UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

NOVA FILOSOFIA PARA O PROJETO DE SOFTWARE PARA SISTEMAS DE ENERGIA ELÉTRICA USANDO MODELAGEM ORIENTADA A OBJETOS

Tese submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Doutor em Engenharia Elétrica.

MARCELO NEUJAHR AGOSTINI

Florianópolis, novembro de 2002.

NOVA FILOSOFIA PARA O PROJETO DE SOFTWARE PARA SISTEMAS DE ENERGIA ELÉTRICA USANDO MODELAGEM ORIENTADA A OBJETOS

MARCELO NEUJAHR AGOSTINI

'Esta tese foi julgada adequada para a obtenção do Título de Doutor em Engenharia Elétrica, Área de Concentração em Planejamento de 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. Aguinaldo Silveira e Silva, Ph.D.
Co-orientador

Prof. Edson Roberto de Pieri, Dr.
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. José Luiz Rezende Pereira, Ph.D.

Prof. Carlos Eduardo Pereira, D.Ing.

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

AGRADECIMENTOS

Agradeço em primeira instância aos meus orientadores, Prof. Ildemar Cassana Decker e Prof. Aguinaldo Silveira e Silva, não somente pela excelente formação profissional que sempre fizeram questão de compartilhar comigo, mas também pela amizade e confiança a mim dedicadas durante estes anos.

Agradeço aos membros da banca examinadora, Prof. José Luiz Rezende Pereira, Prof. Carlos Eduardo Pereira, e Prof. Arlan Luiz Bettiol, os quais contribuíram de forma inequívoca, não somente ao conteúdo em si desta tese, mas também à minha formação.

Agradeço também a todos os professores e colegas do PPGEEL da UFSC, com os quais trabalhei e convivi desde minha chegada ao Programa. Agradeço em especial aos colegas Alessandro Manzoni, João Marco Francischetti Ferreira e Daniel Dotta pelas sempre excelentes discussões e contribuições a este trabalho, em diferentes momentos. Aos colegas Adriano de Souza, Erlon Cristian Finardi, Hugo Gil Congote, Marcos Keller Amboni, Ederson Silveira Costa, Guilherme Belloli Réos, e a tantos outros que no momento peco por não explicitar seus nomes, pela convivência e amizade durante estes anos, que certamente trouxeram também, direta ou indiretamente, grandes contribuições à minha formação pessoal. Ao Laboratório de Planejamento de Sistemas de Energia Elétrica (LabPlan) e a seu corpo docente, que desde o início do meu curso apostaram em minha capacidade, dando-me sempre totais condições para meu desenvolvimento profissional e pessoal.

Agradeço de forma carinhosa à minha namorada e companheira Denise, que compartilhou comigo grande parte deste trabalho, sempre apoiando incondicionalmente as minhas decisões.

Agradeço também carinhosa e respeitosamente aos meus pais, Carlos Gilberto Agostini e Maria Malvina Neujahr Agostini, assim como a todos os meus familiares, especialmente minha *dinda* Noemy Neujahr, pela formação pessoal sólida e justa que sempre me proporcionaram, assim como pelo suporte material, que com certeza foram imprescindíveis para a realização deste trabalho.

Agradeço finalmente à Capes, pelo suporte financeiro ao meu doutoramento.

Resumo da Tese apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Doutor em Engenharia Elétrica.

NOVA FILOSOFIA PARA O PROJETO DE SOFTWARE PARA SISTEMAS DE ENERGIA ELÉTRICA USANDO MODELAGEM ORIENTADA A OBJETOS

Marcelo Neujahr Agostini

Novembro / 2002

Orientador: Prof. Ildemar Cassana Decker, D.Sc.
Co-orientador: Prof. Aguinaldo Silveira e Silva, Ph.D.
Área de Concentração: Sistemas de Energia Elétrica.
Palavras-chave: Modelagem de Sistemas de Energia Elétrica, Modelagem Orientada a Objetos.
Número de páginas: 131.

Neste trabalho é apresentada uma nova filosofia para o desenvolvimento de softwares para Sistemas de Energia Elétrica (SEE), baseada no paradigma de orientação a objetos. Ela engloba desde aspectos gerais, tais como uma delimitação adequada do escopo de projetos de softwares, e uma separação de conceitos através de diferentes abstrações do sistema, até questões específicas de implementação de códigos, passando por princípios como a mantenabilidade, expansibilidade e robustez das estruturas de dados (com conseqüente redução de custos nessas tarefas), reutilização de códigos, e eficiência computacional. Uma base computacional orientada a objetos para a representação de SEE é também desenvolvida, servindo como núcleo para futuros desenvolvimentos. As estruturas de classes da base possuem mecanismos que permitem futuras expansões, no intuito de se representar as mais diversas instâncias dos SEE. Elas prevêem a representação não somente de elementos físicos que formam o sistema, mas também de conceitos abstratos, tais como metodologias de análise e síntese aplicadas a SEE, ferramentas computacionais de apoio ao planejamento e a operação destes sistemas, e também módulos com funções específicas (operações computacionais e matemáticas), envolvidos na construção destas ferramentas. Metodologias de análise e síntese são organizadas segundo estruturas de classes, as quais conectam-se às estruturas representativas dos elementos físicos do sistema elétrico. Duas aplicações são projetadas e implementadas em linguagem de programação C++. Estas aplicações, um fluxo de potência linearizado e uma simulação dinâmica com modelagem detalhada, são agrupadas em um protótipo de ferramenta computacional. O protótipo exemplifica a construção de ferramentas computacionais para SEE, operando sob um ambiente integrado, e desenvolvidas sob a filosofia de projeto de software proposta neste trabalho.

