Campus de São Carlos

"Implementação, em Ambiente Gráfico, de um
Algoritmo de Observabilidade em Redes de
Energia utilizando Vetores Esparsos"

Autor:

Claudio Eduardo Scriptori de Souza

Orientador:

Prof. Newton G. Bretas

UNIVERSIDADE DE SÃO PAULO

**ESCOLA DE ENGENHARIA
DE SÃO CARLOS**

✓

IMPLEMENTAÇÃO, EM AMBIENTE GRÁFICO, DE UM ALGORITMO DE OBSERVABILIDADE EM REDES DE ENERGIA UTILIZANDO VETORES ESPARSOS

CLÁUDIO EDUARDO SCRIPTORI DE SOUZA

Dissertação apresentada à Escola de Engenharia de São Carlos, da Universidade de São Paulo, como parte dos requisitos para a obtenção do Título de Mestre em Engenharia Elétrica.

ORIENTADOR: Prof. Newton G. Bretas

DEDALUS - Acervo - EESC

31100036506

**São Carlos
Fevereiro 1996**

Class.	Tare - EESC
Curr.	5828
Tombo	071/96

Eng. Elétrica

lt 0744530.

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca - EESC-USP

S730i Souza, Cláudio Eduardo Scriptori de
Implementação em ambiente gráfico, de um
algoritmo de observabilidade em redes de energia
utilizando vetores esparsos / Cláudio Eduardo
Scriptori de Souza. -- São Carlos, 1996.
98p.

Dissertação (Mestrado) -- Escola de Engenharia
de São Carlos-Universidade de São Paulo, 1996.
Orientador: Prof. Dr. Newton G. Bretas

1. Observabilidade. 2. Redes de energia. 3.
Programação Windows. 4. Linguagem C++. I. Título

FOLHA DE APROVAÇÃO

Dissertação defendida e aprovada em 28-2-1996
pela Comissão Julgadora:

Prof. Doutor **NEWTON GERALDO BRETAS** - Orientador
(Escola de Engenharia de São Carlos-Universidade de São Paulo)

Prof. Doutor **MARINHO GOMES DE ANDRADE FILHO**
(Instituto de Ciências Matemáticas de S.Carlos - Univ. de São Paulo)

Prof.^a Doutora **MARIA STELA VELUDO DE PAIVA**
(Escola de Engenharia de São Carlos - Universidade de São Paulo)

Presidente da Comissão de Pós-Graduação
Prof. Dr. **EDUARDO GLETO PIRES**

Coordenador da Área de Engenharia Elétrica
Prof.Dr. **GERALDO ROBERTO M. DA COSTA**

*À minha esposa **Denise** e ao meu filho **Thiago**,
que deram sentido a este trabalho e à minha vida.*

AGRADECIMENTOS

Ao Professor Newton G. Bretas pela excelente orientação oferecida durante a elaboração deste trabalho

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq, pela concessão da bolsa de estudos.

À minha mãe, Carmem Campoy Scriptori, e ao meu pai, Silas Mendonça de Souza, pelo apoio, cooperação, amor e carinho dedicados desde os tempos de graduação.

Ao colega Ricardo N. Mori pela ajuda e amizade.

SUMÁRIO

LISTA DE FIGURAS	i
LISTA DE TABELAS	ii
LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS	iii
RESUMO	v
ABSTRACT	vii
1. INTRODUÇÃO	1
2. OBSERVABILIDADE EM REDES DE ENERGIA.....	4
2.1. Estimção de Estado.....	4
2.2. Fatoração Triangular	7
2.2.1. Algoritmo de Observabilidade Utilizando Fatoração Triangular	9
2.3. Teoria de Observabilidade em Rede de Energia Baseada na Fatoração Triangular e no Conceito de Caminho de Grafo	11
2.3.1. Vetores Esparsos e Caminho de Fatoração: algumas propriedades	12
2.3.2. Medidas Irrelevantes e Ramos Não Observáveis	19
2.3.3. Adição de Medidas Utilizando Caminho de Fatoração	20
2.3.4. Algoritmo de Observabilidade Baseado na Fatoração Triangular e no Conceito de Caminho de Grafo.....	22
3. O PROGRAMA.....	29
3.1. Estrutura	29
3.2. Configuração dos Flag's no Barramento.....	31
3.3. Configuração das Coordenadas de Localização dos Barramentos.....	32
3.4. Visualização Gráfica da Rede de Energia.....	32
3.4.1. Algoritmo de visualização gráfica da rede de energia elétrica.....	36

4. TESTES REALIZADOS	39
4.1. Com 6 Barras.....	39
4.2. Com 14 Barras.....	41
4.3. Com 30 Barras.....	43
5. CONCLUSÃO.....	46
ANEXO I	49
ANEXO II	71
REFERÊNCIAS BIBLIOGRÁFICAS.....	96
BIBLIOGRAFIA SUPLEMENTAR	97

LISTA DE FIGURAS

Figura 2.1. - Exemplo de uma rede com 6 nós	26
Figura 2.2. - Ilhas observáveis da rede de seis nós	28
Figura 3.1. - Janela Principal do SIOC	30
Figura 3.2. - Flag's dos barramentos	31
Figura 3.3. - Configuração das coordenadas	32
Figura 3.4. - Fluxograma para construção dos barramentos gráficos	34
Figura 3.5. - Fluxograma para a conexão das linhas nos barramentos.....	35
Figura 4.1. - Diagrama Unifilar com 6 barras.....	39
Figura 4.2. - Resultado da análise de observabilidade para a retirada de fluxo na linha 5	40
Figura 4.3. - Diagrama Unifilar com 14 barras.....	41
Figura 4.4. - Resultado da análise de observabilidade para a retirada da medida de injeção do barramento 5	42
Figura 4.5. - Diagrama Unifilar com 30 barras.....	43
Figura 4.2. - Resultado da análise de observabilidade para a retirada da medida de injeção do barramento 5	46
Figura A1.1. - Janela típica do Windows	61
Figura A1.2. - Esquema de hierarquia das janelas.....	62
Figura A1.3. - Quadro de diálogo Arquivo Abrir	66
Figura A1.4. - Quadro de diálogo Arquivo Salvar.....	67
Figura A1.5. - Quadro de diálogo Entrada de Dados	67
Figura A1.6. - Quadro de diálogo Dados de Linha.....	68
Figura A1.7. - Quadro de diálogo Dados de Barramento	69

LISTA DE TABELAS

Tabela 3.1. - Valor do vetor ilha e suas cores correspondentes	33
Tabela 4.1. - Coordenadas "x" e "y" para o teste de 6 barras	40
Tabela 4.2. - Configuração das linhas para o teste de 6 barras.....	40
Tabela 4.3. - Coordenadas "x" e "y" para o teste de 14 barras	41
Tabela 4.4. - Configuração das linhas para o teste de 14 barras.....	42
Tabela 4.5. - Coordenadas "x" e "y" para o teste de 30 barras	44
Tabela 4.6. - Configuração das linhas para o teste de 30 barras.....	44
Tabela A1.1. - Flags Abrir/Salvar Arquivo	64

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

$\bar{h}(x)$	Vetor de equações não lineares que relaciona os estados às medidas
$\bar{x}$	Vetor dos estados verdadeiros do sistema
$\hat{x}$	Vetor estimado dos estados do sistema
$\bar{z}$	Vetor de medidas do estimador de estado
$\bar{w}$	Vetor dos erros de medidas
conec	Indicador de conexão entre as áreas
DC	Corrente Contínua
DOS	Sistema Operacional de Disco
FB	Fast Back - Volta rápida
FF	Fast Forward - Ida rápida
ff	Flag final
fi	Flag inicial
G	Matriz ganho do estimador de estado
GDI	Interface de Dispositivo Gráfico
GUI	Interface Gráfica do Usuário
H	Matriz jacobiana do estimador de estado
ilha	Vetor de identificação de ilha
it	Número de iterações
llig	Número da linha
lx	Localização do barramento - coordenada x
ly	Localização do barramento - coordenada y
mb	Medida de barra
MDI	Multiple Document Interface
ml	Medida de linha
nb	Número de barras
next	Próximo índice

nf	Nó final
ni	Nó inicial
nl	Número de linhas
nmed	Número de medidas
noex	Número externo da barra
nolig	Número da barra adjacente
npil	Índice inicial dos vetores nolig, lilig, next
PC	Personal Computer
RAM	Random-Access Memory (Acesso a memória randômica)
SIOC	Sistema Integrado de Operação e Controle
VGA	Adaptador de Vídeo Gráfico

RESUMO

A compreensão das dificuldades que envolvem o comportamento do sistema elétrico de potência tem sido cada vez mais importante e a estimação de estado do sistema é de fundamental importância para a sua operação adequada. Entretanto, torna-se necessário saber se o sistema é observável ou não.

Este trabalho, tem como objetivo desenvolver um software que possibilite a visualização gráfica da rede de energia como um todo, ou das sub-redes observáveis do sistema.

Como instrumento teórico foram utilizadas teorias e algoritmos desenvolvidos para a análise de observabilidade de redes do sistema de energia elétrica que utilizam a fatoração triangular da matriz ganho bem como o método de vetores esparsos.

O algoritmo desenvolvido por [2] foi implementado em ambiente gráfico do Windows, buscando a visualização gráfica do resultado numérico, exibido-a na tela do computador. Por ser essencialmente lógico, este algoritmo foi escolhido para a implementação. Como ferramenta, foi utilizado o compilador Borland C++ for Windows versão 4.0.

Os resultados dos testes realizados em redes de 6, 14 e 30 barramentos apontam para:

- 1) a simplificação da análise de observabilidade, utilizando vetores esparsos e fatoração triangular da matriz ganho;
- 2) a similiaridade de comportamento dos três sistemas testados, com fortes indícios de que a rotina desenvolvida funciona satisfatoriamente para qualquer sistema, principalmente para os sistemas de maiores quantidades de barramentos e de linhas;
- 3) a possibilidade de implementação de qualquer resultado numérico, em ambiente gráfico, aproveitando o programa aqui desenvolvido.

ABSTRACT

In the Operating Center of Electrical Energy system has been every time more and more important the understanding of the difficulties related to the electrical power behavior. In order to have adequate operations of the system the state estimation process is very important. However before performing the system state estimation one needs to know if the system is observable otherwise the estimation will not be possible.

This work has as main objective to develop a software that allows one to visualize the whole network in case that network is observable or the observable island of the entire network.

As theoretical background the theories and algorithm using the triangular factorization of gain matrix as well as the concepts contained on factorization path developed by [2] were used.

The algorithm developed by [2] was adapted to the Windows graphical form so that the numerical results of the network observability are shown in the computer screen in graphical form. This algorithm is essentially instead of numerical as the one based on the factorization of gain matrix only [1]. To implement the algorithm previously referred it was used the Borland

C++ compiler for windows version 4.0 due to the facilities for sources generation it presents.

The results of the tests in the network with 6, 14 and 30 bus leads to:

- 1) the simplification of observability analysis, using sparse vectors and triangular factorization of the gain matrix;
- 2) the behavior similarity of the three tested systems with effective clues that the routine developed works well for any systems mainly for systems with bigger quantities of bus and lines;
- 3) the alternative way of presenting numerical results using the program developed here in graphical forms.

1 - INTRODUÇÃO

O domínio dos problemas que envolvem o comportamento do sistema elétrico de potência nos Centros de Controle e Operação das Companhias de Energia Elétrica tem sido cada vez mais importante. Nesses centros existem vários programas especializados que auxiliam uma operação adequada do sistema, bem como uma expansão planejada do mesmo. Dentre estes, a estimação de estado é de capital importância.

A estimação de estado de um sistema consiste no processamento das magnitudes de tensões nos barramentos, do fluxo de potência ativa e reativa nas linhas, assim como das injeções de potência ativa e reativa nos barramentos, de maneira a obter a melhor estimativa do estado do sistema de potência. Entretanto, antes de se fazer a estimação dos estados, é necessário saber se o sistema é observável, isto é, se com o conjunto de medidas disponíveis é possível determinar a amplitude da tensão e o ângulo de fase para todos os barramentos do sistema. Assim, a observabilidade de uma rede depende tanto do número de medidas disponíveis, como de sua topologia, ou seja, de onde estas medidas estão localizadas. Às vezes, uma rede observável torna-se não observável quando há mudanças topológicas

na rede ou falhas na comunicação do sistema de medidas.

Este trabalho aborda duas teorias de análise de observabilidade de redes do sistema de energia elétrica. Em [3] é utilizado o método de fatorização triangular da matriz de estimação de estado dc (matriz ganho G) e, em [1] é utilizado o mesmo método de fatorização triangular além de, concomitantemente, valer-se do método de vetores esparsos. Este trabalho apresenta de maneira sumária os dois algoritmos desenvolvidos em [4] e [2] e, implementa somente o algoritmo desenvolvido em [2].

Os algoritmos anteriores fornecem como resultado se a rede é observável como um todo ou não. No caso desta não ser observável como um todo, são identificadas as porções da rede (sub-rede, ilha) que são possíveis de serem observadas de maneira isolada. Contudo, a forma de apresentação destes resultados é numérica e, por conseguinte, de difícil visualização. A necessidade de visualização dos resultados numéricos em ambiente gráfico é importante.

Desta forma, o objetivo deste trabalho é elaborar um software que possibilite a visualização da rede como um todo, no caso desta ser observável, ou das sub-redes observáveis do sistema através de um diagrama unifilar¹.

O programa aqui desenvolvido é próprio para utilização em empresas de energia elétrica, e tem como característica principal a manipulação de dados do sistema de potência de forma visual-gráfica. Ele

¹ Diagrama Unifilar é um esquema gráfico que se utiliza de símbolos para representar os detalhes de uma rede de energia elétrica.

projeta na tela do computador (monitor) o diagrama unifilar do sistema, bem como as medidas utilizadas, diferenciando as sub-redes, no caso do sistema não ser observável, com cores diferentes.

O programa foi desenvolvido através dos recursos de uma estação gráfica PC 80486, com 8 Mb de memória RAM, 365 Mb de disco rígido, monitor Super VGA colorido com resolução 0.28, executando o sistema operacional DOS versão 6.22 e sobre plataforma Windows versão 3.1. O código do programa foi gerado e compilado utilizando-se o software Borland C++ versão 4.0.

O capítulo 2 apresenta as teorias desenvolvidas por [1] e [3] e os algoritmos de [2] e [4], enquanto que a estrutura do programa desenvolvido neste trabalho é apresentada no capítulo 3. O capítulo 4 apresenta os testes realizados com 14 e 30 barramentos, assim como seus resultados e análises.

O código do programa, em linguagem C++, é apresentado detalhadamente, passo a passo, na sua íntegra, no Anexo II.

2 - OBSERVABILIDADE EM REDES DE ENERGIA

Antes de apresentar, de maneira simples e objetiva, as duas teorias de observabilidade anteriormente anunciadas, ou seja, a teoria de observabilidade desenvolvida por [1] e a desenvolvida por [3], é necessário que se faça um resumo do estimador de estado, tal como se segue.

2.1. - Estimação de Estado [3]

O estimador de estado tem como objetivo estimar a magnitude de tensões e o ângulo do sistema e isto só é possível se existirem medidas suficientes e adequadamente localizadas. Neste caso, se diz que o sistema é observável. Problemas com o sistema de telemedição podem reduzir o número de medidas disponíveis para a estimação de estado, fazendo com que o sistema seja parcialmente observável. Essa falta de medidas poderá ser suprida por pseudo-medidas.

As equações básicas que representam os componentes do sistema elétrico na análise de observabilidade são as mesmas do estimador de estado e do problema do fluxo de carga. A diferença fundamental entre

os dois é que a estimação de estado manipula dados redundantes e sujeitos a erros.

No que se segue, desenvolve-se de maneira sucinta o equacionamento do estimador de estado para efeito de observabilidade.

Num sistema de potência, a equação não linear relacionando às medidas e o vetor de estado deste é dado por:

$$\bar{Z} = \bar{h}(\bar{x}) + \bar{w} \quad (2.1)$$

onde:

$\bar{Z}$ é o vetor de medidas;

$\bar{x}$ é o vetor dos estados verdadeiros do sistema;

$\bar{h}(\bar{x})$ é o vetor de equações não lineares que relaciona os estados às medidas;

$\bar{w}$ é o vetor dos erros de medidas.

Como $\bar{h}(\bar{x})$ é uma função não linear, busca-se determinar $\bar{x}$ iterativamente, e estimar $\hat{\bar{x}}$ tal que minimize $J(\bar{x})$ dada por:

$$J(\bar{x}) = [\bar{Z} - \bar{h}(\bar{x})]^T W^{-1} [\bar{Z} - \bar{h}(\bar{x})] \quad (2.2)$$

onde W^{-1} é uma matriz de ponderação.

