Universidad de la República
Facultad de Ingeniería

Instituto de Computación
Taller V año 2000

Interfaz Gráfica
Proyecto HTS

Tutores
Luis Sierra
Adrián Miranda

Emilio Cardoso
Gonzalo Penadés
Matías Bergengruen

Resumen

Un *grafo* se compone de un conjunto de *nodos* que representan objetos y un conjunto de *arcos* que representan relaciones entre dichos objetos. Los grafos se aplican en diversas áreas entre las que se cuentan:

- Economía e industria: Flujo de trabajo (Workflow), Organigramas, Diagramas de flujo
- Transporte: Redes de subterráneos, carreteras o trenes.
- Computación: Redes de computadoras, Diagramas entidad-relación, Redes de petri, Diagramas de clases

En estas áreas donde la información manejada se puede modelar usando grafos, también es aplicable una interfaz gráfica para permitir al usuario manipular dicha información en forma interactiva. En dichas interfaces suele presentarse la necesidad de ordenar automáticamente un grafo de manera que sea legible. El concepto *legible* depende de cada realidad que el grafo modele, en función de lo cual habrá criterios objetivos y también subjetivos que lo definan.

Este proyecto fue planteado por una empresa externa que procuraba una interfaz gráfica que maneje grafos y permita ordenarlos automáticamente. En este informe se describe el producto buscado, su proceso de desarrollo y el resultado obtenido. También se analiza e implementa un algoritmo para resolver el problema de ordenamiento automático en este caso puntual, descubriendo en el camino un problema mucho más general, que podría ser una nueva área de investigación en el campo del dibujado automático de grafos.

Tabla de contenido

Introducción

HTS Ltda. es una empresa dedicada al desarrollo e implementación de sistemas de gestión para procesos industriales. HTS concibió y desarrolló un conjunto de aplicaciones que componen el sistema denominado *Nous*. Los procesos industriales que Nous gestiona pueden ser modelados a través de diagramas de flujo de trabajo (Workflow). HTS estaba interesada en incluir en Nous una interfaz gráfica que compartiera con el resto del sistema la información de dichos diagramas, permitiendo además manipularla y modificarla visual e interactivamente. Otro requisito era que ordenara un diagrama automáticamente de manera que resultara legible. HTS buscó una solución de este tipo en el mercado sin encontrar nada que cumpliera con sus requisitos específicos, por lo que planteó este proyecto externo.

Los diagramas de flujo de trabajo son un ejemplo de *grafos*. Un grafo se compone de un conjunto *nodos* que representan objetos (reales o abstractos) y un conjunto de *arcos* que representan relaciones entre dichos objetos. Los grafos se aplican en diversas áreas entre las que se cuentan:

- Transporte y comunicaciones: Redes telefónicas, Redes de subterráneos, carreteras o trenes.
- Computación: Redes de computadoras, Diagramas entidad-relación, Redes de petri, Diagramas de clases
- Electrónica: Diseño de Circuitos eléctricos
- Economía e industria: Flujo de trabajo (Workflow), Organigramas, Diagramas de flujo
- Química: Moléculas o compuestos químicos

En estas áreas donde la información manejada se puede modelar usando grafos, también es aplicable una interfaz gráfica como la que pretende HTS, especialmente en lo que se refiere a la ordenación automática de grafos. El problema de ordenar automáticamente un grafo de manera que sea legible es tan complejo como lo es el concepto *legible*, ya que para cada realidad que el grafo modele habrá criterios objetivos y también subjetivos que definan dicho concepto.

Resolver el problema puntual de HTS y al mismo tiempo incursionar en el problema del ordenamiento automático de grafos - con la investigación que esto implica - fueron los factores decisivos para que eligiéramos este proyecto.

En el Capítulo 1 hablaremos de la interfaz gráfica, sus requerimientos, el análisis y diseño realizado y el producto final obtenido. En el Capítulo 2 hablaremos del algoritmo de ordenamiento automático de grafos pedido por HTS. Este algoritmo es una parte del producto que debido a su importancia se presenta en forma separada. En el Capítulo 3 hablaremos de este proyecto, la forma en que se desarrolló y las metodologías aplicadas.

Capítulo 1 - El producto

HTS Ltda. necesita un producto de software que cumpla con ciertos requerimientos. Este capítulo detalla los objetivos del proyecto relativos al producto, los requerimientos del software y los aspectos más relevantes del análisis, diseño e implementación. Finalmente se presenta el producto obtenido y las mejoras que se le podrían realizar a futuro.

1.1 Objetivos

El objetivo de HTS es obtener un software que satisfaga sus requerimientos específicos y que a su vez pueda seguir evolucionando con el tiempo. En las primeras reuniones de definición y diseño del producto, nos propusimos generar:

- ?? un software que cumpliera con los objetivos de HTS
- ?? un software aplicable a otros usos, o al menos un buen punto de partida fácilmente modificable.
- ?? un código fuente entendible, mantenible y de buena calidad.
- ?? documentación clara y completa tanto del software como del proceso de desarrollo.

1.2 Requerimientos

Nous (el sistema de HTS) gestiona procesos industriales que pueden ser modelados a través de diagramas de flujo de trabajo (Workflow). Un ejemplo de esto es la cadena de producción de una mueblería compuesta por dos líneas de producción diferentes (Figura 1). Cada línea de producción es una secuencia de etapas que llevan desde las piezas de madera de un mueble hasta el producto pronto para entregar. El diagrama completo es un *grafo*, las etapas son sus *nodos* y las flechas que indican el flujo de trabajo entre ellas son sus *arcos*. Un mueble pasa por una línea de producción o la otra, según si su terminación es con barnizado o únicamente pintura.

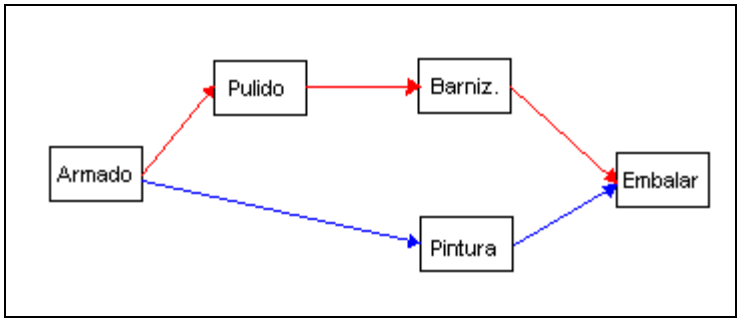


Figura 1 – Ejemplo de proceso industrial: Mueblería

El software que HTS necesita es una interfaz gráfica para el entorno Windows, que comparta con Nous la información de los grafos correspondientes a ciertos procesos industriales. Dicha información incluirá un conjunto de propiedades asociadas a cada nodo o arco. La cantidad de nodos y arcos del grafo no está limitada (normalmente será de decenas o pocos cientos) y podrá haber *circuitos* (esto es, cuando desde un nodo se puede recorrer una secuencia de arcos llegando a él mismo).

✍✍*En adelante nos referiremos al software para HTS como* Interfaz Gráfica *(IG).*

1.2.1 Casos de uso

HTS pretende utilizar la IG de dos formas que se detallan a continuación.

Instalación del sistema Nous en una empresa: Edición

Para este caso de uso la IG presentará la definición de un grafo incluyendo propiedades de sus nodos y arcos; permitirá modificar dicha información gráficamente, ocultar y mostrar nodos o arcos de un tipo dado y ordenar automáticamente el grafo cuando el usuario lo desee.

Presentación gráfica de información de Nous: Visualización

En este caso se visualizará un grafo ordenado automáticamente, permitiendo mover nodos y arcos para mejorar dicho ordenamiento. La definición del grafo, incluyendo propiedades de nodos y arcos, no podrá ser modificada. Existirá la posibilidad de ejecutar aplicaciones actuando sobre los nodos.

1.2.2 Interacción e información compartida con Nous

La IG deberá poder ser iniciada por Nous a través de una línea de comando del sistema operativo Windows. El esquema de interacción será el de la Figura 2.

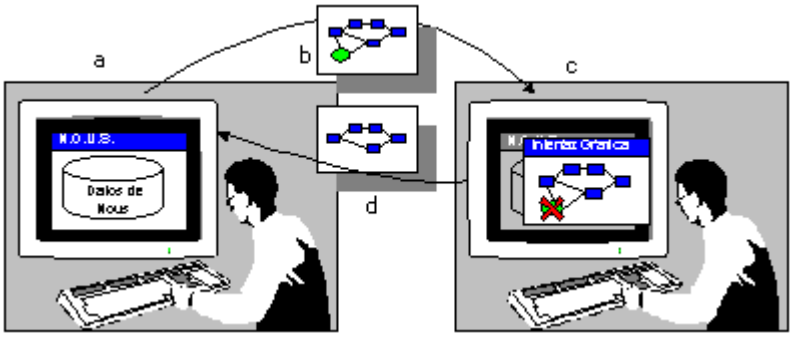


Figura 2 – Esquema de interacción entre la IG y Nous

a) el usuario trabaja sobre la base de datos de Nous.
b) se pasa el control a la IG, pasándole un archivo definición de grafo.
c) el usuario elimina un Nodo y luego cierra la IG.
d) el control vuelve a Nous, incorporando los cambios realizados.

La información compartida pasa de Nous a la IG al iniciarla y no será modificada por los sistemas de HTS durante la ejecución de la misma. Cuando se cierra la IG, el control sobre la información compartida vuelve a Nous.

Los nodos y arcos de un grafo de la IG tendrán un tipo asociado, que define la forma de representarlos gráficamente y permite clasificarlos y asociarles características compartidas. La cantidad de tipos de nodos y arcos no está limitada, pero normalmente será de pocas decenas. La *información de configuración* de la IG incluye la definición de dichos tipos y lo que denominamos *relaciones válidas*, que determinan si un arco de un tipo A puede ir desde un nodo de tipo B hasta otro de tipo C. Esta información, junto con las definiciones de grafos, será compartida entre la IG y Nous.

1.2.3 Interacción del usuario

Se podrá editar y visualizar más de un grafo a la vez, aunque éstos utilicen distinta configuración. Un mismo grafo se podrá visualizar y manipular en distintas escalas de tamaño simultáneamente. Las etiquetas de arcos y nodos, así como las de los tipos, deberán ser visibles. Se podrá ocultar o mostrar todos los nodos o arcos de determinados tipos y ver en detalle las propiedades de un nodo o arco dado.

El usuario podrá acceder a las funcionalidades ofrecidas por la IG a través de comandos de menúes, o usando el ratón directamente sobre los distintos objetos representados. Los comandos de uso más frecuente tendrán asociados aceleradores de teclado o botones en una barra de botones. Habrá *menúes contextuales*, cuyos comandos se relacionan con los objetos seleccionados.

Para el caso de uso de Edición, se podrá utilizar el ratón para insertar, seleccionar, duplicar, eliminar y mover nodos y arcos. También se podrá modificar un arco dado moviendo su etiqueta o agregando, eliminando y moviendo codos. En este caso de uso se podrá además modificar la información de un nodo o un arco dado modificando sus propiedades. En el caso de Visualización sólo estarán disponibles las funcionalidades que no modifiquen la información compartida entre Nous y la IG. En particular se podrá ejecutar una aplicación asociada a un nodo, haciendo doble clic sobre el mismo.

1.2.4 Ordenamiento automático

Los grafos se representarán en dos dimensiones. La IG dispondrá de la funcionalidad de ordenar automáticamente un grafo, reubicando los nodos y arcos en el plano de manera que el dibujo final del grafo sea *legible* (fácil de entender, fácil de recordar y esclarecedor de la realidad que representa). Debido a la complejidad e importancia de este tema, sus requerimientos, análisis y demás aspectos se tratarán en el Capítulo 2 - El Algoritmo.

1.3 Análisis y diseño

En el proceso de desarrollo de este proyecto realizamos un análisis y diseño orientado a objetos. El mismo permite "modelar sistemas de software como colecciones de objetos que cooperan entre sí, tratando objetos individuales como instancias de una clase dentro de una jerarquía de clases" [Booch-94].

