DESARROLLO DEL PROTOTIPO DEL MOTOR DE SIMULACIÓN ELÉCTRICA DEL SISTEMA DE POTENCIA

Por

DIEGO ALEJANDRO ZAPATA LOPEZ

Informe Práctica

Asesor

CARLOS JAIME FRANCO

Universidad Nacional de Colombia

Sede Medellín 2009

Contenido

Introducción

La simulación de los sistemas eléctricos de potencia desempeña un papel cada vez más importante en los procesos de planeación, ya que es un aspecto fundamental para conocer las condiciones normales y anormales para la operación de un sistema de potencia.

En la actualidad existen diversos simuladores de sistemas eléctricos de potencia con los que se pueden conocer las condiciones mencionadas anteriormente, pero dichos simuladores no permiten el acceso a su código interno, ni tampoco son flexibles para la aplicación de nuevas tecnologías.

El propósito fundamental de este proyecto es el desarrollo de un motor de simulación eléctrica en el lenguaje de programación C++, que sea flexible para las aplicaciones de nuevas tecnologías (como los PMU) y que a su vez tenga las mismas funciones de un simulador de sistemas eléctricos de potencia convencional.

En este documento se exponen los principios y la base del desarrollo del prototipo del motor de simulación eléctrica del sistema de potencia. Se presenta inicialmente los fundamentos teóricos empleados para el desarrollo del proyecto, seguido de la descripción detallada de lo implementado hasta el momento.

1. Antecedentes

Ocurrido el evento del apagón del 26 de Abril del 2007 que dejo sin suministro de energía a la mayor parte del país por un tiempo de 4 horas, surgió la necesidad de crear esquemas suplementarios de protección del sistema de potencia que permita tomar acciones correctivas automáticas ante cualquier ocurrencia de eventos de gran magnitud, a partir de ello nace el proyecto SIRENA (Sistema de Respaldo Nacional ante Eventos). Para el proyecto SIRENA se vio la necesidad de crear un motor de simulación eléctrico, el cual pueda ser flexible para cualquier tipo de aplicación que se desee realizar (cosa que el simulador DIgSILENT no permite) y para ello, se tuvo la idea de hacerlo en la plataforma de programación C++ ya que este es un lenguaje que presenta un gran rendimiento y es fácil de adaptar a otros leguajes de programación.

Uno de los objetivos directos del motor de simulación es obtener los resultados del flujo de carga del sistema; para este objetivo se ha establecido un modelo orientado a objetos, en cual se establecen claramente todos los componentes del interconectado y se incluyen métodos propios de cada elemento para calcular el flujo de carga.

2. Generalidades del Proyecto

2.1 Etapas del Proyecto

Etapa de inducción:

- Introducción a los sistemas de potencia.
- Entrenamiento sobre la fase previa del proyecto y los desarrollos realizados (Estudio del prototipo PSDataXchange).
- Aprendizaje de DOM (Document Object Model) y su librería MSXML.

Etapa de desarrollo:

- Definición del diagrama de clases del sistema.
- Definición de clases en C++.
- Programación objetual de la reducción de barras de un sistema.
- Implementación del modulo de lectura de un archivo XML.
- Instalación y pruebas de MTL4 (Matrix Template Library 4).
- Configuración de las librerías Boost para lograr un ordenamiento de mínimo grado.
- Estudio de conceptos eléctricos para desarrollar el flujo de potencia.
- Construcción de la matriz Y-barra para solucionar el sistema.

Etapa final:

- Documentación y pruebas del prototipo.

2.2 Aportes del proyecto

- Documentación y aplicación de la lectura del archivo XML en la plataforma de programación C++.
- Estructura de clases (Programación Orientada a Objetos) para la programación del prototipo de simulación eléctrica.
- Incorporación de nuevas metodologías de desarrollo en el flujo de carga de un sistema de potencia.
- Uso de librerías que optimizan el manejo de memoria en cálculos complejos.

3. Desarrollo del Prototipo

3.1 XML (eXtensible Markup Language)

3.1.1 Introducción a XML

Qué es XML?

XML significa *eXtensible Markup Language*, o lenguaje de anotación extensible que se esta convirtiendo rápidamente en un estándar para el almacenamiento de datos legibles por maquina de forma estructurada. Por otra parte XML es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C) que utiliza etiquetas para marcar, describir, clasificar y organizar información de una manera especifica.

A continuación se tiene un ejemplo pequeño de un archivo XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Agenda>
    <Contacto>
        <Nombre>Giovanny Nese</Nombre>
        <Tel>73313</Tel>
        <Dir>Cra 80 # 53 - 79</Dir>
    </Contacto>
    <Contacto>
        <Nombre>Pedro Perez</Nombre>
        <Tel>3214266</Tel>
        <Dir>Cll 51 # 47 - 17</Dir>
    </Contacto>
</ Agenda >
```

Figura 1. Fragmento XML.

3.1.2 XML – DOM

DOM (Document Object Model)

Una traducción al español no literal, pero apropiada, podría ser Modelo en Objetos para la representación de Documentos o también Modelo de Objetos del Documento.

DOM es una plataforma que proporciona un conjunto estándar de objetos a través de la cual se pueden crear documentos HTML y XML, navegar por su estructura y modificar, añadir y borrar tanto elementos como contenidos. A

través del DOM los programas pueden acceder y modificar el contenido, estructura y estilo de los documentos HTML y XML, que es para lo que se diseñó principalmente.

El responsable del DOM es el consorcio W3C (*World Wide Web Consortium*).

DOM para XML

DOM para XML es un modelo de objetos estándar (propuesto por el W3C) que muestra el contenido de un documento XML. La especificación del Modelo de Objeto de documento (DOM) del W3C define actualmente lo que debería mostrar un DOM como propiedades, métodos y eventos.

Veamos un ejemplo, supongamos el siguiente trozo de código XML:

```
<TABLE>
  <TBODY>
   <TR>
       <TD>Shady Grove</TD>
       <TD>Aeolian</TD>
   </TR>
   <TR>
       <TD>Over the River, Charlie</TD>
       <TD>Dorian</TD>
   </TR>
  </TBODY>
 </TABLE>
```

Figura 2. Fragmento XML.

El DOM representaría esta tabla de la siguiente forma:

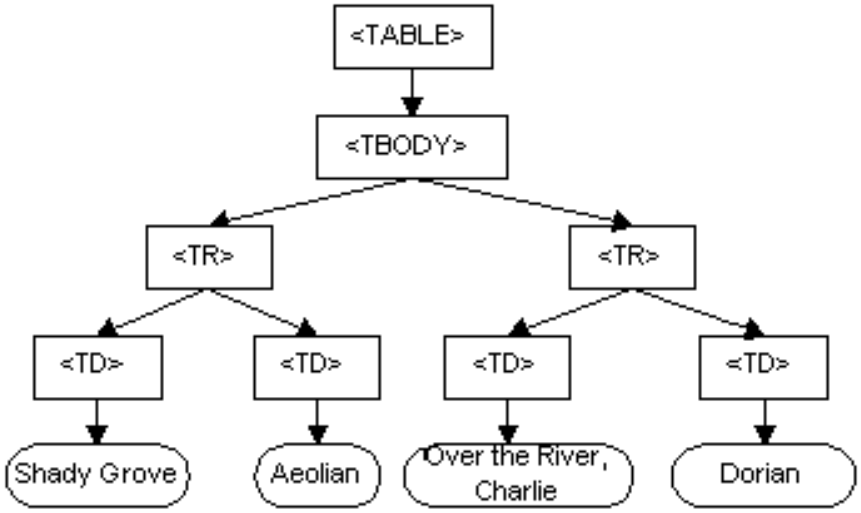


Figura 3. Estructura DOM.

3.1.3 Librería MSXML DOM

Microsoft XML Core Services (MSXML) permite realizar aplicaciones basadas en XML a los usuarios de Microsoft Visual Studio, JScript, Visual Basic Scripting Edition, entre otros.

Para este caso se utilizó la herramienta Microsoft Visual Studio 2008 donde se aplica la librería MSXML6 para cargar y recorrer el archivo XML con los datos del interconectado nacional.

3.1.4 Propiedades y métodos utilizados de la librería MSXML

Una mirada amplia y completa en todas las funciones del DOM sería imposible en el espacio proveído aquí, pero se expondrán las utilizadas hasta ahora.

Clases de MSXML que fueron usadas en el prototipo:

- **XMLDOMDocument:** Es el nodo superior del árbol del documento XML.
- **XMLDOMNode:** Representa un único nodo del árbol del documento XML.
- **XMLDOMNodeList:** Es la colección de todos los objetos del XMLDOMNode.
- **XMLDOMNamedNodeMap:** Es la colección de todos los atributos del árbol del documento XML.

Para cargar y recorrer en C++ el documento XML que contiene los datos del sistema nacional, se utilizaron las siguientes propiedades y métodos de la librería MSXML.

Propiedades utilizadas:

- **lenght** (para IXMLDOMNodeList): Indica el número de artículos en la colección.
- **attributes:** Contiene la lista de atributos del nodo referenciado.
- **text:** Representa el contenido del nodo en formato texto.

Métodos utilizados:

- **selectSingleNode(nombre del nodo específico)**: Retorna una subsección del archivo, partiendo del nombre del nodo solicitado.
- **SelectNodes(“*”):** Devuelve la lista de nodos, de la misma jerarquía, partiendo de un nodo base.
- **getNamedItem(“Name”):** Retorna el atributo con el nombre especificado.
- **item**: Permite acceder a cada nodo de una colección, dentro de una misma jerarquía.

3.2 UML (Unified Modeling Language)

UML Es un lenguaje gráfico para visualizar, especificar, construir y documentar un proyecto de software. Es importante resaltar que UML es un "lenguaje de

modelado" para describir métodos o procesos, además se utiliza para definir un sistema, para detallar los artefactos en el mismo y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Es importante destacar que un modelo UML describe el comportamiento de un sistema, sin apegarse a una plataforma especifica de desarrollo.

3.2.1 Diagrama de Clases

Las clases en este diagrama se representan como rectángulos con tres compartimientos correspondientes al nombre, los atributos y las operaciones respectivamente. Las relaciones pueden ser de varios tipos (generalmente tres: generalización, terminada en un triángulo en uno de los extremos, agregación, terminada en un rombo en uno de los extremos, y asociación, sin ningún elemento en los extremos).

Ejemplo:

La Figura 4 muestra un ejemplo de notación UML para capturar los atributos y acciones de una lavadora.

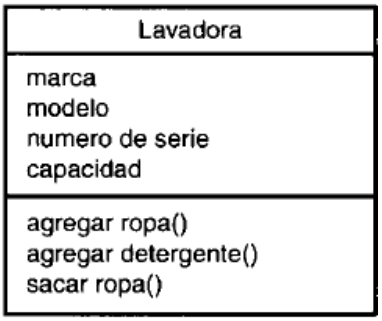


Figura 4. Ejemplo de una clase en UML.

3.2.2 Relaciones para el diagrama de Clases