O valor estimado $\hat{\bar{x}}$ é obtido através da minimização de $J(\bar{x})$, ou seja:

$$\left. \frac{\partial J(\bar{x})}{\partial \bar{x}} \right|_{\bar{x}=\bar{x}^0} = 0, \quad (2.3)$$

onde, $\bar{x}^0$ é um ponto de operação.

Quando lineariza-se $\bar{h}(\bar{x})$ em torno deste ponto, temos:

$$\bar{h}(\bar{x}) \cong \bar{h}(\bar{x}^0) + H(\bar{x}^0)\Delta x \quad (2.4)$$

onde,

$$H(\bar{x}^0) = \left. \frac{\partial \bar{h}}{\partial \bar{x}} \right|_{\bar{x}=\bar{x}^0} \quad \text{e} \quad \Delta \bar{x} = \bar{x} - \bar{x}^0.$$

Com isso, a solução deste problema traduz-se em encontrar iterativamente $\Delta \hat{x}^k$, dado por:

$$G(\hat{x}^k)\Delta \hat{x}^k = H^T(\hat{x}^k)W[\bar{Z} - \bar{h}(\hat{x}^k)] \quad (2.5)$$

com

$$\hat{x}^{k+1} = \hat{x}^k + \Delta \hat{x}^k \quad (2.6)$$

e

$$G(\hat{x}^k) = H^T W H, \quad (2.7)$$

onde H é a matriz Jacobiana de $\bar{h}(\bar{x})$ e G a matriz ganho.

Usando o modelo linearizado de estimador de estado, a equação de estimação (2.7) resultará no ângulo das tensões nos barramentos da rede através da seguinte equação:

$$G\bar{\theta} = H_{P\theta}^T W_P \bar{Z}_P \quad (2.8)$$

com

$$G = H_{P\theta}^T W H_{P\theta} \quad (2.9)$$

onde o sub-índice P representa estar trabalhado somente com as potências

reais do sistema.

Neste caso, então, o vetor de estado $\bar{\hat{x}}$ transforma-se no vetor dos ângulos das tensões dos barramentos da rede. O que realmente importa para a observabilidade é se o sistema dado pela a equação (2.8) tem solução. Desta maneira é comum em observabilidade atribuir valores zero para todo o lado direito de (2.8), isto é:

$$G\bar{\theta} = \bar{0}. \quad (2.10)$$

2.2. - Fatoração Triangular [3]

Em [3] é apresentada uma teoria para observabilidade de rede baseada na fatoração triangular da matriz ganho G , utilizando o modelo linearizado de estimador de estado.

De acordo com essa teoria, o autor resolve o sistema (2.10) efetuando-se a triangularização de G para $G = H^T H$, onde a matriz jacobiana H é formada com impedâncias de linha que assumem valores unitários. Os pesos, também, assumem valores unitários.

Na forma gráfica, este processo transforma a equação (2.10) em:

$$\begin{array}{|c|} \hline \text{Área sombreada} \\ \hline \text{Círculo} \\ \hline \end{array} = \begin{array}{|c|} \hline \bar{\theta}_i \\ \hline \bar{\theta}_i \\ \hline \end{array} = \begin{array}{|c|} \hline \bar{0} \\ \hline \end{array}$$

(2.11)

onde a área sombreada corresponde a possíveis elementos não zero.

Observa-se que o sistema (2.11) apresenta múltiplas soluções, porque nas posições diagonais da forma triangular de G , correspondentes a θ_b , surgiram zeros. Para qualquer θ_b , por exemplo, $\theta_b = (0,1,2,\dots)^T$, é possível resolver a parte superior da equação produzindo um conjunto de valores para θ_a . Nesta situação, (θ_a, θ_b) é um estado não observável.

Para resolver esta dificuldade atribui-se valor 1 aos elementos da diagonal da matriz inferior direita de G , e valores $(0,1,2,\dots)^T$ correspondentes a θ_b , resultando assim no seguinte sistema de equações:

$$\begin{bmatrix} \text{triangular block} \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \bar{\theta}_b \\ \vdots \\ \bar{\theta}_b \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix}$$

(2.12)

É necessário notar que a equação (2.12), em verdade, corresponde às equações de estimação de estado com as medidas originariamente disponíveis acrescentadas das pseudo-medidas atribuídas ao vetor θ_b .

Como consequência, têm-se que a rede seria observável se o conjunto θ_b fosse constituído de um único elemento, ou seja, se na triangularização de G ocorrer um pivô zero na última posição da diagonal. Isto porque, neste caso, a única dependência para a solução do sistema de equações (2.12) seria a de tomar um ângulo como referência.

Em [3] é citado que para qualquer ordem dada na matriz ganho

G , sempre que um pivô zero for encontrado no processo de fatoração triangular, o correspondente θ pertence ao conjunto θ_b anterior e pode assumir um valor arbitrário, traduzindo assim num estado não observável. Em outras palavras, ao pivô zero em G é atribuído o valor 1, e o correspondente valor 0 do lado direito é substituído pelo valor correspondente da pseudo-medida θ . A fatoração triangular pode, então, continuar. A rede é observável se, e somente se, houver apenas um pivô zero na diagonal, o que necessariamente deve ocorrer na última posição desta diagonal. Quando a rede é não observável encontra-se mais de um pivô zero na fatoração triangular de G .

2.2.1. - Algoritmo de Observabilidade Utilizando Fatoração Triangular

Em [3] é apresentado um algoritmo que testa a observabilidade de uma rede e identifica ilhas observáveis (subconjunto da rede total) quando a rede não é observável. O autor aplica um algoritmo de observabilidade, primeiro para um modelo P- θ e, depois, para o modelo Q-V. A interseção do resultado das ilhas observáveis deste dois modelos é a ilha observável do sistema. Entretanto, normalmente ocorrem medidas de potências ativas e reativas aos pares; desta forma a segunda parte do algoritmo de observabilidade raramente será necessária.

Os passos do algoritmo são apresentados a seguir [3]:

Passo 1. Inicializar o conjunto de medidas de interesse como constando de

todas as medidas disponíveis.

Passo 2. Atualizar a rede de energia de interesse removendo todos os ramos que não tenham medida de fluxo nem mesmo medida de injeção nos seus nós terminais.

Passo 3. Formar a matriz ganho G_θ .

Passo 4. Efetuar a fatoração triangular de G_θ introduzindo θ -pseudo-medidas sempre que um pivô zero é encontrado. Parar somente quando um pivô zero surgir necessariamente no fim. Caso contrário prosseguir para o passo 5.

Passo 5. Resolva as equações do estimador de $G_\theta \hat{\theta} = H_{P\theta}^T W_P z_P$, para $\hat{\theta}$, considerando todos os valores das medidas iguais a zero, exceto para as θ -pseudo-medidas que assumem os valores $\theta_k = 0, 1, 2$, e assim por diante.

Passo 6. Estimar o valor do fluxo dos ramos $\hat{P}_{km} = x_{km}^{-1}(\hat{\theta}_k - \hat{\theta}_m)$, sendo $\hat{\theta}_k$ e $\hat{\theta}_m$ os componentes $k^{\text{ésimo}}$ e $m^{\text{ésimo}}$ de θ , para todos os ramos k-m dentro da rede de potência obtida através do passo 2. Parar se não houver fluxo de potência ($P_{km} = 0$). Caso contrário prosseguir para o passo 7.

Passo 7. Atualizar a rede de potência de interesse removendo todos os ramos k-m que tenham $P_{km} \neq 0$, ou seja, ramos inobserváveis.

Passo 8. Atualizar o conjunto de medidas de interesse, removendo as

medidas de injeções de potência, que vêm das barras adjacentes, para pelo menos um dos ramos removidos no passo 7.

Passo 9. Retornar ao passo 2.

Note-se que o algoritmo acima apresentado é iterativo, tal como é evidenciado no passo 9. A iteração ocorre porque as sub-redes identificadas podem somente ser classificadas, teoricamente, como candidatas às ilhas observáveis.

Deve-se salientar que, neste algoritmo, toda vez que alguma medida deva ser descartada, um processo de soluções algébricas (fluxo de carga linearizado) se torna necessário.

O algoritmo de observabilidade anteriormente apresentado foi testado nas redes utilizadas na literatura, para análise de observabilidade.

2.3. - Teoria de Observabilidade em Rede de Energia Baseada na Fatoração Triangular e no Conceito de Caminho de Grafo

Em [1], encontra-se desenvolvida uma teoria de observabilidade de rede baseada na fatoração triangular da matriz ganho G , bem como, no conceito de caminho de grafo desta matriz. A teoria desenvolvida analisa a observabilidade da rede, utilizando-se somente de informações já disponíveis nos centros de operação e não requer nenhuma sub-rotina extra, a não ser as próprias do estimador de estado e disponíveis nos centros de operação das empresas de energia.

A seguir apresentam-se algumas propriedades de vetores esparsos e de caminho de fatoração (factorization path).

2.3.1. - Vetores esparsos e caminho da fatoração: algumas propriedades [1]

Dado o sistema de equações lineares,

$$A\bar{x} = \bar{b} \quad (2.13)$$

onde A é uma matriz quadrada não singular, $\bar{b}$ é o vetor independente e $\bar{x}$ é o vetor de estado desconhecido, o que normalmente se deseja é a atualização do vetor de estado, quando algum b_i do vetor independente é alterado, efetuando-se somente as operações necessárias.

Em [1], é dada a solução para este problema. O caminho de fatoração é apresentado através do caminho de grafo e todas as suas informações estão contidas na fatoração triangular da matriz A .

Uma outra maneira de interpretar vetores esparsos é concluir como as variáveis associadas a eles estão interrelacionadas, ou mesmo se elas não apresentam nenhuma relação entre si.

Algumas vezes, em um estimador de estado de um sistema de potência ocorre a perda de medidas, fazendo-se necessário, então, a atualização do estimador de estado. Mas, antes é necessário saber se o estado do sistema pode ser estimado com o conjunto de medidas que estão disponíveis, isto é, se a rede é observável para este conjunto de medidas.

Para garantir que o estado do sistema possa ser estimado,

apresenta-se, a seguir, algumas propriedades do caminho de fatoraçoão associado à forma triangular da matriz ganho G [1]:

TEOREMA I:

Seja uma rede e associado a ela um grupo de medidas, tais que se pode escrever os seguintes n subconjuntos de equações:

$$A_1 \bar{x}_1 = \bar{b}_1, A_2 \bar{x}_2 = \bar{b}_2, \dots, A_n \bar{x}_n = \bar{b}_n \quad (2.14)$$

com o estado da rede

$$\bar{x}^T = [\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n], \quad (2.15)$$

o vetor independente

$$\bar{b}^T = [\bar{b}_1, \bar{b}_2, \dots, \bar{b}_n] \quad (2.16)$$

e a matriz

$$A = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_n \end{bmatrix} \quad (2.17)$$

com cada A_i sendo uma sub-matriz não singular. Com isso, a rede terá n caminhos de grafos, cada um deles associado a um subgrupo dessas equações e definidos de uma única maneira e desconectado do outro.

Conforme [1], cada sub-matriz terá um único caminho de grafo conectado, associado a ela, desde que esta sub-matriz seja assumida como

não-singular. Os caminhos de grafos associados à cada sub-matriz será separado do outro, visto que o subgrupo de variáveis a elas relacionados são subgrupos desacoplados.

TEOREMA II:

O caminho de fatoração associado ao vetor independente $\bar{b}$ de $G\bar{x} = \bar{b}$ quando efetuada a fatoração triangular da matriz ganho G , resulta num único caminho de grafo se, e somente se, o último elemento da diagonal for zero.

Quando um ângulo de fase da rede não é definido como referência, a triangularização de G reduz a matriz à seguinte forma:

A sub-matriz correspondente a figura anterior, sem a última linha e a última coluna, é não singular, tendo, assim, associado a ela um único caminho de grafo.

Pelo menos um dos elementos da última coluna de G precisa fazer conexão com o grafo daquela sub-matriz anterior, caso contrário aquela variável seria uma variável isolada do sistema. Como consequência,

a coluna associada àquela mesma variável em H não seria uma combinação linear das colunas anteriores de H , como mostrado em [1]. Assim, os caminhos de grafos associados àquela matriz só pode ser um caminho de grafo.

TEOREMA III:

No processo de triangularização da matriz ganho G , quando um pivô zero é encontrado, os nós restantes farão parte de outro caminho de grafo, sendo conexão com os caminhos de grafos anteriores.

Em [3], é demonstrado que quando se encontra um pivô zero no processo de triangularização da matriz, o restante da linha e da coluna em que este pivô é encontrado é zero, tal como se vê abaixo:

Neste caso, um único caminho de grafo é associado à sub-matriz já triangularizada (Teorema II). Como os elementos restantes desta linha são zero, não há maneira de conectar este nó a nenhum outro nó subsequente.

Pode-se afirmar, com este teorema, que a quantidade de caminhos de grafos está relacionado à quantidade de pivôs zeros

encontrada durante o processo de fatoração de G .

TEOREMA IV:

Quando, para o sistema de equações do Teorema I, A for substituída pela matriz ganho G , uma equação de medida é adicionada relacionando nós do mesmo caminho de grafo, embora haja alteração na fatoração triangular da matriz ganho, a quantidade de caminhos de grafos não sofre alteração.

Isto porque, a partir do momento em que esta nova medida não relaciona nenhuma variável de grafos distintos, ela não cria nenhum caminho de interconexão entre os caminhos de grafos existentes.

Observe-se que, em se tratando de medidas de potência, esta nova medida poderá ser tanto medida de fluxo, como medida de injeção.

TEOREMA V:

Se para o sistema de equações do Teorema I, com A substituída por G , for adicionada uma nova medida que relacione nós de caminhos de grafos distintos, duas situações poderão acontecer:

a) A nova medida somente relaciona duas variáveis (medida de fluxo ou equivalente), e então os dois caminhos de grafos originais, tendo duas de suas variáveis relacionadas por essa medida, se juntam formando

somente um caminho de grafo.

b) A nova medida relaciona mais de duas variáveis (medida de injeção), e:

b1) As variáveis pertencem somente a dois caminhos de grafos distintos, e então os dois caminhos de grafos originais, tendo suas variáveis relacionadas através desta medida, se juntam formando somente um caminho de grafo.

A medida de injeção nesta situação é equivalente a medida de fluxo, no sentido de informação de medida, criando um caminho entre aqueles caminhos de grafos isolados e unindo-os, como no caso a.

b2) As variáveis mencionadas acima pertencem a mais de dois caminhos de grafos diferentes, e então o número de caminhos de grafo isolado, na qual tem variáveis relacionadas por esta medida, será diminuído de um.

Em termos de matriz ganho G , cada caminho de grafo isolado terá uma variável livre. Se a nova medida de injeção adicionada apresentar três variáveis de três caminhos de grafos diferentes, o sistema apresentará duas variáveis a serem definidas, isto é, os três caminhos de grafos diferentes serão convertidos em somente dois caminhos de grafos separados. A nova medida contribuirá para definir uma das variáveis livres. Como resultado, o número de caminho de grafo será subtraído de um, sempre que uma medida de injeção for adicionada. Em outras palavras, para a observabilidade da rede como um todo, uma medida de injeção equivale a uma medida de fluxo, havendo diferenças entre elas

somente nos passos intermediários mas não no processo final.

2.3.1. - Observabilidade de Rede com Caminho de Fatoração [1]

A matriz ganho torna-se simples se os valores das medidas do estimador de estado linear forem ajustadas para zero, assim como todos os fatores peso e as impedâncias série das linhas forem ajustadas para um valor unitário. Com isso:

1. Se na fatoração triangular da matriz ganho G houver exatamente um caminho de grafo associado com aquela fatoração, o sistema será observável.
2. Se na fatoração triangular de G ocorrer mais de um caminho de grafo, e se:
 - a) não houver medidas de injeção conectando os nós de caminhos de grafos diferentes (mais que dois), então:
 - a1) o sistema, como um bloco, será não observável, porque resultou em mais de um zero na diagonal,
 - a2) as sub-redes associadas a esses caminhos de grafos constituirão ilhas observáveis desta rede;
 - b) houver medidas de injeção conectando os nós de caminhos de grafos distintos (mais que dois), então:
 - b1) o sistema como um bloco será não observável, porque foi encontrado mais de um zero na diagonal,

- b2) não é possível garantir se a rede associada com cada um dos caminhos de grafos isolados seja observável,
- b3) para encontrar possíveis sub-redes observáveis, é necessário identificar essas medidas e descartá-las, refatorando a nova matriz ganho; no caso desse grupo de medidas não existir mais, o caminho de grafo associado com esta nova matriz fatorada será indicativo das sub-redes observáveis do sistema.

