figura 16. Mosaico de Voronoi de la superficie terrestre.

Por otro lado, DB2 *Spatial* Extender utiliza formatos estándar como *WKT, WKB, GML, ESRI Shape* al igual que *SQL Server 2008 Spatial.*

2.7.5.5 PostGIS

Es una extensión de la base de datos *PostgreSQL,* además es de código abierto y libre distribución, lo que posibilitó el desarrollo de soluciones corporativas con una mejor relación costo-beneficio. Permite el almacenamiento de objetos *GIS*, incluye soporte para índices espaciales *GiST* basados en *R-Tree* y funciones para el análisis y procesamiento de información espacial.

Algunas de las características de esta base de datos son**:**

- Para maximizar el rendimiento *PostGIS* utiliza una representación reducida de la geometría y de la estructura del índice, esto hace que el tamaño de los datos sea más pequeño y como consecuencia se extraiga más rápida esa información de un disco duro y se almacene en caché.

- Las consultas espaciales son optimizadas por sus índices *R-Tree* y su integración con *PostgreSQL query planner*.

- Para permitir que múltiples procesos trabajen concurrentemente y se asegure la integridad de los datos, *PostGIS* utiliza bloqueo a nivel de fila.

- Se cuenta con funcionalidades como soporte básico de topología, transformación de coordenadas y *APIs* de programación.

- Los objetos soportados por *PostGIS* son todos los objetos soportados y funciones especificados en la *OGC (Open Geospatial Consortium)*, y extiende el estándar con soporte de coordenadas *3DZ, 3DM* y *4D* extendiendo los formatos *WKB, WKT* en *EWKB* y *EWKT* respectivamente.

2.7.5.6 *Oracle* Spatial

Como indica [15], *Oracle Spatial* es un paquete que complementa a las bases de datos *Oracle* para permitir en ellas el procesamiento de datos espaciales. La primera versión espacial de *Oracle* fue la 8i R1 en el año 1999,y a partir de ahí sus características espaciales han ido mejorando. Las versiones posteriores a esta son *Oracle Spatial* 9i R1 y 9i R2, *Oracle Spatial* 10g R1 y 10g R2 y *Oracle Spatial* 11g R1 y 11g R2.

En base a [17][16], *Oracle Spatial* proporciona un esquema *SQL* y funciones que facilitan el almacenamiento, recuperación, actualización y consulta de los datos espaciales. En ella se pueden mezclar tanto datos espaciales como datos de otro tipo, facilitando así la relación entre ellos. Entre sus características se encuentran:

- Un esquema llamado *MDSYS* que establece el almacenamiento, la sintaxis y semántica de los tipos de datos espaciales soportados.

- Incorpora un mecanismo de indexación espacial.

- Modelo de datos topológicos para trabajar con datos de puntos, líneas y caras.

- Modelo de datos de red para representar objetos modelados como puntos y enlaces de una red.

- *GeoRaster*, característica que permite almacenar, indexar, consultar y analizar datos *GeoRaster*.

- 3D Geometry que permite el almacenamiento de objetos espaciales tridimensionales.

- Infraestructura de servicios web espaciales para realizar funciones espaciales.

- Servicios administrativos.

Además, *Oracle Spatial* es compatible con varios tipos de datos simples y geometrías de dos dimensiones como puntos, grupos de puntos, líneas rectas, líneas curvas, polígonos curvos, polígonos compuestos, líneas compuestas, círculos y rectángulos optimizados.

Aunque, *Oracle Spatial* también soporta geometrías tridimensionales y de cuatro dimensiones, donde tres o cuatro coordenadas se utilizan para definir cada vértice del objeto que se está definiendo. Por tanto el modelo de datos de *Oracle Spatial* está formado por elementos, geometrías y capas, donde las capas están compuestas por geometrías y éstas a su vez por elementos.

Una característica importante de los elementos (puntos, grupos de puntos, líneas rectas, líneas curvas…) es la tolerancia, la cual se utiliza para asociar un nivel de precisión a la representación de los datos espaciales, es decir, es la distancia que dos puntos pueden estar separados, sin embargo, cuanto menor sea la tolerancia mayor será la precisión con los datos.

En cuanto al modelo de consulta, *Oracle Spatial* tiene un modelo de consulta que consiste en dos filtros: filtro principal y segundo filtro. El primero permite la selección rápida de los registros que pasarán al segundo filtro comparando aproximaciones a la geometría para reducir la complejidad del cálculo, es decir, devuelve un subconjunto al segundo filtro de resultados más exactos. El segundo filtro devuelve la respuesta a la consulta espacial a partir del resultado del primer filtro. La operación del segundo filtro es computacionalmente más costosa.

Para la indexación de datos espaciales, *Oracle Spatial* utiliza la indexación *R-Tree* que consiste en aproximar cada geometría al mínimo rectángulo que encierre la geometría. Como características de este tipo de índice se tiene que si se utiliza consultas del tipo *vecino más cercano*, este índice es de los más rápidos, también ocurre para indexación de datos geodésicos con la consulta *SDO WITHIN DISTANCE* que calcula la distancia entre dos objetos espaciales.

2.7.5.7 Conclusión

Teniendo en cuenta las características relevantes mencionadas en los apartados anteriores de las distintas bases de datos espaciales que existen, se concluye que en la actualidad, cada una de ellas ha alcanzado un nivel adecuado de calidad para poder diseñar bases de datos espaciales con estos sistemas de administración de bases de datos estándar.

Como aspecto común de todas ellas, es que se basan en el modelo Geodésico o Planar según requiera el tratamiento de los datos. Además, todas implementan un tipo de índice

que ayuda en el tratamiento de los datos espaciales para tener mayor eficiencia y eficacia a la hora de acceder a ellos.

Pero hay que tener en cuenta que independientemente de las características que ofrezcan, hay bases de datos como *PostGIS* que son de libre distribución, por tanto abierta a todos los desarrolladores para su mejoría, en cambio, bases de datos como *Oracle Spatial* no, pero que cuenta con una amplia experiencia en este mercado, al ser una de las primeras en tener tratamiento para los datos espaciales, lo que ha conllevado a que sea una de las mejores opciones para implementar un sistema con tratamiento de datos espaciales, teniendo en cuenta además, que al pertenecer a *Oracle*, además de la experiencia con la que cuenta esta gran empresa, cuenta con un gran soporte para cualquier aplicación que haga uso de *Oracle Spatial*, por tanto debido también a su importancia en el mercado a continuación se ofrece un análisis detallado de sus características.

2.7.6 Introducción *Oracle Spatial*

Una vez que está claro el concepto de base de datos espacial y lo que conlleva, además de la visión global de otras bases de datos espaciales, este apartado ofrece información de la base de datos *Oracle Spatial*, debido a que hoy en día es una de las principales bases de datos del mercado con todas las ventajas que ello conlleva, como el amplio número de funcionalidades integradas para el tratamiento de los datos geográficos. Además, debido a que fue una de las primeras en desarrollar este tipo de base de datos, cuenta con más experiencia en este campo.

Oracle Spatial al ser una herramienta comercial, ha tratado de dar un mayor impulso y gran soporte a su herramienta de desarrollo, mejorando sus características de análisis, modelado y visualización para aplicaciones de servicios públicos y otras áreas.

2.7.6.1 Tipos de Datos Espaciales y Metadatos

La implementación del objeto-Relacional de *Oracle Spatial* está formada por un conjunto de tipos de datos de objeto, métodos de tipo y operadores, funciones y procedimientos que utilizan estos tipos.

En base a [17], para almacenar una geometría, ésta se almacena como un objeto en una sola fila, en una columna de tipo *SDO_GEOMETRY*. Por otro lado, la creación de datos espaciales y mantenimiento se realiza con básicas sentencias *DDL (Data Definition Language)* (*CREATE, ALTER, DROP*) y *DML (Data Manipulation Languaje)* (*INSERT, UPDATE, DELETE*).

A continuación se detallan, en base a [17], los principales tipos de datos espaciales y metadatos necesarios para desarrollar una base de datos espacial con *Oracle Spatial.*

Geometry Metadata (Metadato Geométrico)

El *Geometry Metadata* describe las dimensiones, los límites inferior y superior y la tolerancia para los datos espaciales. Cada *Geometry Metadata* se almacena en una tabla global de *MDSYS* y dependiendo del papel que tenga cada usuario, existen tres tipos: *USER_SDO_GEOM_METADATA, ALL_SDO_GEOM_METADATA* y *DBA_SDO_GEOM_METADATA*, donde sólo el primero puede ser actualizado por el usuario y es en el que se insertan los metadatos relacionados con las tablas espaciales.

Cada *Metadata* tiene la definición representada en la figura 17, donde el atributo *TABLE_NAME* representa el nombre de la tabla que contiene la columna de tipo espacial, el atributo *COLUMN_NAME* representa el nombre de la columna de tipo *SDO_GEOMETRY* de la tabla, el atributo *DIMINFO* representa una matriz de longitud variable que representa la dimensión, de manera que hay una entrada por cada dimensión, por ejemplo una entrada sería la longitud y otra la latitud. Dentro de esta dimensión se inserta la tolerancia, la figura 18 representa su sintaxis. Por último el campo *SRID* representa el valor del sistema de coordenadas que se quiere representar por ejemplo, en el caso del sistema de coordenadas de la Tierra es el *8307*. Si no se va a representar un sistema de coordenadas concreto su valor puede ser nulo.

```
(
  TABLE_NAME    VARCHAR2(32),
  COLUMN_NAME   VARCHAR2(32),
  DIMINFO       MDSYS.SDO_DIM_ARRAY,
  SRID          NUMBER
);
```

Figura 17. Definición de Geometry Metadata

La figura 18 representa la sintaxis de un objeto de tipo *MDSYS.SDO_DIM_ARRAY* donde se indica que es un array de tipo *SDO_DIM_ELEMENT*, y éste a su vez es un objeto compuesto por los siguientes atributos: *SDO_DIMNAME* que representa el nombre, *SDO_LB*, *SDO_UB* que representan los valores superiores e inferiores de la dimensión y *SDO_TOLERANCE* que representa el valor de la tolerancia.

```
Create Type SDO_DIM_ARRAY as VARRAY(4) of SDO_DIM_ELEMENT;

Create Type SDO_DIM_ELEMENT as OBJECT (
  SDO_DIMNAME VARCHAR2(64),
  SDO_LB NUMBER,
  SDO_UB NUMBER,
  SDO_TOLERANCE NUMBER);
```

Figura 18. Sintaxis DIMINFO

La figura 19 representa un ejemplo de inserción de un *Geometry Metadata* que pretende representar el sistema de coordenadas de la Tierra. En esta figura se observa una clausula de inserción *INSERT* en la tabla *USER_SDO_GEOM_METADATA* con los valores correspondientes a cada atributo: el nombre de la tabla (*cola_markets_cs*), el nombre de la columna espacial (*shape*), el valor del sistema de coordenadas (8307) y suss dimensiones que en este caso son dos: la longitud con valor de -180 a 180 y la tolerancia con valor 10 (tendrá 10 metros de precisión en la representación de los datos) y la latitud con valor entre -90 y 90 y la tolerancia con valor 10 (tendrá 10 metros de precisión en la representación de los datos igual que la longitud).

```
INSERT INTO USER_SDO_GEOM_METADATA
  VALUES (
  'cola_markets_cs',
  'shape',
  MDSYS.SDO_DIM_ARRAY(
    MDSYS.SDO_DIM_ELEMENT('Longitude', -180, 180, 10),  -- 10 meters tolerance
    MDSYS.SDO_DIM_ELEMENT('Latitude', -90, 90, 10)  -- 10 meters tolerance
     ),
  8307   -- SRID for 'Longitude / Latitude (WGS 84)' coordinate system
);
```

Figura 19. Ejemplo de inserción de USER_SDO_GEOM_METADATA

Objeto tipo SDO_GEOMETRY

Para poder almacenar un tipo de datos espacial es necesario definir el tipo de dato *SDO_GEOMETRY*. Este siempre tiene la estructura que se describe en la figura 20, y es cuando se insertan los datos en la tabla cuando se define el tipo de geometría que se quiere representar, ya sean puntos, líneas, polígonos o otras figuras más complejas.

```
CREATE TYPE sdo_geometry AS OBJECT (
 SDO_GTYPE NUMBER,
 SDO_SRID NUMBER,
 SDO_POINT SDO_POINT_TYPE,
 SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY,
 SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRAY);
```

Figura 20. Estructura tipo de dato SDO_GEOMETRY.

Para entender este tipo de dato es necesario conocer cada uno de sus atributos (figura 20):

- SDO_GTYPE: este atributo indica el tipo de geometría que se quiere representar, esta tiene que estar especificada en el *Geometry Object Model for the OGIS Simple Features for SQL*. El atributo está compuesto por cuatro dígitos de la forma *dltt*, donde *d* identifica el número de dimensiones (2,3 o 4), *l* se utiliza para referencias lineales (aunque su valor por defecto es el 0), y *tt* indica el tipo de geometría con valores de 00 a 07 donde, por ejemplo, el *dl*01 representa un punto, el *dl*02 una línea o curva, el *dl*03 un polígono, etc.

- SDO_SRID: este atributo indica el sistema de coordenadas que se va a representar. En el caso del sistema de coordenadas terrestre el valor sería el 8307, sin embargo, si no hay ningún sistema de coordenadas específico que se quiera representar este valor puede ser nulo.

- SDO_POINT: indica, para el tipo de geometría punto, las coordenadas donde se encuentra ese punto. Como se puede observar en la figura 21, se puede representar en un sistema de coordenadas tridimensional, pero si el caso exige representarlo en un sistema de coordenadas bidimensional el valor de la coordenada *z* puede ser nulo.

```
CREATE TYPE sdo_point_type AS OBJECT (
   X NUMBER,
   Y NUMBER,
   Z NUMBER);
```

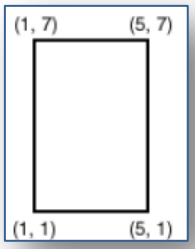
Figura 21. Definición SDO_POINT_TYPE

- SDO_ELEM_INFO: especifica objetos espaciales más complejos definiéndolos mediante una matriz de números de tipo *SDO_ELEM_INFO_ARRAY*. Este atributo especifica cómo interpretar los valores almacenados en el atributo *SDO_ORDINATES*, de manera que indica mediante la definición de un triplete <*SDO_STARTING_OFFSET*, *SDO_ETYPE*, *SDO_INTERPRETATION*> la posición donde comienza un nuevo elemento, como está conectado (líneas, rectas o arcos) y el tipo de objeto que se quiere representar (punto, línea, polígono).

- SDO_ORDINATES almacena los valores de las coordenadas de manera ordenada, donde se localizan los objetos espaciales definidos en *SDO_ELEM_INFO*, de ahí que se utilice siempre en combinación con el atributo *SDO_ELEM_INFO.*

Para entender mejor los conceptos expuestos anteriormente, a continuación se explica cómo se puede representar un rectángulo.

Ejemplo de representación de un Rectángulo




```
INSERT INTO cola_markets VALUES(
  1,
  'cola a',
  MDSYS.SDO_GEOMETRY(
    2003,  -- 2-dimensional polygon
    NULL,
    NULL,
    MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 = exterior)
    MDSYS.SDO_ORDINATE_ARRAY(1,1, 5,7) -- only 2 points needed to
          -- define rectangle (lower left and upper right) with
          -- Cartesian-coordinate data
  )
);
```

Figura 22. Representación rectángulo.

Como se observa en la figura 22, en el momento que se insertan los datos en la base de datos es cuando se define el tipo de geometría que se quiere representar.

Según indica la inserción de los datos del objeto *MDSYS.SDO_GEOMETRY*, el primer atributo especifica que tiene 2 dimensiones (*d = 2*, valor por defecto *l = 0* y es un polígono *tt = 03*, por tanto *dltt = 2003*), el segundo atributo indica que no representa un sistema de coordenadas específico (por eso es *null*), el tercer atributo indica que no es un punto (por este motivo no hace falta definir *SDO_POINT*), y por último el cuarto atributo define el triplete donde el *1003* indica que es un polígono simple, el *3* que es un rectángulo, y el último atributo indica la coordenada inferior izquierda y la coordenada superior derecha del rectángulo.

2.7.6.2 Funciones Geométricas

Una vez que se tienen definidos los tipos de datos espaciales que se desea representar, se puede hacer uso de las funciones geométricas, las cuales pueden manipular los datos espaciales.

Oracle Spatial agrupa sus funciones geométricas en cuatro grupos: relación, validación, funciones sobre un único objeto o sobre dos objetos. A continuación se especifican algunas de ellas:

- Funciones de relación:
  - *WITHIN_DISTANCE:* determina si dos geometrías están a una determinada distancia del otro.

- Funciones de Validación:
  - *VALIDATE_GEOMETRY:* determina si una geometría es válida.

- Funciones sobre un único objeto:
  - *SDO_LENGTH:* calcula la longitud o el perímetro de una geometría.

- Funciones sobre dos objetos:
  - *SDO_DISTANCE:* calcula la distancia entre dos objetos geométricos
  - *SDO_INTERSECTION:* devuelve un objeto geométrico resultado de la intersección topológica entre otros dos objetos geométricos.

Capítulo 3. Diseño

3.1 Presentación del problema

Se quiere diseñar una aplicación que contenga una interfaz gráfica que introduzca cambios en una red de manera manual y automática. Además, estos cambios deben ser visibles al usuario a través de la interfaz gráfica.