Para el diseño de las clases utilizamos Visual Modeler 2.0, la herramienta gráfica de modelado de objetos incluida en el Visual Developer Studio 6.0. Visual Modeler utiliza un subconjunto de los diagramas que define el lenguaje de modelado unificado (UML), permitiendo rápidamente modelar las clases a implementar.

1.3.1 Casos de uso

Nos planteamos dos formas de atacar los dos casos de uso: generando un módulo separado para cada uno, o generando un solo módulo para ambos según ciertos parámetros o configuración.

La principal ventaja de la primer opción es la seguridad, dado que permite distribuir cada módulo con la funcionalidad estrictamente necesaria. Otra ventaja es que un cambio en el código exclusivo de un módulo no impacta en el otro.

La ventaja de la segunda opción es que el código a mantener es uno solo, y también lo es el ejecutable a verificar y distribuir.

Optamos por la segunda opción por dos razones fundamentales: la mayor parte de la funcionalidad es común a ambos módulos y es posible lograr que, a los ojos de los usuarios, no se distinga entre un solo módulo o dos independientes.

1.3.2 Interacción e información compartida con Nous

1.3.2.1 Inicio de la IG

Los requerimientos indican que la IG será iniciada a través de una línea de comandos con parámetros, por lo que decidimos hacerla como un programa ejecutable de Windows que pueda ser iniciado en forma independiente, con o sin parámetros.

1.3.2.2 Intercambio de información

La información compartida no puede ser modificada por la IG y Nous simultáneamente. Decidimos usar archivos de texto con un formato simple, por ser una solución sencilla de implementar y de utilizar que cumple con los requerimientos; tanto para nosotros como para HTS resultó muy práctico poder modificar su contenido con un editor de texto. Se usará un archivo auxiliar para indicar que la IG está corriendo, por ser la manera más fácil para Nous de determinar cuando recuperar el control sobre la información compartida.

Evaluamos otras variantes de interacción de aplicaciones cliente–servidor e intercambio de datos manejadas por Windows. Estas opciones permiten ocultar la independencia de un programa ejecutándolo dentro de una ventana perteneciente al programa; permiten el intercambio de datos en memoria y usan mejores técnicas de pasaje del control de una a otra. Descartamos estas variantes por ser más complejas que la opción elegida.

1.3.2.3 Archivos manejados por la IG

La definición de cada grafo se guardará en un archivo independiente cuyo nombre identificará al grafo definido. La información de ubicación de los nodos y arcos de un grafo se guardará en otro archivo de igual nombre pero distinta extensión. La información de configuración se guardará en un archivo de nombre fijo, ubicado en el directorio de trabajo de la IG. Este archivo se leerá una única vez al iniciar la IG, ya que además de definir los tipos y las relaciones válidas, indicará el modo de ejecución de la IG: Editor o Visualizador. En el primer modo, la IG permitirá manejar varios archivos de definición de grafos simultáneamente. En el segundo sólo manejará un archivo, que debe haber recibido como parámetro; al cerrarse ese archivo se cerrará la IG. Una descripción detallada de dichos archivos se puede ver en los documentos Archivo de configuración y Archivo de Definición de Grafo que forman parte de la documentación del producto.

1.3.2.4 Propiedades fijas vs. dinámicas

Las propiedades disponibles para los nodos y arcos serán fijas. Los arcos tendrán: identificador, etiqueta, nodo origen y nodo destino. Las principales propiedades de los nodos serán: identificador, etiqueta y nivel. Evaluamos la posibilidad de implementar propiedades dinámicas para aumentar la generalidad, definiendo un objeto Propiedad con un nombre, un tipo de dato y un valor. Descartamos esta opción porque agregaba complejidad a la implementación y quitaba amigabilidad frente al usuario al momento de manejar dichas propiedades.

1.3.3 Interacción del usuario con la IG

1.3.3.1 Múltiples documentos y múltiples vistas

Para visualizar simultáneamente un grafo en distintas escalas definimos una *interfaz de múltiples documentos* (MDI) en la que cada documento manejado – cada grafo - se presenta en una o más *ventanas*. Una ventana en la cual se representa un grafo será lo que denominamos una *vista* del grafo. Cada una de ellas tendrá funcionalidad de *zoom* y *barras de desplazamiento* independientes, así como la posibilidad de ocultar o mostrar los nodos o arcos de tipos dados.

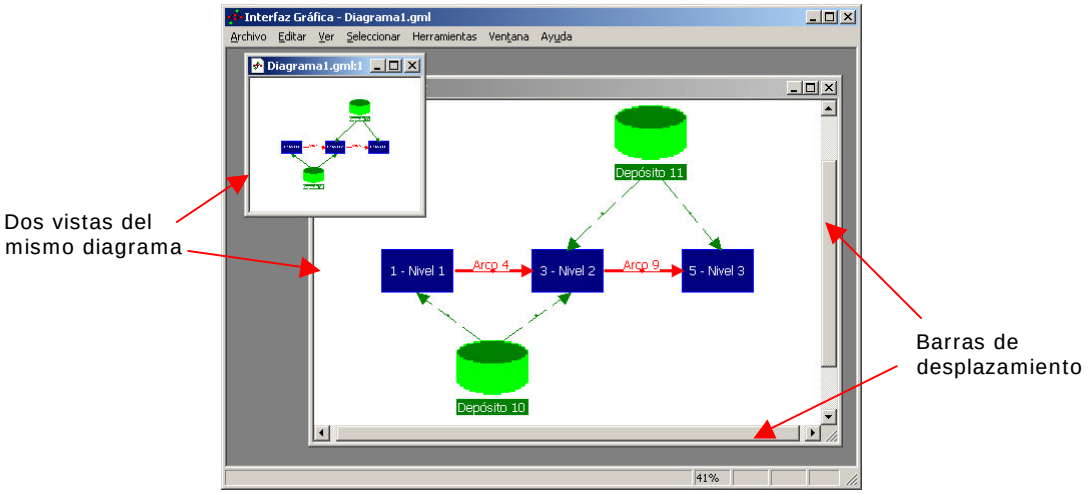


Figura 3 – Dos vistas del mismo grafo en distinta escala

El uso del archivo de configuración desde el directorio de trabajo permitirá que se ejecuten dos *instancias* de la IG con distintas configuraciones, cumpliendo el requerimiento de poder visualizar a la vez grafos de distinta configuración.

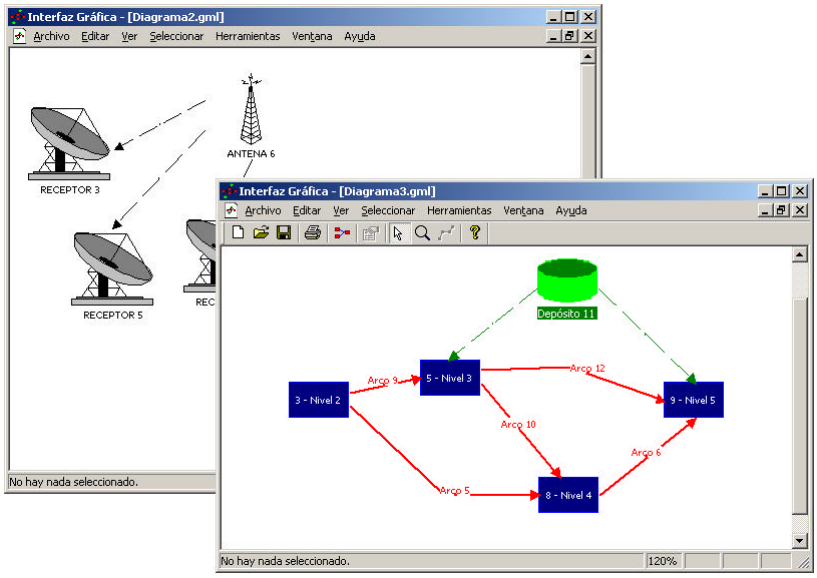


Figura 4 – Dos IG con distintas configuraciones

1.3.3.2 Visualización de propiedades y etiquetas

Habrá dos ventanas flotantes de propiedades que permitirán visualizar o editar (según el caso de uso) las propiedades de un nodo y de un arco a la vez. Las etiquetas de los arcos se mostrarán completamente en el diagrama, mientras que las de los nodos quedarán restringidas por el tamaño del nodo definido en el tipo de nodo. La etiqueta podrá incluir caracteres especiales que hagan referencia a las demás propiedades, de manera que en el dibujo del grafo se arme la etiqueta a mostrar con los valores actuales de las propiedades.

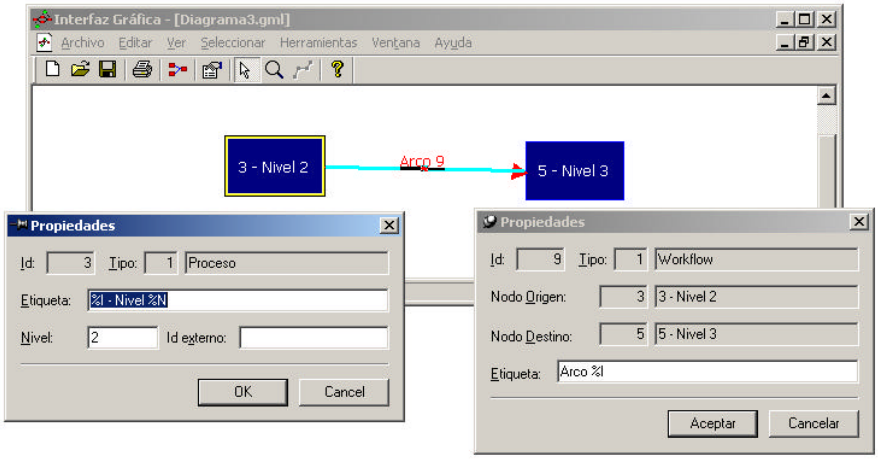


Figura 5 – Propiedades de nodo y arco

1.3.3.3 Programación basada en eventos

En la programación de aplicaciones para entornos gráficos, las acciones del usuario (comandos de menú, clics o movimiento del ratón) se denominan *eventos*. El programador crea funciones que son llamadas por el sistema cuando ocurre un determinado evento y realizan la funcionalidad asociada a dicho evento. Como nuestro diseño es además orientado a objetos, el responsable de realizar la funcionalidad esperada por el usuario es el objeto al cual concierne el evento. Por ejemplo, definimos un objeto Documento que contiene un grafo e implementa la funcionalidad de almacenamiento de los archivos asociados al grafo; cuando el usuario presiona el botón "Guardar", el evento es derivado al objeto Documento que se ocupa de guardar la información del grafo en el disco.

1.3.3.4 Eventos del ratón y modos de trabajo

Los eventos generados por el ratón deben ser manejados de diferente manera dependiendo del *modo de trabajo* (estado) en que se encuentre la IG. Definimos los siguientes modos de trabajo:

- Selección - Es el modo por defecto. Permite seleccionar, mover, borrar y duplicar objetos y también permite usar el menú contextual.
- Zoom – Permite controlar la funcionalidad de zoom mediante el ratón.
- Nuevo nodo – Permite agregar un nodo de un tipo dado en el lugar donde se haga clic con el ratón.

?? Nuevo arco – Permite agregar un arco de un tipo dado, seleccionando el nodo origen y el nodo destino. Controla además que se respeten las relaciones válidas.

?? Edición de arcos - Es el modo que permite editar individualmente el arco seleccionado, moviendo, agregando o quitando codos, o moviendo la etiqueta.

Si el manejo de todos los modos de trabajo se realiza en un solo punto - una única función de respuesta a un evento - el código de la aplicación se torna confuso, difícil de verificar, corregir y mantener. Para evitar estos problemas usamos un enfoque basado en tener objetos diferentes llamados *herramientas* (Tools) para manejar cada modo de trabajo. De esta manera, todos los atributos y funciones que manejan un modo en particular quedan encapsulados dentro de una sola clase. De acuerdo a este enfoque diseñamos una clase genérica (`ToolBase`), para luego derivar clases específicas (`ToolSelect`, `ToolZoom`, etc.) que atienden los eventos del ratón para cada uno de los modos de trabajo.