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

NEW PHILOSOPHY FOR THE PROJECT OF ELECTRIC POWER SYSTEM SOFTWARE USING OBJECT ORIENTED MODELING

Marcelo Neujahr Agostini

November / 2002

Advisor: Prof. Ildemar Cassana Decker, D.Sc.
Co-advisor: Prof. Aguinaldo Silveira e Silva, Ph.D.
Area of Concentration: Electric Energy Systems.
Keywords: Power Systems Modeling, Object Oriented Modeling.
Number of Pages: 131.

In this work a new philosophy for the software development for Electric Power Systems (EPS), based on the Object Oriented Modeling (OOM) paradigm, is presented. General aspects, such as an appropriate scope delimitation of software's projects and a separation of concepts through different abstractions of the system, as well as specific subjects such as code implementation, the concepts of maintainability, expandability and data structures robustness (with consequent reduction of costs in those tasks), code reutilization, and computational efficiency, are discussed. An oriented object computational base for the representation of EPS is also developed, serving as a kernel for future developments. The class hierarchic of that base has mechanisms that allow future expansion, aiming to represent a larger spectrum of concepts associated to EPS. This class structure is suited not only to represent physical elements that form the system, but also abstract concepts, such as analysis and synthesis methodologies applied to EPS. Besides, it supports computational tools for EPS planning and operation, and also modules with specific functions (computational and mathematics operations), used in the construction of these tools. Analysis and synthesis methodologies are organized in class structures connected to the representative structures of the physical elements of the power system. Two applications are designed and implemented in C++ programming language. These applications, a linearized power flow and a dynamic simulation module, form a prototype of a computational tool for EPS analysis. This prototype illustrates the construction of computational tools for EPS, operating under an integrated environment, and being developed applying the philosophy of software design proposed in this work.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS

CCAPI – Control Center Application Program Interface

CER – Compensador Estático de Reativos

CIM – Common Information Model

EPRI – Electric Power Research Institute

ESP – Estabilizador de Sistema de Potência

FACTS – Flexible AC Transmission Systems

LT – Linha de Transmissão

LTC – Load Tap Changer

MOO – Modelagem Orientada a Objetos

MS – Máquina Síncrona

OIS – Operador Independente do Sistema

OMT – Object Modeling Technique

OO – Orientado a Objetos

OOTPS – Object Oriented Tool for Power Systems

RAT – Regulador Automático de Tensão

SIN – Sistema Interligado Nacional

SEE – Sistema de Energia Elétrica

SSL – Solução de Sistemas Lineares

TCSC – Thyristor Controlled Series Capacitor

TRAFO – Transformador

UG – Unidade de Geração

UML – Unified Modeling Language

UPFC – Unified Power Flow Controller

CAPÍTULO 1

1. INTRODUÇÃO

O setor de energia elétrica mundial atravessa atualmente uma fase peculiar de sua história, na qual experimenta uma mudança de ambiente, tanto operacional quanto organizacional. Está-se migrando de um ambiente centralizado e regulamentado, para um novo ambiente descentralizado, onde as empresas do setor têm suas estruturas alteradas, e passam a desempenhar novos papéis, surgindo a figura de novos agentes.

Surge a figura dos *agentes de produção* (geradores de energia), que participam do novo ambiente competitivo visando o *lucro*, como em qualquer outro negócio. Surgem também as figuras dos *comercializadores* e dos *consumidores livres*, estes últimos tendo a opção de escolher o produtor que lhe fornecerá a energia. A energia elétrica passa a ser considerada uma *mercadoria*, comercializada através de transações econômicas entre produtores, comercializadores e consumidores (DE TUGLIE et al., 1999).

A *transmissão* da energia é considerada neste novo ambiente como um *serviço*, gerenciado por um *operador independente do sistema* (OIS). O operador tem como função principal gerenciar a transmissão da energia, comercializada entre produtores e consumidores, mantendo a qualidade em seus pontos de consumo, e o nível de segurança do sistema frente a possíveis falhas, entre outros.

Mudam também as características operacionais do sistema. A rede de transmissão passa a operar mais próxima de seus limites, tanto em relação às capacidades de transmissão, quanto em níveis de segurança. As ações de controle para se manter o sistema em níveis adequados de segurança, tendem a ser vistas como um *serviço ancilar*, com a função de dar um suporte essencial ao funcionamento do sistema elétrico como um todo. Estes serviços devem ser quantificados economicamente, de forma que se tornem atrati-

vos para os agentes que agora atuam no mercado de energia (SHIRMOHAMMADI et al., 1996; ALVARADO, 1996; SOUZA et al., 2002).

A *concorrência* entre os agentes é uma questão a ser observada com cuidado. No novo ambiente, as informações sobre as empresas, principalmente no que se refere a geradores independentes, não estão mais disponíveis abertamente, o que dificulta o processo de planejamento do sistema (DE TUGLIE et al., 1999). Aspectos regulatórios devem ser definidos, para que uma concorrência plena seja mantida entre os agentes.