Para que esto sea posible, el usuario y la propia aplicación deberán poder dibujar en un panel de dibujo nodos y enlaces formando una red, al igual que también podrán borrarlos.

Toda la información asociada a la red se deberá almacenar en una base de datos espacial, ya que se quiere una aplicación que no limite el tamaño de la red y que el acceso a la información se produzca de una manera eficiente, siendo las bases de datos, con su sistema gestor, el que mejor se adapta a estos requisitos. Además, la información que se desea almacenar se refiere a la información de un nodo, como su *id* identificativo y el punto que contiene las coordenadas espaciales del panel donde se encuentra dibujado, es decir se trata de un sistema espacial no georreferenciado al utilizar las coordenadas del panel de dibujo de la interfaz de usuario que se quiere desarrollar.

Por otro lado, la información almacenada referida a los enlaces será su *id* identificativo correspondiente, identificación de nodo de inicio y nodo de fin, para saber cómo estarán conectados entre ellos, y por último el coste correspondiente del enlace.

Una vez explicada la manera de incluir cambios en la red de manera manual (cambios introducidos a través de la interfaz gráfica), queda por analizar el problema de la generación de cambios automáticos en la red, de manera que los cambios que se produzcan en la base de datos se pinten en el panel de dibujo de la interfaz gráfica automáticamente.

Para que el usuario interactúe con la red, es decir la pueda pintar, borrar o modificar, la aplicación deberá tener las siguientes funcionalidades:

- ***Pintar un Nodo***: esto lo permitirá hacer de dos maneras distintas: introduciendo las coordenadas *x* e *y* correspondientes al punto donde se quiere pintar el nodo, o haciendo clic en el panel de dibujo. Cuando se haga esto, inmediatamente deberá aparecer un nodo dibujado. Este nodo puede ser un dibujo con forma circular o con una imagen que represente un nodo. Hay que tener en cuenta que la información asociada al nodo, a sus coordenadas deberá ser almacenada en la base de datos, inmediatamente después de ser creado, previa comprobación de que este nodo no existe.

- ***Pintar un enlace***: la interfaz gráfica deberá tener tres modalidades de dibujo de un enlace. La primera será mediante la introducción de las coordenadas *x* e *y* de inicio y las coordenadas *x* e *y* de fin. Otra modalidad será mediante la introducción de *id* de nodo de inicio e *id* de nodo de fin. La tercera modalidad será mediante el dibujo con el ratón de una línea que una dos nodos en el panel de dibujo.

  Toda la información del enlace deberá ser almacenada inmediatamente en la base de datos previa comprobación de que este enlace no existe ya. Además se deberá tener en cuenta que la conexión entre los distintos nodos será bidireccional, es decir, el enlace de *A* a *B* es igual que el enlace de *B* a *A*.

- ***Borrar un Nodo***: para realizar esta acción deberá haber dos posibilidades. Una mediante la introducción de las coordenadas del nodo a borrar y la otra mediante la introducción del *id* del nodo a borrar. Además, hay que tener en cuenta que si un nodo está asociado a un enlace, éste deberá ser borrado inmediatamente, ya que es inconsistente que haya un enlace sin un nodo asociado.

- ***Borrar un enlace***: al igual que la función borrar nodo, ésta tendrá dos posibilidades: mediante la introducción del *id* del enlace o mediante la introducción de las coordenadas de inicio y de fin del enlace a borrar. Se debe tener en cuenta que al borrar un enlace deberá existir la posibilidad de poder borrar los nodos que tenga asociados, no sin antes solicitar la confirmación del usuario. Esto conlleva que si se decide borrar algún nodo, si éste tiene más enlaces asociados, estos deberán borrarse automáticamente.

  Como se puede observar, la información deberá ser siempre consistente, pues un enlace no puede tener en su información un *id* de nodo que no existe o un *id* de nodo al que no está asociado.

- ***Pintar red***: se deberá pintar la red entera que esté guardada en la base de datos, es decir, se pintará toda la información que hubiera en ese momento respecto a enlaces y nodos asociados

- ***Borrar red***: esta función deberá borrar por completo la red que esté pintada en ese momento en el panel de dibujo, y por tanto en la base de datos.

- ***Cambiar costes***: para cambiar costes se deberá introducir el *id* del enlace sobre el que se va a ejecutar dicha función y el coste nuevo que tendrá ese enlace. Por consiguiente, esa información deberá ser actualizada en la información del enlace guardada en la base de datos.

Para el pintado automático de la red, la aplicación deberá ser capaz de, mediante la activación por parte de un botón, empezar a pintar sin ayuda del usuario una nueva red gracias a los cambios que se producirán en la base de datos. Esto debe ser transparente al usuario que sólo verá como en el panel de dibujo de la interfaz la red va cambiando cada determinado tiempo.

3.2 Arquitectura del sistema

La arquitectura para este sistema es sencilla. En ella, sólo intervienen los siguientes cuatro elementos: el pc necesario para la interacción del usuario con la aplicación y la base de datos así como para el diseño tanto de la interfaz gráfica como de la base de datos; la interfaz de usuario de la aplicación necesaria para interactuar con la base de datos y enseñar al usuario la red pintada y sus cambios; la base de datos, herramienta indispensable para almacenar de forma consistente los datos referidos a la red y que se pueda trabajar con ellos; y por último, para la modalidad de pintado no automático de la red es necesario un usuario que interactúe con la aplicación.

Como se especifica en la figura 22, que representa la arquitectura del sistema que se quiere diseñar para este proyecto, la base de datos y la interfaz de usuario están conectadas entre ellas, gracias a la biblioteca *Java Database Connectivity (JDBC)* descrita en el apartado 2.1. Esta conexión es necesaria debido a que la interfaz de usuario debe reflejar los cambios producidos en los datos que están almacenados en la base de datos y que, a su vez, la base de datos debe reflejar los cambios producidos por el usuario al pintar en la interfaz de usuario.

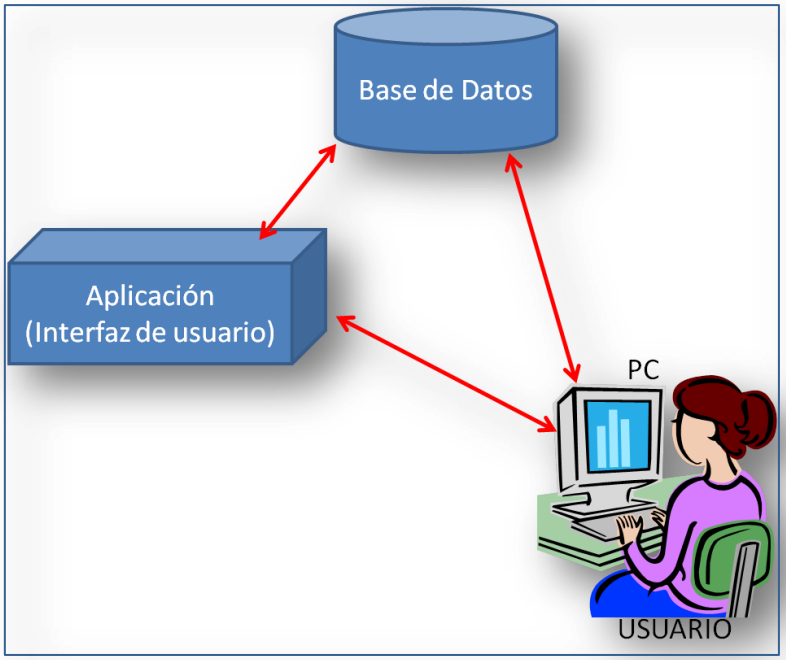


Figura 22. Arquitectura del sistema.

3.3 Diseño de la Base de datos

Este apartado presentará el diseño de la base de datos mediante el diseño inicial de un esquema Entidad-Relación (diseño lógico) y su transformación en un esquema Relacional (diseño físico-lógico).

En el apartado del esquema Entidad-Relación se especificarán las entidades, así como las relaciones entre ellas, al igual que los atributos de las entidades. Por último se enumeraran los supuestos semánticos no recogidos.

Por otro lado, en el esquema Relacional se identificarán las nuevas relaciones creadas a partir del esquema Entidad-Relación, así como las restricciones de integridad de clave primaria y clave ajena y los supuestos semánticos no recogidos.

3.3.1 Esquema Entidad-Relación

En el esquema Entidad-Relación se representarán los aspectos lógicos de los diferentes tipos de datos de la realidad que está siendo analizada, reflejando el contenido semántico de los datos existentes en la aplicación.

En este caso, el esquema estará formado por dos entidades de las cuales se quiere guardar información relativa a sus correspondientes características descriptivas denominadas atributos. Por otro lado, también se reflejarán las interrelaciones que existen entre las entidades. Todo esto se aprecia en la figura 23.

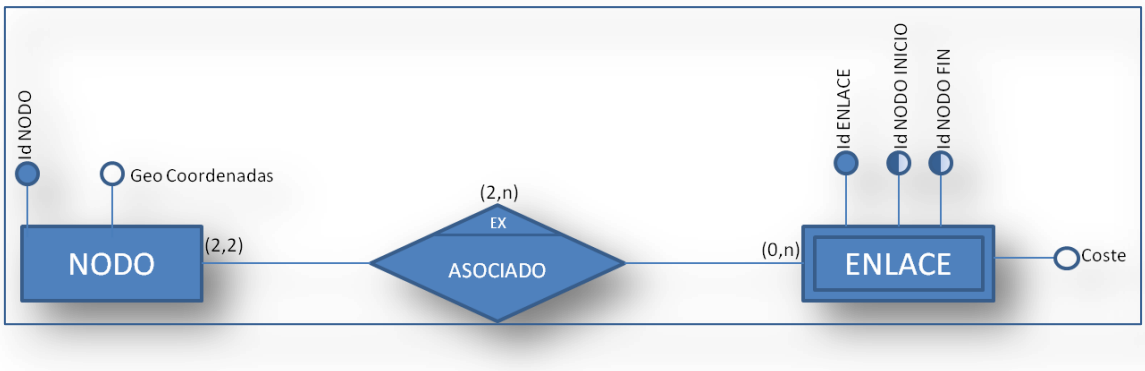


Figura 23. Esquema Entidad-Relación.

Con respecto a las entidades y atributos que aparecen en la figura 23:

- La entidad *NODO* se refiere a la información de cada nodo dibujado en el panel de dibujo de la aplicación y guardado en la base de datos. Sus atributos son: *Id NODO*, clave primaria de la entidad y que identifica a cada nodo de la red; y *Geo Coordenadas*, atributo con información espacial que contendrá las coordenadas en las cuales se encuentra el nodo.

- La entidad *ENLACE* se refiere a la información de cada enlace dibujado en el panel de dibujo de la aplicación y guardado en la base de datos. Es una entidad débil cuya existencia depende de la existencia de algún nodo, y cuyos atributos son: la clave primaria, *Id ENLACE*, que representa al identificador del enlace; el nodo inicio del enlace *Id NODO INICIO;* el nodo fin del enlace *Id NODO FIN;* y el coste, referido al coste que tiene cada enlace. La unión de los atributos *Id NODO INICIO* e *Id NODO FIN* sería una clave alternativa de la entidad.

Una vez detalladas cada entidad y sus atributos correspondientes, es importante especificar que un nodo está asociado a cero o varios enlaces y un enlace tiene una relación de dependencia de existencia con la entidad NODO, asociado a un mínimo de 2 nodos y a un máximo de 2 nodos, es decir, un enlace está asociado a un nodo de inicio y a otro de fin, y en caso contrario no podrá existir. Por tanto la relación existente entre nodo y enlace, será (2,N).

Supuestos semánticos no reflejados

Los supuestos semánticos no reflejados se refieren a las restricciones que el esquema Entidad-Relación no refleja, y por consiguiente queda pendiente a reflejar en el esquema Relacional. A continuación se detallan los supuestos no reflejados:

- *Id NODO INICIO* debe ser distinto a *Id NODO FIN*.
- La red es bidireccional, por lo que se considerará que un enlace será igual a otro si los nodos extremos de ambos son los mismos aunque estén en sentido contrario.
- No pueden existir dos nodos con distinto id de *Nodo* pero iguales coordenadas.
- *Id NODO INICIO* debe existir en la tabla de *NODO*.
- *Id NODO FIN* debe existir en la tabla de *NODO*.

3.3.2 Esquema Relacional

Para reflejar los supuestos semánticos no recogidos en el esquema Entidad-Relación, en este apartado se hace una transformación del mismo a un esquema Relacional en el que cada entidad se transforma en una relación, las relaciones en interrelaciones, se hacen las elecciones de clave primaria y clave alternativa, y se reflejan los tipos de modificaciones y borrados impuestos. En la figura 24 se puede observar el esquema Relacional del sistema.

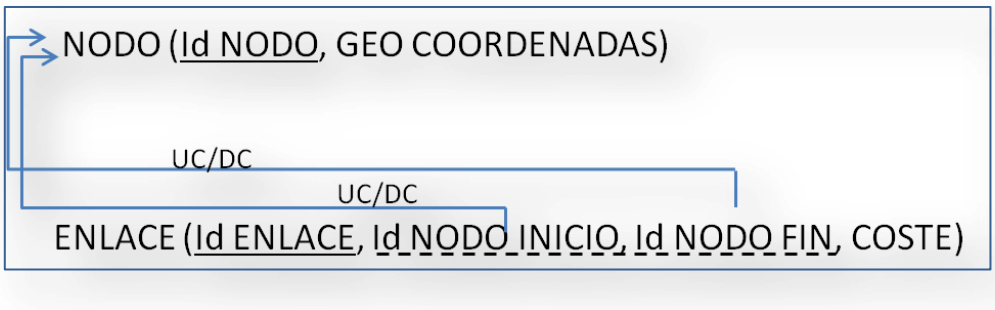


Figura 24. Esquema Relacional.

Según la figura 24, dentro de las relaciones del esquema Relacional se identifica la relación NODO, que contiene los datos de un nodo y que tiene por atributos el identificador del nodo (*Id NODO*) que es identificativo único y por tanto clave primaria, y las coordenadas del mismo en el panel de dibujo de la aplicación (atributo *GEO COORDENADAS*). Cabe destacar que el atributo *GEO COORDENADAS* es de tipo espacial para indicar que un Nodo tiene información espacial referida al punto donde se localiza exactamente en el panel de dibujo, compuesto por las coordenadas *x* e *y*.

Por otro lado, según indica el esquema Relacional, la relación ENLACE almacena toda la información referente al mismo, con su *Id ENLACE* identificativo único, por tanto clave primaria. Con el fin de reflejar el supuesto semántico no recogido en el esquema Entidad-Relación de asegurar que tanto el *Id NODO INICIO* e *Id NODO FIN* deben existir en la relación NODO, *Id NODO INICIO* e *Id NODO FIN*, se definen como claves ajenas, provenientes de la relación NODO debido a la interrelación que hay entre las entidades de (2,N), cuyos borrados y modificaciones se definen en cascada para que en el momento que se elimine o modifique la información de un nodo, inmediatamente se modifique en la relación ENLACE. Además, hay que tener en cuenta que los dos atributos (Id *NODO* INICIO, Id *NODO* FIN) forman una clave alternativa para imponer el supuesto semántico no reflejado en el esquema Entidad-Relación de que no pueden existir dos enlaces con distintos id de enlaces e iguales *Id NODO INICIO* e *Id NODO FIN*. Por último, el atributo *COSTE* especifica el valor del coste asociado a ese enlace.

Para reflejar los demás supuestos semánticos no impuestos en el esquema Entidad-Relación y que tampoco refleja el esquema Relacional, se definen los siguientes casos:

- Es necesario definir un *check* que comprueba que *Id NODO INICIO* debe ser distinto a *Id NODO FIN*.

- Para controlar que no puede existir un enlace igual a otro si los nodos extremos de ambos son los mismos aunque estén en sentido contrario es necesario definir un disparador.

- Para controlar que no puede existir dos nodos con distinto id de nodo pero iguales coordenadas es necesario definir un disparador.

- Controlar la cardinalidad mínima de la relación de un Enlace con, como mínimo dos nodos.

3.4 Diseño inicial de la aplicación

Con el fin de incluir en la aplicación todos los requisitos explicados en el apartado de análisis, se ha decidido diseñar una interfaz como la mostrada en la figura 25. Como se puede comprobar, es un diseño muy sencillo que consta de un panel de dibujo (el recuadro de color blanco en la parte izquierda de la figura 25), y los distintos apartados que proporcionarán las funcionalidades requeridas por la aplicación (todas en la parte derecha de la figura 25): Red, con sus dos botones, *Pintar red y Borrar red;* Nodo, donde se encuentran los botones *Pintar Nodo, Borrar Nodo y* los cuadros de texto donde se introducirán los datos para pintar o borrar el Nodo; Enlace con sus dos botones *Pintar enlace, Borrar enlace* y los cuadros de texto donde se introducirá la información necesaria para pintar o borrar un enlace; y Costes que contiene un botón, *Cambiar costes*, y los cuadros de texto donde se introducirá la información necesaria para cambiar el coste de un enlace.

Conforme se vaya desarrollando la aplicación, el diseño irá cambiando, ya que seguramente se irán introduciendo nuevas funcionalidades o cambiará la ordenación de los componentes.