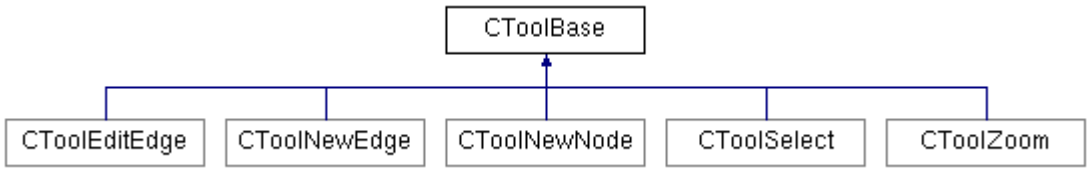


Figura 6 – Herramientas para implementar los modos de trabajo

```
////////////////////////////////////////////////////////////////////
// CLASE:
//    CToolBase
// DESCRIPCION:
//    Clase base de todos los Tools que manejan los eventos de mouse.
//
class CToolBase
{
public:
...
// EVENTOS
   virtual BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message) { return TRUE; }
   virtual void OnMouseMove(UINT nFlags, CPoint point) {}
   virtual void OnLButtonDblClk(UINT nFlags, CPoint point) {}
   virtual void OnLButtonDown(UINT nFlags, CPoint point) {}
   virtual void OnLButtonUp(UINT nFlags, CPoint point) {}
   virtual void OnRButtonDown(UINT nFlags, CPoint point) {}
   virtual void OnRButtonUp(UINT nFlags, CPoint point) {}
...
```

Figura 7 – Extracto de la definición de la clase abstracta CToolBase

A modo de ejemplo, la funcionalidad de seleccionar un objeto del diagrama mediante el ratón está asociada al evento de clic en el modo de trabajo de selección. Cuando este evento ocurre, el método de la clase `CToolSelect` asociado a él es llamado pasándosele la información del punto de la ventana en el cual se hizo clic. La implementación de la funcionalidad de seleccionar implica los siguientes pasos:

1. determinar el objeto sobre el cual se hizo clic
2. hacer que los objetos seleccionados dejen de estarlo
3. seleccionar el objeto del paso 1

1.3.4 Objetos gráficos

En nuestro diseño cada nodo o arco del grafo será lo que llamamos un *objeto gráfico*, un objeto capaz de dibujarse a sí mismo y que conoce su ubicación y tamaño, por lo que puede determinar si un clic en la pantalla lo toca o no.

Los objetos gráficos (GO) derivan de una clase `GOBase`, que declara las funciones principales para el manejo de los objetos a dibujar. Algunas de ellas se muestran en la Figura 8.

```
class GOBase
{
public:
...
// OPERACIONES
   // Devuelve verdadero si el punto esta dentro de la figura.
   virtual bool         HitTest(const CPoint&) const = 0;

   // Devuelve el objeto si es sensible a ser tocado y esta bajo el punto.
   virtual GOBase *     GetHitSensitiveObject(const CPoint& pto);

   // Prepara el dibujado seleccionando los objetos GDI y llama a DoPaint.
   virtual void         Paint(CDC *pDC) const;

   // Dibuja la figura en el hdc dado.
   virtual void         DoPaint(CDC *pDC) const = 0 ;

   // Devuelve verdadero si la figura toca la region dada
   virtual bool         Touches(const CRgn&) const = 0;

   // Le aplica a la figura la transformacion t.
   virtual void         Transform(const CMatrix& t)=0;

   // Dibuja el contorno de la figura.
   virtual void         DrawOutline(CDC *pDC, const CMatrix & t) const {}

// ACCESO
   // Devuelve el area que tengo que invalidar para poder redibujar el objeto
   virtual CRect     GetInvalidArea() const;

   // Muestra u oculta el objeto
   void              SetVisible(bool bVisible=true) { m_bVisible  = bVisible;}

   // Devuelve verdadero si el objeto esta visible
   bool              IsVisible() const { return m_bVisible; }

   // Devuelve verdadero si el objeto es sensible al tacto.
   virtual bool      IsSensitive() const { return m_bSensitive; }
...
};
```

Figura 8 – Extracto de la definición de la clase abstracta `GOBase`

De la clase `GOBase` se derivan clases elementales especializadas en dibujar la figura que identifican (`GORectangle`, `GOLine`, etc.) y la clase `GOGroup` que es una agrupación de dichas clases elementales (ver Figura 9).

La clase `IG_Grafo`, que representa un grafo en la IG, deriva de `GOBase` para heredar la funcionalidad de objeto gráfico. Es análogo el caso de la clase `IG_Seleccion` que mantiene la información de nodos y arcos seleccionados.

Finalmente, de `GOGroup` derivan `IG_Arco` e `IG_Nodo`, lo cual es imprescindible para lograr la configuración dinámica, como se verá en el siguiente punto.

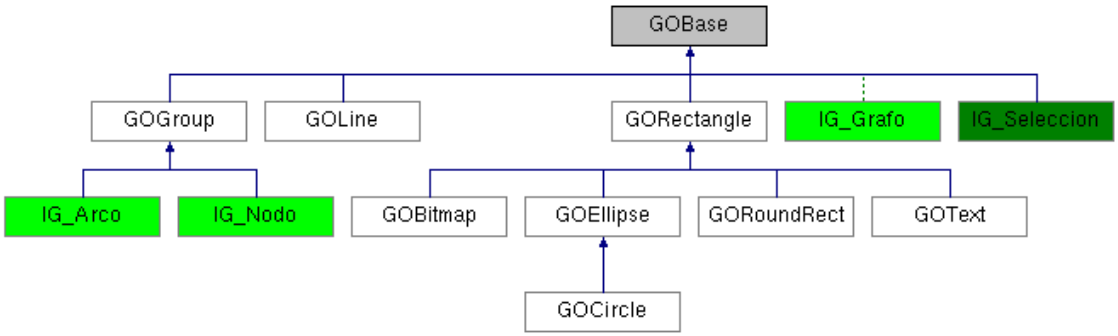


Figura 9 – Diagrama de herencias de GOBase

1.3.5 Configuración dinámica

Uno de los puntos fuertes de los requerimientos de generalidad de la IG era la capacidad de manejar distintas configuraciones. Esto implicaba poder manejar internamente las distintas variantes de definiciones de tipos (de nodos o arcos), y representar conjuntos de dichos tipos de tamaño variable. Por otra parte, las funcionalidades asociadas a los tipos, como visualizar la lista de tipos disponibles, insertar un nodo o arco de un tipo dado y ocultar los nodos y arcos de determinados tipos, debían poder ajustarse a cada configuración automáticamente. Finalmente, el tipo de un nodo (o arco) definía la forma en que éste se dibujaba, por lo que era necesario un mecanismo para que el tipo de nodo le informara al nodo como dibujarse. Para resolver todo esto diseñamos la siguiente jerarquía de clases:

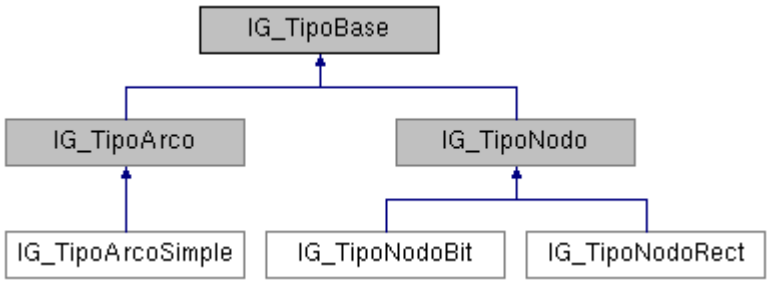


Figura 10 – Diagrama de clases de la configuración

La clase IG_TipoBase es responsable de los atributos comunes a todos los tipos: la etiqueta, el nombre de menú y el dibujo que la representa. La clase IG_TipoNodo (y análogamente IG_TipoArco) es la base de todos los tipos de nodo disponibles. Su principal utilidad está en su función miembro:

```
ConstruirGOGroup(GOGroup * pGrupo, CPoint ptApoyo, string sEtiqueta);
```

Esta función es llamada desde el nodo (que se pasa a sí mismo como el parámetro pGrupo) cada vez que necesita saber como dibujarse. Las clases del tercer nivel son las que efectivamente implementan dicha función como corresponda, creando los objetos gráficos necesarios para representar el nodo (o arco) como están definidos. Por ejemplo, el tipo de nodo rectangular crea un GORectangle y un GOTexto con el texto de la etiqueta, y los agrega al IG_Nodo.

Al iniciar la IG se lee del archivo de configuración la definición de cada tipo, se crea un objeto de la clase correspondiente, se agrega a la estructura de datos que maneja toda la configuración y se genera automáticamente los menúes y barras de botones con leyenda. La Figura 11 muestra dos instancias de la IG con distintas configuraciones.

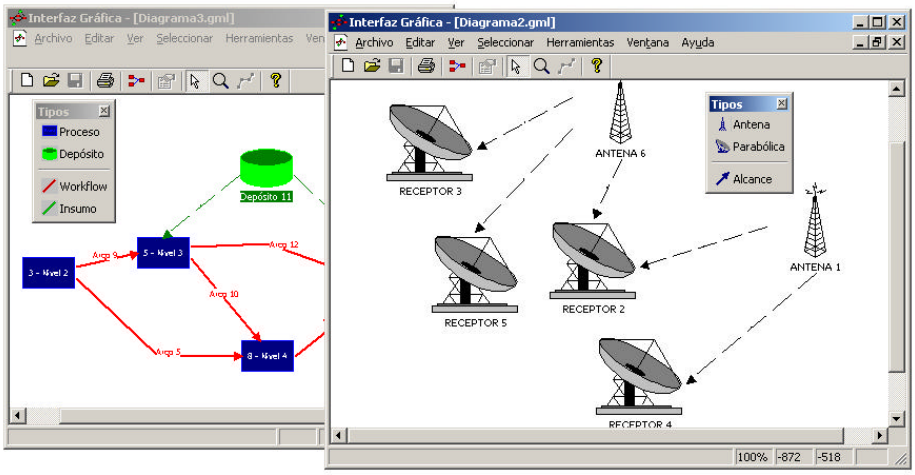


Figura 11 – Captura de pantallas con configuraciones diferentes

1.3.6 El grafo en la IG y GTL

Como se vio en el punto 1.3.4, diseñamos las clases `IG_Grafo`, `IG_Nodo` e `IG_Arco`, para mantener la información en memoria de cada grafo manejado por la IG. En el diseño de estas clases veíamos muchas facetas complicadas en lo que se refiere a mantener consistente la información de relaciones entre nodos y arcos del grafo al insertar y eliminar algunos de ellos. Esto cambió cuando encontramos en Internet la Graph Template Library (GTL), que se describe más adelante en el punto 1.4.2.3. Utilizando GTL pudimos derivar de su clase `graph` nuestro `IG_Grafo` y manejar toda la información y funcionalidad que queríamos asociar a los nodos y arcos gracias a los mapas de nodos y arcos que soporta, como se puede ver en la Figura 12.

```
class IG_Grafo : public graph, protected GOBase
{
public:
   IG_Nodo *   GetNodo(node n) const { return m_Mapa_IGNodos[n]; }
   IG_Arco *   GetArco(edge e) const { return m_Mapa_IGArcos[e]; }

   IG_Nodo  *  NewNodo(int nTipo, CPoint ptApoyo);
   IG_Arco  *  NewArco(int nTipo, node n_org, node n_dest);

   bool        DeleteNodo(IG_Nodo * pNodo);              // Borra un nodo del grafo.
   bool        DeleteArco(IG_Arco * pIGArco);             // Borra un arco del grafo.

protected:
   void        post_new_node_handler(node n);
   void        pre_del_node_handler(node n);
   void        post_new_edge_handler(edge e);
   void        pre_del_edge_handler(edge e);

private:
   node_map<IG_Nodo*>   m_Mapa_IGNodos;         // Nuestra info de Nodo asociado
   edge_map<IG_Arco*>   m_Mapa_IGArcos;         // Nuestra info de Arco asociado
};
```

Figura 12 – Extracto de la definición de la clase `IG_Grafo`

1.3.7 Ordenamiento automático

El ordenamiento automático de un grafo es, desde el punto de vista de un programador, un algoritmo que recibe un grafo arbitrario, que quizás deba cumplir ciertas restricciones o contener información complementaria. El algoritmo realiza una serie de pasos o etapas al cabo de las cuales determina una posición (par de coordenadas en el plano) de cada nodo y una curva definida para cada arco. GTL incluye una clase base abstracta `algorithm` muy sencilla pero a la vez potente, que permite derivar de ella cualquier algoritmo aplicable a un grafo.

```
class algorithm
{
public:
    algorithm () { };
    virtual ~algorithm () { };

   // Applies algorithm to graph g.
   virtual int run (graph& g) = 0;

   // Checks whether all preconditions are satisfied.
   virtual int check (graph& g) = 0;

   // Resets algorithm, i.e. prepares the algorithm to be applied to another graph.
   virtual void reset () = 0;
};
```

Figura 13 – Definición completa de la clase algorithm

La potencia de `algorithm` está en que permite independizar al máximo el algoritmo del resto del software. Para darle más generalidad a la IG, analizamos la posibilidad de manejar diferentes algoritmos, programados en bibliotecas de enlace dinámico (DLL), y referenciados en tiempo de ejecución, en función de una configuración, que se leyera en forma análoga al funcionamiento de los tipos. Descartamos esta idea, por no ser necesario más de un algoritmo para este caso y no ser trivial su desarrollo. El análisis del ordenamiento automático requerido por HTS en particular se detallará en el Capítulo 2 - El Algoritmo.

