ESCOLA POLITÉCNICA
DA
UNIVERSIDADE DE SÃO PAULO

DEPARTAMENTO DE ENGENHARIA DE ENERGIA E
AUTOMAÇÃO ELÉTRICAS

Acoplamento de Circuitos Elétricos com o Método dos Elementos Finitos

Fernando Jaruche Nunes

PROJETO DE FORMATURA/2004

ESCOLA POLITÉCNICA
DA
UNIVERSIDADE DE SÃO PAULO

DEPARTAMENTO DE ENGENHARIA DE ENERGIA E
AUTOMAÇÃO ELÉTRICAS

PROJETO DE FORMATURA / 2004

Acoplamento de Circuitos Elétricos com o Método dos Elementos Finitos

ALUNO: Fernando Jaruche Nunes
ORIENTADORES: Mauricio Caldora Costa,
Silvio Ikuyo Nabeta
COORDENADOR: Carlos Márcio Vieira Tahan

Aos meus pais,

pelo apoio e incentivo.

A Heloísa,

Pela paciência durante a execução deste trabalho.

AGRADECIMENTOS

Ao meu orientador, Dr. Mauricio Caldora Costa, pelo seu empenho e dedicação dados durante o desenvolvimento deste trabalho.

Ao Professor Silvio Ikuyo Nabeta, por sua ajuda e colaboração.

Ao Professor José Roberto Cardoso, Marcelo Facio Palin, Álvaro Batista Dietrich e demais colegas do Laboratório de Eletromagnetismo Aplicado, por todo seu apoio e colaboração.

À FUSP - Fundação de Apoio À Universidade de São Paulo, pela bolsa de iniciação científica concedida.

Ao Departamento de Engenharia de Energia e Automação Elétricas da Escola Politécnica da Universidade de São Paulo.

A todos que contribuíram para a execução deste trabalho.

RESUMO

Máquinas elétricas são dispositivos eletromagnéticos que apresentam geometrias e comportamentos complexos. Simular este tipo de dispositivo requer em muitos casos considerar diferentes fenômenos em um mesmo problema.

Este trabalho destina-se ao desenvolvimento de uma ferramenta computacional para a análise simultânea dos fenômenos eletromagnéticos e de circuitos elétricos.

O Método dos Elementos Finitos (MEF) é empregado em conjunto com a Análise Nodal Modificada (ANM) nesta implementação para a simulação de casos Magnetodinâmicos.

ABSTRACT

Electric machines are electromagnetic devices that present complex geometries and behaviors. The simulation of such device requires in many cases to consider different phenomena in the same problem.

This work is intended to build a computational tool for simultaneous analysis of the electromagnetic phenomena and electric circuits.

The Finite Elements Method (FEM) is used together with Modified Nodal Analysis (MNA) in this implementation to simulate Magnetodynamic cases.

Indice

Indice de Tabelas

Indice de Figuras

1. Introdução

Máquinas elétricas são dispositivos eletromagnéticos que apresentam geometrias e comportamentos complexos. Simular este tipo de dispositivo requer em muitos casos considerar diferentes fenômenos em um mesmo problema.

Estas são compostas basicamente por um circuito magnético apropriado e por um conjunto de condutores cujas correntes produzem fluxo magnético por sua vez capaz de produzir força. No entanto, as máquinas são alimentadas normalmente por tensão e por esta razão não se sabe a priori os valores das correntes que circulam por seus condutores.

Soma-se o fato de que fontes de tensão ou corrente reais possuem características indutivas, resistivas e capacitivas ou mesmo podem ser constituídas por circuitos eletrônicos por ex: inversores.

Logo, para se fazer uma análise completa de uma máquina elétrica é preciso levar em conta os fenômenos eletromagnéticos e os circuitos elétricos externos em conjunto quando se deseja obter resultados próximos da realidade.

2. Objetivo

Este trabalho destina-se ao desenvolvimento de uma ferramenta computacional para a análise simultânea de um dispositivo eletromagnético e um circuito elétrico externo.

Será usado o Método dos Elementos Finitos (MEF), para a analise do dispositivo, em conjunto com a Análise Nodal Modificada (ANM) para a simulação de casos Magnetodinâmicos.

3. Método dos Elementos Finitos

O Método dos Elementos Finitos, aplicado ao eletromagnetismo, tem por objetivo resolver problemas eletromagnéticos de geometrias complexas impraticáveis de serem resolvidas analiticamente pelas equações de Maxwell. Adequado para uma grande gama de problemas o MEF pode ser implementado sem dificuldade e limitações [1].

Este método divide a geometria do problema em várias partes (os elementos finitos). Através destes elementos é montado um sistema de equações para a determinação das grandezas de interesse tais como potencial magnético ou elétrico em cada nó da malha (vértices dos elementos finitos) a partir dos quais se podem calcular campos magnéticos ou elétricos no interior destes elementos. De posse destes dados podemos ainda calcular grandezas como energia, força, torque e parâmetros de circuito elétrico, tais como indutâncias, capacitâncias e resistências.

Usualmente se divide a resolução de um problema pelo MEF em três etapas:

- Descrição da geometria (pré-processamento)
- Resolução do sistema de equações (processamento)
- Exploração de resultados (pós-processamento)

O pré-processamento consiste na descrição geométrica do problema bem como a geração da malha e atribuição de propriedades físicas. Assim para cada região geométrica descrita são associadas propriedades tais como material que a constitui, fontes de campo e condições de contorno do problema.

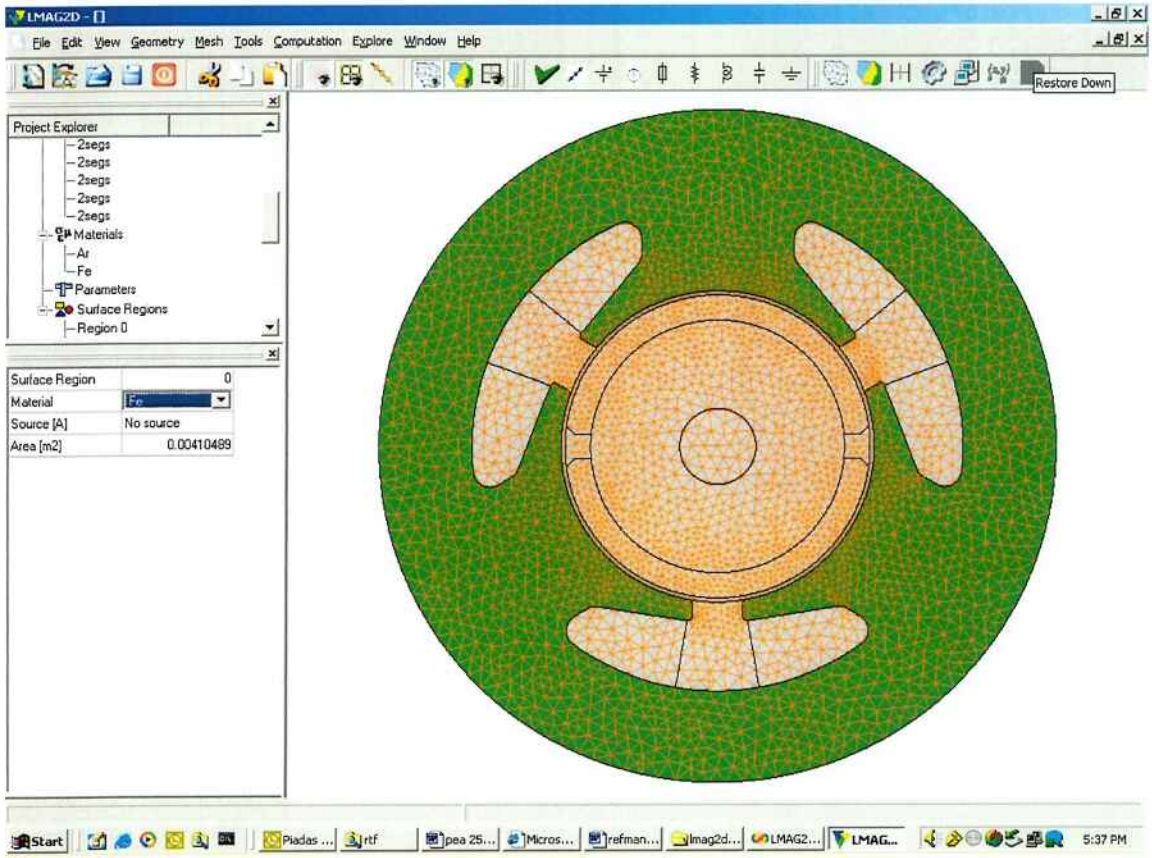


Figura 1 - Definição de geometria, materiais, malha, etc...

A partir da divisão do problema em equações parte-se para a resolução de um sistema linear ou não-linear, geralmente esparso (com poucos elementos não nulos). Esta resolução pode ser direta ou iterativa com coeficientes reais ou complexos.

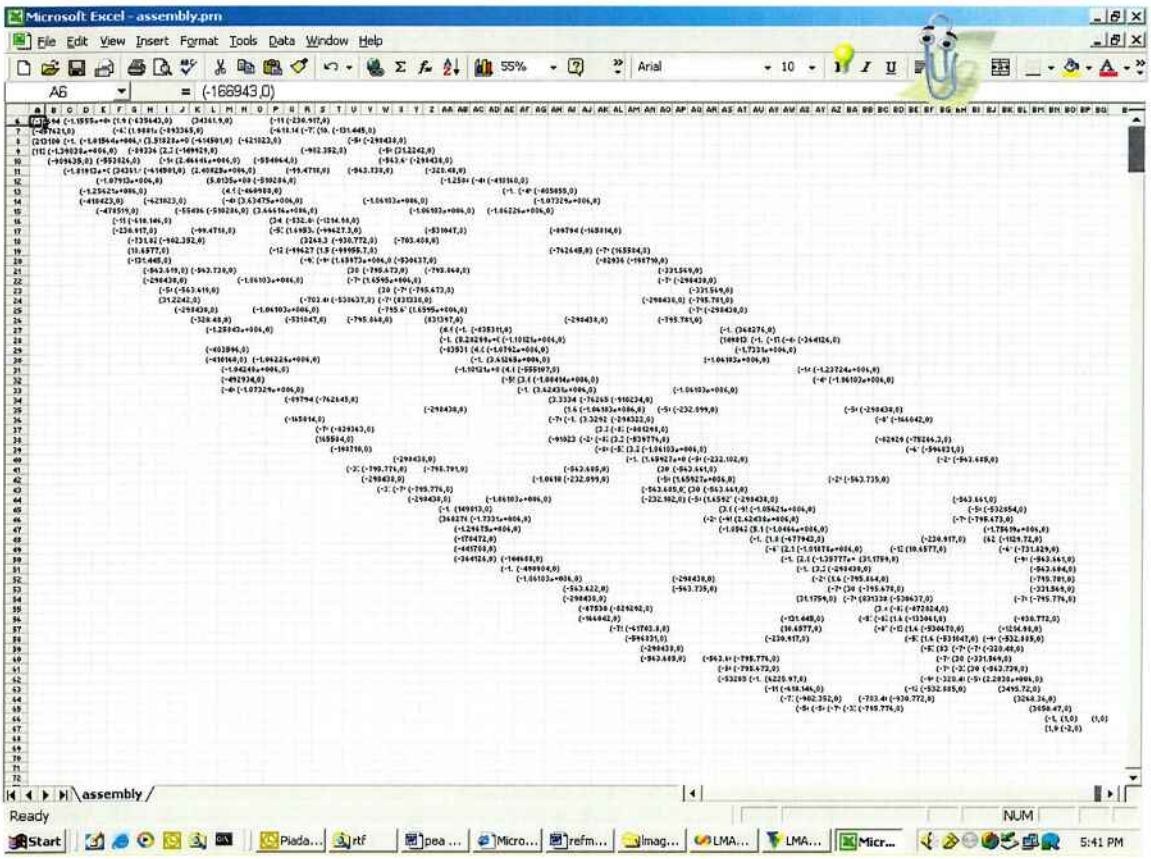


Figura 2 - Sistema linear complexo e esparso (70 elementos)

De posse dos dados calculados no processamento é possível representar os resultados sob a forma de linhas equipotenciais ou mapa de cores de intensidade de campo bem como o cálculo das grandezas citadas anteriormente. Este procedimento é denominado pós-processamento.

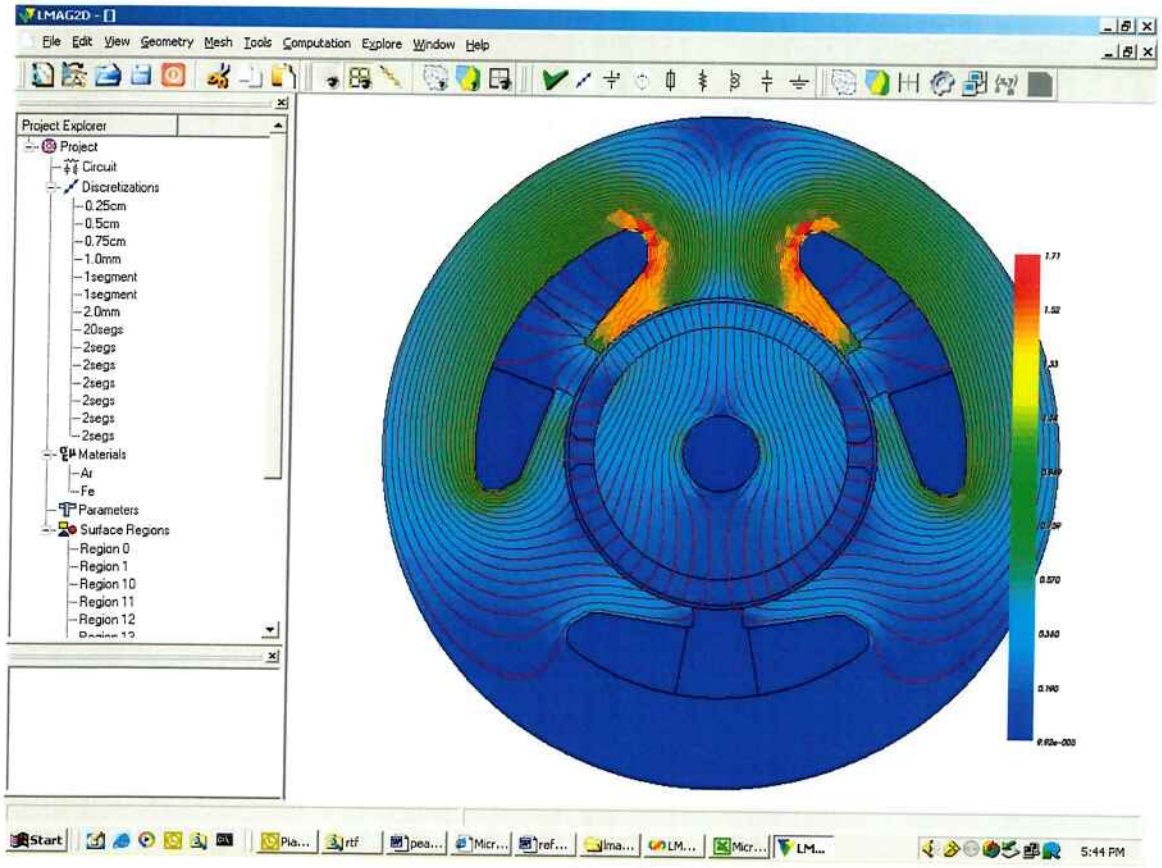


Figura 3 - Equipotenciais e mapa de intensidade de campo

4. O Circuito Elétrico

Usamos as Leis de Kirchoff para resolver os circuitos elétricos. A primeira lei diz que todas as correntes entrando ou saindo de um nó quando somadas se anulam. Ou seja, a corrente não se acumula nos nós do circuito. Analogamente, a segunda lei impõe que dado um laço de circuito a soma das tensões e nos ramos é igual à zero [2].