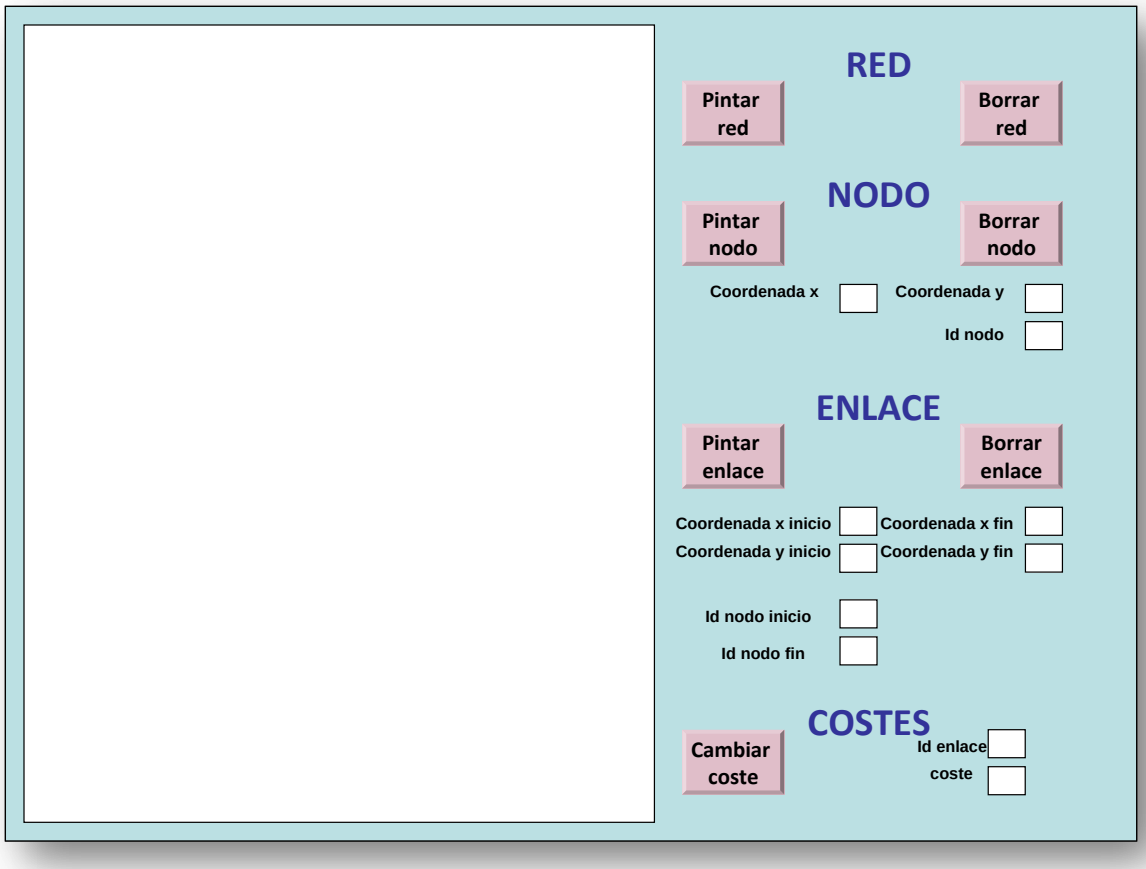


Figura 25. Diseño inicial de la aplicación.

Capítulo 4. Implementación

4.1 Introducción

Una vez que el diseño de la aplicación está finalizado, este apartado se centra en la implementación de la aplicación dividiéndola en dos partes: la implementación de la interfaz gráfica y la implementación de la base de datos. Además, se mostrará la manera en que trabajan juntas.

Hasta ahora se ha abordado el funcionamiento de la aplicación sin entrar en los detalles de cómo se hace posible cada funcionalidad como, por ejemplo, que en el momento en el que se dé clic en el botón cerrar ventana, ésta se cierre; que cuando se dibuja una línea con el ratón en el panel de dibujo se almacene automáticamente en la base de datos la información sobre su posición o asociación con los nodos; o que si se producen cambios sobre la información de la base de datos, éstos se reflejen en el panel de dibujo.

Para que se entienda bien la implementación de la aplicación, se va a dividir su explicación en dos grandes bloques:

- Implementación de la Base de Datos: creación de las tablas, sus funciones, procedimientos, inserciones, disparadores, etc.

- Implementación de la interfaz gráfica: para la implementación de la interfaz gráfica se parte del análisis de un diagrama de clases de la aplicación para tener una visión global del funcionamiento de la aplicación, posteriormente se detalla la implementación del diseño visual para entender cada componente usado en la interfaz y su funcionalidad correspondiente, y, por último, en la implementación de las funcionalidades de la aplicación se detalla qué métodos de las clases correspondientes son necesarios y como se usan para llevar a cabo cada funcionalidad de la aplicación.

Es importante mencionar que, en un principio, el funcionamiento de la aplicación fue implementado para que la interfaz trabajara sin el almacenamiento y la gestión de una base de datos, es decir, toda la información referida a la red, nodos y enlaces era almacenada en memoria principal en una estructura llamada *vector*, pero gracias a la incorporación de una base de datos como medio de almacenamiento de la información de la red, esta estructura desaparece y con ella procedimientos de manejo y almacenamiento de datos que eran necesarios para el buen funcionamiento de la aplicación sin una base de datos.

Cabe destacar que al incorporar la base de datos, la implementación de muchos de los procedimientos de manejo de la información son más sencillos, pues *Oracle* proporciona funciones ya implementadas para ello, por ejemplo, un *borrado en cascada*, sólo es necesario indicarlo en el *script* de creación de la tabla, por el contrario, en la versión en la que no se incluía base de datos, era necesario crear procedimientos que buscaran un objeto a borrar en el *vector* y posteriormente actualizara los demás vectores.

4.2 Implementación de la Base de datos

En el siguiente apartado se explican las decisiones tomadas respecto a la implementación del esquema relacional en el *SGBD Oracle* especificado en el apartado de diseño.

Creación de tablas

Como se pudo observar en al apartado de Diseño, la base de datos está compuesta por dos relaciones que se ven transformadas a tablas en este apartado. De modo que se tendrá la tabla *NODO* y la tabla *ENLACE*.

En la figura 26 se observa la tabla *NODO*, formada por dos tipos de atributos: *id_NODO* y *GEO_COORDENADAS*. El *id_NODO* es un atributo de tipo numérico cuyo valor puede tener hasta 10 cifras, para no limitar el tamaño de la red y que ésta pueda crecer dinámicamente sin miedo a que no se puedan crear más nodos, por causa de que el valor de este atributo no se puede repetir debido a que es declarado como clave principal con la cláusula *CONSTRAINT PK_id_NODO PRIMARY KEY (id_NODO)* como se observa en la figura 26.

El otro atributo declarado, *GEO_COORDENADAS*, es de tipo *SDO_GEOMETRY*. Este es un atributo espacial que almacenará las coordenadas *x* e *y* del Nodo.

```
CREATE TABLE NODO (
  id_NODO NUMBER(10) NOT NULL,
  GEO_COORDENADAS MDSYS.SDO_GEOMETRY NOT NULL,
  CONSTRAINT PK_id_NODO PRIMARY KEY (id_NODO)
);
```

Figura 26. Creación de tabla Nodo.

Por otro lado, la tabla *ENLACE*, que se observa en la figura 27, identifica cuatro atributos, todos de tipo numérico ya que almacenan información de este tipo. El primero de estos atributos es *id_ENLACE*, cuyo valor puede almacenar hasta 10 cifras para que, al igual que en el caso del atributo *id_NODO* de la tabla *NODO*, no se limite el tamaño de la red. Además, cabe destacar que el atributo *id_ENLACE* es declarado como clave principal con la cláusula *CONSTRAINT PK_id_ENLACE PRIMARY KEY (id_ENLACE)* para evitar que este atributo identificativo pueda repetirse.

Por otro lado, es importante identificar que los atributos *id_NODO_INICIO* e *id_NODO_FIN* son claves ajenas provenientes de la tabla *NODO* declaradas con la cláusula *CONSTRAINT nombre FOREIGN KEY (nombre de atributo) REFERENCES tabla ON DELETE CASCADE*, por este motivo los atributos *id_NODO_INICIO* e *id_NODO_FIN* deben tener el mismo tamaño y tipo que el atributo al que referencian (*id_NODO*). Además de ser claves ajenas, los dos atributos son declarados como clave alternativa para lograr que no se pueda almacenar un enlace con esta pareja de atributos repetida, la forma de declarar es mediante la cláusula *CONSTRAINT nombre UNIQUE (nombre atributo1, nombre atributo2)* como se observa en la figura 27.

El atributo *COSTE*, al almacenar un coste numérico, es declarado como numérico cuyo valor puede llegar a tener hasta 10 cifras para no limitar que en algún momento pueda tener un coste alto.

Por último, como se identifica en la figura 27, es necesario declarar un *check* con la clausula *CONSTRAINT nombre CHECK (nombre atributo1 != nombre atributo2)* para controlar que los atributos *id_NODO_INICIO* e *id_NODO_FIN* a la hora de insertar esta tupla nunca van a ser iguales.

```
CREATE TABLE ENLACE (
  id_ENLACE NUMBER(10) NOT NULL,
  id_NODO_INICIO NUMBER(10) NOT NULL,
  id_NODO_FIN NUMBER(10) NOT NULL,
  COSTE NUMBER(10) NOT NULL,
  CONSTRAINT UK_id_NODO_INICIO UNIQUE (id_NODO_INICIO,id_NODO_FIN),
  CONSTRAINT PK_id_ENLACE PRIMARY KEY (id_ENLACE),
  CONSTRAINT FK_id_NODO1 FOREIGN KEY (id_NODO_INICIO) REFERENCES NODO ON DELETE CASCADE,
  CONSTRAINT FK_id_NODO2 FOREIGN KEY (id_NODO_FIN) REFERENCES NODO ON DELETE CASCADE,
  CONSTRAINT CH_NODOS CHECK (id_NODO_INICIO != id_NODO_FIN)
);
```

Figura 27. Creación de tabla Enlace.

Creación del Índice

Para poder acceder a los datos con mayor rapidez y eficacia, evitando que el usuario no se sienta satisfecho con el funcionamiento de la aplicación debido a los retardos para mostrar y acceder a la información, el *SGBD* de *Oracle* ofrece la posibilidad de crear un índice espacial para trabajar con los datos espaciales. Éste deberá ser creado por cada columna de tipo espacial, por lo que en el caso de este trabajo solo será necesario crear uno, ya que sólo existe un atributo de este tipo, el atributo *GEO_COORDENADAS* de la tabla *NODO*. En la figura 28 se observa la forma de crear el índice.

```
CREATE INDEX coordenadas_idx ON NODO (GEO_COORDENADAS) INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Figura 28. Creación del Índice espacial.

Inserción de registros

Después de la creación de las tablas, en este apartado se especificará el modo de inserción de los datos en las mismas, para ello primero se debe especificar la definición del *SRE*, es decir, la inserción en la tabla *USER_SDO_GEOM_METADATA* de los campos espaciales que controlarán el espacio en el que se van a situar los nodos y los enlaces.

Además, es importante mencionar que antes de la inserción de los datos en las tablas es necesario controlar, mediante los disparadores, que se cumplen las condiciones impuestas en la información para ser almacenada.

Por último, es necesario destacar que la inserción de los datos en las tablas se hace a través la interfaz gráfica, por lo que los datos son pasados por parámetros a la sentencia *SQL* que se encarga de la inserción de los datos en las tablas.

Con respecto a la inserción en la tabla *USER_SDO_GEOM_METADATA*, puesto que solo hay un dato espacial a encuadrar dentro de un SRE, la única inserción a realizar será la mostrada en la figura 29. En ella se indica que los datos espaciales hacen referencia a la tabla *NODO* en el atributo *GEO_COORDENADAS* y que en este solo habrá dos coordenadas *x* e *y*, donde *x* sólo puede tomar valores entre 5 y 735 e *y* sólo puede tomar valores entre 15 y 815, pues estos valores son los correspondientes al margen del panel de dibujo de la interfaz gráfica. Por último, el valor de la tolerancia, es decir, el valor de precisión, es de 0.005, que indica la distancia más próxima a la que pueden estar dos nodos. Es importante mencionar que la creación de esta tabla viene impuesta por el *SGBD* de *Oracle* para trabajar con datos espaciales.

```
INSERT INTO USER_SDO_GEOM_METADATA VALUES
('NODO', 'GEO_COORDENADAS',
MDSYS.SDO_DIM_ARRAY(
MDSYS.SDO_DIM_ELEMENT('X', 5, 735, 0.005),
MDSYS.SDO_DIM_ELEMENT('Y', 15, 815, 0.005)),
NULL);
```

Figura 29. Inserción USER_SDO_GEOM_METADATA

Por otro lado, para controlar que los datos son correctos antes de que éstos sean insertados en las tablas, es necesario la creación de los disparadores, en este caso dos: *EXCLUSIVIDAD NODOS* y *EXISTE_ENLACE_POR_NODOS*.

El disparador *EXCLUSIVIDAD_NODOS* es necesario para controlar que no se va a insertar un nodo con distinto identificador pero con iguales coordenadas a un nodo ya existente.

Para esto, el disparador se encarga de seleccionar el valor de la coordenada *x* de aquellos nodos que cumplan la condición de que su coordenada *y* sea igual a la coordenada *y* del nuevo nodo que va a ser insertado, para después comparar si la

coordenada *x* elegida anteriormente es igual también a la coordenada *x* del nuevo nodo que va a ser insertado. Si es así, quiere decir que se trata de insertar un nodo con iguales coordenadas a las de un nodo que ya existe y que por tanto se debe evitar dicha inserción.

Todo este proceso se puede observar en la figura 30. En ésta se observa la implementación del disparador *EXCLUSIVIDAD_NODOS* donde se hace uso de la función *RAISE_APPLICATION_ERROR()* para informar del error de que ya existe un nodo con esas coordenadas. Por otro lado, en caso que el disparador no encuentre información alguna, se controlará la excepción con la cláusula *WHEN NO_DATA_ FOUND* para evitar que se interrumpa la ejecución de la aplicación.

```
create or replace
TRIGGER EXCLUSIVIDAD_NODOS
BEFORE INSERT ON NODO
FOR EACH ROW
DECLARE
    x number(6);
BEGIN
  SELECT N.geo_coordenadas.SDO_POINT.X INTO x FROM NODO N
  WHERE N.geo_coordenadas.SDO_POINT.Y = :new.geo_coordenadas.SDO_POINT.Y;
    IF (x = :new.geo_coordenadas.SDO_POINT.X) THEN
      RAISE_APPLICATION_ERROR(-20001, 'Error de exclusividad:Ya existe un nodo con esas coordenadas');
    END IF;

  EXCEPTION
    WHEN NO_DATA_FOUND THEN x := null;
END;
```

Figura 30. Disparador EXCLUSIVIDAD NODOS.

En cuanto al disparador *EXISTE_ENLACE_POR_NODOS*, controla que no se inserte un enlace que tenga distinto identificador de enlace pero iguales nodos extremos que un enlace existente. Para esto, el disparador selecciona un enlace de la tabla que cumpla la condición que sus nodos extremos sean iguales a los nodos extremos del nuevo enlace que va a ser insertado, independientemente del orden en que estén los nodos insertados en la tupla. Si encuentra alguno, el nuevo enlace no se inserta porque ya existe.

En la figura 31 se observa la implementación de este disparador, donde se hace uso de la función *RAISE_APPLICATION_ERROR()* para informar del error de que ya existe un enlace igual. Además, en caso que el disparador no encuentre información alguna, se controlará la excepción con la cláusula *WHEN NO_DATA_ FOUND* para evitar que se interrumpa la ejecución de la aplicación.

```
create or replace
TRIGGER EXISTE_ENLACE_POR_NODOS
BEFORE INSERT ON ENLACE
FOR EACH ROW
DECLARE
    IdNodo ENLACE.id_ENLACE%TYPE;

BEGIN
  IdNodo := 0;

  SELECT e.id_enlace into idNodo FROM ENLACE E
  where (e.id_nodo_inicio = :new.id_NODO_INICIO and e.id_nodo_fin = :new.id_NODO_FIN)
  or (e.id_nodo_inicio = :new.id_NODO_FIN and e.id_nodo_fin = :new.id_NODO_INICIO);

  RAISE_APPLICATION_ERROR(-20001, 'Error de exclusividad: Ya existe ese enlace');

  EXCEPTION WHEN no_data_found then IdNodo := null;
END;
```

Figura 31. Disparador EXISTE_ENLACE_POR_NODOS.

Una vez que están definidos los disparadores para controlar las inserciones, se especifican las cláusulas *INSERT* que hacen posible la inserción de los datos en las tablas. Debido a que esto se hace a través de la interfaz gráfica, los datos irán "encapsulados" en variables definidas en el código de la aplicación, como se observa en las figuras 32 y 33 que especifican, por ejemplo, las variables *x* e *y* que almacenan las coordenadas *x* e *y*, las cuales fueron introducidas a través de la interfaz gráfica.

Puesto que la tabla *NODO* almacena el identificador del nodo y las coordenadas espaciales del mismo, para la inserción de estos datos es necesario utilizar la cláusula definida en la figura 32, donde se observa que se inserta el identificador del nodo y la información del atributo *GEO_COORDENADAS* de tipo *SDO_GEOMETRY* que contiene los siguientes campos asociados:

- SDO_GTYPE = 2001. Este valor indica que se va a almacenar un punto.
- SDO_SRID = NULL. Indica que se va a usar un sistema de referencia cartesiano.
- SDO_POINT = SDO_POINT_TYPE(X, Y, NULL). Indica un sistema bidimensional donde las coordenadas del punto son *x* e *y*.
- SDO_ELEM_INFO y SDO_ORDINATES = ambos con valores NULL, ya que para un punto no se precisa dar valor a estos atributos, pues estos sólo es necesario definirlos cuando se quiere representar polígonos o líneas.