Las relaciones entre clases se documentan con una descripción de su propósito, su cardinalidad (cuantos objetos intervienen en la relación) y su opcionalidad (cuando un objeto es opcional para intervenir en una relación).

A continuación se muestran las relaciones mas usadas en los diagramas de clases.

- **Asociación (binaria o n-aria)**

  Una asociación binaria se representa mediante una línea sólida que une dos clases, se trata de una relación entre las dos clases no muy fuerte, es decir, no se exige dependencia existencial ni encapsulamiento.

  Para referirse a asociación n-aria, simplemente se lleva a cabo la relación binaria definiendo la multiplicidad de la misma; dicha multiplicidad señala la cantidad de elementos de una clase que pueden relacionarse con los

de una clase asociada. En la figura 5 hay un ejemplo de la relación utilizada en este proyecto.

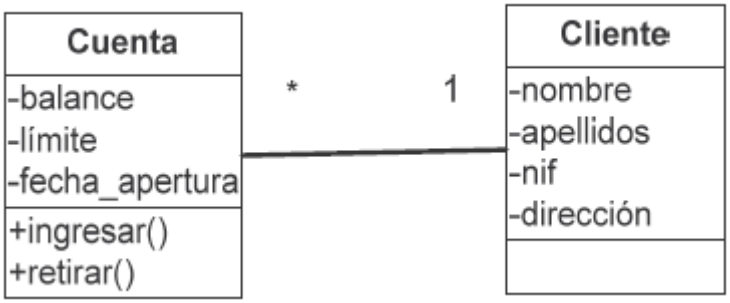


Figura 5. Relación uno a uno o más.

- **Generalización**

  La relación de generalización, denotando herencia entre clases, se representa mediante un triángulo sin rellenar del lado de la superclase. La subclase hereda todos los atributos y mensajes descritos en la superclase.

  UML representa la herencia con una línea que conecta a la clase principal con la secundaria. En la parte de la línea que se conecta con la clase principal, se coloca un triángulo sin rellenar que apunte a la clase principal. Este tipo de conexión se interpreta con la frase "*es un tipo de".*

  Ejemplo:

  Un Mamífero es un tipo de Animal, y un Caballo es un tipo de Mamífero.

  La figura 6 muestra la jerarquía de la herencia, junto con otras clases. Observe la apariencia del triángulo y las líneas cuando varias clases secundarias son herencia de una clase principal.

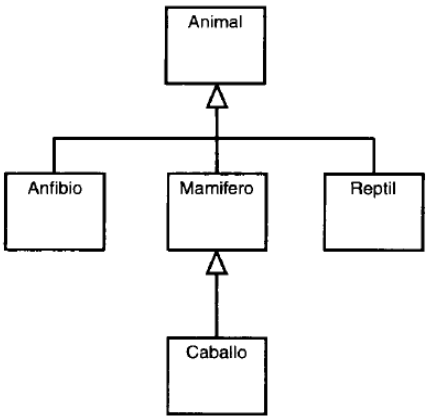


Figura 6. Una jerarquía de herencia en el reino animal.

- **Agregación**

  En ocasiones una clase consta de otras clases. Éste es un tipo especial de relación conocida como agregación o acumulación. Los componentes y la clase que constituyen son una asociación que conforma un todo.

  En UML una asociación por agregación se representa por una línea entre el componente y el todo con un rombo si relleno que conforma al todo.

Figura 7. Sistema de cómputo como una agregación

- **Composición**

  Es una asociación fuerte, que implica tres cosas

  - Dependencia existencial. El elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1, es creado al mismo tiempo.

  - Hay una pertenencia fuerte. Se puede decir que el objeto contenido es parte constitutiva y vital del que lo contiene

  - Los objetos contenidos no son compartidos, esto es, no hacen parte del estado de otro objeto.

Se denota dibujando un rombo relleno del lado de la clase que contiene a la otra en la relación.

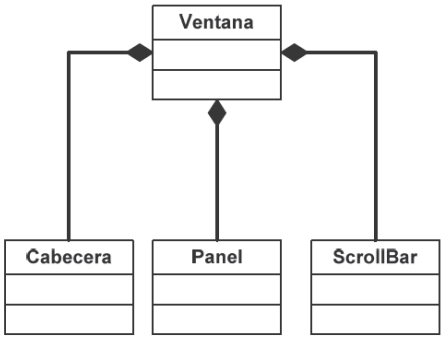


Figura 8. Composición

3.2.3 Diagrama de Clases para el prototipo

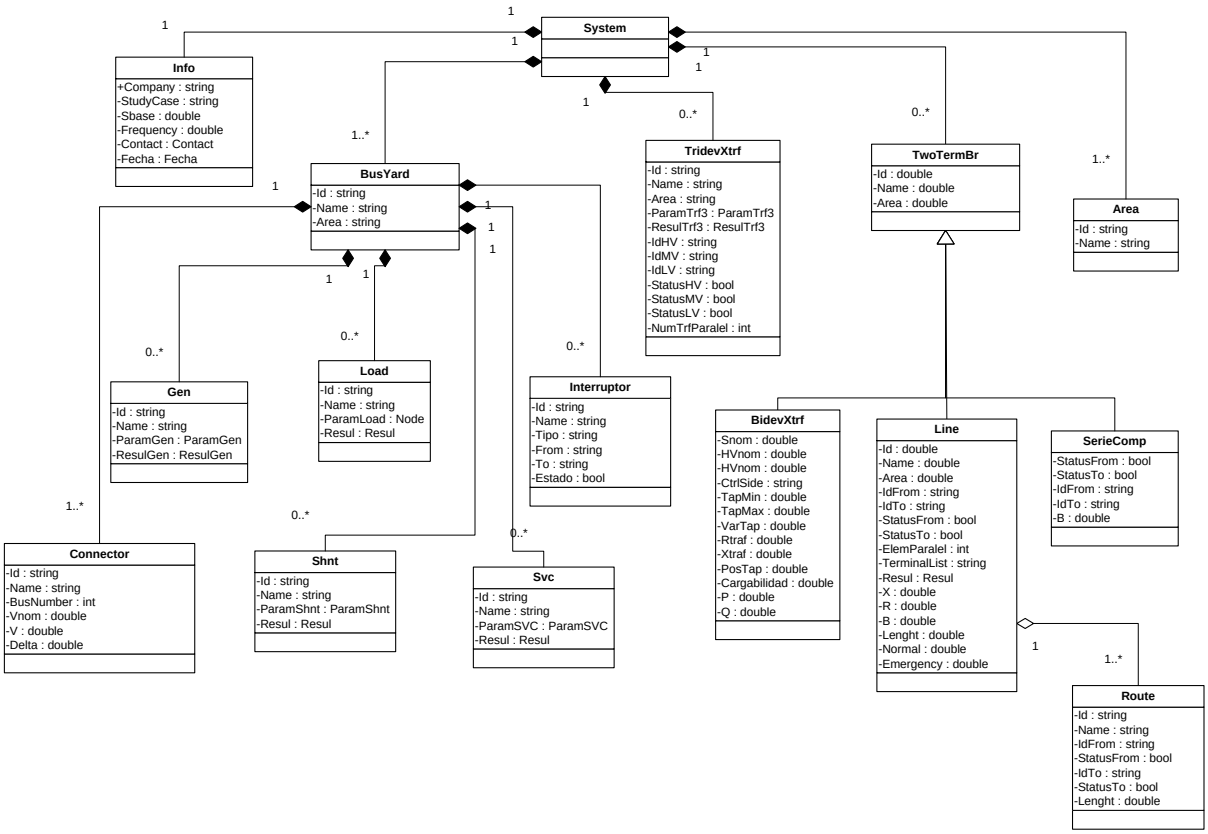


Figura 9. Diagrama de clases del sistema

El diagrama de clases es una herramienta que permite entender de forma grafica la estructura del sistema, y la interrelación de sus elementos. Además es utilizado para mostrar lo que el sistema puede hacer, dando a entender como puede ser construido.

3.3 Programación algoritmo reducción de barras

El objetivo de este algoritmo, es reducir a su minima expresión los diferentes tipos de subestaciones. Cabe resaltar que el algoritmo funciona para cualquier tipo de configuración de subestación (En anillo, interruptor y medio, doble barra, etc.).