Assim, conhecendo as relações entre corrente e tensão dos elementos fundamentais pode-se resolver qualquer tipo de circuito elétrico a partir de métodos de análise descritos adiante.

4.1. Relação entre tensão e corrente nos bipolos fundamentais

A seguir veremos a relação entre tensão e corrente dos principais bipolos (ou elementos de circuito neste documento) e suas constantes. São eles:

- Resistores
- Capacitores
- Indutores

4.1.1. Resistores

No resistor tensão e corrente são variáveis diretamente proporcionais. Assim as relações:

$$v = Ri$$
$$i = Gv \qquad (1)$$

Definem as seguintes constantes:

R = Resistência elétrica (ohms)

G = Condutância elétrica (siemens)

4.1.2. Capacitores

O capacitor é capaz de armazenar cargas elétricas com relação à tensão aplicada, em um instante de tempo, em seus terminais. Desta forma o comportamento do capacitor pode ser expresso por:

$$q(t) = C \cdot v(t) \tag{2}$$

Onde,

C = Capacitância elétrica [F]

Derivando a equação acima e considerando:

$$i(t) = \frac{q(t)}{dt} \tag{3}$$

Onde q(t) é a carga instantânea do capacitor.

Obtemos a relação:

$$i(t) = C\frac{dv(t)}{dt} \tag{4}$$

Isolando v(t) na expressão acima:

$$v(t) = \frac{1}{C}\int_{t_0}^{t} i(\tau)d\tau + v(t_o) \tag{5}$$

Onde v(t0) representa a tensão inicial do capacitor.

Portanto a tensão no capacitor é proporcional a integral da corrente que o atravessou a menos da tensão inicial.

4.1.3. Indutores

O indutor é capaz de armazenar energia magnética através da corrente que o atravessa. Logo podemos definir o comportamento de um indutor da seguinte forma:

$$\psi = L(i) \tag{6}$$

Onde,

Ψ = Fluxo de Indução Magnética (Wb)

L = Indutância elétrica.

Pela lei de Faraday a variação de fluxo de indução magnética dá origem a uma tensão:

$$v(t) = L\frac{di(t)}{dt} \tag{7}$$

Resolvendo a expressão acima em relação à di e integrando no tempo temos:

$$i(t) = \frac{1}{L}\int_{t_0}^{t} v(\tau)d\tau + i(t_o) \tag{8}$$

Onde i(t0) representa a corrente inicial do indutor.

Portanto a corrente no indutor é proporcional a integral da tensão aplicada a menos da corrente inicial

4.2. A Análise Nodal Modificada

A NA se apresentou mais vantajosa computacionalmente que a análise de malhas por isto esta foi escolhida para a implementação. Seria muito complicado numerar as malhas de um circuito ao passo que é simples a numeração de seus nós. Além disso, a AM possui uma limitação: apenas circuitos planares podem ser resolvidos por este método.

Foi implementado mais especificamente a ANM, pois apesar de produzir matrizes maiores que a AN simples é mais fácil de ser implementada visto que não requer transformação de fontes de tensão para fontes de corrente.

4.3. Algoritmo de solução

Para a montagem da Matriz de Admitâncias Nodais [Y] e o Vetor de Componentes Ativos do Circuito [Ic] usa-se o seguinte algoritmo de inspeção do circuito elétrico:

Tabela 1 - Inspeção da matriz [Y] de admitâncias nodais

Termo	Valor
Yii	Somatória das admitâncias dos ramos conectados ao nó **i**
Yij e Yji	Somatória das admitâncias conectadas entre os nós **i** e **j**, com sinal negativo.
Yim e Ymi	1, se o nó **i** for o terminal negativo da fonte de tensão **m**.
	-1, se o nó **i** for o terminal positivo da fonte de tensão **m**.

Tabela 2 - Inspeção da matriz [Ic] de componentes ativos

Termo	Valor
Ici	Somatória das fontes de corrente conectadas ao nó **i**.
Icm	Valor da fonte de tensão m, com sinal negativo.

4.4. Caso Teste

Tomemos como exemplo o circuito, representado na Figura 1, para o teste do algoritmo implementado.

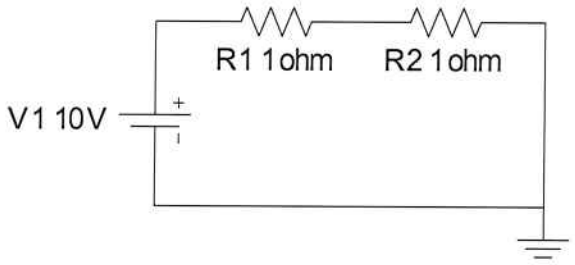


Figura 4 - Circuito elétrico de teste

4.4.1. Equacionamento Teórico

Por inspeção temos que:

$$Y = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 2 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

$$Ic = \begin{bmatrix} 0 \\ 0 \\ -10 \end{bmatrix}$$

Sendo que:

[Y] [V] = [Ic]

$$[V] = \begin{bmatrix} 10 \\ 5 \\ 5 \end{bmatrix}$$

4.4.2. Resultados Obtidos

A partir do programa **TesteCircuito**, cujo código fonte será apresentado no próximo item, obteve-se a seguinte saída:

```
Entrou na EletricCircuit::solve
Criou as matrizes de tamanho 3
Varreu os elementos

Admitancias nodais
(1.000,0.000)     (-1.000,0.000)    (-1.000,0.000)
(-1.000,0.000)    (2.000,0.000)     (0.000,0.000)
(-1.000,0.000)    (0.000,0.000)     (0.000,0.000)

Fontes de corrente
(0.000,0.000)
(0.000,0.000)
(-10.000,0.000)

Tensoes Nodais
(10.000,0.000)
(5.000,0.000)
(5.000,0.000)

Saiu de EletricCircuit::solve
```

Esta saída, por sua vez, confere com o esperado.

4.4.3. Programa em C++

```
/// TesteCircuito.cpp
/// @autor Fernando Jaruche Nunes
/// @date 12/06/2004
#include <lmaglib/circuit/EletricCircuit.h>
#include <lmaglib/circuit/CircuitNode.h>
#include <lmaglib/circuit/Resistor.h>
#include <lmaglib/circuit/VoltageSource.h>
using namespace lmaglib::circuit;

int main(int argc, char* argv[])
{
      EletricCircuit *circuit = new EletricCircuit();

      CircuitNode *no1 = new CircuitNode();
      CircuitNode *no2 = new CircuitNode();
      CircuitNode *no3 = new CircuitNode();

      Resistor *r1 = new Resistor(circuit, 1, no1, no2);
      Resistor *r2 = new Resistor(circuit, 1, no2, no3);
      VoltageSource *v1 = new VoltageSource(circuit, 10, no1, no3);

      no3->setgnd();
      circuit->solve();
      return 0;
}
```

5. Formulação e Desenvolvimento

5.1. Acoplamento Fraco e Forte

O termo "acoplamento" é usado quando um sistema ou formulação é definido em múltiplos domínios, possivelmente coincidente envolvendo variáveis interdependentes que não podem ser determinadas com uma só equação [3][4].

É possível dividir os problemas acoplados em dois grupos: acoplamento forte e fraco. A "força" do acoplamento está relacionada ao nível de dependência entre as variáveis do problema.

O acoplamento forte é o mesmo que um acoplamento total. As equações são combinadas em matrizes e resolvidas simultaneamente. Isto implica que os termos de acoplamento (que definem a junção entre os domínios de estudo) são, também, termos desta matriz.

Já no acoplamento fraco o problema pode ser entendido como um algoritmo em cascata onde os as equações de cada domínio envolvido são resolvidos em passos sucessivos alternados. O acoplamento é dado à medida que a solução de uma equação é transferida como parâmetro para outra antes de se resolver novamente.

Problemas em que os as variáveis dependem umas das outras de forma não linear geralmente requerem um acoplamento forte. Visto que uma pequena variação em uma variável de um subproblema pode gerar uma grande

mudança no problema de estudo. Desta maneira é muito difícil garantir a convergência de um método iterativo como o acoplamento fraco.

Por sua vez o acoplamento fraco é perfeitamente aceitável em situações em que os domínios de estudo estão bem separados. Por exemplo, em um estudo eletromecânico as constantes de tempo mecânicas do problema são muito maiores que as elétricas, assim é possível analisar o circuito elétrico imaginando uma situação mecânica estacionária.

5.2. Acoplamento em máquinas elétricas

Nos problemas de simulação de máquinas elétricas freqüentemente são usadas fontes de tensão para alimentar os enrolamentos cuja corrente produz campo magnético. Mas a corrente que circula no enrolamento é conseqüência não só da tensão aplicada, mas também, da impedância. Por sua vez a impedância do enrolamento é calculada através do MEF. Logo é necessário resolver o MEF, cuja uma ou mais correntes são incógnitas, em conjunto com um circuito que ao menos uma impedância não se sabe o valor. A resolução iterativa dos dois métodos é o acoplamento fraco.

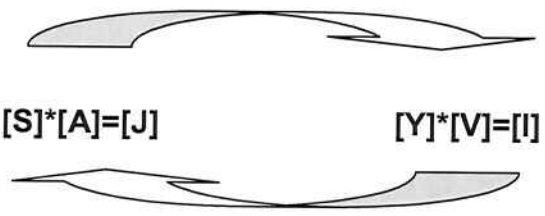


Figura 5 - Acoplamento Fraco

O acoplamento forte, foco deste trabalho, se dá unificando em uma só formulação o Método dos Elementos Finitos (MEF), a Análise Nodal Modificada (ANM) e as relações de acoplamento como podemos observar na Equação 2 que será detalhada adiante.

$$\begin{bmatrix} [MEF] & Acopl. \\ Acopl. & Acopl. \end{bmatrix} \begin{bmatrix} [A] \\ [Y] \end{bmatrix} \begin{bmatrix} [V] \end{bmatrix} = \begin{bmatrix} [J] \\ [I] \end{bmatrix}$$

Figura 6 - Formato da matriz de acoplamento forte

5.3. Regiões Ferromagnéticas e não Condutoras

Ambas as regiões ferromagnéticas e não condutoras não são consideradas no equacionamento, pois não conduzem no sentido perpendicular à geometria.

5.4. Condutores Filiformes

Buscando obter uma relação entre a queda de tensão ΔV nos condutores e V as tensões nodais do circuito elétrico, podemos definir a Matriz de Transformação [T] de modo que [5]:

Tabela 3 - Inspeção da matriz [T] de transformação

Termo	Valor
Tki	1, se o nó **i** for o terminal de entrada da corrente do componente **k**.
	-1, se o nó **i** for o terminal de saída da corrente do componente **k**.

De forma a obter a relação:

$$[\Delta V] = [T][V] \quad (9)$$

e também:

$$[Imef] = -[T]^T[I] \quad (10)$$

Onde I é o vetor formado pelas correntes que percorrem as regiões condutoras do domínio.

Assim,

$$[Y][V] = [Ic] + [Imef] \quad (11)$$

Pode ser expressa por:

$$[Y][V] = [Ic] - [T]^T[I] \quad (12)$$

Seguindo o desenvolvimento feito em [X] chegamos na seguinte equação:

$$\begin{bmatrix} [S] & -[C'] & [0] \\ -[C']^T & -[R'] & [T] \\ [0] & [T]^T & -[Y] \end{bmatrix} \cdot \begin{bmatrix} [A] \\ [I] \\ [V] \end{bmatrix} = \begin{bmatrix} [0] \\ [0] \\ [I_C] \end{bmatrix} \quad (13)$$

Onde:

[S] = Matriz dos elementos do MEF

$$[C] = \frac{L \cdot Ns}{Sk} \int_{Sk} Ni dS \quad (14)$$

$$[R] = \frac{L \cdot Ns^2}{\lambda \cdot Sk^2} \int_{Sk} \frac{1}{\sigma} dSk \quad (15)$$

Ni = função de forma do elemento

Ns = número de condutores (espiras)

Sk = área do condutor

L = profundidade do condutor

λ = fator de empacotamento dos condutores

Os métodos para o calculo do vetor C foram fornecidos para este trabalho e sua implementação segue em anexo juntamente com a documentação do código fonte.

6. Programa Desenvolvido

O programa pode ser dividido nos seguintes módulos

- Classes para resolução de circuitos elétricos
- Assembler
- Interface gráfica
    - Desenho do circuito
    - Integração com o LMAG2D
    - Exploração de Resultados

6.1. Diagrama de Classes

Abaixo segue o diagrama simplificado das classes de descrição de circuitos implementada na LMAGLIB em anexo segue a documentação completa de todas as Classes envolvidas neste trabalho juntamente com o código fonte.

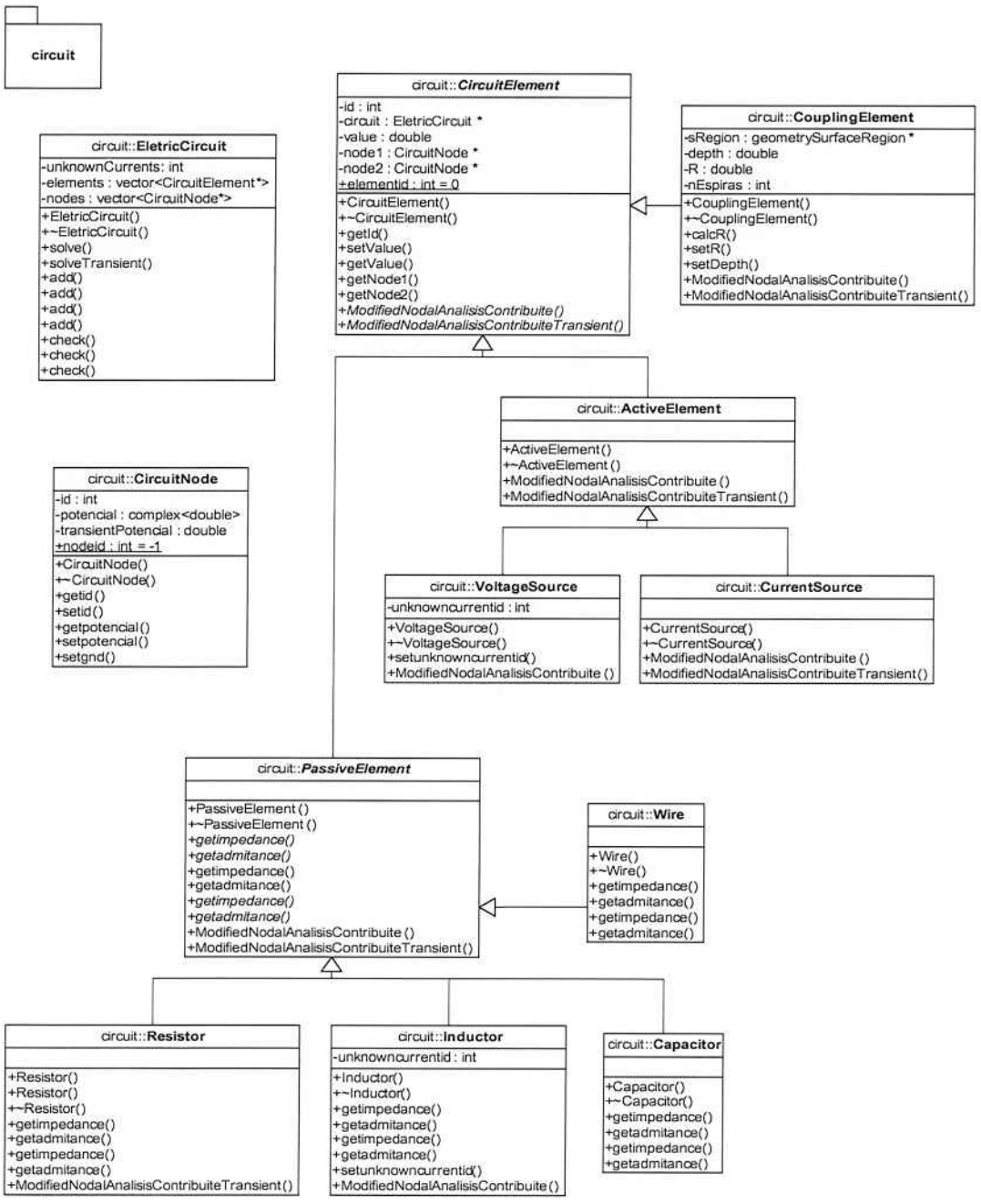


Figura 7 - Diagrama de classes do pacote Circuit

6.1.1. ElectricCircuit

É a classe mais complexa do pacote Circuit destinado para análise de circuitos elétricos. É responsável por alocar dinamicamente espaço para todos os elementos e nós do circuito. Checar se todos os elementos estão ligados, se existe um nó de terra e a numeração dos nós também são rotinas dessa classe.

