UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MODELAGEM E IMPLEMENTAÇÃO DE APLICAÇÕES USANDO UMA BASE COMPUTACIONAL ORIENTADA A OBJETOS PARA SISTEMAS DE ENERGIA ELÉTRICA

Dissertação submetida à

Universidade Federal de Santa Catarina

como parte dos requisitos para a

obtenção do grau de Mestre em Engenharia Elétrica

DANIEL DOTTA

Florianópolis, Agosto de 2003

MODELAGEM E IMPLEMENTAÇÃO DE APLICAÇÕES USANDO UMA BASE COMPUTACIONAL ORIENTADA A OBJETOS PARA SISTEMAS DE ENERGIA ELÉTRICA

DANIEL DOTTA

'Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em Sistemas de Energia Elétrica, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

_____

Prof. Ildemar Cassana Decker, D. Sc.
Orientador

_____

Prof. Edson Roberto de Pieri, D. Sc.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

_____

Prof. Ildemar Cassana Decker, D. Sc.
Presidente

_____

Prof. Aguinaldo Silveira e Silva, Ph. D.

_____

Prof. Arlan Luiz Bettiol, D.S.A.

_____

Eng. Marcelo Neujahr Agostini, D. Eng.

Dedicada a minha família que proporcionou todas as condições para que eu chegasse até aqui, minha mãe ***Juçara****, meus irmãos* ***Geovana*** *e* ***Gabriel*** *e em especial ao meu pai* ***José Dotta*** *pelo seu exemplo de vida.*

Patrão velho, muito obrigado, por este céu azul,
por esta terra tão linda, pelos Campos de Lages...
Muito obrigado, pelas andanças do pago;
pela chinoca faceira e o gosto do mate amargo.
Patrão velho,
muito obrigado pelos fandangos de galpão;
pelos domingos de rodeio,
nos campos do meu rincão;
pela geada caindo tornando em branco o capim;
por esta chama rebelde
que queima dentro de mim.
Muito obrigado por estas almas andarilhas
que como o vento minuano vagueiam pelas coxilhas.
(Nardel Silva e Os Oliveiras)

AGRADECIMENTOS

Ao Professor Ildemar Cassana Decker pelo incentivo, amizade, dedicação e pela confiança que depositou em mim ao longo destes vários anos de LabPlan.

Agradecimento especial aos colegas e amigos Adriano de Souza, João Marco Francischetti Ferreira e Marcelo Neujahr Agostini, pelas suas inúmeras contribuições dadas a minha formação e as sempre agradáveis discussões sobre os mais diversos temas nas tardes de sexta-feira.

A todos os integrantes e ex-integrates do LabPlan que de alguma contribuíram para a minha formação tanto profissional como pessoal. Uma lembrança especial dos colegas de mestrado: Flor, Fon, Rafael, Sica e Zucarato.

A Nadia Orso pelo apoio, paciência e incentivo demostrada no decorrer deste trabalho.

Aos Membros da banca examinadora pelas sugestões que contribuíram para melhorar a qualidade do trabalho final.

Ao CNPq pelo apoio financeiro.

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica

MODELAGEM E IMPLEMENTAÇÃO DE APLICAÇÕES USANDO UMA BASE COMPUTACIONAL ORIENTADA A OBJETOS PARA SISTEMAS DE ENERGIA ELÉTRICA

DANIEL DOTTA

Agosto/2003

Orientador: Ildemar Cassana Decker, D. Sc.

Área de Concentração: Sistemas de Energia Elétrica.

Palavras-chave: Modelagem de Sistemas de Energia Elétrica, Modelagem Orientada a Objetos, Segurança Dinâmica.

Número de Páginas: 113.

A presente dissertação de mestrado descreve os resultados de uma investigação sobre a modelagem e a implementação de aplicações computacionais sob o paradigma de uma nova filosofia de desenvolvimento de software para sistemas de energia elétrica (SEE). Neste contexto utilizam-se abstrações bem definidas, uma base computacional capaz de representar as mais diversas instâncias do SEE, a *Unified Modeling Language* (UML) para a documentação e padrões de projeto orientados a objeto. No âmbito das abstrações há uma nítida separação entre a representação de elementos físicos, aplicações de análise e módulos de funções específicas. Neste trabalho, os instrumentos descritos foram utilizados na modelagem e implementação de três metodologias de análise da operação de SEE: (i) cálculo de fluxo de potência, modelos Newton-Raphson e Desacoplado-Rápido; (ii) avaliação da segurança dinâmica usando modelagem detalhada; (iii) seleção e classificação de contingências críticas usando modelagem simplificada e ambiente de processamento paralelo. O projeto e a implementação das aplicações foram realizados com os recursos da base computacional e com o aproveitamento de códigos já desenvolvidos e testados.

O escopo dos resultados deste trabalho compreende o processo de desenvolvimento de software propriamente dito e o desempenho computacional específico das aplicações implementadas. No primeiro caso destacam-se a verificação das facilidades propiciadas pela base computacional à incorporação de novas metodologias de análise de SEE bem como a geração de aplicações computacionais de fácil manutenção e incorporação de novos modelos e equipamentos. Na avaliação do desempenho computacional foram realizadas simulações com três configurações do sistema elétrico das regiões sul e sudeste do Brasil, obtendo-se resultados comparáveis ao de programas tradicionalmente empregados no setor elétrico brasileiro.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

MODELING AND IMPLEMENTATION OF APPLICATIONS USING A OBJECT-ORIENTED COMPUTATIONAL BASE FOR POWER SYSTEMS

Daniel Dotta

August/2003

Advisor: Ildemar Cassana Decker, D. Sc.

Area of Concentration: Electric Energy Systems.

Keywords: Power Systems Modeling, Object Oriented Modeling, Dynamic Security.

Number of Pages: 113.

The present master dissertation describes the results of an investigation about the modeling and implementation of computational applications based on a new approach for the design of electric power system software. It utilizes well-defined abstractions, a computational base capable to represent a large spectrum of power systems, the Unified Modeling Language (UML) for documentation and object-oriented design patterns. With in the scope of abstractions there is a clear separation between the physical elements representation, analysis applications and modules with specific functions. In this work, those facilities were used to model and implement three methodologies for analysis and operation of power systems: (i) Newton-Raphson and Fast Decoupled load flow methods; (ii) dynamic security assessment using detailed modeling; (iii) selection and classification of critical contingencies using parallel computing. The project and implementation of these applications were realized based on the computational base and with the use of codes already developed and tested. The results of this work comprehend the process of development of software and the computational performance of the implemented applications. The development results is assessed by the facilities provided by the computational base to support new system analysis applications as well as new models and devices. The computational performance was evaluated by simulations with three configurations of the Southern and Southeastern Brazilian electrical systems. The obtained results are compared with the standard programs used in Brazilian electricity industry.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

ASDIN : Avaliação da Segurança Dinâmica;

BCOO : Base Computacional Orientada a Objetos;

CIM : *Commom Information Model;*

EMS : *Energy Management System*;

EPG : Energia Potencial Generalizada;

IEC : *International Electrotechnical Commission*;

MOO : Modelagem Orientada a Objetos;

MPP : *Massive Parallel Processing*;

MPI : *Message Passing Interface*;

NOW : *Network of Workstations*;

PAD : Processamento de Alto Desempenho;

POP : *Pile of PC's*;

PVM : *Parallel Virtual Machine*;

SEE : Sistemas de Energia Elétrica;

SIMSP : Simulador de Sistemas de Potência;

SIME : *Single Machine Equivalent*;

SLEP : Superfície Limite de Energia Potencial;

SIN : Sistema Interligado Nacional;

UML : Unified Modeling Language.

CAPÍTULO 1

INTRODUÇÃO

1.1 Introdução

Na última década, observou-se em muitos países um maior interesse por parte dos governos, grandes industriais e investidores pela reestruturação da indústria de energia elétrica no sentido do estabelecimento de ambientes de competição. Especificamente, nos países em desenvolvimento, as empresas públicas do setor elétrico têm sido privatizadas como pré-condição para que empréstimos sejam liberados por fundos internacionais, ou para atrair investimentos para o setor (DY-LIACCO, 2002). A operação de sistemas desregulamentados é complexa, conseqüentemente apresenta grandes desafios. Os principais motivos desta complexidade são a existência de um grande número de geradores independentes injetando potência na rede elétrica e competindo entre si, o aumento da geração distribuída, e a demanda por alta confiabilidade e qualidade da energia. Além disso, a dificuldade em se planejar e incrementar os sistemas de transmissão faz com que cada vez mais a rede elétrica opere perto dos seus limites de carregamento (BALLANCE et al., 2003). Diante deste cenário, os *Energy Management Systems* (EMS) contemporâneos devem incorporar as melhores tecnológias disponíveis em termos de hardware e software (XINGPING et al., 2002), para que o sistema possa ser operado de uma forma transparente, segura e confiável.

A evolução das tecnologias de software e hardware aplicadas a EMS é comentada em NEYER et al. (1990). Os autores descrevem que no começo da década de noventa, as arquiteturas de hardware apresentavam consideráveis avanços quando comparadas com as arquiteturas de software para centros de controle. Como um exemplo destes avanços, NEYER et al. (1990) descrevem que na década de sessenta e até a metade da década de setenta, utilizavam-se exclusivamente ambientes centralizados e proprietários para a supervisão e controle da operação de SEE. No final

da década de setenta, redes de terminais com velocidade considerável para a época, viabilizaram o início da descentralização, permitindo por exemplo, o controle do sistema através de terminais remotos conectados ao servidor do ambiente. Já nos anos oitenta, com a popularização das redes de computadores, o controle poderia ser realizado através destas redes. Estes desenvolvimentos, em termos de hardware, mostram a disparidade entre as arquiteturas de hardware e software existentes na época. Em contraponto a evolução em termos de hardware, no final da década de oitenta, os softwares utilizados nos centros de controle eram baseados em tecnologias das décadas de cinqüenta e sessenta (NEYER et al., 1990). A necessidade de avanços na área de software aplicados a SEE são relatados em SHEIDT (1987), onde o autor descreve os pontos fortes e fracos das principais linguagens de programação utilizadas no desenvolvimento de software para SEE (ADA, Fortran, Pascal e C), deixando a cargo do leitor a escolha da linguagem mais adequada a sua necessidade. O objetivo central era estimular a discussão em torno das diversas linguagens de programação existentes naquele período.

Outra importante questão naquela época era a dificuldade em se projetar e implementar softwares de grande porte. Como os desenvolvimentos de software eram implementados utilizando-se linguagem estruturada, havia um forte acoplamento entre as estruturas de dados e as metodologias. Este forte acoplamento levava, na maioria das vezes, a códigos inflexíveis e de difícil modificação e adaptação (HAKAVIK et al., 1994). Pequenas mudanças em partes do código acabavam se estendendo por todo o programa.

Na busca de uma possível solução para estes problemas, muitos pesquisadores se dedicaram a explorar as potencialidades de aplicação de técnicas de Modelagem Orientada a Objetos (MOO) (COX, 1986, WHITE et al., 1986) para problemas de SEE. Os aspectos principais de alguns destes trabalhos são abordados nos próximos itens.

1.2 A Primeira Geração de Aplicações de MOO em SEE

As primeiras aplicações de MOO na resolução de problemas de SEE apareceram entre o final da década de oitenta e começo da década de noventa. Um dos primeiros trabalhos foi apresentado por NEYER et al. (1990), que aplicaram técnicas de MOO na

resolução de um problema de fluxo de potência, avaliando-se, dessa forma, as potencialidades destas técnicas na modelagem computacional de redes elétricas. A estrutura proposta é bem simples e parte de uma classe única *Objetos*, subdividindo-se em *Físicos* e *Conceituais*. Naquele trabalho foram relatados problemas relativos ao desempenho computacional do programa implementado, no qual utilizou-se a linguagem de programação Objective C. Alguns testes realizados com códigos escritos e compilados em linguagem C++ apresentaram tempos de processamento 1,5 vezes maiores que aqueles implementados com linguagens tradicionais (Fortran 77).

Nesta mesma linha, HAKAVIK et al. (1994) propuseram uma nova estrutura hierárquica para os elementos de redes elétricas, classificando os elementos em dois grandes grupos: *Conexões* e *Barramentos,* baseados na estrutura física do sistema. Os autores mostraram a possibilidade de se reutilizar códigos já consolidados para a resolução de problemas particulares, como por exemplo a solução de sistemas lineares, mesmo que codificados em outras linguagens; tais códigos foram encapsulados no interior dos objetos da estrutura. Os mesmos autores mostram também que a MOO não causa necessariamente aumento do tempo computacional.

FOLEY et al. (1995) aplicaram a MOO para desenvolver uma aplicação de análise de redes elétricas. O trabalho descreve uma estrutura de classes baseada nos dispositivos físicos dos SEE, com uma classificação de acordo com o número de nós de cada elemento. Nenhum método específico de modelagem foi utilizado, nem para o projeto, nem para a notação das estruturas. Problemas de tempo computacional, semelhantes aos descritos por NEYER, também são relatados.

ZHOU (1996) aplicou a MOO em um poblema de fluxo de potência, relatando bons desempenhos computacionais para esta aplicação. A estrutura de classes está baseada em três elementos: *Barramento*, *Ramo* e *Rede*. A partir daí especializa-os para cada aplicação a ser implementada (fluxo de potência linear e não-linear, etc.).

Em MANZONI (1996) e MANZONI et al. (1999), os autores desenvolveram um simulador da dinâmica de SEE, utilizando MOO. O trabalho apresenta uma estrutura de classes bastante *horizontal*, com a eliminação de vários níveis hierárquicos, normalmente utilizados por outros autores, nas estruturas para a representação de SEE. A MOO foi também aplicada na resolução de sistemas lineares esparsos, resultando em desempenho computacional semelhante ao obtido com programas tradicionais.

FUERTE-ESQUIVEL et al. (1998) desenvolveram um programa de fluxo de potência utilizando MOO na modelagem do sistema elétrico, incluindo a representação de dispositivos FACTS. A estrutura de classes proposta, a exemplo de MAZONI et al. (1996), possui poucos níveis hierárquicos, sendo que de uma classe geral *PowerSystemModel* derivam os objetos do sistema, assim como o método de análise *Flow*, modelado como um objeto de aplicação. Os autores apresentaram resultados de desempenho do código orientado a objetos, implementado em linguagem C++, da ordem de 17% mais lento em relação a implementação usando Fortran 77.

Em ARAUJO et al. (2000), metodologias para a solução de sistemas lineares esparsos de ordem elevada, sob o paradigma da MOO, foram desenvolvidas. Os autores relatam a obtenção de bons tempos computacionais e a aplicabilidade da MOO na solução de sistemas lineares, associados a redes elétricas de grande porte.

Analisando-se os artigos mencionados acima constata-se que os trabalhos desta geração tinham as seguintes características principais:

- buscavam demonstrar a viabilidade de aplicação da MOO na resolução de problemas de SEE;
- procuravam avaliar a demanda de tempo computacional e de consumo de memória de um programa escrito sob o paradigma da MOO, comparativamente aos programas escritos usando linguagem estruturada;
- o entendimento da estrutura de classes do projeto era dificultada pela inexistência de uma linguagem padrão de modelagem e visualização;
- as estruturas de classes eram projetadas e modeladas para a resolução de um problema em particular.

1.3 A Segunda Geração de Aplicações de MOO em SEE

No final da década de noventa, os avanços das arquiteturas de hardware e a consolidação das técnicas e compiladores, baseados no paradigma da MOO, fizeram com que a preocupação com tempo computacional e a demonstração das potencialidades de aplicação de técnicas de MOO em SEE adquirissem menor

relevância. O grande desafio passou a ser o de encontrar uma estrutura de classes que represente o sistema elétrico e sirva como base para a implementação das diversas metodologias de análise e síntese aplicáveis aos SEEs.

Em PANDIT et al. (2000), os autores defendem a separação entre a modelagem dos elementos físicos de um SEE e suas aplicações. Os autores afirmam que a modelagem dos elementos físicos é dependente da aplicação, ou seja, as características dos objetos que modelam elementos físicos de um SEE dependem em parte do tipo de aplicação que se deseja executar. Assim, dividem os atributos destes objetos em dois conjuntos: primários e secundários. O primeiro conjunto é referente às características físicas dos elementos, e o segundo, às aplicações. Sua estrutura de classes deriva de uma classe central, chamada *network*. Esta classe agrega os elementos físicos, e também uma instância de uma matriz esparsa (matriz admitância nodais do sistema). Da classe *network* também derivam as aplicações, tais como fluxo de potência e análise de curto-circuito. O artigo utiliza a notação gráfica do método de Booch (BOOCH, 1998). No ano seguinte, outro trabalho dos mesmos autores (PANDIT et al., 2001) detalha a etapa de um processador de topologia de redes elétricas, projetado segundo os conceitos da MOO.

Em AGOSTINI et al. (2000), os autores identificaram as principais classes relativas aso elementos físicos dos SEE, apresentando exemplos e aplicações. Porém algumas questões ainda necessitavam de maiores desenvolvimentos, principalmente em relação a classificação hierárquica de elementos sem conexões diretas à barras do sistema, bem como em relação a forma de se acomodar as diversas funcionalidades que os elementos físicos podem assumir, diante das diferentes metodologias de análise e síntese a serem desenvolvidas. Em um trabalho posterior, AGOSTINI (2002a) apresenta uma nova filosofia para o desenvolvimento de softwares para Sistemas de Energia Elétrica (SEE), baseada no paradigma da Modelagem Orientada a Objetos (MOO). Ela engloba desde aspectos gerais, relacionado a delimitação adequada do escopo de projetos de softwares, e separação de conceitos através de diferentes abstrações do sistema, até questões específicas de implementação de códigos. Neste escopo são considerados os princípios da mantenabilidade, expansibilidade e robustez das estruturas de dados (com conseqüente redução de custos nessas tarefas), reutilização de códigos, e eficiência computacional. A hierarquia de classes para a representação dos elementos físicos dos SEE, na qual elementos estruturais, responsáveis diretos pela

arquitetura da rede elétrica (barras, linhas de transmissão, etc.) e elementos de composição, partes integrantes de elementos estruturais, são separados em diferentes abstrações. As funcionalidades dos elementos físicos são abstraídas em classes independentes da estrutura principal, permitindo a troca dinâmica de funcionalidades. Outro aspecto a destacar nesta proposta é a utilização de padrões de projeto orientados a objeto, dando suporte ao desenvolvimento de estruturas facilmente expansíveis. O uso da notação da *Unified Modeling Language* (UML) com os seus diagramas para a notação gráfica facilitam a comunicação entre os desenvolvedores propiciando clareza e uniformidade ao projeto.

Contemporaneamente ao trabalho de AGOSTINI, BERRY (2000) apresenta em seu artigo uma introdução ao padrão 61970 do IEC para interfaces entre aplicações utilizadas em *Energy Management System* (EMS). Neste mesmo artigo BERRY comenta que a parte mais adiantada do trabalho era o *Common Information Model* (CIM), uma representação orientada a objetos dos principais elementos que compõem a indústria de energia elétrica (abrangendo desde redes elétricas até aspectos de mercado). Segundo o autor o modelo é projetado como um dicionário de dados facilitando a integração das aplicações e providenciando um caminho padrão para a descrição das estruturas pertencentes ao sistema. O CIM não foi concebido para ser usado diretamente, assim sendo alguns projetos tem adequado o modelo às suas necessidades. O CIM usa a notação da *Unified Modeling Language* (UML) para definir os nomes de cada classe, seus atributos e relações com outras classes.

ARAUJO et al. (2002) apresentam um ambiente integrado para análise de sistemas elétricos onde novos métodos e modelos de componentes podem ser facilmente incorporados ou modificados. Os autores afirmam que para uma correta modelagem dos componentes dos sistemas elétricos, deve-se observar o comportamento físico e a conectividade. Assim sendo, os autores apresentam uma estrutura com quatro níveis hierárquicos com funções e objetivos bem definidos. Três métodos de análise de redes elétricas, com representação trifásica e baseados em injeções de corrente, foram implementados utilizando-se a estrutura de classes proposta.

Em síntese, nos trabalhos desta segunda geração identificam-se as seguintes características principais:

- busca de uma estrutura de classes que represente o sistema elétrico como um todo;
- estrutura de classes expansível, que possibilite acompanhar a evolução da tecnologia em termos de equipamentos e metodologias;
- abstrações bem definidas;
- interfaces de comunicação entre as estruturas de elementos físicos e aplicações;
- utilização de linguagens de visualização, predominantemente a UML.

1.4 Objetivos do Trabalho

O objetivo central do presente trabalho é modelar e implementar metodologias de análise de sistemas de potência, seguindo a nova filosofia de projeto de softwares para SEE proposta em AGOSTINI (2002a). Especificamente, são realizados os seguintes desenvolvimentos:

- modelagem e implementação de aplicações de cálculo de Fluxo de Potência utilizando as metodologias de Newton-Rapshon e Desacoplado Rápido;
- modelagem e implementação de uma aplicação de avaliação global da estabilidade transitória;
- modelagem e implementação de uma ferramenta computacional de Avaliação e Melhoria da Segurança Dinâmica (ASDIN);
- modelagem e implementação de uma aplicação de avaliação global da estabilidade transitória usando técnicas de processamento paralelo;

Juntamente com os desenvolvimentos mencionados acima, realizou-se a concepção e a construção de um ambiente paralelo de Processamento Alto Desempenho e de baixo, custo do tipo cluster BEOWULF, para a execução de aplicações de elevada demanda computacional para a área de SEE.

1.5 Estrutura do Trabalho

No Capítulo 2 é realizada uma descrição da base computacional desenvolvida em AGOSTINI (2002a) e utilizada neste trabalho. São descritas as diversas abstrações utilizadas, bem como a representação por pacotes, de acordo com a notação UML. São também apresentadas as estruturas de classes representativas dos elementos físicos bem como as estruturas relativas a representação das metodologias de análise e síntese dos SEE.

No Capítulo 3 é descrita a modelagem e implementação de duas metodologias de solução de problemas de Fluxo de Potência, detalhando-se as classes projetadas e implementadas neste trabalho. Interfaces funcionais representando o comportamento dos elementos físicos em cada aplicação são descritas.

No Capítulo 4 descreve-se a modelagem e implementação de uma ferramenta computacional de Análise da Segurança Dinâmica, modelada e implementada usando-se diagramas da UML e recursos da Base Computacional Orientada a Objetos supracitada. A metodologia de Análise da Segurança Dinâmica implementada no âmbito deste trabalho é baseada na proposição de SOUZA (1999). Vários aspectos são discutidos ao longo do texto como a importância da reutilização de códigos e a importância do uso da Modelagem Orientada a Objetos para a diminuição no tempo de implementação da ferramenta.

No Capítulo 5 apresenta-se uma abordagem preliminar para a utilização dos recursos da BCOO em ambiente computacionais paralelos. Esta abordagem envolve desde a descrição do ambiente de Processamento de Alto Desempenho e baixo custo implantado até a implementação de um programa paralelo usando recursos da BCOO.

Finalmente, no Capítulo 6 são apresentadas as conclusões finais e as sugestões para trabalhos futuros.

CAPÍTULO 2

BASE COMPUTACIONAL ORIENTADA A OBJETOS

2.1 Introdução

Esta dissertação têm como principais objetivos a modelagem e a implementação de metodologias de análise para SEE, calcados na Base Computacional Orientada a Objetos (BCOO) proposta em AGOSTINI (2002a). O foco principal do presente capítulo é a descrição dos aspectos mais relevantes desta BCOO. Esta descrição se faz necessária para o entendimento dos desenvolvimentos implementados neste trabalho e apresentados nos capítulos três, quatro e cinco.

A apresentação inicia com uma descrição geral da Base Computacional Orientada a Objetos tendo como foco a estrutura organizacional da mesma. A apresentação segue com o detalhamento de cada uma das abstrações que compõem a estrutura da base. São descritas também as relações entre as abstrações da base, destacando-se as relações entre a estrutura das *Interfaces Funcionais* e a estrutura de classes que representam os elementos físicos do sistema.

2.2 Aspectos Gerais da Base Computacional

A concepção da BCOO é realizada observando a proposta de AGOSTINI (2002a) para uma nova geração de softwares para SEE, com as seguintes características:

- estrutura de dados estável e única para um grande conjunto de aplicações;

- maior facilidade para o desenvolvimento, atualização e expansão dos códigos, permitindo agilidade na inclusão de novos modelos de equipamentos e metodologias de análise e síntese;

- facilidades para o desenvolvimento de ferramentas em ambientes integrados e constituídas de uma ampla gama de programas aplicativos;

- elevado grau de modularidade e reutilização de códigos já consolidados, sem perda da eficiência;

- facilidade de gerenciamento de módulos desenvolvidos por diferentes equipes de trabalho;

- banco de dados único para os SEE, com possibilidades de fácil adaptação a novas versões, sem perdas de compatibilidade com versões anteriores.

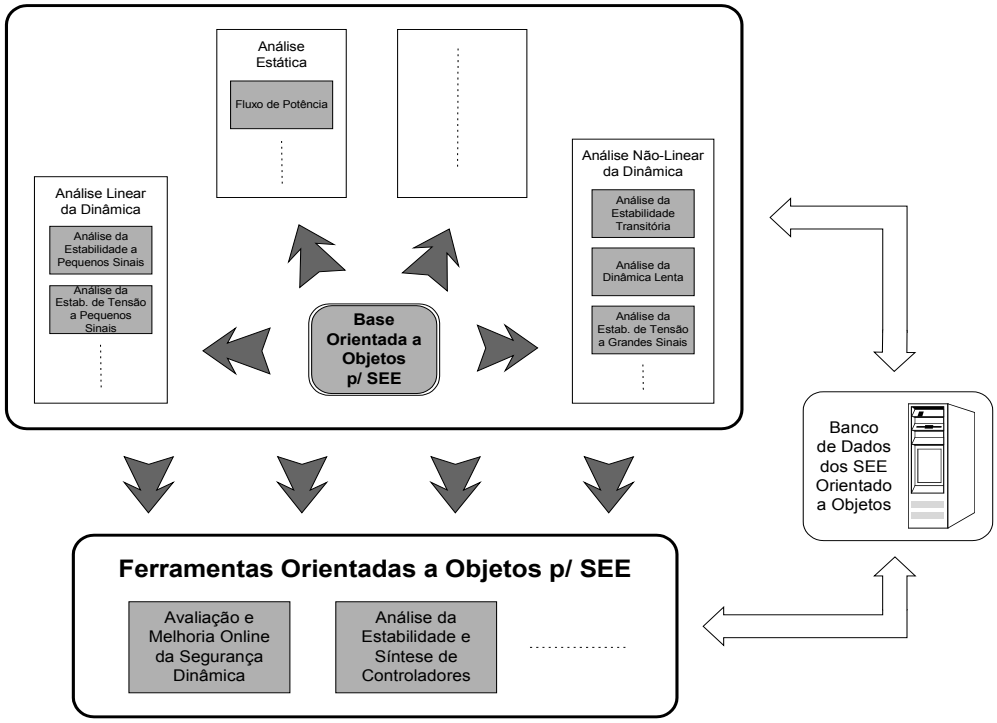

Figura 2.1 – Estrutura Organizacional de Softwares de Análise de SEE

Uma visão geral da estrutura organizacional proposta naquele trabalho para a modelagem computacional de SEE e o desenvolvimento de suas ferramentas computacionais, atendendo os requisitos supracitados é ilustrada pela Figura 2.1.

No centro da parte superior da Figura 2.1 está representada a base computacional orientada a objetos, cuja função é modelar o SEE (elementos físicos), servindo de plataforma para a implementação de diversas metodologias de análise do desempenho de SEE, tais como: cálculo de fluxo de potência, análise da estabilidade a pequenos sinais, etc. Utilizando-se essas aplicações, modeladas e implementadas sob o paradigma da MOO, passa-se para um outro estágio onde poderão ser implementadas ferramentas computacionais, que estão representadas no bloco inferior desta mesma figura. Estas ferramentas são exemplificadas por uma ferramenta para a Avaliação e Melhoria da Segurança Dinâmica, e uma para a Análise da Estabilidade e Síntese de Controladores. Observou-se também, a previsão do desenvolvimento de um banco de dados cuja finalidade é armazenar as informações utilizadas por estas aplicações ou ferramentas. Esta base de dados estará integrada ao ambiente disponibilizando informações às aplicações ou ferramentas a serem executadas.

A idéia principal representada na Figura 2.1 é a de se ter um conjunto de ferramentas computacionais, projetadas e implementadas segundo os preceitos da MOO. Essas ferramentas são organizadas de forma a atuarem em conjunto com uma estrutura hierárquica de classes representativa do SEE. Dessa forma, as ferramentas podem ser flexíveis e robustas, permitindo que manutenções exijam esforços mínimos. Elas dispõem de facilidades para a troca de informações (fluxo de dados) entre si, e com bancos de dados para o armazenamento dos mesmos.

A intenção de AGOSTINI (2002a) foi obter um ambiente integrado de análise e síntese na área de SEE, completamente orientado a objetos, utilizando-se as vantagens que este paradigma oferece, tais como robustez e flexibilidade da estrutura de dados, mantenabilidade e reutilização de códigos.