1.4 Implementación

1.4.1 Lenguaje de programación

A la hora de elegir el lenguaje de programación, evaluamos algunos de ellos junto con bibliotecas y herramientas de desarrollo disponibles en el mercado. Las características consideradas fueron: orientación a objetos, facilidad de uso, madurez y continuidad de la herramienta, disponibilidad de información y código fuente, potencia para usar estructuras complejas y dinámicas en memoria, potencia para usar la interfaz gráfica de usuario (*GUI*) nativa de cada sistema operativo, portabilidad entre las plataformas más conocidas (Windows 9X, Windows NT, Windows 2000, Unix) y experiencia de los programadores.

Como resultado de la evaluación elegimos la opción recomendada por HTS: Visual C++, principalmente por su potencia para usar la GUI del sistema operativo Windows – que es el único requerido por HTS - y por la

experiencia que tenemos en este lenguaje y herramienta en particular. Más detalles del análisis realizado y sus conclusiones se pueden ver en la Evaluación de lenguajes de programación que integra la documentación del proyecto.

1.4.2 Bibliotecas utilizadas

En nuestra implementación tratamos de utilizar, en la medida que fuera posible, bibliotecas estándar portables. A continuación mencionamos tres bibliotecas que fueron fundamentales en nuestro desarrollo.

1.4.2.1 Microsoft Foundation Classes (MFC)

Son clases que encapsulan la interfaz para programación de aplicaciones (API) de Windows y proveen otras facilidades de desarrollo, como el modelo vista-documento y el direccionamiento automático de eventos. Se incluyen con el Visual C++, son de uso libre y portables a otras plataformas.

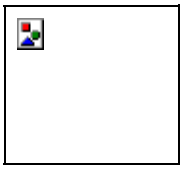
1.4.2.2 Standard Template Library (STL)

Es una biblioteca estándar de templates, especializada en contenedores y clases útiles, como strings y algoritmos aplicables a los contenedores. Se incluye en la mayoría de los compiladores C++ modernos (incluyendo Visual C++), es de uso libre, e independiente de la plataforma.

1.4.2.3 Graph Template Library (GTL)

Es una biblioteca de clases base desarrollada para un proyecto de la Universidad de Passau (Alemania) relacionado con grafos. Define una estructura flexible de soporte para grafos inspirada en y compatible con STL. Uilizamos la versión 0.3.3 que se encuentra disponible en Internet (http://infosun.fmi.uni-passau.de/GTL), junto con documentación detallada. Su uso NO comercial es libre y su uso comercial requiere un permiso por escrito de la Universidad de Passau, a quien pertenece el Copyright. Es independiente de la plataforma.

1.4.3 Reutilización de código de terceros: CodeGuru



CodeGuru (http://www.codeguru.com/) es un sitio de Internet dedicado a ayudar a los programadores de Visual C++/MFC (entre otros), proporcionando la información técnica más robusta y actual junto con fragmentos de código fuente e incluso programas demostrativos. El material suministrado en Codeguru es de libre utilización.

En este sitio obtuvimos código fuente e información detallada sobre el mismo que nos fue muy útil, especialmente con respecto al aprovechamiento de la potencia de interfaz gráfica de usuario (GUI) de MFC. Los artículos utilizados son los siguientes (Título – Autor):

Add Zoom and Scale Capabilities to CScrollView - Brad Pirtle

Describe una clase para el manejo del zoom, que tomamos como base para implementarlo en las vistas de grafos de la IG.

Drawing a bitmap from a BMP file - Anónimo

Incluye dos funciones para la lectura de archivos de mapas de bits (BMP), y su dibujado en la pantalla: `LoadBMP` y `DrawDIB`, que usamos para los nodos de tipo bitmap.

A quick method to load a bitmap file in a CBitmap - Ronen Magid

Sencilla pero potente extensión de la clase `CBitmap` de MFC, usada en nuestra clase CMiBitmap para cargar los botones de la barra de tipos dinámica.

A Pinnable Dialog Base Class - David Hill

Implementa una clase derivada de CDialog de MFC, que nos permitió la creación de los cuadros de diálogo "pinchables" usados para las propiedades de nodos y arcos. Ver Figura 5 – Propiedades de nodo.

Merging Two Menus - Oskar Wieland

Define una función que combina dos menúes, utilizada para la creación de los menúes contextuales.

1.4.4 Estándar de codificación y recomendaciones

Para generar un código fuente uniforme, claro y fácil de mantener por cualquiera de los desarrolladores, se definió un conjunto de estándares y recomendaciones de codificación. Estos se encuentran detallados en la documentación del proyecto en Guías de Codificación C y C++.

Entre las recomendaciones se destacan:

- nomenclatura de identificadores uniforme (prefijos, mayúsculas)
- indentación y uso de tabuladores
- uso de comentarios de archivos, de clases y de funciones
- uso de headers precompilados para acelerar la compilación
- ocultamiento de información (clases de interfaz mínima)
- antes legibilidad y claridad que soluciones "óptimas" o "inteligentes"

1.4.5 Mensajes y registro de errores

La IG guarda un registro de los errores que detecta, presentando al usuario el mensaje de error correspondiente. Este registro es un archivo de texto cuyo contenido está en lenguaje natural y resulta fácil de leer.

Además de mensajes de error, la IG presenta mensajes de advertencia o confirmación de comandos. Evaluamos la posibilidad de hacer que todos los mensajes fueran configurables, y que el usuario pudiera decidir si quiere que le aparezcan o no, con el estilo de "No preguntar de nuevo", pero quedó como mejora a futuro.

1.4.6 Prototipos

A lo largo de la implementación generamos prototipos, incorporando gradualmente nuevas funcionalidades. Cada prototipo se envió a HTS para su revisión, se utilizó para verificar y validar la funcionalidad implementada y precisar o ampliar la faltante.

1.5 El producto final

Nuestro producto final se compone del ejecutable, las bibliotecas utilizadas, casos de uso de ejemplo y la siguiente documentación:

?? Manual del Usuario
   Describe la funcionalidad ofrecida por la IG al usuario final.

?? Manual del Configurador
   Explica la manera de configurar la IG para los distintos casos de uso.

?? Archivo de Definición de Grafo
   Describe el formato, el contenido y la utilización de los archivos con información de grafos usados por la IG.

?? Archivo de Configuración
   Describe el formato, el contenido y la utilización del archivo de configuración de la IG.

?? Entradas del Registro de Windows
   Describe las entradas del registro de Windows que utiliza la IG para personalizar su funcionamiento para cada usuario.

1.5.1 Usuarios

El manual del Usuario distingue los dos modos de ejecución de la IG, los cuales definen los dos tipos de usuarios finales. El usuario visualizador es el más básico y corresponde al modo de ejecución homónimo. El usuario editor corresponde al modo Editor. El configurador no es un usuario final de la IG, sino que la prepara para ser usada por los demás usuarios. Su tarea fundamental es armar el archivo de configuración y preparar el modo de ejecución, por ejemplo, creando directorios de trabajo.

1.5.2 Cumplimiento de los objetivos de HTS

HTS validó el producto final, luego de haberlo probado interactuando con sus sistemas, encontrándolo satisfactorio.

1.5.3 Aplicabilidad a otros casos de uso

La IG aplica sin cambios a diversos casos de uso, en lo que respecta al diseño interactivo de diagramas. Desarrollado nuevos algoritmos, se completaría la funcionalidad de ordenamiento automático para cada caso particular.

1.6 Mejoras a futuro

Durante el proceso de análisis, diseño e implementación registramos todas las mejoras que no estuvieran consideradas en los requerimientos y escaparan al alcance del proyecto. Las principales son:

1.6.1 Más generalidad

?? Algoritmos de ordenamiento configurables.
Ver 1.3.7.

?? Propiedades dinámicas.
Ver 0.

?? Mensajes de error o confirmación configurables.
Ver 1.4.5.

1.6.2 Más funcionalidad

?? Variantes de intercambio de información e interoperabilidad.
Ver 0.

?? Ayuda en línea.
Implementar la funcionalidad de ayuda en línea del Windows, para facilitar el aprendizaje de la herramienta por parte de nuevos usuarios.

?? Comando deshacer.
Es una funcionalidad prácticamente estándar en interfaces gráficas. Hicimos un breve análisis, y fue descartado para no complicar el desarrollo.

?? Utilización del portapapeles de Windows.
Copiar y pegar, no sólo en la misma vista, sino también en diferentes vistas, instancias de la IG e incluso otras aplicaciones, soportando diferentes formatos de portapapeles: texto, wmf, etc.

?? Herramienta de edición de nodos.
Análoga a la del arco, que permita cambiar el tamaño del nodo la posición de la etiqueta y otros parámetros dependientes del tipo de nodo.

?? Comandos de alineación de objetos.
Para ajustar la posición de varios objetos en función de uno de ellos.

?? Movimiento de objetos mediante el teclado.
Actualmente no es posible mover nodos y arcos exclusivamente con el teclado.

?? Grilla y líneas de guía.
Simplifican la edición manual de grafos, porque determinan restricciones en la movilidad de sus objetos componentes.

1.6.3 Estética y presentación

?? Visualización de las relaciones válidas.
Agregar la opción de visualizar gráficamente las relaciones válidas disponibles.

?? Utilización de la barra de estado.
Actualmente se utiliza en una forma estándar. Se podría usar para indicar las posibles acciones con el ratón, en cada momento.

?? Color de fondo del diagrama modificable.
Actualmente toma el del sistema operativo.

?? Etiquetas de arcos con referencias a sus nodos.
Por ahora solo permite referenciar su identificador. Ver 1.3.3.2.

?? Más tipos de nodos.
Nodos elípticos, rectángulos redondeados, de tamaño variable, con efectos tridimensionales, animados.

?? Más tipos de arcos.
Variedad de estilos en los extremos terminales de los arcos, curvas en lugar de segmentos, líneas punteadas representables con grosor mayor a uno.

1.7 Conclusiones

La IG cumple correctamente con los requerimientos de HTS, de acuerdo a nuestra verificación y la validación confirmada por HTS.

Es un software mantenible, entendible y fácilmente modificable, gracias a su diseño fuertemente orientado a objetos y los enfoques particulares utilizados.

Cumple también nuestros objetivos de generalidad, dado que es utilizable por sí sólo o interactuando con otros sistemas, y gracias a su configuración dinámica aplica a muchos casos de uso.