6.1.2. CircuitNode

Esta classe descreve um nó do circuito elétrico. Ela guarda o número identificador do nó e seu potencial elétrico. Um nó é considerado ligado ao terra quando tem um número negativo como identificador.

6.1.3. CircuitElement

Esta é a classe que descreve as características comuns de todos os elementos de circuito, como dois nós, um valor nominal e número identificador.

6.1.3.1. CouplingElement

Esta é a classe do elemento que faz a relação entre o MEF e o circuito elétrico. Possui o número identificador da região que ele representa no MEF bem como outros parâmetros como profundidade de domínio, resistência ôhmica, etc. Esta classe será detalhada adiante.

6.1.3.2. ActiveElement

Esta é a classe que descreve as características comuns de todos os elementos ativos, como fontes de tensão e corrente. É importante lembrar que nesta implementação os elementos ativos usam a convenção do receptor de tensão e corrente.

6.1.3.2.1. VoltageSource

É a classe que descreve uma fonte de tensão senoidal bem como esta contribui para a matriz de admitâncias nodais e o vetor de correntes do circuito.

6.1.3.2.2. VoltageSource

É a classe que descreve uma fonte de corrente senoidal bem como esta contribui para o vetor de correntes do circuito.

6.1.3.3. PassiveElement

Esta é a classe que descreve as características comuns de todos os elementos passivos. E ainda como estes de maneira geral contribuem para a matriz de admitâncias nodais. Cabendo as classes específicas calcularem suas admitâncias.

6.1.3.3.1. Wire

Esta classe define um curto circuito entre dois nós. Este elemento foi derivado de PassiveElement por uma mera questão computacional.

6.1.3.3.2. Resistor

Esta é a classe que descreve o bipolo fundamental resistor e como se calcula sua admitância a partir de seu valor nominal.

6.1.3.3.3. Inductor

Esta é a classe que descreve o bipolo fundamental indutor e como se calcula sua admitância a partir de seu valor nominal.

6.1.3.3.4. Capacitor

Esta é a classe que descreve o bipolo fundamental capacitor e como se calcula sua admitância a partir de seu valor nominal.

6.2. Implementando novos elementos

Neste exemplo vemos como é simples implementar um novo elemento de circuito derivando a classe CircuitElement. Apenas é preciso programar as funções `MNAContribuite`, ou seja, basta descrever qual é a contribuição do elemento para as matrizes [Y] (`nodaladmitancematrix`) e [Ic] (`currentsourcesvector`).

Segue abaixo a implementação completa da fonte de tensão que contribue tanto para a matriz [Y] quanto para a matriz [Ic]. Para mais informações consulte a documentação do código fonte no Anexo A.

VoltageSource.h

```
#ifndef LMAGLIB_CIRCUIT_VOLTAGE_SOURCE_H
#define LMAGLIB_CIRCUIT_VOLTAGE_SOURCE_H

// Standard included files

// lmaglib included files
#include <lmaglib/circuit/Circuit.h>
#include <lmaglib/circuit/ActiveElement.h>

// The namespace of VoltageSource
namespace lmaglib {
	namespace circuit {

class VoltageSource :
	public ActiveElement
{
	int unknowncurrentid;
public:
	VoltageSource(ElectricCircuit *circuit, int value = 1,
CircuitNode *node1 = NULL, CircuitNode *node2 = NULL);
	~VoltageSource(void);

	/// Set the unknown current identification number.
	/// @param id The unknown current identification number.
	inline void setUnknownCurrentID(int id) { unknowncurrentid = id;
return; }

	void MNAContribuite(CMatrix&);

	void MNAContribuite(CVector&);

	void MNAContribuiteTransient(RMatrix&);

	void MNAContribuiteTransient(RVector&);
};
	} // circuit
} // lmaglib

#endif // LMAGLIB_CIRCUIT_VOLTAGE_SOURCE_H
```

VoltageSource.cpp

```
#include <lmaglib/circuit/VoltageSource.h>
#include <lmaglib/circuit/ElectricCircuit.h>

lmaglib::circuit::VoltageSource::VoltageSource(ElectricCircuit
*circuit, int value, CircuitNode *node1, CircuitNode *node2)
      : ActiveElement(circuit, value, node1, node2)
{
}

lmaglib::circuit::VoltageSource::~VoltageSource(void)
{
}

void lmaglib::circuit::VoltageSource::MNAContribuite(CMatrix
&nodaladmitancematrix)
{
      ActiveElement::MNAContribuite(nodaladmitancematrix);

      /* Faz agora as particularidades da fonte de tensão */
      int node1 = getNode1()->getid();
      int node2 = getNode2()->getid();

      int position = this->circuit->numOfNodes() +
                        this->unknowncurrentid;

      if (node1 >= 0) {
            nodaladmitancematrix.add(position, node1, 1);
            nodaladmitancematrix.add(node1, position, 1);
      }
      if (node2 >= 0) {
            nodaladmitancematrix.add(position, node2, -1);
            nodaladmitancematrix.add(node2, position, -1);
      }
}
void lmaglib::circuit::VoltageSource::MNAContribuite(CVector
&currentsourcesvector)
{
      ActiveElement::MNAContribuite(currentsourcesvector);

      /* Faz agora as particularidades da fonte de tensão */
      int position = this->circuit->numOfNodes() +
                        this->unknowncurrentid;
      currentsourcesvector[position] += getValuePhasor();
}

void lmaglib::circuit::VoltageSource::MNAContribuiteTransient(RMatrix
&nodaladmitancematrix)
{
```

```
	ActiveElement::MNAContribuiteTransient(nodaladmitancematrix);

	/* Faz agora as particularidades da fonte de tensão */
	int node1 = getNode1()->getid();
	int node2 = getNode2()->getid();

	int position = this->circuit->numOfNodes() +
			this->unknowncurrentid;

	if (node1 >= 0) {
		nodaladmitancematrix.add(position, node1, 1);
		nodaladmitancematrix.add(node1, position, 1);
	}
	if (node2 >= 0) {
		nodaladmitancematrix.add(position, node2, -1);
		nodaladmitancematrix.add(node2, position, -1);
	}
}
void lmaglib::circuit::VoltageSource::MNAContribuiteTransient(RVector
&currentsourcesvector)
{
	ActiveElement::MNAContribuiteTransient(currentsourcesvector);

	/* Faz agora as particularidades da fonte de tensão */
	int position = this->circuit->numOfNodes() +
			this->unknowncurrentid;
	currentsourcesvector[position] += getValue(circuit->getTime());
}
```

6.3. Elemento de Acoplamento

O elemento de acoplamento é o que faz a ligação entre o circuito elétrico e a região condutora do MEF. Este elemento se diferencia dos demais elementos de circuito pelas seguintes propriedades:

Profundidade (L)

O problema estudado é sempre uma secção bidimensional (região) que explora uma simetria específica de um problema real. Assim este atributo representa o tamanho do sólido simulado na terceira dimensão.

Fator de enchimento (λ)

Este atributo representa o empacotamento de condutores filiformes em uma dada região acoplada. Para condutores maciços o valor deste atributo é 1.

Número de espiras (Ns)

Novamente este atributo é válido apenas para o caso filiforme. Representa o número de espiras que a região condutora possui. Para o caso maciço o valor deste atributo é 1.

Condutividade (σ)

Refere-se ao valor da condutividade elétrica do material que constitui a região condutora do dispositivo analisado. Este valor é obtido através da propriedade física da região associada ao elemento de circuito elétrico.

Resistência (R)

Pode ser calculada por:

$$R = \frac{L \cdot Ns^2}{\sigma \cdot \lambda \cdot S} \qquad (16)$$

Ou mesmo ser atribuída como um valor obtido analiticamente. Desta forma não são necessários os demais parâmetros listados acima.

Região do MEF associada (S)

Para cada elemento de acoplamento elétrico deve existir uma região do MEF associada. Assim, este é o atributo responsável por garantir este vínculo. Da mesma forma existe o mesmo tipo de vínculo na classe de propriedade física de uma região. Relacionando então a região com o elemento elétrico acoplado correspondente.

6.4. Assemblador (Montador)

O sistema de equações mostrado na (13) é montado a partir de cada um dos elementos do MEF e de circuito. Uma classe *"assembler"* é responsável por este processo de montar as equações reunindo as matrizes e vetores locais de cada elemento em um único sistema de equações.

A partir do assembler já construído anteriormente (unicamente para a construção da matriz [S] do MEF) foi implementado uma nova classe que reúne as equações do MEF com as equações de circuitos através dos vetores de acoplamento [C] (14) matriz de resistencias [R] (15) e matriz de transformação [T] (tabela 3) resultando em (13).

6.5. Interface gráfica

6.5.1. Objetivo

A entrada de dados para as simulações de máquinas elétricas é dada pelas seguintes etapas:

6.5.2. Definição da geometria da máquina

A entrada da geometria da máquina pode ser elaborada em qualquer software de CAD que exporte arquivos DXF. No entanto o cruzamento de linhas é proibido, pois gera erro quando da definição das regiões.

6.5.3. Rotina de geração das regiões da geometria

Uma vez devidamente importada a geometria para o programa LMAG2D é gerada uma região geométrica para cada contorno fechado da geometria.

6.5.4. Definição dos materiais de cada região

Uma vez definidas as regiões da geometria o usuário pode definir quais são os materiais que as compõe.

6.5.5. Definição das discretizações

Para se obter resultados precisos, com economia de processamento, é importante definir uma boa discretização. Em estudos eletromagnéticos é sempre uma boa idéia dar maior precisão para as pontas e entreferro.

6.5.6. Definição do circuito elétrico acoplado

Neste estágio definem-se as fontes de excitação do dispositivo simulado bem como os demais componentes do circuito elétrico acoplado.

6.5.7. Definição dos condutores acoplados

Algumas regiões da geometria (ou domínio) são fontes de corrente. A estas fontes deve-se associar o elemento de circuito acoplado equivalente.

6.5.8. Definição das condições de fronteira

É preciso definir os potenciais magnéticos envolvendo a geometria para restringir o domínio de estudo.

6.5.9. Rotinas de geração e apresentação de malha

A partir das discretizações é gerada uma nuvem de pontos que interligados formam a malha que divide o problema.

6.5.10. Rotinas de resolução

Com todos os dados computados parte-se para a resolução do sistema linear do MEF

6.5.11. Rotinas de exploração de resultados

Com o resultado da simulação é possível gerar linhas equipotenciais magnéticas, mapa de cores na geometria e o potencial nos nós do circuito elétrico.

6.6. Integração com o LMAG2D

Visando minimizar o número de caixas de diálogo bem como a produtividade, o LMAG2D possui uma janela de propriedades localizada no canto inferior esquerdo da tela. Nesta janela é possível visualizar e alterar os parâmetros de um ou mais elementos selecionados, por exemplo: regiões, linhas, fios, elementos elétricos, etc.

Seguindo este padrão no módulo de circuitos elétricos, fica fácil a edição dos elementos do circuito. Basta seleciona-lo que a janela de propriedades mostra todos os parâmetros do elemento. Para alterá-lo basta alterar o valor da propriedade desejada e pressionar a tecla "Enter".

Vejamos a edição de um elemento de acoplamento elétrico com o MEF:

Quando selecionado um elemento de acoplamento é possível associar uma região na caixa de propriedades, como mostra a Figura 8.

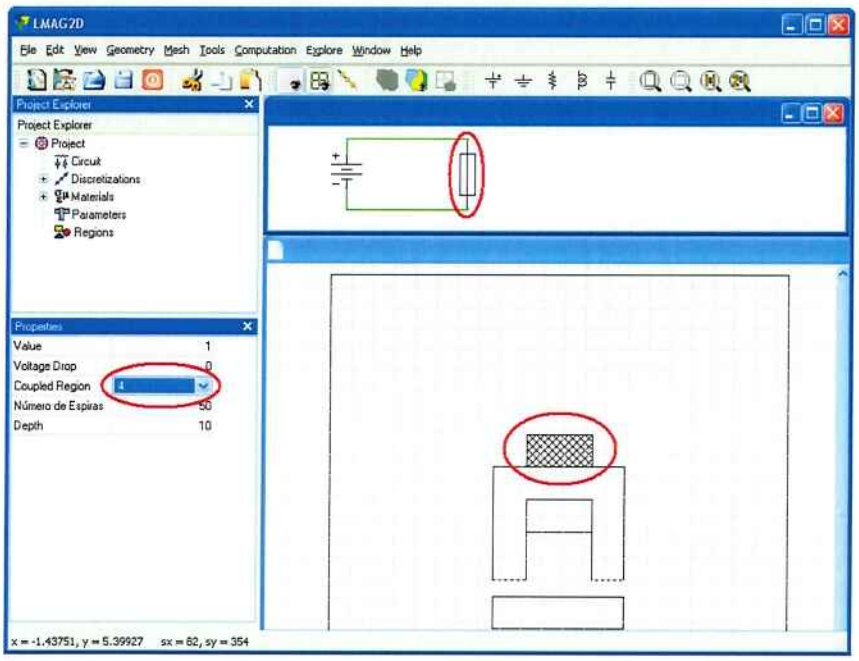


Figura 8 - Caixa de propriedades do elemento acoplado

6.7. Plataforma de Desenvolvimento

A escolha do ambiente de desenvolvimento da Interface gráfica levou em consideração os seguintes aspectos:

- Multiplataforma
- Classes de desenho
- Classes de formulários de interface Homem Máquina

A equipe de desenvolvimento do LMAG2D decidiu por abandonar a antiga versão do programa feito com a biblioteca MFC (Microsoft Foundation Classes) e adotar o Qt como plataforma de desenvolvimento pelos seguintes motivos:

6.7.1. Free Software

Desenvolvida pela TrollTech (http://www.trolltech.com/) a biblioteca Qt é licenciada sob a licença GPL (GNU Public License) (http://www.gnu.org/). Esta licença garante uma série de privilégios tais como o livre acesso ao software, no entanto impõe uma série de obrigações. A mais importante, e relevante para o projeto LMAG2D, é a que diz respeito ao trabalho derivativo. Todo trabalho derivado de um código licenciado sob a GPL deverá ser também licenciado sob esta mesma licença:

> *"Você tem que fazer com que quaisquer trabalhos que você distribua ou publique, e que integralmente ou em partes contenham ou sejam derivados do Programa ou de suas partes, sejam licenciados, integralmente e sem custo algum para quaisquer terceiros, sob os termos desta Licença".*
> Trecho da licença GPL

No entanto, se o Laboratório de Eletromagnetismo Aplicado (LMAG) não concordar ou decidir por distribuir o programa LMAG2D por outros termos que não os da GPL, ainda existe a possibilidade de adquirir a biblioteca Qt licenciada para fins comerciais.