```
"INSERT INTO NODO VALUES ("+id_nodo+","
 + "MDSYS.SDO_GEOMETRY(2001,NULL,MDSYS.SDO_POINT_TYPE("+x+","+y+",NULL),NULL,NULL))");
```

Figura 32. Inserción en la tabla Nodo.

Por último, la inserción en la tabla *ENLACE* viene definida por la clausula *INSERT* especificada en la figura 33, donde se observa cómo se insertan los valores correspondientes al identificador del enlace, el identificador del nodo de inicio, el identificador del nodo de fin y el coste. Estos valores han sido introducidos previamente por el usuario a través de la interfaz gráfica.

```
"INSERT INTO ENLACE VALUES ("+id_enlace+","+id_nodo_inicio+","+id_nodo_fin+","+coste+")"
```

Figura 33. Inserción en la tabla Enlace.

Borrado de registros

Debido a que el borrado de un nodo implementa un borrado en cascada, en el momento que se borra un nodo se borrarán todos sus enlaces asociados. El borrado de registros viene determinado gracias a la clausula *DELETE*, y al igual que en el caso de la inserción de los datos de las tablas, la elección del borrado de un nodo, de un enlace, o de la red entera viene ordenada a través de la interfaz gráfica mediante el paso de los parámetros que indican, por ejemplo, el valor del id de nodo o del enlace que se desea borrar. Esto se observa en el ejemplo de borrado de un enlace en la figura 34 donde se describe la cláusula *DELETE*.

```
"DELETE FROM ENLACE WHERE id_ENLACE = "+ID
```

Figura 34. Borrado de enlace.

Consultas relevantes

Se puede decir que el pilar básico del funcionamiento de esta aplicación son las consultas que permiten obtener los datos que hacen posible la ejecución de todas las funcionalidades, ya que al trabajar con unos datos que están almacenados en un base de datos es necesario la implementación de diversas consultas para obtener los datos necesarios para pintar o borrar la red en el panel de dibujo de la interfaz gráfica.

Según el objetivo de las consultas implementadas éstas se pueden separar en dos grupos:

- Las consultas que extraen toda la información de la tabla nodo o de la tabla enlace, guardando cada campo extraído en una variable para después pasar el valor de esa variable al procedimiento correspondiente. En este caso cada vez que se necesita consultar toda la información de una tabla es para pintar la red entera. En el ejemplo de la figura 35 se observa este tipo de consulta

Figura 35. Consulta de datos de la tabla Nodo.

En la consulta de la figura 35 se emplea un alias *N* de la tabla *NODO* y para obtener las coordenadas se accede al atributo *GEO_COORDENADAS*, y más concretamente al valor de las coordenadas *x* e *y*.

- Por otro lado están las consultas de un dato partiendo de una condición impuesta que debe tener esa información que se quiere extraer, es decir, una consulta que partiendo del identificador de un nodo extraiga las coordenadas de éste o otra consulta que partiendo del identificador de un enlace extraiga el nodo de inicio y de fin del mismo.

  El ejemplo de la consulta que se observa en la figura 36 especifica que partiendo del identificador del enlace se puede extraer que nodo de inicio y de fin tiene ese enlace.

Figura 36. Consulta de Nodo de inicio y de fin de un enlace.

Procedimientos y *Job*

La aplicación tiene una funcionalidad que consiste en un pintado automático. Para realizar esta acción se hará uso de la herramienta *Job* proporcionada por *Oracle*. Ésta permitirá que cada determinado intervalo de tiempo se lance la ejecución de un procedimiento que se encargará de introducir automáticamente datos de nuevos nodos o enlaces en la base de datos para que sean pintados, o introducir datos de nodos o enlaces que ya existen para que sean borrados sin necesidad de la interacción del usuario con la aplicación.

El nombre de este *Job* será *jobPintarRed*, y se encargará de lanzar el procedimiento *Pintar_Red* con una frecuencia de 5 segundos. Además, en el momento de iniciar la aplicación el *Job* estará desactivado para que sea el usuario el que decida si activarlo o no. En la figura 37 se observa la forma de crear un *Job* introduciendo el nombre del *Job*, el tipo, la acción que debe ejecutar (en este caso debe ejecutar el procedimiento *Pintar_Red*), la fecha en la que debe dar comienzo su ejecución (en este caso se impone que sea la fecha del sistema), la frecuencia con la que debe ejecutar el procedimiento, la fecha de finalización de ejecución del *Job* (en este caso es nula) y el estado de activado o desactivado (en un principio el Job esta desactivado).

```
BEGIN
  DBMS_SCHEDULER.create_job(
    job_name => 'jobPintarRed',
    job_type => 'PLSQL_BLOCK',
    job_action => 'BEGIN
        Pintar_Red;
            END;',
    start_date => SYSTIMESTAMP,
    repeat_interval => 'freq=SECONDLY; INTERVAL = 5',
    end_date => NULL,
    enabled => FALSE);
END;
```

Figura 37. Creación de un Job.

Para la activación o desactivación del Job se implementaron los procedimientos *activarJOB y desactivarJOB*, respectivamente, que se ejecutan por decisión del usuario y cuya funcionalidad es, como su propio nombre indica, activar o desactivar *jobPintarRed*. En la figura 38 se observa el procedimiento que activa el *Job* en el que se emplea la función *sys.dbms_scheduler.enable* para realizar tal acción.

```
create or replace PROCEDURE activarJOB IS
BEGIN
     sys.dbms_scheduler.enable(name=>'"KATH"."JOBPINTARRED"');
END;
```

Figura 38. Procedimiento que activa el Job.

En la figura 39 se observa el procedimiento que desactiva el *Job jobPintarRed* mediante la función *sys.dbms_scheduler.disable* proporcionada por *Oracle.*

```
create or replace PROCEDURE desactivarJOB IS
BEGIN
      sys.dbms_scheduler.disable(name=>'"KATH"."JOBPINTARRED"', force => TRUE);
END;
```

Figura 39. Procedimiento que desactiva el Job.

El procedimiento *Pintar_red* se encarga de producir cambios en la red que está almacenada en la base de datos, cambios del tipo borrar un nodo o un enlace existente ya en la base de datos o crear un nuevo nodo o enlace para almacenarlo en la base de datos y por consiguiente pintarlo en el panel de dibujo de la interfaz gráfica. Para ello, el procedimiento parte de la función *dbms_random* que calcula un número y dependiendo de ese número se produce un cambio u otro. Por ejemplo, si el número que devuelve la

función *dbms_random* es un 3 se borra cualquier nodo elegido al azar de la tabla *NODO* mediante la función *dbms_random*. Esto se puede ver en el ejemplo de la figura 40, donde se observa que se selecciona un nodo al azar con la función *dbms_random* y se elige sólo un resultado.

```
SELECT id_NODO INTO id_nodo_inicio
  FROM ( SELECT id_NODO FROM NODO ORDER BY dbms_random.value ) WHERE rownum = 1;
```

Figura 40. Selección de un Nodo mediante la función dbms_random.

En cambio, si el resultado devuelto es un 2, se pintará un enlace eligiendo antes al azar cualquier nodo de inicio y de fin de la tabla nodo que serán conectados entre sí gracias al nuevo enlace. Por otro lado se calcula su identificador, también con la función *dbms_random*, y el coste es cualquier valor elegido al azar por no tener ningún tipo de relevancia en la actualidad. En la figura 41 se observa cómo el procedimiento *Pintar_Red* crea un enlace para que sea pintado en el panel de dibujo de la interfaz gráfica, primero eligiendo dos nodos al azar y posteriormente insertando los datos del nuevo enlace en la tabla *ENLACE* (en este caso el valor elegido para el coste es 45).

```
SELECT id_NODO INTO id_nodo_inicio
  FROM ( SELECT id_NODO FROM NODO ORDER BY dbms_random.value) WHERE rownum =1;
SELECT id_NODO INTO id_nodo_fin
  FROM ( SELECT id_NODO FROM NODO ORDER BY dbms_random.value) WHERE rownum =1;

INSERT INTO ENLACE VALUES (id,id_nodo_inicio,id_nodo_fin,45);
COMMIT;
  --SEÑAL
  message :='pintar enlace';
  DBMS_ALERT.SIGNAL('CAMBIO',message);
COMMIT;
```

Figura 41. Creación de un nuevo Enlace.

Otro aspecto importante a destacar es que cada vez que se produce un cambio se debe enviar una señal a la aplicación para que la red dibujada en el panel de dibujo sea refrescada. La manera de hacerlo se especifica también en la figura 41, entre las instrucciones *COMMIT*, y consiste en enviar una señal a través de la función *DBMS_ALERT.SIGNAL*. Esta señal la espera el procedimiento que se detallará a continuación y que es al que invocó la aplicación para mantenerse informada de todos los cambios que se produjesen en la red. Sin embargo, para entender mejor el funcionamiento de las señales que permiten la concurrencia en la aplicación, en apartados posteriores se detallará la implementación de este mecanismo, ya que este apartado sólo se centra en la implementación de los procedimientos utilizados en la Base de Datos.

El procedimiento *Signal* que se observa en la figura 42 espera la señal que indica que ha habido un cambio en la red ejecutado por el procedimiento *Pintar_Red*. Este procedimiento es bastante sencillo, simplemente registra la señal con la función *DBMS_ALERT.REGISTER(name)* y espera con la función *DBMS_ALERT.WAITONE(name, message, status, timeout)* a que esta señal se produzca en el procedimiento *Pintar_Red.*

Para entender mejor la función *DBMS_ALERT.WAITONE(name, message, status, timeout),* los parámetros que la componen indican lo siguiente :

- *Name*: indica el nombre de la señal que espera.
- *Message*: almacena el contenido del mensaje que se quiera transmitir, es un parámetro informativo de entrada y de salida que puede no tener valor.
- *Status*: puede tener dos valores, 1 o 0, donde 1 indica que el tiempo de espera a expirado y 0 indica que la alerta ha ocurrido.
- *Timeout*: indica el tiempo que debe esperar esta función a que la señal ocurra.

```
create or replace PROCEDURE Signal (MESSAGE in OUT VARCHAR2, status out integer)is