Como datos de entrada para el algoritmo se requieren, principalmente, la lista de conectores e interruptores, con sus respectivos datos (estatus, From y to), esto con el fin de determinar conectividad.

A continuación se muestra la lógica del algoritmo de reducción de barras para una subestación con configuración interruptor y medio como la de la figura 10.

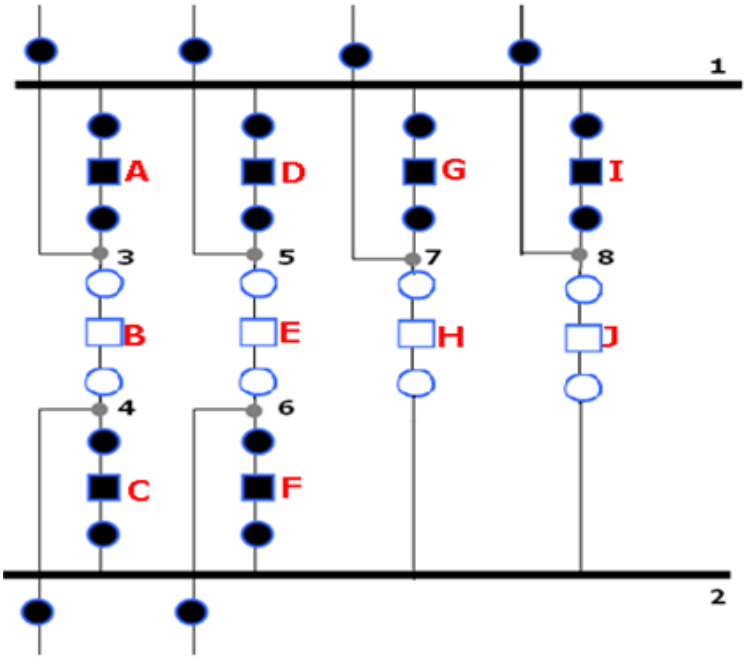


Figura 10. Configuración interruptor y medio.

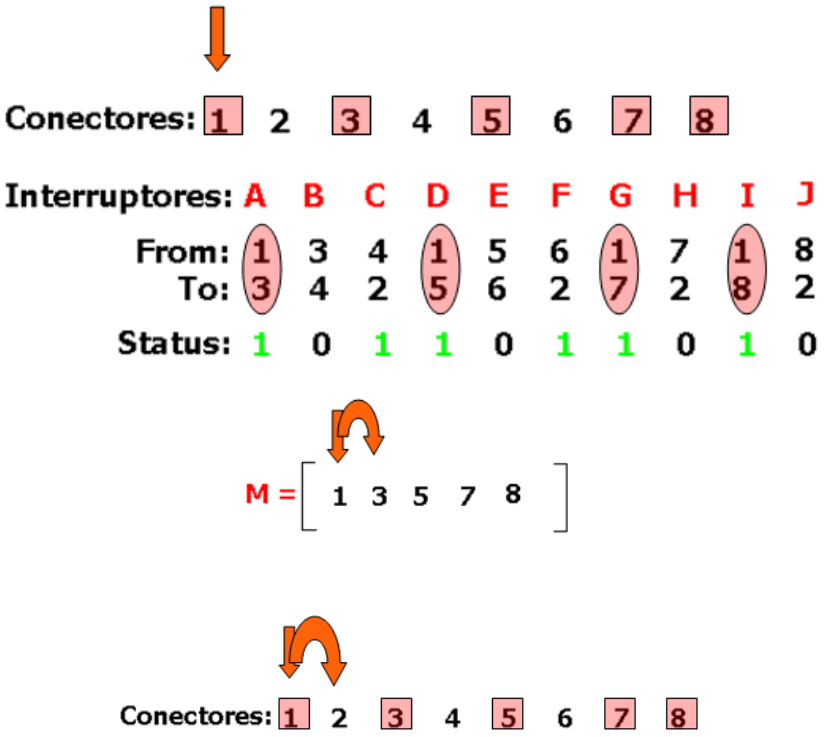
En la figura 10, los interruptores desenergizados son los que están en blanco y los energizados los que está en negro.

Para este caso, los datos que se tienen son:

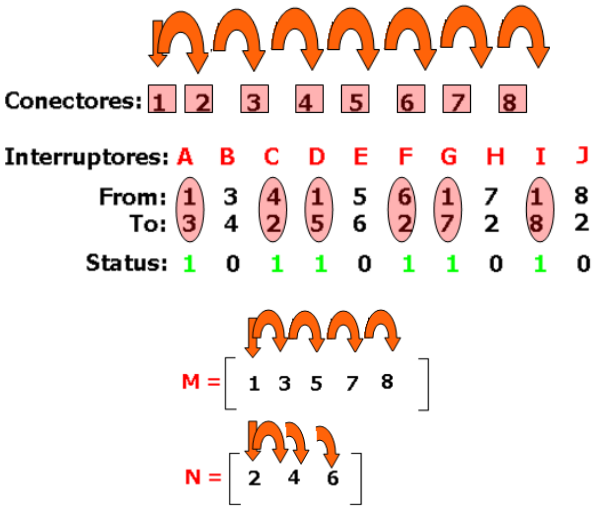
Conectores:	1	2	3	4	5	6	7	8		
Interruptores:	A	B	C	D	E	F	G	H	I	J
From:	1	3	4	1	5	6	1	7	1	8
To:	3	4	2	5	6	2	7	2	8	2
Status:	1	0	1	1	0	1	1	0	1	0

Al tener disponible esta información, el algoritmo actúa de la siguiente manera:

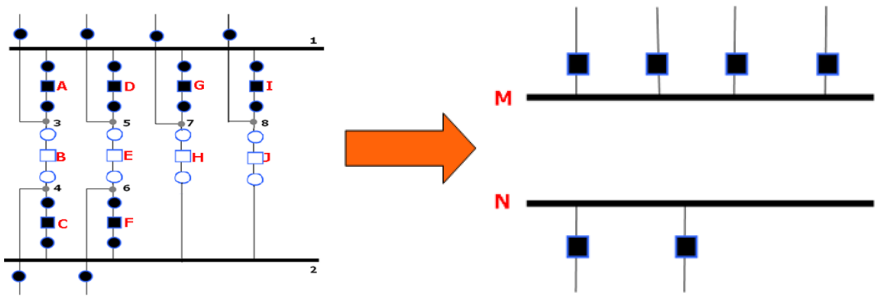
Primero selecciona un conector cualquiera y busca todos los conectores que estén conectados con él (teniendo en cuenta que los interruptores estén cerrados); luego de esto se hace una búsqueda de los vecinos de estos, este proceso se repite recursivamente hasta el punto que no existan vecinos, de esta forma se dice que el conjunto de conectores reunido es un solo punto y se continua el proceso con un conector que no se halla tenido en cuenta en ningún proceso anterior.



El proceso se repite hasta haber recorrido todos los elementos de la lista de conectores.



El número de vectores resultantes, indican el número de nodos en el que se ha dividido la subestación.



Interpretación grafica de la reducción.

Algunos ejemplos gráficos de la reducción.

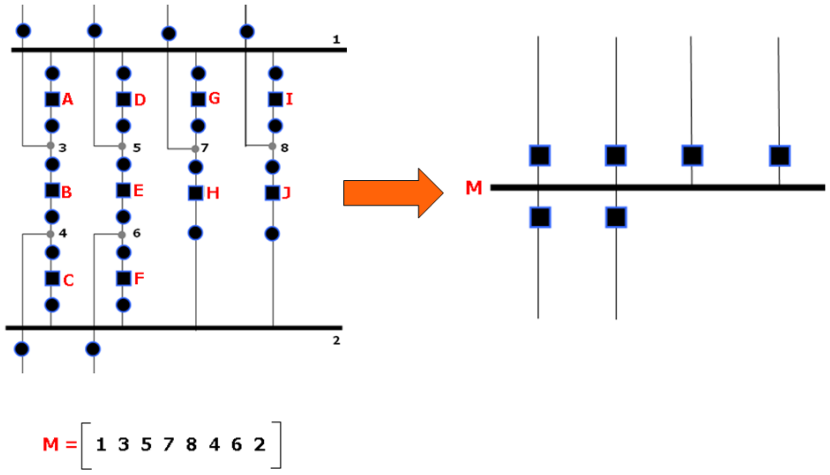


Figura 11. Ejemplo 1 Reducción de barras.

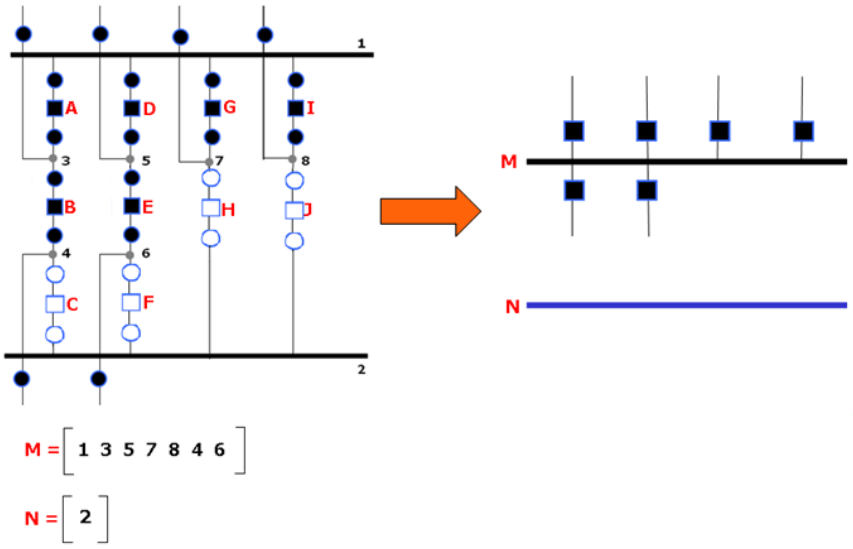


Figura 12. Ejemplo 2 Reducción de barras.

3.4 BOOST C++ Libraries

Boost son un conjunto de librerías de código abierto, que aportan grandes funcionalidades al lenguaje, su principal característica es el amplio uso de plantillas, que dan gran rendimiento y flexibilidad.

Dentro de las funcionalidades de Boost se destacan una gran cantidad de algoritmos matemáticos, que han sido optimizados y desarrollados por múltiples programadores de software libre. El nivel de optimización de código y el manejo de memoria es tal, que ya varas librerías de esta biblioteca pertenecen al estándar C++.

A continuación se presentan los pasos básicos para la instalación de la biblioteca Boost:

1. Descargar las librerías de la página oficial del proyecto (versión comprimida con el código fuente) www.boost.org.

2. Luego de descomprimir el archivo .ZIP se copia la carpeta boost para luego pegarla en la ruta de instalación del Visual Studio 9, generalmente es C:\Archivos de programa\Microsoft Visual Studio 9.0\VC\include.

3.5 MTL4 (Matrix Template Library)

Matrix Template Library (MTL) es una biblioteca de álgebra lineal para programas en C++.

MTL usa programación con plantillas, lo cual reduce considerablemente la extensión del código. Las matrices definidas por esta biblioteca están disponibles para diferentes tipos de datos (float, double, complex<float>, complex<double>, etc.). Además soporta matrices densas y dispersas.