6.7.2. Multiplataforma

Programas desenvolvidos com a biblioteca Qt podem ser compilados e executados em várias plataformas, dentre elas:

- Qt/Windows (MS Windows 95/98/Me, NT4, 2000 e XP)
- Qt/X11 (Linux, Solaris, HP-UX, IRIX, AIX)
- Qt/Mac (Apple Mac OS X)
- Qt/Embedded (embedded Linux, Celulares e Handhelds)

6.7.3. Compatibilidade com o módulo de processamento paralelo

A migração para um sistema compatível com o Unix tal como o Linux foi importante para o módulo de processamento paralelo que usa algumas rotinas específicas desta plataforma.

6.7.4. Riqueza de classes para desenho 2D (QCanvas)

Foram usadas as seguintes classes para a representação do circuito elétrico:

- QCanvasItem – Classe abstrata para todos os itens geométricos.
- QCanvasLine – Segmento de reta. Foi usada para os fios
- QCanvasSprite – Classe para figuras. Foi usada para representar os diversos elementos de circuito.

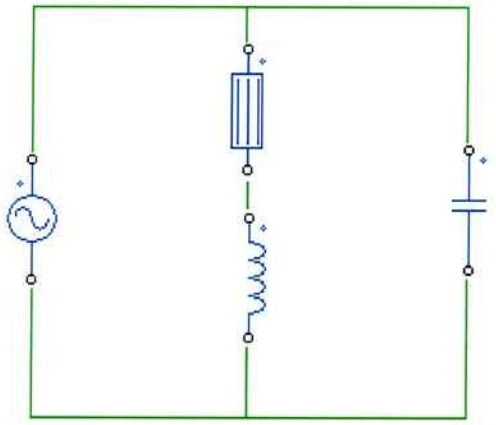


Figura 9 - Circuito contruido com as classes do Qt

6.7.5. Ferramentas de desenvolvimento

Tanto o MS Visual Studio® (Windows) quanto o KDevelop (Linux) possuem ferramentas integradas para o desenvolvimento em Qt. Desta forma, cada programador pode optar pela sua ferramenta favorita de desenvolvimento.

As ferramentas suportadas são:

- Apple ProjectBuilder
- Borland C++
- Compaq C++/DEC cxx
- g++/egcs
- HPaCC
- HPCC
- IBM C++ v. 3.6
- IBM VisualAge C++ v. 5
- Kai C++
- Microsoft Visual C++
- SGI MipsPRO C++

- Sun Workshop C++

6.7.6. Caso de Uso

O programa LMAG2D é desenvolvido simultaneamente por diversos programadores. Apesar de cada um ter escolhido sua ferramenta de desenvolvimento própria e sistemas operacionais diferentes o código fonte funciona da mesma forma para todos eles.

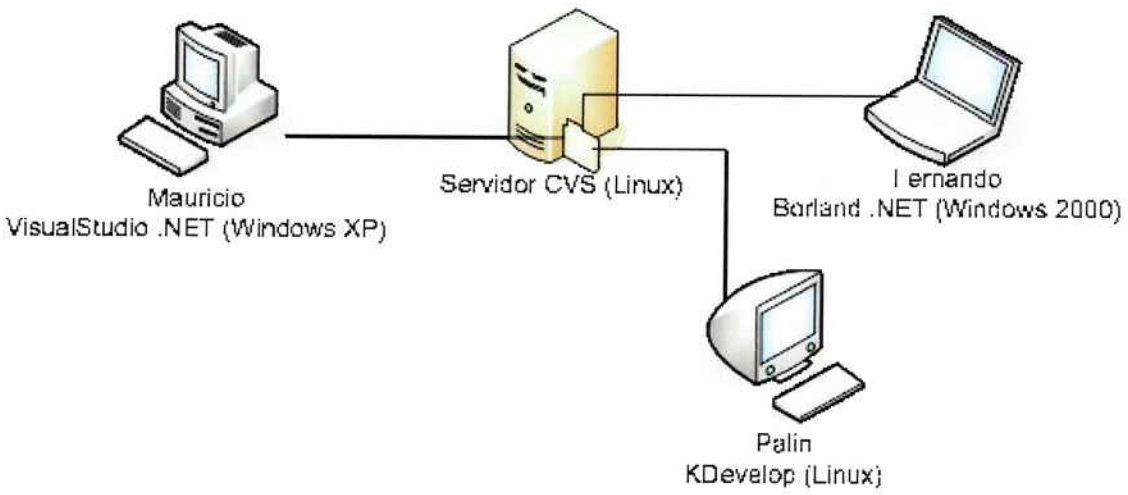


Figura 10 - Diferentes plataformas e OS no desenvolvimento com Qt

Desta forma a contribuição do Mauricio pode ser aproveitada pelo Palin e pelo Fernando. Isto sem a necessidade de alterar o código fonte para adaptá-lo às peculiaridades de cada plataforma de desenvolvimento.

7. Aplicação em casos reais

Conforme dito anteriormente o LMAG2D simula apenas geometrias bidimensionais. Por isso o exemplo a seguir mostra como simplificar os problemas tridimensionais fazendo uso da simetria.

7.1. Eletroímã

A seguir usaremos o clássico exemplo do eletroímã constituído de um núcleo em U e uma bobina que, quando energizada, traciona um outro núcleo I.

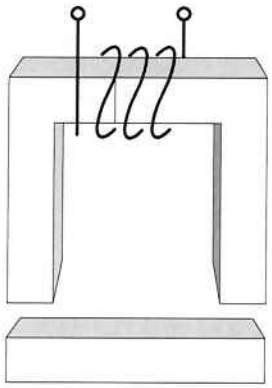


Figura 11 - Eletroímã U

Visando facilitar o equacionamento e simulação, o problema é reduzido para um caso bidimensional. Para tal estabelecemos o seguinte corte que explora as simetrias do problema:

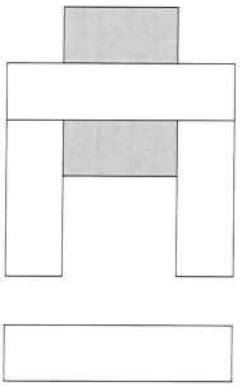


Figura 12 - Representação em corte do eletroímã

Acima observamos a representação bidimensional do problema proposto. Em branco temos o núcleo em U e em cinza os condutores da bobina. Apesar de separados no plano os dois condutores representam um único enrolamento. Desta forma podemos imaginar uma corrente fluindo por esta bobina como entrando em uma das regiões condutoras e saindo pela outra.

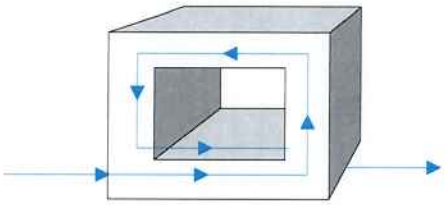


Figura 13 - Representação 3D da bobina

No entanto, esta representação não descreve o encontro dos condutores que só podem ser observados em três dimensões. Esta "borda" não contribui significativamente com campo magnético, mas possui as mesmas características de resistência e indutância do resto da bobina.

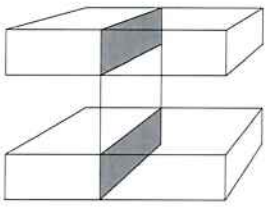


Figura 14 - Regiões 2D simuladas

Assim, para refinar a solução adicionamos ao circuito "indutâncias de cabeça de bobina" ao problema. Estas indutâncias podem ser calculadas

deterministicamente e representam com qualidade os trechos restantes da bobina simulada.

O acoplamento entre os problemas de circuito e elementos finitos se dá através de elementos de ligação representados juntamente os indutores na figura abaixo.

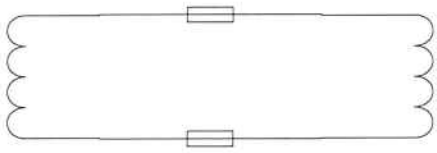


Figura 15 - Circuito elétrico acoplado

Logo cada um desses elementos está associado a uma região do MEF

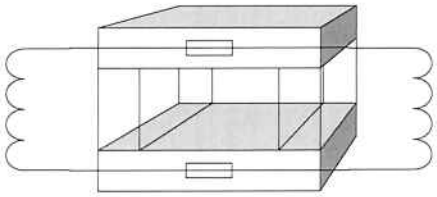


Figura 16 - Acoplamento dos elementos de circuito elétrico

Assim, ligando a bobina a uma fonte senoidal temos o seguinte circuito equivalente:

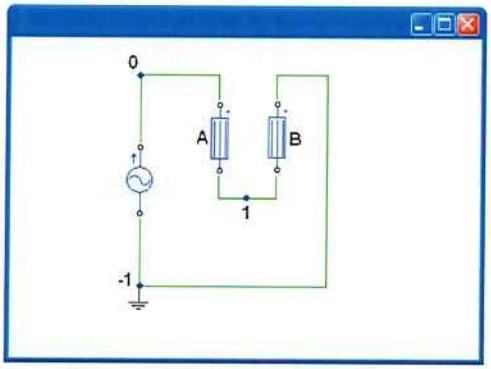


Figura 17 - Fonte de Corrente de 1A alimentando a bobina

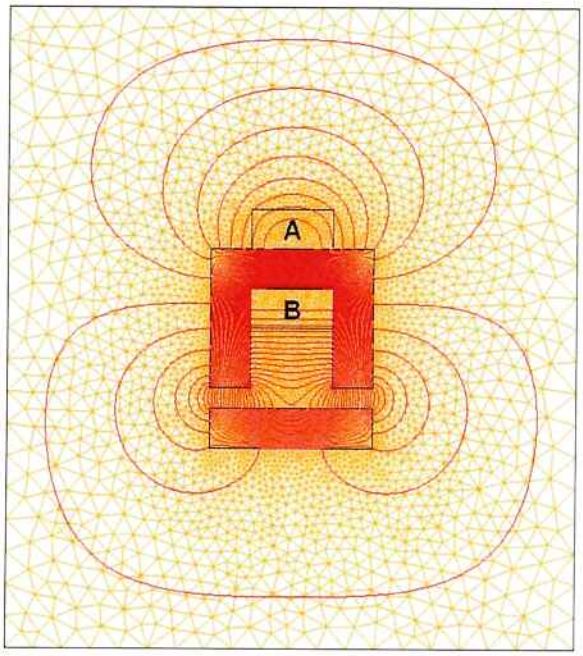


Figura 18 - Equipotenciais - Eletroímã

Tabela 4 - Resultados - Eletroímã

	Real	Imaginário	Módulo	Fase
V0	1.25651	-0.96654	1.58525	-0.6557
V1	0.628253	-0.48327	0.792624	-0.6557

7.2. *Transformador*

O exemplo seguinte é de um transformador ferromagnético com condutores filiformes no primário e secundário num total de 100 espiras. Ligado ao primário uma fonte de 110V e um resistor de 1kΩ enquanto o secundário fica em curto circuito [6].

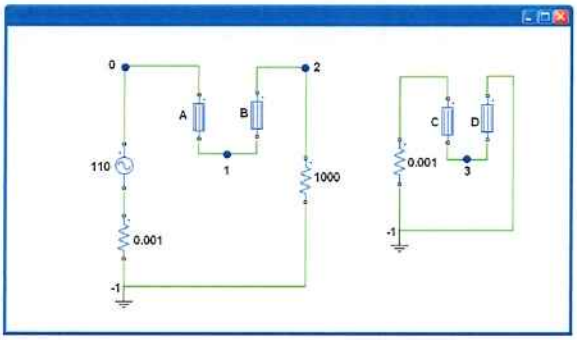


Figura 19 - Fonte de Tensão de 110V no primário com secundário em curto

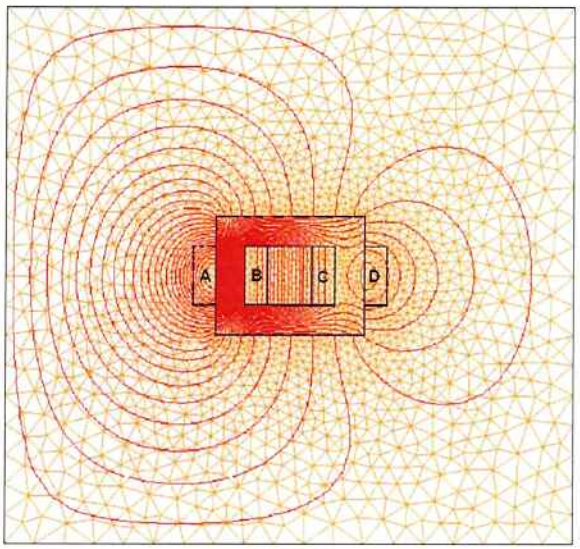


Figura 20 - Equipotenciais – Transformador

Tabela 5 - Resultados - Transformador

	Real	Imaginário	Módulo	Fase
V0	110	0	110	0
V1	109.928	0.07099	110.072	3.14095
V2	109.88	0.181692	110.12	3.13994
V3	0.01196	0.021452	0.024561	2.07942
I0	0.11012	-0.00018	0.11012	-0.00165

8. Conclusão

O programa desenvolvido mostrou de uma implementação simples para o acoplamento entre as equações do Método dos Elementos finitos (MEF) e a Análise Nodal Modificada (ANM) para circuitos de topologias quaisquer.

Soma-se ainda a possibilidade do uso de fontes de tensão e corrente sem a necessidade de transformação.

As simulações feitas a partir do módulo de circuitos implementado para o LMAG2D mostraram algumas aplicações do método para casos reais.

9. Referências Bibliográficas

[1] Cardoso, J. R. "Introdução ao Método dos Elementos Finitos", Publicação Independente.

[2] Orsini, Luiz de Queiroz; Consonni, Denise; "Curso de Circuitos Elétricos", Editora Edgard Blücher LTDA, 2002.

[3] Bastos, João Pedro A.; Sadowski, Nelson; "Electromagnetic Modeling by Finite Element Methods"

[4] Kay Hameyer, Ronnie Belmans, "Numerical Modeling and Design of Electrical Machines and Devices", WIT press, 1999.

[5] Costa, M. C. "Análise Nodal Aplicada no Acoplamento Circuito Elétrico - Elementos Finitos na Magnetodinâmica", Dissertação de Mestrado, EPUSP, 1998.

[6] Costa, M. C.; Nabeta, S. I.; Abe, N. M.; Cardoso, J. R.; "A nodal approach applied to electric circuits coupling in magnetodynamic 2D FEM"; Proceedings of the 11th Conference on the computation of electromagnetic fields - COMPUMAG, vol. 2, 1997, pp 753-754.

[7] Costa, M. C.; Mendes, R. H.; Nabeta, S. I.; Cardoso, J. R.; "Simulation of Induction Machine by 2D Finite Element Method coupled with Electric Circuits using the Modified Nodal Analysis"; Proceedings of the CEFC 98.