Os elementos físicos do sistema estão classificados e representados através de uma estrutura hierárquica de classes. Classifica-se os elementos físicos pertencentes ao sistema em dois grandes grupos: elementos *estruturais* e de *composição*. O primeiro grupo representa elementos conectados diretamente às barras do sistema, formando assim, a sua estrutura básica. Entende-se por elementos de composição, elementos que não estão conectados diretamente a alguma barra, porém por meio de composição podem vir a originar elementos estruturais.

Uma funcionalidade interessante presente nesta base é a existência de uma estrutura de classes paralela a dos elementos físicos, e denominadas de *Interfaces*

Funcionais. As interfaces funcionais são responsáveis pela adequação dos elementos físicos às aplicações, isolando-se o comportamento físico do elemento e fornecendo à aplicação as informações necessárias à execução de uma determinada tarefa. Para a implementação desta funcionalidade foi utilizado o padrão orientado a objetos *Adapter* (GAMMA et al., 2000). A aplicação do padrão de projetos *Adapter* facilita a expansão da estrutura, sem a necessidade de modificações nas características básicas dos elementos físicos.

As metodologias de análise e síntese, que chamaremos daqui em diante de aplicações, são também organizadas segundo uma estrutura de classes. Esta estrutura é conectada à estrutura representativa dos elementos físicos do sistema, atuando sobre os mesmos.

Uma terceira estrutura de classes é a das Facilidades Computacionais que são implementadas sob a forma de pacotes e bibliotecas. Os objetos pertencentes a esta estrutura tem como função primordial a execução de tarefas auxiliares como apresentação e armazenamento de dados, gerenciamento de telas, tomadas de tempo de execução, etc.

A implementação computacional das estruturas e funcionalidades desta base foram realizadas utilizando a linguagem de programação C++ (STROUSTRUP, 1997). A principal razão para a utilização da linguagem C++ é a qualidade do suporte que a mesma proporciona à implementação de aplicações modeladas seguindo o paradigma da MOO.

2.3 Modelagem do Sistema: Abstrações

Conforme a teoria de MOO, a Abstração pode ser definida como sendo um exame seletivo de determinados aspectos de um problema (RUMBAUGH J., 1994). Quando aplicam-se os conceitos de MOO na resolução de um determinado problema, deve-se tomar o cuidado de isolar somente o necessário para a modelagem do mesmo. Um bom conhecimento do sistema a ser modelado é necessário da parte do projetista (há técnicas e ferramentas utilizadas para este fim), para que o mesmo capte a quantidade exata de informações necessárias. Este, com certeza, é um dos momentos críticos no desenvolvimento de um projeto baseado em MOO. É o instante em que se

realiza a formulação das estruturas de classes e objetos que compõem o sistema. Em AGOSTINI (2002a) foram propostas três principais abstrações, listadas a seguir:

- Abstração do Sistema Elétrico
- Abstração das Aplicações
- Abstração das Facilidades Computacionais

Todas as abstrações supracitadas fazem parte de uma abstração mais geral chamada de Ferramentas Computacionais, conforme apresentado na Figura 2.2. A abstração do Sistema Elétrico contém as classes que representam os elementos físicos do sistema. As estruturas de classes desta abstração são compostas pelo seguintes conjuntos de classes: Barra, Linhas de Transmissão, Transformadores, Cargas, Unidades de Geração, etc.

A Abstração das Aplicações concentra as classes que representam as metodologias, as quais são derivadas de uma classe base chamada *C_Application*. As aplicações atuam em conjunto com o sistema elétrico, pois os algoritmos de análise e síntese são executados sobre uma base de dados.

Há também a Abstração das Facilidades Computacionais onde são modeladas as funcionalidades que facilitam a implementação de novas aplicações. Por exemplo, um programador que implementará uma nova metodologia para o cálculo de Fluxo de Potência, não precisará estar preocupado com pacotes de leitura e escrita de dados, pois os mesmos já estarão implementados no âmbito desta Abstração.

Englobando todas as abstrações identifica-se a Abstração das Ferramentas Computacionais. Esta abstração tem como foco principal a criação de ferramentas finais para o setor elétrico utilizando os objetos das outras abstrações. Criando-se uma ferramenta computacional têm-se basicamente um objeto SEE, uma ou mais aplicações sendo executadas sobre o sistema elétrico, e alguns objetos realizando tarefas auxiliares.

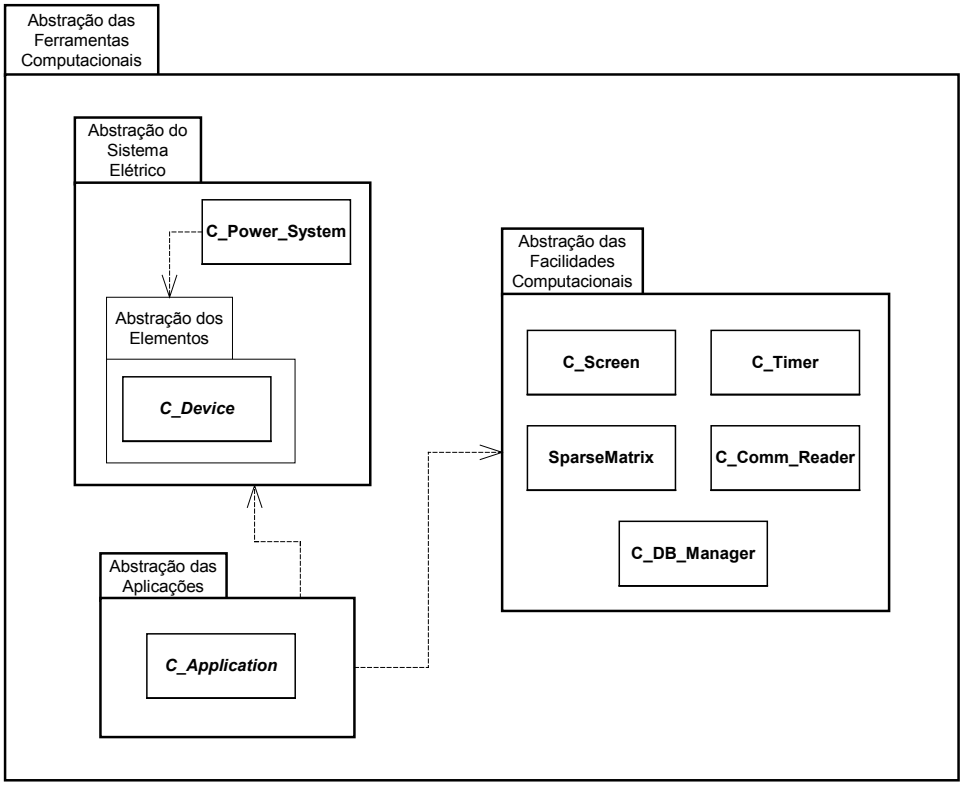

Figura 2.2 - Abstrações na Modelagem Computacional de SEE

2.3.1 Abstração do Sistema Elétrico

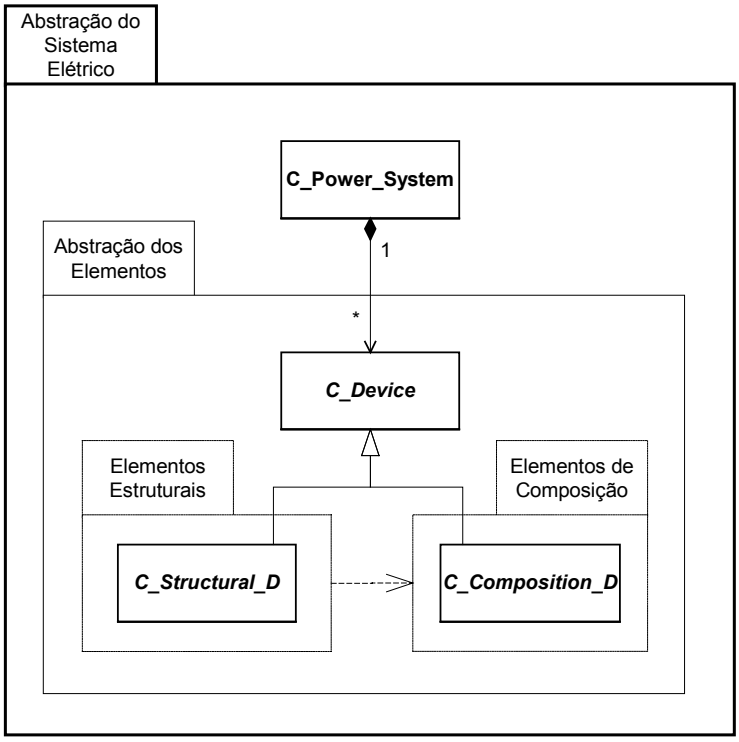

Figura 2.3 – Abstração do Sistema Elétrico

No contexto desta abstração identificam-se duas entidades: a classe *C_Power_System* e a Abstração dos Elementos. A classe *C_Power_System* modela o SEE como um todo, sendo formada pela composição de diversos elementos do sistema. O detalhamento da Abstração do Sistema Elétrico é ilustrado pela Figura 2.3.

Na entidade identificada como Abstração dos Elementos do SEE, tem-se a classe abstrata *C_Device*, que serve de base para todas as outras classes que modelam os elementos físicos, das mais diversas naturezas.

2.3.1.1 Modelagem dos Elementos Físicos

Segundo AGOSTINI (2002a), a obtenção de uma estrutura de classes genérica para a representação de SEE deve cumprir alguns requisitos:

- representar a grande maioria dos elementos de um SEE;
- permitir expansões, objetivando acomodar elementos que poderão vir a ser modelados no futuro;
- facilitar estas expansões e manutenções, fazendo com que futuras alterações, tanto em nível de projeto, como em nível de códigos-fonte, tenham um impacto mínimo nas classes já desenvolvidas;
- ser simples, facilitando seu entendimento para os projetistas e suas equipes;
- ser eficiente, permitindo um bom desempenho computacional na execução das metodologias de aplicação.

Observando estes requisitos, e com o objetivo de conceber uma estrutura genérica para a representação do SEE, AGOSTINI (2002a) adotou o seguinte critério básico para a classificação hierárquica das classes representantes dos elementos físicos destes sistemas. Entende-se por elemento físico todo dispositivo que esteja conectado de uma forma ou de outra ao sistema elétrico, constituindo-o.

AGOSTINI (2002a) propõe então que seja aplicada uma classificação baseada na estrutura física do sistema elétrico. Esta classificação leva em conta em primeira instância o número de conexões à barras que cada elemento apresenta, identificando-se aí pelo menos três classes: as barras propriamente ditas, elementos série (ou branch, com duas conexões) e elementos em derivação (ou shunt, com uma conexão).

Há, entretanto, um problema no momento da aplicação deste tipo de classificação, pois há determinados elementos que não possuem conexão(ões) direta(s) à alguma barra. Porém, quando agregados de uma certa maneira, originam elementos série ou derivação (ou outro tipo qualquer com mais de duas conexões). Um exemplo que pode ser citado é a unidade de geração, elemento tipicamente conectado em derivação ao sistema. Uma unidade de geração pode ser formada por diversos outros elementos, tais como máquina síncrona (MS), regulador de tensão (RAT), regulador de velocidade, estabilizador de sistema de potência (ESP), turbina, caldeira (no caso de unidades térmicas), etc. Estes elementos não possuem conexões diretas à barras, assim sendo estas classes não pertencem a uma estrutura hierárquica baseada no número de conexões.

Com base no exposto acima identificam-se duas abstrações, representadas na Figura 2.4 e denominadas de *Elementos Estruturais* e *Elementos de Composição*. A primeira constitui-se dos elementos ditos estruturais, por formarem a estrutura básica da rede elétrica, conectados através das barras. Nesta estrutura, todos os elementos derivam da classe abstrata denominada *C_Structural_D*. A segunda estrutura constitui-se dos elementos ditos de composição (classe *C_Composition_D*) que, apesar de não possuírem conexões diretas a rede elétrica, formam elementos estruturais através do mecanismo de composição da MOO.

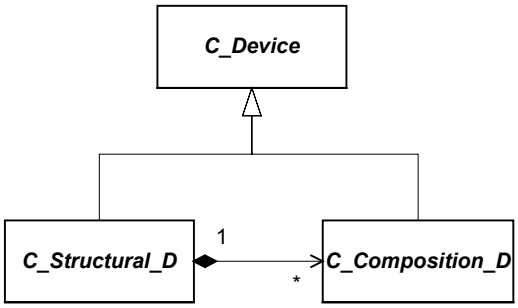

Figura 2.4 – Estrutura de Classes dos Elementos Físicos

Observa-se na Figura 2.4 uma relação de dependência dos elementos estruturais para com os elementos de composição. Nem todos os elementos estruturais são necessariamente formados por elementos de composição, mas o fato de alguns o serem é caracterizado como uma associação entre as classes.

2.3.2 Abstração das Aplicações

Dentro da Abstração das Aplicações são modeladas as mais variadas metodologias de análise e síntese para SEE. O detalhamento desta abstração é mostrado na Figura 2.5. Nesta abstração encontra-se uma classe denominada *C_Application*, a qual serve como base para a estrutura hierárquica das classes representativas de todas as aplicações.

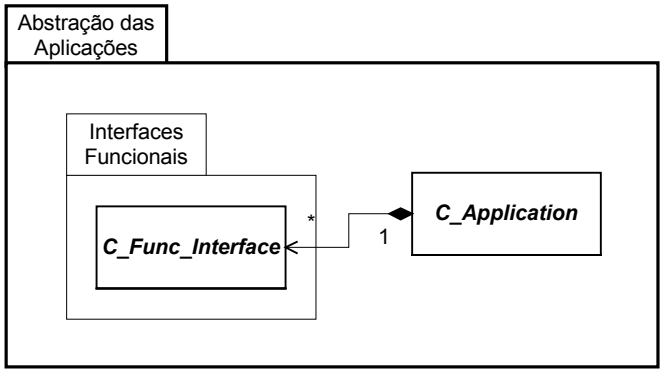

Figura 2.5 – Abstração das Aplicações

A representação das metodologias de análise e síntese (ou aplicações) por uma estrutura hierárquica de classes, tende a ser mais simples que a representação dos elementos físicos dos SEE, principalmente pelo fato de serem conceitos abstratos e não possuírem uma estrutura física. Isto facilita a elaboração desta estrutura, já que não existe uma estrutura de conexão bem definida entre as metodologias delineando as conexões entre as suas classes representativas. Um outro ponto a ser observado é que mudanças na hierarquia de classes desta abstração trarão pouco impacto no projeto de ferramentas computacionais.

Na Figura 2.6 é apresentada a estrutura de classes proposta por AGOSTINI (2002a) para a representação das aplicações. Como já foi citado, todas as aplicações derivam de uma classe abstrata *C_Application*. As diversas metodologias de análise e síntese são agrupadas e representadas por classes abstratas, derivadas de *C_Application*. Cada conjunto de aplicações representando a estrutura acima pode ser detalhado, de acordo com as diversas metodologias que o integram.

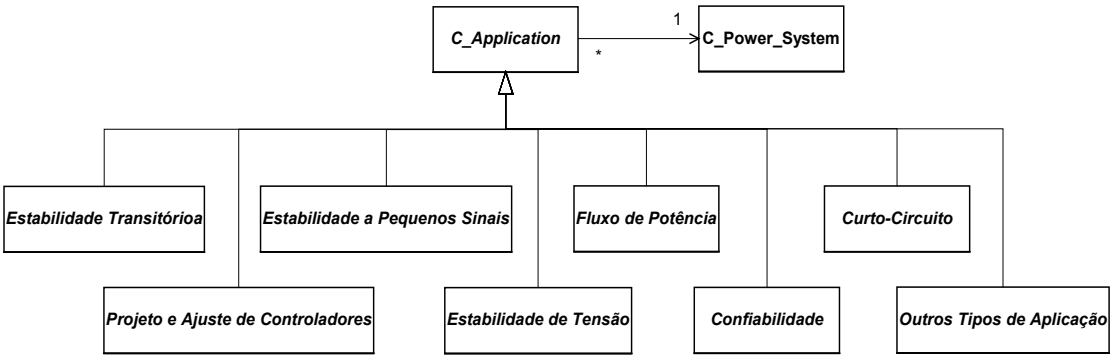

Figura 2.6 – Estrutura de Classes das Aplicações

Verifica-se, a partir da Figura 2.6, que a estrutura hierárquica de classes das aplicações é independente da estrutura que representa os elementos físicos do SEE. As aplicações possuem uma ligação ao objeto SEE (tipo *C_Power_System*), sobre o qual atuam. Em linhas gerais, as aplicações seguem o seguinte padrão de execução:

- cada aplicação solicita uma série de informações ao SEE e seus elementos formadores;
- montam-se as estruturas matemáticas (matrizes, sistemas lineares, conjuntos de equações algébricas e diferenciais, problemas de otimização, etc.);
- as estruturas matemáticas são então resolvidas, obtendo-se a solução do problema a ser resolvido;
- caso seja necessário, grandezas que dizem respeito ao estado do SEE podem ser atribuídas novamente ao objeto SEE e aos seus componentes.

O padrão de projeto *Composite* (GAMMA et al., 2000) é utilizado na modelagem das aplicações, facilitando o reaproveitamento de metodologias implementadas. Dessa forma, aplicações existentes podem ser agrupadas originando uma nova aplicação.

As estruturas de classes foram projetadas de uma maneira que representem as diferentes funcionalidades que os elementos físicos podem assumir frente às aplicações. As características dos elementos físicos (atributos e métodos, considerando-se as classes que os representam) são divididas em dois grupos básicos: *características reais e características funcionais*. Entende-se por características reais a relação direta com os elementos reais do sistema elétrico, tais como suas conexões às barras ou entre si, seus parâmetros, etc. Já as características funcionais estão relacionadas às funcionalidades desempenhadas por cada elemento, quando uma determinada aplicação está sendo

executada sobre o sistema. Estas características são, na maioria das vezes, específicas para cada aplicação, o que impede conceitualmente que sejam armazenadas nas mesmas classes que representam o elemento físico, o que tornaria tal elemento dependente das inúmeras metodologias que podem ser executadas sobre o sistema.

Foi então definido um arranjo de classes visando permitir que ambos os conjuntos de características sejam representados, de forma que as características funcionais possam variar independentemente das características reais dos elementos. Em AGOSTINI (2002a), propõe-se o uso de um padrão de projeto orientado a objetos, chamado *Adapter* (GAMMA et al., 2000), na definição deste arranjo. Este padrão facilita a reutilização de classes já existentes, através de uma adaptação da sua interface.

Na modelagem dos atributos e métodos dos elementos do sistema, as características físicas são modeladas nas classes que representam os elementos físicos, enquanto que as características relativas às aplicações são acomodadas em uma estrutura de classes paralela, que representa exclusivamente as funcionalidades dos elementos em cada aplicação. Considerando-se a estrutura do padrão *Adapter*, as classes da estrutura paralela adaptam as classes da estrutura representativa dos elementos físicos, adicionando a essas as funcionalidades necessárias para uma determinada aplicação. Cada aplicação contém um conjunto de *Interfaces Funcionais* dos elementos físicos, disponíveis para a modelagem das características específicas dos elementos frente à aplicação. As relações entre elementos estruturais, elementos de composição, interfaces funcionais e aplicações podem ser observadas na Figura 2.7. Todo elemento físico (seja estrutural ou de composição) possui um conjunto de interfaces funcionais, cada uma relacionada a uma aplicação específica. Cada interface funcional pertence ao escopo de uma aplicação, e conecta-se a um elemento físico, complementando na aplicação em questão.

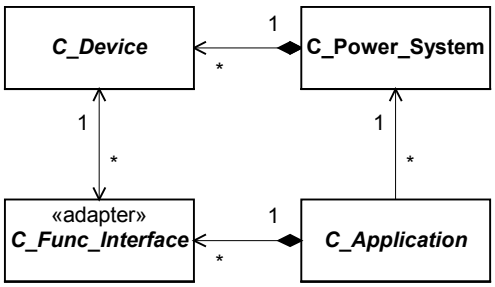

Figura 2.7 – Representação da Relação entre Elementos Físicos e Aplicações

Uma vantagem importante da construção das interfaces funcionais utilizando o padrão *Adapter* é permitir trocas dinâmicas de interfaces. Uma determinada ferramenta computacional pode disponibilizar diversas metodologias de análise e síntese de forma integrada. A troca de uma metodologia para outra pode ser feita em tempo de execução, sem a necessidade de se recriar toda a estrutura física do sistema; cada aplicação cria e destrói seu conjunto de interfaces funcionais para os elementos físicos. Diversas aplicações podem existir ao mesmo tempo, trabalhando de forma sincronizada, pois os elementos físicos podem ter mais de uma interface funcional associada.

Uma profunda descrição das estruturas de classes, tanto das aplicações, como das interfaces funcionais dos elementos físicos, é apresentada em AGOSTINI (2002a).

2.3.3 Abstrações das Ferramentas e Facilidades Computacionais

Os produtos finais a serem obtidos com a utilização da filosofia de projeto apresentada em AGOSTINI (2002a) são ferramentas computacionais, que podem ser utilizadas na forma de programas. Para que um programa computacional aplicável possa ser construído, além das estruturas de dados e funcionalidades que representam a parte do mundo real que se quer estudar, outras entidades devem ser consideradas, as quais são responsáveis por tarefas que auxiliam na execução do mesmo. Tarefas como gerenciamento de telas, leitura e escrita de dados, monitoração de desempenhos computacionais parciais e globais, etc. Incluem-se também entidades com conotações matemáticas, tais como matrizes, vetores, sistemas lineares, etc.

Estas atividades puramente auxiliares são abstraídas da modelagem do SEE em si, e delimitadas por uma abstração denominada Facilidade Computacionais. Estas facilidades são projetadas e implementadas sob a forma de pacotes computacionais independentes, sempre sob o paradigma da MOO. Os pacotes assim criados são então utilizados no projeto das aplicações de análise e síntese, e finalmente na construção das ferramentas para o setor elétrico.

Uma ferramenta computacional é a entidade de mais alto nível no projeto do software. Em termos de MOO, um objeto pode representar a ferramenta, sendo o objeto de mais alto nível no código. Em AGOSTINI (2002a) as ferramentas computacionais são descritas por um objeto formado de um objeto SEE, uma ou mais aplicações que são executadas sobre o SEE, e um conjunto de objetos realizando tarefas puramente computacionais. A abstração das ferramentas computacionais é representada na Figura

2.8. Nesta figura, uma ferramenta genérica é representada pela classe *C_OOTPS* (Ferramenta Orientada a Objetos para Sistemas de Potência).

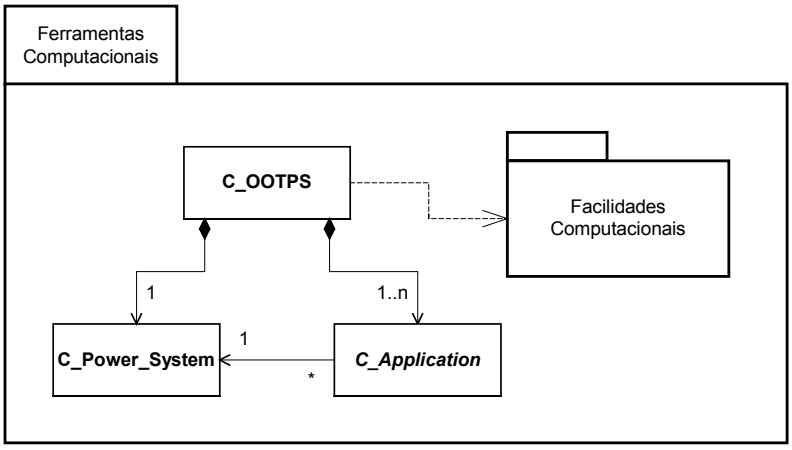

Figura 2.8 – Abstração das Ferramentas e Facilidades Computacionais

Deve-se tomar o cuidado de distinguir o conceito de aplicações do conceito de ferramentas computacionais. As aplicações possuem conotação metodológica, representando as metodologias de análise e síntese aplicadas aos SEE. Já uma ferramenta executa uma ou um conjunto de aplicações em um determinado SEE, sob o aspecto computacional. A ferramenta deve gerenciar recursos de leitura e escrita em arquivos, impressões na tela, leitura e execução de comandos (via teclado ou via arquivo de comando), etc., o que ela faz através das facilidades computacionais.

Em AGOSTINI et al. (2002a) foram implementadas ferramentas computacionais para a validação da filosofia proposta naquele trabalho. O autor desenvolveu uma metodologia de Fluxo de Potência Linear e uma outra para a Simulação Dinâmica com Modelagem Detalhada. Como o presente trabalho visa dar continuidade aos desenvolvimentos daquela tese foram implementadas duas metodologias para cálculo de Fluxo de Potência Não-Linear, uma metodologia para Seleção de Contingências Críticas e uma ferramenta computacional para Avaliação e Melhoria da Segurança Dinâmica para SEE. As respectivas modelagens e implementações são descritas nos capítulos três e quatro desta dissertação.

2.4 O Protótipo OOTPS

Um protótipo de ferramenta computacional foi desenvolvido em AGOSTINI (2002a), com a finalidade de exemplificar a criação de ferramentas computacionais para

o setor elétrico. O protótipo foi chamado OOTPS (*Object Oriented Tool for Power Systems*).

O protótipo é representado como uma classe concreta, chamada *C_OOTPS*, a qual origina, em tempo de execução, um objeto - ferramenta, chamado OOTPS. No âmbito do trabalho de AGOSTINI duas aplicações foram implementadas: um fluxo de potência linearizado (classe *C_Flow_DC*) e uma simulação dinâmica com modelagem detalhada via método Alternado Implícito (classe *C_SIMSP*). Neste trabalho foram adicionadas quatro novas aplicações: um fluxo de potência Newton-Rapshon (classe *C_Flow_NR*), um fluxo de potência Desacoplado-Rápido (classe *C_Flow_FD*), uma aplicação para seleção e classificação de contingências críticas (classe *C_SLEP*) e uma metodologia de Avaliação da Segurança Dinâmica que integra todas as aplicações acima em uma ferramenta computacional (classe *C_ASDIN*). Esta classe é apresentada na Figura 2.9.

Os principais atributos e métodos da classe são:

- ✓ *Power_System: apontador para o objeto SEE;*
- ✓ *Flow_DC: apontador para a sua aplicação de fluxo de potência linearizado;*
- ✓ *SIMSP: apontador para a sua aplicação de simulação dinâmica;*
- ✓ *DB_Manager: apontador para o seu objeto gerenciador de leitura de dados de SEE;*
- ✓ *Comm_Reader: apontador para seu objeto leitor de comandos;*
- ✓ *Execute(): gerencia todo o processo de execução da ferramenta;*
- ✓ *Read_System(): solicita a leitura dos dados do SEE ao seu objeto DB_Manager;*
- ✓ *Manage_Applications(): cria as aplicações e gerencia seu processo de execução;*
- ✓ *Batch_Mode(): configura a ferramenta para o método de entrada de comandos via arquivo em lote;*
- ✓ *Interactive_Mode(): configura a ferramenta para o método de entrada de comandos via teclado;*
- ✓ *Reset(): reinicia a ferramenta.*

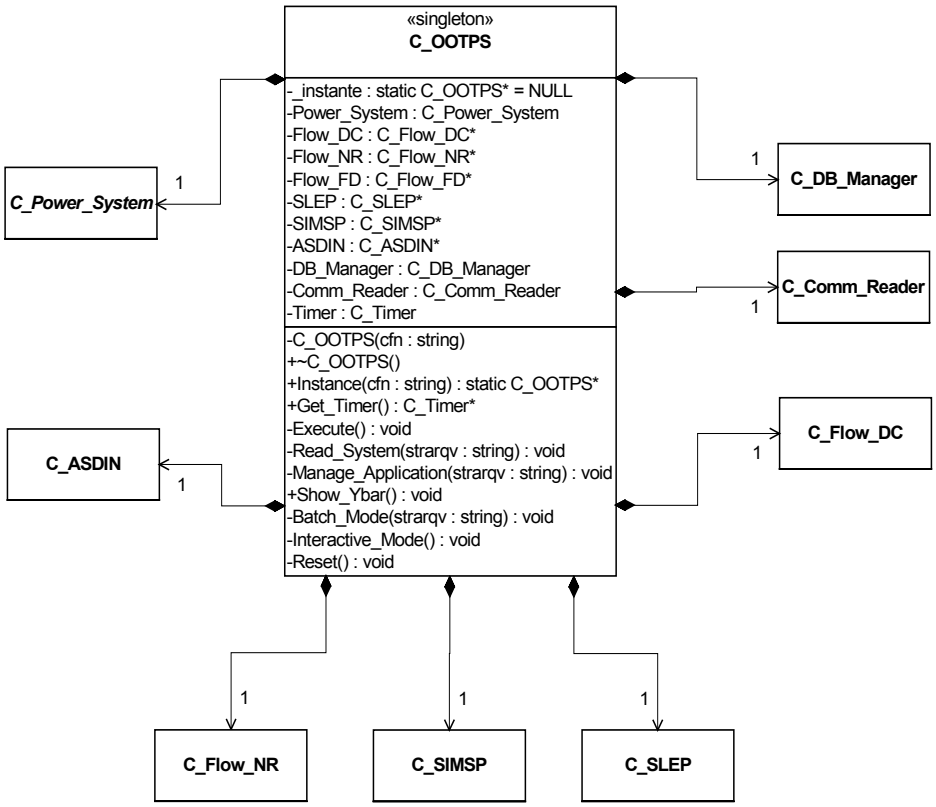

Figura 2.9 - Classe *C_OOTPS*

O conjunto de atributos da classe *C_OOTPS*, listados acima, representam apontadores para os objetos que a constituem. O objeto *Power_System*, o *DB_Manager*, e o *Comm_Reader* são automaticamente criados juntamente com a ferramenta. O SEE é criado vazio, sendo que seu preenchimento, ou seja, a criação dos elementos que o constituem, é realizado durante a leitura dos dados, pelo objeto *DB_Manager*. As aplicações somente são criadas quando um comando para tal é enviado à ferramenta.