BEGIN
  DBMS_ALERT.REGISTER('CAMBIO');
  DBMS_ALERT.WAITONE('CAMBIO',MESSAGE,status, 30000);
END;
```

Figura 42. Procedimiento Signal.

4.3 Diagrama de Clases de la Aplicación

El diagrama de clases de la aplicación queda representado en la figura 14, mostrando las relaciones entre las clases que forman el sistema, como las relaciones de herencia, las relaciones reflexivas y las relaciones asociativas. Además indica los atributos y métodos de cada clase.

Para entender la aplicación desde el punto de vista analítico, en el diagrama de clases se puede observar que hay tres relaciones de herencia, la correspondiente a la clase nodo que hereda de su clase padre, la clase abstracta *JComponent*; la clase *Aplicación* que hereda de su clase padre, la clase abstracta *JFrame*; y la clase *MyThread* que hereda de su clase padre, la clase abstracta *Thread*.

Con respecto a las otras relaciones que aparecen en el diagrama de clases (figura 43) se tiene que:

- Un objeto de la clase nodo es un objeto único ya que no pueden existir dos nodos iguales. Además, un objeto nodo pertenece a sólo un objeto de la clase *Aplicación*, en cambio un objeto de la clase *Aplicación* puede tener cero o varios objetos nodo, esto es debido a que en un principio puede existir en la aplicación un solo nodo, no existir ninguno o existir varios.

  Por otro lado la relación que existe entre la clase nodo y la clase *Enlace* define que entre cada relación de objeto nodo con otro objeto nodo (relación reflexiva) existe un único enlace. Esto tiene sentido ya que un enlace no puede tener asociados más de dos nodos. Sin embargo, un mismo nodo puede estar asociado a más de un enlace, ya que ese mismo nodo puede estar asociado a otro nodo con otro enlace.

- Un objeto de la clase *Enlace* es un objeto único ya que no pueden existir dos enlaces iguales. Además, un objeto *Enlace* pertenece a un solo objeto de la clase *Aplicación*. Sin embargo, un objeto de la clase aplicación puede tener cero o varios objetos *Enlace*.

  Por otro lado, como se observa en el diagrama de clases, los atributos de la clase *Enlace idNodoInicio* e *idNodoFin* presentan la restricción de que el valor de estos atributos nunca deben ser iguales.

- La clase *MyThread* se relaciona tanto con la clase *Aplicación* con una relación 1 a 1, como con la clase *Base de Datos* con una relación de 1 a 1 como se observa en el diagrama de clases de la aplicación de la figura 43.

  Por tanto un objeto de la clase *MyThread* pertenece sólo a un objeto de la clase *Aplicación*, y el objeto de la clase *Aplicación* sólo puede tener un objeto de la clase *MyThread*. Por otro lado, ese mismo objeto *MyThread* sólo podrá tener un solo objeto de la clase *Base de Datos* y ese objeto de la clase *Base de Datos* pertenecerá a un único objeto de la clase *MyThread*.

- Por último un objeto de la clase *Base de Datos* pertenece sólo a un objeto de la clase *Aplicación* y por tanto este objeto sólo tendrá un objeto de la clase *Base de Datos*.

En resumen un objeto de la clase *Aplicación* tendrá cero o varios objetos de la clase *Nodo* y de la clase *Enlace*, además sólo trabajará con un objeto de la clase *MyThread* y con un objeto de la clase *Base de Datos*.

En la figura 43 se observa el diagrama de clases de la interfaz explicado anteriormente.

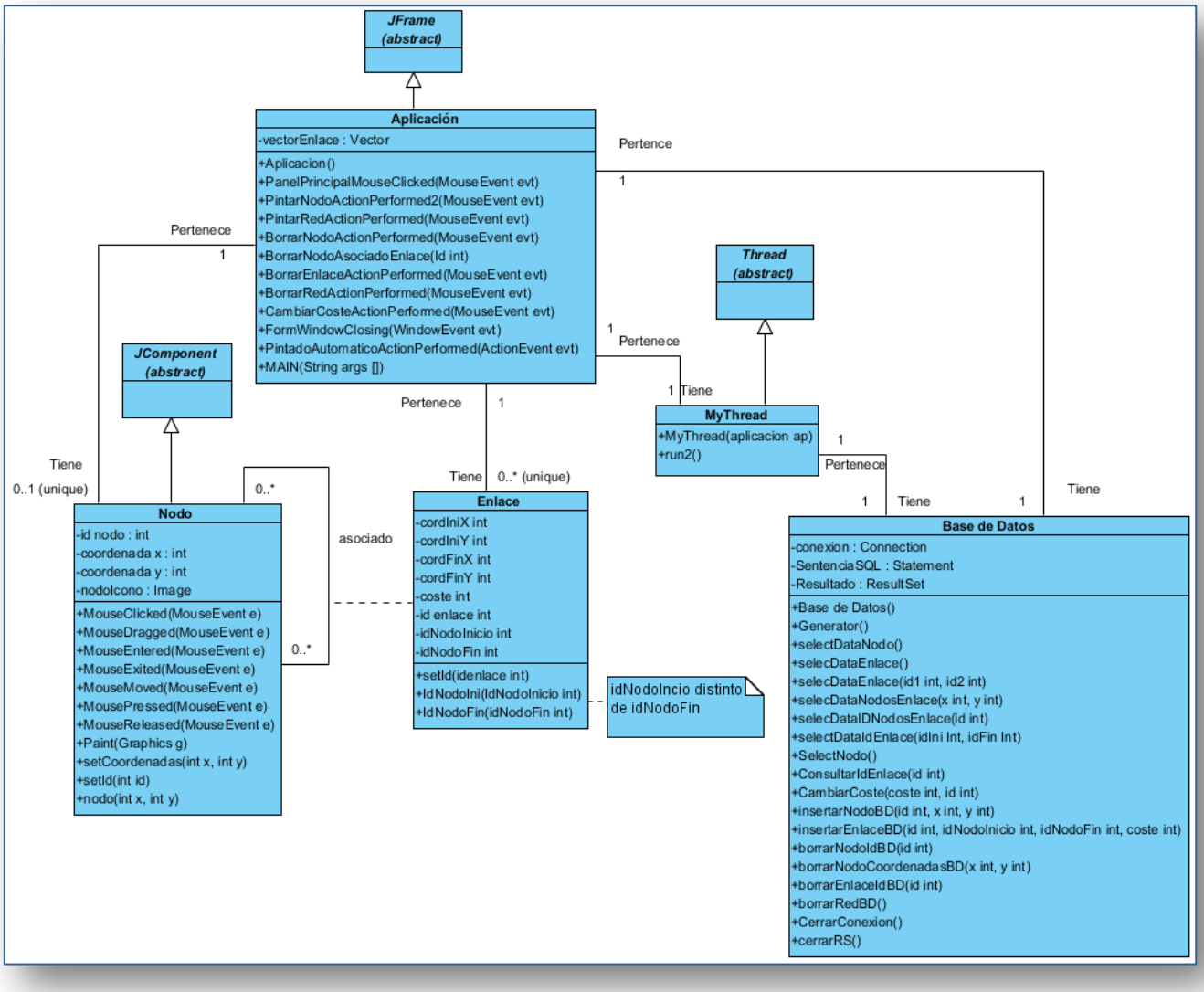


Figura 43. Diagrama de clases.

4.4 Implementación de la Interfaz

Para el diseño visual no hizo falta implementar código de manera manual, ya que la *IDE NetBeans* posibilita la manera de hacerlo de una forma visual, debido a que contiene una *paleta* donde se encuentran todos los componentes y basta con arrastrarlos al lugar preferido.

Una vez que el componente se encuentra adherido al formulario se pueden cambiar sus propiedades correspondientes según convengan, propiedades como: el título, el nombre del componente, la imagen que se quiera establecer a ese componente, la fuente, el tamaño, color… etc. En cuanto a la posición de los componentes, hay que tener en cuenta que dependiendo del gestor de diseño, los componentes se pueden insertar en una posición u otra. En este caso se utilizó el gestor de diseño *AbsoluteLayout*, de manera que la posición de los componentes dependió del diseñador.

Al mismo tiempo que se dibuja en el formulario cada componente y se van cambiando sus propiedades, la *IDE NetBeans* automáticamente va desarrollando el código correspondiente, desde la creación del componente como objeto, hasta los valores de cada propiedad del mismo. Por este motivo el diseño no fue especialmente complicado, pero no hay que olvidarse que aún no se ha implementado ninguna funcionalidad. Es decir, en ese momento la interfaz es en sí una imagen como se muestra en la figura 44, donde se puede observar los componentes por la que está formada la interfaz.

- Los botones *Pintar Nodo, Pintar Red, Pintar Enlace, y Cambiar Coste* son imágenes de un botón en color azul; *Borrar Nodo, Borrar Red, Borrar Enlace* son imágenes de un botón en color rojo para representar la criticidad de su acción.
- Las etiquetas *RED, NODO, ENLACE y COSTES* son imágenes utilizadas para resaltar la división de las principales funcionalidades. Con este fin también se añadieron los separadores.
- Cada cuadro de texto tiene asociado su descripción mediante una etiqueta de texto. La funcionalidad de esto es indicar que información se debe introducir en cada uno de ellos.
- Los cuadros de selección *Pintar Enlace con el ratón, Pintar Nodo con el ratón y Activar Pintado Automático* se sitúan apartados de los demás componentes aunque con diseño similar a las etiquetas de texto para indicar que hay una semejanza en sus funciones.
- Al panel de dibujo se le asignó un color claro de fondo para que la red que se pinte en él se vea claramente, y es de grandes dimensiones para no condicionar demasiado al tamaño de la red.

Por otro lado, se puede comparar la figura 25 con la figura 44**,** y observar la diferencia del diseño inicial con el diseño final. En cuanto a componentes, por ejemplo, los cuadros de selección no existían en el diseño inicial debido a que en un principio no se necesitaban para el correcto funcionamiento de la interfaz, además en el diseño final se utilizan imágenes para resaltar los botones y las etiquetas que intuyen la división de los cuatro apartados.

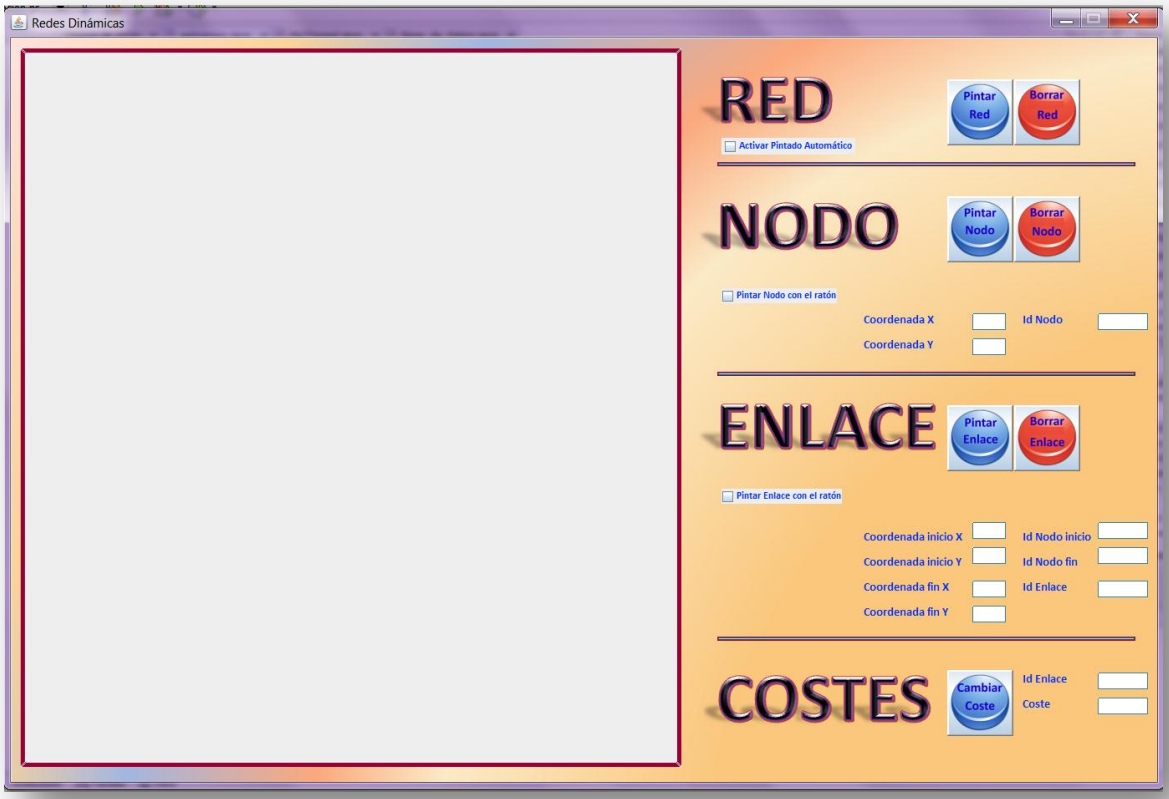


Figura 44. Diseño visual final de la interfaz.

4.5 Implementación de las funcionalidades

Una vez analizado el diagrama de clases de la aplicación e implementado el diseño visual de la interfaz Gráfica, es importante especificar la implementación de cada funcionalidad de la aplicación. Además, aunque no se indique en estas explicaciones, en cada método se incluyó un tratamiento de excepciones para controlar acciones que no podrían realizarse correctamente como introducir una coordenada con formato de texto.

Dicho esto, se explica a continuación como se implementó cada funcionalidad de la aplicación para hacer posible el simular el comportamiento de una red dinámica.

Pintar

Pintar consiste en representar toda la información referida a los nodos y enlaces almacenados en la Base de Datos en el panel de dibujo de la interfaz gráfica. Esta funcionalidad se ejecuta cada vez que se produce cualquier cambio en la base de datos, por lo que es necesario que cada vez que esto se produzca se invoque al método *Paint().*

Toda la implementación de pintado está definida en el método *Paint(),* el cual se ejecuta con cada llamada *repaint(),* de manera que cada vez que se realiza esta llamada, el método *Paint()* pinta toda la información que está almacenada en la base de datos.

Para pintar toda la información referida a la red, este método invoca a un procedimiento de la clase *Base de Datos* encargado de extraer toda la información de cada tabla (la tabla *NODO* y la tabla *ENLACE*) y almacenarla en un objeto *ResultSet* gracias a la consulta que ejecuta el objeto *Statement.* Una vez que se ha obtenido el resultado basta con recorrer el objeto *ResultSet* y en cada iteración ir pintando en el panel de dibujo de la interfaz cada nodo (con su imagen, identificador y coordenadas) o un enlace (con su información de coste e identificador) invocando los métodos de pintado del objeto *Graphics* que se observan en la figura 44, los cuales se encargan de pintar la imagen del nodo, con un texto indicando las coordenadas y su identificador, o una línea que une dos nodos en el caso de tener que pintar un enlace, éste también acompañado de un texto referido al coste y a su identificador.

En la figura 45 se observan métodos de pintado como *setColor(),* que cambia el color con el que se va a pintar (en el caso de la figura 45 a azul); *drawString(),* que pinta en la pantalla un texto en las coordenadas que se indiquen (en el caso de la figura 45 el texto indica las coordenadas de un nodo); y por último el método *drawImage()* que pinta una imagen que en el caso de la figura 45 coincide con la imagen del nodo en las coordenadas que indican *x* e *y* con un determinado tamaño.