2.3.2. - Medidas Irrelevantes e Ramos Não Observáveis

Suponha-se um estado não observável $\bar{\theta}$, obtido através de uma metodologia qualquer. Organiza-se $\bar{\theta}$ de forma que os componentes associados aos subgrupos isolados que são observáveis sejam agrupados em conjuntos desacoplados. Suponha-se que eles formem três grupos de valores, $\bar{\theta}_\alpha$, $\bar{\theta}_\beta$, $\bar{\theta}_\gamma$, isto é, $\bar{\theta} = (\bar{\theta}_\alpha, \bar{\theta}_\beta, \bar{\theta}_\gamma)$. A sub-rede $\bar{\theta}_\alpha$ é composta dos seus próprios nós e dos ramos conectados a eles, aplicando-se o mesmo para as sub-redes $\bar{\theta}_\beta$ e $\bar{\theta}_\gamma$.

Agrupa-se as medidas associadas aos três grupos $\bar{\theta}_\alpha$, $\bar{\theta}_\beta$ e $\bar{\theta}_\gamma$ de uma maneira isolada. E, coloca-se também, consecutivamente, as medidas (injeções) onde os nós conectados por elas pertençam a sub-redes diferentes. Na forma de equação tem-se:

$$\begin{array}{|c|c|c|} \hline H_\alpha & & \\ \hline & H_\beta & \\ \hline & & H_\gamma \\ \hline & & H_{\alpha\beta\gamma} \\ \hline \end{array}
 \begin{array}{|c|} \hline \theta_\alpha \\ \hline \theta_\beta \\ \hline \theta_\gamma \\ \hline \end{array}
 =
 \begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

Com isso, duas questões podem ser adequadas:

- Como encontrar os subgrupos $\overline{\theta}_\alpha$, $\overline{\theta}_\beta$ e $\overline{\theta}_\gamma$ sem precisar

resolver $G\overline{\theta} = \overline{0}$?

- Isto é possível?

Sim, é possível. Os nós dos subgrupos $\overline{\theta}_\alpha$, $\overline{\theta}_\beta$ e $\overline{\theta}_\gamma$ serão constituídos por aqueles nós pertencentes aos três caminhos de grafos distintos associados à forma triangular de G . O conjunto de medidas correspondentes a $H_{\alpha\beta\gamma}$ será constituída pelas medidas (injeções) que relacionem as variáveis dos caminhos de grafos distintos.

Os ramos não observáveis serão aqueles que não participam de nenhum dos caminhos de grafos existentes. Assim, essas medidas são irrelevantes no que diz respeito a identificação de possíveis sub-redes observáveis. De qualquer modo, a proposta principal é a de transformar a rede numa rede observável como um todo, e nesse sentido elas não são irrelevantes.

2.3.3. - Adição De Medidas Utilizando Caminho De Fatoração

Quando se analisa observabilidade de rede, três situações

diferentes poderão acontecer:

1. A primeira situação desejável, é que a rede seja observável em relação ao grupo de medidas.
2. A segunda situação é obter uma rede com vários caminhos de grafos associados a ela, e sem nenhuma medida de injeção que relacione nós de sub-redes, associados a caminhos de grafos distintos.
3. A terceira situação ocorre quando o sistema é composto de diferentes caminhos de grafos e tem medidas de injeção que relacione nós das sub-redes, associados com caminhos de grafos distintos.

Neste caso, o que se pretende, é saber qual a menor quantidade de medidas a ser adicionada ao conjunto de medidas já existentes, para tornar a rede observável como um todo.

A solução para as segunda e terceira situações é exatamente a adição de medidas de fluxo ou de injeção que relacione nós de todos os caminhos de grafos distintos, ligando-os entre si. Esses caminhos de grafos interligados por estas medidas, serão transformados num único caminho de grafo e a rede associada a ele será uma rede observável. Neste caso, as medidas adicionadas não contaminarão o resultado do estimador de estado. Elas forçarão as sub-redes a terem o mesmo ângulo de fase como referência. Como consequência, a rede como um todo, pode tornar-se observável num único passo, evitando-se qualquer refatoração de matriz. Não é nem mesmo necessário conhecer as possíveis sub-redes observáveis que possam existir. Contudo, os fluxos dos ramos não observáveis

dependerão da qualidade das medidas quando adicionadas.

A seguir, apresenta-se o algoritmo desenvolvido por [2], que se utiliza da teoria acima apresentada.

A razão de se implantar este algoritmo e não o de [4], baseia-se no fato de que ele constitui-se essencialmente de princípios lógicos, não requerendo em momento algum solução de qualquer sistema algébrico. Com isto, este algoritmo, torna-se significativamente mais rápido.

2.3.4. - Algoritmo De Observabilidade Baseado Na Fatoração Triangular e No Conceito De Caminho De Grafo [2]

Em [2] é apresentado um algoritmo baseado na teoria desenvolvida pelo próprio autor. As sub-redes observáveis da rede serão constituídas pela interseção das sub-redes correspondentes aos modelos P- θ e Q-V.

Os passos do algoritmo são os seguintes:

Passo 1. Com os grupos de medidas disponíveis, formar a matriz H .

Passo 2. Formar a matriz ganho G .

Passo 3. Realizar a fatoração triangular de G .

Passo 4. Encontrar os caminhos de fatoração associados com a referida

triangularização. No caso de se obter apenas um único caminho de grafo, a rede é observável. Pare. Caso contrário prosseguir para o passo 5.

Passo 5. Se ocorrer mais de um caminho de grafo associado à referida fatoração, então a rede é não observável como um todo.

O algoritmo, tal como apresentado acima, requer a identificação dos caminhos de grafos associados à forma triangular da matriz ganho G . Apresenta-se, a seguir, um esquema para encontrar estes caminhos.

Esquema para encontrar os caminhos de fatoração de $G(U)$.

Define-se as variáveis:

$V_{next}(i)$ - é o elemento de valor mais baixo, não zero, na linha "k" de U .

($i = 1, \dots, N$, com N = número de barras).

$$V_{diag}(i) = \begin{cases} 1 - & \text{que significa um elemento da diagonal não nulo} \\ 0 - & \text{que significa um elemento da diagonal nulo} \end{cases} \quad i = 1, \dots, N;$$

$V_{color}(i)$ - cores (zero para todas as barras), com uma variável de cor auxiliar.

O algoritmo para o esquema acima segue os seguintes passos:

Passo 1. Definir $St_{bus} = 1$ (inicializando a barra).

Passo 2. Fazer $Pos = St_{bus}$.

Passo 3. Fazer $k = V_{next}(Pos)$

Passo 4. Checar $V_{color}(k)$

- Se diferente de 0, fazer $V_{color}(n) = V_{color}(Pos)$
 $(n = St_{bus}, \dots, Pos)$ e ir para 5
- Se igual a 0 e
 - $V_{dig}(k) \neq 0$, fazer $Pos = k$, e ir para 3. Senão,
 - $V_{dig}(k) = 0$, fazer $Color = Color + 1$ e $V_{color}(n) = Color$,
 $(n = St_{bus}, \dots, k)$

Passo 5. Procurar por $V_{color}(n) = 0$ ($n = St_{bus}, \dots, N$) - se no mínimo um deles for zero, fazer $St_{bus} = n$,

Passo 6. Se $n = N$, pare. Senão, retorne ao passo 2.

Com este algoritmo, todos os nós de "cores" iguais pertencerão

ao mesmo caminho de grafo.

Este processo é extremamente rápido visto que não implica em nenhum cálculo, e as variáveis envolvidas são diretamente indexadas. Além disso, os valores das variáveis sempre são valores inteiros.

Segue-se um esquema para encontrar as possíveis sub-redes observáveis da rede inteira:

Identificação das Sub-redes Observáveis [2]:

Passo 1. Na medida em que se obtém mais de um caminho de grafo associado à triangularização de G , e:

a) se não há medidas (injeções) relacionando nós de diferentes caminhos de grafos, então as sub-redes associadas a cada caminho de grafo constituirá sub-redes observáveis. Pare.

b) se há medidas de potência (injeções) relacionando nós de diferentes caminhos de grafos, então nada pode ser dito sobre a observabilidade da sub-rede associada com esses caminhos de grafos. Prosseguir para o passo 2.

Passo 2. Identificar essas medidas e removê-las do grupo original de medidas. Observar que há medidas irrelevantes com relação a estimação de estado.

Passo 3. Atualizar a fatorização triangular.

Passo 4. Retornar ao passo 1.

O algoritmo poderá ser um processo iterativo no caso de acontecer a situação b do passo 1. Isto se deve ao fato de que quando se identificam as medidas irrelevantes e elas são descartadas, outras medidas irrelevantes poderão surgir.

Na figura 2.1 é exemplificado o algoritmo com uma rede de seis nós para ilustrar como o mesmo funciona. Para isto, todos os ramos de impedância são iguais a $j1$ e todos os pesos são iguais a 1.

Figura 2.1. - Exemplo de uma rede com 6 nós

Passo 1. O grupo de medidas de interesse é $\{P_1, P_2, P_3, P_4\}$.

Passo 2. A matriz ganho G é obtida:

	1	2	3	4	5	6	
$G =$	5	-3	-2				1
	-3	2	1				2
	-2	1	2	-4	1	1	3
			-4	10	-4	-3	4
			1	-4	2	1	5
			1	-3	1	1	6

Passo 3. Efetua-se a fatoração triangular:

$$U = \begin{array}{c} \begin{array}{cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{array}{|c|c|c|c|c|c|} \hline 1 & -3/5 & -2/5 & & & \\ \hline & 1 & -1 & & & \\ \hline & & 1 & -3 & 1 & 1 \\ \hline & & & 1 & -1 & 0 \\ \hline & & & & 0 & 0 \\ \hline & & & & & 0 \\ \hline \end{array} & \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \end{array} \end{array}$$

Neste caso dois caminhos de grafos associados à triangularização ocorreram (dois pivôs zero); eles são os grafos associados aos nós: $\{1,2,3,4,5\}$ e $\{6\}$; significando que o sistema é dividido em duas ou mais ilhas observáveis. Um nó isolado é considerado uma ilha de um só nó. O pivô zero ocorre nos nós 5 e 6, como se pode observar.

Procurando por Ilhas Observáveis

Passo 1. Visto que há mais de um caminho de grafo associado a fatorização de G , é necessário procurar por possíveis medidas de injeção referentes as variáveis (nós) destes caminhos de grafos.

Passo 2. Efetuando-se a procura, observa-se que P_2 é uma medida que se refere às variáveis dos caminhos de grafos 1 e 2. Deste modo, esta medida deve ser removida.

Passo 3. Atualiza-se a fatoração triangular da matriz ganho G .

Passo 4. Ocorrem três caminhos de grafos associados a fatoração; eles são os grafos relativos aos nós: $\{1,2,3\}$, $\{4,5\}$ e $\{6\}$.

Passo 5. Não existem mais medidas (injeção) referentes aos nós de diferentes caminhos de grafos. Assim, a rede toda é constituída por três sub-redes observáveis, como mostra a figura 2.2:

Figura. 2.2. - Ilhas observáveis da rede de seis nós

3 - O PROGRAMA

Considerando os algoritmos de observabilidade já descritos e utilizando-se do ambiente Windows, foi elaborado o programa SIOC que é apresentado a seguir.

3.1. - Estrutura do Programa

O programa inicia com uma janela principal do tipo Tapplication¹ normal sobreposta, com uma barra de título, um quadro do sistema à esquerda da barra de título, ícones de maximização e minimização à direita da barra de título, uma barra de menu, uma moldura para a alteração do tamanho, uma barra de "status" na parte inferior da janela e uma barra de ferramentas abaixo da barra de menu.

Pode-se observar a sua estrutura na figura abaixo:

¹ TApplication é derivada de TModule; TApplication tem sua atuação padrão orientada para objeto em uma aplicação Windows. Geralmente é utilizada para aplicação de janela mãe ou janela principal.

Figura 3.1. - Janela Principal do SIOC

A partir da janela principal são gerenciados todos os processos do programa, como: abrir arquivo, salvar arquivo, sair do programa, alterar dados do banco de dados, visualizar resultados.

O programa permite que o usuário crie um banco de dados do sistema que ele desejar. Este banco de dados contém informações sobre o número de barras (nb), o número de linhas (nl), um vetor que indica a barra inicial (ni), um vetor que indica a barra final (nf) das linhas do sistema, um vetor que indica as medidas na linha (ml), um vetor que indica o flag inicial (fi) e outro que indica o flag final (ff) de cada conexão, um vetor que indica as medidas nos barramentos (mb) e os vetores de localizações (lx e ly).

Nos itens abrir arquivo, salvar arquivo e alterar dados, são utilizados quadros de diálogos que representam, para o sistema Windows, uma maneira padrão de receber entrada de controle do usuário. Para uma compreensão mais detalhada deste programa (SIOC), o Anexo I descreve

os passos seguidos para sua construção, motivo pelo qual recomenda-se uma atenta leitura e devida apreciação do mesmo.

3.2. - Configuração dos Flag's no Barramento

Cada barramento contém seis flag's de conexões possíveis. Estes flag's disponíveis são fixos e seguem caminhos específicos, que satisfazem todas as direções possíveis para a conexão com o próximo barramento.

Figura 3.2. - Flag's dos barramentos

A Figura 3.2 mostra os Flag's disponíveis de cada barra e sua característica.

Conforme se pode observar, há seis flag's disponíveis, sendo três superiores e três inferiores. Os valores dos campos Flag Inicial e Flag Final dependem diretamente dos valores de entrada e saída dos Flag's. De acordo com a conveniência, conectamos os flag's iniciais e finais aos valores fornecidos.

3.3. - Configuração das Coordenadas de Localização dos Barramentos

As coordenadas estão em valores de pixels e são fixadas no canto superior esquerdo do barramento, em relação às bordas da janela:

Figura 3.3. - Configuração das coordenadas

Cada usuário determinará como se configurará seu sistema, tendo, portanto, livre acesso a estes valores.

Para manter um padrão, determinou-se que a primeira linha e a primeira coluna de barramentos se iniciem a uma distância de 40 pixels horizontais e 40 pixels verticais das bordas e com distância entre eles de 80 pixels, tanto vertical como horizontal.

É de grande importância salientar que o usuário tem livre acesso na configuração da rede.

3.4. - Visualização Gráfica da Rede de Energia

O algoritmo de observabilidade desenvolvido por [2] e adaptado

para este trabalho fornece um vetor chamado "ilha", que informa o sub-grupo observável a que o barramento pertence.

Como já foi visto, os barramentos pertencentes ao mesmo sub-grupo formam uma ilha observável, podendo ser demonstradas graficamente. Este é o interesse primordial deste trabalho.

Os valores contidos neste vetor (ilha) foram relacionados a uma cor independente. Foram criados 16 arquivos bitmap's de barramentos idênticos em sua dimensão e de cores diferentes. As cores correspondentes constam da tabela 3.1.

Tabela 3.1. - Valor do vetor ilha e suas cores correspondentes

Valor	Cor	Valor	Cor
1	Preto	9	Cinza Escuro
2	Azul Claro	10	Mostarda
3	Lilás	11	Roxo
4	Azul	12	Vermelho Escuro
5	Amarelo	13	Verde Musgo
6	Vermelho Claro	14	Verde Escuro
7	Verde Claro	15	Azul Escuro
8	Cinza Claro	16	Branco

Para o algoritmo construir o diagrama unifilar, há que identificar o valor da ilha de cada barramento. Tendo o valor da ilha, o algoritmo transfere o arquivo bitmap correspondente na posição especificada pelo usuário, através das coordenadas do barramento, da seguinte forma:

Fig. 3.4. - Fluxograma para construção dos barramentos gráficos

Pode-se observar que em havendo mais de 15 ilhas observáveis, o que é muito difícil de acontecer, a cor correspondente a estas ilhas é branca, cujo valor assumido é 16. Isso fará com que o barramento não seja

apresentado já que o fundo da área de apresentação é da mesma cor.

Após serem colocados todos os barramentos, há que se conectá-los conforme suas ligações, o que é feito da seguinte maneira:

Fig. 3.5. - Fluxograma para a conexão das linhas nos barramentos

A função *DrawLinha()* é que faz a conexão entre os barramentos.

Seu desenvolvimento depende diretamente dos valores dos flag inicial ($fi[i]$) e final ($ff[i]$). $lx[m]$ e $ly[m]$ são os valores das coordenadas do barramento inicial e $lx[n]$ e $ly[n]$ são os valores das coordenadas do barramento final. Estes valores correspondem ao canto superior esquerdo do barramento.

O algoritmo da função *DrawLinha()* é apresentado a seguir:

3.4.1. - Algoritmo de visualização gráfica da rede de energia elétrica

O algoritmo apresenta os seguintes passos:

Passo 1. Verificar a cor dos barramentos a serem conectados através da variável *corLinha* e selecionar o pincel correspondente.