O método principal da ferramenta é o *Execute()*, o qual gerencia todo o processo de execução do protótipo. O método basicamente recebe comandos através do seu objeto *Comm_Reader*, identifica-os e executa-os.

Após criado, o objeto *OOTPS* (do tipo *C_OOTPS*) automaticamente cria seus componentes *Power_System*, *DB_Manager* e *Comm_Reader*. A partir daí começa a receber comandos através *Comm_Reader*, e executá-los. Caso o comando seja para ler os dados do SEE, a ferramenta solicita ao seu objeto *DB_Manager* que faça a leitura, de acordo com o tipo de arquivo de dados, e monte o objeto *Power_System*. Caso o comando seja para criar uma determinada aplicação, o método *Manage_Applications()* é ativado. Este identifica qual a aplicação, cria-a e continua a aguardar comandos através

do objeto *Comm_Reader*. Uma vez criada uma aplicação, comandos específicos de cada aplicação podem ser enviados à ferramenta, para que os parâmetros da aplicação sejam configurados (o que é feito pela ferramenta, através do método *Set_Param()* da classe *C_Application* – classe base para todas as aplicações). Após configurada a aplicação, um comando tipo "EXEC" instrui o objeto OOTPS a solicitar à aplicação que efetue sua execução (através do método *Execute()* de *C_Application*). A aplicação é então executada, trocando dados com o SEE sob estudo. Um comando tipo "EXIT", recebido logo após a execução da aplicação, termina o método *Manage_Applications()*, retornando o controle do fluxo do programa ao método *Execute()*. Este processo repete-se até que o método *Execute()* de *C_OOTPS* receba um comando tipo "EXIT", o qual instrui a ferramenta a se encerrar, apagando seus objetos componentes, e por fim a si própria. A ferramenta computacional OOTPS pode ser considerada como sendo um *Framework* para a área de SEE. Mais detalhes sobre essa afirmação são apresentados no próximo item.

2.4.1 *Framework*

Um *framework* é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma categoria de software em específico [DEUTSCH (1989) e JOHNSON (1988)]. GAMMA et al. (2000) citam uma série de pontos importantes encontrados em um *framework*:

- um *framework* serve como base para a criação de uma aplicação específica, através da criação de subclasses modeladas para esta aplicação;

- o *framework* dita a arquitetura da aplicação, definindo a estrutura geral, sua divisão em classes e objetos e em consequência as responsabilidades chave das classes de objetos, como estas colaboram, e o fluxo de controle;

- um *framework* predefine os parâmetros citados acima de maneira que o projetista possa se concentrar nos aspectos específicos da sua aplicação;

- *frameworks* enfatizam a reutilização de projetos em relação a reutilização de código, embora um *framework*, geralmente, inclua subclasses concretas que podem ser utilizadas imediatamente.

Além dos itens citados acima, um *framework* deve ser de fácil utilização e, conseqüentemente, bem documentado. No caso do OOTPS, há uma grande variedade de documentação disponível com diagramas de classes, atividades e de seqüência, bem como uma documentação escrita mostrando como implementar novas aplicações. A validação do OOTPS como sendo um *framework* para a área de SEE foi realizada no âmbito deste trabalho com a implementação das aplicações descritas, em detalhes, nos próximos capítulos.

CAPÍTULO 3

MODELAGEM DE APLICAÇÕES: FLUXO DE POTÊNCIA

3.1 Introdução

Neste capítulo descreve-se a modelagem computacional orientada a objetos da aplicação Fluxo de Potência, segundo a filosofia de desenvolvimento de softwares para SEE proposta por AGOSTINI (2002a), bem como a sua implementação na Base Computacional descrita no capítulo anterior. Especificamente, foram modelados e implementados algoritmos de Newthon-Raphson convencional e o algoritmo Desacoplado Rápido, descritos em MONTICELLI (1983).

O capítulo está dividido em duas partes principais. A primeira consiste da descrição da formulação básica do problema de Cálculo de Fluxo de Potência, usada como base para a proposição das estruturas de classes para o métodos Newton-Raphson e Desacoplado-Rápido. Na segunda parte realiza-se uma breve introdução a cada método em específico, seguindo-se então com o detalhamento da implementação e apresentação das estruturas de classes e dos diagramas de atividades próprios para cada método.

3.2 Modelagem para o Problema de Fluxo de Potência

O cálculo de fluxo de potência é uma das mais importantes aplicações utilizadas na análise de sistemas de energia elétrica. A partir dos resultados deste tipo de cálculo executam-se estudos de projeto, planejamento e operação de sistemas de potência. A modelagem utilizada neste tipo de problema é estática, a rede é representada por um conjunto de equações e inequações algébricas invariantes no tempo, que são

tradicionalmente resolvidas através de métodos numéricos. Existem várias técnicas de solução desenvolvidas para este fim.

De acordo com MONTICELLI (1983), os componentes de um sistema de energia elétrica podem ser classificados em dois grupos: os que estão ligados entre um nó qualquer e o nó terra, como é o caso de geradores, cargas, reatores e capacitores; e os que estão ligados entre dois nós quaisquer da rede, como é o caso de linhas de transmissão, transformadores e defasadores. A classificação segundo o número de conexões ao elemento nó (barra) é também utilizada para classificar os elementos estruturais na BCOO (AGOSTINI et al., 2002a). Isto está ilustrado na Figura 3.1:

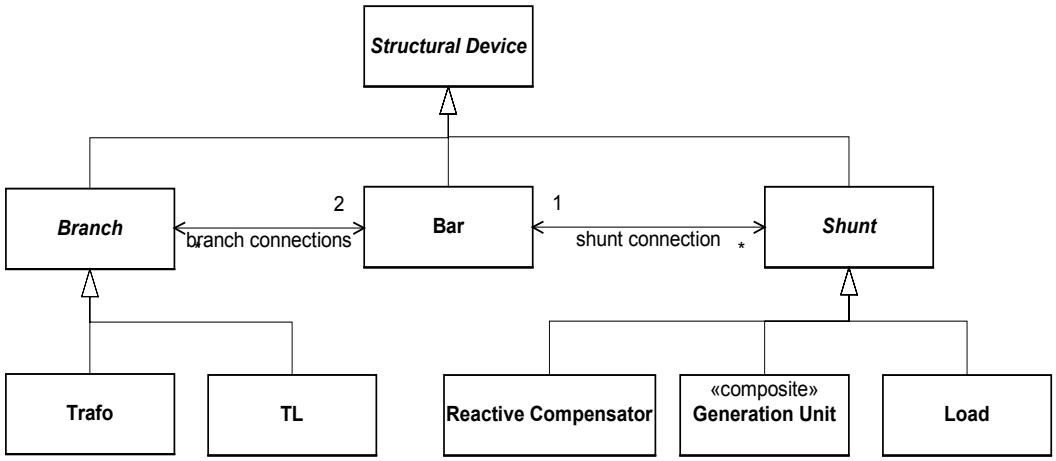

Figura 3.1 – Digrama de Classes dos Elementos Estruturais do SEE

Na formulação matemática do Fluxo de Potência, os geradores e cargas são, em geral, modelados através de injeções de potência aplicados às barras. Na formulação básica do problema de fluxo de potência, para cada barra da rede elétrica, são associadas quatro variáveis, sendo que duas delas entram no problema como dados e duas delas como incógnitas(MONTICELLI, 1983):

- V_k – magnitude da tensão nodal (barra k);
- θ_k - ângulo da tensão nodal;
- P_k – geração líquida (geração menos carga) de potência ativa;
- Q_k – injeção líquida de potência reativa.

Dependendo de quais variáveis nodais entram como dados e quais são consideradas como incógnitas, definem-se três tipos básicos de barras:

- *PQ* – são dados P_k e Q_k, e calculados V_k e θ_k;
- *PV* – são dados P_k e V_k, e calculados Q_k e θ_k;
- Folga – são dados V_k e θ_k, e calculados P_k e Q_k.

As barras dos tipos PQ e PV são utilizadas para representar, barras de carga e barras de geração respectivamente (incluindo-se os compensadores síncronos). A barra de folga fornece a referência angular do sistema e é utilizada para fechar o balanço de potência de todo o sistema. Em termos gerais, o modelo matemático do problema de fluxo de potência é formado por duas equações para cada barra, cada uma delas representado o fato de as potências ativas e reativas injetadas em uma barra serem iguais a soma dos fluxos correspondentes que deixam a barra através de cargas, linhas de transmissão, transformadores, etc. Isto pode ser expresso matematicamente como segue:

$$P_k = \sum_{m \in \Omega k} P_{km}(V_k, V_m, \theta_k, \theta_m)$$

$$Q_k + Q_k^{sh}(V_k) = \sum_{m \in \Omega k} Q_{km}(V_k, V_m, \theta_k, \theta_m) \qquad (2.1)$$

onde:

k: 1, ... NB, sendo NB o número de barras da rede

Ω_k: conjunto de barras vizinhas da barra k

V_k, V_m: magnitudes das tensões das barras terminais do ramo k-m

θ_k, θ_m: ângulos das tensões das barras terminais do ramo k-m

P_{km}: fluxo de potência ativa no ramo k-m

Q_{km}: fluxo de potência reativa no ramo k-m

Q_{sh}: componente da injeção de potência reativa devida ao elemento *shunt* da barra k

A maioria dos algoritmos para o cálculo de fluxo de potência são desenvolvidos a partir desta equação. O interesse na formulação genérica do problema é o conhecimento da formulação básica e conceitual do mesmo para que os diagramas de classes que compõem as estruturas desta aplicação possam ser modelados corretamente,

facilitando futuras expansões. Neste problema em específico, a utilização dos recursos da BCOO é facilitada pois os elementos principais (barra, gerador, linha de transmissão, etc), descritos na formulação básica e necessários para a cálculo de um problema de fluxo de potência, já estão classificados e implementados em código. Entretanto, nem todos os elementos da formulação básica estão implementados fazendo-se então necessário a adequação dos objetos que representam os elementos físicos à aplicação de fluxo de potência não-linear. Esta adequação, como mencionado no capítulo anterior, é realizada através das *Interfaces Funcionais*. Com base na formulação exposta acima, propõem-se a implementação de oito *Interfaces Funcionais* para esta aplicação. A estrutura de classes dessas Interfaces para o problema de fluxo de potência não-linear está ilustrada nas Figuras 3.2 e 3.3.

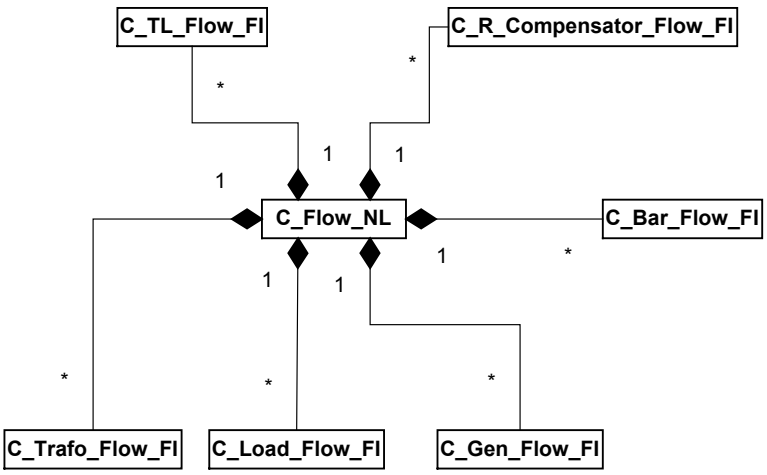

Figura 3.2 – Diagrama de Classes: Fluxo de Potência Não-Linear

As *Interfaces Funcionais* trabalham em conjunto com o objeto fluxo de potência, fazendo parte do mesmo. Quando o objeto fluxo de potência é destruído as *Interfaces Funcionais* também o são.

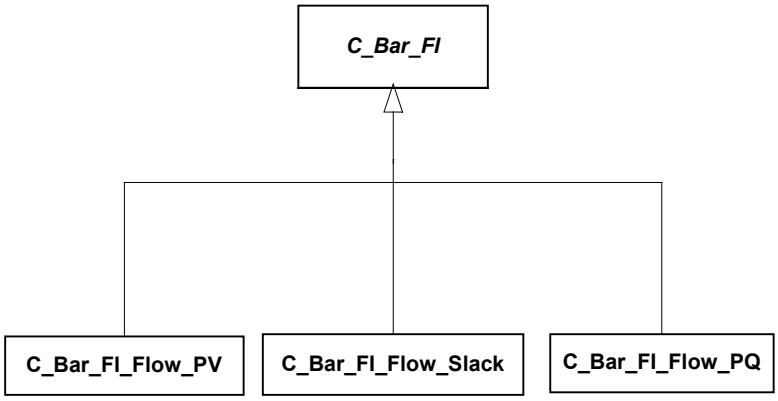

Figura 3.3 – Representação das *Interfaces Funcionais* Barras

A abstração e a modelagem das classes que compõem as aplicações de fluxo de potência não-linear foram realizadas com relativa naturalidade pois a formulação do problema facilita a abstração das classes. Os principais elementos necessários ao cálculo de fluxo de potência, geradores e cargas são modelados como injeções de potência, as barras estão divididas em três arranjos conceituais (*PQ, PV e Folga*) e os elementos série seguem os modelos equivalentes π descritos em MONTICELLI (1983). Um maior detalhamento das *Interfaces Funcionais* bem como da classe Fluxo de Potência será realizado nos próximos itens.

3.3 Modelagem para o Método de Newton Raphson

O processo iterativo para a solução do problema de fluxo de potência via método de Newton-Raphson inicia com a estimação dos valores iniciais para a magnitude e ângulo das tensões nas barras do sistema. Em seguida realiza-se o cálculo dos desbalanços de potência ativa ΔP para barras PV e PQ, e de potência reativa ΔQ para as barras PQ utilizando-se as equações abaixo:

$$\begin{aligned} \Delta\vec{P} &= \vec{P}^{esp} - \vec{P}(\vec{V},\vec{\theta})^{cal} \\ \Delta\vec{Q} &= \vec{Q}^{esp} - \vec{Q}(\vec{V},\vec{\theta})^{cal} \end{aligned} \tag{2.2}$$

O resultado do cálculo dos desbalanços de potência ativa e reativa são comparados com uma tolerância pré-especificada, e caso a tolerância seja atendida o processo iterativo é interrompido. Caso contrário há a necessidade da formação de uma matriz Jacobiana para a resolução de um sistema algébrico linear do tipo:

$$\begin{bmatrix} \Delta\vec{P} \\ \Delta\vec{Q} \end{bmatrix} = \begin{bmatrix} H & N \\ M & L \end{bmatrix} \begin{bmatrix} \Delta\vec{\theta} \\ \Delta\vec{V} \end{bmatrix} \tag{2.3}$$

Os elementos das submatrizes jacobianas ***H,N,M*** e ***L*** são dados por:

$$H = \begin{cases} H_{km} = \partial P_k / \partial \theta_m = V_k V_m (G_{km} \operatorname{sen}\theta_{km} - B_{km} \cos\theta_{km}) \\ H_{kk} = \partial P_k / \partial \theta_k = -Q_k - V_k^2 B_{kk} \end{cases};$$

$$N = \begin{cases} N_{km} = \partial P_k / \partial V_m = V_k (G_{km} \cos\theta_{km} + B_{km} \operatorname{sen}\theta_{km}) \\ N_{kk} = \partial P_k / \partial V_k = V_k^{-1} (P_k + V_k^2 G_{kk}) \end{cases};$$

$$M = \begin{cases} M_{km} = \partial Q_k / \partial \theta_m = -V_k V_m (G_{km} \cos\theta_{km} + B_{km} \operatorname{sen}\theta_{km}) \\ M_{kk} = \partial Q_k / \partial \theta_k = P_k - V_k^2 G_{kk} \end{cases}; \tag{2.4}$$

$$L = \begin{cases} L_{km} = \partial Q_k / \partial V_m = V_k (G_{km} \operatorname{sen}\theta_{km} - B_{km} \cos\theta_{km}) \\ L_{kk} = \partial Q_k / \partial V_k = V_k^{-1} (Q_k - V_k^2 B_{kk}) \end{cases};$$

onde:

$Y_{km} = G_{km} + jB_{km}$: admitância de transferência entre a barra k e m ($k \neq m$)

$Y_{kk} = G_{kk} + jB_{kk}$: admitância própria da barra k

A dimensão do sistema linear é dada pela composição da dimensão das submatrizes do Jacobiano, ou seja, a dimensão é igual a duas vezes o número de barras PQ adicionado do número de barras PV. Após a resolução do sistema linear atualizam-se os valores de tensão e ângulo nas barras do sistema e calculam-se as novas injeções de potência ativa e reativa nas barras do sistema. Para as barras PV, verificam-se os limites de geração de potência reativa. Caso a mesma esteja fora dos limites fixa-se o reativo no limite extrapolado e transforma-se a barra PV em uma barra PQ. Uma nova iteração então é iniciada calculando-se novamente os desbalanços de potência ativa e reativa nas barras e somente cessando com o atendimento da tolerância pré-especificada.

O Diagrama de Atividades desta implementação é apresentado na Figura 3.4:

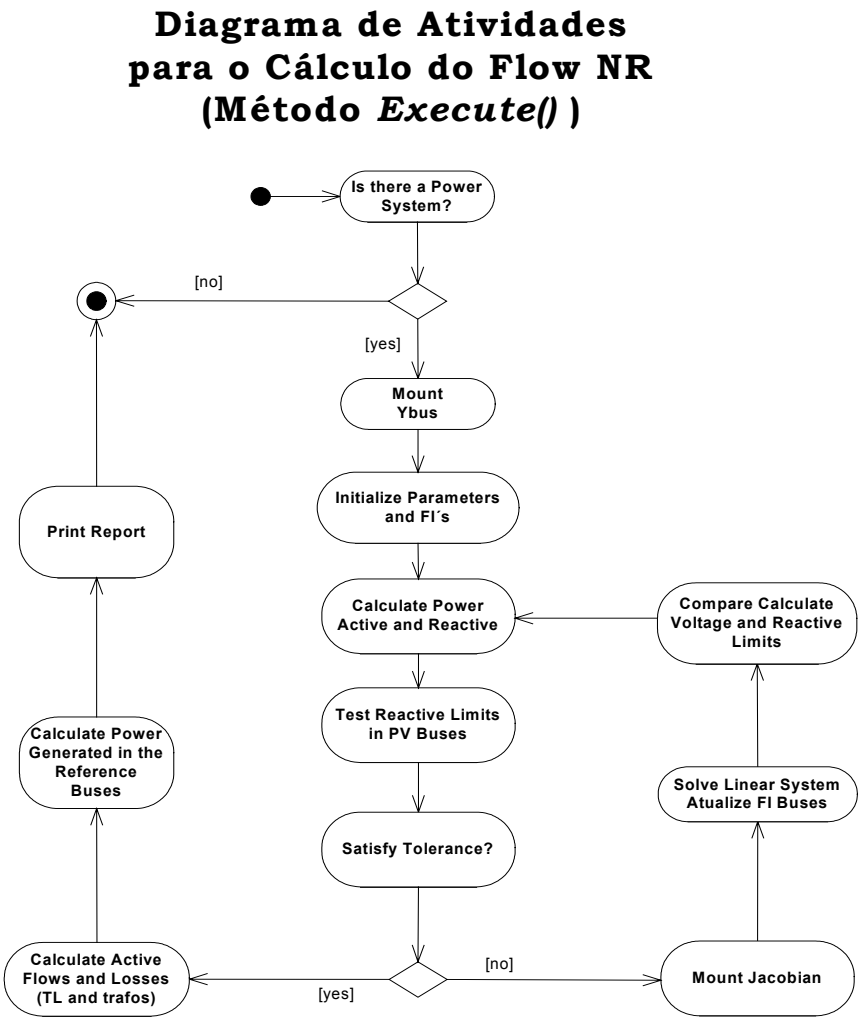

Figura 3.4 – Diagrama de Atividades para a Execução de um Fluxo de Potência por Newton-Rapshon

Nesta implementação, quando a metodologia é ativada pelo método *Execute()* (método herdado da classe *C_Application*), o objeto *C_Flow_NR* verifica em primeiro lugar se existe um SEE para que a análise possa ser realizada. Caso exista, as *Interfaces Funcionais* do Fluxo de Potência são inicializadas, a matriz de admitâncias nodais, é montada e os parâmetros do fluxo de potência e das *Interfaces Funcionais* são inicializados.

Após a inicialização dos parâmetros realiza-se o cálculo das potências ativas e reativas nas barras PQ e PV. A potência reativa é calculada a cada iteração nas barras PV para que o controle de tensão das mesmas possa ser ativado. A potência reativa calculada nas barras PV a cada iteração deve ser comparada com os valores máximos e mínimos definidos nos dados de entrada. Caso algum dos limites de potência reativa da barra PV seja violado, transforma-se a mesma em barra PQ. Como na barra PV o

parâmetro Q^{esp} (potência reativa especificada na barra) não havia sido especificado, atribui-se então a Q^{esp} o valor do limite violado durante o processo de cálculo. Em seguida verifica-se se a tolerância especificada (desbalanços de potências ativa e reativa) está sendo atendida. Caso isso não aconteça, a Matriz Jacobiana do sistema é montada para que o sistema linear seja calculado. O resultado da resolução deste sistema linear é um vetor de correção que contém ângulos e tensões das barras PV e PQ.

Para as barras PV os novos valores das tensões são comparadas com o valor especificado nos dados de entrada (V^{esp}), e caso seja possível ela será novamente transformada em uma barra PQ (MONTICELLI 1983). Ilustrando este procedimento com um exemplo, observa-se o caso em que uma barra PV é transformada em barra PQ, quando seu limite superior de geração de potência reativa é violado. Na próxima iteração o seu valor de tensão é comparado com o valor de tensão especificado. Caso o mesmo seja maior que o valor especificado (maior suporte de potência reativa) ela é novamente transformada em barra PV, caso contrário continuará se comportando como uma barra PQ.

Realizada a conferência dos valores de tensão das barras PV, fecha-se o laço principal calculando-se novamente as potências ativas e reativas nas barras PQ e PV. Satisfeita a tolerância calculam-se as potências geradas na barra de referência, os fluxos e as perdas nos dispositivos séries do sistema. Finalizando o programa é impresso um relatório contendo dados de barra (tensão, ângulo e potências) e linhas (fluxos e perdas).

A estrutura de classes para a aplicação de cálculo de Fluxo de Potência por Newton Rapshon é mostrada na Figura 3.5.

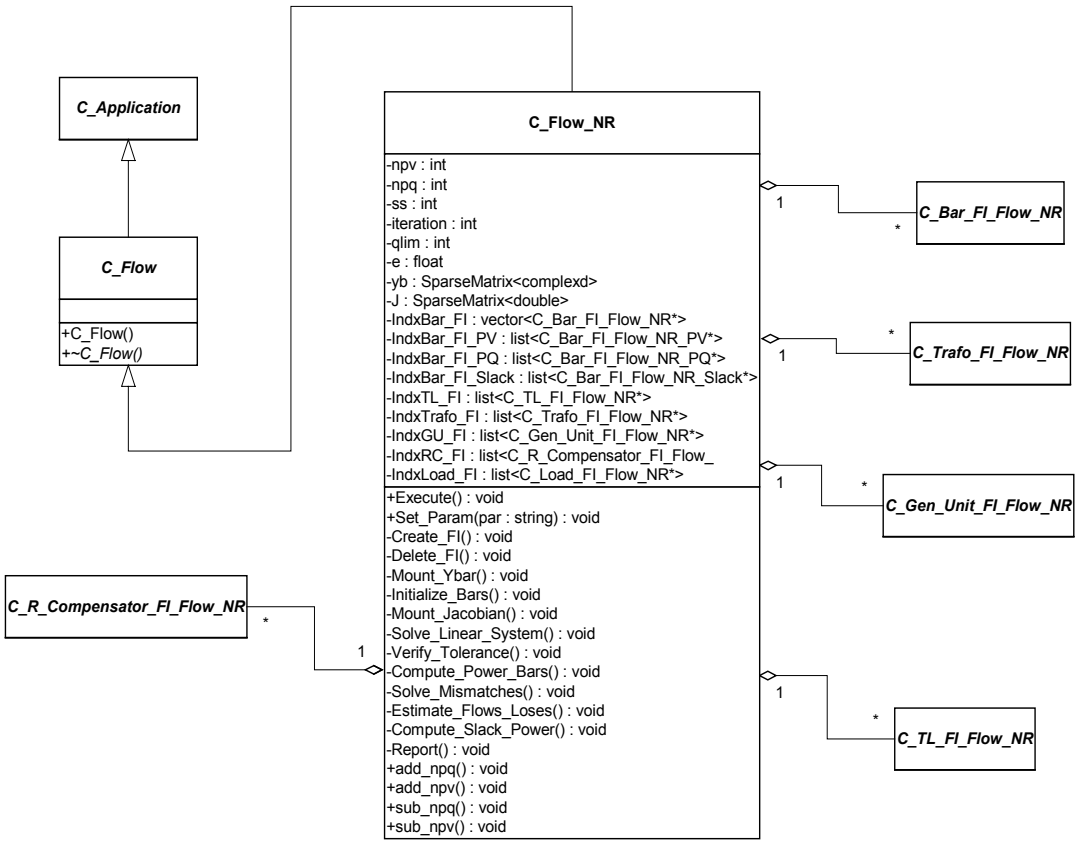

Figura 3.5 – Visão Geral da Aplicação Fluxo de Potência via Método de Newton-Rapshon

Os principais atributos e métodos de *C_Flow_NR* são:

- ✓ *npv, npq e ss*: número de barras PV, PQ e dimensão total do sistema, respectivamente;
- ✓ *qlim*: opção de controle de potência reativa nas barras PV;
- ✓ *e:* tolerância de convergência;
- ✓ *yb*: matriz que armazena a Ybarra do sistema;
- ✓ *J:* matriz responsável pelo armazenamento da matriz Jacobiana do sistema;
- ✓ *IndxBar_FI*: vetor de interfaces funcionais de barras;;
- ✓ *IndxBar_FI_Slack:* lista de interfaces funcionais para a barra de referência;
- ✓ *IndxBar_FI_PV*: lista de interfaces funcionais referentes às barras PV;
- ✓ *IndxBar_FI_PQ*: lista de interfaces funcionais referentes às barras PQ;
- ✓ *IndxTL_FI:* lista de interfaces funcionais das linhas de transmissão;
- ✓ *IndxTrafo_FI*: lista de interfaces funcionais dos transformadores;
- ✓ *IndxGU_FI*: lista de interfaces funcionais dos geradores;
- ✓ *IndxRC_FI*: lista de interfaces funcionais dos compensadores reativos;
- ✓ *Indx_Load:* lista de interfaces funcionais das cargas;

- ✓ *Execute():* método herdado da classe *C_Application*, responsável pela execução da aplicação;
- ✓ *Set_Param():* método herdado da classe *C_Application*, responsável pela atribuição dos parâmetros da aplicação;
- ✓ *Create_FI():* cria as interfaces funcionais dos elementos;
- ✓ *Delete_FI():* elimina as interfaces funcionais;
- ✓ *Mount_Ybar():* monta a matriz Ybarra;
- ✓ *Initialize_Bars():* inicializa os parâmetros do fluxo e das Interfaces Funcionais do sistema;
- ✓ *Mount_Jacobian():* calcula a matriz Jacobiana;
- ✓ *Compute_Power_Bars():* realiza o cálculo das potências nas barras PQ e PV;
- ✓ *Verify_Tolerance():* verifica se a tolerância esta sendo atendida;
- ✓ *Solve_Linear_System():* resolve um sistema linear do tipo $A.x = b$;
- ✓ *Compute_Slack_Power():* calcula as potências na barra de referência;
- ✓ *Solve_Mismatches():* ativa os métodos das Interfaces Funcionais que calculam os desbalanços de potências na barra especificada;
- ✓ *Estimate_Flows_Loses():* ativa os métodos das Interfaces Funcionais (linhas e transformadores) para que calculem os seus fluxos e perdas;
- ✓ *Report():* imprime o relatório de resultados.

3.3.1 Interfaces Funcionais - Método de Newton Raphson

Neste item apresenta-se a uma descrição detalhada das *Interfaces Funcionais* que atuam na aplicação de Fluxo de Potência via método de Newton-Raphson. Nesta implementação definiram-se oito *Interfaces Funcionais* principais que são descritas a seguir:

3.3.1.1 Classe *C_Bar_FI_Flow_NR*:

A classe *C_Bar_FI_Flow_NR* é uma classe abstrata que representa o objeto físico barra. Esta classe providencia a base para a criação de outras classes barras que não representam entidades físicas mas sim conceituais. Estas entidades conceituais facilitam a implementação dos algoritmos para o cálculo de Fluxo de Potência Não-Linear. As barras em um programa de Fluxo de Potência Não-Linear desempenham diferentes funções que necessitam ser representadas de diferentes formas. Assim sendo a formulação básica de um problema de fluxo de potência define três tipos básicos de barras (PQ, PV e FOLGA) e foram estes os tipos modelados neste trabalho. Observa-se porém que outros tipos de barras (PQV, P e V por exemplo) podem ser facilmente definidos conforme a necessidade, porém grande parte dos problemas são resolvidos somente utilizando este tipo de modelagem.