Capítulo 2 - El Algoritmo

El problema del dibujado automático de grafos, consiste en encontrar un algoritmo que a partir de la información de un grafo produzca un dibujo *legible* del mismo, es decir fácil de entender, fácil de recordar y esclarecedor de la realidad que representa. En cada caso se define el concepto *legible* en función de ciertos criterios estéticos. Algunos ejemplos pueden ser evitar la superposición de objetos o el cruzamientos de arcos, también minimizar la cantidad de codos en los arcos o el largo de los mismos. El problema planteado por HTS implica un conjunto de criterios estéticos que define los requerimientos del algoritmo de dibujado automático a desarrollar. Este capítulo detalla dichos requerimientos, los puntos relevantes del análisis, diseño e implementación, presenta el algoritmo obtenido y las mejoras que se le podría realizar a futuro.

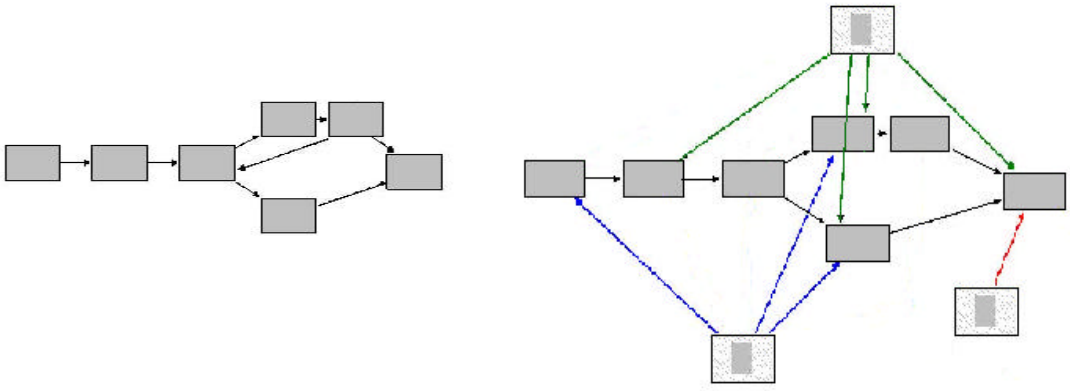
2.1 Requerimientos

El grafo que queremos representar está formado por un conjunto de nodos principales que determinan una jerarquía, un conjunto de nodos libres y un conjunto de arcos. Un resumen de los requerimientos se lista a continuación:

- Ordenar los nodos principales, minimizando cortes de arcos entre ellos
- Ubicar los nodos libres en función de su relación con los principales
- Evitar la superposición de nodos
- Evitar la superposición de arcos
- No se considera en la minimización de cruces de arcos, los arcos entre nodos de un mismo nivel.

Los criterios estéticos empleados y el funcionamiento esperado se detallan en el documento Requerimientos del ordenamiento automático.

Los siguientes diagramas muestran el resultado esperado por el cliente en casos de uso reales:



Caso 1 del cliente *Caso 2 del cliente*

Figura 14 – Ejemplos de resultados esperados por HTS

2.2 Análisis y diseño

El objetivo del diseño es obtener un algoritmo que satisfaga los requerimientos del ordenamiento automático de grafos. El algoritmo final diseñado consta de tres etapas. Primero se arma una versión reducida del grafo original denominada *grafo intermedio*, que esencialmente mantiene los nodos principales y los arcos entre ellos. Luego se ordena el grafo intermedio separando sus nodos por niveles, y ordenándolos dentro de cada nivel de manera de minimizar los cortes entre arcos. Finalmente, utilizando la ordenación del grafo intermedio, se asignan coordenadas a los nodos principales del grafo original, y luego se ubican los nodos libres evitando superponerlos. Estos tres pasos se detallan a continuación aplicándose al grafo de la Figura 15 como ejemplo.

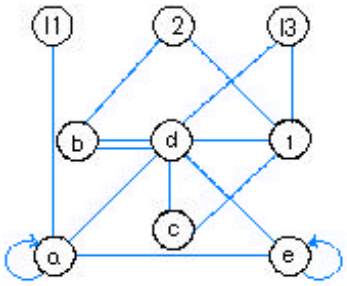


Figura 15 – Grafo inicial

El grafo del ejemplo está compuesto por seis nodos principales, con dos nodos por nivel (nivel 1: a y b; nivel 2: c y d; nivel 3: e y f) y tres nodos libres (l1, l2 y l3).

2.2.1 Armado del grafo intermedio

En esta etapa se genera una copia del grafo original, eliminando:

- nodos libres y sus arcos
- arcos entre nodos de un mismo nivel
- auto-arcos (arcos con cuyo nodo origen es el nodo destino)
- repeticiones de arcos entre dos nodos (arcos múltiples)

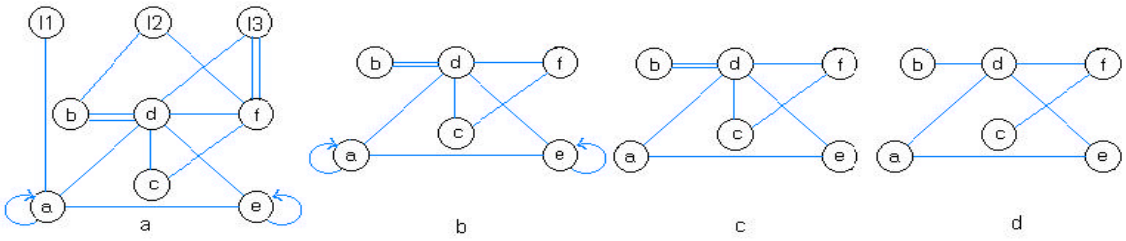


Figura 16 – Secuencia de pasos de la etapa inicial.

En la Figura 16 se muestra la secuencia de pasos que se aplican al grafo inicial: (a) grafo copia del grafo original; (b) se eliminan los nodos libres l1, l2, l3*; (c) se eliminan los auto-arcos; (d) se eliminan los arcos entre nodos del mismo nivel, obteniéndose un grafo solamente con nodos principales.*

2.2.2 Ordenamiento de grafo intermedio

En esta etapa se insertan *nodos auxiliares* y se aplica la fase dos de un algoritmo especializado en grafos jerárquicos llamado Sugiyama (ver 2.2.5). Estos dos pasos se detallan a continuación.

2.2.2.1 Nodos auxiliares

Por cada arco que une nodos con una diferencia de niveles mayor a uno, se crea un nodo auxiliar por cada nivel intermedio y un arco entre los nodos recién creados. El arco original es sustituido por la secuencia de arcos recientemente creada.

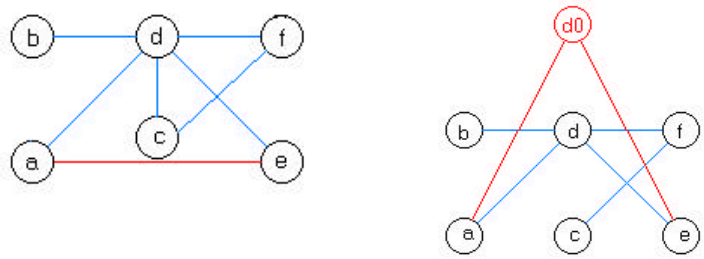


Figura 17 – Inserción del nodo auxiliar d0

En el ejemplo se inserta el nodo auxiliar d0 *y se crean los arcos* a-d0, d0-e *en sustitución del arco* a-e

2.2.2.2 Ordenamiento de nodos principales

Como se ve en la Figura 18-a, el grafo sólo tiene arcos entre nodos de niveles consecutivos. La fase dos de *Sugiyama* ordena los nodos dentro de cada nivel, asignándole posiciones de forma de minimizar los cortes de arcos entre niveles consecutivos. Se itera en los niveles del grafo, tomando dos niveles a la vez, considerando los nodos de un nivel como fijos y reordenando los del otro. Esta fase se explica en más detalle en el punto 2.2.5 - Fase dos de Sugiyama.

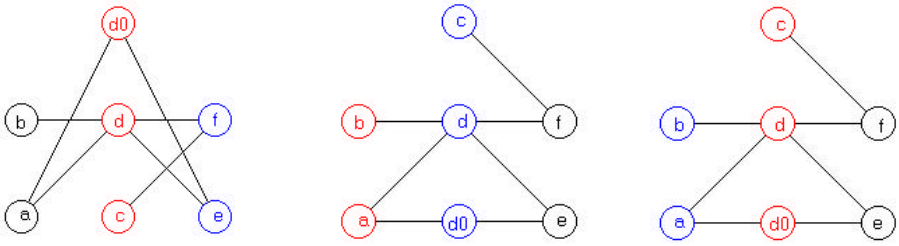


Figura 18 – Ejemplo de secuencia de iteraciones en los niveles del grafo.

En cada paso se toma a los nodos de color azul como fijos y se ordenan los nodos de color rojo.

2.2.3 Asignación de coordenadas

Esta es la etapa final del algoritmo completo. En ella se realizan los siguientes pasos ejemplificados en la Figura 19:

1. Asignar coordenadas del plano (x e y) a los nodos del grafo original tomando el ordenamiento obtenido en el grafo intermedio, balanceando los nodos de cada nivel.
2. Sustituir los nodos auxiliares del grafo intermedio, por codos en el arco del grafo original.
3. Ubicar los nodos libres en el grafo original.
   Los nodos libres se ubican en una de dos franjas, una por encima del grafo principal y otra por debajo. Para elegir la franja se traza una línea horizontal imaginaria que divide el grafo en dos y se compara la cantidad de arcos que unen al nodo libre con nodos de la parte superior e inferior, tomándose la que tenga mas arcos.
4. Asignar coordenadas (x e y) a los nodos libres evitando la superposición de los mismos.

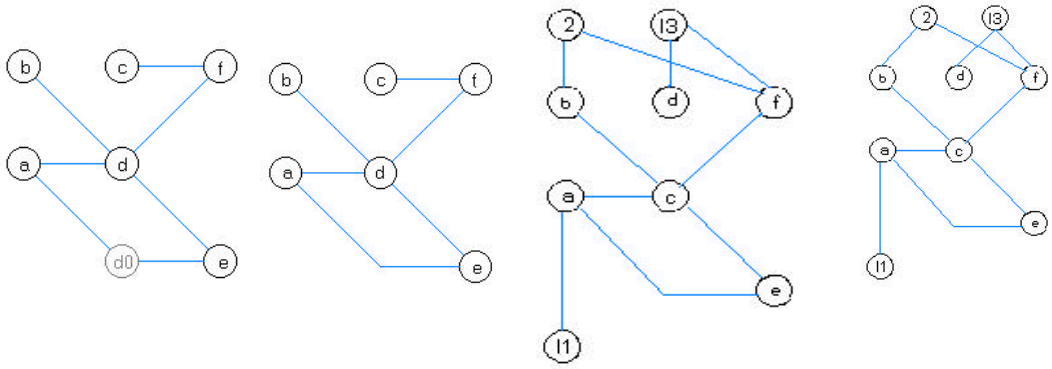


Figura 19 – Asignación de coordenadas a los nodos

2.2.4 Grafo resultante

Como resultado del algoritmo se obtiene el grafo original, con la información de posicionamiento de sus nodos y arcos (incluyendo codos). La distribución de los puntos de contacto de los arcos sobre los nodos, se realiza fuera del algoritmo. Ver 2.3.4 - Distribución de puntos de contacto. El resultado del caso del ejemplo se puede ver en la Figura 20.

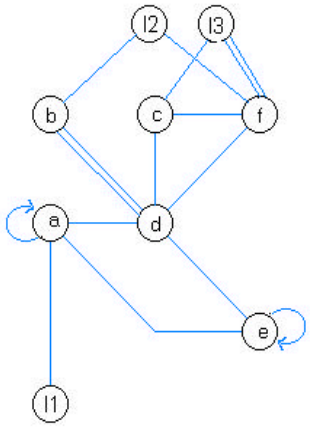


Figura 20 – Grafo resultante.