Frente a tudo isto, os agentes necessitam cada vez mais de mecanismos modernos e confiáveis para a análise do sistema, que representem de forma eficiente suas novas características, tanto técnicas quanto econômicas. Inúmeras áreas de pesquisa devem ser fomentadas, desde áreas técnicas relativas a sistemas de energia elétrica, passando por pesquisas que envolvem questões econômicas juntamente com aspectos técnicos, até áreas como a *engenharia de software*.

Um ambiente competitivo como o que se apresenta para o setor elétrico tem características bastante dinâmicas, onde mudanças ocorrem rapidamente, e devem ser assimiladas também rapidamente pelos seus agentes (HANDSCHIN et al., 1998). Questões como o surgimento de novos equipamentos (e por conseqüência novos modelos computacionais), aperfeiçoamentos de modelos já existentes e o surgimento de novas metodologias de análise (incorporando aspectos técnicos e econômicos), devem contar com um suporte eficiente para a implementação destas mudanças sob a forma de ferramentas computacionais disponíveis ao setor (ZHOU, 1996; MANZONI et al., 1999; AGOSTINI et al., 2000).

Associada a estas necessidades está a inerente complexidade de problemas relacionados a SEE. Estes sistemas são na sua maioria interligados, originando sistemas maiores e mais complexos, considerados de grande porte. Uma grande quantidade e variedade de equipamentos (alguns milhares) encontram-se conectados aos sistemas, devendo ser representados de maneira adequada a cada tipo de estudo. Diversas metodologias de análise e síntese podem ser aplicadas nestes estudos, fazendo com que a integração entre estas metodologias, e entre elas e os SEE sob estudo, torne-se mais e mais complexa à medida que novos equipamentos são desenvolvidos e conectados aos sistemas, e novas metodologias são desenvolvidas. Os esforços para expansões e manutenções nas ferramentas computacionais do setor elétrico estão diretamente relacionados a esta complexidade.

1.1. Os Softwares de Análise de SEE

Os softwares de análise utilizados atualmente na indústria de energia elétrica como apoio às áreas de planejamento e operação são, em geral, muito eficientes. Apesar disso, estes softwares possuem características que, frente ao novo ambiente competitivo que se apresenta para o setor, necessitam ser revisadas (NEYER et al., 1990; ZHOU, 1996; ZHU et al., 1997). Pode-se citar, entre outras:

- → estruturas de dados vulneráveis a manutenções;
- → dificuldades para a integração entre módulos;
- → investimentos elevados para manutenções e atualizações;
- → dificuldades no gerenciamento de módulos desenvolvidos por diferentes equipes e fabricantes;
- → utilizam como entrada de dados arquivos em formatos muitas vezes de difícil compatibilização com novas ferramentas ou novas versões de ferramentas já existentes.

Sendo programas baseados em metodologias convencionais de projeto de software, a principal atenção é voltada para a funcionalidade do sistema (modelagem e programação orientada a funções) (ZHU et al., 1997). As estruturas de dados não possuem funcionalidades claras, existindo apenas para o armazenamento dos dados dos SEE, em tempo de execução do software.

Muitas vezes, alterações em uma porção específica do código produzem *efeitos colaterais* em outras rotinas do programa (ZHOU, 1996). São efeitos extremamente inconvenientes, que acabam causando problemas adicionais, além daqueles que motivaram inicialmente o processo de manutenção. A solução destes problemas tende a ser complicada, e muitas vezes leva à reescrita de grandes porções de código.

1.2. Requisitos Para uma Nova Geração de Softwares

Apesar do sucesso e qualidade dos softwares existentes, características como as citadas anteriormente constituem obstáculos frente aos novos desafios colocados aos SEE atualmente. As mudanças intrínsecas ao novo cenário operacional, a introdução da nova geração de equipamentos baseados em eletrônica de alta potência (Flexible AC Transmission Systems – FACTS), e mesmo o aumento na complexidade dos modelos matemáticos

representativos do sistema, assim como a inclusão de novos modelos, são de difícil incorporação nas ferramentas computacionais atualmente disponíveis.

Tendo em vista estes aspectos, propõe-se uma nova geração de softwares, com as seguintes características:

→ estrutura de dados estável, 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 no gerenciamento de módulos desenvolvidos por diferentes equipes de trabalho;

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

Para este novo conjunto de softwares, tem-se como premissa básica a construção da estrutura de dados representativa das mais diversas instâncias dos SEE. A partir do momento que se tem uma estrutura estável, que represente de forma eficiente os elementos dos SEE, entende-se que seja natural o projeto e a implementação de metodologias de análise e síntese para estes sistemas, atuando sobre essa estrutura.

Alterações nas metodologias tendem a produzir impactos menores em outras porções de código. A inclusão de novos modelos de elementos também se torna mais natural, desde que a estrutura de dados tenha seus elementos devidamente *encapsulados* (cada elemento deve ser conhecido externamente pela sua *interface*, sendo que seu funcionamento interno é completamente independente do restante do sistema).

Inclui-se nestes aspectos tanto a modelagem computacional das ferramentas disponíveis ao setor elétrico, como também os bancos de dados para armazenagem dos dados dos SEE. Procura-se com isso torná-los de fácil adaptação a manutenções nos códigos que os acessam, ainda assim mantendo total compatibilidade com versões anteriores desses códigos.

Uma análise dos requisitos apresentados acima leva naturalmente a se considerar a *Modelagem Orientada a Objetos* (MOO) para a construção desta nova geração de programas. Com a MOO torna-se natural a construção da estrutura representativa do sistema, através de estruturas hierárquicas de classes. Os elementos do SEE e seus modelos são representados por *objetos*, devidamente encapsulados dentro da estrutura.