Passo 2. Testar os valores das coordenadas dos barramentos e atribuir um valor a uma variável chamada *teste*, conforme os seguintes critérios:

Situação	Teste
0
1
2
3
4
5
6
7
$bi = bf$	8

onde "bi" significa Barramento Inicial e "bf" Barramento Final

Passo 3. Testar o valor do flag inicial e traçar uma linha, salvando as coordenadas finais numa variável denominada "pf", da seguinte forma:

Flag	Posição
1
2
3
4
5
6

Passo 4. Testar o valor do flag final, mover o ponteiro para a coordenada desejada e traçar uma linha da seguinte forma:

Flag	Posição
1
2
3
4
5
6

Passo 5. Testar a variável teste e traçar uma linha conectando as linhas dos passos 2 e 3, utilizando a coordenada da variável "pf".

Fazendo "cx" igual a coordenada "x" do ponto "pf", traça-se uma linha até as coordenadas "cx" e "cy". Manter "cy" igual a coordenada "y" do ponto "pf".

Como o último ponto de parada foi no final da linha traçada para o flag "ff", continua-se a linha até a coordenada x da variável "pf" e até a coordenada y da variável "pf", unindo as duas linhas anteriormente traçadas. Podemos observar que para este primeiro estágio não importa o valor da variável teste, com exceção do valor 8 que significa a sobreposição das barras, o que ocasiona um erro.

Desta forma, demonstrou-se o desenvolvimento do trabalho proposto. O interesse deste trabalho, não é discutir a linguagem de programação utilizada, mas sim a idéia para solucionar o problema de visualização gráfica da rede de energia. De qualquer forma, as dúvidas sobre a linguagem computacional utilizada, poderão ser sanadas através do Anexo I, do Anexo II e nas bibliografias suplementares citadas.

4 - TESTES REALIZADOS

Foram realizados testes com 6, 14 e 30 barras, cujos resultados e respectivas análises constam deste capítulo.

4.1. - Com 6 Barras

Este teste é composto de 6 barras e de 16 linhas, conectado da seguinte maneira:

Figura 4.1 - Diagrama Unifilar com 6 barras

Os dados dos barramentos da Figura 4.1 são as seguintes:

Tabela 4.1 - Coordenadas 'x' e 'y' para o teste de 6 barras

BARRA	MB	CX	CY	BARRA	MB	CX	CY
Barra 1	1	40	40	Barra 4	1	280	120
Barra 2	0	40	200	Barra 5	0	360	120
Barra 3	0	120	120	Barra 6	1	280	200

Mais uma vez, salienta-se que estas coordenadas estão em pixel e são tomadas a partir dos cantos superiores esquerdo da moldura de visualização do diagrama.

As linhas estão configuradas da seguinte maneira:

Tabela 4.2 - Configuração das linhas para o teste de 6 barras

LINHA	ML	BI	BF	FI	FF	LINHA	ML	BI	BF	FI	FF
Linha 1	1	1	2	2	5	Linha 4	0	3	4	3	1
Linha 2	0	1	3	3	5	Linha 5	1	4	5	3	1
Linha 3	0	3	2	1	6	Linha 6	0	4	6	2	5

Retiramos a medida de fluxo da linha 5 e observamos o seguinte:

Figura 4.2 - Resultado da análise de observabilidade para a retirada da medida de fluxo na linha 5

Podemos observar que a rede é dividida em duas ilhas observáveis

composta pelos barramentos {1; 2; 3} e {4; 6} e uma ilha não observável composta pelos barramentos {5}.

4.2 - Com 14 Barras

Este teste é composto de 14 barras e de 16 linhas, conectado da seguinte maneira:

Figura 4.3 - Diagrama Unifilar com 14 barras

Os dados dos barramentos da Figura 4.3 são as seguintes:

Tabela 4.3 - Coordenadas 'x' e 'y' para o teste de 14 barras

BARRA	MB	CX	CY	BARRA	MB	CX	CY
Barra 1	1	40	280	Barra 8	0	120	200
Barra 2	1	40	380	Barra 9	1	280	120
Barra 3	1	280	380	Barra 10	0	120	120
Barra 4	1	280	280	Barra 11	1	200	120
Barra 5	1	120	280	Barra 12	1	40	40
Barra 6	1	40	120	Barra 13	1	120	40
Barra 7	0	200	200	Barra 14	0	280	40

Vale salientar que estas coordenadas estão em pixel e são tomadas a partir dos cantos superiores esquerdo da moldura de visualização do diagrama.

As linhas estão configuradas da seguinte maneira:

Tabela 4.4 - Configuração das linhas para o teste de 14 barras

LINHA	ML	BI	BF	FI	FF	LINHA	ML	BI	BF	FI	FF
Linha 1	0	12	6	2	5	Linha 11	1	9	7	2	5
Linha 2	0	12	13	3	1	Linha 12	1	9	4	3	6
Linha 3	0	13	6	2	6	Linha 13	1	7	4	2	5
Linha 4	0	13	14	3	1	Linha 14	0	1	5	3	1
Linha 5	0	14	9	2	5	Linha 15	0	5	4	6	4
Linha 6	0	6	10	3	1	Linha 16	1	1	2	1	4
Linha 7	0	10	11	3	1	Linha 17	0	5	2	2	5
Linha 8	0	11	9	3	1	Linha 18	0	4	2	1	6
Linha 9	0	6	5	2	5	Linha 19	0	4	3	2	5
Linha 10	1	8	7	3	1	Linha 20	0	2	3	2	2

Retiramos a medida de injeção do barramento 5 e observamos o seguinte:

Figura 4.4 - Resultado da análise de observabilidade para a retirada da medida de injeção do barramento 5

Podemos observar que a rede é dividida em uma ilha observável composta pelos barramentos {1; 2; 3; 4; 5; 7; 8; 9} e uma ilha não observável composta pelos barramentos {6; 10; 11; 12; 13; 14}.

Tal resultado também foi encontrado por [4], onde o autor mostra que para as medidas originais a rede como um todo é observável, e ao ser retirada a medida do barramento 5 encontrou-se o mesmo resultado do que apresentado acima.

4.3. - Com 30 Barras

Este teste é composto de 30 barras e de 30 linhas, conectado da seguinte maneira:

Figura 4.5 - Diagrama Unifilar do Teste de 30 Barras

As coordenadas "x" e "y" para a distribuição dos barramentos conforme a Figura 4.5 são as seguintes:

Tabela 4.5 - Coordenadas 'x' e 'y' para o teste de 30 barras

BARRA	MB	CX	CY	BARRA	MB	CX	CY
Barra 1	1	40	40	Barra 16	1	280	280
Barra 2	1	200	40	Barra 17	1	440	280
Barra 3	0	360	40	Barra 18	1	600	280
Barra 4	1	520	40	Barra 19	1	40	360
Barra 5	1	680	40	Barra 20	0	200	360
Barra 6	0	120	120	Barra 21	0	360	360
Barra 7	0	280	120	Barra 22	0	520	360
Barra 8	0	440	120	Barra 23	0	680	360
Barra 9	1	600	120	Barra 24	0	120	440
Barra 10	0	40	200	Barra 25	1	280	440
Barra 11	0	200	200	Barra 26	0	440	440
Barra 12	0	360	200	Barra 27	0	600	440
Barra 13	0	520	200	Barra 28	1	200	520
Barra 14	0	680	200	Barra 29	0	360	520
Barra 15	0	120	280	Barra 30	1	520	520

As linhas estão configuradas da seguinte maneira:

Tabela 4.6 - Configuração das linhas para o teste de 30 barras

LINHA	ML	BI	BF	FI	FF	LINHA	ML	BI	BF	FI	FF
Linha 1	0	1	2	3	1	Linha 16	1	12	21	2	5
Linha 2	0	2	3	3	1	Linha 17	1	13	17	2	5
Linha 3	0	2	4	5	5	Linha 18	1	14	18	1	5
Linha 4	1	3	8	3	5	Linha 19	0	15	19	2	5
Linha 5	0	4	5	3	1	Linha 20	0	16	20	2	5
Linha 6	1	5	14	3	6	Linha 21	0	17	22	2	5
Linha 7	0	1	6	2	5	Linha 22	0	18	23	2	5
Linha 8	0	6	7	3	1	Linha 23	1	19	24	2	5
Linha 9	1	6	10	1	5	Linha 24	1	20	21	2	2
Linha 10	1	7	11	3	5	Linha 25	1	22	27	2	5
Linha 11	1	8	12	1	5	Linha 26	0	24	25	3	1
Linha 12	1	8	9	3	1	Linha 27	1	25	29	2	5
Linha 13	0	9	13	3	5	Linha 28	1	26	27	3	1
Linha 14	1	10	15	2	5	Linha 29	0	28	29	3	1
Linha 15	1	11	16	2	5	Linha 30	0	29	30	3	1

Retiramos a medida de injeção do barramento 5 e observamos o seguinte:

Figura 4.6 - Resultado da análise de observabilidade para a retirada da medida de injeção do barramento 5

Podemos observar que a rede é dividida em três ilhas observáveis composta pelos barramentos {6; 10; 15; 19; 24; 25; 28; 29; 30}, {3; 7; 8; 9; 11; 12; 13; 16; 17; 20; 21; 22; 26; 27} e {5; 14; 18; 23} e uma ilha não observável composta pelos barramentos {1; 2; 4}. Na realidade pode-se considerar três ilhas independentes, cada uma contendo apenas um barramento.

5 - CONCLUSÃO

Este trabalho consiste em uma demonstração clara e objetiva de como a apresentação de resultados numéricos das redes de energia, em ambiente gráfico, é importante. Desta forma, a análise e interpretação da observabilidade em redes de energia bastante didática e de fácil entendimento para diferentes níveis de operadores do sistema.

Neste trabalho se enfatiza o quanto a análise de observabilidade utilizando vetores esparsos e fatoração triangular da matriz ganho é simplificada. Isto se dá pelo fato de não exigir nenhum cálculo extra, além dos corriqueiros para a estimação de estado, bem como nenhuma subrotina especial. O único passo que poderá levar o sistema a ser iterativo é o correspondente à retriangularização da matriz ganho, isto em caso de descarte de medidas. Dessa forma, pode-se afirmar que o método de observabilidade aqui apresentado é essencialmente lógico em vez de numérico. Este é o motivo pelo qual foi implementado o algoritmo desenvolvido por [2].

A rotina gráfica desenvolvida neste trabalho é simples e eficiente, porque o caminho efetuado por ela relaciona somente coordenadas x e y ,

não envolvendo nenhum banco de dados de caminhos pré-definidos e nenhum cálculo vetorial. O que se tem é um caminho fixo, pré-definido e amarrado às entradas e saídas dos barramentos, que por sua vez, também são pré-fixados. Todavia, pode-se melhorar esta rotina com métodos de busca e heurísticas complementares. Porém, quanto mais aperfeiçoada for a rotina de busca, mais consistente o sistema ficará, tirando do usuário o poder de alteração do sistema gráfico. Tal como está apresentado neste trabalho, o sistema coloca todo o controle à disposição do usuário.

As vantagens de se ter todo o processo em ambiente gráfico consistem em:

- 1) ter uma visualização global do comportamento de todo o sistema de energia elétrica. Na medida que é apresentado o resultado de observabilidade para o grupo de medidas disponíveis, tem-se diretamente na tela do computador a visualização das ilhas observáveis de todo o sistema, facilitando, assim, a análise pelo usuário;
- 2) ter uma interação mais rápida com o sistema e poder alterar dados diretamente. No momento em que é apresentado o resultado de observabilidade para o grupo de medidas disponíveis, sendo possível adicionar e/ou retirar medidas para uma nova situação.

Fica como sugestão de continuidade ao projeto, a apresentação, em ambiente gráfico, da análise de fluxo de carga e estimação de estado que podem ser feitas de maneira similar ao trabalho aqui desenvolvido. Em

outras palavras, os resultados do fluxo de carga e do estimador de estado podem ser implementados no gráfico correspondente à rede de energia.

ANEXO I - PROGRAMAÇÃO WINDOWS

Segundo PETZOLD (1993), a programação para o Windows implica em uma escolha do tipo tudo ou nada. Não se pode criar um aplicativo MS-DOS, e usar o Windows somente nos gráficos. Quando se quer usar o Windows, é necessário comprometer-se em criar um aplicativo inteiramente voltado para o Windows.

No Windows tudo está interconectado. Para se desenhar um gráfico na tela do vídeo, necessita-se de algo chamado "*handle* do dispositivo de contexto". Para obtê-lo, é necessário o "*handle* da janela". Para obter este último, é preciso criar uma janela e estar preparado para receber "mensagens". Para receber e processar as mensagens, há que se ter um "procedimento de janela". É assim que se começa a codificar um programa para Windows.

Linguagem C++

À medida em que os programas aumentam de tamanho e complexidade, os programadores costumam desdobrá-los em partes

menores e mais gerenciáveis chamados funções. Cada função deve realizar uma tarefa específica. Quando esta tarefa precisa ser realizada, basta chamar a função e fornecer a ela a informação necessária para seu processamento. Observa-se, porém, a necessidade de acessar várias funções paralelamente aos dados locais, e uma maneira de se fazer isso é através do *objeto*, que tem a tarefa de juntar dados e código do programa numa simples função.

A programação orientada para objeto junta tanto dados quanto funções, fazendo com que os detalhes internos das partes complexas do programa fiquem simplesmente fora de vista. A linguagem C++ é uma ferramenta que facilita o trabalho de programação orientada para *objeto*.

A linguagem C++ não apenas permite agrupar as seções do programa em objetos fáceis de se entender, como também possibilita a utilização de bibliotecas de classes, ou bibliotecas inteiras, que podem gerar muitos tipos de objetos. Especificamente neste trabalho, é utilizado o software Borland C++ versão 4.0. A Borland criou uma biblioteca chamada ObjectWindowsLibrary (OWL) para ser utilizada para o Windows. Quando se cria uma janela deriva-se um objeto independente da biblioteca de classes Borland. O mesmo ocorre quando se cria um quadro de diálogos. Com base em objetos, os programas Windows em C++ ficam muito mais curtos e a linguagem Borland C++ torna tudo muito mais simples.

Funções

Cada função deve ser projetada para realizar uma tarefa específica. Se a função está realizando mais de uma tarefa, é necessário dividi-la em duas ou mais funções, em que cada uma delas tenha um nome único e específico. Os nomes de funções devem corresponder à operação realizada por aquela função, a fim de facilitar a compreensão do código criado.

Uma função em C++ tem a seguinte estrutura:

```
tipo_retorno nome_funcao(lista_parametros)
{
    declaracoes_de_variaveis;
    comandos;
}
```

Como por exemplo:

```
void exibe_mensagem(void)
{
    cout << "Olá C++" << endl;
}
```

E para executar a função, utiliza-se o seguinte comando:

```
exibe_mensagem();
```

Os parênteses que se seguem após o nome da função informam ao compilador do C++ que o programa está utilizando uma função.

Quando o programa encontra a chamada de função, ele começa a executar os comandos de dentro dela. Quando a função termina, a execução continua no comando seguinte à chamada da função. Pode-se observar que este é um processo seqüencial.

Para aumentar a capacidade das funções, o C++ permite que os

programas passem informações, que são parâmetros para as funções. Quando uma função utilizar parâmetros, deve-se dizer ao C++ o tipo de cada parâmetro, tal como *int*, *float*, *char*, etc... como no exemplo abaixo:

```
void exibe_numero(int valor)
{
    cout << "O valor do parâmetro é " << valor << endl;
}
```

O C++ substituirá o valor passado para cada ocorrência do nome do parâmetro *valor* dentro da função, como:

```
exibe_numero(1001);
void exibe_numero(1001)
{
    cout << "O valor do parâmetro é " << 1001 << endl;
}
```

Ao se tentar passar um valor de tipo diferente para a função, como valor de ponto flutuante para um parâmetro de tipo *int*, o compilador vai gerar um erro.

Em muitos casos, uma função realizará algum tipo de cálculo. A função retornará então o resultado ao chamador. Quando uma função retornar um resultado, deve-se informar ao C++ o tipo do valor, tal como *int*, *float*, *char*, etc. Se uma função não retornar um valor, o nome da função é precedido pelo tipo *void*. Para retornar um valor ao chamador, a função utiliza o comando *return*, e quando o programa encontrar este comando a execução da função termina e o valor especificado é retornado ao chamador.

Estruturas

Uma estrutura especifica um esqueleto com o qual se pode mais tarde declarar uma variável. O programa define primeiramente a estrutura e depois declara as variáveis da mesma. Para definir uma estrutura, os programas utilizam a palavra-chave *struct*, geralmente seguida de um nome. Dentro da estrutura, especifica-se o tipo e o nome de um ou mais membros, como por exemplo:

```
struct nome
{
    int nome_membro_1;
    float nome_membro_2;
} variavel;
```

As estruturas do C++ permitem que os programas agrupem informações relacionadas, cujos tipos podem ser diferentes, em uma variável.