MTL proporciona una notación natural e intuitiva, con la que encapsula al usuario el manejo de memoria y la implementación de algoritmos en operaciones complejas.

Los pasos básicos para el uso de esta librería se muestran a continuación:

1. Descargar la versión más actual de la página oficial de la librería (un archivo comprimido con extensión .ZIP con el código fuente de las bibliotecas) www.osl.iu.edu/research/mtl/mtl4.
2. Luego de descomprimir el archivo, la carpeta más importante es la llamada boost, de la cual se copia su contenido y se pega en las librerías del Boost (algunas carpetas ya existen, solo se agrega el nuevo contenido referente a Mtl).

3.6 Módulos del prototipo

3.6.1 Modulo de lectura

La función Leer_XML, es la encargada de cargar el archivo XML migrado de DIgSILENT.

Leer_XML (Sistema, Nombre_archivo)

El primer parámetro para esta función es un objeto tipo **Sistema** en el cual quedará representado todo el sistema de potencia.

El segundo parámetro es una cadena de caracteres en la que se describe el nombre completo del archivo XML a cargar (incluyendo extensión).

En esta función se recorre el archivo XML en forma de árbol, guardando la información del interconectado en el objeto sistema; en dicho recorrido se diferencia cada componente junto con sus características específicas.

3.6.2 Modulo Llenado Map y Lista de Nodos

En este bloque se lleva a cabo la asociación nodo – posición en matriz.

Luego de realizar la reducción de barras cada subestación queda con un conjunto de vectores en donde cada vector es un solo punto común. Para construir el listado de nodos se toma un representante de cada vector y se le asigna una posición de matriz, es importante resaltar que para cada conjunto la posición de matriz será la misma que la de su representante. Luego de hacer estas asociaciones tenemos en la variable Map un mapeo completo de todo el sistema, además en el listado de nodos se tendrá guardado el nombre de un representante de cada grupo.

Además en este método se crea un conjunto de nodos ficticios, equivalentes a los transformadores tridevanados, con la notación: NF (Nodo Fake) + id del transformador.

Por ultimo en esta función se recorren todos los elementos que conecten nodos (líneas, transformadores y compensaciones serie), con el objetivo de determinar que nodos salen del sistema, para de esta forma solo trabajar con los nodos que hagan parte del sistema eléctrico.

Llenado_Map_Nodos (Sistema, Lista_Nodos, Map, Listado_Nodos_Desconectados)

Sistema: descripción completa del interconectado.

Lista_Nodos: conjunto de nodos representados en la matriz de admitancia.

Map: asociación nombre_nodo – posición_matriz.

Listado_Nodos_Desconectados: Listado de los nodos que no se tendrán en cuenta en el flujo de carga.

3.6.3 Modulo Construcción Matriz de admitancia

Este bloque construye la matriz de admitancias recorriendo las líneas, transformadores (tanto bidevanados como tridevanados) y capacitores Serie. Es de vital importancia resaltar que al recorrer las líneas se tiene en cuenta el efecto capacitivo que estas presentan (B/2 en cada nodo relacionado).

Construcción_Matrix (Matriz, Listado_Lineas, Map, Sistema)

Matriz: arreglo de admitancias del sistema.

Listado_Lineas: listado de las líneas del interconectado.

Map: es el encargado de realizar el mapeo entre nombre de nodo y posición de matriz.