Esta nova abordagem propicia uma redução de custos nas atividades de gerenciamento, desenvolvimento, manutenção e atualização de softwares de grande porte, requeridos pela indústria de energia elétrica hoje.

1.3. Objetivos do Presente Trabalho

Nos últimos anos vários trabalhos têm surgido tratando da aplicação de técnicas de MOO no desenvolvimento de programas para o setor de energia elétrica. A maioria trata de aplicações em fluxo de potência (NEYER et al., 1990; ZHOU, 1996), encontrando-se também aplicações na área de interfaces gráficas homem - máquina (FOLEY et al., 1993; LI et al., 1993), simulação da dinâmica (VANTI, 1994; MANZONI et al., 1999), sistemas de distribuição (BRITTON, 1992; ZHU et al., 1997), planejamento (OLIVEIRA et al., 1996; HANDSCHIN et al., 1998), solução de sistemas lineares (ARAUJO et al., 2000; AGOSTINI et al., 2002), e modelagem computacional genérica de SEE (AGOSTINI et al., 2000; PANDIT et al., 2000).

Seguindo esta tendência, o presente trabalho tem por objetivo principal propor uma nova filosofia de projeto de software para a área de SEE, aplicando-se técnicas de MOO. Para isso, aplicam-se os preceitos da MOO na representação das mais diversas instâncias dos SEE, desenvolvendo-se uma base computacional orientada a objetos, possibilitando a implementação de uma ampla gama de metodologias de análise e síntese na área de SEE. As estruturas de classes modelam computacionalmente tanto os elementos físicos destes sistemas, quanto as metodologias de análise e síntese neles aplicadas, e também as ferramentas computacionais finais para o setor. A modelagem proposta visa facilitar a integração entre módulos desenvolvidos separadamente (normalmente por diferentes equipes), na construção de ferramentas mais complexas.

As estruturas de classes possuem um alto grau de generalização, tal que permita especializações com o objetivo de se incorporar futuros avanços, tanto em termos de novos modelos de equipamentos, quanto de novas metodologias de análise e síntese, bem como a construção de novas ferramentas. Busca-se, assim, minimizar esforços e custos destas atualizações.

1.4. Descrição do Trabalho

O foco principal deste trabalho, que é a aplicação de MOO na modelagem computacional de SEE e no desenvolvimento de suas ferramentas computacionais, é ilustrado na Figura 1-1.