Para utilizar ou atribuir valor a um membro, utilizamos operador de ponto do C++, como por exemplo:

```
variavel.membro = um_valor;
uma_variavel = variavel.outro_membro;
```

Quando uma função modifica o membro de uma estrutura, o chamador deve passar a estrutura para a função pelo endereço. A função, por sua vez, utiliza um ponteiro para uma estrutura. Para acessar o membro de uma estrutura, a função deve utilizar a seguinte forma:

```
valor = variavel->membro;
variavel->outro_membro = um_valor;
```


Ponteiros

Para modificar um parâmetro em uma função, os programas devem passar o endereço do parâmetro para a função e, então, utilizar um ponteiro para o parâmetro dentro da função. Quando os programas trabalham com strings de caracteres e vetores, é comum utilizar ponteiros para manipular os elementos do vetor.

Sabe-se que um ponteiro contém um endereço de memória. Para declarar um ponteiro para um "string" de caracteres, por exemplo, a função simplesmente precede o nome da variável com um asterisco, como a seguir:

```
void uma_funcao(char *string);
```

Classes

A melhoria de programação mais significativa da linguagem C++ com relação à linguagem C ANSI é que a primeira pode usar "classes". Uma classe é algo parecido com uma estrutura de dados em C, exceto que uma classe pode ser definida para conter tanto funções quanto dados.

Segundo HOLZNER (1994), uma classe forma um tipo formal, assim como quando se declara os campos de uma estrutura. Quando se declara as variáveis daquele tipo, elas próprias, em si mesmas, são os objetos. É assim que funciona em C++; as classes são os tipos formais, e os

objetos são as variáveis específicas desse tipo.

A classe é a principal ferramenta da programação orientada para objetos do C++. No C++ pode-se utilizar uma classe para definir os objetos. O objetivo é incluir o máximo possível de informações sobre o objeto dentro da classe.

Uma classe é definida como estrutura (struct), exceto que uma classe também pode conter protótipos de funções, como por exemplo:

```
class relógio
{
    int data_interna;
    public:
    int data_externa;
    int hora(int t);
};
```

Esta classe inclui alguns dados: data_interna e data_externa; e inclui, também, uma função chamada hora(). Se fosse uma estrutura, poder-se-ia ter somente incluído os dados:

```
struct relógio_estru
{
    int data_interna;
    int data_externa;
};
```

Após criar as classes, há que se inicializá-las. As funções que iniciam os objetos são chamadas de "construtores". Os construtores são funções-membro automaticamente invocadas quando o objeto é definido pela primeira vez, iniciam o objeto da forma como desejada. E quando se termina de utilizar o objeto é necessário que se libere a memória utilizada, através dos "destrutores".

O construtor é utilizado para alocar espaço de memória. O construtor é igual a qualquer outra função-membro, exceto que ele tem o

mesmo nome da classe, como por exemplo:

```
class relógio
{
    int data_interna;

    public:
        relógio(int mem);
        int data_externa;
        int hora(int t);
};
```

A função de destrutor executa automaticamente cada vez que um objeto é destruído. Se for definida uma função de destrutor dentro das classes, o C++ invoca automaticamente o destrutor para cada objeto, quando o programa terminar.

Para declarar um destrutor também é simples, acrescenta-se um til (~) antes do nome do destrutor, da forma como segue:

```
class relógio
{
    int data_interna;

    public:
        int data_externa;
        relógio(int mem);
        ~relógio(void);
        int hora(int t);
};
```

Gerenciamento de Memória

Para alocar memória em C, normalmente usa-se a função **malloc()**, mas em C++ utiliza-se a palavra-chave **new**, a qual aloca memória e retorna um ponteiro com o tipo apropriado, assim:

```
int int_ponteiro*;
int_ponteiro = new int;
```

Para “desalocar” a memória no destrutor, usamos a palavra-chave **delete**. Como já foi mencionado, o destrutor é chamado quando o objeto não é mais necessário. Se não for usado o **delete**, a área dos dados que alocamos com **new** permanecerá na memória.

Anatomia de um programa para Windows em C++

Os programas desenvolvidos para o DOS são escritos de forma seqüencial, ou seja, uma declaração sempre vem depois de outra. O controle segue uma seqüência de linha em linha.

No Windows, um aplicativo apresenta todas as opções possíveis na tela, em forma de objetos visuais, para que o usuário faça a seleção. Isso define uma programação “voltada para o evento”, ou seja, o usuário é responsável pelo fluxo do programa e não quem o programou.

A programação Windows não é apenas voltada para o evento, mas é naturalmente “orientada para o objeto”. Isto é, cada módulo tem seu objeto e seu código e dados associados, e cada um dos objetos pode ser independente dos outros.

Sobre o Windows:

Breve Histórico Sobre o Windows:

A Microsoft começou realmente a trabalhar no Windows em 1983, somente dois anos depois do aparecimento do PC. No entanto, a versão original, o Windows 1.01, não foi realmente comercializada antes de 1985. Essa versão deveria ser executada na máquina padrão da época: um IBM PC com dois drives de disquete de 360K, 256K de RAM, e um processador 8088. A tela era disposta lado a lado automaticamente; ou seja, as janelas eram organizadas para cobrir toda a tela.

A versão Windows 2 surgiu dois anos depois. Pela primeira vez, as janelas poderiam se sobrepor na tela. Entretanto, o Windows 2 só poderia ser executado no modo real 80x86 o que limitava para um total de 1M de memória. Durante algum tempo, o Windows dividiu-se em Windows 286 e Windows 386 para tirar vantagens das capacidades do então recente chip 80386.

Em maio de 1990, a Microsoft lançou o Windows 3.0. A aparência e o jeito do Windows 3.0 significavam um grande aperfeiçoamento com relação a seus ancestrais. A versão 3.1, lançada em abril de 1992, aperfeiçoava a versão 3.0, especialmente no que dizia respeito a gerenciar arquivos. Atualmente, muitos usuários utilizam o Windows como seu principal sistema operacional do PC.

Do ponto de vista da programação, um dos recursos mais importantes do Windows 3+ é que ele pode suportar a memória estendida (até 16M de RAM). E, no modo 386 avançado, o Windows utiliza as

capacidades de memória virtual do 80/386 para dar aos programadores acesso a uma quantidade de memória até quatro vezes maior do que a memória real instalada.

O Windows:

O Windows é muito diferente do DOS. Uma das diferenças fundamentais é que o Windows é uma interface Gráfica (GUI - Interface Gráfica do **Usuário**), algumas vezes também chamada de "interface visual" ou "ambiente gráfico com janelas". Uma das idéias principais é a de que a maioria das opções disponíveis são apresentadas ao usuário de uma só vez, na forma de objetos na tela, como ferramentas prontas para serem utilizadas.

Todas as interfaces gráficas do usuário utilizam gráficos em monitores formados por mapas de bits. Os gráficos fazem uma melhor utilização da tela, transmitindo informações de uma maneira visual mais rica.

Os programas feitos para o Windows podem utilizar gráficos e textos formatados tanto no vídeo como na impressora. Uma interface gráfica não é somente mais atraente em aparência, mas também pode transmitir um grande número de informações ao usuário. Estes programas não acessam diretamente o hardware dos dispositivos gráficos, como a impressora e o monitor de vídeo. O Windows contém uma linguagem de programação de gráficos, chamada de **GDI**, que permite a fácil apresentação de gráficos e

textos formatados.

A Filosofia da GDI:

O sistema da GDI (Interface de Dispositivo Gráfico) é construído de forma que o Windows possa determinar o que o controlador pode fazer sozinho e em que ele requer ajuda. Por exemplo, se o hardware de placa de vídeo inclui um co-processador gráfico que pode desenhar figuras,, a GDI o aproveita; caso contrário, a própria GDI precisa processá-lo e passá-los ao controlador.

Os dispositivos gráficos de saída tais como vídeo, impressora, traçadores, etc... estão divididos em dois grupos: mapas de bits e vetoriais. A maioria dos dispositivos de saída para o PC são de mapas de bits, o que significa que eles representam as figuras em padrões de pontos.

As Partes de Uma Janela:

Uma janela típica do Windows 3+ será utilizada em nosso aplicativo e se encontra na Figura 3.1. No canto superior esquerdo da janela, há o quadro de menu de sistema, o qual, quando selecionado, exhibe um menu que tipicamente permite ao usuário mover a janela, fechá-la ou minimizá-la. Na parte superior central fica a barra de títulos, que fornece o título do aplicativo juntamente com o arquivo, Banco de Dados, no caso do SIOC, que está aberto e maximizado.

Figura A1.1 - Janela típica do Windows

À direita da barra de títulos estão os quadros Minimizar e Maximizar, que permitem ao usuário reduzir a janela até um ícone (chamado estado *icônico* do aplicativo) ou expandi-lo totalmente, em geral ocupando toda a tela. Abaixo da barra de títulos há uma barra de menus que oferece as opções de menus disponíveis no momento para o aplicativo. Em quase todos os aplicativos independentes, haverá uma barra de menus com pelo menos um item de menu: o menu Arquivo. Esse é o menu que geralmente oferece o item Sair na sua parte inferior.

Abaixo da barra de menus, mostrada na Figura 3.1, está a área do cliente. Na verdade, essa área ocupa toda a janela abaixo da barra de menus, exceto pelas bordas e barras de rolagem. Esta é a área de desenho, com a qual se trabalha diretamente no Borland C++. Esta é a parte da janela em que se coloca as outras partes do programa.

À direita da área do cliente está uma barra de rolagem vertical. A

barra de rolagem permite visualizar uma subseção do todo.

Na parte inferior da janela está outra barra de rolagem, que rola no sentido horizontal. Tudo na janela, exceto a área do cliente, é chamado área não-cliente; até mesmo a borda faz parte da área não-cliente. O Windows é responsável pela área não-cliente e o programa aplicativo pela área do cliente.

Configurações de Janelas de Multitarefa

As janelas filhas criadas são da classe TMDIChild. Essas janelas são chamadas "janelas MDI" (Multiple Document Interface). Cada janela MDI pode exibir um processo diferente, que é gerenciado pelo objeto TMDIClient. O objeto TMDIClient é a janela que cobre a área de cliente da janela principal, que no nosso caso é uma TMDFrame, um objeto derivado de TApplication. O esquema é mostrada na figura a seguir:

Figura A1.2 - Esquema da hierarquia das janelas

Na verdade, o programa gerencia as janelas-filhas com um objeto TMDIClient. O TMDIClient é uma janela que cobre a área de cliente do objeto TMDFrame e permite mover as outras janelas e exibir-las ou ocultá-

las na área de cliente da janela principal.

Quadros de Diálogo

No Windows, os quadros de diálogo representam a forma padrão de receber entrada de controle do usuário. O usuário do Windows utiliza os quadros de diálogo para abrir e renomear arquivos, personalizar janelas ou parâmetros de aplicativos, selecionar cores, ou quase tudo que uma simples seleção de menu pode especificar. Um dos quadros de diálogo mais comuns são os quadros de diálogos de lista.

Os usuários do Windows estão acostumados com os quadros de lista. Um quadro de lista apresenta muitas opções de "strings" de texto para a seleção. Um clique em uma das opções relacionadas ilumina e seleciona essa opção. Esses quadros são utilizados quando temos diversas opções predeterminadas que queremos apresentar ao usuário.

Os quadros de diálogos utilizados pelo SIOC foram criados pelo Resource Workshop que vem junto com o pacote da Borland C++ versão 4.0. Esta ferramenta facilita a criação tanto de quadros de diálogos como menus, teclas aceleradoras, mapas de bitmap, etc.... Com ele foi possível utilizar todos esses recursos sem digitar uma linha de código sequer.

A maioria dos quadros de diálogos devem ter um botão Cancelar, que permite ao usuário fechar o quadro de diálogo sem fazer nenhuma modificação durante o programa. O botão Ok é mais comum, pois sem ele o usuário não teria como informar ao programa que já utilizou o quadro e quer

retornar ao programa.

Quadro Abrir Arquivo

A maneira de trabalhar com os quadros de diálogos do próprio Windows, chamados de quadros comuns, é carregar primeiro o objeto-membro de Tdata do diálogo com os valores padrões (default) que se deseja. Em seguida, usa-se Execute() para exibir o quadro de diálogo. Para o quadro de diálogo Abrir, é necessário preencher uma estrutura do tipo TOpenSaveDialog::TData - ou seja, essa estrutura é um membro da classe TOpenSaveDialog. Os membros de dados desta estrutura são:

```
public:
    DWORD   Flags;
    DWORD   Error;
    char*    FileName;
    char*    Filter;
    char*    CustomFilter;
    int      FilterIndex;
    char*    InitialDir;
    char*    DefExt;
```

O parâmetro Flags, diferente do flags utilizado pelo programa SIOC, é preenchido com os valores da Tabela A1.1. Pode-se definir a extensão padrão, no quadro de diálogo Abrir, preenchendo o campo DefExt, e, no nosso caso, a extensão é ".sc".

Tabela A1.1 - Flags Abrir/Salvar Arquivo

Flags	Significado
OFN_READONLY	Marca o quadro Somente Leitura no quadro de diálogo
OFN_OVERWRITEPROMPT	Gera um quadro de mensagem se o arquivo já existir - o usuário precisa confirmar se deseja sobregravá-lo
OFN_HIDEREADONLY	Oculto o quadro Somente Leitura
OFN_NOCHANGEDIR	Desativa modificações de diretório
OFN_SHOWHELP	Faz o quadro de diálogo mostrar o botão Ajuda
OFN_ALLOWMULTISELECT	Permite que arquivos múltiplos sejam selecionados - eles serão separados por espaços na string final
OFN_PATHMUSTEXIST	Usuário só pode entrar nomes de caminhos válidos
OFN_FILEMUSTEXIST	Usuário só pode selecionar nomes de arquivos que existam

OFN_CREATEPROMPT	Pergunta se o usuário quer criar um arquivo se ele ainda não existir
OFN_SHAREAWARE	Faz o erro OF_SHARINGVIOLATION ser ignorado
OFN_NOREADONLYRETURN	Especifica que o arquivo selecionado não será somente leitura e não estará num diretório protegido contra gravação

Com isso foi possível criar a estrutura de dados do tipo TOpenSaveDialog chamada data e preencher da seguinte forma:

```
void siocMDIClient::Abrir()
{
    static TOpenSaveDialog::TData data(OFN_HIDEREADONLY |
                                       OFN_FILEMUSTEXIST,
                                       "Banco de Dados (*.sc) |*.sc|",
                                       0, 0, "SC");
    if((new TFileOpenDialog(this, data))->Execute()==IDOK)
    {
        ...
    }
}
```

Como se pode observar no código acima, a estrutura de dados passa para o construtor TFileOpenDialog(). A estrutura de dados (data) é passada para TFileOpenDialog() por referência. Os campos dessa estrutura serão preenchidos para que se possa examinar o retorno da função Execute().

Quando o usuário seleciona Abrir, no menu Arquivo, o quadro de diálogo que aparece é:

Figura A1.3 - Quadro de diálogo Arquivo Abrir

Pode-se observar que o diretório de inicialização é o próprio diretório onde o programa está, pois na estrutura não foi definido o diretório de inicialização.

Quadro Salvar Arquivo

O quadro de diálogo Salvar Arquivo utiliza o mesmo conjunto de flags que está na Tabela A1.1. Desta vez, porém, foi preenchido a estrutura TOpenSaveDialog::TData da seguinte forma:

```
void siocMDIClient::Salvar()
{
    static TOpenSaveDialog::TData data(OFN_HIDEREADONLY |
                                        OFN_OVERWRITEPROMPT,
                                        "Banco de Dados (*.sc) |*.sc|",
                                        0, "", "SC");
    if((new TFileSaveDialog(this, data))->Execute() == IDOK)
    {
    }
}
```

Como é possível observar no código acima, a estrutura de dados passa para o construtor TFileSaveDialog() e não pelo TFileOpenDialog, como foi utilizado anteriormente.

Com isso, quando o usuário seleciona o item Salvar do menu

Arquivo, tem-se o seguinte quadro na tela:

Figura A1.4 - Quadro de diálogo Arquivo Salvar

Quadro de Diálogo de Entrada de Dados do Sistema

O quadro de diálogo que permitem a edição e/ou a alteração dos dados do sistema é o quadro "Entrada de Dados":

Figura A1.5 - Quadro de Diálogo Entrada de Dados

Este quadro de diálogo permite que o usuário inicialize e/ou altere os dados do sistema, informando quantas barras e quantas linhas o sistema contém.

Quadro de Diálogo dos Dados de Linha

O quadro de diálogo "Dados de Linha" refere-se a alteração e/ou

entrada dos dados de linha. A quantidade de linhas contidas na lista depende diretamente dos valores do quadro de diálogo Entrada de Dados. O item Linhas refere-se ao nome da linha¹ cuja identificação depende diretamente do usuário.

Figura A1.6 - Quadro de Diálogo Dados de Linha

O item Comprimento refere-se ao comprimento da linha em metros. Este item poderá ser utilizado no futuro para o cálculo da impedância da linha e para o estudo de fluxo de carga ou para qualquer outra finalidade; no momento não tem nenhuma utilidade.