[8] Stroustrup, B., "The C++ Programming Language", Addison-Wesley, 1997.

[11] Prosise, J. "Programming Windows 95 with MFC", 1996.

Anexo A – Documentação do Código Fonte

LMAG2D

Fernando Jaruche Nunes
Version 2004
11/29/2004

Table of Contents

LMAG2D Hierarchical Index

LMAG2D Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

LMAG2D Class Index

LMAG2D Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

LMAG2D Namespace Documentation

lmaglib::circuit Namespace Reference

lmaglib::circuit

Detailed Description

Electrical circuits (Fernando Jaruche Nunes).

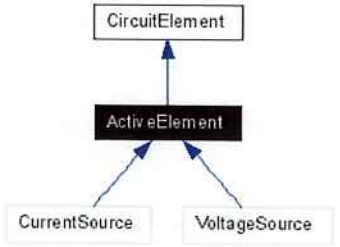
Classes

- class **ActiveElement**
- class **Capacitor**
- class **CircuitElement**
- class **CircuitNode**
- class **CouplingElement**
- class **CurrentSource**
- class **ElectricCircuit**
- class **Inductor**
- class **PassiveElement**
- class **Resistor**
- class **VoltageSource**
- class **Wire**

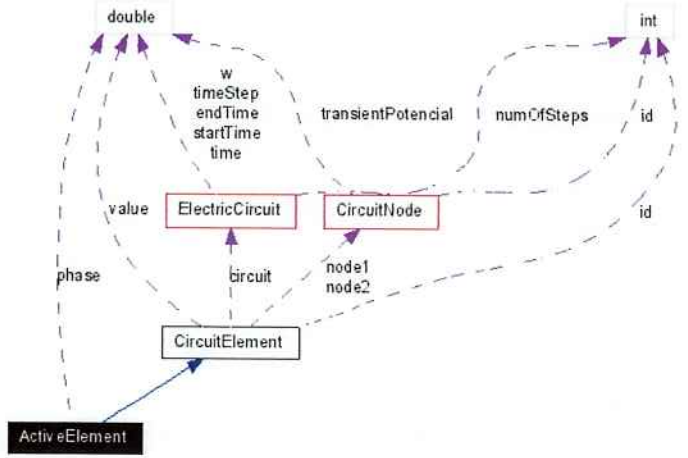
LMAG2D Class Documentation

ActiveElement Class Reference

lmaglib::circuit::ActiveElementInheritance diagram for ActiveElement:



Collaboration diagram for ActiveElement:



Detailed Description

Active element description and base class.

Public Member Functions

- **ActiveElement** (**ElectricCircuit** *circuit, int **value**=1, **CircuitNode** ***node1**=NULL, **CircuitNode** ***node2**=NULL)
- double **getPhase** (void)
- void **setPhase** (double phase)
- double **getValue** (double timeStep)
- double **getValue** (void)
- complex< double > **getValuePhasor** (void)
- void **MNAContribuite** (CMatrix &)
- void **MNAContribuite** (CVector &)
- void **MNAContribuiteTransient** (RMatrix &)
- void **MNAContribuiteTransient** (RVector &)

Private Attributes

- double **phase**

Constructor & Destructor Documentation

ActiveElement (ElectricCircuit * *circuit*, int *value* = 1, CircuitNode * *node1* = NULL, CircuitNode * *node2* = NULL)

This constructor calls **CircuitElement::CircuitElement()** as **CircuitElement** principal constructor.

See also:

CircuitElement::CircuitElement()

```
10 : CircuitElement(circuit, value, node1, node2), phase(0)
11 {
12 }
```

Member Function Documentation

double getValue (void) [inline]

Get element nominal value.

Returns:

Element nominal value.

```
Reimplemented from CircuitElement (p.16).53 { return value; }
```

void MNAContribuite (CVector & *currentsourcesvector*) [virtual]

Calculate the active element contributions to the modified nodal Analisys method. It update nodal admitance matriz and current sources vector in correct places.

Parameters:

nodaladmitancematrix Nodal admitance matrix reference.
currentsourcesvector Current sources vector reference.

See also:

CircuitElement::MNAContribuite()


```
Implements CircuitElement (p.16).37 {
38
39 }
```

void MNAContribuite (CMatrix & *nodaladmitancematrix*) [virtual]

Calculate the active element contributions to the modified nodal Analisys method. It update nodal admitance matriz and current sources vector in correct places.

Parameters:

nodaladmitancematrix Nodal admitance matrix reference.
currentsourcesvector Current sources vector reference.

See also:

CircuitElement::MNAContribuite()

```
Implements CircuitElement (p.16).26 {
27
28 }
```

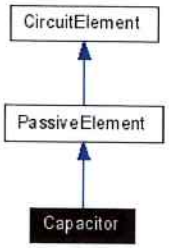
void MNAContribuiteTransient (RMatrix &) [virtual]

Calculate the circuit element contributions to the modified nodal Analisys method in transient simulation.

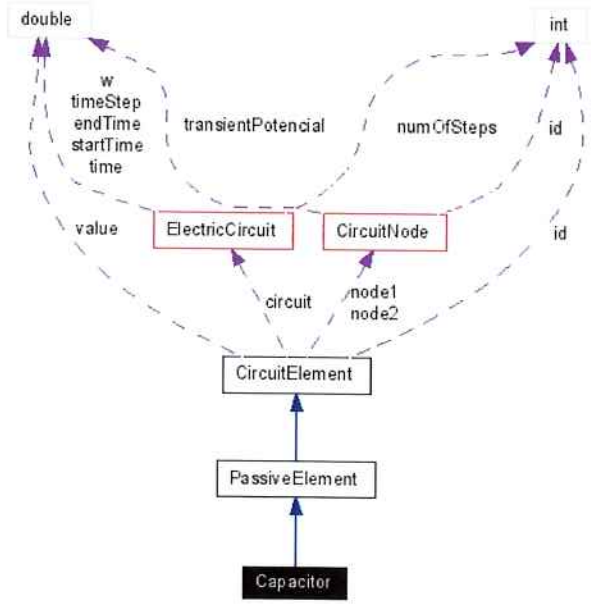
```
Implements CircuitElement (p.16).42 {
43
44 }
```

Capacitor Class Reference

lmaglib::circuit::CapacitorInheritance diagram for Capacitor:



Collaboration diagram for Capacitor:



Detailed Description

Capacitor description.

Public Member Functions

- **Capacitor** (**ElectricCircuit** *circuit, int **value**=1, **CircuitNode** ***node1**=NULL, **CircuitNode** ***node2**=NULL)
- complex< double > **getimpedance** (double w)
- complex< double > **getadmitance** (double w)
- double **getimpedance** (double i, double iLast, double timeStep)
- double **getadmitance** (double i, double iLast, double timeStep)

Constructor & Destructor Documentation

Capacitor (ElectricCircuit * *circuit*, int *value* = 1, CircuitNode * *node1* = NULL, CircuitNode * *node2* = NULL)

See also:

CircuitElement::CircuitElement(CircuitNode *, CircuitNode *)

```
7         : PassiveElement(circuit, value, node1, node2)
8 {
9 }
```

Member Function Documentation

double getadmitance (double *i*, double *iLast*, double *timeStep*) [virtual]

See also:

PassiveElement::getadmitance(double i, double iLast, double timeStep)

```
Implements PassiveElement (p.37).44 {
45          return 1.0/getimpedance(i, iLast, timeStep);
46 }
```

complex< double > getadmitance (double *w*) [virtual]

See also:

PassiveElement::getadmitance(double)

```
Implements PassiveElement (p.37).27 {
28          complex<double> result (0, w*this->getValue());
29          return result;
30 }
```

double getimpedance (double *i*, double *iLast*, double *timeStep*) [virtual]

See also:

PassiveElement::getimpedance(double i, double iLast, double timeStep)

```
Implements PassiveElement (p.38).36 {
37         return (1/getValue())*((i+iLast)/2*timeStep);
38 }
```

complex< double > getimpedance (double *w*) [virtual]

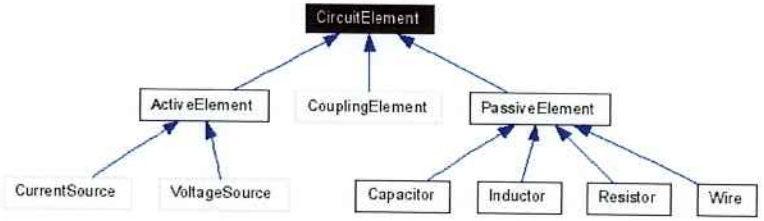
See also:

PassiveElement::getimpedance(double)

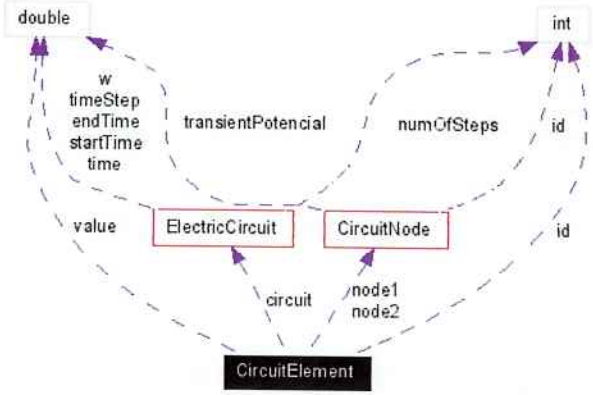
```
Implements PassiveElement (p.38).19 {
20         return 1.0/this->getadmitance(w);
21 }
```


CircuitElement Class Reference

lmaglib::circuit::CircuitElementInheritance diagram for CircuitElement:



Collaboration diagram for CircuitElement:



Detailed Description

Bipole circuit element description and base class.

Public Member Functions

- **CircuitElement** (**ElectricCircuit** *circuit, int **value**=1, **CircuitNode** ***node1**=NULL, **CircuitNode** ***node2**=NULL)
- int **getId** (void)

- void **setID** (int **id**)
- void **setNode1** (**CircuitNode** *node)
- void **setNode2** (**CircuitNode** *node)
- void **setValue** (double)
- double **getValue** (void)
- double **getValue** (int timeStep)
- **CircuitNode** * **getNode1** ()
- **CircuitNode** * **getNode2** ()
- virtual void **MNAContribuite** (CMatrix &)=0
- virtual void **MNAContribuite** (CVector &)=0
- virtual void **MNAContribuiteTransient** (RMatrix &)=0
- virtual void **MNAContribuiteTransient** (RVector &)=0

Static Public Member Functions

- void **MNAContribuite** (CMatrix &nodaladmitancematrix, CVector &currentsourcesvector)
- void **MNAContribuiteTransient** (RMatrix &nodaladmitancematrix, RVector &currentsourcesvector)

Protected Attributes

- int **id**
- **ElectricCircuit** * **circuit**
- double **value**
- **CircuitNode** * **node1**
- **CircuitNode** * **node2**

Constructor & Destructor Documentation

CircuitElement (ElectricCircuit * *circuit*, int *value* = `1`, CircuitNode * *node1* = `NULL`, CircuitNode * *node2* = `NULL`)

circuit element constructor. It calls this->**CircuitElement()** and associate node1 and node2 as element nodes.

Parameters:

node1 The first element node
node2 The second element node

```
11 {
12         id = 0;
13
14         this->value = value;
15
16         this->circuit = circuit;
17         this->circuit->add(this);
18
19         if (node1 == NULL) {
20                 this->node1 = new CircuitNode();
21         }
22         else {
23                 this->node1 = node1;
24         }
25
26         if (node2 == NULL) {
27                 this->node2 = new CircuitNode();
28         }
29         else {
30                 this->node2 = node2;
31         }
32
```



```
33         circuit->add(this->node1);
34         circuit->add(this->node2);
35 }
```

Member Function Documentation

int getId (void)

Get element identification number.

Returns:

The element identification number.

```
46 {
47         return id;
48 }
```

double getValue (void) `[inline]`

Get element nominal value.

Returns:

Element nominal value.

```
Reimplemented in ActiveElement (p.9).76 { return value; }
```

void MNAContribuite (CMatrix & *nodaladmitancematrix*, CVector & *currentsourcesvector*) `[inline, static]`

Calculate the circuit element contributions to the modified nodal Analisys method.

```
98         {
99                 MNAContribuite(nodaladmitancematrix, currentsourcesvector);
100        }
```

virtual void MNAContribuite (CVector &) `[pure virtual]`

Calculate the circuit element contributions to the modified nodal Analisys method.

Implemented in ActiveElement **(*p.9*), and** PassiveElement **(*p.38*).virtual void MNAContribuite (CMatrix &) `[pure virtual]`**

Calculate the circuit element contributions to the modified nodal Analisys method.

Implemented in ActiveElement **(*p.10*),** Inductor **(*p.35*), and** PassiveElement **(*p.38*).virtual void MNAContribuiteTransient (RMatrix &) `[pure virtual]`**

Calculate the circuit element contributions to the modified nodal Analisys method in transient simulation.

Implemented in ActiveElement **(*p.10*),** PassiveElement **(*p.39*), and** Resistor **(*p.51*).void setValue (double *val*)**

Set element nominal value.

Parameters:

val Element nominal value.

```
65 {
66         value = val;
67 }
```

Member Data Documentation

int id `[protected]`

The circuit element identification number.

CircuitNode* node1 `[protected]`

First element node. Current in this element is NEGATIVE if it go out throw this node.

CircuitNode* node2 `[protected]`

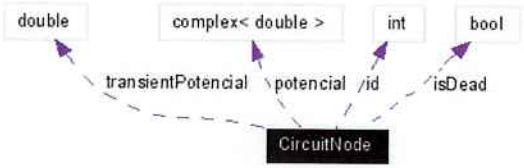
Second element node. Current in this element is POSITIVE if it go out throw this node.

double value `[protected]`

The nominal element value.

CircuitNode Class Reference

lmaglib::circuit::CircuitNodeCollaboration diagram for CircuitNode:



Detailed Description

circuit node description.

Public Member Functions

- **CircuitNode** (void)
- int **getid** (void)
- void **setid** (int)
- complex< double > **getpotencial** (void)
- void **setpotencial** (complex< double >)
- void **setgnd** (void)

Public Attributes

- bool **isDead**

Private Attributes

- int **id**
- complex< double > **potencial**
- double **transientPotencial**

Constructor & Destructor Documentation

CircuitNode (void)

circuit node constructor. It increses this->nodeid every time it is called.

```
8 {
9         this->id = 0;
10 }
```

Member Function Documentation

int getid (void)

Get node identification number.

Returns:

The node identification number.

```
21 {
22         return id;
23 }
```

complex< double > getpotencial (void)

Get node potencial value in volts.

Returns:

The node potencial value in volts.

```
39 {
40         return this->potencial;
41 }
```

void setgnd (void)

Set node as ground setting identification number as zero.

```
56 {
57         this->id = -1;
58 }
```

void setid (int *id*)

Set node identification number.

Parameters:

id Node identification number.

```
30 {
31         this->id = id;
32 }
```

void setpotencial (complex< double > *potencial*)

Set node potencial value in volts.

Parameters:

potencial The node potencial value in volts.

```
48 {
49         this->potencial = potencial;
50 }
```

Member Data Documentation

int id [private]

The circuit node identification number.

complex<double> potencial [private]

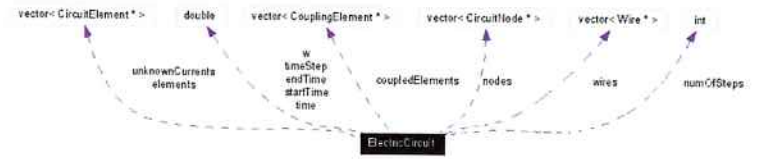
The node potencial value in volts.

double transientPotencial [private]

The transient node potencial value in volts.