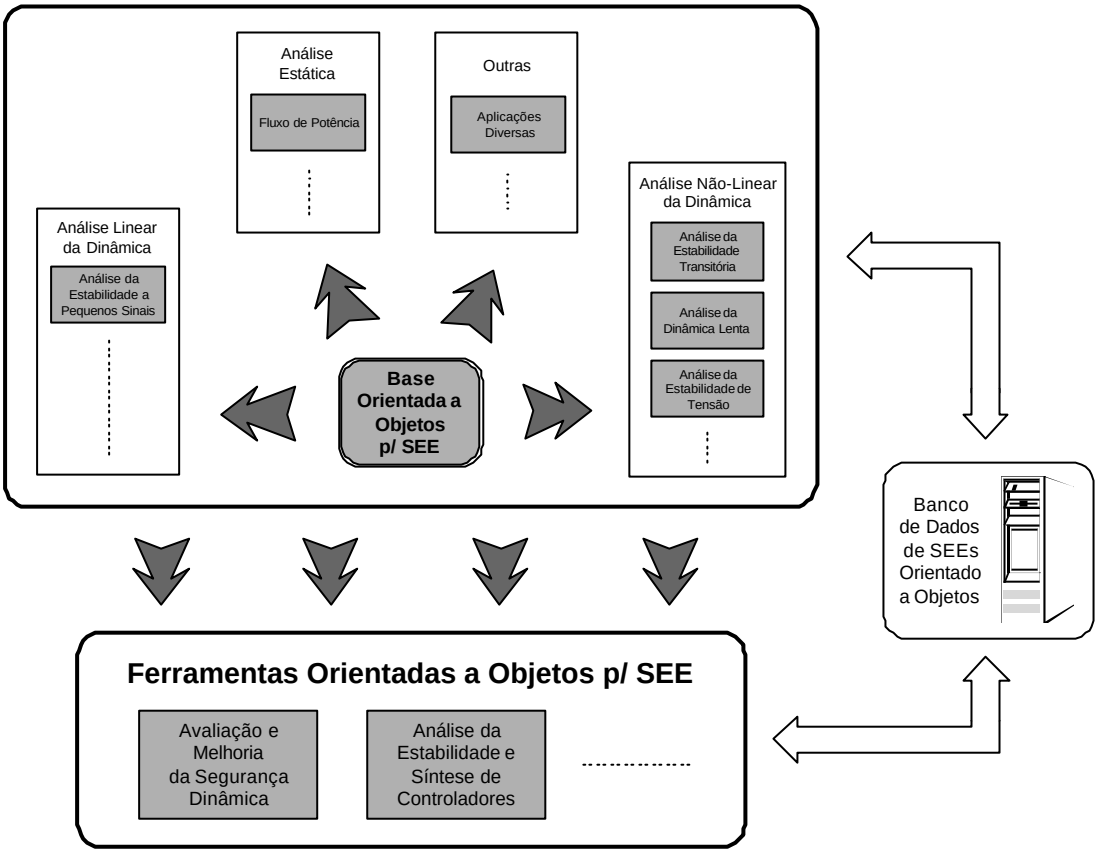


Figura 1-1 – Visualização Geral do Trabalho

No centro desta figura encontra-se representada a base computacional orientada a objetos, cuja função é modelar os mais diversos aspectos dos SEE, servindo de plataforma para a implementação de metodologias como fluxo de potência, análise da estabilidade transitória, etc. Utilizando-se essas aplicações, modeladas e implementadas sob os preceitos da MOO, são construídas ferramentas computacionais, representadas no bloco inferior, e exemplificadas por uma ferramenta para a *Avaliação e Melhoria da Segurança Dinâmica*, e uma para a *Análise da Estabilidade e Síntese de Controladores*. Um banco de dados orientado a objetos armazena os dados dos SEE, comunicando-se de forma integrada com as ferramentas do ambiente.

As ferramentas computacionais são organizadas de forma a atuar em conjunto com a estrutura hierárquica de classes representativa do SEE. Dessa forma, as ferramentas são flexíveis e robustas, permitindo que manutenções exijam esforços mínimos. Elas facilitam também a troca de informações (fluxo de dados) entre si, e com bancos de dados orientados a objetos para o armazenamento de dados de SEE. A compatibilidade entre diferentes versões de uma determinada ferramenta, ou mesmo entre diferentes ferramentas, é facilitada, uma vez que essa compatibilidade se dá através de um banco de dados orientado a objetos, projetado em conformidade com as estruturas de classes existentes em tempo de execução.

É importante destacar a abrangência do tema, no qual pode-se visualizar com clareza a necessidade da incorporação de diversos trabalhos científicos, em diversos níveis (doutorado, mestrado, e outros). Desta forma pretende-se convergir para um ambiente integrado de análise e síntese na área de SEE, completamente orientado a objetos, utilizando-se das vantagens que este paradigma oferece, tais como robustez e flexibilidade da estrutura de dados, e mantenabilidade e reutilização de códigos.

O presente trabalho de tese concentra-se na construção da base computacional representante das mais diversas instâncias dos SEE. Para isso, inicialmente realiza-se um estudo da conceituação da MOO, juntamente com metodologias de projeto disponíveis na bibliografia. Diversos métodos de projeto orientados a objeto são considerados no desenvolvimento do trabalho, com especial atenção ao método *Object Modeling Technique* (OMT) (RUMBAUGH et al., 1994), e às notações gráficas da *Unified Modeling Language* (UML) (BOOCH et al., 2000). Aspectos gerais sobre a aplicação de MOO na representação e projeto de softwares para SEE são também discutidos. Uma revisão bibliográfica de trabalhos já publicados sobre o assunto é realizada.

Em seguida, representam-se as mais diversas instâncias de um SEE segundo estruturas hierárquicas de classes. Os elementos físicos do sistema são classificados em dois grandes grupos: elementos estruturais e elementos de composição. O primeiro grupo representa elementos conectados diretamente às barras do sistema, formando, assim, a sua estrutura básica. Trata-se por elementos de composição, elementos que não estão conectados diretamente a alguma barra, mas quando agrupados de uma certa maneira, dão origem a elementos estruturais.

As funcionalidades dos elementos físicos, as quais são dependentes da metodologia de análise ou síntese que se está aplicando sobre o sistema, são encapsuladas em classes paralelas às classes representativas dos elementos físicos, e denominadas *interfaces funcionais*. Dessa forma isolam-se as características puramente físicas dos elementos,

válidas em qualquer aplicação considerada, das características comportamentais, que dizem respeito a uma ou outra metodologia de análise ou síntese em específico. Utiliza-se para a montagem deste mecanismo o padrão de projeto orientado a objetos *Adapter*, discutido em detalhes no Capítulo 2. Com isso, facilitam-se expansões futuras das estruturas, permitindo-se que novas metodologias e novos elementos sejam facilmente incorporados às estruturas da base computacional.

As metodologias de análise e síntese, sinônimos de *aplicações* neste trabalho, são também organizadas segundo estruturas de classes, as quais conectam-se às estruturas representativas dos elementos físicos do sistema elétrico, atuando sobre elas.

Paralelamente às estruturas de classes relativas às mais diversas instâncias de um SEE, classes representando facilidades computacionais são projetadas e implementadas, sob a forma de pacotes e bibliotecas. Dentre estas figuram classes para leitura e armazenamento de dados, gerenciamento de telas, medidas de tempos de execução, armazenamento e tratamento de matrizes e sistemas lineares esparsos de grande porte, etc.

Duas aplicações são projetadas utilizando-se a base computacional desenvolvida, e implementadas em linguagem de programação C++ (STROUSTRUP, 1997). Estas aplicações, um fluxo de potência linearizado e uma metodologia de simulação dinâmica com modelagem detalhada, em conjunto com as facilidades computacionais, dão origem a um protótipo de ferramenta computacional (*OOTPS*), o qual exemplifica a construção e o funcionamento de aplicações para SEE, operando sob um ambiente integrado, e desenvolvidas sob a filosofia de projeto de software proposta neste trabalho.

1.5. Organização do Texto

No Capítulo 2 é descrito o paradigma da Modelagem Orientada a Objetos, com ênfase nos aspectos gerais da sua teoria e conceituação. São apresentadas e descritas algumas metodologias de projeto disponíveis na literatura, assim como o conceito de padrões de projeto, juntamente com alguns destes padrões, utilizados neste trabalho.