A função primordial da *Interface Funcional* barra é a de servir de estrutura de dados para o cálculo do fluxo de potência. Assim sendo, ela é responsável pelo armazenamento de dados como tensão, ângulo, tipo, posição da matriz jacobiana, potência líquida, etc. Além disso, ela realiza algumas operações como por exemplo o cálculo da potência líquida injetada na barra. Este método percorre as listas de elementos *Shunt* (basicamente cargas e geradores) solicitando as potências de cada dispositivo. As injeções de potências líquidas são então armazenada nos atributos P^{esp} e Q^{esp} da Interface Funcional da barra. A estrutura de classes das Interfaces Funcionais barras, para a metodologia de fluxo de potência via Newton-Raphson, é mostrada na Figura 3.6:

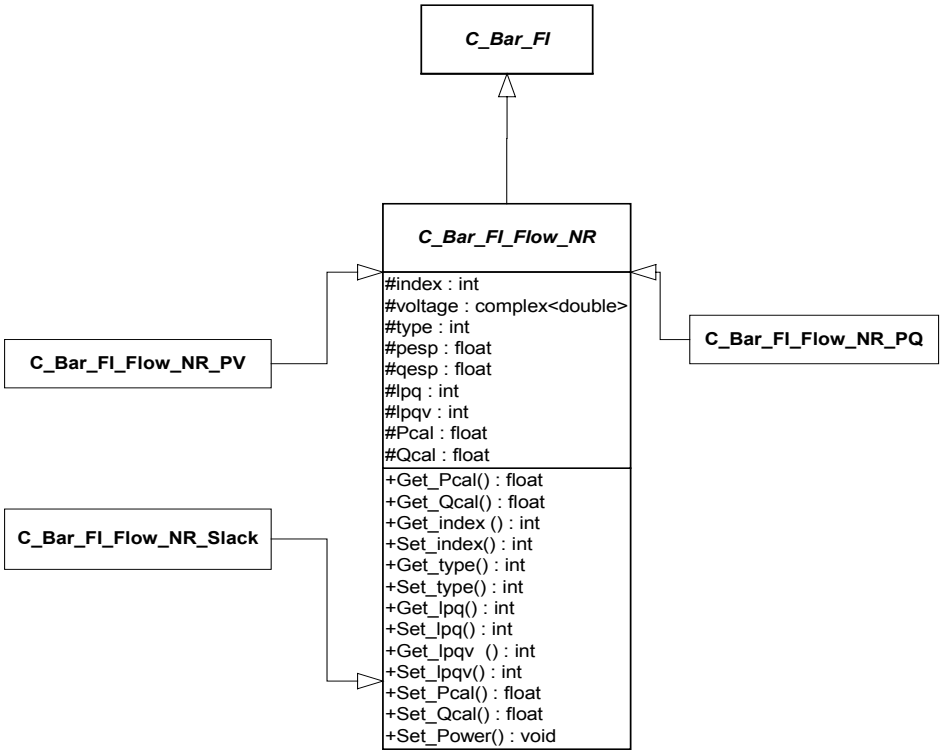

Figura 3.6 – Interface Funcional *C_Bar_FI_Flow_NR*

Os principais atributos e métodos utilizados nesta classe são:

- ✓ *index:* índice da barra e de localização na matriz de admitâncias nodais;
- ✓ *voltage*: tensão na barra;
- ✓ *type*: tipo da barra (PV, PQ ou FOLGA);
- ✓ *pesp*: potência ativa total injetada na barra;
- ✓ *qesp*: potência reativa total injetada na barra;
- ✓ *Pcal*: potência ativa calculada a cada iteração;
- ✓ *Qcal*: potência reativa calculada a cada iteração;

- ✓ *lpqv*: índice de localização da barra nas submatrizes H,N e M;
- ✓ *lpq*: índice de localização da barra nas submatrizes N,M e L;
- ✓ *Get_Pcal()*: retorna a potência ativa calculada na iteração;
- ✓ *Get_Qcal()*: retorna a potência reativa calculada na iteração;
- ✓ *Set_Pcal()*: altera valor de *Pcal*;
- ✓ *Set_Qcal()*: altera valor de *Qcal*;
- ✓ *Get_index()*: retorna o atributo *index*;
- ✓ *Set_index()*: altera o conteúdo de *index*;
- ✓ *Get_type()*: retorna o tipo da barra;
- ✓ *Set_type()*: configura o tipo da barra;
- ✓ *Set_Power()*: calcula a injeção de potência total em cada barra;
- ✓ *Get_lpq() e Get_lpqv()*: retorna o índice de posição na matriz Jacobiana;
- ✓ *Set_lpq() e Set_lpqv()*: altera o valor dos atributos *lpq* e *lpqv*;

A partir desta classe abstrata derivam-se três outras classes concretas representando as barras PQ, PV e FOLGA. Esta divisão é necessária para uma correta representação e separação conceitual dos tipos de barras formuladas para o problema de fluxo de potência. Além disso, estas interfaces contém atributos e métodos específicos que devem ser colocados em classes em separado. Um exemplo é a classe *C_Bar_FI_Flow_NR_PV* , que contém atributos e métodos específicos de uma barra PV como por exemplo os limites de geração de reativo. Estes limites são próprios de uma barra PV pois a mesma conta com um dispositivo de controle de tensão. Este controle de tensão é realizado através da injeção de reativo na barra. O elemento físico que realiza esta injeção tem limites que devem ser respeitados e são representados no objeto *Bar_FI_Flow_NR_PV* pelos atributos *Qmin* e *Qmax*. Estes atributos somente dizem respeito a barra PV já que a barra PQ não dispõe de dispositivos para o controle de tensão. Há também métodos específicos desta barra que são utilizados para a configuração do controle de reativo. São eles: *Test_Reative_Limits*() e *Set_Reative_Limits()*. A classe *C_Bar_FI_Flow_NR_PV* com seus atributos e métodos é apresentada na Figura 3.7.

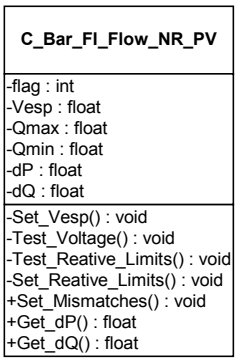

Figura 3.7 – Interface Funcional *C_Bar_FI_Flow_NR_PV*

Os principais atributos e métodos desta classe são:

- ✓ *flag*: sinaliza se a barra mudou o seu tipo para PQ durante o processo iterativo;
- ✓ *Vesp*: variável que guarda o valor especificado da tensão;
- ✓ *Qmax*: valor máximo de potência reativa;
- ✓ *Qmin*: valor mímino de potência reativa;
- ✓ *dP*: valor do desbalanço de potência ativa;
- ✓ *dQ*: valor do desbalanço de potência reativa;
- ✓ *Set_Reactive_Limits()*: altera os limite de reativo;
- ✓ *Test_Reactive_Limits()*: verifica se o limite de reativo máximo e mínimo esta sendo respeitados;
- ✓ *Test_Voltage()*: testa a tensão em uma barra PV que foi modificada para PQ. Verifica se a mesma pode retornar a sua condição original;
- ✓ *Set_Mismatches()*: calcula o valor dos desbalanços de potência;

As *Interfaces Funcionais C_Bar_FI_Flow_NR_Slack* e *C_Bar_FI_Flow_NR_PQ* não são apresentadas aqui pois as mesmas apresentam pequenas diferenças em termos de atributos e métodos, quando comparadas com a *Interface Funcional C_Bar_FI_Flow_NR_PV*. Elas contém os métodos descritos nesta interface menos os métodos relativos ao controle de limite de reativo na barra. Há uma exceção quanto à *Interface Funcional C_Bar_FI_Flow_NR_Slack* que não contém o método *Set_Mismatches()*; pois a barra de folga não necessita realizar tal operação. O exposto acima ratifica a necessidade da divisão entre as *Interfaces Funcionais* barra. Observa-se ainda que outros métodos específicos de controle podem ser implementados e que dizem respeito somente às barras PQ ou PV, como por exemplo o controle de limite de tensão nas barras PQ.

3.3.1.2 Classe *C_TL_FI_Flow_NR*

A classe *C_TL_FI_Flow_NR* realiza todo o trabalho de manipulação dos parâmetros das linhas de transmissão para que os mesmos se adeqüem as necessidades específicas do Fluxo de Potência Newton Raphson. O método *Set_LT_Params()* calcula, a partir dos parâmetros físicos do elemento, os parâmetros do modelo π para linhas de transmissão. O resultado do método são os valores de *Yeq* e *Ysh* atualizados. Estes parâmetros serão usados na montagem da matriz de admitâncias nodais do sistema (Ybarra). O método *Calc_Flows_Loses()* calcula os fluxos e as perdas nas linhas de transmissão com base nas tensões e ângulos resultantes do fluxo de potência. Os métodos *Get_Ploss(), Get_Qloss(), Get_Pflow() e Get_Qflow()* retornam os valores calculados em *Calc_Flows_Loses()*, que posteriormente serão impressos em um relatório. A classe *C_TL_FI_Flow_NR* é mostrada na Figura 3.8.

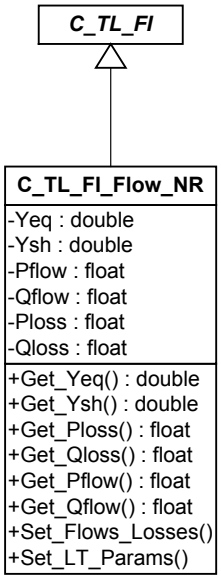

Figura 3.8 – Interface Funcional *C_TL_FI_Flow_NR*

Seus principais atributos e métodos são:

- ✓ *Yeq*: admitância equivalente em série do circuito;
- ✓ *Ysh*: metade do valor da admitância equivalente em derivação;
- ✓ *Pflow*: potência ativa circulante na linha;
- ✓ *Qflow*: potência reativa circulante na linha;
- ✓ *Ploss*: perdas de potência ativa na linha;
- ✓ *Qloss*: perdas de potência reativa na linha;
- ✓ *Get_Yeq()*: retorna o atributo Yeq;
- ✓ *Get_Ysh()*: retorna o atributo Ysh;

- ✓ *Get_Ploss()*: retorna o valor da potência ativa perdida na linha;
- ✓ *Get_Qloss()*: retorna o valor da potência reativa perdida na linha;
- ✓ *Get_Pflow()*: retorna o valor da potência ativa circulante na linha;
- ✓ *Get_Qflow()*: retorna o valor da potência reativa circulante na linha;
- ✓ *Calc_Flows_Losses()*: calcula os fluxos de potência (ativa e reativa) nas linhas e as perdas;
- ✓ *Set_LT_Params()*: configura os parâmetros da linha de transmissão;

O modelo para linha de transmissão implementado na classe *C_TL_FI_Flow_NR* bem como os outros modelos implementados nas próximas classes descritas neste item podem ser também aproveitados na implementação de outras metodologias para o cálculo de Fluxo de Potência Não-Linear. Isto porque a maioria destas metodologias seguem a formulação básica descrita no item 3.2 deste capítulo.

3.3.1.3 Classe *C_Trafo_FI_Flow_NR*

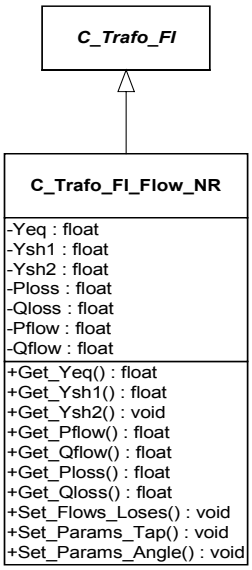

Figura 3.9 – Interface Funcional *C_Trafo_FI_Flow_NR*

- ✓ *Yeq*: representa a admitância série equivalente do circuito;
- ✓ *Ysh1:* representa a admitância em derivação, acoplada a barra 1 da linha;
- ✓ *Ysh2*: representa a admitância em derivação, acoplada a barra 2 da linha;
- ✓ *Set_Params_Tap():* configura os parâmetros do transformador em fase;
- ✓ *Set_Params_Angle():* configura os parâmetros do transformador defasador.

A classe *C_Trafo_FI_Flow_NR* tem uma modelagem semelhante à classe *C_TL_FI_Flow_NR*, já que ambas correspondem a elementos série do sistema. No entanto a classe *C_Trafo_FI_Flow_NR* tem alguns métodos a mais para o tratamento de taps e defasagem angulares presentes no transformador. O método *Set_Params_Tap()*

calcula os parâmetros do transformador em fase com base no modelo π para transformadores, quando há defasagem angular. O método *Set_Params_Angle()* calcula os parâmetros do transformador levando em consideração a defasagem angular do transformador. Os principais atributos e métodos e que não estão presentes na classe *C_TL_FI_Flow_NR,* são apresentados na Figura 3.9.

3.3.1.4 Classe *C_Load_FI_Flow_NR*

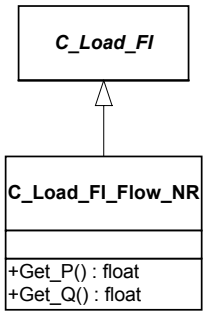

Figura 3.10 – Interface Funcional classe *C_Load_FI_Flow_NR*

- ✓ *Get_P():* retorna o valor da potência ativa da carga;
- ✓ *Get_Q()*: retorna o valor da potência reativa da carga.

Na classe *C_Load_FI_Flow_NR* implementou-se somente o modelo de carga por injeção de potência. Neste tipo de modelagem um objeto do tipo *Load_FI_Flow_NR* deve basicamente armazenar a sua potência ativa e reativa e retorná-la quando solicitado. A incorporação de outros modelos pode ser facilmente realizado pela introdução de métodos e atributos específicos. A classe *C_Load_FI_Flow_NR* com seus atributos e métodos é apresentada na Figura 3.10.

3.3.1.5 Classe *C_Gen_Unit_FI_Flow_NR*

A classe *C_Gen_Unit_FI_Flow_NR* representa o gerador como uma injeção de potência. Aqui valem os mesmas observações realizadas para as cargas. A diferença básica está na necessidade de representação e tratamento dos limites de potências reativas das barras PV. A classe *C_Gen_Unit_FI_Flow_NR* com seus atributos e métodos é apresentada na Figura 3.11.

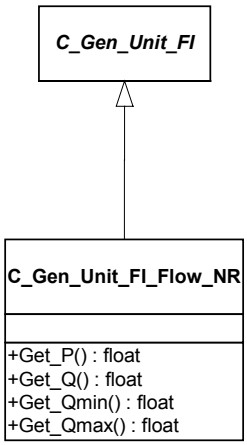

Figura 3.11 - Interface Funcional classe *C_Gen_Unit_FI_Flow_NR*

- ✓ *Get_P()*: retorna o valor da potência ativa injetada pelo gerador;
- ✓ *Get_Q()*: retorna o valor da potência reativa injetada pelo gerador;
- ✓ *Get_Qmin()*: retorna potência reativa máxima que pode ser absorvida pelo gerador;
- ✓ *Get_Qmax()*: retorna potência reativa máxima que pode ser fornecida pelo gerador;

3.3.1.6 Classe *C_R_Compensator_FI_Flow_NR*

A classe *C_R_Compensator_FI_Flow_NR* representa o comportamento dos compensadores reativos (bancos de capacitores e indutores), conectados em derivação nas barras do SEE. A classe *C_R_Compensator_FI_Flow_NR* com seus atributos e métodos é apresentada na Figura 3.12.

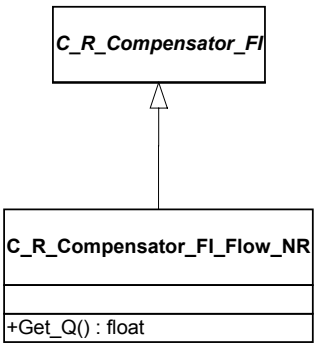

Figura 3.12 – Interface Funcional classe *C_R_Compensator_FI_Flow_NR*

- ✓ *Get_Q()*: retorna o valor da potência reativa do compensador. O valor será positivo se o compensador é um banco de capacitores e negativo caso seja um reator.

3.4 Modelagem para o Método Desacoplado Rápido

Uma característica inerente a qualquer sistema de potência é a forte dependência entre os fluxos de potência ativa e os ângulos de tensões nas barras, e entre os fluxos de potência reativa e a magnitude de tensões nas barras. De uma forma geral pode ser observado que uma variação de potência ativa implica em uma variação semelhante no ângulo, e tem pequeno efeito sobre a magnitude da tensão. Da mesma forma, uma variação na potência reativa implica em uma variação semelhante no modulo da tensão e tem pequeno efeito sobre a magnitude angular. Considerando o sistema linear Jacobiano do método Newton-Raphson, estas aproximações podem ser representadas pelas seguintes expressões:

$$\Delta\vec{P} = H\Delta\vec{\theta} \qquad (2.5)$$

$$\Delta\vec{Q} = L\Delta\vec{V}$$

onde:

$\Delta\vec{P}$: vetor de desbalanços de potência ativa;

H: submatriz H (Jacobiano);

$\Delta\vec{\theta}$: vetor de correção angular;

$\Delta\vec{Q}$: vetor de desbalanços de potência reativa;

L: submatriz L (Jacobiano);

$\Delta\vec{V}$: vetor de correção do módulo da tensão.

O desacoplamento fundamentado nas suposições acima descritas, anteriormente, possibilita a montagem de dois subproblemas distintos que são resolvidos alternadamente, utilizando-se valores atualizados de ângulos e tensões a cada iteração. O processo inicia pela resolução do subproblema Pθ utilizando os valores atualizados de tensão. O resultado deste subproblema é o vetor de correção angular Δθ. Após a atualização dos ângulos, os mesmo são utilizados como dados de entrada no cálculo do subproblema QV, que então fornece as tensões que serão novamente utilizadas no subproblema Pθ. Ao final de cada subproblema um teste de convergência é realizado, verificando-se os desbalanços de potência. Como os subproblemas tem processos de

convergência independentes, há casos onde a velocidade de convergência é maior em um determinado subproblema.

3.4.1 Método Desacoplado Rápido

No método desacoplado rápido consideram-se as seguintes simplificações adicionais na formação da matriz Jacobiana (STOTT, 1974):

- $\cos\theta_{km} \cong 1$;
- $B_{km} >> G_{km}\,\mathrm{sen}\,\theta_{km}$;
- $B_{kk}V_k^2 >> Q_k$;
- consideram-se em todas as barras, tensões iguais a 1 pu.

Com estas simplificações as matrizes H e L tornam-se constantes e são denominadas, B'e B'', respectivamente. Estas duas submatrizes têm estruturas e dimensões diferentes. A matriz B' tem dimensão igual ao número de barras do sistema menos a barra de referência. Esta matriz é a mesma utilizada na resolução do fluxo de potência linearizado. A matriz B'' tem dimensão igual ao número de barras PQ do sistema.

As matrizes B'e B'' são utilizadas no método desacoplado rápido durante a resolução dos subproblemas Pθ e QV, e são calculadas uma única vez durante a resolução do problema. Esta simplificação altera o processo de convergência do método, exigindo um maior número de iterações durante o processo de convergência. As equações dos subproblemas são representadas algebricamente como segue:

$$\begin{aligned} \Delta\vec{P}/V &= B'\Delta\vec{\theta} \\ \Delta\vec{Q}/V &= B''\Delta\vec{V} \end{aligned} \tag{2.6}$$

onde:

$$\begin{aligned} B'_{km} &= -x_{km}^{-1} \\ B'_{kk} &= \sum_{m}^{NB} x_{km}^{-1} \\ B''_{km} &= -B_{km} \\ B''_{kk} &= -B_{kk} \end{aligned} \tag{2.7}$$

O diagrama de atividades do Fluxo de Potência Desacoplado Rápido é mais complexo que o diagrama de atividades do Fluxo de Potência Newton-Raphson. Isto se deve basicamente ao desacoplamento dos problemas a serem resolvidos Pθ-QV. Apresenta-se o diagrama de atividades para esta metodologia na Figura 3.13.

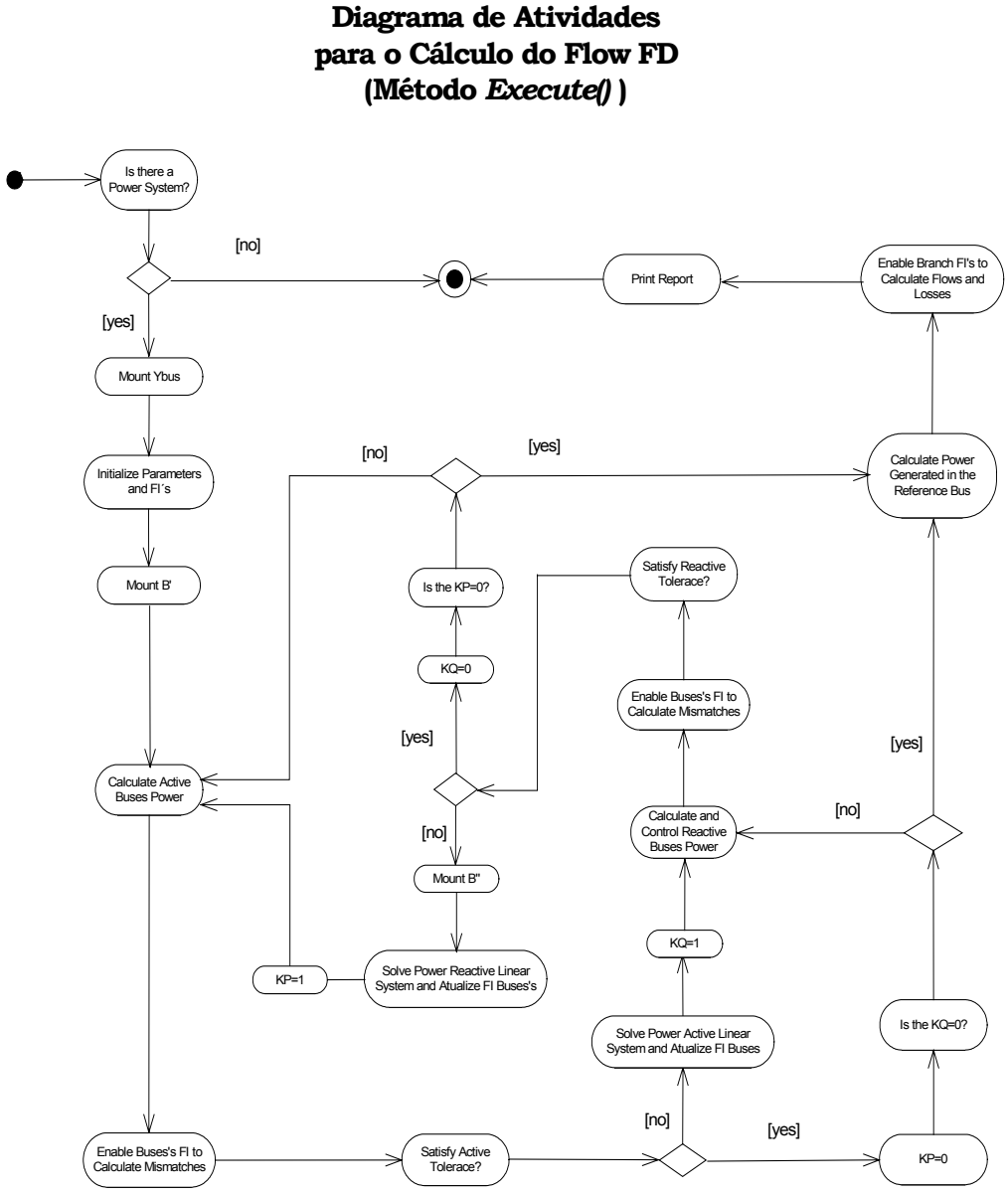

Figura 3.13 - Diagrama de Atividades do Fluxo de Potência Desacoplado Rápido

Como a formulação básica usada pelo método Desacoplado Rápido é a mesma utilizada pelo método de Newton-Raphson, pois ambos resolvem um problema de fluxo de potência não-linear, não há diferenças em termos de estrutura de classes entre as duas metodologias. A diferença básica está nos métodos (em termos de classes) e passos utilizados para a resolução do problema. Estas diferenças, em termos de métodos e

passos, podem ser facilmente verificadas comparando-se os diagramas de atividades das duas metodologias. Para uma correta separação entre as metodologias implementou-se um outro conjunto de classes para a resolução do cálculo de fluxo de potência pelo método Desacoplado Rápido. Encontram-se na classe *C_Flow_FD* alguns atributos e métodos semelhantes ao da classe *C_Flow_NR*. A estrutura da classe com os principais atributos e métodos é apresentada na Figura 3.14.

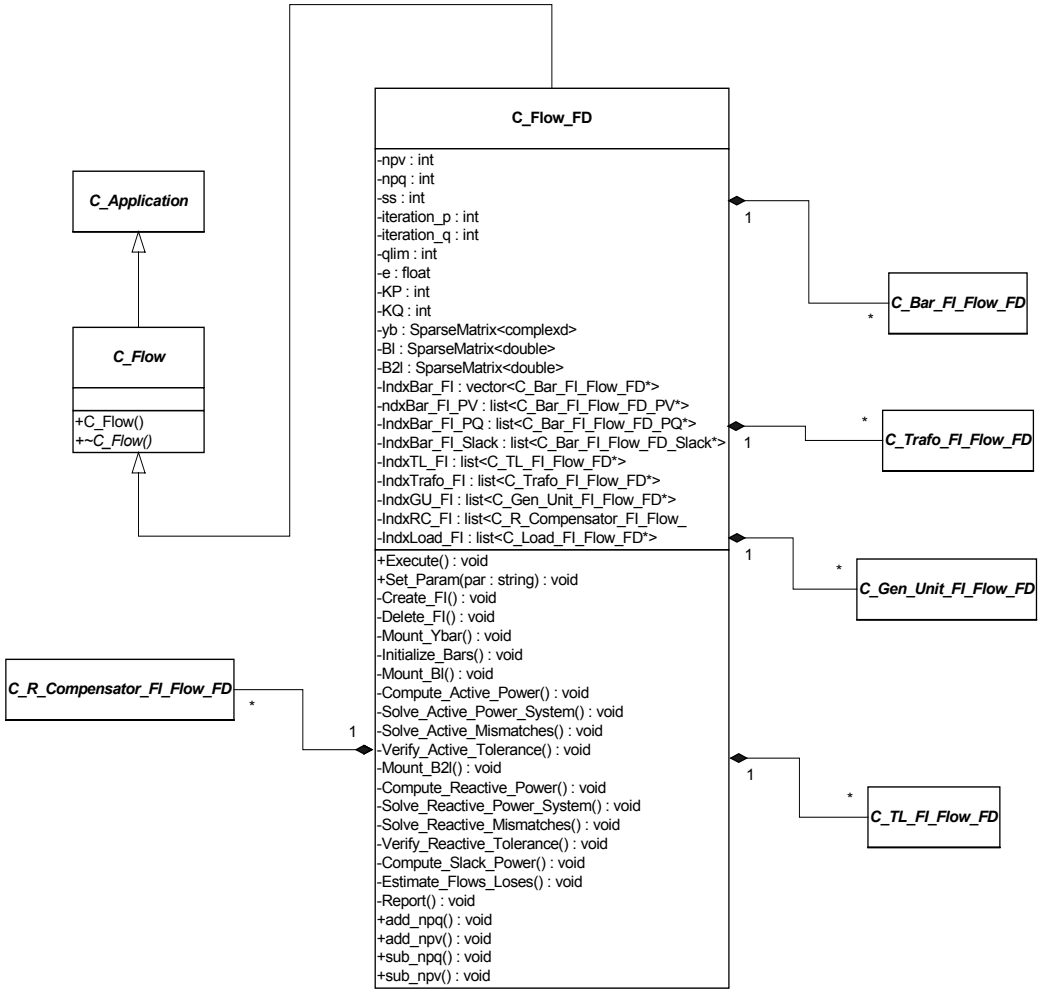

Figura 3.14 – Classe *C_Flow_FD*

- ✓ *KP*: indica se o subproblema Pθ convergiu;
- ✓ *KQ*: indica se o subproblema QV convergiu;
- ✓ *Bl*: matriz B';
- ✓ *B2l*: matriz B'';
- ✓ *IndxBar_FI*: vetor de interfaces funcionais de barras;
- ✓ *IndxBar_FI_Slack*: lista de interfaces funcionais para as barras de referência;

- ✓ *IndxBar_FI_PV*: lista de interfaces funcionais para as barras PV;
- ✓ *IndxBar_FI_PQ*: lista de interfaces funcionais para as barras PQ;
- ✓ *IndxTL_FI*: lista de interfaces funcionais das linhas de transmissão;
- ✓ *IndxTrafo_FI:* lista de interfaces funcionais dos transformadores;
- ✓ *IndxGU_FI:* lista de interfaces funcionais dos geradores;
- ✓ *IndxRC_FI:* lista de interfaces funcionais dos compensadores reativos;
- ✓ *Indx_Load:* lista de interfaces funcionais das cargas;
- ✓ *Mount_Bl()*: monta matriz B';
- ✓ *Solve_Active_Power_System()*: resolve o sistema linear do subproblema Pθ;
- ✓ *Solve_Active_Mismatches():* aciona as Interfaces Funcionais das barras PQ e PV para que as mesmas calculem os desbalanços de potências (subproblema Pθ);
- ✓ *Verify_Active_Tolerance()*: verifica se a tolerância do subproblema Pθ está sendo atendida.
- ✓ *Mount_B2l():* Monta a matriz B'';
- ✓ *Solve_Rective_Power_System()*: vesolve o sistema linear do subproblema QV;
- ✓ *Solve_Reactive_Mismatches():* ativa as Interfaces Funcionais das barras PQ e PV para que as mesmas calculem os desbalanços de potências (subproblema QV);
- ✓ *Verify_Rective_Tolerance()*: verifica se a tolerância reativa do subproblema QV esta sendo atendida.