2.2.5 Fase dos de Sugiyama

El algoritmo de Sugiyama es una heurística que resuelve el problema de ordenamiento de grafos jerárquicos. Este algoritmo se ejecuta en cuatro fases, en al primera fase se asignan niveles a los nodos, transformando el grafo en un grafo jerárquico. En la segunda fase se minimizan los cortes de los arcos, permutando los nodos de cada nivel. En la tercer fase se designan coordenadas a los nodos para obtener un buen dibujo y en la cuarta se sustituyen los nodos auxiliares por codos en los arcos [Bowman-81].

Para entender la fase dos de Sugiyama debemos definir el concepto de *baricentro* de un nodo.

Baricentro
El baricentro de un nodo es el valor promedio de la posición de los nodos adyacentes a él. La Figura 22 presenta una definición más formal.

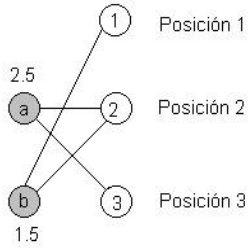


Figura 21 – Valor del baricentro de a y b.

$$Bar(v) = \begin{cases} \sum_{w \in adj(v)} \frac{Pos(w)}{|adj(v)|} & \text{Si } adj(v) \neq 0 \\ 0 & \text{en otro caso} \end{cases}$$

Figura 22 – Definición formal de baricentro

Parent-Baricenter:
El baricentro respecto al nivel anterior (parent).

Child-Baricenter:
El baricentro calculado respecto al nivel siguiente (child).

La fase dos del algoritmo se detalla a continuación:

```
FASE Dos
Se itera en los niveles del grafo y se ejecutan los siguientes pasos, donde L es el
nivel del grafo y M una ordenación
Ordeno niveles
1-M'= ordenación del nivel 'L' basado en el nivel 'L+1' y aplicando child-barycenter
2-Si M' tiene menos cruzamientos que M, entonces M=M' y movimientos=true
3-M'= ordenación del nivel 'L' basado en el nivel 'L-1' y aplicando parent-barycenter
4-Si M' tiene menos cruzamientos que M, entonces M=M' y movimientos=true
5-Si movimientos=true & iteraciones < max_iteraciones vuelvo a (1) sino voy a (6)
Intercambio nodos con igual baricentro
6-M=intercambio los nodos con el mismo parent-barycenter en el nivel 'L'
7-Si los nodos del nivel 'L+1' dejan de estar ordenados por el parent-barycenter voy a
(10)
8-M=intercambio los nodos con el mismo child-barycenter en el nivel 'L+1'
9-Si los nodos del nivel 'L' dejan de estar ordenados por el child_baricenter voy a (10)
si no fin.
10-si iteraciones_fase_2 < max voy a (1) si no fin.
11-fin
```

Figura 23 – Algoritmo de la segunda fase del algoritmo de Sugiyama

En el documento Algoritmos de dibujado de grafos se encuentra un análisis más detallado del algoritmo.

2.2.6 Opciones no implementadas

En esta sección detallaremos algunas variantes al algoritmo que fueron analizadas pero no implementadas. La principalmente razón para no implementar dichas variantes fue no excedernos demasiado en los plazos previstos.

2.2.6.1 Grafos no conexos

En caso de que el grafo intermedio no sea conexo, se ejecuta el algoritmo de ordenamiento Sugiyama a cada componente conexa. Luego cada una se ubica en una franja horizontal del plano distinta, definiendo entre ellas más zonas para ubicar nodos libres. En la Figura 24 se muestra un ejemplo del resultado esperado.

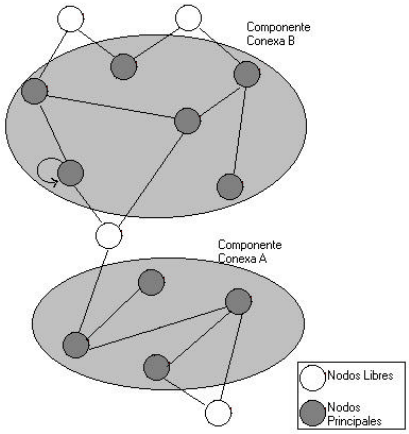


Figura 24 – Ejemplo de grafo principal no conexo

2.2.6.2 Ubicación de nodos libres

Con los nodos libres se utiliza un proceso similar, generando componentes conexas únicamente con nodos libres denominadas *grafos cero*. Cada grafo cero es planarizado para lograr una mejor representación visual evitando cruzamiento entre sus arcos. Luego se ubican en las zona determinadas para nodos libres de manera que se minimicen los cruces de arcos con los nodos con nivel. A cada grafo cero se le puede aplicar una rotación en torno a su centro para intentar obtener menos cruces de arcos con los nodos con nivel.

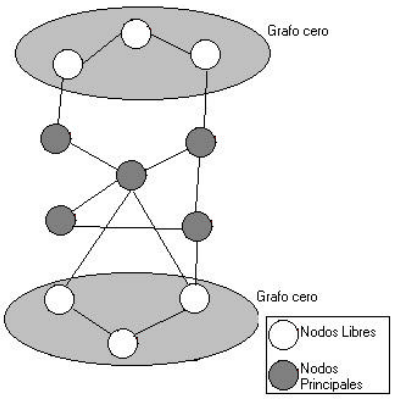
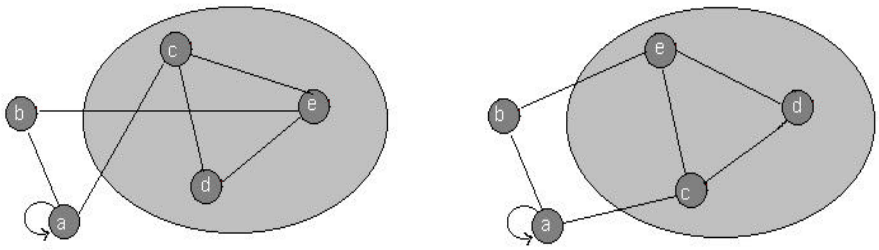


Figura 25 – Ejemplo de distintos grafos cero

2.2.6.3 Un nuevo problema de ordenamiento

Durante el análisis y la búsqueda de algoritmos existentes, observamos que todos los algoritmos encontrados aplicaban a grafos considerando todos sus nodos y arcos (aunque en alguna fase se limiten a una parte de ellos). Esto nos llevó a plantearnos un nuevo problema de ordenamiento más general, que consiste en ordenar ciertos nodos y arcos en función de un conjunto de criterios estéticos, pero tomando otros nodos y arcos como fijos. Buscamos alguna referencia a este problema, sin hallar nada que nos orientara en el camino a seguir.



(a) (b)

Figura 26 – Ejemplo de algoritmo genérico.

En la figura (a), se ordenan los nodos de la zona gris, dejando fijos a los nodos fuera de esta zona, el resultado obtenido se ve en la figura (b).

Si existiera un algoritmo genérico que resolviera el problema mencionado, se podría aplicar a nuestro caso particular, ordenando primero los nodos principales con Sugiyama, y luego aplicar una variante de planarización que tome los nodos principales como fijos y ordene los nodos libres.

El mejor acercamiento que logramos en nuestro caso concreto fue el de sustituir las componentes conexas del grafo intermedio por "nodos gigantes" y planarizar el grafo resultante. De esta manera se minimizan los cortes de arcos entre todos los nodos libres, permaneciendo únicamente los cortes de los arcos que van a nodos principales con los arcos de la componente conexa en cuestión.

2.3 Implementación

2.3.1 Arcos múltiples

Los *arcos múltiples* entre un par de nodos se toman como un solo arco en el momento de ordenar el grafo. En la tercer etapa del algoritmo se representan paralelos entre si para evitar superposición o cruzamientos.

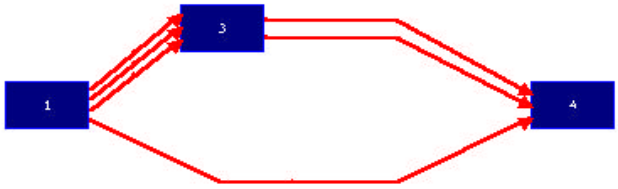


Figura 27 – Representación de arcos múltiples entre nodos

2.3.2 Retro-arcos

Los *retro-arcos* (o arcos hacia atrás) se consideran como arcos hacia delante. Esto significa que no hay distinción en el algoritmo al momento de tomar los arcos: si hay un arco hacia delante y un retro-arco en el mismo par de nodos, estos se toman como arcos múltiples

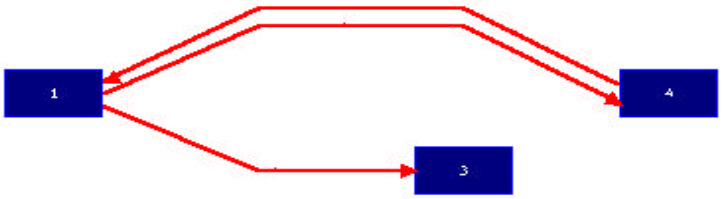


Figura 28 – Visualización de retro-arcos entre un par de nodos

Las siguientes figuras ilustran dos modos adicionales analizados para la representación de los retro-arcos:

Figura 29 – Visualización de retro-arcos como arcos independientes

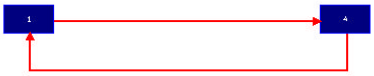


Figura 30 – Visualización de retro-arcos como arcos ortogonales

2.3.3 Cruzamiento de arcos múltiples

Los arcos múltiples entre un par de nodos se toman como un solo arco. Ésta decisión iguala la posibilidad de que ocurra un cruce de arcos entre dos arcos simples, con el cruce de un arco simple con arcos múltiples.

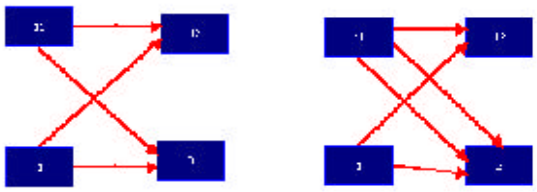


Figura 31 – Cruzamiento de arcos simples vs. múltiples

De esta manera se mejora la performance del algoritmo pero se sacrifica una parte estética del resultado. La solución a este problema, consiste en asignar un *peso* a los arcos de modo que los arcos simples tendrán un peso igual a uno y los arcos múltiples serán considerados por el algoritmo como un único arco de peso igual a la cantidad de arcos entre el par de nodos. En la minimización de cruces de arcos, se debe tomar en cuenta el peso de cada arco de modo de evitar el cruzamiento de mayor peso.

2.3.4 Distribución de puntos de contacto

El último paso del ordenamiento automático del grafo, es llamar a la actualización de los puntos de contacto de los arcos sobre cada nodo. Esto se maneja con una función que distribuye los arcos sobre los cuatro lados del nodo según la posición del nodo opuesto. Luego los puntos de contacto sobre un mismo lado se distribuyen uniformemente. En esta función se procesan también los auto-arcos, como se ve en la Figura 32.

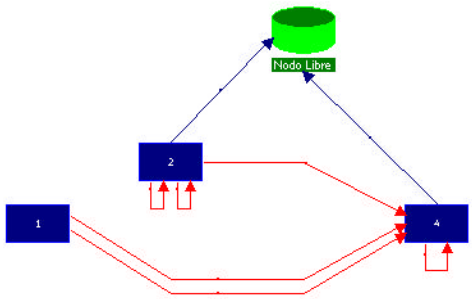


Figura 32 – Distribución uniforme de los arcos en el nodo

2.3.5 Implementación del algoritmo

El algoritmo de ordenamiento de grafos se implementa en la clase una clase `IG_GrafoIntermedio` derivada de `algorithm` (ver 1.3.7). El seudocódigo de su método `run` se puede ver en la Figura 33. En él se crea el grafo intermedio que es pasado como parámetro al método `run` de un objeto de la clase `IGSugiyama`. La clase `IGSugiyama` también deriva `algorithm` y únicamente implementa la fase dos de Sugiyama. En la Figura 34 se muestra un seudocódigo de la clase Sugiyama.

```
1- generar grafo intermedio
2- llamar a la segunda fase de Sugiyama. Ver Figura 34
3- actualizar los datos del grafo original con el resultado obtenido de sugiyama
4- colocar nodos libres
```

Figura 33 – Seudocódigo de `IG_GrafoIntermedio::run()`.

```
1- agregar nodos auxiliares
2- asignar posiciones iniciales a los nodos en cada nivel
3- ejecutar la segunda fase del algoritmo de sugiyama. Ver Figura 23
```

Figura 34 – Seudocódigo de la clase `IGSugiyama::run()`.