No Capítulo 3 são abordados aspectos gerais relativos à aplicação de técnicas de MOO na representação e modelagem de SEE, e no desenvolvimento de suas ferramentas computacionais. Apresentam-se aspectos como a adaptabilidade dos SEE à modelagem orientada a objetos e dificuldades na definição de uma estrutura eficiente para sua representação. Questões como desempenho de códigos orientados a objetos e flexibilidade das estruturas de classes frente a expansões são abordadas. Discute-se também alguns aspectos a respeito da reutilização de códigos existentes, de forma a não ferir os preceitos

da MOO. Ao final do capítulo é apresentada uma revisão bibliográfica de trabalhos publicados na área.

No Capítulo 4 são apresentadas as diversas abstrações do domínio da aplicação abordado no trabalho. As diferentes instâncias de um SEE são separadas em abstrações, representadas por pacotes, de acordo com a notação UML, facilitando com isso a modelagem das estruturas de classes que representam os SEE. São tratadas também as relações entre estas abstrações.

No Capítulo 5 são apresentadas as estruturas de classes representativas dos elementos físicos dos SEE, detalhando-se as classes (seus atributos e métodos). É abordada em detalhes a distinção entre os elementos físicos estruturais e de composição, juntamente com os relacionamentos entre estes dois grandes conjuntos.

No Capítulo 6 são apresentadas as estruturas de classes para representação das metodologias de análise e síntese (ou aplicações), detalhando-se algumas classes projetadas e implementadas neste trabalho. As interfaces funcionais, que representam o comportamento dos elementos físicos em cada aplicação, são também apresentadas.

No Capítulo 7 é tratada em específico a abstração das ferramentas computacionais. Um protótipo de ferramenta computacional, projetado e implementado utilizando-se a filosofia proposta no trabalho, é apresentado, juntamente com resultados de simulações. São apresentadas também algumas classes da abstração das facilidades computacionais, projetadas e implementadas ao longo do desenvolvimento desta tese, e utilizadas na implementação do protótipo.

Finalizando, o Capítulo 8 apresenta as conclusões do trabalho, juntamente com as suas principais contribuições. Sugestões de trabalhos futuros são também apresentadas.

CAPÍTULO 2

2. MODELAGEM ORIENTADA A OBJETOS

A engenharia de software está em constante desenvolvimento de suas técnicas de projeto, tendo em vista o também constante incremento de complexidade dos softwares a serem desenvolvidos e mantidos. Dentre os recentes paradigmas surgidos nessa área destaca-se a *Modelagem Orientada a Objetos* (MOO), base para diversas técnicas de projeto de software compatíveis com projetos complexos, mantendo a facilidade e agilidade de manutenção dos mesmos.

Neste capítulo apresentam-se alguns conceitos básicos da MOO, assim como métodos de projeto disponíveis na literatura, utilizados como base teórica no desenvolvimento do presente trabalho. São comentados métodos como OMT (RUMBAUGH et al., 1994), UML (BOOCH et al., 2000), Booch (BOOCH, 1994; BOOCH, 1998), Objectory, CRC, e Fusion (COLEMAN et al., 1996). É apresentado também o conceito de *padrões de projeto* (GAMMA et al., 2000), descrevendo-se alguns destes padrões, utilizados neste trabalho.

2.1. A Modelagem Orientada a Objetos

A MOO pode ser definida como uma maneira de se projetar sistemas, na qual se está interessado, em primeira instância, na clareza e organização do projeto. Ela tem por base uma representação clara e efetiva do mundo real que se pretende estudar com o sistema, de forma a facilitar ao máximo o desenvolvimento e a manutenção do projeto.

Em se tratando de projetos de sistemas de software utilizando-se técnicas baseadas na MOO, ocorre um deslocamento do esforço de desenvolvimento para as fases iniciais

de análise do problema. Algumas vezes pode parecer estranho despender a maior parte do tempo de trabalho nestas fases iniciais, mas esse esforço adicional é compensado pela implementação mais rápida e simples dos códigos. Como o projeto resultante é mais *limpo* e adaptável, futuras modificação são mais facilmente realizadas.

Ao contrário de metodologias formais de projeto de software, que dão maior ênfase à funcionalidade do projeto, metodologias baseadas na MOO priorizam a *estrutura de dados* do sistema, visando agrupar dados (atributos) e funcionalidades (métodos) nos objetos. Essa diferença de enfoque proporciona ao processo de desenvolvimento do projeto uma base mais estável e permite a utilização de um único conceito em todo o processo: o *objeto* (RUMBAUGH et al., 1994). O objeto é a entidade fundamental do paradigma, e possui atributos (características próprias) e métodos (formas de manipular seus atributos). Procura-se, em um primeiro instante, representar sob a forma de objetos computacionais os objetos do mundo real, que fazem parte do *domínio de aplicação* do projeto.

Entende-se por domínio de aplicação a delimitação conceitual do escopo onde se está interessado em aplicar a orientação a objetos (RUMBAUGH et al., 1994). Este é um conceito chave do paradigma, pois o início de um projeto orientado a objetos (OO) está baseado em uma delimitação precisa do domínio da aplicação, com a finalidade de se limitar a área a ser atacada no projeto.