```
g2.setColor(Color.BLUE);
g2.drawString( "[" + X + ", " + Y + "]",X,Y);
g2.setColor(Color.red);
g2.drawString("id: "+ID,X,Y + 40);
g2.drawImage(nod, X, Y, 30, 30, this);
```

Figura 45. Métodos de pintado del objeto Graphics.

Para poder pintar correctamente cada elemento de la red se utiliza el objeto *Graphics* del *PanelPrincipal*, pues se debe hacer en las coordenadas del panel donde se va a dibujar y no en las coordenadas del formulario completo, pues si se emplea el objeto *Graphics* del formulario el elemento se pintaría en las coordenadas incorrectas. En la figura 46 se observa esta situación donde el rectángulo azul representa la interfaz completa y el cuadrado gris representa el panel de dibujo y, por tanto, las coordenadas correctas que se deben utilizar para el pintado.

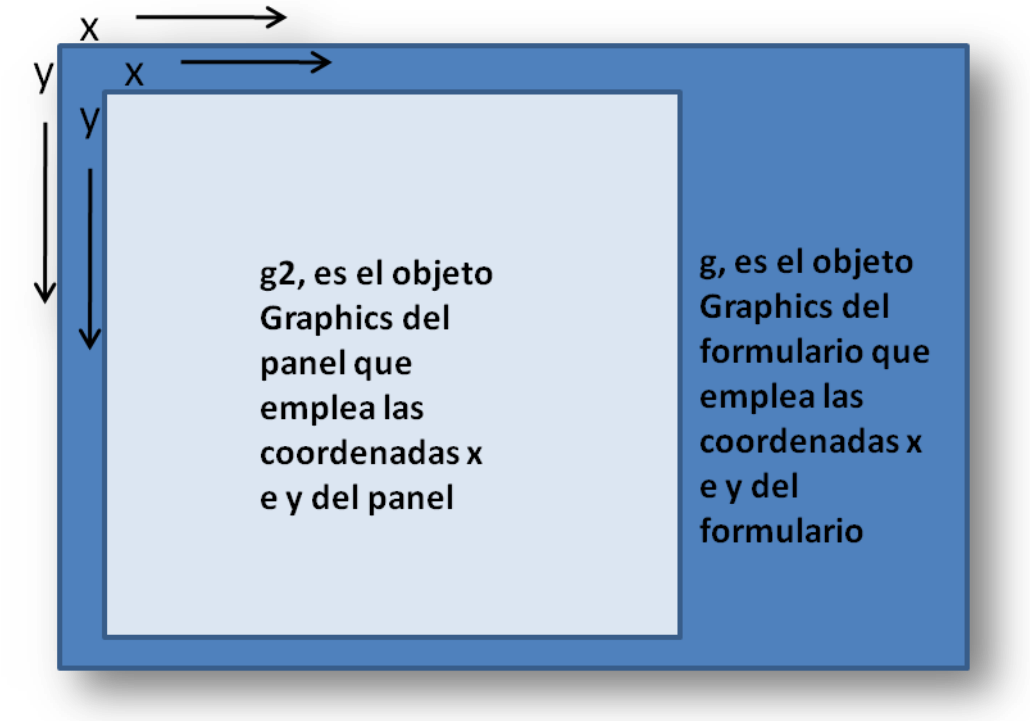


Figura 46. Distintos objetos Graphics.

En la figura 47 se puede observar la red pintada compuesta de nodos y enlaces.

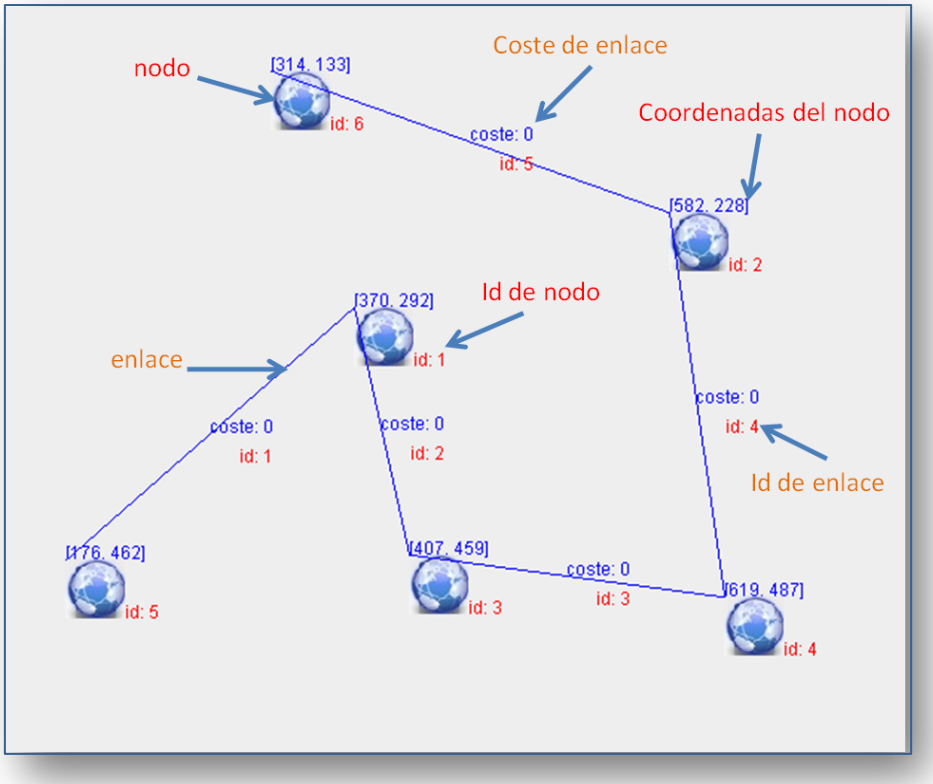


Figura 47. Red pintada.

Pintar Red

Esta funcionalidad consiste en pintar toda la red almacenada en la base de datos en el momento de dar clic en el botón *Pintar Red*. Este botón invoca al método *jPintarRedActionPerformed()* cuya funcionalidad es pintar la red entera mediante una llamada al método *repaint()* que a su vez invocará al método *Paint()*, el cual se encargará de leer directamente de la base de datos toda la información que tiene almacenada y pintarla como se explicó anteriormente.

Borrar Red

La funcionalidad *Borrar Red* consiste en borrar por completo la red que está pintada en el panel de dibujo de la interfaz gráfica y, por tanto, almacenada en la base de datos.

Para ejecutar esta funcionalidad, en el momento de dar clic en el botón *Borrar Red* se invoca al método *jBorrarRedActionPerformed()*, cuya funcionalidad es borrar todo el contenido de la base de datos. Esta acción se resume en invocar al método *borrarRedBD()* de la clase base de datos que se encarga de ejecutar una sentencia *SQL* que borra todo el contenido de la tabla *NODO*, y por consecuencia del borrado en cascada también todos los enlaces asociados a esos nodos.

Por último se invoca al método *Paint()* con la llamada a *repaint()*, y dado que ya no existe información en la base de datos, no habrá información para pintar, por tanto no se pinta nada y el panel de dibujo se queda vacío.

Pintar *Nodo*

Esta funcionalidad se encarga de pintar en el panel de dibujo de la interfaz gráfica un nodo en la posición indicada por el usuario. Habrá dos posibilidades para pintar un nodo: mediante la acción de dar clic al botón *Pintar Nodo* insertando previamente las coordenadas del mismo, o directamente dando clic en el panel en la posición donde se quiere pintar el nodo:

- En el momento de dar clic en el botón *Pintar Nodo,* se invoca el método *jPintarNodoActionPerformed()* cuya funcionalidad es pintar un nodo en el panel con las coordenadas que sean introducidas en el cuadro de texto.

  Una vez que se obtienen las coordenadas de los cuadros de texto y se ha generado el identificador del nodo (mediante al función *Random()*) estos datos se pasarán como parámetro al método *insertarNodoBD*() de la clase base de datos que se encarga de insertar una tupla en la tabla nodo con la información que se acaba de obtener. En ese momento se realizan dos comprobaciones: si existe otro nodo con ese identificador, mediante la restricción de clave primaria, o si existe otro nodo con esas coordenadas, mediante el disparador *EXCLUSIVIDAD_NODOS.* Si el nodo ya existe se comunica tal hecho al usuario mediante un cuadro de diálogo.

- En cuanto a la posibilidad de pintar un nodo mediante la acción de dar clic en el panel de dibujo, esto provoca que se invoque el método *PanelPrincipalMouseClicked(MouseEvent evt),* cuya funcionalidad es capturar las coordenadas del nuevo nodo al dar clic en el panel. El evento *Mouse Clicked* es del panel principal, esto quiere decir que sólo se produce si se da clic en el panel principal, lo que evitará que se dibuje un nodo fuera del área del panel de dibujo de la interfaz gráfica.

  Para que se ejecute este método, primero se comprueba que el cuadro de selección *Pintar Nodo con el ratón* está activado. Una vez hecho esto, cada vez que se produzca el evento del clic, que se pulse el botón del ratón, se capturarán las coordenadas gracias a *evt.getX()* y *evt.getY(),* se generará el nuevo identificador del nodo, y se ejecutará la inserción en la base de datos exactamente igual que en el caso anterior.

Después de insertar la información del nodo en la base de datos, se llama al método *Paint()* con la llamada *repaint()* para que pinte el nuevo nodo, añadiéndolo al panel como un componente. En el apartado de pintar enlace se explicará el porqué debe añadirse como un componente al panel de dibujo.

En la figura 48 se observan las dos posibilidades de pintar un nodo.

Figura 48. Modalidad de pintar un nodo.

Borrar *Nodo*

Esta funcionalidad consiste en borrar un nodo elegido mediante sus coordenadas o mediante su identificador introducido en los cuadros de texto correspondientes en el momento de dar clic en el botón *Borrar Nodo* invocando al método *jBorrarNodoActionPerformed().*

Para ejecutar esta función primero se comprueba la modalidad elegida, si es mediante coordenadas o mediante el identificador, para elegir si invocar al procedimiento *borrarNodoIdBD(id)* de la clase base de datos, que ejecuta una sentencia *SQL* que busca en la base de datos un nodo asociado a ese identificador y lo borra, o invocar al procedimiento *borrarNodoCoordenadasBD(X,Y)* de la clase *Base de Datos*, que ejecuta una sentencia *SQL* que busca en la base de datos el nodo asociado a esas coordenadas y lo borra. Si la consulta no devuelve ningún resultado, se le informa al usuario que no existe el nodo que desea borrar.

Por último debe pintarse la nueva situación de la red llamando al método *Paint()* con la llamada *repaint().*

La figura 49 enseña las dos modalidades de borrado.

Figura 49. Modalidad de borrado de Nodo.

La figura 50 enseña el cambio que tiene la red cuando se borra un nodo con más de un enlace asociado, se observa que al borrar el nodo con id = 2 desaparece tanto el nodo

como los enlaces que tenía asociados (por el tipo de borrado en cascada implementado en la base de datos).

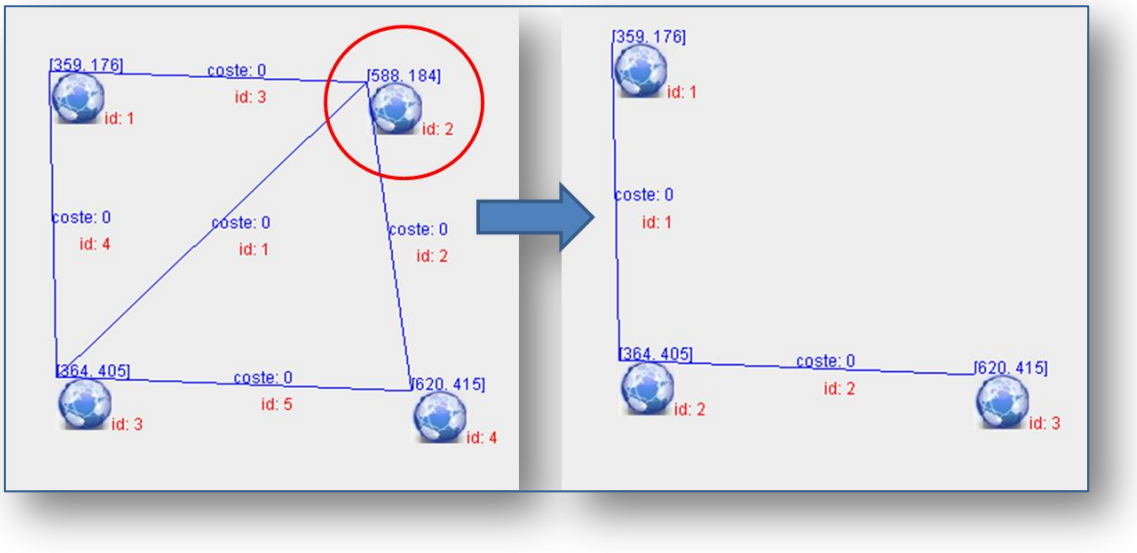


Figura 50. Transcurso de borrado.

Pintar *Enlace*

Pintar un enlace consiste en pintar una línea que conecte dos nodos en el panel de dibujo de la interfaz gráfica. Para pintar un enlace hay tres posibilidades: introducir el identificador del nodo de inicio y el identificador del nodo de fin, introducir las coordenadas de inicio y las coordenadas de fin, o dibujar una línea con el ratón que una dos nodos seleccionando anteriormente el cuadro de selección *Pintar Enlace con el ratón.*

En la figura 51 se observan estas tres modalidades.

Figura 51. Modalidades de pintado de enlace.

A continuación se detallan las distintas modalidades de pintado de un enlace:

- En el momento de dar clic en el botón *Pintar Enlace* se llama al método *jPintarEnlaceActionPerformed(),* el cual comprueba primero si se va a pintar por coordenadas de inicio y fin o por identificador de nodo de inicio e identificador de nodo de fin.

  En el caso que el pintado sea por los identificadores de nodo de inicio y de fin, se inserta esta información en la base de datos, junto con el identificador del nuevo enlace generado anteriormente con la función *Random()* y el coste, invocando al procedimiento *insertarEnlaceBD(id, idNodo inicio,idNodo fin, coste)* de la clase *Base de Datos*, el cual ejecuta la inserción en la base de datos, previa comprobación de que los identificadores de nodos existen en la tabla de nodos gracias a que tanto el identificador del nodo de inicio como el del nodo de fin son claves ajenas de la tabla *NODO*.

  Por otro lado, si el pintado es por coordenadas de inicio o de fin de enlace, dado que para insertar un enlace en la base de datos se necesita saber el identificador del nodo de inicio y el del nodo de fin, se deberá ejecutar una consulta sobre la tabla de *NODO* que obtenga dichos identificadores de nodo a partir de las coordenadas introducidas. Si la consulta no devuelve ningún resultado quiere decir que no existe un nodo con esas coordenadas.

  En ambos casos, si la consulta no devuelve ningún resultado se informa al usuario del error y no se inserta el enlace.

- Para pintar un enlace con ayuda del ratón se implementó la clase *Nodo* como una clase que hereda de la clase *JComponent*, ya que es necesario que un nodo sea un componente y que implemente sus propios eventos de ratón *MouseListener y MouseMotionListener* y su propio método *Paint().*

  Para dibujar una línea con el ratón se implementaron los siguientes métodos ejecutados en el siguiente orden: primero al dar clic en el componente nodo se invoca al método *mousePressed(MouseEvent e)*; después se empieza a dibujar la línea en el panel, al mismo tiempo que se arrastra el ratón por el panel, en ese momento se invoca al método *mouseDragged(MouseEvent e)* y *Paint(Graphics g)* para ir pintando la línea; por último al soltar el ratón, se invoca el método *mouseReleased(MouseEvent e)* y *Paint(Graphics g)* para pintar el resultado final. A continuación se detalla la implementación de cada método:

  - *Paint(Graphics g):* este método se encarga de pintar la información que se va almacenando en el vector de enlace a medida que el ratón se va desplazando por el panel de dibujo de la interfaz gráfica a la hora de pintar el enlace, ya que cada vez que el ratón se mueve por el panel tiene

nuevas coordenadas de fin, por lo que es esta información la que se va actualizando en el vector y por consiguiente se va pintado.

La forma de pintar el enlace según se va moviendo el ratón por el panel es muy sencilla, basta con pintar con cada llamada a este evento el enlace que está almacenado en este vector y que está cambiando de tamaño con cada movimiento de ratón.

Es importante mencionar que este método sólo se encarga de pintar la información del vector de enlaces y no toda la información de la base de datos como el método *Paint()* principal. Esto es así porque era muy costoso, en cuanto a rendimiento, que los dos métodos *paint()* pintaran la información de la base de datos debido a que ralentizaba demasiado el pintado de la red en el panel (por cada movimiento de ratón era necesario que los dos métodos consultaran en la base de datos para pintar). Es decir, existen dos métodos *paint()* distintos: el principal que pinta la información almacenada de la base de datos y el método *paint()* redefinido de la clase *Nodo* que sólo pinta el enlace que está dibujando el movimiento del ratón.

- *mousePressed(MouseEvent e):* se ejecuta en el momento de dar clic en el panel, en ese momento se capturan las coordenadas donde se ha dado clic y se comprueba si en las coordenadas apuntadas existe un componente nodo. Si es así, y además el cuadro de selección *Pintar Enlace con ratón* está seleccionado, se inserta por el momento un enlace con iguales coordenadas de inicio que de fin en el vector de enlaces. Hay que tener en cuenta que este enlace en ese momento es un punto.

- *mouseDragged(MouseEvent e):* este evento ocurre cuando el ratón se va moviendo por el panel y el cuadro de selección *Pintar Enlace con el Ratón* esta seleccionado. Al último enlace creado se le va actualizando la información de sus coordenadas de fin y se va repintando conforme el ratón se va moviendo, esto es posible gracias a que a las coordenadas iniciales se les va sumando las coordenadas del punto donde se encuentra el ratón en ese momento.

- *mouseReleased(MouseEvent e):* en el momento de soltar el ratón, se comprueba si se ha soltado el ratón en un componente nodo. Si es así y además el cuadro de selección *Pintar Enlace con ratón* está seleccionado, las coordenadas de componente nodo se introducen en las coordenadas de fin del enlace que se estaba pintando, posteriormente con la información guardada en el vector de enlaces, se consulta en la base de datos el identificador de nodo correspondiente a esas coordenadas

almacenada y una vez que está toda la información completa del enlace se inserta la tupla con los datos en la Base de Datos.

Si por el contrario al soltar el ratón se comprueba que no se ha soltado en las coordenadas de un componente nodo, el enlace finalmente no se inserta en la base de datos y por consiguiente no se pinta.

En cualquiera de los casos de pintado de enlace, antes de insertar la información en la base de datos se dispara el disparador *EXISTE_ENLACE_POR_NODOS* que comprueba que no existe otro enlace igual independientemente de la dirección que tenga. Además, siempre que se inserte definitivamente el enlace en la base de datos se invoca al método *Paint()* principal para pintar el enlace creado junto con la red que ya existía.

En la figura 52 se observa el proceso de dibujo de la línea, donde primero se selecciona un componente nodo, se va pintando la línea y por último al soltar el ratón en otro componente nodo el enlace queda pintado definitivamente.

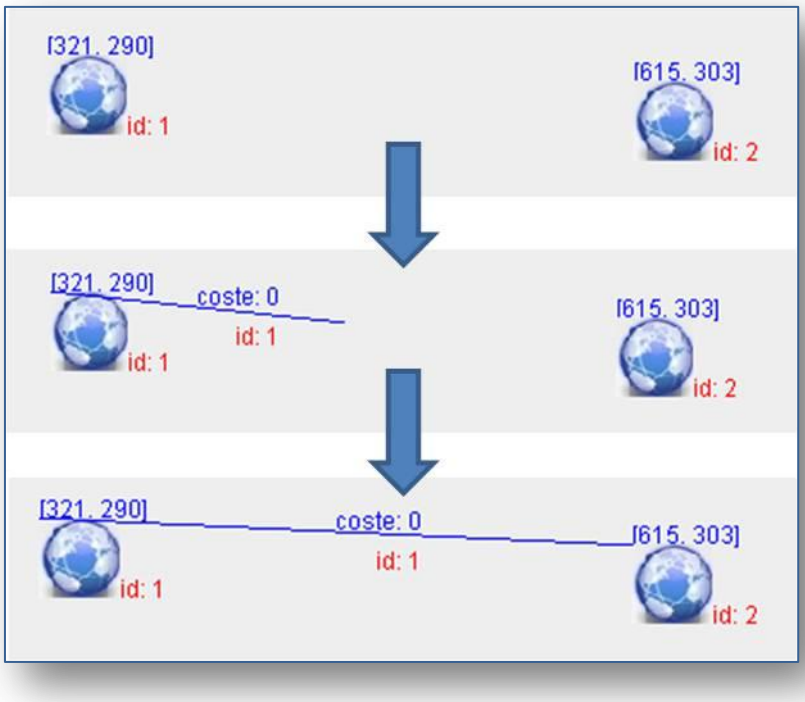


Figura 52. Transcurso de dibujo de enlace con el ratón.

Borrar un *Enlace*

Para borrar un enlace existen distintas posibilidades: mediante la introducción de las coordenadas de inicio y coordenadas de fin, o mediante la introducción del identificador del enlace.

En la figura 53 se puede observar las posibilidades de borrado del enlace.

Figura 53. Modalidades de borrado de enlace.

En el momento de dar clic en el botón *Borrar Enlace* se llama al método *jBorrarEnlaceActionPerformed()*. Este método comprueba si el borrado es mediante las coordenadas de inicio y de fin o mediante la introducción del identificador del enlace a borrar para obtener correctamente los datos de los cuadros de texto correspondientes. Una vez hecha esta comprobación la implementación del borrado del enlace es la siguiente:

- Si el borrado es por identificador del enlace se consulta en la base de datos si éste existe, y en ese caso se llama al método *BorrarNodoAsociadoEnlace()* encargado de consultar en la base de datos qué nodos tiene asociados ese enlace para preguntar al usuario si desea borrar uno de los nodos o los dos nodos. En el caso de confirmar el borrado también de los nodos, estos se borrarían al igual que sus enlaces asociados debido al borrado en cascada como. Si la respuesta del usuario es negativa, simplemente se borrará el enlace.

- En caso de que el borrado sea por las coordenadas de inicio y fin del enlace primero se consulta en la base de datos los identificadores de los nodos que se correspondan con esas coordenadas, ya que en el momento que se obtienen los identificadores de los dos nodos, tanto de inicio como de fin, se puede obtener el identificador del enlace a borrar que corresponde a esos nodos y así llamar al método *BorrarNodoAsociadoEnlace()*. Por último se preguntará al usuario si desea borrar los nodos asociados y se borrará el enlace.

En ambos casos, en el momento que la consulta no obtenga un resultado se informará al usuario del error de inexistencia del enlace a borrar o de alguna coordenada errónea que implique que no existe ningún identificador de Nodo asociado a esas coordenadas. Por último se pintarán los cambios producidos llamando al método *Paint()* con la llamada *repaint()*.

Cambiar costes

Esta funcionalidad consiste en cambiar el coste de un enlace en el momento de dar clic al botón *Cambiar Coste* invocando el método *jCambiarCosteActionPerformed()*, cuya funcionalidad es recoger la información del

identificador del enlace y del coste que se va actualizar de los cuadros de texto correspondientes. Una vez que se tiene esta información se consulta en la base de datos, con el método *CambiarCoste* de la clase *Base de Datos*, el enlace asociado a ese identificador para después cambiar su coste por el coste que se introdujo. Si la consulta no devuelve ningún resultado se informa al usuario de que no existe ningún enlace con ese identificador.