Nesta metodologia também foi implementado o controle de tensão em barras PV, que é semelhante ao descrito para o Fluxo de Potência Newton Rapshon. A única diferença aqui é que ao invés de alterar-se a matriz Jacobiana inserindo ou retirando linhas ou colunas das barras em transição (PV para PQ o de PQ para PV), estas operações são realizadas somente na matriz B''.

3.4.2 Interfaces Funcionais – Método Desacoplado Rápido

O conjunto de Interfaces Funcionais definidos para o Fluxo de Potência é Desacoplado Rápido é basicamente o mesmo que o definido para o método de Newton-Raphson. A diferença básica esta nos métodos que compõem as barras, para que seja possível realizar a montagem das matrizes B' e B'' e o controle de tensão nas barras PV.

3.4.2.1 Classe *C_Bar_FI_Flow_FD*:

A classe *C_Bar_FI_Flow_FD* é uma classe abstrata que modela o comportamento do objeto físico barra no fluxo de potência Desacoplado Rápido. Esta classe providencia a base para a criação de outras classes barras que não representam entidades físicas e sim conceituais. Da mesma forma que no Fluxo de Potência Newton-Raphson, são definidas aqui as três classes concretas que representam as barras do

sistema em um Fluxo de Potência: *C_Bar_FI_Flow_FD_PV, C_Bar_FI_Flow_FD_PQ* e *C_Bar_FI_Flow_FD_Slack.* A estrutura de classes barra é apresentada abaixo.

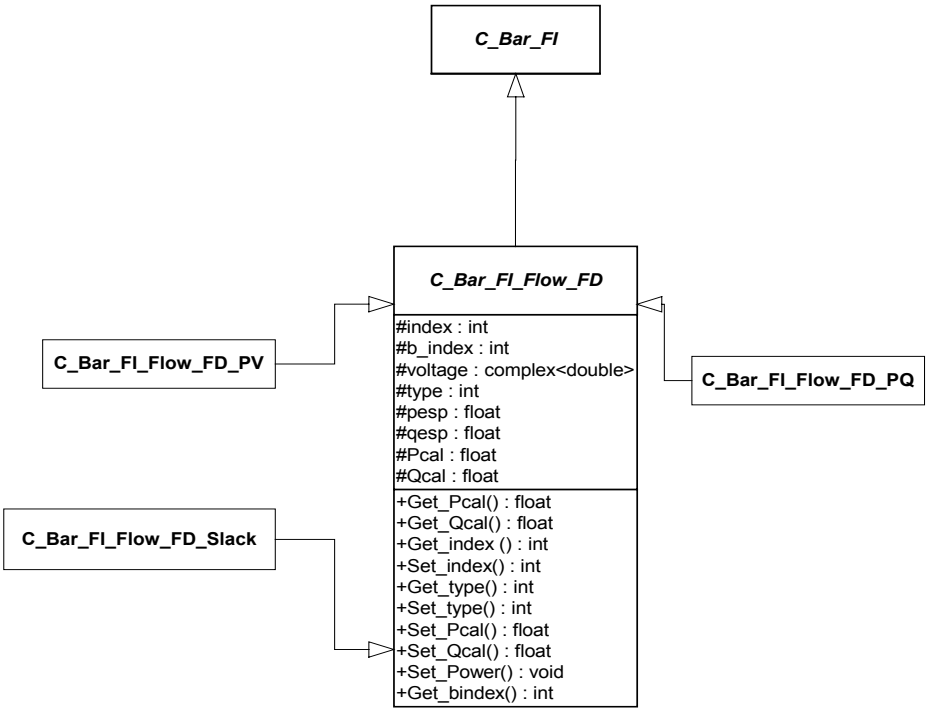

Figura 3.15 – Interface Funcional *classe C_Bar_FI_Flow_FD*

Os principais atributos e métodos utilizados na classe *C_Bar_FI_Flow_FD* são:

- ✓ *index:* índice da barra e de localização na matriz de admitâncias nodais;
- ✓ *voltage*: tensão na barra;
- ✓ *type*: tipo da barra (PV, PQ ou FOLGA);
- ✓ *pesp*: potência ativa total injetada na barra;
- ✓ *qesp*: potência reativa total injetada na barra;
- ✓ *Pcal*: potência ativa calculada a cada iteração;
- ✓ *Qcal*: potência reativa calculada a cada iteração;
- ✓ *b_index*: índice de localização da barra na matrizes B';
- ✓ *Get_Pcal()*: retorna a potência ativa calculada na iteração;
- ✓ *Get_Qcal()*: retorna a potência reativa calculada na iteração;
- ✓ *Set_Pcal()*: altera o valor de *Pcal*;
- ✓ *Set_Qcal()*: altera o valor de *Qcal*;
- ✓ *Get_index()*: retorna o índice da barra;
- ✓ *Set_index()*: altera o valor do atributo *index*;
- ✓ *Get_type()*: retorna o tipo da barra;
- ✓ *Set_type()*: altera o atributo *type*;

- ✓ *Set_Power()*: calcula a injeção de potência total em cada barra;
- ✓ *Get_bindex()*: retorna índice de posição da matriz B';
- ✓ *Set_bindex()*: configura o índice de posição na B';

É relevante, citar aqui a *Interface Funcional* das barras PV, para fins de comparação com a *Interface Funcional* das barras PV desenvolvida para o método Newton-Raphson. O seu diagrama de classe está mostrado na Figura 3.16.

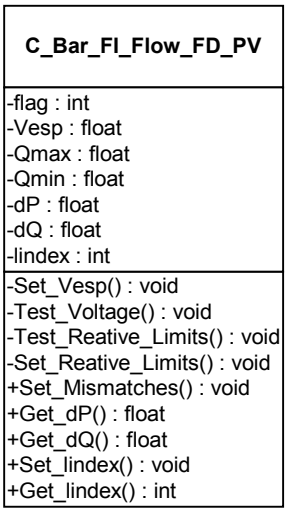

Figura 3.16 – Interface Funcional *C_Bar_FI_Flow_FD_PV*

Os principais atributos e métodos da classe são:

- ✓ *flag*: sinaliza se a barra mudou o seu tipo para PQ durante o processo iterativo;
- ✓ *Vesp*: valor especificado da tensão;
- ✓ *lindex*: posição da barra na matriz B''.
- ✓ *Qmax*: valor da máxima potência reativa que pode ser gerada;
- ✓ *Qmin*: valor da máxima potência reativa que pode ser absorvida;
- ✓ *dP*: valor do desbalanço de potência ativa;
- ✓ *dQ*: valor do desbalanço de potência reativa;
- ✓ *Get_Pd():* retorna potência ativa demandada na barra;
- ✓ *Set_Pd():* altera valor da potência ativa demandada na barra;
- ✓ *Get_Qd()*: retorna potência reativa demandada na barra;
- ✓ *Set_Qd():* altera o valor da potência reativa demandada na barra;
- ✓ *Set_Reactive_Limits()*: configura o limite de reativo máximo e mínimo na barra PV;
- ✓ *Test_Reactive_Limits()*: verifica se os limites de reativo, máximo e mínimo, estão sendo respeitados;
- ✓ *Test_Voltage()*: verifica se a tensão em uma barra PV que foi modificada para PQ, pode retornar a sua condição original.

- ✓ *Set_Mismatches()*: calcula a diferença entre o valor calculado e o especificado de potência ativa e reativa na barra;
- ✓ *Set_lindex()*: retorna o indicador de posição da barra na matriz B'';
- ✓ *Get_lindex()*: configura o indicador de posição da barra na matriz B''.

As outras interfaces funcionais de barras não serão apresentadas aqui pois as mesmas não apresentam diferenças relevantes em relação às implementadas para a metodologia de Fluxo de Potência Netwon-Raphson. Elas contém os métodos descritos nesta interface menos os métodos relativos ao controle de limite de potência reativa na barra. A diferenças fundamentais estão nos atributos e métodos que guardam as posições das barras nas matrizes B' e B''. As demais *Interfaces Funcionais* (Carga, Gerador, Compensador, Linha de Transmissão e Transformador) também não serão citadas pois não apresentam diferenças às apresentadas no Fluxo de Potência Newton Raphson.

3.5 Validação das Aplicações - Fluxo de Potência Não Linear

Diversos SEE representando equivalentes do sistema interligado nacional (SIN) foram analisados utilizando-se as aplicações para cálculo de fluxo de potência não-linear implementadas neste trabalho. Seus resultados numéricos (ângulos, tensões e perdas) foram comparados com resultados do programa ANAREDE (CEPEL, 1999), software tradicional no setor elétrico brasileiro. Os valores obtidos foram os mesmos, desde sistemas de menor porte até sistemas mais complexos, validando as aplicaçòes. Alguns dos sistemas testes utilizados são apresentados abaixo:

- Sis45: Equivalente de uma configuração do sistema Sul brasileiro;
- Sis730: Configuração equivalente do sistema interligado das regiões Sul e Sudeste brasileiras para o ano de 1987;
- Sis1916: Configuração do sistema interligado das regiões sul e sudeste brasileiras.

Para comparações em termos de desempenho computacional, utilizou-se como plataforma para as simulações, um microcomputador com processador AMD Athlon 1GHz, executando o sistema operacional Windows 2000. O compilador utilizado foi o

Microsoft Visual C++ 6.0, com as opções padrões de otimização de códigos. Os sistemas de menor porte (até 730 barras) exigem pouco esforço computacional, e não permitem comparações precisas. Conseqüentemente não foi possível a realização de estudos detalhados para estes sistemas. No caso do Sis1916 o tempo computacional é um pouco mais elevado o que possibilita a comparação entre os programas. A execução do *framework* OOTPS para a aplicação de fluxo de potência não-linear, para o Sis1916, consome um tempo aproximadamente 40% superior ao programa ANAREDE. Esta diferença assemelha-se ao apresentado em AGOSTINI (2002a), qualificando os desenvolvimentos realizados neste trabalho (apresentou-se naquele trabalho o valor de 37%). Como descrito em AGOSTINI (2002a) a diferença em termos de tempo de processamento não é necessariamente uma limitação da MOO em si, ou de linguagens de programação com suporte a MOO (tais como a linguagem C++), mas sim uma característica de projetos de softwares (e por conseqüência, de seus códigos) baseados nas estruturas de dados dos sistemas. Nesses, a execução de determinadas tarefas tende a envolver o processamento de diversas outras tarefas menores, que não são necessárias naqueles softwares. As trocas de mensagens entre objetos, no caso da MOO, é imprescindível para se manter a eficiência de uma estrutura de classes, em termos de aspectos como mantenabilidade, reusabilidade, clareza de projeto, etc.

CAPÍTULO 4

MODELAGEM DE APLICAÇÃO: AVALIAÇÃO E MELHORIA DA SEGURANÇA DINÂMICA DE SISTEMAS DE ENERGIA ELÉTRICA

4.1 Introdução

Neste capítulo descreve-se a modelagem orientada a objetos de uma ferramenta computacional para Avaliação e Melhoria da Segurança Dinâmica de SEE (ASDIN), bem como a sua implementação na Base Computacional descrita no Capítulo 2. Especificamente, foi modelada e implementada a metodologia descrita em SOUZA (1999), cujo trabalho segue o paradigma da programação funcional usando Fortran 77. Esta metodologia é agora modelada segundo a filosofia de desenvolvimento de softwares para SEE, proposta por AGOSTINI (2002a).

O presente capítulo inicia com uma descrição sintética dos módulos que compõem a metodologia seguindo a proposição de SOUZA (1999). Na seqüência é apresentado todo o estudo realizado para a implementação do ASDIN com a apresentação de Diagramas de Casos de Uso, Diagramas de Classes e de Atividades. As diversas metodologias, alterações em metodologia existentes e estruturas de dados identificadas com os dois primeiros diagramas são comentadas e justificadas nos itens que seguem a discussão.

4.2 Segurança da Operação de SEE

A operação segura de sistemas de energia elétrica interconectados é uma tarefa desafiante devido a sua natureza dinâmica intrínseca, dado que carga, geração, topologia e parâmetros de operação estão sujeitos a constantes variações. Além da influência destas características naturais do sistema físico, a reestruturação da indústria de energia elétrica bem como o nível crescente de exigências em relação ao atendimento dos requisitos de qualidade e segurança do sistema introduzem uma maior complexidade na operação do sistema (SOUZA et al. 2002).

Dentre os instrumentos de apoio à operação, a função avaliação da *segurança on-line* da segurança dinâmica é reconhecidamente uma das mais necessárias. Esta ferramenta pode assegurar a confiabilidade operacional do sistema e auxiliar na identificação de situações potencialmente críticas em relação a segurança. Contudo, diferentemente da análise da segurança estática, a análise dinâmica, reconhecidamente mais complexa e onerosa, respectivamente, em termos metodológicos e computacionais, não se encontra completamente desenvolvida.

Na busca de novas metodologias para suprir as necessidades acima, inúmeros trabalhos de pesquisa têm sido desenvolvidos e os maiores desafios encontrados estão no desenvolvimento de metodologias rápidas e automáticas de avaliação da estabilidade transitória com modelos detalhados, na definição de índices confiáveis, no desenvolvimento de metodologias que definam efetivas ações de controle para a melhoria da segurança, quando necessários; e na realização destes estudos em tempos compatíveis com as necessidades dos EMS (KUNDUR, 1998, DEMAREE et al., 1994, SOUZA et al., 2000).

Procurando vencer estes desafios, SOUZA (1999) abordam o problema da avaliação e segurança dinâmica na sua totalidade, considerando a avaliação global da estabilidade transitória via métodos rápidos, a avaliação com modelos detalhados das contingências críticas e a definição de ações de melhoria da segurança, sempre que necessário. A metodologia apresentada por SOUZA (1999) é descrita no próximo item.

4.2.1 Metodologia Proposta por SOUZA (1999)

A metodologia e o protótipo computacional propostos por SOUZA (1999) são ilustrados pela Figura 4.1. Uma descrição sintética de cada módulo é apresentada na seqüência.

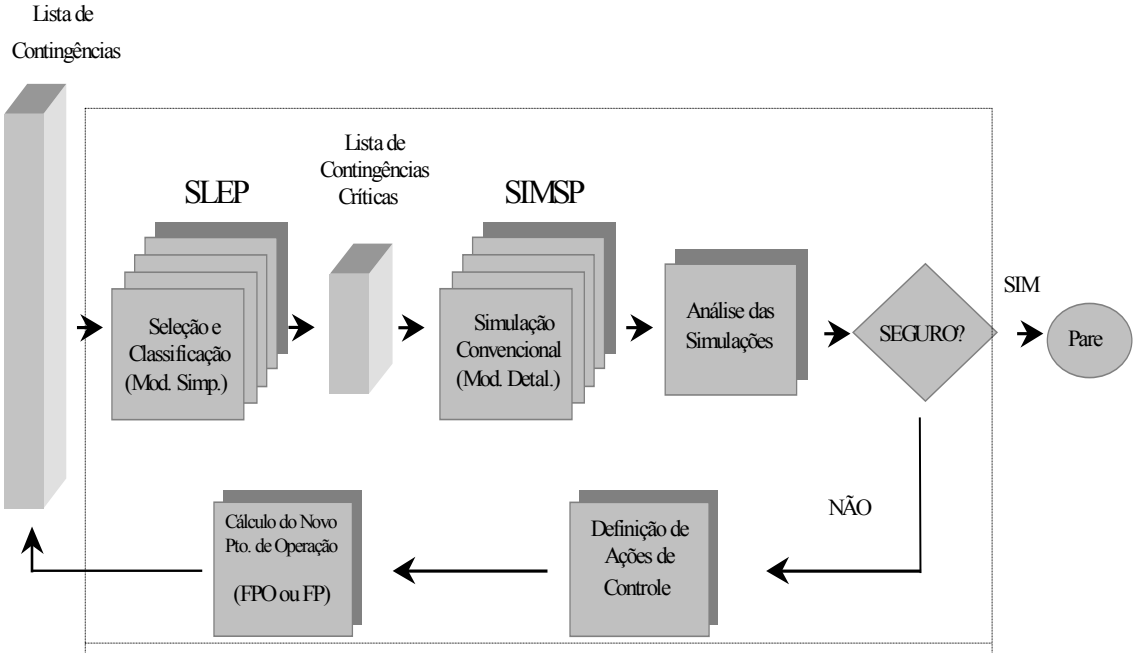

Figura 4.1 – Representação da Metodologia de Avaliação e Melhoria da Segurança Dinâmica – Digrama Esquemático

- **Módulo de seleção e classificação de contingências críticas**: a partir de um conjunto de contingências pré-definidas, selecionam-se apenas as mais críticas, que podem por em risco a segurança do sistema, as quais deverão ser estudadas detalhadamente na etapa posterior. Para esta tarefa pode-se utilizar métodos rápidos, geralmente com modelagem simplificada, para selecionar as contingências críticas, e classificá-las segundo seu grau de severidade. Desta forma, procura-se reduzir o tempo computacional, analisando-se detalhadamente somente os casos mais críticos. O método SLEP iterativo (Fonseca e Decker, 1985) é um dos dois métodos rápidos que apresentaram os melhores resultados em termos de confiabilidade e precisão segundo a avaliação da Força Tarefa da CIGRE (CIGRE, 1992). Por este motivo este método foi adotado na implementação do protótipo em SOUZA (1999);
- **Módulo de simulação no domínio do tempo**: as contingências classificadas como críticas são novamente avaliadas através da simulação no

domínio do tempo com modelagem dinâmica detalhada. A solução do conjunto de equações algébrico-diferenciais não-lineares que descrevem a dinâmica do sistema é realizada pelo Método Alternado Entrelaçado Implícito, através do programa SIMSP. Este programa foi desenvolvido na UFSC, utilizando MOO e C++ (MANZONI, 1996);

- **Módulo de avaliação automática da estabilidade transitória:** nesta etapa da metodologia foi desenvolvido uma técnica de análise automática dos resultados da simulação, baseada na definição de Energia Potencial Generalizada (LA SCALA et al., 1996), classificando as contingências em estáveis ou instáveis. A principal motivação na escolha do método da EPG foi a necessidade do desenvolvimento de um teste capaz de confiavelmente classificar se uma trajetória é instável ou estável, quando completamente conhecida em um período de tempo crítico de simulação da dinâmica para fins de avaliação da estabilidade transitória (de 5 a 8 segundos). Posteriormente, são obtidas as margens em energia das contingências classificadas como instáveis, através da utilização de uma técnica baseada no método SIME (ZHANG et al., 1997). A escolha do método SIME deve-se às facilidades de implementação e computação no cálculo das margens instáveis, e ao curto período de simulação requerido para a execução desta tarefa;

- **Módulo de melhoria da segurança dinâmica:** se o sistema for inseguro, para uma ou mais contingências, são definidas ações de controle preventivo do tipo redespacho de potência ativa, empregando-se o Método da Direção S Modificado (SOUZA, 1999), para modelos detalhados, devido às suas facilidades de implementação;

- **Módulo de fluxo de potência:** com os redespachos propostos na etapa anterior para melhoria da segurança do sistema, faz-se necessário a determinação do novo ponto de operação através de um programa de fluxo de potência. Em SOUZA (1999) esta etapa é implementada utilizando-se as rotinas básicas de cálculo de fluxo de potência do programa ANAREDE (CEPEL, 1999).

Para a validação da metodologia proposta, SOUZA (1999) implementaram um protótipo computacional que incorpora todos os módulos mencionados no item anterior.

Este protótipo foi na época desenvolvido usando técnicas de processamento paralelo para que a restrição do tempo computacional fosse atendida.

Alguns dos sistemas testes utilizados naquele trabalho são apresentados abaixo e foram assim denominados:

- Sis45: Equivalente de uma configuração do sistema Sul brasileiro;
- Sis730: Configuração equivalente do sistema interligado das regiões Sul e Sudeste brasileiras para o ano de 1987;
- Sis1916: Configuração do sistema interligado das regiões sul e sudeste brasileiras.

Um detalhamento destes sistemas é apresentado na Tabela 4.1:

Denominação dos SEE	Nº de Barras	Nº de Linhas	Nº de Geradores	Nº de Contingências
Sis45	45	72	10	124
Sis730	730	1146	82	1846
Sis1916	1916	2788	98	4113

Tabela 4-1 – Parâmetros dos Sistemas Testes Utilizados nas Simulações

Várias simulações foram realizadas em SOUZA (1999) utilizando-se o protótipo paralelo desenvolvido e observou-se que o método atingiu o objetivo proposto, melhorando o ponto de operação inicial dos sistemas de tal forma a garantir a estabilidade transitória para todas as contingências em estudo. Maiores informações em termos de implementação incluindo diagramas e a análise completa das simulações utilizando o protótipo computacional estão descritos em SOUZA (1999).

4.3 Modelagem do ASDIN seguindo o paradigma da BCOO

Este item trata especificamente do processo de modelagem e implementação de uma ferramenta computacional de Avaliação da Segurança Dinâmica (ASDIN), baseada

na metodologia proposta por SOUZA (1999), seguindo a nova filosofia de desenvolvimento de software para SEE descrita em AGOSTINI (2002a). Os passos principais na modelagem e implementação desta ferramenta são apresentados e comentados como segue:

- ***Análise dos Casos de Uso***

Nesta fase as principais metodologias e o escopo das mesmas no âmbito da ferramenta computacional são identificadas. Verificando-se os recursos disponibilizados pela BCOO em termos de estruturas de classes e aplicações já implementadas identifica-se o que pode ser reaproveitado, o que é necessário readequar e as novas estruturas e metodologias que necessitam ser modeladas e implementadas. Após esta delimitação realiza-se a análise detalhada do que foi proposto e, logo em seguida, documenta-se as funcionalidades de cada Caso de Uso identificado.

- ***Concepção dos Diagramas de Classes***

Com a definição do escopo de cada aplicação e o esboço das relações entre as mesmas realiza-se a concepção da estrutura de classes da ferramenta. Os elementos de dados são identificados baseados no relato do diagrama de Caso de Uso. Os relacionamentos entre classes são melhor definidos e documentados. As similaridades e diferenças entre a estrutura de classes concebida e o diagrama de Caso de Uso é analisada . A análise confrontando a estrutura de classes com o proposto no diagrama de Casos de Uso propiciam uma sistemática favorável a otimização destes diagramas.

- ***Desenvolvimento do Diagrama de Atividades***

Com os diagramas de Casos de Uso e de Classes consolidados, implementa-se um Diagrama de Atividades para a simulação do fluxo de trabalho entre as classes que compõem a aplicação. A implementação do Diagrama de Atividades é importante para a identificação e correção de possíveis problemas na estrutura de classes. Similaridades e diferenças entre os diagramas de classes e atividades são examinadas auxiliando na otimização dos modelos.

Estes passos foram realizados observando-se os preceitos da nova filosofia de software para SEE tais como delimitação adequada do escopo de projetos, robustez das estruturas de dados e reutilização de códigos. Com relação ao último preceito realizou-

se o reaproveitamento de código tanto em termos de utilização das classes e aplicações existentes no âmbito da BCOO quanto em termos de aplicações consolidadas e que não faziam parte do escopo da mesma. O detalhamento de cada uma destas fases é apresentado nos próximos itens. Após a conclusão destes passos, o desenvolvedor está habilitado a migrar da fase de concepção para a fase de implementação propriamente dita.

4.3.1 Casos de Uso

Quando se projeta um sistema espera-se que o mesmo comporte-se de acordo com as diretrizes pré-especificadas. Um caso de uso especifica o comportamento de um sistema ou parte de um sistema e é uma descrição de um conjunto de seqüências de ações, incluindo variantes realizadas pelo sistema para produzir um resultado apreciável (BOOCH et al., 2000).

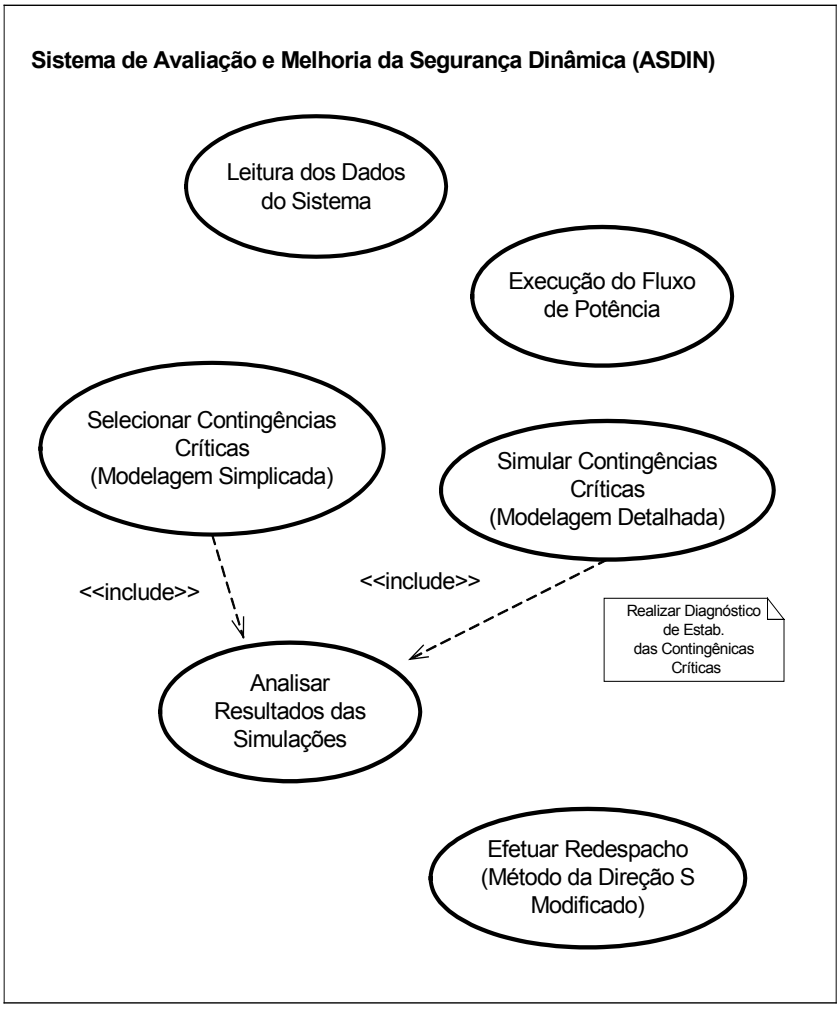

Figura 4.2 - Diagrama de Casos de Uso para a ferramenta ASDIN

Este diagrama auxilia muito no entendimento do que se está procurando fazer e no levantamento dos requisitos necessários para a construção de um determinado sistema, sem ser necessário especificar como este comportamento é implementado. Além disso, os casos de uso servem para ajudar a validar a arquitetura e para verificar o sistema à medida em que a implementação está evoluindo. O diagrama de Casos de Uso inicial para a ferramenta computacional ASDIN está ilustrado na Figura 4.2.

A partir deste Diagrama de Casos de Uso que fornece um panorama geral do funcionamento do ASDIN, inicia-se um processo de identificação das metodologias e estruturas de dados que estarão compondo cada Caso de Uso identificado neste diagrama. Os diagramas de Casos de Uso facilitam a visão do sistema como um todo e a delimitação do que realmente cada módulo deve fazer. O levantamento de requisitos, metodologias e estruturas de dados que compõe cada Caso de Uso ilustrado acima e é o objetivo dos comentários que seguem:

- ***Leitura dos Dados do Sistema:*** a leitura dos dados do sistema encontra-se implementada no âmbito da BCOO. A leitura é realizada utilizando-se os objetos derivados da Classe *C_DB_Manager* contidos na Abstração das Facilidades Computacionais. Nesta fase de leitura dos dados também é prevista a criação dos objetos físicos que compõe o sistema elétrico que será analisado. Maiores informações sobre como é realizado o processo de leitura de dados usando recursos da BCOO podem ser encontrados em AGOSTINI 2002.

- ***Cálculo de Fluxo de Potência:*** Após a leitura dos dados se faz necessário a obtenção do ponto de operação do sistema. A obtenção deste ponto é o resultado da atuação de um objeto de cálculo de fluxo de potência sob a base. No âmbito deste trabalho foram desenvolvidas duas metodologias de cálculo de fluxo de potência de acordo com a BCOO (vide Capítulo 3). Como observado no Capítulo 3 este cálculo é realizado utilizando como base as Interfaces Funcionais desenvolvidas para o programa de fluxo de potência. A única modificação necessária na classe Fluxo de Potência foi a inserção de um método que realize a atualização do sistema físico a partir dos valores das Interfaces Funcionais. Esta atualização se faz necessária para que outras

metodologias que estejam sendo processadas simultaneamente possam acessar os dados atualizados do sistema.