As classes, definidas durante o projeto do sistema, dão forma aos objetos, ditos *instâncias* das classes. Muitas vezes, os conceitos de classe e objeto são utilizados sem distinção, inclusive neste texto. Porém, é importante observar que existe sim uma distinção. Uma classe pode ser considerada como o molde de um objeto, ou um conjunto deles, enquanto que cada objeto tem suas características definidas pela sua classe de origem.

Os conceitos de classe e objeto são bastante genéricos no paradigma da MOO, sendo praticamente independente de métodos de projeto. Já outros conceitos possuem uma maior dependência do método de projeto OO utilizado, sendo que pequenas variações podem ser encontradas nas definições de um método para outro. Na descrição dos conceitos apresentados a seguir, é dada uma maior ênfase ao método OMT (RUMBAUGH et al., 1994) por sua simplicidade e objetividade, e à UML (BOOCH et al., 2000), por seu potencial na representação visual de sistemas OO.

Os objetos e classes relacionam-se entre si através das *ligações* e *associações* (ou *relacionamentos*). Da mesma forma que as classes e os objetos, diz-se que uma ligação é uma instância de uma associação. As classes contêm associações entre si, quer dizer, determinados tipos de objetos estarão ligados a outros tipos de objetos. Existem alguns tipos especiais de associações, como *agregação, composição* e *herança.*

Quando um objeto é formado por um conjunto de objetos que podem existir sozinhos, existe aí um agregado. Já um composto é formado por um conjunto de objetos específicos, cuja existência não tem sentido sem a existência do composto. A composição é um tipo mais forte de agregação.

Já o mecanismo de herança permite que uma determinada classe, dita *derivada* (*descendente* ou *subclasse*) herde características de uma outra classe, a *base* (ou *superclasse*). Assim, é possível concentrar características comuns a um grupo de classes em uma classe base, fazendo com que todas as classes que contenham essas características sejam descendentes da base. A classe base pode, inclusive, não conter nenhuma instância, e somente servir de generalização para um conjunto de classes descendentes, essas sim originando objetos. Esse tipo de classe base é chamado *classe abstrata* (por não originar objetos diretamente). Mecanismos como a composição, agregação e a herança fornecem subsídios ao projeto, com o objetivo de hierarquizar as estruturas de classes. Permitem, assim, a modelagem dos dados segundo uma estruturação bem definida de objetos.

Classes base abstratas auxiliam na padronização de interfaces, fazendo-se uso do conceito de *polimorfismo*. O polimorfismo permite *sobrecarregar* determinadas operações e métodos, de forma que uma mesma operação (ou método) realize diferentes procedimentos, quando aplicada em objetos diferentes. O compilador encarrega-se de definir qual é o procedimento correto a aplicar quando determinada operação (ou método) é ativada em um objeto específico. Desta maneira, métodos de interface podem ser concentrados nas classes base, tendo cada classe derivada a sua implementação particular de cada método.

Outros dois importantes conceitos na MOO são a *abstração* e o *encapsulamento de dados*. No projeto de determinada classe deve-se sempre seguir a idéia de que os objetos desta classe devem ter seus atributos encapsulados no seu interior, sem que outros objetos tenham acesso direto a esses atributos. Dessa forma, o projeto de uma determinada parte do software simplesmente abstrai o restante, preocupando-se somente com as características intrínsecas dos objetos que estão sob modelagem. Isto facilita o desenvolvimento realizado por diferentes equipes, uma vez que a comunicação entre os objetos é definida pelas suas interfaces, sem interessar como o objeto manipula seus atributos em seu interior. Além disso, uma futura manutenção em determinada porção do software tende a não causar *efeitos colaterais* no restante do código, pois uma vez definidas as interfaces dos objetos, as suas características internas são isoladas do restante do software.

Note-se que a abstração é utilizada em vários momentos do projeto, e não somente na modelagem dos dados em si. O mecanismo da abstração permite que se concentre os esforços de projeto em características relevantes para cada momento do desenvolvimento, isolando aspectos menos importantes. Considerando-se a notação da UML (BOOCH et al., 2000), apresentada na seção 2.2.2, diferentes abstrações em um projeto podem ser representadas por *pacotes*.

É relevante notar que as técnicas de projeto de software baseadas no paradigma da MOO são essencialmente formas de se modelar o mundo real, traduzindo-o para o escopo de um sistema computacional. A implementação do código propriamente dito, como é visto adiante, na descrição dos métodos de projeto, é apenas uma das fases do projeto como um todo, sendo que a idéia central do paradigma está em se modelar sob a forma de objetos a região do mundo real a qual se tem interesse em estudar com o software.

Isto leva naturalmente à dissociação do projeto OO de qualquer linguagem de programação específica. Uma vez projetado o software, isto é, tendo-se as estruturas de classes prontas, juntamente com diagramas que definem os relacionamentos entre classes e objetos, e suas funcionalidades, parte-se para a implementação propriamente dita do sistema utilizando-se uma determinada linguagem de programação. Atualmente, uma das linguagens de programação mais utilizadas é a linguagem C++, devido ao seu excelente suporte a MOO e seu desempenho computacional comparável à linguagens tradicionais de programação científica (FORTRAN, por exemplo). Outras linguagens atuais, tal como Java (ARAÚJO et al., 2000), também apresentam suporte eficiente aos conceitos da MOO.

2.2. Métodos de Projeto Orientados a Objeto

Existem diversos métodos de projeto de software baseados no paradigma da MOO, cada um com suas particularidades, conceitos específicos, etapas, etc. Nesta seção são apresentados resumidamente alguns destes métodos.