Sistema: Objeto con toda la información recopilada en el archivo XML.

3.6.4 Modulo de ordenamiento

En este método se modela el sistema como un grafo; se utilizan funciones de BOOST para representar la matriz de admitancia como un grafo bidireccional, para de esta forma calcular el orden óptimo de los nodos en la matriz (según lo establece el método de ordenamiento de mínimo grado). En esta función también se actualiza el Map, para hacer un mapeo correcto entre los nodos y su correspondiente posición en la matriz. Además el listado de nodos se reorganiza quedando acorde a las posiciones en la matriz.

Ordenamiento (Matriz, Lista_nodos, Vector_orden, Map)

Matriz: arreglo a ordenar.

Lista_nodos: los nombres que aparecen en este vector corresponden a los nodos de la matriz de admitancia; para este método el vector entra en forma invertida (como resultado de una manipulación anterior de este vector en la eliminación de nodos desconectados).

Vector_orden: para este método ingresa vacío, y luego sale con el siguiente formato.

> [4, _, _, _] la posición en el vector indica el nodo y el valor en cada espacio del vector indica la ubicación ideal del correspondiente nodo (para este ejemplo el nodo 0 debe quedar en la posición 4).

Map: es el encargado de realizar el mapeo entre nombre de nodo y posición de matriz.

3.6.5 Modulo de Construcción Vector Potencia

Para este arreglo se buscan todas las generaciones y cargas del sistema; es clave que dicho vector sea coherente con los nodos de la matriz (misma posición para todos), para de esta manera relacionar de forma clara los “Load” y “Gen” a sus respectivas barras.

Construcción_Vector_P (Vector_P, Map, Sistema)

Vector_P: vector donde quedará guardadas todas las generaciones y cargas.

Map: elemento que retorna la posición de matriz de un determinado nodo.

Sistema: Objeto con toda la información recopilada en el archivo XML.

3.6.6 Modulo Creación Imagen

Este modulo es de gran utilidad, ya que permite graficar la matriz de admitancias en un archivo BMP, para esto se crea la estructura básica del archivo y se pinta píxel por píxel, especificando la escala de grises de cada color primario (RGB).

Crear_Imagen (Matriz, Nombre_Archivo)

Matriz: arreglo del cual se desea generar un grafico.

Nombre_Archivo: nombre con el que quedará guardado el BMP (incluye extensión).

3.6.7 Modulo Creación Texto

Adicionalmente se tiene dos módulos importantes para exportación de datos, los cuales pueden crear archivos de texto partiendo de matrices y vectores.

Crear_txt_matrix (Matriz, Nombre_archivo)

Crear_txt_vector (Vector, Nombre_archivo)

Matriz o Vector: arreglo del cual se desea generar un archivo de texto.

Nombre_Archivo: nombre con el que quedará guardado el TXT (incluye extensión).

3.7 Caso de estudio

Como caso de estudio se tomó un sistema de diez barras con tres transformadores, tres generaciones, cuatro cargas (Load) y una compensación serie (CompSerie). Aunque esta configuración sea pequeña, abarca gran cantidad de conceptos y elementos, que a la larga, ponen a prueba la toda funcionalidad del prototipo PSDataXchange.

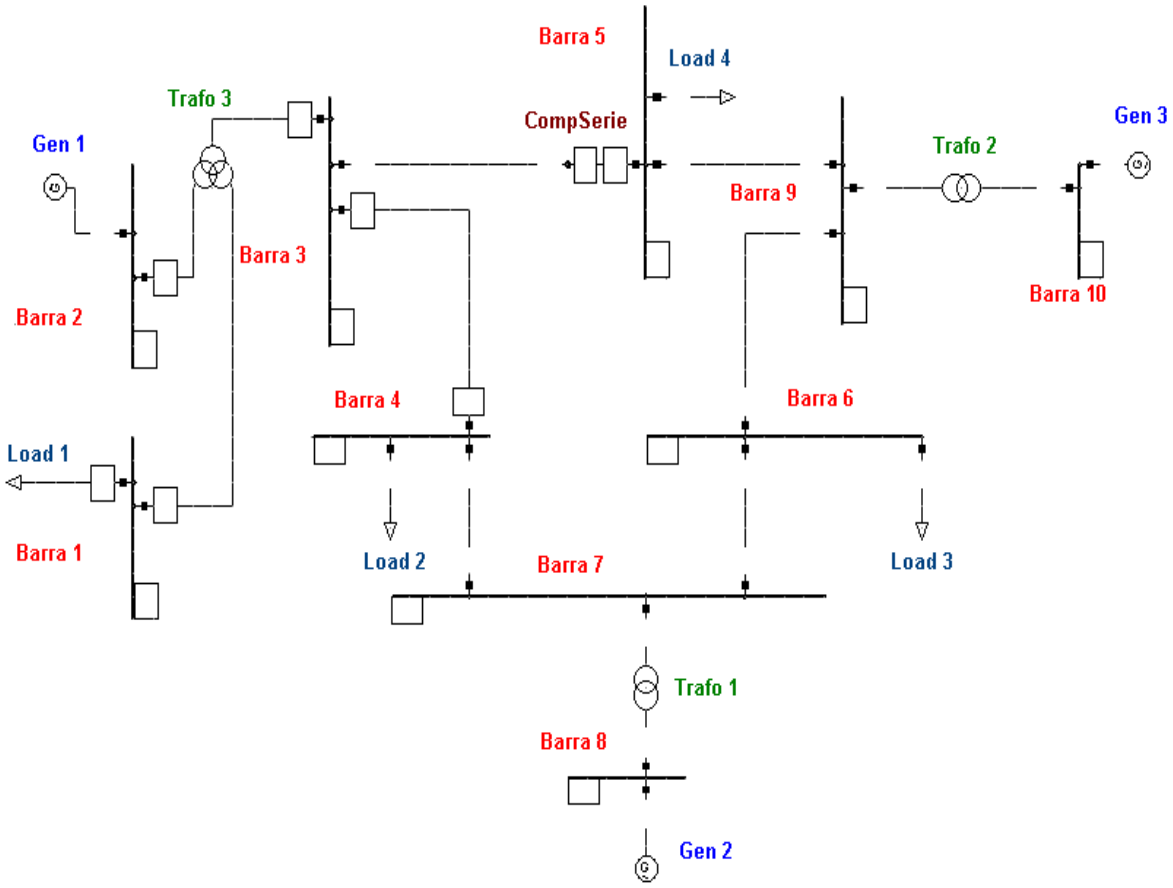


Figura 13 Diagrama de diez barras.

A continuación se muestra un fragmento de código XML, en el cual se describe el transformador tridevanado referenciado en la figura 13. Esta información proviene de una migración llevada a cabo en DIgSILENT PowerFactory 14.