- ***Selecionar Contingências Críticas:*** Esta etapa deverá fazer uma leitura do sistema físico e selecionar as possíveis contingências críticas para o ponto de operação pré calculado. No âmbito da BCOO não há nenhuma metodologia implementada para a realização desta etapa. É facilmente percebido que a metodologia natural para a realização desta tarefa é o método SLEP iterativo, pois o mesmo já havia sido implementado em SOUZA (1999). Será necessário então a criação de uma classe *C_SLEP* para a realização da seleção das contingências críticas. Esta classe usará trechos de códigos em FORTRAN já bem testados e desenvolvidos como métodos da classe. O resultado da aplicação deste método sob o sistema é dado por um conjunto de contingências críticas que contém diversas informações como barra sob curto, linha retirada, tempo crítico, etc. Com esta quantidade significante de dados a serem armazenados e que necessitam ser disponibilizados às outras aplicações que compõe o ASDIN, identifica-se a necessidade da criação de uma estrutura de dados para o armazenamento destas informações.

- ***Analisar Contingências Críticas Detalhadamente:*** Após a fase de seleção uma lista de contingências críticas é disponibilizada para a análise detalhada. Cada uma destas contingências precisa ser simulada detalhadamente. Inserida no âmbito da BCOO encontra-se uma metodologia de simulação dinâmica, baseada no programa SIMSP (MANZONI, 1996) e foi naturalmente escolhida para a realização desta etapa. Somente dois procedimentos devem ser adicionados à classe para que ela possa trabalhar junto com o ASDIN. Os parâmetros de simulação e uma metodologia de diagnóstico automático da instabilidade. SOUZA (1999) implementou uma técnica de detecção automática da instabilidade. Este procedimento de diagnóstico deve ser realizado durante o processo de simulação no domínio do tempo, pois consiste basicamente da monitoração da evolução temporal dos ângulos das máquinas monitoradas. Este diagnóstico é colocado como um método na classe *C_SIMSP* e uma estrutura de dados deve ser criada para o

armazenamento destas informações. Um método para a configuração automática dos eventos a serem simulados, deve também ser implementada nesta classe. A configuração destes parâmetros é atualmente realizada através da leitura do arquivo de controle. Estes parâmetros incluem desde dados da contingência até parâmetros de simulação, como passo de integração e tempo de simulação. Identifica-se então uma outra estrutura de dados que deverá ser criada e que abrigará todas estas informações.

- ***Analisar os Resultados da Simulação Detalhada:*** A função deste passo é calcular as margens em energia das contingências críticas (utilizando-se uma técnica baseada no método SIME) e efetuar a classificação das contingências críticas segundo a sua severidade em termos de margens. A contingência com maior grau de severidade será então encaminhada ao módulo de melhoria para que o redespacho de potência seja efetivado. Esta etapa será implementada como um método da ferramenta computacional ASDIN.

- ***Efetuar Redespacho de Potência Ativa:*** quando necessário será definido um redespacho de potência ativa baseado no método da Direção S Modificado. Como esta etapa está implementada em código FORTRAN basta adequá-la à filosofia da BCOO, usando o conceito de reaproveitamento de código. Observa-se entretanto que há metodologias mais detalhadas que podem ser utilizadas para este fim e descritas em COSTA (1997) e VANTI (2003). Estas metodologias também podem ser implementadas em termos de classes ou como métodos da classe ASDIN. Finalizando neste Caso de Uso executa-se novamente cálculo de fluxo de potência para o sistema.

Do exposto anteriormente fica evidente a necessidade da criação de uma classe que gerencie todos estes módulos e que implemente métodos que correspondam a determinadas atividades. O objeto *ASDIN* será o responsável pela gerência de todo o processo.

4.3.2 Diagrama de Classes

De acordo com as constatações obtidas através da análise do diagrama de Casos De Uso propõe-se o seguinte diagrama de classes para a ferramenta computacional.

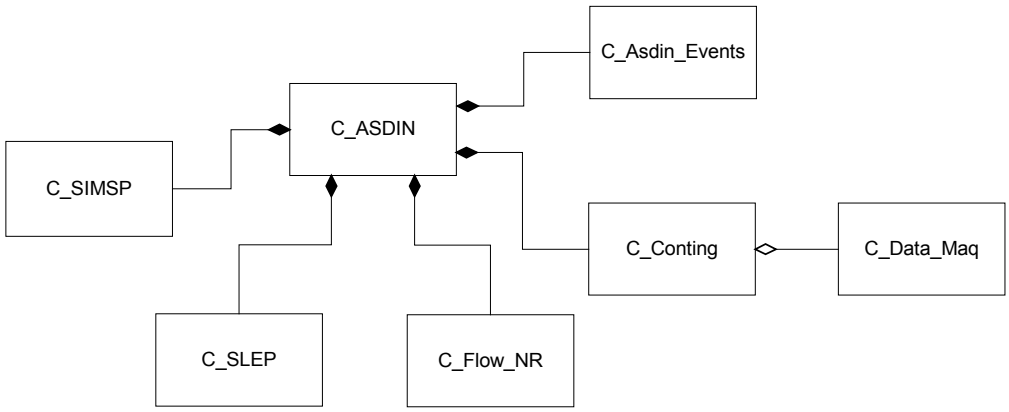

Figura 4.3 – Diagrama de Classes da Ferramenta ASDIN

Neste diagrama observa-se a classe *C_ASDIN* no centro e as diversas metodologias atuando em composição com a mesma para a solução de um determinado problema. As estruturas de dados são representadas pelas classes *C_Conting*, *C_Data_Maq* e *C_Asdin_Events*, criadas para auxiliar na disponibilização, de acordo com o paradigma da MOO, dos diversos dados necessários à execução da ferramenta computacional. Uma descrição aprofundada apresentando os métodos e atributos das classes apresentadas é realizada a seguir.

4.3.2.1 Classe *C_ASDIN*:

A classe *C_ASDIN* gerencia a aplicação de Avaliação e Melhoria da Segurança Dinâmica. Esta classe é apresentada na Figura 4.4.

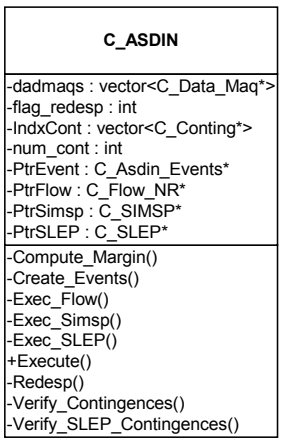

Figura 4.4 – Classe *C_ASDIN*

Os principais atributos e métodos desta classe são:

- ✓ *dadmaqs*: vetor que contém ponteiros para os objetos máquinas monitoradas em cada contingência simulada;
- ✓ *flag_redesp*: variável auxiliar para controle de redespacho;
- ✓ *IndxCont*: vetor que contém ponteiros para as contingências críticas fornecidas pelo SLEP;
- ✓ *num_cont*: variável que contém o número de contingências críticas que estão sendo analisadas;
- ✓ *PtrEvent*: ponteiro para o objeto *C_Event*;
- ✓ *PtrFlow*: ponteiro para o objeto *C_Flow_NR*;
- ✓ *PtrSimsp*: ponteiro para o objeto *C_SIMSP*;
- ✓ *PtrSLEP*: ponteiro para o objeto *C_SLEP*;
- ✓ *Compute_Margin()*: método que realiza o cálculo das margens em energia das contingências classificadas como instáveis;
- ✓ *Create_Events()*: cria o objeto que contém os dados de simulação que serão fornecidos ao programa SIMSP;
- ✓ *Exec_Flow()*: método que realiza a criação e execução do objeto *Flow_NR*;
- ✓ *Exec_SLEP()*: método que realiza a criação e execução do objeto *SLEP*;
- ✓ *Exec_SIMSP()*: método que realiza a criação e execução do objeto *SIMSP*;
- ✓ *Execute()*: método que gerencia toda a execução da aplicação *ASDIN*;
- ✓ *Redesp()*: método do redespacho de potência ativa;
- ✓ *Verify_Contingences()*: realiza a seleção e classificação das contingências oriundas do processo de simulação detalhada;
- ✓ *Verify_SLEP_Contingences()*: realiza a eliminação das contingências duplicadas (circuitos duplos) e com tempo crítico zero (contingências que provocam ilhamento do gerador) fornecido pelo o objeto SLEP.

O objeto *ASDIN* faz o gerenciamento de todo o processo de execução da metodologia, criando e destruindo objetos que trabalham na análise do sistema. Métodos foram implementados para a realização de algumas etapas do processo onde utilizou-se o reaproveitamento de trechos de códigos desenvolvidos em FORTRAN.

4.3.2.2 Classe *C_SLEP*:

O objeto SLEP tem a função de realizar a seleção das contingências críticas do sistema. Para isto utiliza-se o método SLEP iterativo. O método SLEP iterativo encontra-se implementado em um programa totalmente escrito em FORTRAN e é uma metodologia bem desenvolvida e testada. Trechos de códigos deste programa em FORTRAN foram largamente reaproveitados na implementação da classe *C_SLEP*. Esta classe juntamente com os seus atributos e métodos são apresentados na Figura 4.5.

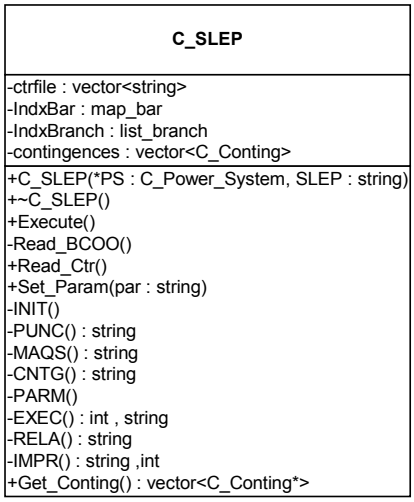

Figura 4.5 – Classe *C_SLEP*

- ✓ *ctrfile:* vetor de *strings* que contém os comandos do arquivo de controle do programa SLEP iterativo (SLEPM.CTR);
- ✓ *contingences:* vetor que contém as contingências críticas que compõe o sistema;
- ✓ *IndxBar*: contém ponteiros para as barras físicas do sistema;
- ✓ *IndxBranch:* contém ponteiros para os elementos série do sistema (linha de transmissão e transformadores);
- ✓ *Execute():* gerencia a execução da metodologia SLEP;
- ✓ *Read_BCOO():* faz a transferência de dados da BCOO para os commons do FORTRAN;
- ✓ *Read_Ctr():* método que realiza a leitura do arquivo de controle do programa SLEP;
- ✓ *INIT():* método em FORTRAN que realiza a inicialização de variáveis de controle do programa SLEP;
- ✓ *PUNC():* método em FORTRAN que realiza a leitura de arquivos de formato estendido;
- ✓ *MAQS:* método em FORTRAN que realiza a leitura do arquivo de máquinas que será usado pelo programa SLEP;
- ✓ *CNTG():* método em FORTRAN que realiza a leitura de arquivos com contingências pré-determinadas. Este método também possibilita a geração automática de contingências;
- ✓ *PARM():* método em FORTRAN que realiza a alteração das variáveis do programa responsáveis pelo controle do processamento;
- ✓ *EXEC():* método em FORTRAN que realiza a execução do SLEP iterativo;
- ✓ *RELA():* realiza a configuração dos parâmetros para a geração do relatório;
- ✓ *IMPR():* realiza a impressão de relatório;

Como pode ser observado vários métodos foram implementados utilizando-se códigos FORTRAN retirados do protótipo desenvolvido em SOUZA (1999). A utilização destes trechos de código na construção desta classe representou uma grande economia em termos de tempo de desenvolvimento, evitando o trabalho de reescrita do código e da necessidade de testes exaustivos de operação do mesmo. A reutilização de

códigos é bem vista na área de desenvolvimento de softwares para SEEs, pois há uma grande quantidade de programas já desenvolvidos e largamente utilizados no setor fora do paradigma da orientação a objetos, e quando apropriado, podem ser utilizados como parte de desenvolvimentos futuros.

4.3.2.3 Classe *C_SIMSP*

A classe *C_SIMSP* encontra-se no âmbito da BCOO, completamente modelada e implementada sob a nova filosofia de softwares para SEE, diferentemente da implementação do programa SIMSP utilizada em SOUZA (1999). A estrutura de classes da aplicação SIMSP, modelada e implementa em AGOSTINI (2002a) e utilizada neste trabalho, é apresentada na Figura 4.6:

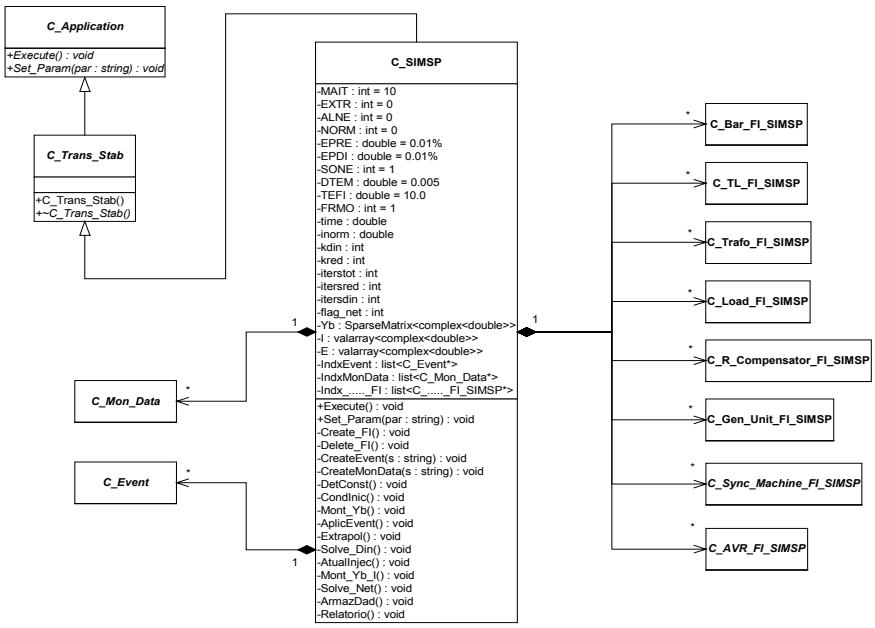

Figura 4.6 – Estrutura de Classe da Aplicação SIMSP

A classe *C_SIMSP* é formada por diversas outras classes, tais como dados de monitoração e eventos a serem simulados, e as interfaces funcionais dos elementos físicos. Maiores detalhes desta aplicação são descritos em AGOSTINI (2002a).

No presente trabalho necessitou-se implementar dois novos métodos na classe *C_SIMSP* são eles: Diagnóstico Automático de Estabilidade e Configuração Automática de Parâmetros de Simulação. A implementação dos métodos não compromete em nada a utilização da metodologia como uma aplicação em separado. Realizou-se também a

implementação de um método *Configure()* nas classes que representam os eventos em uma simulação detalhada (classe derivadas de *C_Mon_Data*). A criação e configuração dos parâmetros dos objetos evento em uma simulação detalhada convencional são realizados no momento da leitura do arquivo de controle do mesmo. Como no ASDIN o objeto SIMSP não utiliza arquivo de controle, houve a necessidade da criação de um método *Configure()* que realize a configuração destes parâmetros. As modificações na classe *C_SIMSP* (assim como nas classes associadas a mesma) são apresentadas na Figura 4.7.

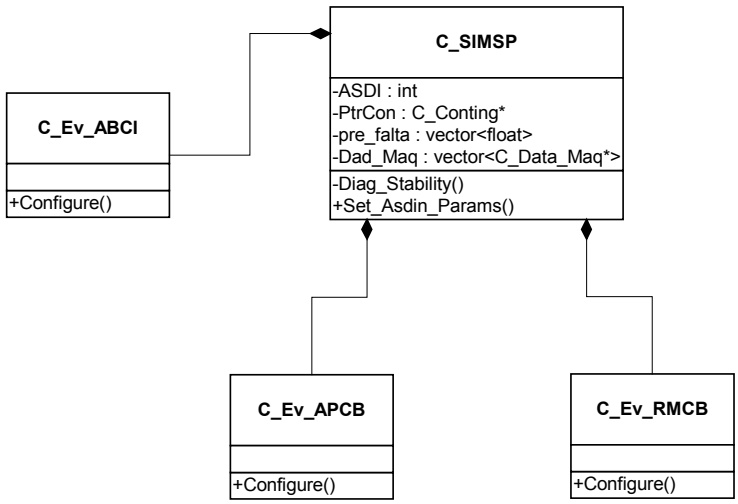

Figura 4.7 – Modificações nas Classes da Aplicação SIMSP

- ✓ *ASDI*: informa ao objeto SIMSP se ele está participando da solução de um problema de ASDIN;
- ✓ *PtrCon*: ponteiro para o objeto contingência. Esta estrutura de dados contém todas as informações sobre a contingência a ser analisada;
- ✓ *pre_falta*: guarda informações do ângulo em regime permanente das máquinas que farão parte da simulação dinâmica detalhada;
- ✓ *Dad_Maq*: vetor que contém ponteiros para as máquinas síncronas sob análise;
- ✓ *Diag_Stability()*: método que realiza o diagnóstico rápido de estabilidade e armazena os dados que precisam ser monitorados;
- ✓ *Set_ASDIN_Params()*: realiza a configuração dos parâmetros para a simulação dos eventos.

4.3.2.4 Classe *C_Flow_NR*

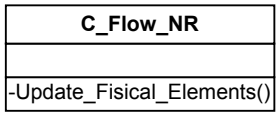

Figura 4.8 – Modificações na Classe *C_Flow_NR*

As modificações necessárias realizadas na classe *C_Flow_NR* para que a mesma pudesse ser incorporada a ferramenta computacional ASDIN não foram de grande vulto. O único método criado é o que realiza a atualização dos elementos físicos da BCOO pois os mesmos precisam ser lidos pelas metodologias de análise posteriores. O método aciona as Interfaces Funcionais das barras para que as mesmas realizem esta tarefa. Apresenta-se na Figura 4.8 as modificações necessárias para a utilização do objeto Fluxo de Potência como parte da metodologia ASDIN.

4.3.2.5 Classe *C_Conting*

Esta estrutura contém os dados relativos às contingências críticas selecionadas pelo método SLEP iterativo. Esta classe funciona analogamente a um banco de dados armazenando todas as informações relativas as contingências críticas selecionadas. Estes dados estão disponíveis a todos os objetos que formam o ASDIN, representando a estrutura principal de dados que compõe a ferramenta. O diagrama de classe juntamente com seus atributos e métodos é mostrado na Figura 4.9:

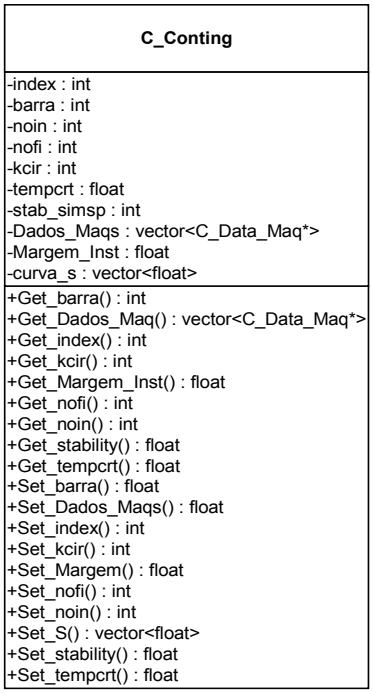

Figura 4.9 – Classe *C_Conting*

- ✓ *index*: índice da contingência crítica;
- ✓ *barra*: número da barra onde o curto-circuito foi aplicado;
- ✓ *noin*: número da barra de origem da linha de transmissão ou transformador a ser retirado;

- ✓ *nofi*: número da barra de destino da linha de transmissão ou transformador a ser retirado;
- ✓ *kcir*: número do circuito a ser retirado;
- ✓ *tempcrt*: tempo crítico de eliminação do defeito para esta contingência crítica;
- ✓ *stab_simsp*: indica se a condição é estável ou instável após a simulação desta como evento da metodologia SIMSP;
- ✓ *Margem_Inst*: margem em energia calculada;
- ✓ *Dados_Maqs*: vetor que contém os ponteiros para os objetos *C_Data_Maq*. Estes objetos realizam a monitoração das máquinas síncronas presentes na simulação;
- ✓ *curva_s*: contém os coeficientes de sensibilidade para as Unidade de Geração do sistema;
- ✓ *Get_barra()*: retorna o número da barra onde ocorre o curto-circuito;
- ✓ *Get_Dados_Maq()*: retorna vetor com ponteiros para os objetos *C_Data_Maq;*
- ✓ *Get_index():* retorna o índice da contingência;
- ✓ *Get_kcir():* retorna o número do circuito retirado;
- ✓ *Get_Margem_Inst()*: retorna a margem de instabilidade calculada para esta contingência;
- ✓ *Get_nofi():* retorna o número da barra de destino da linha de transmissão ou transformador a ser retirado;
- ✓ *Get_noin():* retorna o número da barra de origem da linha de transmissão ou transformador a ser retirado;
- ✓ *Get_Stability():* retorna se o evento é instável ou estável após a simulação dinâmica;
- ✓ *Get_tempctr():* retorna o tempo crítico para eliminação do defeito;
- ✓ *Set_barra():* configura o número da barra que está sob curto-circuito;
- ✓ *Set_Dados_Maqs():* configura o vetor com ponteiros para os objetos *Dad_Maqs*;
- ✓ *Set_index():* configura o índice da contingência;
- ✓ *Set_kcir():* configura o número do circuito a ser retirado;
- ✓ *Set_Margem():* configura o valor da margem em energia calculada;
- ✓ *Set_nofi():* configura a barra de destino da linha de transmissão ou transformador a ser retirado;
- ✓ *Set_noin():* configura a barra de origem da linha de transmissão ou transformador a ser retirado;
- ✓ *Set_S():* configura os coeficientes de sensibilidade para a metodologia da direção S;
- ✓ *Set_stability():* configura a contingência como estável ou não de acordo com o resultados da análise detalhada;
- ✓ *Set_tempcrit():* estabelece o valor do tempo crítico para eliminação de defeito.

4.3.2.6 Classe *C_Dad_Maqs*

A classe foi criada para o armazenamento dos parâmetros das máquinas síncronas e dados que devem ser monitorados durante o processo de simulação no domínio do tempo. Estes dados são necessários em diversos etapas do processo de

Avaliação da Segurança Dinâmica de SEE. A classe com seus métodos e atributos é apresentada na Figura 4.10.

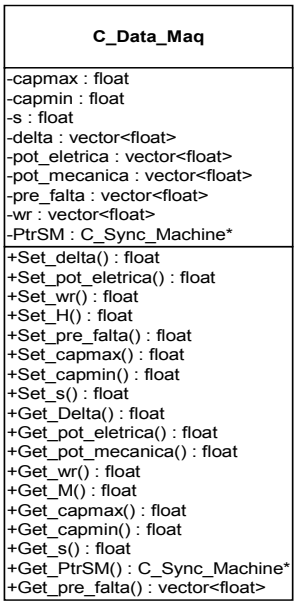

Figura 4.10 – Classe *C_Data_Maq*

- ✓ *capmax*: capacidade máxima de geração da unidade;
- ✓ *capmin*: capacidade mínima de geração da unidade;
- ✓ *s*: retornao o coeficiente da Direção S da unidade da unidade de geração;
- ✓ *delta*: vetor que contém os valores dos angulas das máquinas até o momento onde a instabilidade é detectada;
- ✓ *pot_eletrica*: vetor que contém os valores de potência elétrica até o momento onde a instabilidade é detectada;
- ✓ *pot_mecanica*: vetor que contém os valores de potência mecânica até o momento onde a instabilidade é detectada;
- ✓ *pre_falta*: vetor que contém o ângulo das máquinas no instante pré-falta;
- ✓ *wr:* vetor que contém as velocidades angulares a cada passo de integração das máquinas síncronas que fazem parte da simulação;
- ✓ *PtrSM:* ponteiro para a máquina síncrona que esta sendo monitorada;
- ✓ *Set_delta():* armazena o valor do ângulo da máquina síncrona a cada passo de integração;
- ✓ *Set_pot_eletrica():* armazena o valor da potência elétrica a cada passo de integração;
- ✓ *Set_wr():* armazena o valor da velocidade angular da máquina síncrona a cada passo de integração;
- ✓ *Set_H():* armazena o valor da constante de inércia da máquina síncrona;
- ✓ *Set_pre_falta():* armazena vetor pré-falta(ângulos das máquinas em regime permanente);
- ✓ *Set_capmax():* configura o valor da capacidade máxima de geração da unidade;
- ✓ *Set_capmin():* configura o valor da capacidade mínima de geração da unidade;
- ✓ *Set_s():* configura o coeficiente da direção S da máquina síncrona monitorada;

- ✓ *Get_delta():* retorna o valor do ângulo das máquinas síncronas de acordo com o passo de integração escolhido;
- ✓ *Get_pot_eletrica():* retorna o valor da potência elétrica de acordo com o passo de integração escolhido;
- ✓ *Get_pot_mecanica():* retorna o valor da potência mecânica de acordo com o passo de integração escolhido;
- ✓ *Get_wr():* retorna velocidade angular de acordo com o passo de integração escolhido;
- ✓ *Get_M():* retorna o valor da constante de inércia da máquina síncrona;
- ✓ *Get_capmax():* retorna o valor da capacidade máxima de geração da unidade;
- ✓ *Get_capmin():* retorna o valor da capacidade mínima de geração da unidade;
- ✓ *Get_s():* retorna o valor do coeficiente da direção S da máquina síncrona;
- ✓ *Get_PtrSM():* retorna o ponteiro para máquina síncrona monitorada pelo objeto *C_Data_Maq;*
- ✓ *Get_pre_falta():* retorna o valor do ângulo pré-falta.

Cada máquina síncrona que participa da simulação detalhada terá um objeto *Data_Maq* associado a mesma. Este objeto é responsável pelo armazenamento dos valores contidos nos vetores acima descritos. Os valores contidos nestes vetores são, posteriormente, utilizados para o cálculo de margem em energia do evento, adotada na classificação das contingências de acordo com a sua severidade.

4.3.2.7 Classe *C_ASDIN_Events*

Esta classe contém os parâmetros de simulação para o objeto SIMSP. Os atributos e métodos que compõe esta classe são mostrados na Figura 4.11.

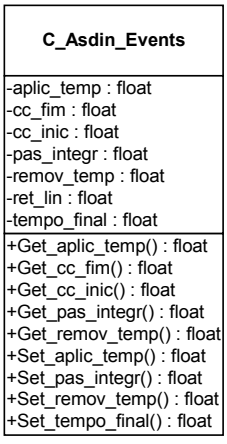

Figura 4.11 – Classe *C_ASDIN_Events*

- ✓ *aplic_temp*: tempo de aplicação do curto-circuito;

- ✓ *cc_fim*: instante de tempo de final de aplicação do curto-circuito;
- ✓ *cc_inic*: instante de tempo inicial de aplicação do curto-circuito;
- ✓ *pas_integr*: passo de integração para a realização da simulação detalhada;
- ✓ *remov_temp*: tempo de remoção do curto-circuito;
- ✓ *ret_lin*: tempo necessário para a remoção da linha de transmissão ou transformador;
- ✓ *tempo_final*: tempo final de simulação;
- ✓ *Get_aplic_temp()*: retorna tempo de aplicação do curto-circuito;
- ✓ *Get_cc_fim()*: retorna instante de tempo final de aplicação do curto-circuito;
- ✓ *Get_cc_inic()*: retorna instante de tempo inicial de aplicação do curto-circuito;
- ✓ *Get_pas_integr()*: retorna o passo de integração da simulação detalhada;
- ✓ *Get_remov_temp()*: retorna o tempo de remoção do curto-circuito;
- ✓ *Set_aplic_temp()*: estabelece o tempo de aplicação de curto-circuito;
- ✓ *Set_pas_integr()*: estabelece o passo de integração da simulação detalhada;
- ✓ *Set_remov_temp()*: estabelece o tempo para remoção do curto-circuito;
- ✓ *Set_tempo_final()*: estabelece o tempo final de simulação.

Os parâmetros de simulação podem ser facilmente modificados através dos métodos *Get*/*Set* implementados nesta classe.

4.3.3 Diagrama de Atividades

Após a apresentação do Diagrama de Classes do ASDIN apresenta-se o Diagrama de Atividades. Este diagrama é essencialmente um gráfico de fluxo mostrando o fluxo de controle de uma atividade para a outra, conforme apresentado na Figura 4.12.