2.3.6 Utilización de bibliotecas

Se destaca la utilización de los métodos de la clase `graph` de la biblioteca GTL, por ejemplo métodos que permiten obtener los nodos adyacentes a un nodo, o todos los arcos de un nodo e iteradores que permiten iterar sobre nodos o arcos de un grafo. Se utilizaron también contenedores de la biblioteca STL, por ejemplo listas de nodos ordenadas por el baricentro y mapas de nodos ordenados por su nivel. Estos contenedores se utilizaron para facilitar la programación y mejorar la performance de algunas operaciones sobre el grafo, como por ejemplo obtener todos los nodos de un nivel determinado.

```
//! Mapa de nodos.
/*! Informacion de los nodos del grafo Sugiyama asociados a los nodos del grafo*/
node_map<IG_Sgym_Nodo*> m_Mapa_IGNodos;

//! Mapa de nodos.
/*! Mapa de nodos accesibles por el identificador del nodo. */
map<int, node>          m_id_2_node;      // Nodos por id

//! Mapa de arcos.
/*! Mapa de arcos accesibles por el identificador del arco */
map<int, edge>          m_id_2_edge;      // Arcos por id

//! Lista de posiciones segun el nivel.
/*! Lista de pares, posicion-nivel */
list<PosNivel>          m_lPosNivel;

//! Mapa de lista de nodos
/*! Mapa de lista de nodos, accesibles por el nivel. */
map<int, lnodos>        m_mapa_nivel;   // Nodos ordenados por nivel.
```

Figura 35 – Extracto de la utilización de contendores de la clase STL.

2.4 *Algoritmo resultante*

A continuación se muestran los mismos casos de uso enviados por el cliente, ordenados por el algoritmo.

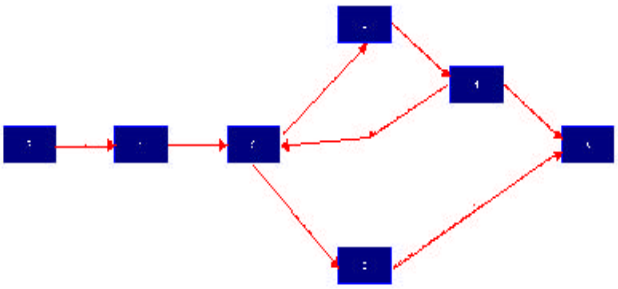


Figura 36 – Caso 1 del cliente resuelto por la IG

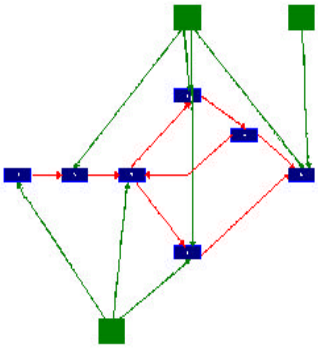


Figura 37 – Caso 2 del cliente resuelto por la IG

2.5 *Mejoras a futuro*

En el análisis del algoritmo se detallan algunos puntos que no fueron implementados. Estos se pueden implementar en un futuro, logrando un mejor resultado. A continuación se listan estos puntos así como otras mejoras realizables no analizadas.

2.5.1 Mejoras Visuales

?? Minimización de cortes
Minimizar los cortes no solo entre arcos, sino entre nodos y arcos.

?? Posicionamiento de etiquetas
Ubicar de manera inteligente las etiquetas en los arcos de forma tal que estas no se superpongan entre sí o con los nodos.

?? Sentido de la representación del grafo
Parametrizar el sentido de representación del grafo, no solo de izquierda a derecha, sino en los cuatro posibles sentidos.

?? Animación del ordenamiento
Mostrar el ordenamiento del grafo como una animación, en la cual cada nodo se desplace de su posición anterior a la nueva asignada por el algoritmo.

2.5.2 Mejoras del algoritmo

?? Separa el grafos principal en componentes conexas. Ver 2.2.6.1.

?? Ordenar nodos libres planarizando grafos cero. Ver 2.2.6.2.

?? Encontrar un algoritmo para el problema genérico. Ver 2.2.6.3.

2.6 Conclusiones

Se buscó, diseñó e implementó un buen algoritmo que satisface las necesidades del cliente. El algoritmo obtenido resuelve el problema de ordenamiento automático de grafos y con ello se cumple el objetivo, pero se puede mejorar como se indica en la sección anterior: Mejoras a futuro.

Durante el análisis nos planteamos un problema de ordenamiento automático de grafos más general, sobre el cual no encontramos información ni estudio previo. Creemos que un algoritmo genérico que resuelva este problema, se aplicaría a nuestro caso concreto y a muchos otros. Pensamos que el problema planteado es un buen punto de partida para futuros análisis sobre algoritmos de ordenamiento automático de grafos.

Capítulo 3 - El proyecto

Al comenzar este proyecto vimos que nos enfrentábamos al desarrollo de un software que implicaba la organización de un grupo de trabajo, por lo que hallamos importante aplicar procedimientos aprendidos en la carrera y en nuestras experiencias laborales. En este capítulo detallamos como se organizó el grupo y cual fue la dinámica del proyecto.

3.1 Organización del grupo de trabajo

La etapa de organización del grupo de trabajo fue muy importante dentro del proyecto ya que en ella se instrumentaron procedimientos para llevar a cabo el trabajo en forma ordenada. Se consideraron los siguientes puntos:

3.1.1 Comunicación

La comunicación interna del grupo y con los tutores se realizó a través de correo electrónico, reuniones y por teléfono. Se creó un grupo de correo electrónico en Internet que permitió enviar mensajes a través de una única dirección de correo, así como respaldar dichos mensajes y archivos en Internet. Las comunicaciones más relevantes fueron guardadas como documentación. A medida que surgían dudas o se debían tomar decisiones de diseño del producto, se llevaban a cabo reuniones con el cliente (el tutor externo por parte de HTS) para resolverlas. Se mantuvo un historial de las resoluciones tomadas en cada una de estas reuniones.

3.1.2 División del proyecto

Dividimos el proyecto en tres sub-proyectos para poder atacar el desarrollo de la IG en forma modularizada, distribuyendo y aprovechando así el tiempo de implementación. Los sub-proyectos definidos fueron: dibujado, algoritmo y comunicación.

El dibujado involucró el manejo y representación gráfica de los datos (por ejemplo, insertar y mover un nodo o cambiar las propiedades de un arco). El sub-proyecto del algoritmo se ocupó del ordenamiento automático del grafo. Por último, la comunicación comprendió el almacenamiento de datos y la interacción de la IG con otros sistemas.

3.1.3 Responsabilidades

Las resoluciones más relevantes del proyecto fueron tomadas por el grupo en su conjunto en forma democrática y compartiendo las responsabilidades.

Se nombró un responsable para cada uno de los sub-proyectos definidos quien debía asegurar que se estuviese cumpliendo con los requerimientos definidos. Dichos requerimientos no debían ser implementados únicamente por el responsable, sino que este podía solicitar ayuda al resto del grupo.

3.1.4 Asignación de roles

Se definieron tres roles: responsable de gestión de la configuración (RGC), vocero y secretario.

El RGC (ver punto 3.2.1) definió y documentó la estructura de directorios que se utilizó para almacenar los documentos y el código fuente generados a lo largo del proyecto. También mantuvo las versiones de dichos documentos y llevó a cabo un mecanismo de respaldos.

El secretario se ocupó de registrar los puntos relevantes de cada reunión, mantener las actas de reuniones actualizadas y recabar toda la información necesaria antes de cada reunión.

El vocero centralizó la comunicación del grupo con los tutores (resoluciones tomadas, fijación de reuniones, etc.).

El responsable de cada tarea varió a lo largo del proyecto, en función de circunstancias especiales o buscando la mayor practicidad. El rol más fijo fue el de RGC, que cambió una sola vez cuando el responsable original debió ausentarse casi por un mes.

3.1.5 Estándares

Se definieron estándares para la generación de la documentación del proyecto y la codificación del software.

En cuanto a la documentación del proyecto, el objetivo fue mantener un criterio en la presentación y organización de cada documento. Para esto se creó una plantilla de documentos donde se define dicho criterio. Esta plantilla se encuentra en la documentación del proyecto Plantilla Taller 5.dot.

Uno de los objetivos del producto era obtener un código fuente final que fuese claro y fácil de mantener. Para esto, se usaron estándares y recomendaciones de codificación para ayudar a que el código fuente fuese uniforme, sin importar el programador que lo desarrolló. Estos estándares se encuentran en la documentación del proyecto Guías de Codificación C y C++.

3.1.6 Documentación

La documentación generada en el proyecto se clasificó en los siguientes grupos:

Comunicaciones

Historial de los correos más relevantes.

Diseño

Información del análisis y el diseño del software.

- Presentación de HTS Ltda.
  Carta de presentación de la empresa HTS Ltda.
- Propuesta de proyecto inicialmente planteado por HTS
  Presentación del proyecto de HTS a la facultad de ingeniería en febrero del año 2000.
- Requerimientos iniciales planteados por HTS
  Planteo inicial de los requerimientos del software deseados por HTS.
- Historial de los requerimientos
  Detalla evolución de los requerimientos de la IG.
- Especificación de requerimientos
  Detalla los requerimientos finales de la IG.
- Análisis y diseño del software
  Análisis y diseño detallado de la IG.
- Especificación funcional
  Detalla todas las funcionalidades provistas por la IG.
- Requerimientos del ordenamiento automático
  Detalla los requerimientos finales en cuanto al ordenamiento automático de grafos.
- Algoritmos de dibujado de grafos
  Describe brevemente los métodos de ordenamiento de grafos estudiados, haciendo principal hincapié en el método Sugiyama (ya que es el utilizado por la IG).

Informes

Análisis de herramientas e informe final.

- Evaluación de lenguajes de programación
  Detalla el relevamiento y evaluación de las características de los lenguajes, bibliotecas y herramientas consideradas para la programación de la IG.
- El formato GML
  Describe detalladamente qué el formato GML de especificación de y como debe utilizarse.
- Informe final
  Informe final del proyecto de taller 5.

Organización

Organización del grupo.

- Gestión de configuración
  Define todo lo relativo a la gestión de configuración del software.
- Guías de codificación en C y C++
  Detalla los estándares de codificación y algunas recomendaciones a tener en cuenta.
- Horas dedicadas al taller
  Planilla que indica una aproximación de las horas dedicadas al proyecto.
- Plan de validación
  Define el plan de validación de la IG por parte de HTS.
- Plantilla de documentos
  Plantilla base de todos los documentos generados durante el proyecto.
- Vocabulario
  Define los términos utilizados en la documentación del proyecto.
- Bibliografía
  Planilla que detalla la bibliografía utilizada durante el proyecto.

Producto

Documentación del producto. Ver 1.5 - El producto final.

Reuniones

Actas de todas las reuniones realizadas.

3.2 Herramientas utilizadas

3.2.1 Gestión de configuración del software

La gestión de configuración del software (GCS) es una disciplina que se encarga de coordinar el desarrollo de un software y controlar los cambios y la evolución de las componentes del mismo [Ghezzi-91]. Llamaremos *componentes* del software a la documentación y el código fuente generado durante el proyecto.

Al iniciar el proyecto se definió un documento de configuración donde se detalla la estructura de directorios que se utilizó para almacenar las componentes generadas. Este documento se encuentra junto con la documentación del proyecto Gestión de Configuración.doc.