ElectricCircuit Class Reference

lmaglib::circuit::ElectricCircuitCollaboration diagram for ElectricCircuit:



Detailed Description

The main circuit class. Here circuit components and nodes are stored.

Public Member Functions

- void **setStartTime** (double startTime)
- void **setTimeStep** (double timeStep)
- void **setStep** (int step)
- int **getYSize** ()
- double **getTime** ()
- void **setNumOfSteps** (int numOfSteps)
- void **setEndTime** (double endTime)
- void **solve** (void)
- void **solveTransient** (void)
- void **add** (CouplingElement *)
- void **add** (**Inductor** *)
- void **add** (VoltageSource *)
- void **add** (**CircuitElement** *)
- void **add** (**CircuitNode** *)
- bool **check** (void)
- void **check** (**CircuitElement** *)
- void **check** (**Wire** *)
- void **getAdmitanceMatrix** (CMatrix &)
- void **getAdmitanceMatrix** (RMatrix &)
- void **getCurrentSourcesVector** (CVector &)
- void **getCurrentSourcesVector** (RVector &)
- void **getTransformationMatrix** (CMatrix &)
- void **getTransformationMatrix** (RMatrix &)
- void **getOhmnicResistenceMatrix** (CMatrix &)
- void **getOhmnicResistenceMatrix** (RMatrix &)
- int **numOfNodes** ()
- int **numOfCoupledElements** ()
- void **numberNodes** ()
- size_t **numOfUnknownCurrents** ()

- double **getFrequency** ()

Public Attributes

- vector< **Wire** * > **wires**
- vector< **CircuitElement** * > **unknownCurrents**
- vector< **CircuitElement** * > **elements**
- vector< CouplingElement * > **coupledElements**
- vector< **CircuitNode** * > **nodes**
- double **w**
- double **startTime**
- double **timeStep**
- int **numOfSteps**
- double **endTime**
- double **time**

Member Function Documentation

void add (CircuitNode * *node*)

Add a node to the circuit

Parameters:

node a circuit node pointer.

```
85 {
86         node->setid(nodes.size());
87         nodes.push_back(node);
88 }
```

void add (CircuitElement * *element*)

Add an element to the circuit.

Parameters:

element an circuit element pointer.

```
75 {
76         element->setID(elements.size());
77         elements.push_back(element);
78 }
```

void add (VoltageSource * *element*)

Add the voltage source's unknown current to the circuit.

Parameters:

element an voltage source element pointer.

```
65 {
66         element->setUnknownCurrentID(unknownCurrents.size());
67         unknownCurrents.push_back(element);
68 }
```

void add (Inductor * *element*)

Add the indutor's unknown current to the circuit.

Parameters:

element an inductor element pointer.

```
55 {
56         element->setUnknownCurrentID(unknownCurrents.size());
57         unknownCurrents.push_back(element);
```

```
58 }
```

void add (CouplingElement * *cElement*)

Add an Coupled element to the circuit.

Parameters:

element an circuit element pointer.

```
45 {
46          cElement->setCouplingID(coupledElements.size());
47          coupledElements.push_back(cElement);
48 }
```

void check (Wire * *wire*)

This method search for wires and replace it with short circuit.

```
119 {
120         /* Coloca o mesmo id nos dois nÍ¿¼ (o menor deles) */
121         int node1id = wire->getNode1()->getid();
122         int node2id = wire->getNode2()->getid();
123
124         if (node1id > node2id)
125                 wire->getNode1()->setid(node2id);
126         else
127                 wire->getNode2()->setid(node1id);
128
129         /* Lembrar de remover o fio da lista de elementos */
130
131         return;
132 }
```

void check (CircuitElement * *element*)

This method is used by lmaglib::circuit::ElectricCircuit::checkcircuit(void).

```
111 {
112         return;
113 }
```

bool check (void)

Replace two nodes connected by a wire into one and find if the circuit contain errors:

- Ground existance.
- etc...
- **Returns:**

  true on success or false on error

```
98 {
99          /* Varre o vetor dos elementos */
100         for (unsigned int i = 0; i < this->elements.size(); i++) {
101                 this->check(this->elements[i]);
102         }
103
104         return true;
105 }
```

void getAdmitanceMatrix (CMatrix & *nodaladmitances*)

Calculates the Admitance Matrix Y from a given circuit

```
136 {
137         unsigned int size = numOfNodes()+numOfCoupledElements();
138         nodaladmitances.resize(size, size);
139
140         /* Varre o vetor dos elementos de circuito */
141         for (unsigned int i = 0; i < elements.size(); i++)
```



```
142         {
143                 elements[i]->MNAContribuite(nodaladmitances);
144         }
145 }
```

void getOhmnicResistenceMatrix (RMatrix & *ohmic_resistance_matrix*)

```
221 {
223         unsigned int size = coupledElements.size();
224         ohmic_resistance_matrix.resize(size, size);
225         /* Varre o vetor dos elementos de circuito acoplados */
226         for (unsigned int i = 0; i < size; i++) {
227                 ohmic_resistance_matrix.set(i,i,coupledElements[i]-
>calcOhmicResistence());
228         }
229 }
```

void getOhmnicResistenceMatrix (CMatrix & *ohmic_resistance_matrix*)

```
197 {
199         unsigned int size = coupledElements.size();
200         ohmic_resistance_matrix.resize(size, size);
201         /* Varre o vetor dos elementos de circuito acoplados */
202         for (unsigned int i = 0; i < size; i++) {
203                 ohmic_resistance_matrix.set(i,i,coupledElements[i]-
>calcOhmicResistence());
204         }
205 }
```

void getTransformationMatrix (RMatrix & *transformationmatrix*)

Calculates the Transformation Matrix T from a given circuit

```
209 {
210         transformationmatrix.resize(coupledElements.size(),
numOfNodes()+unknownCurrents.size());
211
212         /* Varre o vetor dos elementos */
213         for (unsigned int i = 0; i < coupledElements.size(); i++)
214         {
215                 transformationmatrix.set(i, coupledElements[i]->getNode1()-
>getid(), -1);
216                 transformationmatrix.set(i, coupledElements[i]->getNode2()-
>getid(), 1);
217         }
218 }
```

void getTransformationMatrix (CMatrix & *transformationmatrix*)

Calculates the Transformation Matrix T from a given circuit

```
185 {
186         transformationmatrix.resize(coupledElements.size(),
numOfNodes()+unknownCurrents.size());
187
188         /* Varre o vetor dos elementos */
189         for (unsigned int i = 0; i < coupledElements.size(); i++)
190         {
191                 transformationmatrix.set(i, coupledElements[i]->getNode1()-
>getid(), -1);
192                 transformationmatrix.set(i, coupledElements[i]->getNode2()-
>getid(), 1);
193         }
194 }
```

void numberNodes ()

```
252 {
253         for (int i = 0; i < nodes.size(); i++)
254         {
255                 nodes[i]->isDead = true;
256         }
```

```
257
258         for (int i = 0; i < elements.size(); i++)
259         {
260                 elements[i]->getNode1()->isDead = false;
261                 elements[i]->getNode2()->isDead = false;
262         }
263
265         //vector <CircuitNode*>::iterator it;
266         //for (it = nodes.begin(); it != nodes.end(); it++)
267         //{
268         //      if ((*it)->isDead) {
269         //              nodes.erase(it);
270         //              //delete (*it);
271         //      }
272         //}
273
274         nodes.clear();
275
276         for (int i = 0; i < elements.size(); i++)
277         {
278                 CircuitNode* node1 = elements[i]->getNode1();
279                 CircuitNode* node2 = elements[i]->getNode2();
280
281                 if (!node1->isDead) {
282                         nodes.push_back(node1);
283                         node1->isDead = true;
284                 }
285
286                 if (!node2->isDead) {
287                         nodes.push_back(node2);
288                         node2->isDead = true;
289                 }
290         }
291
292     int grounds = 0;
293         for (int i = 0; i < nodes.size(); i++)
294         {
295                 if (nodes[i]->getid() >= 0)
296                         nodes[i]->setid(i-grounds);
297                 else
298                         grounds++;
299         }
```

300 }void solve (void)

Solve the eletrical circuit. It means that this method calculates all nodes potencials and the current throw sources and inductors. This function is mtl library dependent.

```
18 {
19         int size = numOfNodes()+unknownCurrents.size();
20         CMatrix nodaladmitances(size, size);
21         CVector currentsources(size);
22         CVector unknownvariables(size);
23
24         for (unsigned int i = 0; i < elements.size(); i++) {
25                 elements[i]->MNAContribuite(nodaladmitances, currentsources);
26         }
27
28         lmaglib::solvers::ICCG < complex<double> > solver;
29     unknownvariables = solver.solveSystem(nodaladmitances, currentsources);
30
31     cout << "Tensoes Nodais" << endl;
32     for (int i = 0; i < size; i++) {
33         cout << unknownvariables[i] << endl;
34     }
35     cout << endl;
36
37 }
```


Member Data Documentation

vector<CouplingElement*> coupledElements

Coupled circuit elements vector.

vector<CircuitElement*> elements

circuit elements vector.

vector<CircuitNode*> nodes

circuit nodes vector.

vector<CircuitElement*> unknownCurrents

Number of circuit unknown currents.

double w

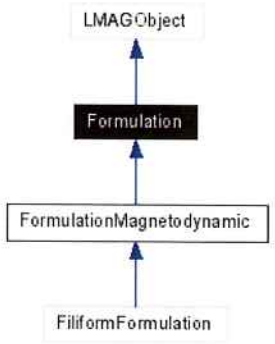
Sinoidal frequency in rad/s

vector<Wire*> wires

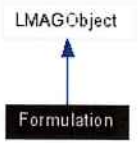
Number of os connections between elements.

Formulation Class Reference

lmaglib::formulations::FormulationInheritance diagram for Formulation:



Collaboration diagram for Formulation:



Detailed Description

This is the base virtual class that implements the Mathematical formulations for the problems we solve. The concrete classes that implement the formulations implement one function that gets the ShapeFunctions, the Shape Functions Derivative, the element physical characteristics (material, sources, etc ...) and put all these things together as a value. For instance, for a static formulation, what the formulation does is to evaluate: k grad(Ni) grad(Nj) and return this to the Integration method that is evaluating the local matrix of the element.

To extend the program and have new kinds of Mathematical formulations, we can add a class to the **Formulation** hierarchy and overwrite the functions: computeMatrixFormulation computeVectorFormulation

Also, the **Formulation** Factory "createFormulation" method must be modified so that this new kind of **Formulation** can be instantiated

Public Member Functions

- virtual **~Formulation** ()
  Virtual destructor.

- virtual void **computeMatrixFormulation** (Matrix< double > &formMatrix, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- virtual void **computeVectorFormulation** (Vector< double > &formVector, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- virtual void **computeMatrixFormulation** (Matrix< complex< double > > &formMatrix, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- virtual void **computeVectorFormulation** (Vector< complex< double > > &formVector, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- virtual **Formulation** * **clone** () const =0
- virtual void **computeResidualMatrixFormulation** (Matrix< double > &formMatrix, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- virtual void **computeResidualVectorFormulation** (Vector< double > &formVector, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- virtual void **computeCouplingVectorFormulation** (Vector< complex< double > > &formVector, const Element &elem, const Coordinates &coord, double domainDepth=0) const

Member Function Documentation

virtual Formulation* clone () const `[pure virtual]`

Creates a new **Formulation** with the same type and contents of the actual **Formulation**. This is a virtual function that should be implemented in the derived classes.

Implemented in FormulationMagnetodynamic ***(p.31)*.virtual void computeMatrixFormulation (Matrix< complex< double > > & *formMatrix*, const Element & *elem*, const Coordinates & *coord*, Coordinates * *zal* = 0) const** `[inline, virtual]`

Computes the formulation Matrix. This method is called by Element::compute LocalVector, so that, for each formulation, the AlgebraicExpression that is being computed can change, depending on the formulation.

For instance, for magnetostatics 2D formulation, the matrix is equal to:

k grad(Ni) grad (Nj) the object coord is the Gauss coordinates for integration. This coordinates is used only in boundary element method calculations. In all others cases "zal" is zero and it is not used.

```
Reimplemented in FormulationMagnetodynamic (p.32).124                {
125                 }
```

virtual void computeMatrixFormulation (Matrix< double > & *formMatrix*, const Element & *elem*, const Coordinates & *coord*, Coordinates * *zal* = 0) const `[inline, virtual]`

Computes the formulation Matrix. This method is called by Element::compute LocalVector, so that, for each formulation, the AlgebraicExpression that is being computed can change, depending on the formulation.

For instance, for magnetostatics 2D formulation, the matrix is equal to:

k grad(Ni) grad (Nj) the object coord is the Gauss coordinates for integration. This coordinates is used only in boundary element method calculations. In all others cases "zal" is zero and it is not used.

```
Reimplemented in FormulationMagnetodynamic (p.31).96                {
97                }
```

virtual void computeVectorFormulation (Vector< complex< double > > & *formVector*, const Element & *elem*, const Coordinates & *coord*, Coordinates * *zal* = 0) const [inline, virtual]

Computes the formulation contribution for the right side vector. This method is called by Element::computeLocalVector, so that, for each formulation, the AlgebraicExpression that is being integrated can change. the object coord is the Gauss coordinates for integration. This coordinates is used only in boundary element method calculations. In all others cases "zal" is zero and it is not used.

```
Reimplemented in FormulationMagnetodynamic (p.32).136               {
137               }
```

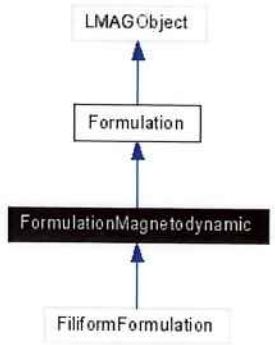
virtual void computeVectorFormulation (Vector< double > & *formVector*, const Element & *elem*, const Coordinates & *coord*, Coordinates * *zal* = 0) const [inline, virtual]

Computes the formulation contribution for the right side vector. This method is called by Element::computeLocalVector, so that, for each formulation, the AlgebraicExpression that is being integrated can change. the object coord is the Gauss coordinates for integration. This coordinates is used only in boundary element method calculations. In all others cases "zal" is zero and it is not used.

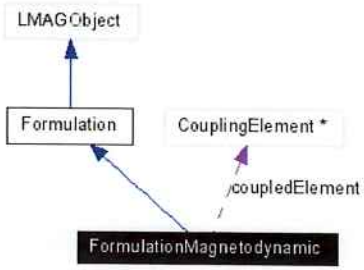
```
Reimplemented in FormulationMagnetodynamic (p.32).108               {
109               }
```


FormulationMagnetodynamic Class Reference

lmaglib::formulations::FormulationMagnetodynamicInheritance diagram for FormulationMagnetodynamic:



Collaboration diagram for FormulationMagnetodynamic:



Detailed Description

Author:
Fernando Jaruche Nunes

Version:
1.0

Date:
01/04/2004

Public Member Functions

- **FormulationMagnetodynamic * clone** () const
- void **computeMatrixFormulation** (Matrix< complex< double > > &formMatrix, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- void **computeVectorFormulation** (Vector< complex< double > > &formVector, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- void **computeResidualMatrixFormulation** (Matrix< complex< double > > &formMatrix, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- void **computeResidualVectorFormulation** (Vector< complex< double > > &formVector, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- void **computeCouplingMatrixFormulation** (Vector< complex< double > > &formVector, const Element &elem, const Coordinates &coord, double domainDepth=1) const
  Implementado nas classes abaixo.