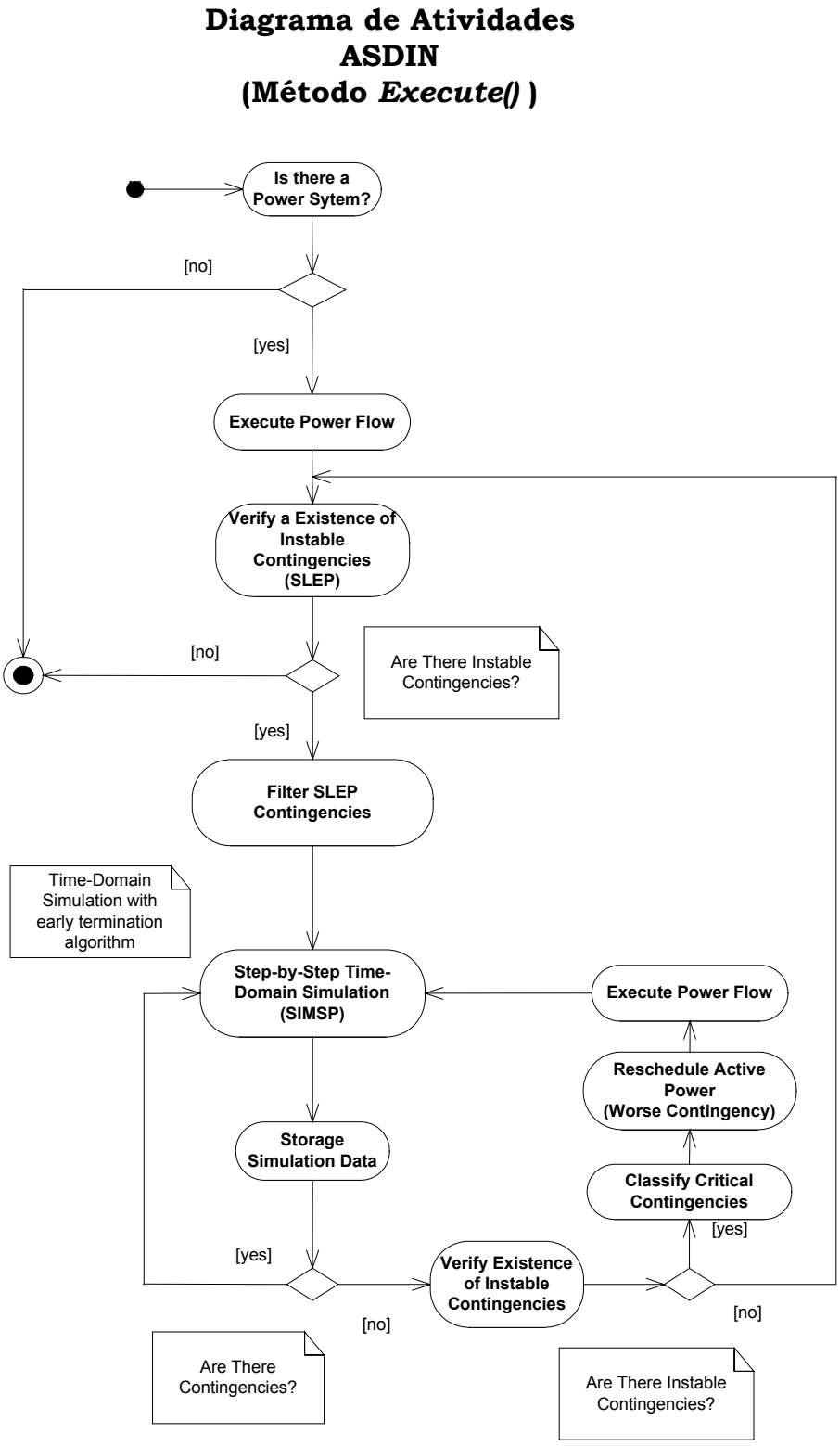

Figura 4.12 – Diagrama de Atividades da ferramenta ASDIN

Observando-se o Diagrama de Atividades da ferramenta ASDIN verifica-se a existência de dois laços principais de análise das contingências críticas. Um laço é

composto pelo processo de análise utilizando-se de modelos simplificados e um segundo laço utilizando-se de modelos detalhados.

O processo tem início com a verificação da existência de contingências críticas pelo objeto SLEP. Esta verificação pode resultar em um conjunto de contingências críticas selecionadas. Caso haja contingências críticas cada uma delas é novamente analisada usando-se a simulação no domínio do tempo com modelos detalhados. Após esta análise, caso se confirme a instabilidade das contingências, as mesmas são classificadas de acordo com a sua severidade. A contingência mais severa é selecionada e utilizada como entrada para o processo de redespacho de potência ativa. É a partir desta contingência que as ações de redespacho são efetuadas. Após a realização desta etapa, o sistema é novamente analisado utilizando modelagem detalhada e ações de redespacho são efetuadas de acordo com a necessidade até que todas as contingências críticas tenham sido eliminadas.

Com a eliminação das contingências críticas pelo primeiro laço, o segundo laço é utilizado como salvaguarda de que as ações de redespacho realizadas no primeiro laço tenham realmente deixado o sistema livre das contingências mais severas. O objeto SLEP é então novamente executado e caso haja necessidade todo o processo de análise detalhada é novamente inicializado.

4.4 Validação da Ferramenta Computacional ASDIN

A validação desta ferramenta computacional foi feita por meio de simulações com os sistemas Sis45, Sis730 e Sis1916 descritos anteriormente. Utilizou-se para estudos comparativos o protótipo computacional desenvolvido em SOUZA (1999). Na Tabela 4.1 apresenta-se os valores dos redespachos totais, para o sistema Sis730, obtidos com o ASDIN e com o protótipo de SOUZA (1999).

ASDIN	**Protótipo** (Souza 1999)	**Erro**
689.4 MW	695.5 MW	0.88%

Tabela 4-2-Comparação de Resultados do Redespacho de Potência Ativa

A comparação entre os redespachos finais é interessante pois o mesmo acumula os erros de todas as metodologias que compõem a ferramenta (Fluxo de Potência,

Simulação Detalhada e Redespacho). Observa-se na Tabela 4.1 que a diferença entre os redespachos fica abaixo de 1%, validando-se assim a ferramenta computacional. Os parâmetros relevantes envolvidos nas simulações são descritos em termos de itens de acordo com a etapa a ser simulada, como segue:

- **Seleção e classificação de contingências (*C_SLEEP*)**: As contingências consideradas são do tipo curto-circuito trifásico sólido com posterior desligamento permanente do elemento de rede incidente à barra sob-defeito. As máquinas síncronas são representadas pelo modelo clássico e as cargas são representadas por impedâncias constantes. Para a seleção das contingências considera-se um tempo de permanência do defeito de 0,20 segundos. Assim, as contingências identificadas como instáveis para este tempo especificado são consideradas contingências críticas. O método numérico para cálculo da trajetória é o da Série de Taylor com seis termos e passo de integração de 0,010 segundos.

- **Simulação no domínio do tempo (*C_SIMSP*):** As contingências consideradas são do tipo curto-circuito trifásico sólido com posterior desligamento permanente do elemento de rede incidente à barra sob-defeito. Com relação à modelagem dos elementos do SEE foram considerados, especificamente, os efeitos subtransitórios devido aos enrolamentos amortecedores para as máquinas síncronas; a representação dos reguladores automáticos de tensão em todas as máquinas síncronas de acordo com o modelo IEEE Tipo 1; e para as cargas, o modelo polinomial com 100% de impedância constante.

Como relação ao desempenho computacional, simulou-se o sistema Sis730 a partir de um conjunto de 671 contingências, sendo que nove destas contingências foram selecionadas como críticas. A plataforma computacional utilizada é a mesma que esta descrita no Capítulo 3. Na Tabela 4.2 apresenta-se os tempos computacionais obtidos com o ASDIN comparados às simulações seriais realizadas em SOUZA (1999).

ASDIN	**Protótipo**	**Diferença**
11 min	24,6 min	55,28%

Tabela 4-3- Comparação de Tempos Computacionais

Observa-se um ganho de 55,28% com relação aos processamentos apresentados em SOUZA (1999). Esta comparação serve apenas como uma estimativa pois as

simulações foram realizadas em plataformas distintas. SOUZA (1999) utilizou como plataforma computacional estações IBM Power PC de 200MHz, com sistema operacional AIX e compiladores proprietários da IBM.

CAPÍTULO 5

MODELAGEM DE APLICAÇÃO: AVALIAÇÃO GLOBAL DA ESTABILIDADE TRANSITÓRIA USANDO COMPUTAÇÃO PARALELA

5.1 Introdução

O presente capítulo trata da modelagem e implementação paralela do método SLEP iterativo seguindo a filosofia proposta por AGOSTINI et al. (2002a) e do ambiente computacional de Processamento de Alto Desempenho do LabPlan, implementado para a execução do mesmo. Deseja-se com isso realizar uma investigação preliminar do emprego da BCOO e a filosofia de desenvolvimento de softwares, para ambientes computacionais paralelos.

O capítulo inicia com uma descrição sucinta dos aspectos principais da tecnologia de clusters de computadores, destacando-se os do tipo BEOWULF. Na seqüência é realizada uma descrição das motivações e detalhes de implementação do ambiente de Processamento de Alto Desempenho (PAD) do LabPlan. O capítulo termina com uma breve discussão sobre a implementação paralela realizada usando os recursos da Base Computacional Orientada a Objetos. Discute-se os principais pontos desta implementação e as técnicas utilizadas para a realização da mesma tanto em termos de Modelagem Orientada a Objetos quanto em termos de técnicas de processamento paralelo.

5.2 Clusters de Computadores

As arquiteturas usadas em computação paralela sempre foram uma promessa em termos de aumento de desempenho quando comparadas as arquiteturas tradicionais como a de Von Neumann. Porém o alto custo em termos de hardware e a complexidade na programação formavam um conjunto de fatores que dificultavam a concretização desta promessa (SCHIAVONE et al., 2000).

Na última década a comodidade de uso e o poder computacional proporcionados pelos desktops impulsionaram o uso do PC (*Personal Computer*). Combinado a isto o incremento de largura de banda e o baixo custo das tecnologias de rede têm aberto uma porta para um novo tipo de computador paralelo de baixo custo baseado em clusters de PCs, as vezes referidos como *networks of workstations* (NOWs) ou pilhas de PCs (POPs). Como definição básica, um cluster é um sistema de processamento paralelo ou distribuído, composto de uma coleção de computadores – os nós – interconectados por uma rede de alta velocidade e agrupados de forma a trabalharem como um recurso computacional único e integrado. Cada nó consiste em um sistema mono ou multiprocessado, com memória, dispositivos de E/S e sistema operacional. Os nós podem se localizar em um gabinete único ou estarem fisicamente separados .

No trabalho de BACKER (1999) estão sintetizadas algumas das vantagens do processamento em cluster, são elas:

- desempenho das estações utilizadas nos clusters tem aumentado rapidamente nos últimos anos. Isto provavelmente deve continuar por vários anos, com microprocessadores mais rápidos e máquinas multiprocessadas chegando ao mercado;

- à medida que novas tecnologias e protocolos são implementados em redes locais, a largura de banda entre estações de trabalho vem aumentando enquanto a latência está diminuindo;

- os clusters de estações de trabalho são mais fáceis de integrar às redes existentes do que computadores paralelos especializados;

- são uma alternativa barata e prontamente disponível, pois são dispositivos montados usando componentes de massa facilmente encontrados no mercado;

- os clusters podem ser facilmente expandidos; a capacidade dos nós pode ser facilmente aumentada pelo acréscimo de memória ou microprocessadores adicionais.

Por outro lado, algumas desvantagens do processamento em cluster, são listadas a seguir:

- a programação paralela não é uma técnica comummente dominada pelos pesquisadores e é bem mais complexa do que a programação seqüencial;

- há tipos de problemas científicos que não são adequados para o processamento paralelo, portanto não são adequados para execução em clusters;

- a latência associada ao uso da rede de interconexão e o *overhead* da pilha de protocolos de comunicação são, em geral, significativamente maiores que em outras arquiteturas, como as MPP (*Massively Parallel Processing*), prejudicando o uso de clusters em aplicações com granularidade mais fina.

5.2.1 Classificação de Clusters de Computadores

BACKER (1999) classificou os clusters de várias formas. Os critérios adotados foram:

a) Finalidade de Aplicação:

- Processamento de Alto Desempenho (PAD)

- Alta Disponibilidade

b) Utilização dos Nós:

- Clusters Dedicados – o poder de processamento das estações é de uso exclusivo das aplicações submetidas ao *cluster*;

- Clusters Não-Dedicados – as estação são utilizadas prioritariamente para uso pessoal e somente quando estão ociosas é que processam as aplicações paralelas.

c) Hardware dos Nós:

- Clusters de PCs;
- Clusters de Estações de Trabalho;
- Cluster de Symetric Multiprocessors (SMPs), máquinas com 2 ou mais microprocessadores compartilhando todos os recursos disponíveis (barramento, memória, sistema de E/S).

d) Sistema Operacional dos Nós:

- GNU/Linux, Solaris, Windows NT, AIX, etc.

e) Configuração dos Nós:

- Homogêneos – todos os nós possuem arquiteturas similares e executam o mesmo sistema operacional;
- Heterogêneos – os nós possuem diferentes arquiteturas e executam diferentes sistemas operacionais.

5.2.2 Clusters do Tipo BEOWULF

Clusters baseados em GNU/Linux são conhecidos como *Beowulf Clusters*, e foram inicialmente desenvolvidos no CESDIS (*Center of Excellence in Space Data and Information Sciences*) da NASA em 1994. A idéia de *Beowulf Clusters* é minimizar a relação de custo-desempenho usando hardware de baixo custo e pacotes baseados em software livre para a implementação do ambiente computacional.

Os componentes adotados são fabricados segundo padrões largamente difundidos na indústria, beneficiando-se de preços reduzidos já que os componentes são submetidos a grande competição e produção em massa. Assim os componentes geralmente são padronizados e a troca de fornecedor pode ser realizada de acordo com a necessidade do desenvolvedor.

Aplicações para este tipo de cluster são encontrados em diversas áreas da ciência entre elas pode-se citar:

- Previsão do Tempo;

- Dinâmica de Fluidos;
- Farmacocinética;
- Seqüenciamento Genético;
- Processamento de Imagens.

Na área de Sistemas de Energia Elétrica são encontrados muitos problemas de porte computacional elevado e com potencial para a utilização de processamento paralelo. Esta é a razão para a construção de um cluster Beowulf no LabPlan – Laboratório de Planejamento de Sistemas de Energia Elétrica da UFSC. Os detalhes são apresentados nos próximos itens.

5.3 Ambiente de PAD do LabPlan

O ambiente de Processamento de Alto Desempenho (PAD) desenvolvido no Labplan foi implementado para atender a demanda computacional de três projetos de pesquisa. Para o planejamento da operação de sistemas hidrotérmicos, quando trata-se do planejamento da operação de longo prazo, procura-se determinar estratégias ótimas de geração, de modo a minimizar o custo da operação ao longo de todo o período de planejamento. Especificamente para o sistema brasileiro, a solução passa pela resolução de um problema de Programação Dinâmica Estocástica Dual (MACEIRA, 1993), onde as usinas podem ser representadas sob a forma individualizada permitindo uma modelagem mais realista do parque de geração hidráulico. Este tipo de abordagem auxilia na maximização dos recursos energéticos garantindo o uso eficiente da água. Observa-se que a representação individualizada das usinas, sob o ponto de vista prático, é inviável com a utilização dos recursos computacionais convencionais (FINARDI et al., 2000). O algoritmo entretanto apresenta uma grande quantidade de processamento e baixo nível de comunicação, evidenciando a possibilidade de paralalelização do mesmo.

Na área de meteorologia aplicada a problemas de SEE, cita-se a execução de um Modelo Numérico de Previsão de Tempo que realiza Previsão Quantitativa de Chuva (PQC) para a região Sul do Brasil. O custo computacional de execução deste modelo é facilmente entendido quando observamos o que necessita-se modelar para a resolução do problema. Modelos Numéricos de Previsão de Tempo nada mais são que equações

matemáticas que representam a atmosfera como um todo juntamente com os fenômenos associados a mesma.

A saída deste modelo (quantidade de chuva) é usada como condição inicial para a execução de um modelo hidrológico de previsão de afluências desenvolvido pelo IPH (Instituto de Pesquisas Hidrológicas do Rio Grande do Sul). O conhecimento destas afluências é de grande interesse para a realização do despacho de potência das usinas no curto prazo(otimização energética), bem como para fins de segurança no caso de previsão de cheias. A relevância deste tema é ainda maior quando observamos que a matriz energética brasileira é composta por 94% de geração hidráulica.

Pesquisas realizadas na área de Segurança Dinâmica também exigem elevada demanda computacional. A principal necessidade desta área é com relação ao atendimento dos requisitos de tempos computacionais do ambiente de operação *on-line*. Os operadores precisam obter informações corretas e em tempo hábil para a realização do processo de tomada de decisão. Estas questões são tratadas em maior detalhe em SOUZA (1999).

Especificamente neste capítulo realiza-se a implementação de uma metodologia de Seleção e Classificação de Contingências Críticas usando modelagem simplificada para SEEs baseada no SLEP iterativo. Dado o elevado custo computacional da aplicação a paralelização do mesmo trás bons ganhos em termos de desempenho da aplicação. Maiores detalhes com relação ao ambiente de PAD do LabPlan serão apresentados nos próximos itens.

5.3.1 Descrição do Hardware

O ambiente atual de PAD do LabPlan consiste de um cluster de microcomputadores formado por 16 PCs, sendo que cada PC também é chamado de Nó. A conexão de rede existente entre os Nós é composta por uma rede padrão *Fast Ethernet* de 100 Mbps. O sistema operacional utilizado foi o GNU/LINUX Mandrake Click, distribuição projetada e otimizada para atender aos desenvolvedores de clusters. Este cluster pode ser classificado como sendo do tipo BEOWULF, como pode ser observado de acordo com os itens que compõe o ambiente. O sistema é apresentado na Figura 5.1.

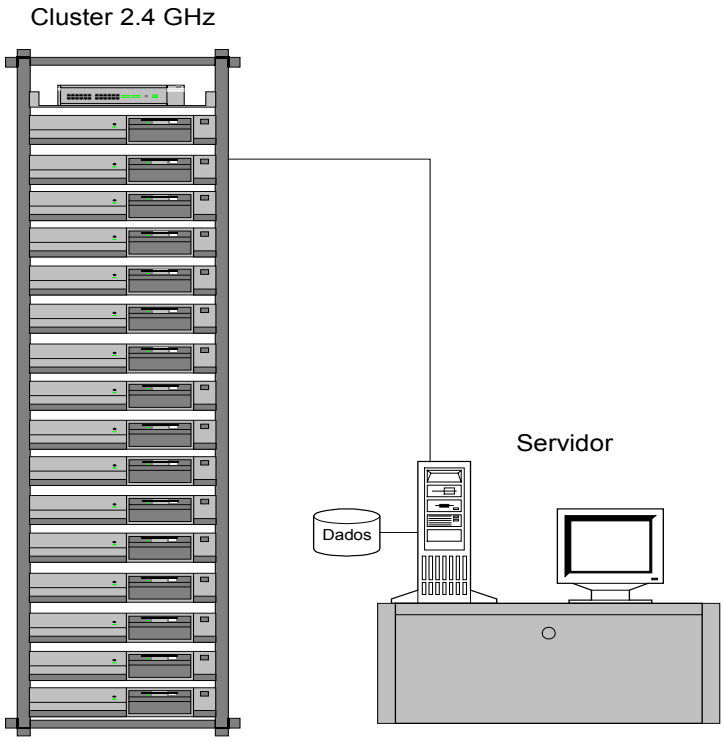

Figura 5.1 – Representação Esquemática do Ambiente de PAD

As configurações do Servidor e de cada Nó são descritas a seguir:

Cluster 2.4 GHz (Servidor)	
Componente	**Especificação**
Processador	Pentium IV
Placa	Intel SE7505
Cache L2	512KB
Memória RAM	RAMBUS 533 Mhz
Placa Rede	100 Mbits
HD SCSI	70 GB

Cluster 2.4 GHz (Nós)	
Componente	**Especificação**
Processador	Pentium IV
Placa	Intel SE7505
Cache L2	512KB
Memória RAM	DDR 333 MHz
Placa Rede	100 Mbits

Tabela 5-1- Configuração de Hardware

Um dos aspectos importantes de um ambiente de PAD é a sua eficiência computacional. No caso do cluster sob descrição, foram realizados testes de desempenho executando-se o software Linpack . O software Linpack é uma coleção de

subrotinas para a resolução de sistemas de equações lineares. Com base nestas subrotinas o Top500 disponibiliza o *Linpack Benchmark* e o HPL– *High Performance Linpack* para a execução em ambientes paralelos. Este *benchmark* é utilizado como teste para a classificação dos 500 computadores mais rápidos mundo. Nos resultados com o software Linpack, obteve-se o desempenho máximo de 32.4 Gigaflops com dezesseis CPUs, simulando-se a resolução de um sistema linear de tamanho 20000x20000.

5.3.2 Implementação e Administração do Sistema

Os PCs não foram projetados e construídos para operar em ambientes de processamento intensivo e alta disponibilidade. Assim sendo, é comum que alguns destes componentes apresentem problemas ao longo do tempo de acordo com uma grande gama de motivos. Este tipo de problema tem uma maior importância quando a dimensão do sistema vai aumentando (20, 40, 100 nós por exemplo). Para tentar contornar estes problemas vários métodos e ferramentas, tanto em termos de software com hardware, foram e estão sendo empregados para que estes problemas sejam minimizados.

Em termos de hardware, as principais dificuldades estão relacionadas a temperatura do ambiente, dada pelo excesso de calor gerado pelo agrupamento de várias CPUs em um mesmo local. Um outro problema importante é com relação a capacidade e facilidade de se efetuar a manutenção de componentes defeituosos do sistema, para que as paradas necessárias para manutenção sejam realizadas em um curto espaço de tempo. Constatou-se que a resolução destes problemas passa por um bom projeto do sistema físico do ambiente de PAD (gabinetes e racks). Para a solução do problema da temperatura optou-se pela instalação de um sistema de ventiladores localizados no interior dos gabinetes. Estes ventiladores ajudam na retirada do calor existente no interior do gabinete. O calor retirado do gabiente é dissipado do interior do rack através de um sistema de exaustão localizado na parte superior do mesmo. Este sistema auxilia na retirada do calor gerado pela operação das diversas CPUs. A proposta com relação a manutenabilidade do ambiente é melhor entendida observando-se a Figura 5.2.

Figura 5.2 – Vista Frontal do Ambiente de PAD

Nesta figura observa-se o sistema de gabinetes industriais móveis adotado. Este sistema já projetado para esta tarefa, facilita muito a troca de componentes com defeito já que basta puxar, o gabinete, abrí-lo e realizar assim o processo de manutenção.

Em termos de software foi adotado o uso da tecnologia *diskless*, a qual consiste basicamente em não se usar discos rígidos nos Nós do cluster. Desse modo reduz-se o número de falhas provocados por problemas em discos rígidos (principalmente relacionados a sistema operacional) e a não necessidade de instalação do sistema operacional em todos os Nós do ambiente. A facilidade de adminstração deste sistema também deve ser destacada já que basicamente resume-se a intervenções no servidor do ambiente. O processo de inicialização é totalmente realizado remotamente usando recursos da placa de rede juntamente com pacotes disponibilizados pelo projeto *Linux Terminal Server Project*. Ressalta-se ainda que uma das principais características de um cluster BEOWULF está sendo atendida, pois há uma grande economia em termos de aquisição de discos rígidos.

Em um sistema com elevado número de nós a serem administrados é praticamente indispensável o uso de uma ferramenta de monitoração. Para atender a esta necessidade foi instalado o sistema de monitoração Ganglia, software desenvolvido pela

Universidade da Califórnia (Berkley) para a monitoração de seus recursos computacionais. Este sistema realiza a monitoração de vários parâmetros de hardware (como velocidade do cooler, temperatura do processador, temperatura da placa mãe, etc) e de parâmetros de sistema como o uso da memória, o uso da placa de rede, o processamento realizado, etc. Estas informações são armazenadas em uma base de dados no servidor e ficam disponíveis via internet em forma de gráficos. Exemplos destes gráficos são mostrados na Figura 5.3.

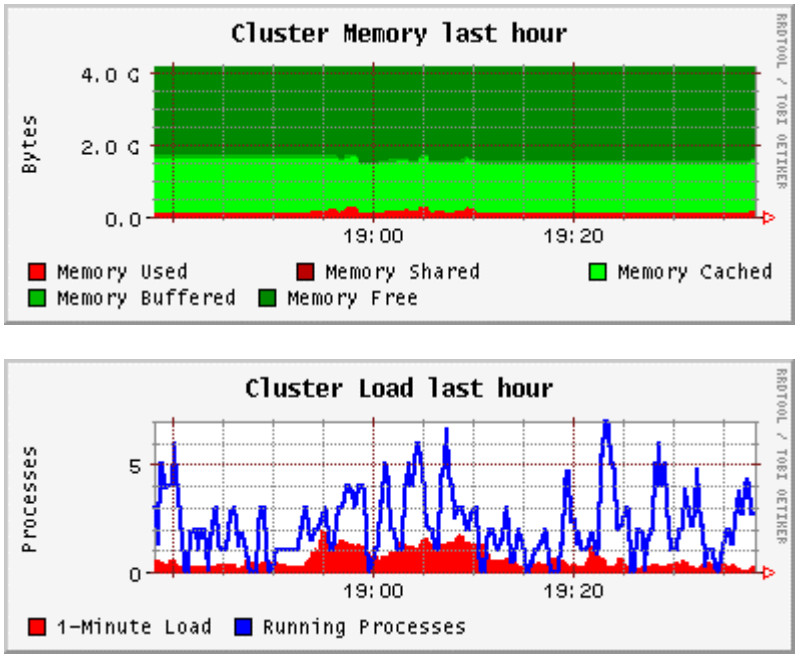

Figura 5.3 – Exemplos de Saídas do Software de Monitoração Ganglia

Estas informações são de grande importância para o entendimento e conhecimento do sistema. De posse destes dados identificam-se com maior facilidade os gargalos do sistema e consequentemente consegue-se um melhor direcionamento do investimento em upgrade dos componentes de hardware.

Outro aspecto importante é com relação a segurança dos dados do sistema. Um sistema de filtro de pacotes (firewall), baseado em IPTABLES, foi instalado para que este item tão relevante nos dias de hoje seja atendido.

5.3.3 Ambiente de Desenvolvimento

O ambiente de PAD do LabPlan conta hoje com um ambiente de desenvolvimento onde compiladores, editores de texto, bibliotecas numéricas e de troca de mensagem (PVM e MPI) estão disponíveis aos usuários do sistema. Além dos

tradicionais pacotes disponibilizados pela GNU, foi adquirida uma licença do compilador desenvolvido pelo Grupo de Portland. A aquisição deste compilador foi motivada pela não existência de um compilador Fortran 90 livre que estivesse no nível do desenvolvido pelo Grupo de Portland. Estão disponíveis também para uso compiladores desenvolvidos pela Intel e que são gratuitos (não livres) para uso não comercial, quando utilizados sob a plataforma GNU/Linux. O compilador da Intel tem chamado a atenção de toda a comunidade de desenvolvedores BEOWULF pela sua qualidade e os ganhos em termos de desempenho.

Contudo, o ambiente de PAD do LabPlan ainda carece de melhores ferramentas de desenvolvimento de softwares e soluções de acesso amigável (ambiente gráfico) ao ambiente. Neste sentido novos pacotes vem sendo pesquisados, priorizando pacotes de software livre para que este problema seja sanado. Como principal contribuição cita-se o Anjuta (anjuta.sourceforge.net), um ambiente integrado de desenvolvimento de software totalmente livre e comparável aos diversos pacotes de desenvolvimento disponíveis no mercado(C Builder, Visual Studio, etc.). Testes preliminares foram realizados e mostraram a sua compatibilidade com aplicações desenvolvidas em outros ambientes. A facilidade do uso do mesmo no desenvolvimento de programas para o setor elétrico. Esta é no momento a principal meta do grupo de administração de sistemas do cluster LabPlan. As oportunidades e a aplicabilidade de software livre na indústria de energia elétrica atual são discutidos no artigo de KLEIN (2003).

5.4 Modelagem do Método SLEP para Ambiente Paralelo

Neste item realiza-se a descrição de uma modelagem e implementação preliminar de um método para seleção de contingências críticas utilizando recursos da BCOO (SLEP iterativo). Uma das motivações para esta modelagem é o propósito de, em curto espaço de tempo, adaptar a ferramenta de Avaliação da Segurança Dinâmica (ASDIN), descrita no capítulo quatro, totalmente para ambientes de processamento paralelo, incluindo-se, também, novos desenvolvimentos no método de ações de controle para a melhoria da segurança, seguindo os preceitos da nova filosofia proposta por AGOSTINI (2002a). Para que este objetivo seja alcançado a paralelização da etapa de seleção das contingências críticas é de fundamental importância. Procurando

contribuir para a construção desta ferramenta e com base nas constatações presentes em SOUZA (1999), onde o autor mostra que o ganho de tempo computacional paralelizando-se o processo de seleção de contingências críticas é maior quando comparado com processos de análise detalhada e de melhoria, reaproveitou-se o método SLEP iterativo paralelo implementado naquele protótipo.

Baseado no exposto acima implementou-se a classe *C_SLEP*. A classe é responsável pela seleção das contingências críticas utilizando técnicas de processamento paralelo no âmbito da BCOO. Esta classe foi implementada a partir do programa SLEP paralelo implementado em SOUZA (1999). As principais características daquele programa são as seguintes:

- paralelização a nível estrutural, distribuindo a análise das contingências para serem executadas concorrentemente em vários processadores;
- paradigma do paralelismo de dados, particionando a lista de contingências em blocos menores a serem enviados aos processadores participantes do ambiente de computação paralela definido;
- modelo de programação mestre-escravo;
- biblioteca de troca de mensagens PVM.