O item Barra Inicial refere-se a barra na qual a linha se inicial e o item Barra Final onde a mesma termina, e representam, respectivamente, os vetores "bi" e "bf".

¹ Nome da Linha: este item ainda não faz parte do banco de dados, mas pode ser incluído sem nenhum problema.

O item Medida refere-se a existência ou não da medida de fluxo na linha, e representa o vetor "mb". Se o valor do campo Medida for igual a 0 significa que não há medida de fluxo e se for igual a 1 significa que há. Este campo é importante para o estudo de observabilidade do sistema.

Quadro de Diálogo dos Dados de Barramento

O quadro de diálogo Dados de Barramentos refere-se a alteração e/ou a entrada dos dados dos barramentos. Da mesma forma que no quadro de diálogo Dados de Linha, a quantidade de barras contidas na lista de barramentos, depende diretamente do valor do quadro de diálogo Entrada de Dados.

Figura A1.7 - Quadro de Diálogo "Dados de Barramentos"

O item Barra refere-se ao nome do barramento, cuja identificação

depende diretamente do usuário.

O item Medida refere-se à existência ou não da medida de injeção de potência no barramento. Se o valor do campo Medida for igual a 0 significa que não há medida de infecção e se for igual a 1 significa que há. Este campo é importante para o estudo de observabilidade do sistema.

Os itens do grupo Localização, tanto no quadro de diálogo Dados de Linha quanto no quadro de diálogo Dados de Barra, fazem referência à disposição do sistema em seu modelo real, o que torna necessário que se faça um croqui do sistema antes mesmo deste ser inserido no programa.

Este croqui deverá conter as coordenadas dos barramentos e como estes estão distribuídos no real. A escala e a distribuição deverá ser efetuada de forma particular, visando a melhor distribuição do sistema para que não haja dificuldades na visualização do esquema.

ANEXO II - LISTAGEM DO PROGRAMA

Listagem completa do programa SIOC - Sistema Integrado de Operação e Controle:

Arquivos de cabeçalho utilizado pelo programa

```
#include <owl\owlpch.h>
#include <owl\applicat.h>
#include <owl\mdi.h>
#include <owl\editfile.h>
#include <owl\statusba.h>
#include <owl\controlb.h>
#include <owl\scroller.h>
#include <owl\buttonga.h>
#include <owl\gadget.h>
#include <owl\gadgetwi.h>
#include <owl\radiobut.h>
#include <owl\groupbox.h>
#include <owl\dc.h>
#include <owl\mdi.rh>
#include <owl\dialog.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "base.h"
#include "sioc.h"
```

```
#define USE_BWCC;
```

Variáveis públicas utilizadas pela rotina gráfica

```
BOOL FlagR, FlagD;
```

Declaração das variáveis globais utilizadas pela rotina de observabilidade

```
int nl, nb, i,
    ml[NL], mb[NB], ni[NL], nf[NL], ilha[NB],
    lx[NB], ly[NB], fi[NL], ff[NL];
```

```
int nmed,          // Número de medidas
    noex[NB],      // Numero externo da barra
    it;            // Número de iterações
```

```
float  G[NB][NB];
       H[NMED][NB];
```

Variáveis relativas a topologia da rede

```
int  npil[NB],    // Índice Inicial dos vetores NOLIG, LILIG, NEXT
     nolog[NL2],  // Número da barra adjacente*/
     lilig[NL2],  // Número da linha*/
     next[NL2],   // Próximo índice, se zero indica fim*/
     prox[NB];
```

```
BOOL conec;      // Indica se há conexão entre as áreas
```

```
IMPLEMENT_CASTABLE1(TBaseWindow, TWindow);
```

Classe siocLinhaDlg

```
class siocLinhaDlg : public TDialog
{
private:
    char  str[16];
    int  nia[NL], nfa[NL], mla[NL], fia[NL], ffa[NL];
    int  i, sel;
public:
    siocLinhaDlg(TWindow* parent, const char* name);
    void SetupWindow();
    BOOL EvInitDialog(HWND);
    void selecaoLista();
    void mudaBarraIni();
    void mudaBarraFin();
    void mudaMedida();
    void mudaFlagIni();
    void mudaFlagFin();
    void CmOk();

    DECLARE_RESPONSE_TABLE(siocLinhaDlg);
};

DEFINE_RESPONSE_TABLE1(siocLinhaDlg, TDialog)
    EV_CHILD_NOTIFY(IDOK, BN_CLICKED, CmOk),
    EV_LBN_SELCHANGE(IDD_LISTLIN, selecaoLista),
    EV_EN_CHANGE(IDD_LINBI, mudaBarraIni),
    EV_EN_CHANGE(IDD_LINBF, mudaBarraFin),
    EV_EN_CHANGE(IDD_MEDLIN, mudaMedida),
    EV_EN_CHANGE(IDD_FLAGINI, mudaFlagIni),
    EV_EN_CHANGE(IDD_FLAGFIN, mudaFlagFin),
END_RESPONSE_TABLE;
```

Contrutor da classe siocLinhaDlg

```
siocLinhaDlg::siocLinhaDlg(TWindow* parent, const char* name)
    :TDialog(parent, name), TWindow(parent)
{
}
```

Configura a janela da classe siocLinhaDlg

```
void siocLinhaDlg::SetupWindow()
{
    TDialog::SetupWindow();
}
```

Inicializa as variáveis do quadro de diálogo referente aos dados de linha

```

BOOL siocLinhaDlg::EvInitDialog(HWND)
{
    sel = 0;
    for(i = 1; i <= nl; i++)
    {
        wsprintf(str, "Linha %d", i);
        SendDlgItemMessage(IDD_LISTLIN, LB_ADDSTRING, 0, (LPARAM) str);
        mla[i] = ml[i];
        nia[i] = ni[i];
        nfa[i] = nf[i];
        fia[i] = fi[i];
        ffa[i] = ff[i];
    }
    return TRUE;
}

```

Controle do botão "Ok" do quadro Dados de Linha

```

void siocLinhaDlg::CmOk()
{
    for(int i = 1; i <= nl; i++)
    {
        ml[i] = mla[i];
        ni[i] = nia[i];
        nf[i] = nfa[i];
        fi[i] = fia[i];
        ff[i] = ffa[i];
    }

    TDialog::CmOk();
}

```

Controle da seleção das linhas do quadro Dados de Linhas

```

void siocLinhaDlg::selecaoLista()
{
    sel = (int) SendDlgItemMessage(IDD_LISTLIN, LB_GETCURSEL);

    sel++;
    if(sel > 0)
    {
        wsprintf(str, "Linha %d", sel);
        SetDlgItemText(IDD_NOMLIN, str);
        wsprintf(str, "%d", mla[sel]);
        SetDlgItemText(IDD_MEDLIN, str);
        wsprintf(str, "%d", nia[sel]);
        SetDlgItemText(IDD_LINBI, str);
        wsprintf(str, "%d", nfa[sel]);
        SetDlgItemText(IDD_LINBF, str);
        wsprintf(str, "%d", fia[sel]);
        SetDlgItemText(IDD_FLAGINI, str);
        wsprintf(str, "%d", ffa[sel]);
        SetDlgItemText(IDD_FLAGFIN, str);
    }
}

```

Controle da variável barra inicial no quadro Dados de Linha

```

void siocLinhaDlg::mudaBarraIni()
{
    GetDlgItemText(IDD_LINBI, str, 16);
    nia[sel] = atoi(str);
}

```



```

}
```

Controle da variável barra final no quadro Dados de Linha

```

void siocLinhaDlg::mudaBarraFin()
{
    GetDlgItemText(IDD_LINBF, str, 16);
    nfa[sel] = atoi(str);
}

```

Controle da variável medida no quadro Dados de Linha

```

void siocLinhaDlg::mudaMedida()
{
    GetDlgItemText(IDD_MEDLIN, str, 16);
    mla[sel] = atoi(str);
}

```

Controle da variável Flag inicial no quadro Dados de Linha

```

void siocLinhaDlg::mudaFlagIni()
{
    GetDlgItemText(IDD_FLAGINI, str, 16);
    fia[sel] = atoi(str);
}

```

Controle da variável Flag final no quadro Dados de Linha

```

void siocLinhaDlg::mudaFlagFin()
{
    GetDlgItemText(IDD_FLAGFIN, str, 16);
    ffa[sel] = atoi(str);
}

```

Classe siocBarraDlg

```

class siocBarraDlg : public TDialog
{
private:
    char str[25];
    int mba[NB], lxa[NB], lya[NB];
    int sel, i;
public:
    siocBarraDlg(TWindow* parent, const char* name);
    void SetupWindow();
    BOOL EvInitDialog(HWND);
    void selecaoLista();
    void mudaMedida();
    void mudaLocX();
    void mudaLocY();
    void CmOk();

    DECLARE_RESPONSE_TABLE(siocBarraDlg);
};

DEFINE_RESPONSE_TABLE1(siocBarraDlg, TDialog)
    EV_CHILD_NOTIFY(IDOK, BN_CLICKED, CmOk),
    EV_LBN_SELCHANGE(IDD_LISTBAR, selecaoLista),
    EV_EN_CHANGE(IDD_MEDBAR, mudaMedida),
    EV_EN_CHANGE(IDD_LOCK, mudaLocX),
    EV_EN_CHANGE(IDD_LOCY, mudaLocY),
END_RESPONSE_TABLE;

```

Contrutor da classe siocBarraDlg

```
siocBarraDlg::siocBarraDlg(TWindow* parent, const char* name)
    :TDialog(parent, name), TWindow(parent)
{
}

```

Configura a janela da classe siocBarraDlg

```
void siocBarraDlg::SetupWindow()
{
    TDialog::SetupWindow();
}

```

Inicializa as variáveis do quadro de diálogo referente aos dados de barra

```
BOOL siocBarraDlg::EvInitDialog(HWND)
{
    sel = 0;
    for(i = 1; i <= nb; i++)
    {
        wsprintf(str, "Barramento %d", i);
        SendDlgItemMessage(IDD_LISTBAR, LB_ADDSTRING, 0, (LPARAM) str);
        mba[i] = mb[i];
        lxa[i] = lx[i];
        lya[i] = ly[i];
    }
    return TRUE;
}

```

Controle do botão "Ok" do quadro Dados de Barra

```
void siocBarraDlg::CmOk()
{
    for(i = 1; i <= nb; i++)
    {
        mb[i] = mba[i];
        lx[i] = lxa[i];
        ly[i] = lya[i];
    }

    TDialog::CmOk();
}

```

Controle da seleção das barras do quadro Dados de Barras

```
void siocBarraDlg::selecaoLista()
{
    sel = (int) SendDlgItemMessage(IDD_LISTBAR, LB_GETCURSEL);

    sel++;
    if(sel > 0)
    {
        wsprintf(str, "Barramento %d", sel);
        SetDlgItemText(IDD_NOMBAR, str);
        wsprintf(str, "%d", mba[sel]);
        SetDlgItemText(IDD_MEDBAR, str);
        wsprintf(str, "%d", lxa[sel]);
        SetDlgItemText(IDD_LOCKX, str);
        wsprintf(str, "%d", lya[sel]);
        SetDlgItemText(IDD_LOCY, str);
    }
}

```

Controle da variável medida no quadro Dados de Barra

```

void siocBarraDlg::mudaMedida()
{
    GetDlgItemText(IDD_MEDBAR, str, 16);
    mba[sel] = atoi(str);
}

void siocBarraDlg::mudaLocX()
{
    GetDlgItemText(IDD_LOCX, str, 16);
    lxa[sel] = atoi(str);
}

void siocBarraDlg::mudaLocY()
{
    GetDlgItemText(IDD_LOCY, str, 16);
    lya[sel] = atoi(str);
}

```

Classe siocDadosDlg

```

class siocDadosDlg : public TDialog
{
private:
    char str[16];
public:
    siocDadosDlg(TWindow* parent, const char* name);
    void SetupWindow();
    BOOL EvInitDialog(HWND);
    void CmOk();

    DECLARE_RESPONSE_TABLE(siocDadosDlg);
};

DEFINE_RESPONSE_TABLE1(siocDadosDlg, TDialog)
    EV_CHILD_NOTIFY(IDOK, BN_CLICKED, CmOk),
END_RESPONSE_TABLE;

siocDadosDlg::siocDadosDlg(TWindow* parent, const char* name)
    :TDialog(parent, name), TWindow(parent)
{
}

void siocDadosDlg::SetupWindow()
{
    TDialog::SetupWindow();
}

BOOL siocDadosDlg::EvInitDialog(HWND)
{
    wsprintf(str, "%d", nb);
    SetDlgItemText(IDD_BARRAS, str);
    wsprintf(str, "%d", nl);
    SetDlgItemText(IDD_LINHAS, str);
    return TRUE;
}

void siocDadosDlg::CmOk()
{
    GetDlgItemText(IDD_BARRAS, str, 16);
    nb = atoi(str);
    GetDlgItemText(IDD_LINHAS, str, 16);
}

```

```

    nl = atoi(str);
    TDialog::CmOk();
}

```

Classe siocOpcDlg

```

class siocOpcDlg : public TDialog
{
private:
    TGroupBox* GrupoOpcoes;
    TRadioButton* Ob;
    TRadioButton* Fluxo;
    TRadioButton* Estado;

public:
    siocOpcDlg(TWindow* parent, const char* name, int* nOpcaoAtual);
    int* Selecao;
    void SetupWindow();
    void CmOk();

    DECLARE_RESPONSE_TABLE(siocOpcDlg);
};

DEFINE_RESPONSE_TABLE1(siocOpcDlg, TDialog)
    EV_CHILD_NOTIFY(IDOK, BN_CLICKED, CmOk),
END_RESPONSE_TABLE;

siocOpcDlg::siocOpcDlg(TWindow* parent, const char* name, int*
nOpcaoAtual)
    :TDialog(parent, name), TWindow(parent)
{
    Selecao = nOpcaoAtual;
    GrupoOpcoes = new TGroupBox(this, IDC_GRPPOPC);
    Ob = new TRadioButton(this, IDD_OBSER, GrupoOpcoes);
    Fluxo = new TRadioButton(this, IDD_FLUXO, GrupoOpcoes);
    Estado = new TRadioButton(this, IDD_ESTIMA, GrupoOpcoes);
}

void siocOpcDlg::SetupWindow()
{
    TDialog::SetupWindow();

    switch(*Selecao)
    {
        case IDD_OBSER:
            Ob->Check();
            break;
        case IDD_FLUXO:
            Fluxo->Check();
            break;
        case IDD_ESTIMA:
            Estado->Check();
            break;
        default:
            break;
    }
}

void siocOpcDlg::CmOk()
{
    if(Ob->GetCheck())
        *Selecao = IDD_OBSER;
}

```



```

else
    if(Fluxo->GetCheck())
        *Selecao = IDD_FLUXO;
    else
        *Selecao = IDD_ESTIMA;
TDialog::CmOk();
delete GrupoOpcoes;
delete Ob;
delete Fluxo;
delete Estado;
}

```

Classe TResultWindow

```

class TResulWindow : public TBaseWindow
{
private:
    TBitmap* BarraBit;

public:
    TResulWindow();
    ~TResulWindow();
    void Paint(TDC& dc, BOOL, TRect&);
};

TResulWindow::TResulWindow() : TBaseWindow()
{
    Attr.Style |= WS_VSCROLL | WS_HSCROLL;
    Scroller = new TScroller(this, 7, 16, 40, 40);
    SetBkgndColor(TColor::Gray);
    FlagR = TRUE;
}

TResulWindow::~~TResulWindow()
{
    delete BarraBit;
    FlagR = FALSE;
}

void TResulWindow::Paint(TDC& dc, BOOL, TRect&)
{
    char str[80];
    int cy, cont;

    TMemoryDC memDC(dc);

    dc.Rectangle(10, 10, 550, 960);
    dc.MoveTo(10, 50);
    dc.LineTo(550, 50);
    dc.MoveTo(10, 100);
    dc.LineTo(550, 100);
    dc.MoveTo(270, 50);
    dc.LineTo(270, 960);
    dc.SetBkMode(TRANSPARENT);
    dc.TextOut(120, 25, "RESULTADO DO PROGRAMA DE OBSERVABILIDADE", -
1);
    wsprintf(str, "Número de Barras do Sistema: %d", nb);
    dc.TextOut(20, 65, str, -1);
    wsprintf(str, "Número de Linhas do Sistema: %d", nl);
    dc.TextOut(280, 65, str, -1);
    cy = 115;
    dc.TextOut(20, cy, "Barra", -1);
}