- void **computeMatrixFormulation** (Matrix< double > &formMatrix, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
  Caso transitório:.

- void **computeVectorFormulation** (Vector< double > &formVector, const Element &elem, const Coordinates &coord, Coordinates *zal=0) const
- void **computeCouplingMatrixFormulation** (Vector< double > &formVector, const Element &elem, const Coordinates &coord, double domainDepth=1) const
  Implementado nas classes abaixo.

Public Attributes

- Imaglib::circuit::CouplingElement * **coupledElement**

Member Function Documentation

FormulationMagnetodynamic* clone () const `[virtual]`

Creates a new **Formulation** with the same type and contents of the actual **Formulation**. This is a virtual function that should be implemented in the derived classes.

Implements Formulation **(*p.28*).void computeMatrixFormulation (Matrix< double > & *formMatrix*, const Element & *elem*, const Coordinates & *coord*, Coordinates * *zal* = 0) const** `[virtual]`

Caso transitório:.

Computes the formulation Matrix. This method is called by Element::compute LocalVector, so that, for each formulation, the AlgebraicExpression that is being computed can change, depending on the formulation.

For instance, for magnetostatics 2D formulation, the matrix is equal to:

k grad(Ni) grad (Nj) the object coord is the Gauss coordinates for integration. This coordinates is used only in boundary element method calculations. In all others cases "zal" is zero and it is not used.

Reimplemented from Formulation **(*p.28*).void computeMatrixFormulation (Matrix< complex< double > > & *formMatrix*, const Element & *elem*, const Coordinates & *coord*, Coordinates * *zal* = 0) const** `[virtual]`

Computes the formulation Matrix. This method is called by Element::compute LocalVector, so that, for each formulation, the AlgebraicExpression that is being computed can change, depending on the formulation.

For instance, for magnetostatics 2D formulation, the matrix is equal to:

k grad(Ni) grad (Nj) the object coord is the Gauss coordinates for integration. This coordinates is used only in boundary element method calculations. In all others cases "zal" is zero and it is not used.

Reimplemented from Formulation **(*p.28*).void computeVectorFormulation (Vector< double > & *formVector*, const Element & *elem*, const Coordinates & *coord*, Coordinates * *zal* = 0) const** `[virtual]`

Computes the formulation contribution for the right side vector. This method is called by Element::computeLocalVector, so that, for each formulation, the AlgebraicExpression that is being integrated can change. the object coord is the Gauss coordinates for integration. This coordinates is used only in boundary element method calculations. In all others cases "zal" is zero and it is not used.

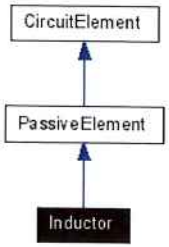
Reimplemented from Formulation **(*p.29*).void computeVectorFormulation (Vector< complex< double > > & *formVector*, const Element & *elem*, const Coordinates & *coord*, Coordinates * *zal* = 0) const** `[virtual]`

Computes the formulation contribution for the right side vector. This method is called by Element::computeLocalVector, so that, for each formulation, the AlgebraicExpression that is being integrated can change. the object coord is the Gauss coordinates for integration. This coordinates is used only in boundary element method calculations. In all others cases "zal" is zero and it is not used.

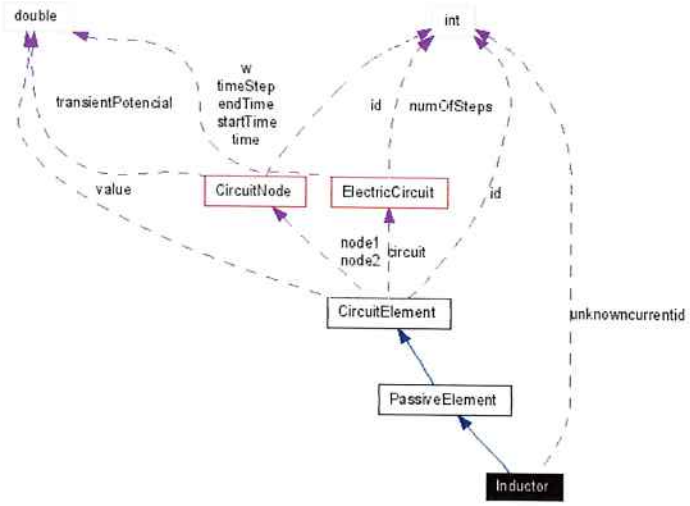
Reimplemented from Formulation **(*p.29*).**

Inductor Class Reference

lmaglib::circuit::InductorInheritance diagram for Inductor:



Collaboration diagram for Inductor:



Detailed Description

Inductor description.

Public Member Functions

- **Inductor** (**ElectricCircuit** *circuit, int **value**=1, **CircuitNode** ***node1**=NULL, **CircuitNode** ***node2**=NULL)
- void **setUnknownCurrentID** (int **id**)
- complex< double > **getimpedance** (double w)
- complex< double > **getadmitance** (double w)
- double **getimpedance** (double i, double iLast, double timeStep)
- double **getadmitance** (double i, double iLast, double timeStep)
- void **MNAContribuite** (CMatrix &)

Private Attributes

- int **unknowncurrentid**

Constructor & Destructor Documentation

Inductor (ElectricCircuit * *circuit*, int *value* = 1, CircuitNode * *node1* = NULL, CircuitNode * *node2* = NULL)

See also:
CircuitElement::CircuitElement(CircuitNode *, CircuitNode *)

```
8         : PassiveElement(circuit, value, node1, node2)
9 {
10         this->circuit->add(this);
11 }
```

Member Function Documentation

double getadmitance (double *i*, double *iLast*, double *timeStep*) [virtual]

See also:
PassiveElement::getadmitance(double i, double iLast, double timeStep)

```
Implements PassiveElement (p.37).46 {
47          return 1.0/this->getimpedance(i, iLast, timeStep);
48 }
```

complex< double > getadmitance (double *w*) [virtual]

See also:
PassiveElement::getadmitance(double)

```
Implements PassiveElement (p.37).30 {
31          return 1.0/this->getimpedance(w);
32 }
```

double getimpedance (double *i*, double *iLast*, double *timeStep*) [virtual]

See also:

PassiveElement::getimpedance(double i, double iLast, double timeStep)

```
Implements PassiveElement (p.38).38 {
39          return getValue()*(1-i/iLast)/timeStep;
40  }
```

complex< double > getimpedance (double *w*) [virtual]

See also:

PassiveElement::getimpedance(double)

```
Implements PassiveElement (p.38).21 {
22          complex<double> result (0, w*this->getValue());
23          return result;
24  }
```

void MNAContribuite (CMatrix & *nodaladmitancematrix*) [virtual]

Despite been a passive element inductor have some peculiar contributions to the modified nodal Analisys method. These peculiar contributions are made here after a call to PassiveElement::MNAContribuite(MTLMatrix &, MTLVector&)

See also:

PassiveElement::MNAContribuite(MTLMatrix &, MTLVector&)

```
Reimplemented from PassiveElement (p.38).67 {
68          this->PassiveElement::MNAContribuite(nodaladmitancematrix);
69
70          /* Faz agora as particularidades do indutor. */
71          int position = this->circuit->numOfNodes() + this->unknowncurrentid;
72          nodaladmitancematrix.add(position, position, getimpedance(circuit-
>getFrequency()));
73  }
```

void setUnknownCurrentID (int *id*)

Set the unknown current identification number.

Parameters:

id The unknown current identification number.

```
55  {
56          this->unknowncurrentid = id;
57  }
```

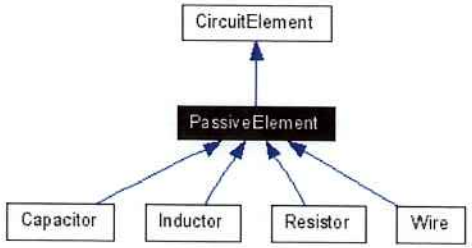
Member Data Documentation

int unknowncurrentid [private]

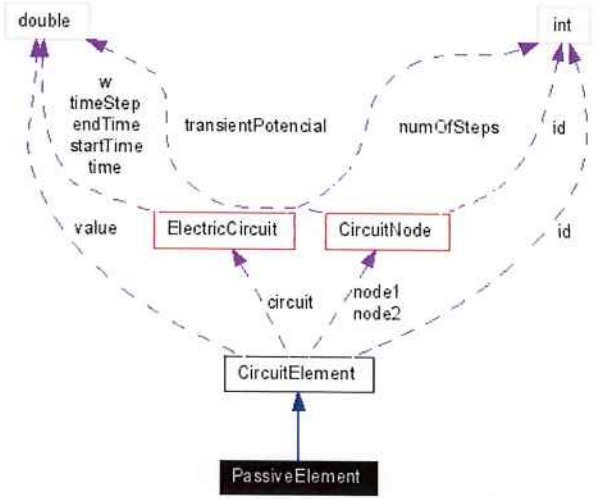
Identification number of unknown current that flows throw inductor.

PassiveElement Class Reference

lmaglib::circuit::PassiveElementInheritance diagram for PassiveElement:



Collaboration diagram for PassiveElement:



Detailed Description

Passive element description and base class.

Public Member Functions

- **PassiveElement** (**ElectricCircuit** *circuit, int **value**=1, **CircuitNode** ***node1**=NULL, **CircuitNode** ***node2**=NULL)
- virtual complex< double > **getimpedance** (double w)=0
- virtual complex< double > **getadmitance** (double w)=0
- complex< double > **getimpedance** (void)
- complex< double > **getadmitance** (void)
- virtual double **getimpedance** (double i, double iLast, double timeStep)=0
- virtual double **getadmitance** (double i, double iLast, double timeStep)=0
- void **MNAContribuite** (CMatrix &)
- void **MNAContribuite** (CVector &)
- void **MNAContribuiteTransient** (RMatrix &)
- void **MNAContribuiteTransient** (RVector &)

Constructor & Destructor Documentation

PassiveElement (ElectricCircuit * *circuit*, int *value* = 1, CircuitNode * *node1* = NULL, CircuitNode * *node2* = NULL)

This constructor calls CircuitElement::CircuitElement(CircuitNode *, CircuitNode *) as **CircuitElement** principal constructor.

See also:

CircuitElement::CircuitElement(CircuitNode *, CircuitNode *)

```
9         : CircuitElement(circuit, value, node1, node2)
10 {
11 }
```

Member Function Documentation

virtual double getadmitance (double *i*, double *iLast*, double *timeStep*) [pure virtual]

Calculate the passive element transient admitance. Admitance depends on passive element type so it is not implemented here.

Returns:

The passive element transient admitance.

Implemented in Capacitor (***p.12***), Inductor (***p.34***), Resistor (***p.50***), **and** Wire (***p.54***).**complex< double > getadmitance (void)**

A temporary work around. I will remove it soon.

```
29 {
30         return this->getadmitance(1);
31 }
```

virtual complex<double> getadmitance (double *w*) [pure virtual]

Calculate the passive element admitance. Admitance depends on passive element type so it is not implemented here.

Returns:

The passive element admitance.

Implemented in Capacitor (***p.12***), Inductor (***p.34***), Resistor (***p.50***), **and** Wire (***p.54***).**virtual double getimpedance (double *i*, double *iLast*, double *timeStep*)** [pure virtual]

Calculate the passive element transient impedance. Impedance depends on passive element type so it is not implemented here.

Returns:

The passive element transient impedance.

Implemented in Capacitor (***p.12***), Inductor (***p.34***), Resistor (***p.50***), **and** Wire (***p.54***).**complex< double > getimpedance (void)**

A temporary work around. I will remove it soon.

```
21 {
22         return this->getimpedance(1);
23 }
```

virtual complex<double> getimpedance (double *w*) [pure virtual]

Calculate the passive element impedance. Impedance depends on passive element type so it is not implemented here.

Returns:

The passive element impedance.

Implemented in Capacitor (***p.13***), Inductor (***p.35***), Resistor (***p.51***), **and** Wire (***p.55***).**void MNAContribuite (CVector & *currentsourcesvector*)** [virtual]

Calculate the passive element contributions to the modified nodal Analisys method. It update nodal admitance matriz and current sources vector in correct places.

Parameters:

nodaladmitancematrix Nodal admitance matrix reference.
currentsourcesvector Current sources vector reference.

See also:

CircuitElement::MNAContribuite()

```
Implements CircuitElement (p.16).65 {
66         return;
67 }
```

void MNAContribuite (CMatrix & *nodaladmitancematrix*) [virtual]

Calculate the passive element contributions to the modified nodal Analisys method. It update nodal admitance matriz and current sources vector in correct places.

Parameters:

nodaladmitancematrix Nodal admitance matrix reference.
currentsourcesvector Current sources vector reference.

See also:

CircuitElement::MNAContribuite()

```
Implements CircuitElement (p.16).Reimplemented in Inductor (p.35).42 {
43         int node1 = this->getNode1()->getid();
44         int node2 = this->getNode2()->getid();
45
46         /* A contribuição do elemento ligado ao terra deverá ser suprimida */
47         if (node1 >= 0)
48                 nodaladmitancematrix.add(node1,node1,getadmitance());
```

```
49          if (node2 >= 0)
50                  nodaladmitancematrix.add(node2,node2,getadmitance());
51          if (node1 >= 0 && node2 >= 0)
52                  nodaladmitancematrix.add(node1,node2, -getadmitance());
53          if (node1 >= 0 && node2 >= 0)
54                  nodaladmitancematrix.add(node2,node1, -getadmitance());
55  }
```

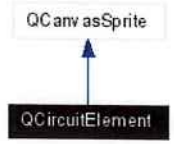
void MNAContribuiteTransient (RMatrix &) [inline, virtual]

Calculate the circuit element contributions to the modified nodal Analisys method in transient simulation.

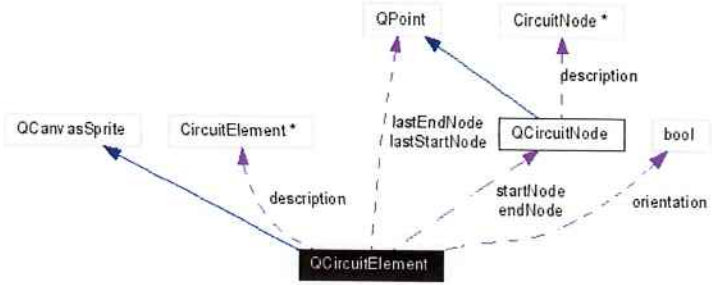
Implements **CircuitElement** (*p.16*).Reimplemented in **Resistor** (*p.51*).81 {}

QCircuitElement Class Reference

QCircuitElementInheritance diagram for QCircuitElement:



Collaboration diagram for QCircuitElement:



Detailed Description

Author:
jaruche

Public Member Functions

- **QCircuitElement** (QCanvasPixmapArray *a, QCanvas *canvas, CircuitElement *physicalDescription)
- void **moveBy** (double dx, double dy)
- **QCircuitNode * getNearestNode** (QPoint p)
- void **setNearestNode** (QPoint p, CircuitNode *descr)
- bool **updateNodes** ()

Public Attributes

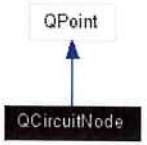
- CircuitElement * **description**

Private Attributes

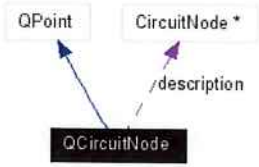
- **QCircuitNode startNode**
- QPoint **lastStartNode**
- **QCircuitNode endNode**
- QPoint **lastEndNode**
- bool **orientation**
  True Vertical, False horizontal.