El control de versiones de las primeras componentes generadas fue administrado exclusivamente por el RGC (ver punto 0). Una vez que se comenzó con la implementación, se introdujo el uso de un manejador de versiones (*VSS* - Visual Source Safe), ya que el volumen de componentes generadas era muy difícil de administrar. Cada vez que se desea trabajar sobre una componente, ésta debe ser bloqueado para que ningún otro

integrante del grupo la modifique al mismo tiempo. Al finalizar el trabajo sobre dicha componente, debe ser ingresada nuevamente al VSS, asociándole una descripción de los cambios realizados, quedando nuevamente disponible.

3.2.2 Cronogramas

Para la elaboración de los cronogramas se utilizó Microsoft Project 2000.

3.2.3 Diseño orientado a objetos

Para el diseño de clases utilizamos Visual Modeler 2.0, la herramienta gráfica de modelado de objetos incluida en el Visual Developer Studio 6.0. Visual Modeler utiliza un subconjunto de los diagramas que define el lenguaje de modelado unificado (UML), permitiendo rápidamente modelar las clases a implementar.

3.2.4 Entrono de desarrollo

Para el desarrollo de la IG utilizamos Microsoft Visual C++, la herramienta de desarrollo en C++ incluida en el Visual Developer Studio 6.0. Por más detalles ver el punto 1.4.1.

3.2.5 Detección de errores

Se utilizaron dos herramientas para la detección de errores y mejoramiento de la performance. Ambas forman parte de un paquete que interactúa con Visual C++ en forma amigable, integrándose a su entorno de desarrollo.

NuMega TrueTime 1.22

Es una herramienta utilizada para el análisis de performance de una aplicación en tiempo de ejecución. Calcula el tiempo transcurrido en cada una de las funciones del código fuente, permitiendo detectar las secciones críticas.

NuMega BoundsChecker 6.20

Es una herramienta para detectar errores o posibles mejoras en aplicaciones desarrolladas en C, C++ y Delphi. Por ejemplo, desperdicios de memoria, errores del API de windows, errores de ejecutables o librerías dinámicas entre otros.

3.2.6 Documentador automático

Para la generación de la documentación técnica se utilizó un documentador automático (Doxygen 1.2.7), el cual genera documentación en formato HTML o ayuda de windows (HLP) en función de los comentarios que se encuentren en el código fuente.

3.3 Desarrollo del proyecto

3.3.1 Cronograma inicial

El cronograma inicial del proyecto se muestra en la Figura 38:

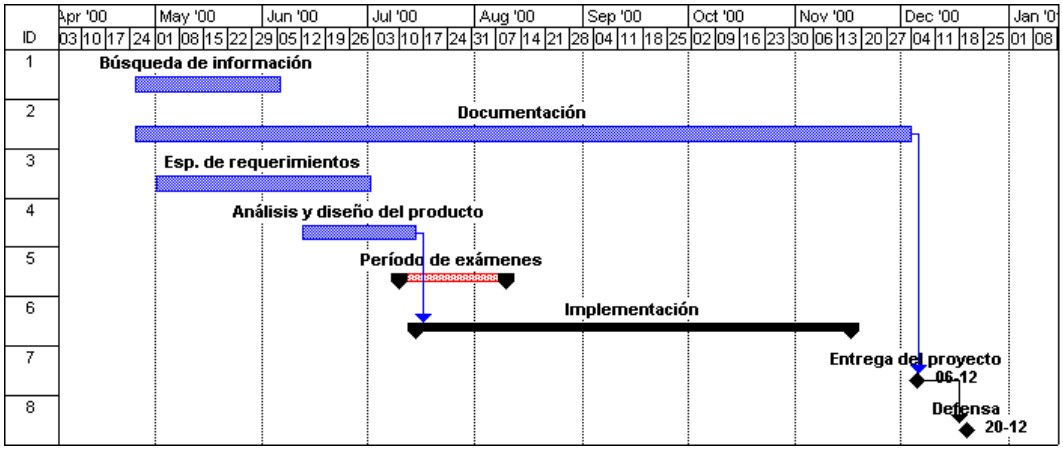


Figura 38 – Cronograma inicial

3.3.2 Definición de requerimientos

Los requerimientos iniciales de la IG eran muy generales y estaban vagamente especificados. Durante la implementación de la IG, se introdujeron cambios y nuevos requerimientos a pedido de HTS, así como nuevas funcionalidades sugeridas por nosotros - convenientes para la IG – que debían ser aprobadas por HTS (por ejemplo: guardar la ubicación de los objetos dentro de un diagrama).

La definición imprecisa de los requerimientos antes de comenzar con la implementación de la IG tuvo como consecuencia que las etapas de especificación de requerimientos, análisis, diseño e implementación se solaparan en el cronograma (ver cronograma resultante en el punto 0).

A lo largo de la implementación se generaron prototipos que fueron verificados inicialmente por nosotros y luego entregados al cliente para su validación.

Al finalizar la implementación, las funcionalidades provistas por la IG tuvieron una dimensión mucho mayor de la planteada inicialmente (por ejemplo: uso del zoom, definición dinámica de tipos de nodos y arcos, guardar la ubicación de los objetos dentro del diagrama).

3.3.3 Plan de validación

El proyecto se dio por concluido al aplicar el plan de validación acordado previamente con el cliente. El documento que detalla dicho plan se encuentra incluido en la documentación del proyecto Plan de Validación.doc.

3.3.4 Horas dedicadas

Basándose en registros de actividad y estimando las horas que no fueron registradas se realizó el cálculo de las horas dedicadas al proyecto. Los resultados se muestran en la Figura 39.

Tarea	Período Aprox.	Horas
Análisis, Diseño, Documentos	***22/04/2000 al 19/10/2000***	***601.00***
Búsqueda de información	22/04/2000 al 19/10/2000	117.00
Elaboración de Documentos	22/04/2000 al 17/05/2001	484.00
Implementación	***20/10/2000 al 10/04/2001***	***943.00***
Implementación Algoritmo	20/10/2000 al 10/04/2001	280.00
Implementación Dibujador	20/10/2000 al 10/04/2001	392.00
Implementación Comunicación	20/10/2000 al 10/04/2001	231.00
Implementación Otros	20/10/2000 al 10/04/2001	40.00
Reuniones	***22/04/2000 a la finalización***	***154.50***
Reuniones con tutores	22/04/2000 a la finalización	56.50
Reuniones internas	22/04/2000 a la finalización	98.00
Informe Final	***09/05/2001 a la finalización***	***525.00***
Elaboración	09/05/2001 a la finalización	525.00
Imprevistos	***22/04/2000 a la finalización***	***-***
Períodos de exámenes, viajes de trabajo, casamientos	22/04/2000 a la finalización	-
	TOTAL DE HORAS -->	***2223.50***
	TOTAL DE HORAS POR RECURSO -->	***741,17***

Figura 39 – Horas dedicadas al proyecto

3.3.5 Cronograma resultante

El siguiente cronograma muestra la evolución real del proyecto:

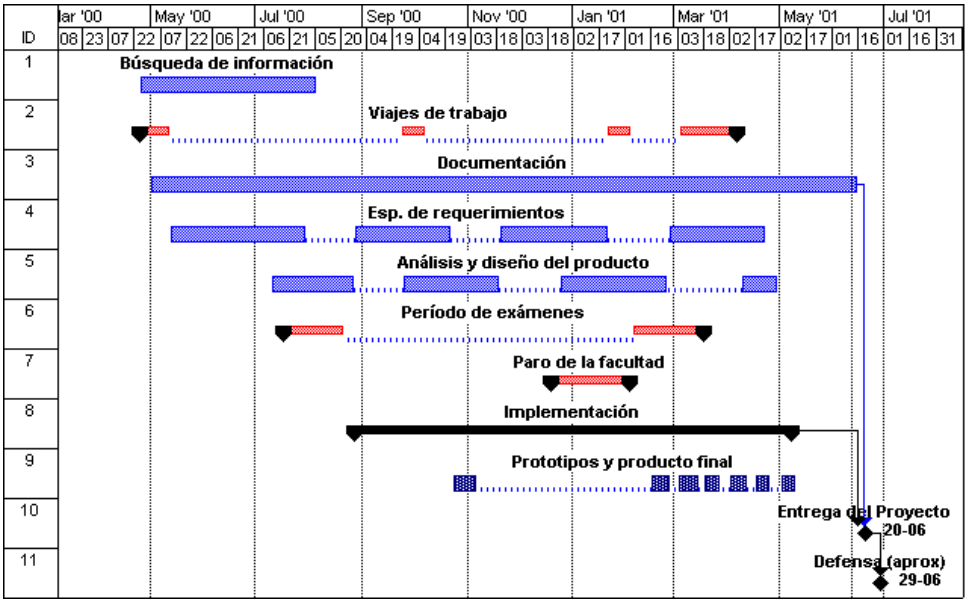


Figura 40 – Cronograma final

3.4 Conclusiones

Las metodologías de trabajo y la organización realizada contribuyeron a que el trabajo en equipo funcionara en forma eficiente, correcta y armoniosa, aún en aspectos delicados como la dirección del proyecto y la toma de decisiones importantes.

Las herramientas y las técnicas de trabajo (diseño, implementación, verificación, documentación) ayudaron al cumplimiento de nuestros objetivos y a la obtención de un producto satisfactorio para el cliente.

En cuanto al desarrollo del proyecto, analizando la desviación respecto del cronograma establecido inicialmente concluimos que se debería haber tomado en cuenta los siguientes puntos importantes:

- Tomar en cuenta en el cronograma inicial del proyecto posibles imprevistos como demoras del cliente, viajes de trabajo, asuntos personales.
- Prever holguras para cada una de las etapas del proyecto.
- Definir fechas límites para el corrimiento de las etapas. Por ejemplo, fijando una fecha límite para la recepción de nuevos requerimientos.
- Hacer un seguimiento del cronograma inicial, re-estimando los plazos a medida que se van concretando los requerimientos, el análisis y el diseño.
- Obtener la aprobación de los requerimientos, el diseño y los planes de validación del producto por parte del cliente antes de comenzar la implementación, aunque con esto se sacrifique en cierto grado la calidad del producto final obtenido.

Conclusiones generales

La IG cumple con los requerimientos de HTS y está abierto a que su funcionalidad aumente fácilmente. Cumple con nuestros objetivos de generalidad por lo que es aplicable a diversos usos que, al igual que HTS, no encuentren soluciones de este tipo en el mercado.

Este proyecto fue para nosotros una muy buena experiencia de trabajo en equipo, organización y aplicación de metodologías, técnicas y conocimientos aprendidos durante la carrera. No sólo en cuanto a los resultados obtenidos sino también en lo que respecta al trabajo realizado.

Por otra parte fue una interesante experiencia académica en la cual incursionamos en el problema de dibujado de grafos, conociendo los algoritmos existentes que resuelven dicho problema. Finalmente encontramos un problema más general que podría abrir una nueva área de estudio en el problema de dibujado de grafos.

Bibliografía

[Atallah-99]
Mikhail J.Atallah, *Algorithms and Theory of Computation Handbook*, Atallah, USA, 1999

[Booch-94]
Grady Booch, *Object Oriented Analysys and Design with Aplications*, Addison-Wesley, 1994.

[Bowman-81]
Ivan Bowman, *Methods for Visual Understanding of Hierarchy System Structures*, http://plg.uwaterloo.ca/~itbowman/CS746G/Notes/Sugiyama1981_MVU/, 1981.

[Ceballos-91]
Fco. Javier Ceballos, *Curso de Programación C++ Orientada a Objetos*, Addison Wesley Iberoamericana, 1991.

[Foley-96]
Foley, Van Dam, Feiner, Huges, Phillips, *Introducción a la graficación por computador*, Addison Wesley, 1996.

[Ghezzi-91]
Ghezzi, Jazayeri, Mandrioli, *Fundamentals of Software Engineering*, Prentice Hall, 1991.

[Kernighan-98]
Brian W. Kernighan, Denise M. Ritchie, *The C programming language*, Prentice Hall, 1998.

[Prosise-96]
Jeff Prosise, *Programming Windows 95 with MFC*, Microsoft Press, 1996.

[Stroustroup-97]
Bjarne Stroustroup, *The C++ Programming Language*, Addison Wesley, 1997.

Índice alfabético

Índice de figuras

Índice general