```

```

dc.TextOut(70, cy, "Medida", -1);
dc.TextOut(130, cy, "Ilha", -1);
dc.TextOut(180, cy, "Cor", -1);
cont = 0;
for(i = 1; i <= nb; i++)
{
    cy = cy + 25;
    wsprintf(str, "%d", i);
    dc.TextOut(35, cy, str, -1);
    wsprintf(str, "%d", mb[i]);
    dc.TextOut(90, cy, str, -1);
    wsprintf(str, "%d", ilha[i]);
    dc.TextOut(140, cy, str, -1);
    if(ilha[i] <= 0 || ilha[i] > 15)
        cont = 16;
    else
        cont = ilha[i];
    BarraBit = new TBitmap(*GetApplication(), cont);
    memDC.SelectObject(*BarraBit);
    dc.BitBlt(175, cy + 5, 40, 3, memDC, 0, 0, SRCCOPY);
    delete BarraBit;
}
cy = 115;
dc.TextOut(280, cy, "Linha", -1);
dc.TextOut(330, cy, "Medida", -1);
dc.TextOut(390, cy, "B. Inicial", -1);
dc.TextOut(460, cy, "B. Final", -1);
for(i = 1; i <= nl; i++)
{
    cy = cy + 25;
    wsprintf(str, "%d", i);
    dc.TextOut(295, cy, str, -1);
    wsprintf(str, "%d", ml[i]);
    dc.TextOut(350, cy, str, -1);
    wsprintf(str, "%d", ni[i]);
    dc.TextOut(410, cy, str, -1);
    wsprintf(str, "%d", nf[i]);
    dc.TextOut(480, cy, str, -1);
}
}

```

Classe TDiagrWindow

```

class TDiagrWindow : public TBaseWindow
{
public:
    TDiagrWindow();
    ~TDiagrWindow();

    void DrawLinha(TDC& dc, int xi, int yi, int flagi, int xf, int
yf,
                    int flagf, int corLinha);
    void Paint(TDC& dc, BOOL, TRect&);

private:
    TBitmap* BarraBit;
    TBitmap* BarraMed;
    int i, j, corlinha;
};

TDiagrWindow::TDiagrWindow() : TBaseWindow()
{

```

```

Attr.Style |= WS_VSCROLL | WS_HSCROLL;
Scroller = new TScroller(this, 7, 16, 80, 60);
SetBkgndColor(TColor::White);
FlagD = TRUE;
}

TDiagrWindow::~TDiagrWindow()
{
    delete BarraBit;
    delete BarraMed;
    FlagD = FALSE;
}

```

Rotina que desenha as linhas de conexão dos barramentos na tela do computador

```

void TDiagrWindow::DrawLinha(TDC& dc, int xi, int yi, int flagi,
                             int xf, int yf, int flagf, int corLinha)
{
    int    cx, // Variável da coordenada x
           cy, // Variável da coordenada y
           teste;
    TPoint pf; // Vetor que armazena a coordenada final para o primeiro passo da
               // ligação
    TColor cor;
    const TColor DefPalette[] = {RGB(255,255,255), RGB( 0, 0, 0),
                                  RGB( 0,255,255), RGB(255, 0,255), RGB( 0, 0,255),
                                  RGB(255,255, 0), RGB(255, 0, 0), RGB( 0,255, 0),
                                  RGB(192,192,192), RGB(128,128,128), RGB(128,128, 0),
                                  RGB(128, 0,128), RGB(128, 0, 0), RGB( 0,128,128),
                                  RGB( 0,128, 0), RGB( 0, 0,128), RGB(255,255,255)};

    cor = DefPalette[corLinha];

    TPen* Pincel = new TPen(cor, 1, PS_SOLID);

    dc.SelectObject(*Pincel);

    if(xi < xf && yi < yf) teste = 0;
    if(xi < xf && yi > yf) teste = 1;
    if(xi > xf && yi < yf) teste = 2;
    if(xi > xf && yi > yf) teste = 3;
    if(xi < xf && yi == yf) teste = 4;
    if(xi > xf && yi == yf) teste = 5;
    if(xi == xf && yi < yf) teste = 6;
    if(xi == xf && yi > yf) teste = 7;
    if(xi == xf && yi == yf) teste = 8;

    switch(flagi)
    {
        case 1:
            cx = xi + 10;
            cy = yi;
            MoveTo(dc, cx, cy);
            cy = cy + 20;
            LineTo(dc, cx, cy);
            cx = cx - 20;
            LineTo(dc, cx, cy);
            dc.GetCurrentPosition(pf);
            break;
        case 2:
            cx = xi + 20;

```

```
        cy = yi;
        MoveTo(dc, cx, cy);
        cy = cy + 40;
        LineTo(dc, cx, cy);
        dc.GetCurrentPosition(pf);
        break;
    case 3:
        cx = xi + 30;
        cy = yi;
        MoveTo(dc, cx, cy);
        cy = cy + 20;
        LineTo(dc, cx, cy);
        cx = cx + 20;
        LineTo(dc, cx, cy);
        dc.GetCurrentPosition(pf);
        break;
    case 4:
        cx = xi + 10;
        cy = yi;
        MoveTo(dc, cx, cy);
        cy = cy - 20;
        LineTo(dc, cx, cy);
        cx = cx - 20;
        LineTo(dc, cx, cy);
        dc.GetCurrentPosition(pf);
        break;
    case 5:
        cx = xi + 20;
        cy = yi;
        MoveTo(dc, cx, cy);
        cy = cy - 40;
        LineTo(dc, cx, cy);
        dc.GetCurrentPosition(pf);
        break;
    case 6:
        cx = xi + 30;
        cy = yi;
        MoveTo(dc, cx, cy);
        cy = cy - 20;
        LineTo(dc, cx, cy);
        cx = cx + 20;
        LineTo(dc, cx, cy);
        dc.GetCurrentPosition(pf);
        break;
    default:
        break;
} // Fim do flagi

switch(flagf)
{
    case 1:
        cx = xf + 10;
        cy = yf;
        MoveTo(dc, cx, cy);
        cy = cy + 20;
        LineTo(dc, cx, cy);
        cx = cx - 20;
        LineTo(dc, cx, cy);
        break;
    case 2:
        cx = xf + 20;
        cy = yf;
```



```
        MoveTo(dc, cx, cy);
        cy = cy + 40;
        LineTo(dc, cx, cy);
        break;
    case 3:
        cx = xf + 30;
        cy = yf;
        MoveTo(dc, cx, cy);
        cy = cy + 20;
        LineTo(dc, cx, cy);
        cx = cx + 20;
        LineTo(dc, cx, cy);
        break;
    case 4:
        cx = xf + 10;
        cy = yf;
        MoveTo(dc, cx, cy);
        cy = cy - 20;
        LineTo(dc, cx, cy);
        cx = cx - 20;
        LineTo(dc, cx, cy);
        break;
    case 5:
        cx = xf + 20;
        cy = yf;
        MoveTo(dc, cx, cy);
        cy = cy - 40;
        LineTo(dc, cx, cy);
        break;
    case 6:
        cx = xf + 30;
        cy = yf;
        MoveTo(dc, cx, cy);
        cy = cy - 20;
        LineTo(dc, cx, cy);
        cx = cx + 20;
        LineTo(dc, cx, cy);
        break;
    default:
        break;
} // Fim do flagf

switch(teste)
{
    case 0:
        cx = pf.x;
        LineTo(dc, cx, cy);
        cy = pf.y;
        LineTo(dc, cx, cy);
        break;
    case 1:
        cx = pf.x;
        LineTo(dc, cx, cy);
        cy = pf.y;
        LineTo(dc, cx, cy);
        break;
    case 2:
        cx = pf.x;
        LineTo(dc, cx, cy);
        cy = pf.y;
        LineTo(dc, cx, cy);
        break;
```

```

    case 3:
        cx = pf.x;
        LineTo(dc, cx, cy);
        cy = pf.y;
        LineTo(dc, cx, cy);
        break;
    case 4:
        cx = pf.x;
        LineTo(dc, cx, cy);
        cy = pf.y;
        LineTo(dc, cx, cy);
        break;
    case 5:
        cx = pf.x;
        LineTo(dc, cx, cy);
        cy = pf.y;
        LineTo(dc, cx, cy);
        break;
    case 6:
        cx = pf.x;
        LineTo(dc, cx, cy);
        cy = pf.y;
        LineTo(dc, cx, cy);
        break;
    case 7:
        cx = pf.x;
        LineTo(dc, cx, cy);
        cy = pf.y;
        LineTo(dc, cx, cy);
        break;
    case 8:
        break;
    default:
        break;
} // Fim do teste
delete Pincel;
}

void TDiagrWindow::Paint(TDC& dc, BOOL, TRect&)
{
    TMemoryDC memDC(dc);

    int m, n, nlb[NB], cont;
    char str[16];

    for(i = 1; i <= nb; i++)
    {
        n = 0;
        for(j = 1; j <= nb; j++)
        {
            if(ni[j] == i) n++;
            if(nf[j] == i) n++;
        }
        nlb[i] = n;
    }

    n = 0;

    cont = 1;

    for(i = 1; i <= nb; i++)
    {

```

```

        if(ilha[i] > 15 || ilha[i] <= 0)
            cont = 16;
        else
            cont = ilha[i];
        BarraBit = new TBitmap(*GetApplication(), cont);
        memDC.SelectObject(*BarraBit);
        dc.BitBlt(lx[i], ly[i], 40, 3, memDC, 0, 0, SRCCOPY);
        if(mb[i] == 1)
        {
            BarraMed = new TBitmap(*GetApplication(), cont + 20);
            memDC.SelectObject(*BarraMed);
            dc.BitBlt(lx[i] + 40, ly[i] - 2, 10, 7, memDC, 0, 0, SRCCOPY);
            delete BarraMed;
        }
        wsprintf(str, "%d", i);
        dc.TextOut(lx[i] - 7, ly[i] + 3, str, -1);
        delete BarraBit;
    }

    for(i = 1; i <= nl; i++)
    {
        m = ni[i]; n = nf[i];
        if(ilha[m] == ilha[n])
        {
            if(ilha[m] <= 0 || ilha[m] > 15)
                cont = 16;
            else
                cont = ilha[m];
            DrawLinha(dc, lx[m], ly[m], fi[i], lx[n], ly[n], ff[i], cont);
        }
    }
}

```

CLASSE SIOCMDICLIENT - Classe Cliente
--

```

class siocMDIClient : public TMDIClient
{
public:
    struct SAVERESTORE
    {
        int NumLinhas, NumBarras,
            BarIni[NL], BarFin[NL], MedLinha[NL],
            FlagIni[NL], FlagFin[NL],
            MedBarra[NB], LocX[NB], LocY[NB];
    };

    int nOpcao;

    siocMDIClient();
    virtual ~siocMDIClient();

protected:
    virtual void SetupWindow();

protected:
    void Novo();
    void Abrir();
    void Salvar();
    void Sair();
    void Opcoes();
    void Linhas();
    void Barras();
}

```

```

    void Dados();
    void VisResul();
    void VisDiagr();
    void Sobre();
    void Observab();

    DECLARE_RESPONSE_TABLE(siocMDIClient);
};

DEFINE_RESPONSE_TABLE1(siocMDIClient, TMDIClient)
    EV_COMMAND(IDM_NOVO, Novo),
    EV_COMMAND(IDM_ABRIR, Abrir),
    EV_COMMAND(IDM_SALVAR, Salvar),
    EV_COMMAND(CM_EXIT, Sair),
    EV_COMMAND(IDM_SOBRE, Sobre),
    EV_COMMAND(IDM_OPcoes, Opcoes),
    EV_COMMAND(IDM_LINHAS, Linhas),
    EV_COMMAND(IDM_BARRAS, Barras),
    EV_COMMAND(IDM_DADOS, Dados),
    EV_COMMAND(IDM_RESULT, VisResul),
    EV_COMMAND(IDM_DIAGRAMA, VisDiagr),
END_RESPONSE_TABLE;

siocMDIClient::siocMDIClient() : TMDIClient()
{
    SetBkgndColor(RGB(0xc0, 0xc0, 0xc0));

    nOpcao = IDD_OBSER;
}

siocMDIClient::~~siocMDIClient()
{
    Destroy();
}

void siocMDIClient::SetupWindow()
{
    TMDIClient::SetupWindow();
}

void siocMDIClient::Observab()
{
    int aux, i, j, k, in, ind, no, indice, ina, l, n4, n5, lff = 0;
    char flag, kstop, prim, test[NB];
    double m, pivo;
    int vetor[NB], adj, ilh, n1, n2;



Inicialização dos vetores



    for(i = 0; i < NB; i++)
        noex[i] = npil[i] = 0;
    for(i = 0; i < NL2; i++)
        nolog[i] = lilig[i] = next[i] = 0.0;

    nmed = 0;
    ind = 0;
    j = 0;

    for(i = 1; i <= nb; i++)
        if(mb[i] == 1) nmed++;

```



```

for(i = 1; i <= nl; i++)
    if(ml[i] == 1) nmed++;

```

***Esta função monta os vetores npil, nolog, lilig, next,
os quais descrevem a topologia da rede***

```

lff = 1;

for(i = 1; i <= nb; i++) npil[i] = 0;
for(i = 1; i <= 2*nl; i++)
{
    next[i] = 0;
    nolog[i] = 0;
    lilig[i] = 0;
} //end for
for(i = 1; i <= nl; i++)
{
    prim = '1';
    k = ni[i];
    l = nf[i];
    kstop = '0';
    do
    {
        if(kstop == '0' && prim != '1')
        {
            aux = k;
            k = l;
            l = aux;
            kstop = '1';
        } //end if
        prim = '0';
        ina = 0;
        in = npil[k];
        if(in == 0)
            npil[k] = lff;
        else
        {
            n5 = next[l];
            n4 = next[nolog[in]];
            flag = '0';
            while(n5 > n4 && flag != '1')
            {
                ina = in;
                in = next[in];
                if(in == 0)
                {
                    next[lff] = in;
                    next[ina] = lff;
                    flag = '1';
                } //end if
            }
            else
                n4 = next[nolog[in]];
        } //end while
        if(flag != '1')
        {
            if(ina == 0)
            {
                next[lff] = in;
                npil[k] = lff;
            } //end if
        }
        else

```

```

        {
            next[lff] = in;
            next[ina] = lff;
        } //end else
    } //end if
} //end else
nolig[lff] = 1;
lilig[lff] = i;
lff++;
} //end do
while(kstop == '0');
) // end for
conec = TRUE;
it = 0;

while(conec == TRUE)
{
    it++;
    int i, j, k, ind, lin, tmp;
    float temp1, temp2;
    for(i = 0; i < NMED; i++)
        for(j = 0; j < NB; j++)
            H[i][j] = 0.0;

    lin = 0;

    for(i = 1; i <= nl; i++)
    {
        if(ml[i] == 1)
        {
            lin++;
            H[lin][ni[i]] = 1.0;
            H[lin][nf[i]] = -1.0;
        } //end if
    } //end for
    for(i = 1; i <= nb; i++)
    {
        if(mb[i] == 1)
        {
            lin++;
            ind = npil[i];
            tmp = 0;
            while(ind!=0)
            {
                H[lin][nolig[ind]] = -1.0;
                ind = next[ind];
                tmp++;
            } //end while
            H[lin][i] = tmp;
        } //end if
    } //end for

    for(i = 1; i <= nb; i++)
        for(j = 1; j <= nb; j++)
        {
            G[i][j] = 0.0;
            for(k = 1; k <= nmed; k++)
            {
                temp1=H[k][i];
                temp2=H[k][j];

```

```

        G[i][j] += temp1*temp2;
    }
}

```

Método de gauss

```

for(k = 1; k <= nb; k++)
{
    if(fabs(G[k][k]) > 1.0e-5)
    {
        m = 1.0/G[k][k];
        for(j = k; j <= nb; j++) G[k][j] *= m;
        for(i = k+1; i <= nb; i++)
        {
            pivo = G[i][k];
            for(j = k; j <= nb; j++)
            {
                G[i][j] -= (pivo*G[k][j]);
                if(fabs(G[i][j]) < 1.0e-5) G[i][j] = 0.0;
            } //end for
        } //end for
    } //end if
    else G[k][k] = 0.0;
} //end for

```

Cria a árvore

```

for(i = 1; i <= nb; i++)
    for(j = i+1; j <= nb; j++)
        if(G[i][j] != 0.0)
        {
            prox[i] = j;
            break;
        } //end if
    else
        if(j == nb) prox[i] = 0;

```

Determina as Ilhas

```

ind = 0;
for(i = 1; i <= nb; i++) test[i] = '0';
for(i = 1; i <= nb; i++)
{
    in = 0;
    no = i;
    if(test[no] == '0')
    {
        while((test[no] == '0') && (no))
        {
            in++;
            vetor[in] = no;
            test[no] = '1';
            no = prox[no];
        } //end while
        if(prox[vetor[in]])
            indice = ilha[no];
        else
            indice = ++ind;
    } //end if
    for(j = 1; j <= in; j++)
        ilha[vetor[j]] = indice;
} //end for