El las figuras 54 se observa la red sin haberse producido aún el cambio.

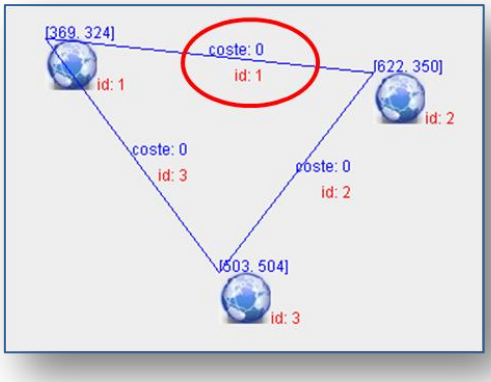


Figura 54. Coste.

En la figura 55 se observa la introducción de los datos en la interfaz gráfica del coste del enlace que se va a modificar y en la figura 56 la red con el cambio producido.

Figura 56. Cambiar coste.

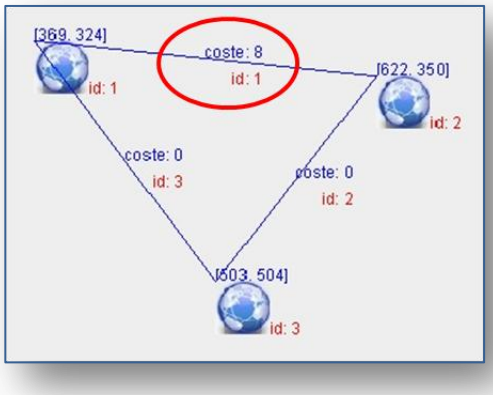


Figura 55. Cambio de coste producido.

Pintado automático

La funcionalidad del pintado automático consiste en conseguir que se produzcan cambios en la red sin ayuda de un usuario que interactúe con la aplicación. Esto es posible gracias a la introducción de las señales que permiten que la base de datos interactúe con la interfaz gráfica, así como a la introducción de concurrencia para que al mismo tiempo el usuario pueda seguir pintando en ella. A continuación se detalla la manera de implementarlo:

Primero se crea la clase *MyThread* que extiende de la clase *Thread*. Esta clase creará una nueva instancia de la clase *Base de Datos* que será con la que trabaje para evitar utilizar la instancia de la clase *Base de Datos* que usa la clase *Aplicación* y producir interbloqueos.

En el momento en que el usuario active la casilla *Activar Pintado Automático* se llamará al procedimiento implementado en la clase *Base de Datos activarJOB*. De esta manera el *Job jobPintarRed* empezará a ejecutar el procedimiento *Pintar_Red* encargado de realizar cambios en la base de datos cada 5 segundos. Además se realizará la activación de *MyThread* mediante la llamada a la función *start()* para que empiece a ejecutarse paralelamente a la instancia de la clase *Aplicación*.

En el momento de la activación de *MyThread* empezará a ejecutar su método *run()*, cuya función será llamar de manera reiterada al procedimiento *Signal* para que éste espere a que le llegue la señal enviada desde el procedimiento *Pintar_Red*. Este bucle (implemenatdo con un *while*) continuará mientras que una variable global de esta clase no cambie a estado *false* el método run().

En cada iteración del bucle, en el momento en el que el procedimiento *Signal* recibe una señal, se invoca al método *Paint()* de la clase *Aplicación* para pintar el nuevo cambio producido en la red.

Por último cuando el usuario desactiva la casilla de *Activar Pintado Automático* se llamará al procedimiento *desactivarJob* para que el *Job jobPintarRed* deje de funcionar y por tanto de ejecutar el procedimiento que provoca los cambios en la red. Además, será en este momento cuando la variable de control que permite que el *while* llame constantemente al procedimiento *Signal* dentro del método *run()* cambie el valor de *true* a *false* para que el *while* termine su ejecución.

En el siguiente apartado se presentará una batería de pruebas de algunas de las funcionalidades más relevante de la aplicación para demostrar que la aplicación funciona correctamente.

Capítulo 5. Pruebas

5.1 Introducción

En cada paso de la implementación se iban ejecutando pruebas para verificar el correcto funcionamiento de la aplicación o para encontrar fallos en la misma y poder encontrar una solución para eliminarlos. Pero aunque el desarrollo de la aplicación ya esté finalizado, es importante presentar un pequeño resumen de pruebas tanto para ver los resultados obtenidos como para comprobar que todo funciona como se esperaba, y para encontrar posibles fallos.

En este apartado se enumerarán una serie de pruebas ejecutadas sobre la aplicación para demostrar que las funcionalidades más relevantes se realizan correctamente, y por tanto demostrar que se resolvieron los problemas presentados al inicio del desarrollo de este proyecto.

El modo de prueba consiste en ejecutar acciones sobre la aplicación como pintar un nodo con las mismas coordenadas de otro nodo; pintar un enlace con los nodos extremos iguales que otro enlace ya existente; o intentar borrar un nodo que no existe entre otras. Por tanto, las pruebas que se van a ejecutar son pruebas de errores, es decir, pruebas que describan el comportamiento que debe tener la aplicación ante una situación en la que se ejecuta una acción errónea.

5.2 Pruebas realizadas

Este apartado contiene una lista de las pruebas más relevantes realizadas sobre la aplicación así como sus resultados, para comprobar que se han cumplido los objetivos impuestos. A continuación se enumeran las siguientes pruebas:

- **Prueba 1**: pintar un nodo con iguales coordenadas al de un nodo ya existente. Para ello, como se observa en la figura 57, se introducen las mismas coordenadas que el de un nodo ya existente y se observa que el resultado obtenido es que aparece un cuadro de diálogo informando del error.

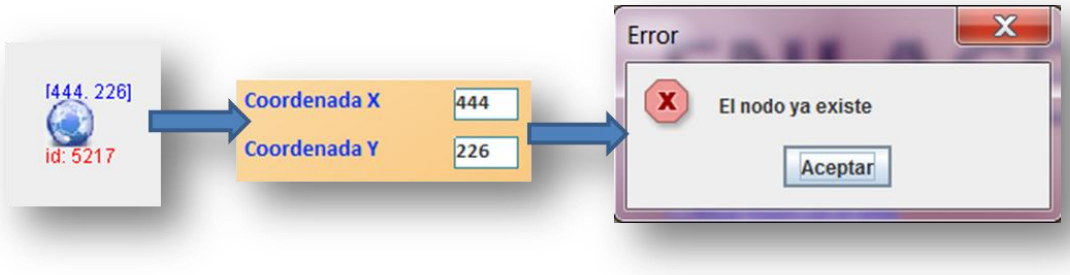


Figura 57. Prueba 1.

- **Prueba 2**: pintar un enlace con las coordenadas de los nodos extremos que otro enlace ya contenga. Para ello, como se observa en la figura 58, se introducen las coordenadas de los nodos extremos que se quieren unir con un nuevo enlace, existiendo ya uno creado con anterioridad con esas características, y se observa que aparece un cuadro de diálogo informando del error.

Figura 58. Prueba 2.

- **Prueba 3**: pintar un enlace con las mismas coordenadas en los nodos extremos que un enlace ya existente pero en sentido contrario. Para ello, como se observa en la figura 59, se introducen las coordenadas de los nodos extremos que se quieren unir mediante un nuevo nodo y que ya tenían un enlace asociado pero en sentido contrario. Al hacer esto se observa que aparece un cuadro de diálogo informando del error.

Figura 59. Prueba 3.

- **Prueba 4**: pintar un enlace con los mismos identificadores de nodos extremos que un enlace ya existente. Como se observa en la figura 60, si se introducen los identificadores de los nodos extremos de un enlace que ya existe para intentar pintar un nuevo enlace aparece un cuadro de diálogo informando del error.

Figura 60. Prueba 4.

- **Prueba 5**: pintar un enlace con los mismos identificadores de nodos extremos que un enlace ya existente pero en sentido contrario. Como se observa en la figura 61, si se introducen los identificadores de los nodos extremos de un

enlace que ya existe, en sentido contrario, para intentar pintar un nuevo enlace aparece un cuadro de diálogo informando del error.

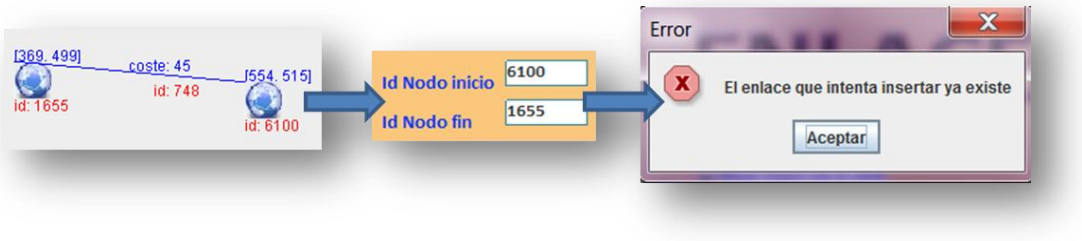


Figura 61. Prueba 5.

Además se comprueba el funcionamiento correcto de los disparadores *EXCLUSIVIDAD_NODOS* y *EXISTE_ENLACE_POR_NODOS* que no permiten la inserción de nodos en cuyas coordenadas ya exista otro nodo o de enlaces que unan nodos ya enlazados respectivamente. Por ejemplo, en el caso de la prueba 1 el disparador *EXCLUSIVIDAD_NODOS* se ejecuta y no permite que se inserte un nuevo nodo con las mismas coordenadas de un nodo ya existente, y en el caso de la prueba 3 y 5 se ejecuta el disparador *EXISTE_ENLACE_POR_NODOS* para evitar que sea insertado un nuevo enlace pero con los nodos extremos iguales que los de un enlace que ya exista, pero en sentido contrario.

Además, gracias a los controles propios de la base de datos se evitan violaciones de las restricciones de integridad como en el caso de prueba 2 y 4, en la que se comprueba, gracias a la restricción de integridad de clave alternativa de los nodos extremos de un enlace, que no se puede insertar otro enlace con nodos extremos iguales.

Capítulo 6. Conclusión y Líneas Futuras

6.1 Conclusiones y líneas futuras

En este apartado se recogen todas las conclusiones obtenidas tras el desarrollo de este proyecto en el que se pretendía desarrollar una aplicación con una interfaz gráfica que fuera capaz de representar el comportamiento de una red dinámica dibujando en su panel de dibujo todos los cambios que se producían sobre ella a la vez que la información de esta red era almacenada en una base de datos espacial.

El desarrollo de este proyecto se puede dividir en dos grandes etapas. La primera en la que se desarrolló una interfaz gráfica sin una base de datos asociada que almacenara toda la información de la red pero que dibujaba en el panel de dibujo la red, debido a que ésta estaba almacenada en memoria principal. La segunda etapa es aquella en la que se desarrolló una interfaz gráfica asociada a una base de datos espacial que almacenaba toda la información referida a la red.

En la primera etapa de desarrollo del presente proyecto la toma de contacto con *Java* fue muy complicada, ya que había desconocimiento total del lenguaje, sin embargo, gracias a la amplia documentación y también a un gran *IDE* como *NetBeans*, que existe para este lenguaje, poco a poco la interfaz gráfica fue tomando forma. Se cumplía de esta manera el objetivo que pedía la representación en una interfaz gráfica de una red dinámica.

En cuanto a la segunda etapa, esta fue más compleja, debido a la integración de una base de datos espacial que almacenara toda la información referente a la red que se pintaba. En primer lugar, antes de integrar la base de datos con la interfaz gráfica se obtuvo una visión global de lo que eran los sistemas de información geográfica y lo interesante e importante que son hoy en día para muchos campos de investigación, ya que ayudan a representar los datos espaciales, intentado asemejarlos en todo momento con la realidad gracias a los tipos de datos espaciales y funciones que tienen este tipo de bases de datos.

Por otro lado, la integración de la base de datos con la interfaz gráfica tiene conclusiones muy positivas, ya que aparte de poder disponer de la información de una red en todo momento para analizar el comportamiento de una red dinámica, la complejidad de la implementación de la aplicación disminuye considerablemente, al poder hacer uso de funciones de borrado o inserción en una base de datos que ya están implementadas y que en un primer momento, sin una base de datos, era necesario escribir código para controlar cualquier tipo de situación, que por el contrario con un

buen diseño de la base de datos, muchas restricciones de integridad quedaban impuestas. En resumen las líneas de código se redujeron considerablemente.
Además, con la utilización de una base de datos espacial, se pueden representar datos geométricos, como un simple punto, gracias a las características y funciones del paquete *Oracle Spatial*.

De esta manera, el objetivo global de la aplicación quedaba cumplido: existía una interfaz gráfica que dibujaba el comportamiento de una red dinámica a la vez que almacenaba toda la información de ésta en una base de datos espacial. Pero faltaba un último paso para que el funcionamiento completo de la aplicación fuera del todo correcto: la integración de una ejecución con *hilos*, gracias a la clase *Thread* de *Java*, en la aplicación para que el rendimiento de ésta no se viera afectado al poder ejecutar varias tareas a la vez. Se concluye por tanto que gracias a la multitarea el rendimiento de la aplicación mejoró notablemente.

Por último, comparando el tiempo que tarda en pintar la red la interfaz gráfica cuando el almacenamiento de la información de la red está en memoria principal con el tiempo que tarda en pintar la red la interfaz gráfica cuando el almacenamiento de la información de la red está en la base de datos, se comprueba que se tarda más tiempo en pintar la red cuando está almacenada en la base de datos, ya que cada vez que se desee pintar la red se debe acceder a la base de datos mediante unas sentencias *SQL* y esto consume tiempo. El tiempo son 0,8 segundos frente a 2,5 segundos, y aunque la diferencia puede ser pequeña, hay que tener en cuenta que la red es relativamente pequeña, en cambio, si la red fuera muy grande, como por ejemplo una red de contactos en una red social, el tiempo se vería incrementado considerablemente. Cabe destacar que gracias a la optimización de las consultas se reduce un poco ese tiempo.

Además se concluye también que si la aplicación es ejecutada dentro de la red donde se encuentra el servidor donde está la base de datos, la ejecución es mucho más rápida que si se ejecuta fuera de esa red.

Aunque el problema haya sido resuelto, es evidente que si se sigue trabajando sobre él se obtendría mejorías o nuevas funcionalidades interesantes de cara al estudio del comportamiento de redes dinámicas. Por este motivo se especifican a continuación las siguientes líneas futuras a tener en cuenta para el presente proyecto:

1. **Modificación de la red con el ratón:** esta es una funcionalidad bastante interesante que puede ser implementada y que consiste en seleccionar un componente con la ayuda del ratón y arrastrarlo por todo el panel para cambiar su posición, o si se desea borrarlo, que se pueda hacer dando simplemente clic con el ratón sobre él, previa confirmación por parte del usuario de que esto es lo que se desea hacer. Además, si se arrastra por el panel de dibujo, por ejemplo, un nodo y éste tiene asociado un enlace, éste debe también ser cambiado de posición por el panel de dibujo, como si el nodo y el enlace estuvieran unidos. Hay que tener en cuenta que la información de localización cambia, por tanto debe cambiar también en la base de datos.

2. **Resolución de la interfaz gráfica:** por falta de tiempo no se pudo adaptar la aplicación a cualquier resolución de pantalla para evitar que en un momento dado si el tamaño de la interfaz gráfica es mayor al tamaño de la pantalla de un

ordenador no se pueda visualizar completamente la interfaz. Por tanto sería interesante que la interfaz gráfica se adaptara sin problemas a cualquier resolución de pantalla.

3. **Implementación del cálculo de costes:** partiendo de una serie de parámetros como la distancia entre dos nodos se puede asignar un coste al enlace gracias a funciones de cálculo de distancia entre dos objetos que vienen implementadas en el paquete *Oracle Spatial*.

4. **Mejora del refresco del dibujo:** por falta de tiempo no se ha podido ahondar más en este tema, el cual es bastante complejo, pero se podría investigar más sobre cómo hacer posible que el repintando de la red se ejecutara más rápido y que, por ejemplo, para pintar un nuevo nodo no haga falta volver a pintar la red entera, ya que esto ralentiza la ejecución de la aplicación.

5. **No sobreponer un nodo encima de otro:** puede existir algún caso en el que la modalidad de pintado automático pinte un nodo encima de otro, esto, a nivel visual, puede resultar un poco confuso, pero se podría resolver implementado alguna función, o utilizando funciones del paquete *Oracle Spatial*, que antes de pintar un nodo calculara en un área determinado alrededor suyo si existe algún otro nodo, de esta forma se evitaría que se superponga un nodo encima de otro.

6. **No almacenar información fuera del área especificada en el *Metadata:*** aunque la aplicación está diseñada de manera que si se da clic con el ratón fuera del área del panel de dibujo no se dibuje nada en él, en la base de datos se podría almacenar información sobre un nodo con coordenadas fuera del área del panel de dibujo. En un principio se suponía que con la especificación en el *Metadata* del espacio geográfico donde se pretendía dibujar la red esto quedaba controlado, pero sin embargo este control se ha comprobado que no existe. Por tanto, es interesante investigar más sobre este tema para controlar dichas inserciones, ya que parece ser que es el propio paquete *Oracle Spatial* el que no controla esto.

7. **Diseño visual de la interfaz gráfica:** el diseño visual de la interfaz gráfica se podría mejorar tomando como referencia el amplio número de patrones de diseño que existen para hacer una interfaz de usuario más intuitiva y con un diseño más vistoso.

Acrónimos

A continuación se presenta la siguiente lista de acrónimos utilizados.

- *API: Application Programming Interface.*
- *JDBC: Java Database Connectivity.*
- *SQL: Structured Query Language.*
- *ADO: Active Data Objects.*
- *ODBC: Open Data Base Connectivity.*
- *IDE: Integrated Development Environment.*