```
+ <PSDX:BusYardList>
- <PSDX:XtrfList>
  - <PSDX:Xtrf Id="T3_tridev" Name="T3_tridev" Area="Ant01" Type="Tridev">
      <PSDX:HV Status="1">dos1</PSDX:HV>
      <PSDX:MV Status="1">uno1</PSDX:MV>
      <PSDX:LV Status="1">diez1</PSDX:LV>
      <PSDX:NumTrfParalel>1</PSDX:NumTrfParalel>
    - <PSDX:ParamTrf3>
        <PSDX:ZHvMv RHvMv="0.000000" XHvMv="0.0705" />
        <PSDX:ZMvLv RMvLv="0.000000" XMvLv="0.4275" />
        <PSDX:ZLvHv RLvHv="0.000000" XLvHv="0.1250" />
        <PSDX:SHVnom>200.000000</PSDX:SHVnom>
        <PSDX:SMVnom>200.000000</PSDX:SMVnom>
        <PSDX:SLVnom>40.000000</PSDX:SLVnom>
        <PSDX:HVnom>230.000000</PSDX:HVnom>
        <PSDX:MVnom>18.000000</PSDX:MVnom>
        <PSDX:LVnom>11.000000</PSDX:LVnom>
        <PSDX:CtrlSide>HV</PSDX:CtrlSide>
        <PSDX:TapMinAlta>-12.000000</PSDX:TapMinAlta>
        <PSDX:TapMaxAlta>8.000000</PSDX:TapMaxAlta>
        <PSDX:VarTapAlta>1.250000</PSDX:VarTapAlta>
        <PSDX:TapMinMedia>0.000000</PSDX:TapMinMedia>
        <PSDX:TapMaxMedia>0.000000</PSDX:TapMaxMedia>
        <PSDX:VarTapMedia>0.000000</PSDX:VarTapMedia>
        <PSDX:TapMinBaja>0.000000</PSDX:TapMinBaja>
        <PSDX:TapMaxBaja>0.000000</PSDX:TapMaxBaja>
        <PSDX:VarTapBaja>0.000000</PSDX:VarTapBaja>
      - <PSDX:Ratings>
          <PSDX:EmergencyH>234.000000</PSDX:EmergencyH>
          <PSDX:EmergencyM>234.000000</PSDX:EmergencyM>
          <PSDX:EmergencyL>78.000000</PSDX:EmergencyL>
        </PSDX:Ratings>
```

Figura 14 Descripción Trafo 3.

La correspondiente matriz de admitancia del sistema se presenta en la tabla 1, en dicha tabla se presentan las impedancias de todas las barras, incluyendo el nodo ficticio del transformador tridevanado y un nodo referente a la compensación serie.

	Barra 1	**Barra 8**	**Barra 10**	**Barra 2**	**NF_Trafo_3**	**Barra 3**
Barra 1	4,14938				-4,14938	
Barra 8		17,3611				
Barra 10			17,0648			
Barra 2				5,36193	-5,36193	
NF_Trafo_3	-4,14938			-5,36193	0,890618	8,62069
Barra 3					8,62069	11,4794
Barra 7		-17,3611				
Barra 9			-17,0648			
CompSerie						-13,8889
Barra 6						
Barra 4						-6,21118
Barra 5						

Tabla 1a. Matriz dispersa del sistema.

	Barra 7	**Barra 9**	**CompSerie**	**Barra 6**	**Barra 4**	**Barra 5**
Barra 1						
Barra 8	-17,3611					
Barra 10		-17,0648				
Barra 2						
NF_Trafo_3						
Barra 3			-13,8889		-6,21118	
Barra 7	39,9954			-10,8696	-11,7647	
Barra 9		32,8678		-5,88235		-9,92063
CompSerie			2,25248			11,6364
Barra 6	-10,8696	-5,88235		16,7519		
Barra 4	-11,7647				17,9759	
Barra 5		-9,92063	11,6364			-1,71578

Tabla 1b. Matriz dispersa del sistema (continuación).

Los resultados del flujo de carga DC obtenidos por DIgSILENT se muestran junto con los datos calculados por el prototipo PSDataXchange.

Barras	DIgSILENT	PSDataXchange
Barra 1	-10,2954798	-10,2954771
Barra 2	8,502977	8,502821
Barra 3	1,2542065	1,2542064
Barra 4	-5,2139785	-5,2139785
Barra 5	1,9160459	1,9160459
Barra 6	-3,7071065	-3,7071065
Barra 7	-2,5411823	-2,5411823
Barra 8	0	0
Barra 9	2,9047226	2,9047226
Barra 10	5,7586254	5,7586254

Tabla 2. Ángulos del sistema

Se consideró Gen 2 como generador de referencia, con una potencia calculada de 77 MW en ambos aplicativos. Es importante resaltar que el sistema, en el aplicativo PSDataXchange, fue resuelto por el método del bi-gradiente conjugado estabilizado, en contraste a DIgSILENT que usa el tradicional método de Newton-Raphson.

4. Anexos

4.1 Operadores en C++

Operador punto “.”: sirve para acceder a un elemento de un objeto usando el objeto real.