```

Verifica conexão de fluxo

```

for(i = 1; i <= nl; i++)
{
  if(ml[i] != 0)
  {
    if(ilha[ni[i]] != ilha[nf[i]])
    {
      n1 = 0; n2 = 0;
      ind = npil[ni[i]];
      ilh = ilha[nf[i]];
      while(ind != 0)
      {
        adj = nolog[ind];
        if(ilh == ilha[adj])
        {
          if(mb[adj] == 1) n1++;
          if(ml[lilig[ind]] != 0) n1++;
        } //end if
        ind = next[ind];
      } //end while
      ind = npil[nf[i]];
      ilh = ilha[ni[i]];
      while(ind != 0)
      {
        adj = nolog[ind];
        if(ilh == ilha[adj])
        {
          if(mb[adj] == 1) n2++;
          if(ml[lilig[ind]] != 0) n2++;
        } //end if
        ind = next[ind];
      } //end while
    } //end if
    if(n1 > n2)
      ilha[ni[i]] = ilha[nf[i]];
    else
      ilha[nf[i]] = ilha[ni[i]];
  } //end if
} //end for

```

Verifica conexão

```

conec = FALSE;
for(i = 1; i <= nb; i++)
{
  if(mb[i] == 1)
  {
    ilh = ilha[i];
    ind = npil[i];
    while(ind)
    {
      if(ilh != ilha[nolog[ind]])
      {
        mb[i] = 0;
        nmed--;
        ind = 0;
        conec = TRUE;
      }
    }
    else
      ind = next[ind];
  }
}

```



```

    }
  }
}

```

Cria Janela de Visualização do Diagrama Unifilar

```

void siocMDIClient::VisDiagr()
{
    TMDIChild* Diagrama;
    if(!FlagD)
    {
        Diagrama = new TMDIChild(*this, "Diagrama Unifilar",
                                new TDiagrWindow);
        Diagrama->SetIcon(0, 103);
        Diagrama->Attr.X = 0;
        Diagrama->Attr.Y = 0;
        Diagrama->Attr.W = GetSystemMetrics(SM_CXSCREEN)/2;
        Diagrama->Attr.H = 300;
        Diagrama->Create();
    }
    else
        UpdateWindow();
}

```

Cria Janela de Visualização dos Resultados Numéricos

```

void siocMDIClient::VisResul()
{
    TMDIChild* Resultado;
    if(!FlagR)
    {
        Resultado = new TMDIChild(*this, "Resultados", new TResulWindow);
        Resultado->SetIcon(0, 90);
        Resultado->Attr.X = GetSystemMetrics(SM_CXSCREEN)/2;
        Resultado->Attr.Y = 0;
        Resultado->Attr.W = GetSystemMetrics(SM_CXSCREEN)/2;
        Resultado->Attr.H = 300;
        Resultado->Create();
    }
    else
        UpdateWindow();
}

```

Abre Quadro de Diálogo de Explicação

```

void siocMDIClient::Sobre()
{
    TDialog* dialogo = new TDialog(this, "CaixaSobre");
    dialogo->Execute();
}

```

Abre um Novo Banco de Dados

```

void siocMDIClient::Novo()
{
    CloseChildren();
    for(i = 0; i < NL; i++)
    {
        ml[i] = 0;
        ni[i] = 0;
        nf[i] = 0;
        fi[i] = 0;
    }
}

```

```

    ff[i] = 0;
}
for(i = 0; i < NB; i++)
{
    mb[i] = 0;
    ilha[i] = 0;
    lx[i] = 0;
    ly[i] = 0;
}
siocMDIClient::Dados();
siocMDIClient::Barras();
siocMDIClient::Linhas();
siocMDIClient::VisResul();
siocMDIClient::VisDiagr();
}

```

Abre um Banco de Dados Existente

```

void siocMDIClient::Abrir()
{
    static TOpenSaveDialog::TData
        data(OFN_HIDEREADONLY | OFN_FILEMUSTEXIST,
            "Banco de Dados (*.sc)|*.sc|",
            0, 0, "SC");
    if((new TFileOpenDialog(this, data))->Execute()==IDOK)
    {
        if(strlen(data.FileName))
        {
            FILE *load;
            SAVERESTORE *Restore;

            Restore = new SAVERESTORE;

            for(i = 0; i < NL; i++)
            {
                ml[i] = 0; ni[i] = 0; nf[i] = 0;
                fi[i] = 0; ff[i] = 0;
            }
            for(i = 0; i < NB; i++)
            {
                mb[i] = 0; ilha[i] = 0;
                lx[i] = 0; ly[i] = 0;
            }
            if(!Restore)
            {
                return;
            }

            if((load = fopen(data.FileName, "r+b")) == NULL)
            {}
            else
            {
                fread(Restore, sizeof(SAVERESTORE), 1, load);
                fclose(load);
                nb = Restore->NumBarras;
                nl = Restore->NumLinhas;
                memcpy(ni, Restore->BarIni, (nl+1) * sizeof(int));
                memcpy(nf, Restore->BarFin, (nl+1) * sizeof(int));
                memcpy(ml, Restore->MedLinha, (nl+1) * sizeof(int));
                memcpy(fi, Restore->FlagIni, (nl+1) * sizeof(int));
                memcpy(ff, Restore->FlagFin, (nl+1) * sizeof(int));
                memcpy(mb, Restore->MedBarra, (nb+1) * sizeof(int));
            }
        }
    }
}

```

```

        memcpy(lx, Restore->LocX, (nb+1) * sizeof(int));
        memcpy(ly, Restore->LocY, (nb+1) * sizeof(int));
    }
    delete Restore;
    switch(nOpcao)
    {
        case IDD_OBSER:
            siocMDIClient::Observar();
            break;
        case IDD_FLUXO:
            break;
        case IDD_ESTIMA:
            break;
    }
    siocMDIClient::VisResul();
    siocMDIClient::VisDiagr();
}
}
}

```

Salva o Banco de Dados Ativo

```

void siocMDIClient::Salvar()
{
    static TOpenSaveDialog::TData
        data(OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT,
            "Banco de Dados (*.sc)|*.sc|",
            0, "", "SC");
    if((new TFileSaveDialog(this, data)->Execute() == IDOK)
    {
        FILE *banco;
        SAVERESTORE *Save;

        Save = new SAVERESTORE;
        if((banco = fopen(data.FileName, "w+b")) == NULL)
        {
            delete Save;
            return;
        }
        Save->NumLinhas = nl;
        Save->NumBarras = nb;
        memcpy(Save->BarIni, ni, (nl+1) * sizeof(int));
        memcpy(Save->BarFin, nf, (nl+1) * sizeof(int));
        memcpy(Save->MedLinha, ml, (nl+1) * sizeof(int));
        memcpy(Save->FlagIni, fi, (nl+1) * sizeof(int));
        memcpy(Save->FlagFin, ff, (nl+1) * sizeof(int));
        memcpy(Save->MedBarra, mb, (nb+1) * sizeof(int));
        memcpy(Save->LocX, lx, (nb+1) * sizeof(int));
        memcpy(Save->LocY, ly, (nb+1) * sizeof(int));
        fwrite(Save, sizeof(SAVERESTORE), 1, banco);
        fclose(banco);
        delete Save;
    }
}

```

Abre Quadro de Diálogo de Opções

```

void siocMDIClient::Opcoes()
{
    siocOpcDlg* dialogo = new siocOpcDlg(this, "CaixaOpcoes",
    &nOpcao);
    if(dialogo->Execute() == IDOK)
    {

```

```
switch(nOpcao)
{
    case IDD_OBSER:
        siocMDIClient::Observab();
        break;
    case IDD_FLUXO:
        break;
    case IDD_ESTIMA:
        break;
    default:
        break;
}
siocMDIClient::VisResul();
siocMDIClient::VisDiagr();
}
```

Abre Quadro de Diálogo Dados de Barra

```
void siocMDIClient::Barras()
{
    siocBarraDlg* dlgBarra = new siocBarraDlg(this, "CaixaBarra");
    if(dlgBarra->Execute() == IDOK)
    {
        switch(nOpcao)
        {
            case IDD_OBSER:
                siocMDIClient::Observab();
                break;
            case IDD_FLUXO:
                break;
            case IDD_ESTIMA:
                break;
            default:
                break;
        }
        siocMDIClient::VisResul();
        siocMDIClient::VisDiagr();
    }
}
```

Abre Quadro de Diálogo Dados do Sistema

```
void siocMDIClient::Dados()
{
    siocDadosDlg* dlgDados = new siocDadosDlg(this, "CaixaDados");
    if(dlgDados->Execute() == IDOK)
    {
        switch(nOpcao)
        {
            case IDD_OBSER:
                siocMDIClient::Observab();
                break;
            case IDD_FLUXO:
                break;
            case IDD_ESTIMA:
                break;
            default:
                break;
        }
        siocMDIClient::VisResul();
        siocMDIClient::VisDiagr();
    }
}
```



```

}
```

Abre Quadro de Diálogo Dados de Linha

```

void siocMDIClient::Linhas()
{
    siocLinhaDlg* dlgLinha = new siocLinhaDlg(this, "CaixaLinha");
    if(dlgLinha->Execute() == IDOK)
    {
        switch(nOpcao)
        {
            case IDD_OBSER:
                siocMDIClient::Observab();
                break;
            case IDD_FLUXO:
                break;
            case IDD_ESTIMA:
                break;
            default:
                break;
        }
        siocMDIClient::VisResul();
        siocMDIClient::VisDiagr();
    }
}
}
```

Sai do Programa

```

void siocMDIClient::Sair()
{
    CloseChildren();
    Destroy();
}
}
```

CLASSE TSIOCAPP - Classe Principal

```

class TSiocApp : public TApplication
{
public:
    TSiocApp();
    virtual ~TSiocApp();
    void InitMainWindow();
};
```

Classe TSiocApp

```

TSiocApp::TSiocApp() : TApplication("Sistema Integrado de Operação e
Controle")
{
}

TSiocApp::~TSiocApp()
{
    TApplication::~~TApplication();
}

void TSiocApp::InitMainWindow()
{
    nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ? SW_SHOWMAXIMIZED :
nCmdShow;

    TDecoratedMDIFrame* frame = new TDecoratedMDIFrame(Name,
```

```

        TResID(1), *(new siocMDIClient), TRUE);

frame->SetIcon(this, IDI_SIOC);

frame->AssignMenu(MDI_MENU);

frame->Attr.AccelTable = MDI_MENU;

TControlBar* controlbar = new TControlBar(frame);
controlbar->Insert(*new TButtonGadget(IDM_NOVO, IDM_NOVO));
controlbar->Insert(*new TButtonGadget(IDM_ABRIR, IDM_ABRIR));
controlbar->Insert(*new TButtonGadget(IDM_SALVAR, IDM_SALVAR));
controlbar->Insert(*new TSeparatorGadget(6));
controlbar->Insert(*new TButtonGadget(IDM_IMPRIMIR, IDM_IMPRIMIR));
controlbar->Insert(*new TSeparatorGadget(6));
controlbar->Insert(*new TButtonGadget(IDM_OPCOES, IDM_OPCOES));
controlbar->Insert(*new TSeparatorGadget(6));
controlbar->Insert(*new TButtonGadget(IDM_LINHAS, IDM_LINHAS));
controlbar->Insert(*new TButtonGadget(IDM_BARRAS, IDM_BARRAS));
controlbar->Insert(*new TSeparatorGadget(6));
controlbar->Insert(*new TButtonGadget(IDM_SOBRE, IDM_SOBRE));
controlbar->SetHintMode(TGadgetWindow::EnterHints);
frame->Insert(*controlbar, TDecoratedFrame::Top);

TStatusBar *statusbar = new TStatusBar(frame,
                                     TGadget::Recessed,
                                     TStatusBar::CapsLock |
                                     TStatusBar::NumLock |
                                     TStatusBar::Overtime);
frame->Insert(*statusbar, TDecoratedFrame::Bottom);

MainWindow = frame;

EnableBWCC();
}

```

Rotina principal do programa SIOC

```

int OwlMain(int, char* [])
{
    TSiocApp App;
    int resposta;

    resposta = App.Run();

    return resposta;
}

```

REFERÊNCIAS BIBLIOGRÁFICAS

1. BRETAS, N.G. - **Network Observability: a theory based on triangular factorization and path graph concepts - PART I.** Submitted to presentation at IEEE PES Winter Meeting, 1995.
2. BRETAS, N.G. et al - **Network Observability: algorithms based on triangular factorization and path graph concepts.** Submitted to presentation at IEEE PES Winter Meeting, 1995.
3. MONTICELLI, A & WU, F.F. - **Network Observability: theory.** IEEE Trans. On PAS, V.104, n° 5, May 1985.
4. MONTICELLI, A. & WU, F.F. - **Network Observability: identification of observable island and measurement placement.** IEEE Trans. On PAS, V.104, n° 5, May 1985.
5. PETZOLD, C. - **Programando Para Windows 3.1.** São Paulo, Editora Makron Books, 1993.
6. TINNEY, W.F., BRANDWAJN, V., CHAN, S.M. - **Sparse Vector Methods.** IEEE Trans. On PAS, V.104, n° 2, Feb 1985

BIBLIOGRAFIA SUPLEMENTAR

ALLMONG, J.J.; IRISARRI, G.D. & SASSON, A.M. - **An Examination of Solvability for State Estimation Algorithms.** IEEE PES Winter Meeting. Paper A80 008-3, 1980.

BRETAS, N.G. - **Network Observability: a theory based on triangular factorization and path graph concepts - PART I.** Submitted to presentation at IEEE PES Winter Meeting, 1995.

BRETAS, N.G. et al - **Network Observability: algorithms based on triangular factorization and path graph concepts.** Submitted to presentation at IEEE PES Winter Meeting, 1995.

CLEMENTS, K.A. & WOLLENBERG, B.F. - **An Algorithm for Observability Determination in Power System State Estimation.** IEEE PES Summer Power Meeting. Paper A75 447-3, 1975.

CLEMENTS, K.A.; KRUMPHOLZ, G.R. & DAVIS, P.W. - **Power System State Estimation Residual Analysis: an algorithm using network topology.** IEEE Trans. Power Apparatus and Systems, vol. 100, n° 4, pp. 1779-1787, Abril 1981.

CLEMENTS, K.A.; KRUMPHOLZ, G.R. & DAVIS, P.W. - **Power System State Estimation with Measurement Deficiency: an algorithm that Determines the Maximal Observable Subnetwork.** IEEE Trans. Power Apparatus and Systems, vol. 101, n° 9, pp. 3044-3052, Setembro 1982.

CLEMENTS, K.A.; KRUMPHOLZ, G.R. & DAVIS, P.W. - **Power System State Estimation with Measurement Deficiency: an Observability/Measurement Placement Algorithm.** IEEE Trans. Power Apparatus and Systems,

- vol. PAS-102, pp. 2012-2020, Julho 1983.
- HANDSCHIN, E. & BONGERS, C. - **Theoretical and Practical Considerations in the Design of State Estimators for Electric Power Systems.** In Computerized Operation of Power Systems, S. C. Savulescu, Editor, Elsevier, pp. 104-136, 1972.
- HOLZNER, S. - **Borland C++ - Programação for Windows.** São Paulo, Editora Makron Books, 1994.
- HORTON, J.S. & MASIELLO, R.D. - **On-Line Decoupled Observability Processing.** PICA Proc. pp. 420-426, 1977.
- JAMSA, K.A. - **Salvo pelo C++.** Rio de Janeiro, Livros Técnicos e Científicos Editora, 1994.
- KRUMPHOLZ, G.R.; CLEMENTS, K.A & DAVIS, P.W. - **Power System Observability: a practical algorithm using network topology.** IEEE Trans. Power Apparatus and Systems, vol. 99, pp. 1534-1542, Julho-Agosto 1980.
- MONTICELLI, A & WU, F.F. - **Network Observability: theory.** IEEE Trans. On PAS, V.104, n° 5, May 1985.
- MONTICELLI, A. & WU, F.F. - **Network Observability: identification of observable island and measurement placement.** IEEE Trans. On PAS, V.104, n° 5, May 1985.
- NORTON, P. & YAO, P. - **Programando em Borland C++ Para Windows.** Rio de Janeiro, Berkeley Brasil Editora, 1992.
- PETZOLD, C. - **Programando Para Windows 3.1.** São Paulo, Editora Makron Books, 1993.
- QUINTANA, V.H.; SIMÕES-COSTA, A. & MANDRIL, A. - **Power System Observability Using a Direct Graphic Theoretic Approach.** IEEE Trans. on PAS, V.101, n° 3, Março 1982.
- SCHILDT, H. - **Windows NT - Manual de Programação.** São Paulo, Editora Makron Books, 1994.
- TINNEY, W.F., BRANDWAJN, V., CHAN, S.M. - **Sparse Vector Methods.** IEEE Trans. On PAS, V.104, n° 2, Feb 1985.