QCircuitNode Class Reference

QCircuitNodeInheritance diagram for QCircuitNode:



Collaboration diagram for QCircuitNode:



Detailed Description

Author:
jaruche

Public Member Functions

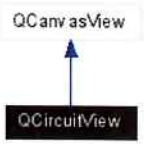
- **QCircuitNode** (int xpos, int ypos, CircuitNode *descr)

Public Attributes

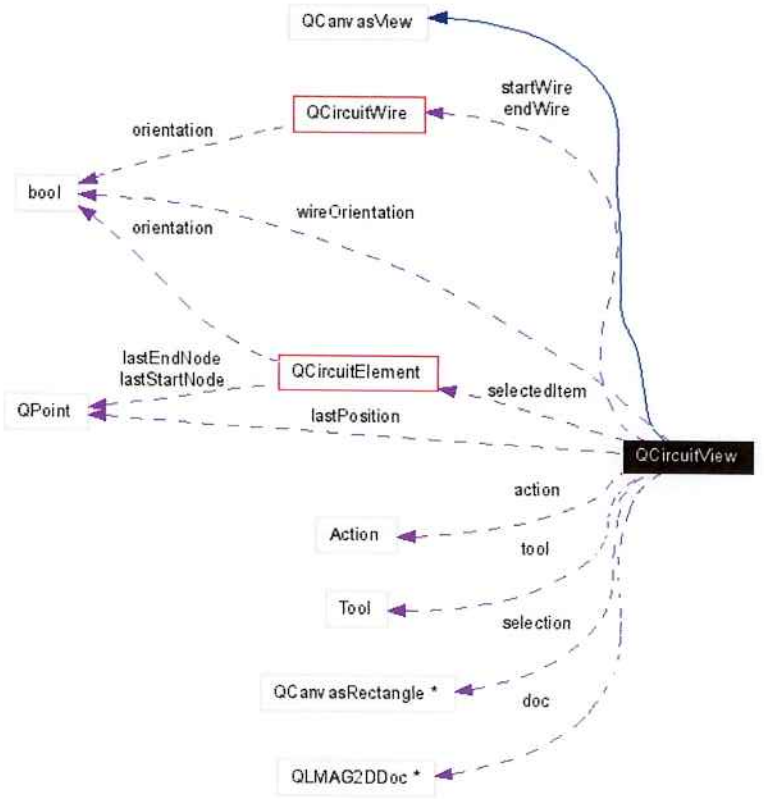
- CircuitNode * **description**

QCircuitView Class Reference

QCircuitViewInheritance diagram for QCircuitView:



Collaboration diagram for QCircuitView:



Detailed Description

Author:
jaruche

Public Slots

- void **viewportResizeEvent** (QResizeEvent *e)

- void **contentsMousePressEvent** (QMouseEvent *e)
- void **contentsMouseReleaseEvent** (QMouseEvent *e)
- void **contentsMouseMoveEvent** (QMouseEvent *e)

Public Member Functions

- **QCircuitView** (QLMAG2DDoc *pDoc, QWidget *parent=0, const char *name=0, WFlags f=0)
- void **updateNodes** ()

Public Attributes

- QPoint **lastPosition**
- **QCircuitElement * selectedItem**
- **QCircuitWire * startWire**
- **QCircuitWire * endWire**
- QCanvasRectangle * **selection**
- Action **action**
- Tool **tool**
- bool **wireOrientation**

Protected Member Functions

- virtual void **focusInEvent** (QFocusEvent *event)

Private Attributes

- QLMAG2DDoc * **doc**
  The one and only document of this view.

Member Function Documentation

void contentsMouseMoveEvent (QMouseEvent * e) [slot]

```
267 {
268     QPoint p = e->pos();
269     QPoint endNode;
270     QPoint delta = p-lastPosition;
271
272     switch (action) {
273               case DRAWING_WIRE:
274                       if ( startWire->startPoint() == startWire->endPoint() &&
275                               endWire->startPoint() != endWire->endPoint() ) {
276                               wireOrientation = !wireOrientation;
277                               startWire->setOrientation(!wireOrientation);
278                               endWire->setOrientation(wireOrientation);
279                       }
280
281                       if (wireOrientation) {
282                               endNode.setX(startWire->getStartNode()->x());
283                               endNode.setY(p.y());
284                       }
285                       else {
286                               endNode.setX(p.x());
287                               endNode.setY(startWire->getStartNode()->y());
288                       }
289                       startWire->setEndNode(endNode);
290
291                       // startWire->getEndNode() == endWire->getStartNode()
```



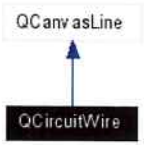
```
292                       // Por isso nao precisa fazer de novo
293
294                       endWire->setEndNode(p);
295
296               break;
297
298               case SELECTING:
299                       selection->setSize(delta.x(), delta.y());
300                       break;
301
302               case TRACKING_ELEMENT:
303                       selectedItem->moveBy(p.x() - lastPosition.x(),p.y() -
lastPosition.y());
304                       updateNodes();
305
307                       lastPosition = p;
308                       break;
309
310               default: // soh está mechendo o mouse, não precisa dar update.
311                       return;
312                       break;
313     }
314
315     canvas()->update();
316 }
```

void contentsMouseReleaseEvent (QMouseEvent * e) [slot]

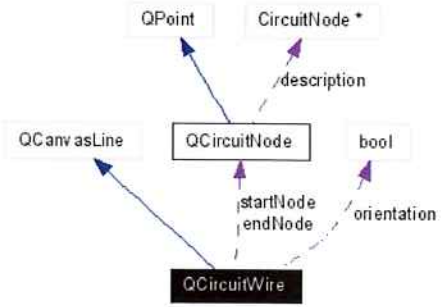
```
238 {
239         /* O Correto é mudar a ferramenta e acao */
240         /* Solta o item que está sendo arrastado. */
241     switch (e->button()) {
242               case Qt::LeftButton:
243                       selectedItem = false;
244                       switch (action) {
245                               case DRAWING_WIRE:
246                                       break;
247
248                               case SELECTING:
249                                       delete selection;
250                                       action = NOTHING;
252                                       break;
253
254                               default:
255                                       action = NOTHING;
256                                       break;
257                       }
258                       break;
259
260               default: break;
261         }
262
263     canvas()->update();
264 }
```

QCircuitWire Class Reference

QCircuitWireInheritance diagram for QCircuitWire:



Collaboration diagram for QCircuitWire:



Detailed Description

Author:
jaruche

Public Member Functions

- **QCircuitWire** (QCanvas *canvas, bool **orientation**)
- **QCircuitNode * getStartNode** ()
- void **setStartNode** (**QCircuitNode** *n)
- void **setStartNode** (QPoint &p)
- **QCircuitNode * getEndNode** ()
- void **setEndNode** (**QCircuitNode** *n)
- void **setEndNode** (QPoint &p)
- void **setNodes** (**QCircuitNode** *sn, **QCircuitNode** *en)

- void **setOrientation** (bool o)
- bool **getOrientation** ()
- void **moveBy** (double dx, double dy)
- bool **updateNodes** ()

Protected Member Functions

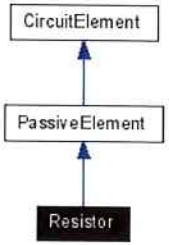
- void **draw** (QPainter &p)

Private Attributes

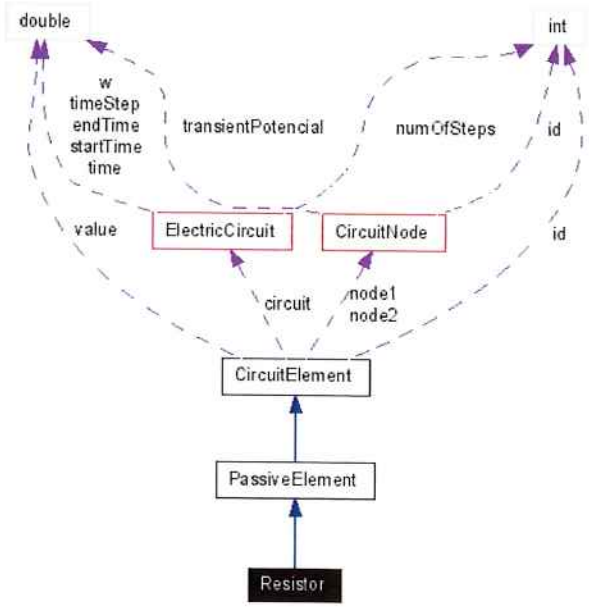
- **QCircuitNode * startNode**
- **QCircuitNode * endNode**
- bool **orientation**
  True Vertical, False horizontal.

Resistor Class Reference

lmaglib::circuit::ResistorInheritance diagram for Resistor:



Collaboration diagram for Resistor:



Detailed Description

Resistor description.

Public Member Functions

- **Resistor** (**ElectricCircuit** *circuit, int **value**=1, **CircuitNode** ***node1**=NULL, **CircuitNode** ***node2**=NULL)
- complex< double > **getimpedance** (double w)
- complex< double > **getadmitance** (double w)
- double **getimpedance** (double i, double iLast, double timeStep)
- double **getadmitance** (double i, double iLast, double timeStep)
- void **MNAContribuiteTransient** (RMatrix &)
- void **MNAContribuiteTransient** (RVector &)

Constructor & Destructor Documentation

Resistor (ElectricCircuit * *circuit*, int *value* = 1, CircuitNode * *node1* = NULL, CircuitNode * *node2* = NULL)

See also:
CircuitElement::CircuitElement(CircuitNode *, CircuitNode *)

```
7           : PassiveElement(circuit, value, node1, node2)
8 {
9 }
```

Member Function Documentation

double getadmitance (double *i*, double *iLast*, double *timeStep*) [virtual]

See also:
PassiveElement::getadmitance(double i, double iLast, double timeStep)

```
Implements PassiveElement (p.37).43 {
44          return 1.0/this->getimpedance(i, iLast, timeStep);
45 }
```

complex< double > getadmitance (double *w*) [virtual]

See also:
PassiveElement::getadmitance(double)

```
Implements PassiveElement (p.37).27 {
28          return 1/this->getValue();
29 }
```

double getimpedance (double *i*, double *iLast*, double *timeStep*) [virtual]

See also:

PassiveElement::getimpedance(double i, double iLast, double timeStep)

```
Implements PassiveElement (p.38).35 {
36          return getValue();
37 }
```

complex< double > getimpedance (double *w*) [virtual]

See also:

PassiveElement::getimpedance(double)

```
Implements PassiveElement (p.38).19 {
20          return this->getValue();
21 }
```

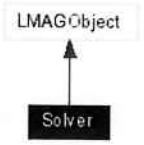
void MNAContribuiteTransient (RMatrix &) [virtual]

Calculate the circuit element contributions to the modified nodal Analisys method in transient simulation.

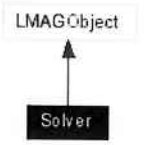
```
Reimplemented from PassiveElement (p.39).48 {
49          int node1 = getNode1()->getid();
50          int node2 = getNode2()->getid();
51
52          if (node1 >= 0)
53                  nodaladmitancematrix.add(node1,node1,getValue());
54          if (node2 >= 0)
55                  nodaladmitancematrix.add(node2,node2,getValue());
56          if (node1 >= 0 && node2 >= 0)
57                  nodaladmitancematrix.add(node1,node2,-getValue());
58          if (node1 >= 0 && node2 >= 0)
59                  nodaladmitancematrix.add(node2,node1,-getValue());
60 }
```


Solver Class Template Reference

lmaglib::solvers::SolverInheritance diagram for Solver:



Collaboration diagram for Solver:



Detailed Description

template<class T> class lmaglib::solvers::Solver< T >

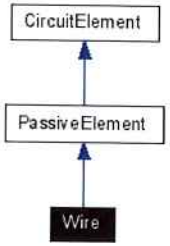
This class solves the matrix equation system that was mounted by the Assembler. In this first version of the system, this is a concrete class. However, this class can define a hierarchy that can be easily extended to include new kinds of solvers.

Public Member Functions

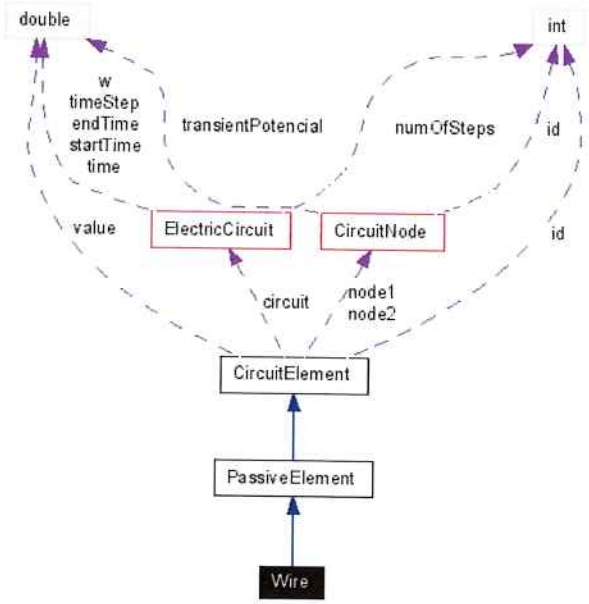
- virtual Vector< T > **solveSystem** (SparseMatrix< T > &A, Vector< T > &b)=0
  Solves the system given by equation A.x = b, and returns x.

Wire Class Reference

Imaglib::circuit::WireInheritance diagram for Wire:



Collaboration diagram for Wire:



Detailed Description

Wire description. **Wire** is an element that have the same potencial between its nodes.

Public Member Functions

- **Wire** (**ElectricCircuit** *circuit, int **value**=1, **CircuitNode** ***node1**=NULL, **CircuitNode** ***node2**=NULL)
- complex< double > **getimpedance** (double w)
- complex< double > **getadmitance** (double w)
- double **getimpedance** (double i, double iLast, double timeStep)
- double **getadmitance** (double i, double iLast, double timeStep)

Constructor & Destructor Documentation

Wire (ElectricCircuit * *circuit*, int *value* = 1, CircuitNode * *node1* = NULL, CircuitNode * *node2* = NULL)

See also:

CircuitElement::CircuitElement(CircuitNode *, CircuitNode *)

```
7           : PassiveElement(circuit, value, node1, node2)
8 {
9 }
```

Member Function Documentation

double getadmitance (double *i*, double *iLast*, double *timeStep*) [virtual]

Be carefull. This function returns always 1/0 that is not a number.

See also:

PassiveElement::getadmitance(double i, double iLast, double timeStep)

```
Implements PassiveElement (p.37).47 {
48          return 1.0/this->getimpedance(i, iLast, timeStep);
49 }
```

complex< double > getadmitance (double *w*) [virtual]

Be carefull. This function returns always 1/0 that is not a number.

See also:

PassiveElement::getadmitance(double)

```
Implements PassiveElement (p.37).29 {
30          return 1.0/this->getimpedance(w);
31 }
```

double getimpedance (double *i*, double *iLast*, double *timeStep*) [virtual]

Be carefull. This function returns always zero.

See also:

PassiveElement::getimpedance(double i, double iLast, double timeStep)

```
Implements PassiveElement (p.38).38 {
39          return 0.0;
40 }
```

complex< double > getimpedance (double *w*) [virtual]

Be carefull. This function returns always zero.

See also:

PassiveElement::getimpedance(double)

```
Implements PassiveElement (p.38).20 {
21          return 0.0;
22 }
```


Index