```
class P_ejemplo

{

        int num;

        public:

        void est_num(int val) {num=val;}

        void mostrar_num();

};

void P_ejemplo :: mostrar_num()

{

        cout << num << "\n";

}

main(void)

{

        P_ejemplo ob; //declarar un objeto

        ob.est_num(1); //acceso a ob directamente

        ob.mostrar_num();

        return 0;

}
```

Operador de resolución de ámbito "::": es útil para distinguir entre variables miembro de una clase y otro tipo de variables.

```
class X
{
        int m;
        public :
        void Setm (int);
        void Getm (void) {cout <<m;}
};

void main( )
{
        X x;
        x.Setm (5);
        x.Getm ( );
}

void X::Setm (int m)
{
        X::m=m; //para distinguir el parámetro m, de miembro m de la clase X
}
```

Operador flecha "->": es fundamental para acceder a un elemento específico de un objeto cuando se usa un puntero al objeto.

```
main(void)

{

        P_ejemplo ob, *p;    //declarar un objeto y un puntero a él

        ob.est_num(1); //acceso a ob directamente

        ob.mostrar_num();

        p=&ob; //asignar a p la dirección de ob

        p->mostrar_num(); // acceder a ob usando un puntero

        return 0;

}
```

4.2 Función virtual

Es una función que se declara en la clase base como "virtual" y se redefine en una o más clases derivadas, esto con el fin de otorgar varias funcionalidades al mismo método.

```
class B
{
        public: virtual int fun(int x) {return x * x;}   // virtual
};

class D1 : public B             // Clase derivada-1
{
        public:  int fun (int x) {return x + 10;}        // virtual
};

class D2 : public B             // Clase derivada-2
{
        public:  int fun (int x) {return x + 15;}        // virtual
};

int main(void)
{
        B b; D1 d1; D2 d2;
        cout << "El valor es: " << b.fun(10)     << endl;          // imprime 100
```

```
    cout << "El valor es: " << d1.fun(10)    << endl;        // imprime 20
    cout << "El valor es: " << d2.fun(10)    << endl;        // imprime 25
    cout << "El valor es: " << d1.B::fun(10) << endl;        // imprime 100
    cout << "El valor es: " << d2.B::fun(10) << endl;        // imprime 100
    return 0;
}
```

4.3 Puntero

Es una variable que puede almacenar la dirección en memoria de otra variable, además el puntero debe ser de igual tipo que la variable cuya dirección en memoria contiene, o dicho de otra forma, la variable a la que apunta.

Para acceder al valor que se encuentra en la dirección apuntada por un puntero se debe "dereferenciar" el puntero colocando * delante del mismo.

```
int *puntero;           // Declara un puntero a una variable tipo int

int variable;

variable = 123456;

puntero = &variable;                // Almacena la dirección de variable en
puntero

std::cout << puntero <<"\n";        /// muestra una dirección de memoria Ej.:
                                    0012FF60

std::cout << *puntero <<"\n";       ////  123456, se logra dereferenciar el puntero

std::cout << variable << "\n";      ////  123456
```

Nota: El operador & retorna una dirección de memoria, en este caso la dirección donde se encuentra la información del elemento denominado "variable".

4.4 Paso de argumentos por valor y referencia

Por valor: el paso por valor significa la función recibe una copia de los valores de los parámetros que se le pasan como argumentos. Las variables reales no se pasan a la función, sólo copias de su valor y no se ven modificadas luego de ejecutarse la operación.

```
void demo(int valor)        ///  prototipo de una función con argumentos pasados
                            por valor

{

    valor = 5;

}
```

```
void main()

{

        int n=10;

        demo(n);

        cout<<n<<endl;       ////   n continua con el valor de 10

}
```

Por referencia: Cuando una función debe modificar el valor de la variable pasada como parámetro, se debe pasar el parámetro por referencia. En este método, el compilador no pasa una copia del valor del argumento; en su lugar, pasa una referencia, que indica a la función dónde existe la variable en memoria.

```
void funcion(int& m)   ///  paso por referencia con el operador "&"
{
  m = m - 5;
}

int main()
{
  int a, b;
  a = 10;
  b = 20;

  funcion(a);
  cout << a << endl;         ///   a vale 5, su valor se vio afectado por la función
  return 0;
}
```

Nota: no se puede llamar a la función con parámetros constantes.

4.5 Iterador

Es una especie de puntero utilizado por un algoritmo para recorrer los elementos almacenados en un contenedor (vector, Map, list, etc.).
Como característica especial, acceden directamente a memoria para recorrer su contenedor asociado, haciéndolos muy eficientes.

```
map<CString,int>::iterator col = Map.find( "A" );       // se crea un iterador a map
                                                        iniciando en el elemento
                                                        solicitado

(*col).first;                    // Se accede al primer elemento "A", como un puntero
```

(*col).second;

4.6 Insertador

Mtl::matrix::inserter es el elemento usado para ingresar datos en una matriz, su principal función es acumular un conjunto de datos para luego insertarlos en su destrucción. Dado que la inserción de datos en matrices dispersas es lenta, este elemento es fundamental para agilizar dicho proceso, puesto que garantiza un número bajo de accesos directos a los datos de la matriz.

- matrix::inserter<Matrix> ins(m); ///// inserta y actualiza los valores dados

ins[0][0] << 8.0;
ins[0][0] << 4.0; ////// En la posición (0,0) de la matriz m esta el 4

- matrix::inserter<Matrix, update_plus<value_type> > ins(m, 3);
//// Suma el valor dado a la posición de matriz referenciada (es cero por defecto)

ins[0][0] << 8.0;
ins[0][0] << 4.0; ////// En la posición (0,0) de la matriz m esta el 12

5. Bibliografía

- Propuesta de un modelo estándar XML para intercambio de datos de análisis eléctrico. Andrés Fernando Bedoya Cadena
- C++ for Mathematicans. Edward Scheinerman.
- Análisis de Sistemas de Potencia. Grainger, Jhon J. Stevenson, William D., septiembre de 2004. ISBN 0-07-061293-5.
- W3C Recommendation, “Extensible Markup Language” Version 1.0, Octubre de 2000, Disponible en www.w3.org/TR/REC-xml.
- The Boost Graph Library, Disponible en www.boost.org.
- The Matrix Template Library, Disponible en www.osl.iu.edu/research/mtl/mtl4/
- Bicgstab (l) for linear equations involving unsymmetrical matrices with complex spectrum, Gerard L.G. Sleijpen and Diederik R. Fokkema. Volumen 1, pp. 11-32, Septiembre 1993, ISSN 1068-9613.
- Computational methods for electric power systems, Mariesa Crow.
- Sparse matrix techniques in power systems, Krishina M. Sambarapu and S. Mark Halpin. ISBN 1-4244-1051-7.
- Power flow, Gerald B. Sheblé, Iowa University.
- Power flow solution by Newton’s method, William F. Tinney and Clifford E. Hart, members IEEE.