O processo mestre é responsável pela inicialização dos escravos, execução das operações de entrada e saída de dados, gerenciamento dos processo escravos, e partição das contingências entre os processadores. Os processos escravos são responsáveis pelo processamento das contingências. Maiores detalhes sobre a implementação computacional do programa SLEP paralelo são encontrados em SOUZA (1999).

A classe *C_SLEPP* é mais simples que a classe *C_SLE*. O objeto *SLEPP* basicamente lê o estado do sistema, disponível nos objetos que representam os elementos físicos na BCOO, e transfere este conjunto de dados para o processo mestre, que então assume o controle do programa disponibilizando como saída as contingências críticas do sistema. A implementação desta classe foi realizada utilizando-se as bibliotecas de trocas de mensagens do PVM, mesmo sabendo-se que esta biblioteca não conta com um desenvolvimento continuado desde o surgimento de bibliotecas de trocas de mensagens construídas observando o padrão MPI. Esta escolha foi feita devido a

intensa utilização de códigos implementados no âmbito do protótipo encontrado em SOUZA (1999).

O processo de execução deste objeto dentro da ferramenta OOTPS, descrita no Capítulo 2, é mostrado na Figura 5.4.

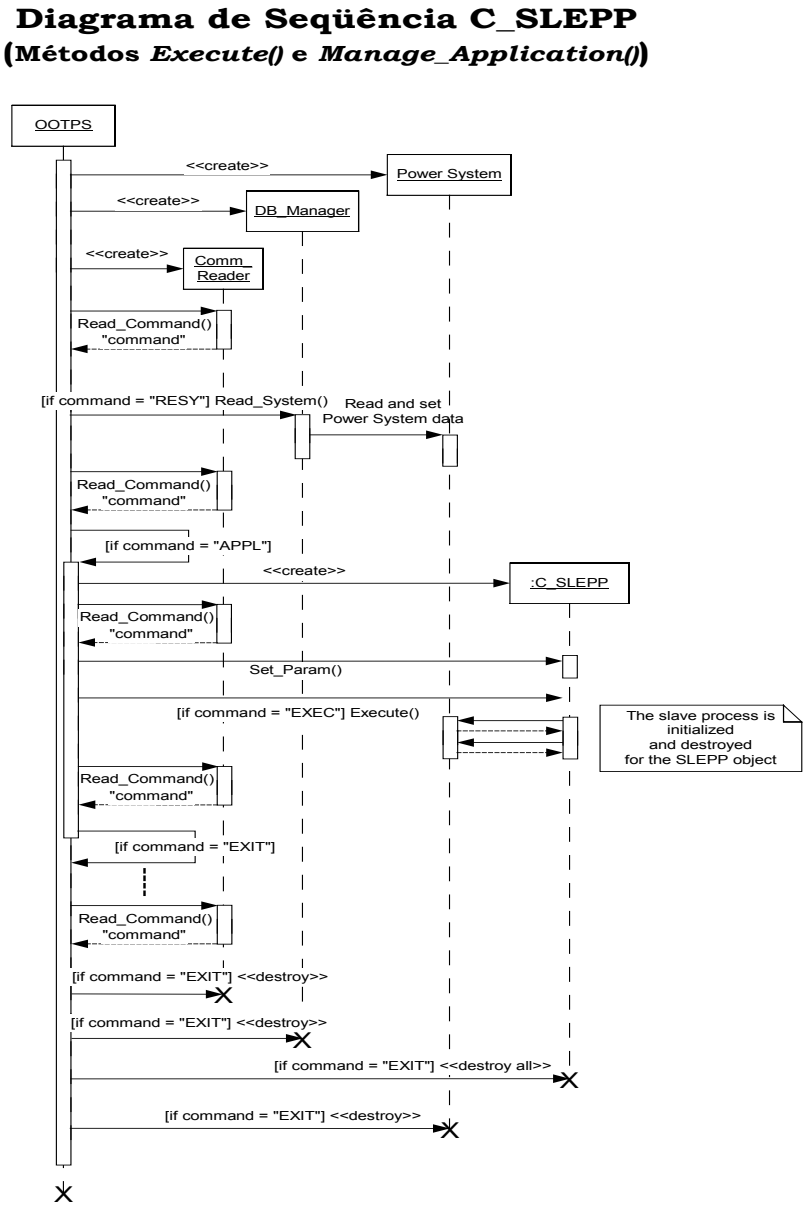

Figura 5.4 – Diagrama de Seqüência da Aplicação *C_SLEPP*

Este diagrama dá uma noção geral de como a classe *C_SLEPP* trabalha em conjunto com a base de dados orientada a objetos. A inclusão desta metodologia no ambiente da BCOO foi realizada com relativa facilidade, o que nos dá uma boa indicação de que a implementação de um programa de Análise da Segurança Dinâmica usando técnicas de processamento paralelo não será de grande dificuldade.

Entretanto, é aconselhável a realização de um estudo mais profundo e detalhado sobre como serão realizadas as implementação de programas paralelo de acordo com a nova filosofia de softwares para SEE.

5.5 Validação do Método SLEP para Ambiente Paralelo

A validação desta ferramenta computacional foi feita por meio de simulações com os sistemas Sis45, Sis730 e Sis1916 descritos no Capítulo 4. Utilizou-se para estudos comparativos o protótipo computacional desenvolvido em SOUZA (1999). Os valores obtidos foram os mesmos para os diversos sistemas testados validando a aplicação. Com relação ao desempenho computacional a execução do método utilizando recursos do *framework* OOTPS da BCOO não trouxe grandes prejuízos com relação ao tempo computacional. Não houveram grandes mudanças em termos de código, o *framewok* apenas transfere os dados para o objeto *SLEPP* que realiza o processamento. O objeto SLEPP não mais é do que o SLEP iterativo desenvolvido por SOUZA (1999). Os tempos computacionais (ambiente de PAD LabPlan) para a seleção e classificação das contingências críticas utilizando como entrada o sistema Sis730 e um conjunto de 1846 contingências são apresentados na Figura 5.5.

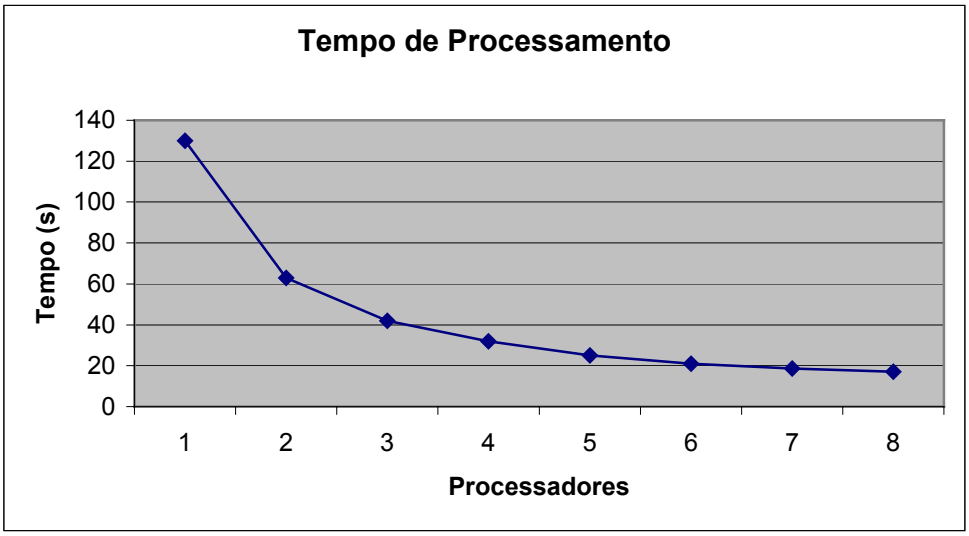

Figura 5.5 – Tempos de Processamento

No caso específico da simulação com oito processadores obteve-se um tempo de processamento total de 17 segundos e uma eficiência de aproximadamente 96%.

CAPÍTULO 6

CONCLUSÕES

Neste trabalho foram modeladas e implementadas metodologias de análise de sistemas de potência seguindo a nova filosofia de projeto de softwares para Sistemas de Energia Elétrica (SEEs), usando Modelagem Orientada a Objetos (MOO), proposta em AGOSTINI (2002a). Utilizou-se como ponto de partida para os desenvolvimentos realizados uma Base Computacional Orientada a Objetos (BCOO) para SEE. Nesta base estão representados segundo estruturas hierárquicas de classes, as mais diversas instâncias de um SEE desde os elementos físicos do sistema, até metodologias de aplicação e ferramentas computacionais finais. Estas estruturas oferecem um conjunto de abstrações bem definidas e que facilitam a adição de novas classes.

O ambiente integrado da BCOO mostrou sua eficácia proporcionando uma plataforma de comunicação entre as diversas metodologias que compõem a ferramenta computacional de Avaliação da Segurança Dinâmica (ASDIN) implementada. A estrutura de objetos que representa os elementos físicos armazena e disponibiliza o estado do sistema para todas as aplicações que operam sobre o mesmo, aumentando assim a precisão numérica e diminuindo o tempo de processamento, já que estes dados encontram-se disponíveis em tempo de execução na memória do computador.

A documentação da BCOO é de fácil entendimento e compreensão, pois utiliza a notação gráfica da *Unified Modeling Language* (UML), padronizada pelo *Object Management Group* (OMG). Uma fácil compreensão das estruturas de classes aumenta a eficiência quando do desenvolvimento de softwares de grande porte, pois facilita a visualização do projeto como um todo.

A implementação das aplicações é facilitada pela utilização do *framework* OOTPS, que fornece as diretrizes necessárias para a execução das mesmas, poupando tempo de desenvolvimento e permitindo que o projetista concentre-se na resolução dos problemas específicos de implementação da sua metodologia. Este *framework* integra as

diversas abstrações que fazem parte da BCOO, incorporando-as em uma sólida ferramenta amplamente testada no âmbito deste trabalho.

6.1 Contribuições Principais do Trabalho

- Modelagem e implementação de aplicações de cálculo de Fluxo de Potência:

Duas aplicações para o cálculo de Fluxo de Potência não-linear foram projetadas utilizando-se a BCOO, e implementadas em linguagem de programação C++. O processo de modelagem destas aplicações foi facilitado pelo fato da formulação do problema de Fluxo de Potência não-linear adequar-se bem às estruturas da BCOO e das interfaces funcionais, descritas no Capítulo 3. Na implementação das metodologias foi utilizado o *framework* OOTPS, o qual realiza todo o processo de leitura de dados, criação dos objetos que representam os elementos físicos e a montagem da rede elétrica, bastando ao desenvolvedor ler ou transferir estes dados para as interfaces funcionais da sua metodologia. Estas aplicações foram validadas tanto numericamente quanto em termos de desempenho computacional (tempo de execução total em torno de dois segundos para o Sis1916) comparando-se os resultados com os do software ANAREDE do CEPEL (1999).

- Modelagem e implementação de uma aplicação de Seleção e Classificação de Contingência Críticas:

Uma aplicação de avaliação global da estabilidade transitória, seleção e classificação de contingências, foi modelada seguindo os preceitos da filosofia para o desenvolvimento de softwares para SEE, descrita neste trabalho. Códigos desenvolvidos em linguagens estruturadas, já devidamente testados e validados, foram reutilizados. A implementação desta aplicação comprovou a capacidade da BCOO em incorporar adequadamente desenvolvimentos realizados em linguagens estruturadas. A reutilização de códigos já desenvolvidos e testados no setor elétrico é de fundamental importância para a validação da BCOO como um ambiente de desenvolvimento integrado de ferramentas para o setor.

- Modelagem e implementação de uma ferramenta computacional de Avaliação e Melhoria da Segurança Dinâmica (ASDIN):

Como principal contribuição deste trabalho cita-se o desenvolvimento de uma aplicação de Avaliação e Melhoria da Segurança Dinâmica, sob a nova filosofia de projeto de softwares para SEE. Esta aplicação engloba metodologias de seleção e classificação de contingências críticas, simulação no domínio do tempo com modelagem detalhadas e avaliação automática da estabilidade transitória, melhoria da segurança dinâmica (método da Direção S Modificado) e um módulo de fluxo de potência, integradas em uma só ferramenta computacional. Estas metodologias estão conectadas e trocam informações pela estrutura de objetos físicos disponibilizada pela BCOO. Destaca-se a utilização neste trabalho de Diagramas de Casos de Uso para a correta delimitação das abstrações que compõem a aplicação. Uma classe foi criada para o gerenciamento dos objetos responsáveis pela execução de cada uma das metodologias que compõem a aplicação. A BCOO suportou adequadamente o desenvolvimento desta aplicação de grande porte para análise de SEE, oferecendo todos os recursos necessários para sua implementação.

As simulações realizadas com esta ferramenta computacional validaram a estrutura de objetos físicos disponibilizada pela BCOO, a qual disponibiliza um ambiente de comunicação entre os diversos objetos de cada metodologia. A ferramenta foi validada, tanto em termos numéricos quanto em termos de desempenho computacional, usando-se sistemas reais e de grande porte e tomando-se como padrão o protótipo desenvolvido em SOUZA (1999).

- Modelagem e implementação de uma ferramenta de Seleção e Classificação de Contingências Críticas, utilizando Processamento Paralelo:

Realizou-se a modelagem e implementação de uma aplicação para a Avaliação Global da Estabilidade Transitória, segundo a nova filosofia para projeto de softwares para SEE e utilizando técnicas de processamento paralelo. Esta etapa representou uma investigação preliminar da utilização dos recursos da BCOO na execução de programas paralelos, com resultados amplamente satisfatórios sob os aspectos de vialibilidade e desempenho computacional.

Complementando o trabalho implementou-se um sistemas de processamento de alto desempenho (PAD) e baixo custo baseado em cluster de microcomputadores. O baixo custo deste sistema deve-se ao uso de uma arquitetura de hardware para PCs e

software livre. A aplicação de seleção e classificação de contingências críticas foi instalada e operada neste ambiente, validando-o.

6.2 Sugestão para Trabalhos Futuros

No sentido de dar continuidade e possibilitar a melhoria dos trabalhos desenvolvidos nesta dissertação, são apresentadas as seguintes sugestões para futuros trabalhos, divididas em questões específicas e questões gerais:

Questões Específicas:

- adicionar à classe *C_SIMSP* a capacidade de realizar a simulações no domínio do tempo usando metodologias de passo variável;
- substituir o método da Direção S de melhoria da segurança dinâmica por métodos mais eficientes, que façam uso de técnicas de otimização;
- buscar e implementar ferramentas de controle de versões e sincronização de diagramas de classe com o código desenvolvido.
- em termos de clusters, investigar os benefícios do uso de rede dual (uma rede para dados e outra para controle)

Questões Gerais:

- continuar as pesquisas para a adequação da BCOO a ambientes de processamento paralelo para a execução de programas que exigem elevada demanda computacional em SEE;
- projetar e implementar outras metodologias de análise de SEE seguindo a proposição de AGOSTINI (2002a);
- pesquisar melhores soluções em termos de armazenamento de dados e interfaces gráficas utilizando os recursos da BCOO.

REFERÊNCIAS BIBLIOGRÁFICAS

AGOSTINI, M. N.; DECKER, I. C.; SILVA, A. S.; 2000. Desenvolvimento e Implementação de uma Base Computacional Orientada a Objetos para Aplicações em Sistemas de Energia Elétrica. In. CONGRESSO BRASILEIRO DE AUTOMÁTICA – CBA (13.: Set: Florianópolis, SC). *Anais*. Florianópolis. P. 1850-1856.

AGOSTINI, M. N.; 2002a. *Nova Filosofia para o Projeto de Softwares para Sistemas de Energia Elétrica Usando Modelagem Orientada a Objetos*. Florianópolis. Tese (Doutorado em Engenharia Elétrica) - Centro Tecnológico, Universidade Federal de Santa Catarina.

AGOSTINI, M. N.; FERREIRA, J. M. F.; DECKER, I. C. et al.; 2002b. Object Oriented Matrix Structure for the Development of Computing Tools in Electric Power Systems. In. SYMPOSIUM OF SPECIALISTS IN ELECTRIC OPERATIONAL AND EXPANSION PLANNING - SEPOPE (8. : Maio : Brasília, DF). *Anais*. Brasília.

ARAUJO, L. R.; PEREIRA, J. L. R.; 2000. Solução de Redes Elétricas de Grande Porte, Usando Programação Orientada a Objetos. In. CONGRESSO BRASILEIRO DE AUTOMÁTICA – CBA (13.: Set : Florianópolis, SC). *Anais*. Florianópolis. P. 604-609.

ARAUJO, L. R; GARCIA, P. A. N., PEREIRA, J. L. R., et. al.; 2002. Modelagem Orientada a Objetos Aplicada na Solução de Programas de Distribuição. In. CONGRESSO BRASILEIRO DE AUTOMÁTICA – CBA (2. : Set : Natal, RN). *Anais*. Natal.

ATANACKOVIC, D.; McGILLIS, D.; GALIANA, F. D.; 1998. A New Tool for Substation Design. *IEEE Transactions on Power Systems*, New York, v. 13, n. 4 (Nov.), p. 1500-1506.

BAKER, M.; BUYA, R.; 1999. Cluster Computing at a Glance. Hight Performance Cluster Computing: Architectures and Systems. Prentice Hall PTR. Disponível em: http://www.csse.monash.edu.au/~rajkumar/cluster/index.html

BALLANCE, J. W.; BHARGAVA, B.; RODRIGUEZ, G. D.; 2003. Monitoring Power System Dynamics using Phasor Measurement Technology for Power System Dynamic Security Assessment. In. IEEE BOLOGNA POWER TECH CONFERENCE – POWERTECH 23.: Jun.: Bologna, Itália.

BECKER, D.; FALK, H.; GILLERMAN, J.; et al.; 2000. Standards-Based Approach Integrates Utility Applications. *IEEE Computer Applications in Power*, New York, v. 13, n. 4 (Oct.), p. 13-20.

BERRY, T.; 2000. Standards for Energy Management System Application Program Interfaces, International Conference on Eletric Utility Deregulation and Restructuring and Power Tecnologies.

BOOCH,G.; 1998. Object Solutions: Managing the Object-Oriented Protect. Menlo Park: Addison-Wesley.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I.; 2000. *UML - Guia do Usuário*: O mais avançado tutorial sobre Unified Modeling Language (UML), elaborado pelos próprios criadores da linguagem. Rio de Janeiro : Campus.

BRADLEY, M. E.; BUSHNELL, M. J.; MACLEAN, S. I.; 1999. Object-Oriented Creation of Fault Sequences for Online Transient Stability Analysis. In: POWER SYSTEMS COMPUTATION CONFERENCE (13. : Jun. : Trondheim). *Proceedings*. Trondheim. p. 654-660.

BRITTON, J.; 1992. An Open, Object-Based Model as the Basis of an Architecture for Distribution Control Centers. *IEEE Transactions on Power Systems*, New York, v. 7, n. 4 (Nov.), p. 1500-1508.

CEPEL – Centro de Pesquisas de Energia Elétrica; 1999. Programa de Análise de Redes – ANAREDE : Manual do Usuário. Rio de Janeiro, RJ.

CIGRE Task Force; 1992. Assessment of Practical Fast Transient Stability Methods, Convener S. Greves.

COSTA, L.; 1997. *Melhoria da Segurança Dinâmica Utilizando Técnicas de Otimização. Florianópolis. Dissertação de Mestrado* - Centro Tecnológico, Universidade Federal de Santa Catarina.

COX, J. B.; 1986. Object-Oriented Programming. Addison-Wesley Publishing Company. Massachusetts.

DE VOS, A.; WIDERGREN, S. E.; ZHU J.; 2001. XML for CIM Model Exchange. Innovative Computing for Power - Electric Energy Meets the Market 22nd *IEEE Power Engineering Society International Conference*, 20-24.

DEMAREE, K. et al.; 1994. An On-line Dynamic Security Analysis System Implementation, *IEEE Transactions on Power Systems*, v. 9, no 4, pp. 1716-1722.

DEUTSCH, L. P.; Design reuse and frameworks in the Smalltalk-80 system. In Ted J. Biggerstaff and Alan

DY-LIACCO T.; 2002. *Control Centers Are Here To Stay*. *IEEE Computer Applications in Power*, New York, v. 15, n. 4 (Oct.), p. 18-23.

FINARDI E. C.; 1999. *Planejamento da Operação Hidrotérmica Utilizando Processamento de Alto Desempenho. Florianópolis*. *Dissertação de Mestrado.* Centro Tecnológico, UFSC.

FLINN, D.; DUGAN, R. C.; 1992. A Database for Diverse Power System Simulation Applications. *IEEE Transactions on Power Systems*, New York, v. 7, n. 2, p. 784-790.

FOLEY, M.; BOSE, A.; MITCHELL, W. et al.; 1993. An Object Based Graphical User Interface for Power Systems. *IEEE Transactions on Power Systems*, New York, v. 8, n. 1, p. 97-104.

FOLEY, M.; BOSE, A.; 1995. Object-Oriented Online Network Analysis. *IEEE Transactions on Power Systems*, New York, v. 10, n. 1, p. 125-132.

FONSECA, L. G. S., DECKER, I. C.; 1985. Iterative Algorithm for Critical EnergyDetermination in Transient Stability of Power System, *Proceedings of the IFAC Symposium Planning & Operation in Electric Energy Systems*, Rio de Janeiro, RJ, Brasil, pp. 483 - 489.

FONSECA, L. G. S.; PEDROSO, A. S.; 1990 Correção da Segurança Dinâmica em Sistemas de Potência de Grande Porte Via Redespacho, *VIII CBA - Congresso Brasileiro de Automática*, Belém, PA, Brasil.

FUERTE-ESQUIVEL, C. R.; ACHA, E.; TAN, S. G. et al.; 1998. Efficient Object Oriented Power Systems Software for the Analysis of Large-Scale Networks Containing FACTS Controlled Branches. *IEEE Transactions on Power Systems*, New York, v. 13, n. 2 (May), p. 464-472.

GAMMA, E.; HELM, R.; JOHNSON, R. et al.; 2000. *Padrões de Projeto* : Soluções Reutilizáveis de Software Orientado a Objetos. Porto Alegre : Bookman.

HAKAVIK, B.; HOLEN, A. T.; 1994. Power System Modelling and Sparse Matrix Operations Using Object-Oriented Programming. *IEEE Transactions on Power Systems*, New York, v. 9, n. 2 (May), p. 1045-1051.

HANDSCHIN, E.; HEINE, M.; KÖNIG D. et al.; 1998. Object-Oriented Software Engineering for Transmission Planning in Open Access Schemes. *IEEE Transactions on Power Systems*, New York, v. 13, n. 1 (Feb.), p. 94-100.

JOHNSON R. E.; FOOTE B.; 1988. *Designing reusable classes. Journal of Object-Oriented Programming*. V. 1, p. 22-35.

KLEIN, S. A.; 2003. *Danger and Opportunity. IEEE Power & Energy*. V. 1, n. 1 (May/June), p. 80.

KUNDUR, P.; MORISON, K.; 1998. On-line Dynamic Security Assessment of PowerSystems, *VI SEPOPE – Symposium of Specialists in Electric Operational and Expansion Planning*, Salvador, BA, Brazil.

LA SCALA, M., et al.; 1996. A Qualitative Approach to the Transient Stability Analysis, *IEEE Transactions on Power Systems*, v. 11, no 4, pp. 1996-2002.

LEE, S. J.; LIM, S.; AHN, B. S.; 1998. Service Restoration of Primary Distribution Systems Based on Fuzzy Evaluation of Multi-Criteria. *IEEE Transactions on Power Systems*, New York, v. 13, n. 3 (Aug.), p. 1156-1163.

LOSI, A.; RUSSO, M.; 2000. An Object Oriented Approach to Load Flow in Distribution Systems. In: 2000 IEEE PES SUMMER MEETING (July : Seattle). *Proceedings*. Seattle.

MANZONI, A.; 1996. *Desenvolvimento de um Módulo Dinâmico para Simuladores de Ensino e Treinamento em Sistemas de Energia Elétrica Usando Programação Orientada a Objetos*. Florianópolis. Dissertação (Mestrado em Engenharia Elétrica) - Centro Tecnológico, Universidade Federal de Santa Catarina.

MANZONI, A.; SILVA, A. S.; DECKER, I. C.; 1999. Power Systems Dynamics Simulation Using Object-Oriented Programming. *IEEE Transactions on Power Systems*, New York, v. 14, n. 1 (Feb.), p. 249-255.

MIAO, H.; SFORNA, M.; LIU, C. C.; 1996. A New Logic-Based Alarm Analyzer for Online Operational Environment. *IEEE Transactions on Power Systems*, New York, v. 11, n. 3 (Aug.), p. 1600-1606.

MONTICELLI, A.; 1983. *Fluxo de Carga em Redes de Energia Elétrica*. São Paulo : Edgard Blücher.

NEYER, A. F.; WU, F. F.; IMHOF, K.; 1990. Object-Oriented Programming for Flexible Software: Example of a Load Flow. *IEEE Transactions on Power Systems*, New York, v. 5, n. 3 (Aug.), p. 689-696.

OLIVEIRA F., D.; GALIANA, F. D.; 1996. A Model for the Planning of Electric Energy Systems Including Exergetic Considerations. *IEEE Transactions on Power Systems*, New York, v. 11, n. 2 (May), p. 675-682.

PANDIT, S.; SOMAN, S. A.; KHAPARDE, S. A.; 2000. Object-Oriented Design for Power System Applications. *IEEE Computer Applications in Power*, New York, v. 13, n. 4 (Oct.), p. 43-47.

PANDIT, S.; SOMAN, S. A.; KHAPARDE, S. A.; 2001a. Object-Oriented Network Topology Processor. *IEEE Computer Applications in Power*, New York, v. 14, n. 2 (April), p. 42-46.

PANDIT, S.; SOMAN, S. A.; KHAPARDE, S. A.; 2001b. Design of Generic Direct Sparse Linear System Solver in C++ for Power System Analysis. *IEEE Transactions on Power Systems*, New York, v. 16, n. 4 (Nov.), p. 647-652.

SCHIAVONE, G.A.; TRACY, J.; PALANIAPPAN, R.; 2000. Preliminary investigations into distributed computing applications on a Beowulf cluster, Fourth IEEE International Workshop on , 24-26.

SILVA, M. P.; SARAIVA, J. T.; SOUZA, A. V.; 2000. A Web Browser Based DMS – Distribution Management System. In: 2000 IEEE PES SUMMER MEETING (July : Seattle).*Proceedings*. Seattle.

SHEIDT, J. L.; MILLER, S. E.; KLEIN S. A.; ENNS, M. K.; SAVALESCU, S. C.; 1987. Future Role of High Level Languages in Power System Control Centers. In.: IEEE/PES Summer Meeting. Relatório preparado pelo IEEE Working Group on Power System Control (WG 73-3)

SOUZA, A.; 1999. *Avaliação da Segurança Dinâmica Usando Modelos Detalhados e Processamento Distribuído*. Dissertação de Mestrado, UFSC, Florianópolis, SC, Brasil.

SOUZA, A.; DECKER, I. C.; AGOSTINI, M. N.; BETTIOL, A. L. Sistema Computacional Baseado em Cluster de Microcomputadores para Avaliação e Melhoria da Segurança Dinâmica On-line. Anais do VII Simpósio de Especialistas em Planejamento da Operação e Expansão Elétrica (VIII SEPOPE), Brasília, DF, Maio 2002.

STOTT, B.; ALSAÇ O.; 1974. Fast Decoupled Load Flow. In.: IEEE Transactions on Power Applications and Systems, New York. P. 859-869.

STROUSTRUP, B.; 1997. *The C++ Programming Language*. 3. ed. Reading: Addison-Wesley.

VAAHEDI, E.; CHANG, Y. A.; SASAN M. et al.; 2001. A Future Application Environment for B.C. Hydro's EMS. IEEE Transactions on Power Systems, v. 16, n. 1.

VANTI, M. R. V.; 2003. *Melhoria da Segurança Dinâmica por Otimização e Algoritmos para Programação Não-Linear*. Florianópolis. Tese (Doutorado em Engenharia Elétrica) - Centro Tecnológico, Universidade Federal de Santa Catarina.

WANG, X.; Schultz N. N.; 2000. Development of Three-Phase Distribuition Power Flow Using Common Information Model, In.: 2000 POWER ENGINEERING SOCIETY SUMMER MEETING,. IEEE, Volume: 4 , 16-20 July Page(s): 2320 - 2325 vol. 4

WHITE, E.; MALLOY, R.; 1986. Object-Oriented Programming. Byte, Series of Articles on Object-Oriented Programming. August pp. 137-235.

XINGPING W.; ZHANG Y.; WANG, X.; 2002. A New Generation EMS. Proceedings. PowerCon 2002. International Conference on , Volume: 1 , 13-17 Oct. 2002

ZHANG, Y., et al.; 1997. SIME: A Hybrid Approach to Fast Transient Stability Assessment and Contingency Selection, *Electrical Power & Energy Systems*, v. 19, no 3, pp. 195-208.

ZHOU, E. Z.; 1996. Object-Oriented Programming, C++ and Power System Simulation. *IEEE Transactions on Power Systems*, New York, v. 11, n. 1 (Feb.), p. 206-215.