- *JFC: Java Foundation Classes.*
- *J2EE: JAVA Platform Enterprise Edition.*
- *XML: Extensible Markup Language*
- *HTTP: Hyper Text Transfer Protocol.*
- *AWT: Abstract Window Toolkit.*
- *GIS: Geographic Information System.*
- *UdD: Universo del discurso.*
- *SGBD: Database Management System.*
- *DB2: Sistema de gestión de base de datos propiedad de IBM.*
- *IBM: International Business Machines.*
- *DEC: Digital Equipment Corporation*
- *ANSI: American National Standards Institute.*
- *ISO: Organización Internacional de Normalización.*
- *GPS: Global Positioning System.*
- *WGS84: World Geodetic System.*
- *OLE DB: Object Linking and Embedding for Databases.*
- *WKT: Well Known Text.*
- *WKB: Well Known Binary.*
- *GML: Geography Markup Language.*
- *ESRI: Enviromental Systems Research Institute.*
- *OGC: Open Geospatial Consortium.*
- *DDL: Lenguaje definición de datos o de descripción.*
- *DML: Lenguaje de Manipulación de Datos.*
- *SRID: Spatial Reference System Identifier.*

Referencias

[1] **Rogers Cadenhead, Laura Lemay. Programación *Java* 6.** Año 2008. Editorial ANAYA.

[2] **Francisco Javier Ceballos Sierra. Visual Basic.NET Lenguaje y Aplicaciones.** Año 2006. Editorial RA-MA. 2º Edición.

[3] **Michael K. Glass, Yann Le Scouarnec, Elizabeth Naramore, Gary Mailer, Jeremy Stolz, Jason Gerner. Fundamentos desarrollo de web con PHP, Apache y My*SQL*.** Año 2004. Editorial ANAYA.

[4] **Jose lopez Quijado. Domine PHP y My*SQL*, Programación dinámica en el lado del servidor.** Año 2006. Editorial RA-MA.

[5] **blog *Ante Todo…Mucha Calma.*** Disponible [Internet]: <http://elmeyer.blogspot.com/2009/03/visual-basic-vs-java.html> [19 de octubre de 2011].

[6] **Jose Rafael Garcia-Bermejo Giner. *JAVA* SE6 & *SWING*.** Año 2007. Editorial PEARSON Prentice Hall.

[8] **Francisco Javier Ceballos. *JAVA* 2, Interfaces Gráficas y aplicaciones para internet**. Año 2008. Editorial Ra-Ma. 3º Edición.

[9] **Ernesto Pimentel, Francisco duran, Francisco Gutiérrez. Programación orientada a objetos con *Java*.** Año 2007. Editorial THOMSON.

[10] **Ramez Elmasri, Shamkant B. Navathe. Fundamentos de Sistemas de Bases de Datos**. Año 2007. Editorial Addison Wesley. 5º Edición.

[11] **Raghu Ramakrishnan, Johannes Gehrke. Sistemas de Gestión de Bases de Datos**. Año 2006. Editorial Mc Graw Hill. 3º Edición.

[12] **Abraham Silberschatz. Fundamentos de diseño de Bases de Datos**. Año 2007. Editorial Mc Graw Hill. 5º Edición.

[13] **Wikipedia. base de datos espacial.** Disponible [Internet]: <http://es.wikipedia.org/wiki/Base_de_datos_espacial>[18 de Octubre de 2011]

[14] **Página Web de ESRI.** Disponible [Internet]: <http://www.esri.com/software/arcgis/geodatabase/index.html>[18 de Octubre de 2011].

[15] **Byron Heriberto Narváez Rodriguez. Año 2009. "ESTUDIO COMPARATIVO DE GEODATABASE APLICADO AL**

LEVANTAMIENTO DE LA LÍNEA BASE EN LAS COMUNIDADES DE LA COCIHC, FUNDACIÓN M.A.R.CO". Escuela Superior Politécnica de Chimborazo. Disponible [Internet]: <http://dspace.espoch.edu.ec/bitstream/123456789/59/1/18T00390.pdf>[18 de Octubre de 2011].

[16] **CAPITULO 3 Geodatabase.** Disponible [Internet]: <http://dspace.ups.edu.ec/bitstream/123456789/56/9/Capitulo3.pdf>[18 de Octubre de 2011].

[17] ***Oracle* Spatial. User`s Guide and Reference.** Disponible [Internet]: <http://download.*Oracle*.com/docs/cd/B10500_01/appdev.920/a96630/sdo_intro.htm>[18 de Octubre de 2011].

[18] **Wikipedia. Lenguaje de programación.** [http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n]. Fecha de revisión: 18 de Octubre de 2011.

[19] **Página Web AlegsaOnLine.com**. Disponible [Internet]: <http://www.alegsaonline.com/art/13.php>[18 de Octubre de 2011].

[20] **Wikipedia. Interfaz (Java)**. Disponible [Internet]: <http://es.wikipedia.org/wiki/Interfaz_%28*Java*%29>[18 de Octubre de 2011].

[21] **Wikipedia. Java (lenguaje de programación).** Disponible [Internet]: <http://es.wikipedia.org/wiki/*Java*_%28lenguaje_de_programaci%C3%B3n%29#Historia> [18 de Octubre de 2011].

Anexo I. Planificación

Para el éxito de cualquier proyecto es necesario una buena planificación, al igual que una correcta asignación de los recursos. Esto lleva a dividir el desarrollo del proyecto en etapas, donde cada una de ellas consiste en un conjunto de tareas similares que se encargan de realizar una pequeña parte del proyecto y donde todas esas partes juntas forman el proyecto entero.

El proyecto se ha dividido en cuatro etapas: *Planificación y Definición, Análisis y Diseño, Implementación* e *Integración y Pruebas*. La realización de cada estapa se realiza de manera secuencial, y la finalización de una es el comienzo de otra. Esto es debido en gran parte a que sólo existe un recurso humano encargado de realizar el proyecto y un solo recurso material, un ordenador.

Según transcurre el desarrollo de cada etapa, se realiza la debida documentación de cada etapa, para tener constancia de todo el trabajo realizado.

A continuación se explica en más detalle que tareas aborda cada etapa.

- ***Planificación y Definición***

  En esta etapa nace la idea básica en la que va a consistir el proyecto y se identifican las ideas y necesidades del usuario para definir y analizar las funcionalidades del sistema. Esto lleva a definir la arquitectura del sistema y los requisitos de la interfaz. En cuanto al software, se definen y desarrollan sus requisitos y se formulan las posibles aproximaciones a la solución del problema. Por último, una vez que se tiene definido el problema a resolver y como va ser el planteamiento de la solución, se hace una estimación del tiempo y esfuerzo de la realización del proyecto.

- ***Análisis y Diseño***

  Después de definir los requisitos principales que debe albergar el proyecto, se realiza el diseño detallado de la arquitectura del sistema, así como el diseño de la base de datos y de la interfaz.

- ***Implementación***

  Esta estapa se divide en dos tareas fundamentales: la programación del código fuente y las distintas pruebas individuales para que, posteriormente, se realicen las pruebas pertinentes de cada módulo implementado y su correspondiente verificación.

- ***Integración y Pruebas***

Esta etapa consiste en la integración de cada módulo implementado. Acto seguido es necesario verificar y validar esta integración. En el momento que la integración está finalizada y que se han realizado correctamente las pruebas, se procede a la aceptación del producto. Es en este momento cuando se realiza la documentación definitiva del proyecto.

Todo este proceso de etapas que se ejecutan de manera secuencial se observa en la figura 62, en la que también se muestra la asignación de recursos para cada una de ellas. Además, cada etapa tienen una duración determinada, aunque cabe destacar que la etapa más duradera es la etapa de implementación.

	Nombre de tarea	Duración	Comienzo	Fin
1	Hito de inicio	0 días	mié 01/09/10	mié 01/09/10
2	Etapa de Planificación y Definición	22 días	mié 01/09/10	jue 30/09/10
3	Etapa de Análisis y Diseño	75 días	lun 04/10/10	vie 14/01/11
4	Etapa de Implementación	99 días	jue 20/01/11	mar 07/06/11
5	Etapa de Integración y Pruebas	77 días	mar 14/06/11	mié 28/09/11
6	Hito de fin	0 días	mié 28/09/11	mié 28/09/11

Figura 62. Tareas de planificación.

En cuanto a la planificación definitiva representada con un diagrama de Gant (figura 63), el proyecto tiene inicio el 1 de septiembre del 2010 y finaliza el 28 de septiembre del 2011.

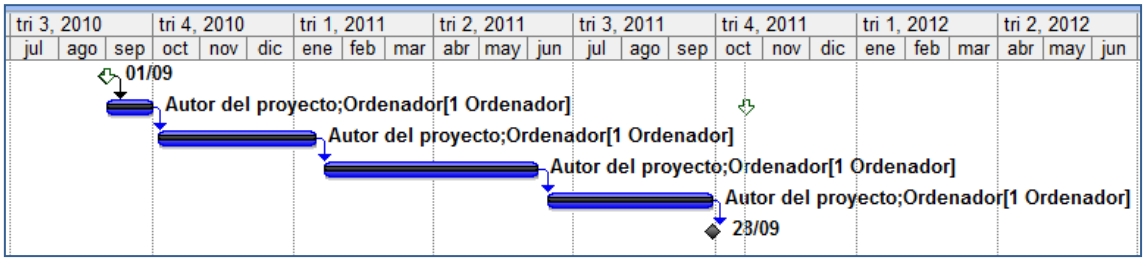


Figura 63. Diagrama de Gant.

Anexo II. Presupuesto

A continuación se presenta el presupuesto del presente proyecto que incluye el desglose de costes directos de personal, materiales y de costes indirectos, así como el coste total del proyecto.

1.- Autor:					
	Salazar Sepulveda, Katherin				
2.- Departamento:					
	Informática				
3.- Descripción del Proyecto:					
- Titulo	**Simulador de redes dinámicas almacenadas en bases de datos**				
- Duración (meses)	**13**				
Tasa de costes Indirectos:		**20%**			
4.- Presupuesto total del Proyecto (valores en Euros):					
54.197	Euros				
5.- Desglose presupuestario (costes directos)					
PERSONAL					
Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)
Salazar Sepulveda, Katherin		Analista y Programador	16,69	2.694,39	44.969,37
					0,00
					0,00
		Hombres mes	**16,69**	**Total**	44.969,37
[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)					
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)					

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador	900,00	100	13	60	195,00
					0,00
				Total	195,00

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
	Total	0,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	44.969
Amortización	195
Subcontratación de tareas	0
Costes de funcionamiento	0
Costes Indirectos	9.033
Total	54.197

El presupuesto total de este proyecto asciende a la cantidad de Cincuenta y Cuatro Mil Ciento Noventa y Siete EUROS.

Leganés a 18 de Octubre de 2011

El Ingeniero proyectista

Fdo. Katherin Salazar Sepúlveda.

Anexo III. Manual de usuario

El siguiente manual va dirigido al usuario de la aplicación, de manera que se explicará la funcionalidad asociada a cada uno de los elementos que forman parte de la interfaz final de la aplicación (figura 64) para que, aunque se ha intentado crear una interfaz gráfica sencilla e intuitiva para evitar que el usuario caiga en confusión al intentar realizar alguna de las funcionalidades, no haya ninguna duda en la forma de usarla.

Dicho esto, en el momento en que se ejecuta la aplicación aparecerá la interfaz gráfica de la figura 64. Es importante mencionar que no hay más ventanas, por lo que el aspecto de la interfaz gráfica no cambiará en ningún momento.

Se observa en el cuadrado de la izquierda de la figura 64 el panel donde se va a dibujar la red, y en el panel de la derecha se observan los botones y cuadros de texto necesarios para realizar las distintas funcionalidades explicadas más adelante.

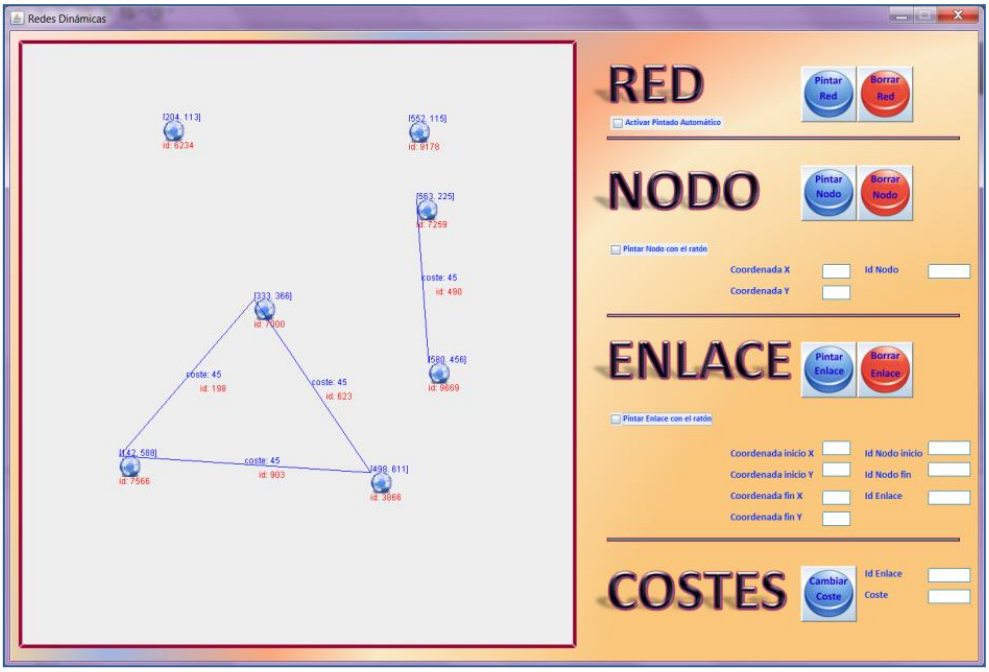


Figura 64. Interfaz gráfica de la aplicación.

Funcionalidades

A continuación se detalla cómo realizar cada funcionalidad con la interfaz gráfica:

- **Pintar la red**

  Para pintar la red que en ese momento esté almacenada en la base de datos, se dará clic sobre el botón *Pintar Red,* y automáticamente aparecerá pintada toda la información almacenada en la base de datos. Si no aparece nada, quiere decir que no hay información almacenada.

- **Borrar la red**

  Para borrar la red que en ese momento esté almacenada en la base de datos, y por tanto pintada en el panel de dibujo, se dará clic sobre el botón *Borrar Red*, y automáticamente aparecerá un cuadro de diálogo para confirmar con el usuario si desea de verdad borrar la red dibujada en el panel de dibujo. En caso afirmativo, la red desaparece y por tanto se borra de la base de datos. En caso negativo no se ejecuta ninguna acción.

- **Activar pintado automático.**

  Para activar el pintado automático de la red basta con activar el cuadro de selección *Activar Pintado Automático,* y para detener el pintado automático se desactiva el mismo cuadro de selección.

  En la figura 65 se aprecian los botones *Pintar Red* y *Borrar Red* y el cuadro de selección *Activar Pintado Automático.*

Figura 65. Sección Red.

- **Pintar un nodo.**

  - Con un clic de ratón sobre el panel de dibujo: para pintar con el ratón un nodo sobre el panel de dibujo basta con activar la casilla *Pintar Nodo con el Ratón*, de esta manera si se da clic en el panel automáticamente se pintará un nodo en esa posición.

  - Introduciendo coordenadas en los cuadros de texto: para pintar un nodo a partir de unas coordenadas basta con introducir las coordenadas del nodo en los cuadros de texto *Coordenada X* y *Coordenada Y* de la sección NODO y posteriormente dar clic al botón *Pintar Nodo*.

Automáticamente aparecerá en el panel de dibujo un nodo con esas coordenadas.

- **Borrar un nodo.**

    - Introduciendo coordenadas en los cuadros de texto: si se desea borrar un nodo mediante sus coordenadas se introducen en los cuadros de texto *Coordenada X* y *Coordenada Y* de la sección NODO las coordenadas correspondientes al nodo que se desea borrar y automáticamente desaparecerá el nodo y los enlaces a los que esté unido.

    - Introduciendo el identificador del nodo en el cuadro de texto: para borrar un nodo mediante su identificador basta con introducirlo en el cuadro de texto *id nodo* y posteriormente dar clic en el botón *Borrar Nodo.* Automáticamente desaparecerá el nodo que se desea borrar y los enlaces a los que esté unido.

En la figura 66 se aprecian los botones *Pintar Nodo* y *Borrar Nodo,* el cuadro de selección *Pintar Nodo con el ratón* y los cuadros de texto *Coordenada X, Coordenada Y* e *Id Nodo.*

Figura 66. Sección NODO.

- **Pintar un enlace.**

    - Introduciendo los identificadores de los nodos extremos: si se desea unir dos nodos mediante un enlace basta con introducir en los cuadros de texto *Id Nodo inicio* e *Id Nodo fin* de la sección ENLACE los identificadores de los nodos y posteriormente dar clic al botón *Pintar Enlace*. Automáticamente aparecerá pintado un enlace que una los dos nodos.

    - Introduciendo coordenadas de los nodos extremos: si se desea unir dos nodos mediante un enlace basta con introducir en los cuadros de texto *Coordenada inicio X, Coordenada inicio Y, Coordenada Fin X, Coordenada fin X* y *Coordenada fin Y* de la sección ENLACE las

coordenadas de los nodos y posteriormente dar clic al botón *Pintar Enlace*. Automáticamente aparecerá pintado un enlace que una los dos nodos.

- Mediante el dibujo del ratón: si se desea unir dos nodos dibujando el enlace con el ratón basta con activar primero el cuadro de selección *Pintar Enlace con ratón* y posteriormente seleccionar con el ratón el primer nodo e ir arrastrando el ratón por el panel hasta el otro nodo. En el momento que se suelte el ratón en el nodo fin aparecerá dibujado completamente el nuevo enlace.

- **Borrar un enlace.**

  - Mediante la introducción del identificador del enlace: para borrar un enlace mediante su identificador basta con introducirlo en el cuadro de texto *Id Enlace* de la sección ENLACE y posteriormente dar clic en el botón *Borrar Enlace*. Después aparecerán dos cuadros de dialogo, uno por cada nodo extremo, preguntando si desea borrar también sus nodos extremos. En caso afirmativo se borrará el nodo que haya confirmado, y en caso negativo no se borrará el nodo elegido pero siempre desaparecerá el enlace que en un principio se deseaba borrar.

  - Mediante la introducción de las coordenadas de los nodos extremos: si se desea borrar un enlace mediante sus coordenadas basta con introducir en los cuadros de texto *Coordenada inicio X, Coordenada inicio Y, Coordenada Fin X, Coordenada fin X* y *Coordenada fin Y* de la sección ENLACE las coordenadas de los nodos y posteriormente dar clic al botón *Borrar Enlace*. Igual que en el caso anterior, aparecerán dos cuadros de dialogo, uno por cada nodo extremo, preguntando si se desea borrar también sus nodos extremos. En caso afirmativo se borrará el nodo que se haya confirmado y en caso negativo no se borrará el nodo elegido, pero siempre desaparecerá el enlace que en un principio se deseaba borrar.

En la figura 67 se aprecian los botones *Pintar Enlace, Borrar Enlace,* los cuadros de texto *Coordenada inicio X, Coordenada inicio Y, Coordenada Fin X, Coordenada fin X, Coordenada fin Y, Id Nodo inicio, Id Nodo fin, Id Enlace* y el cuadro de selección *Pintar Enlace con el ratón.*

Figura 67. Sección Enlace.

- **Cambiar el coste de un enlace**

  Para cambiar el coste de un enlace simplemente se introduce el identificador del enlace en el cuadro de texto *Id enlace* y el nuevo coste en cuadro de texto *Coste*, ambos de la sección COSTE, y posteriormente se da clic sobre el botón *Cambiar Coste*. Automáticamente aparecerá el nuevo valor del coste en el enlace.

En la figura 68 se aprecia el botón *Cambiar Coste* y los cuadros de texto *Id Enlace* y *Coste.*

Figura 68. Sección Costes.