Alguns autores defendem que seguir um método específico no projeto de um sistema orientado a objetos não é relevante, mas sim o é preocupar-se em criar um bom projeto, seguindo os preceitos gerais do paradigma da MOO em si (LARMAN, 2000). Ainda assim, no presente trabalho, apesar de se aproveitar aspectos de praticamente todos os métodos aqui apresentados, utiliza-se de forma mais específica os preceitos do método OMT - *Object Modeling Technique* (RUMBAUGH et al., 1994), e as notações gráficas da UML - *Unified Modeling Language* (BOOCH et al., 2000). O OMT, conforme abordado a

seguir, é um método simples e eficaz, que torna o processo de desenvolvimento de um projeto de software OO claro e específico. É importante destacar que métodos mais recentes, como Fusion, ou a própria UML, tem por base o OMT. Para a notação gráfica das estruturas desenvolvidas neste trabalho utiliza-se a UML, pela sua posição consolidada na área de modelagem visual de projetos orientados a objetos.

2.2.1. Object Modeling Technique

O OMT, ou Técnica de Modelagem de Objetos, é um método de projeto de sistemas orientados a objetos, que foi desenvolvido na General Electric, onde James Rumbaugh, seu principal colaborador, trabalhou. O método é descrito detalhadamente em RUMBAUGH et al. (1994), e é aqui apresentado de forma resumida.

2.2.1.1. Os Modelos do OMT

O OMT representa o sistema em projeto através de três visões ortogonais, chamadas modelos: o *Modelo de Objetos*, o *Modelo Funcional* e o *Modelo Dinâmico*.

O Modelo de Objetos representa os aspectos estáticos e estruturais do sistema; representa seus dados, organizados sob a forma de classes e objetos. O modelo descreve as classes com suas identidades, seus relacionamentos, seus atributos e suas operações. Dá a base para os outros modelos. É representado graficamente por *Diagramas de Classes*, onde as classes são organizadas em níveis hierárquicos, mostrando a forma dos objetos do sistema.

O Modelo Funcional descreve *o que* o sistema faz e *com quem*, independente de quando o faz. Ele é representado por *Diagramas de Fluxo de Dados*, os quais mostram as dependências entre os valores de saída e de entrada em um determinado procedimento.

O Modelo Dinâmico descreve os aspectos de um sistema relacionados ao tempo e à seqüência de operações que assinalam modificações nos estados do sistema. Ele incorpora o controle de fluxo do sistema. Representado por *Diagramas de Estados*, este modelo mostra *quando* ocorre cada evento, e a seqüência das atividades executadas pelos objetos.

- **Relações Entre os Modelos**

Cada um dos três modelos descreve um aspecto isolado do sistema, mas contém referências aos outros modelos. O modelo de objetos, por exemplo, descreve a estrutura

de dados sobre a qual atuam os modelos dinâmico e funcional. As operações do modelo de objetos (métodos das classes) são ativadas pelos eventos do modelo dinâmico, e correspondem às funções do modelo funcional. Já o modelo dinâmico descreve a estrutura de controle dos objetos, mostrando o momento em que as funções do modelo funcional devem acontecer.

Podem existir alguns conflitos ao se definir em qual modelo deve aparecer certa informação. Isto é natural, uma vez que os modelos são criados com base em abstrações do sistema real, e sabe-se que podem existir várias abstrações para um mesmo problema, e nenhuma delas será completa e absoluta. Assim, o melhor enfoque é simplificar a descrição sem sobrecarregar o modelo com tantas construções. O método permite o uso de notações específicas e linguagem natural para representar detalhes por ora não colocados nos modelos.

2.2.1.2. As Fases do OMT

Considerando-se que o OMT é uma metodologia de projeto de software, tem como objetivo básico a produção organizada de softwares, utilizando para isso um conjunto de técnicas predefinidas e convenções de notação. Uma metodologia costuma ser apresentada como uma série de etapas. O OMT define três etapas distintas: *Análise, Projeto e Implementação*. A Figura 2-1 mostra estas etapas.

Na Análise, está-se preocupado com a compreensão e a modelagem inicial da aplicação e do domínio em que ela atua. A entrada principal desta fase normalmente é um enunciado que descreve o problema a ser solucionado e oferece uma visão geral do sistema. Outras entradas para a análise são entrevistas com usuários e especialistas na área de conhecimento do domínio da aplicação. A saída da etapa de análise é um conjunto com um primeiro esboço dos três modelos do método, com especial atenção ao Modelo de Objetos.

A fase de Projeto subdivide-se em Projeto dos Objetos e Projeto do Sistema. O Projeto dos Objetos refina os modelos provenientes da análise. Durante esta fase ocorre um deslocamento na ênfase dos conceitos da aplicação propriamente dita, em direção aos conceitos computacionais. São escolhidos os algoritmos básicos para a implementação das principais funções do sistema. Com base nesses algoritmos, a estrutura do Modelo de Objetos é otimizada com a finalidade de se obter uma implementação eficiente, sem se desviar a atenção dos preceitos do paradigma. Começam a aparecer aí algumas dependências do projeto, ainda que sutis, com as possíveis linguagens de programação que

poderão vir a ser utilizadas na implementação. Determina-se nesta fase como será a implementação de cada associação e de cada atributo. Por fim, os subsistemas são empacotados em blocos de código.

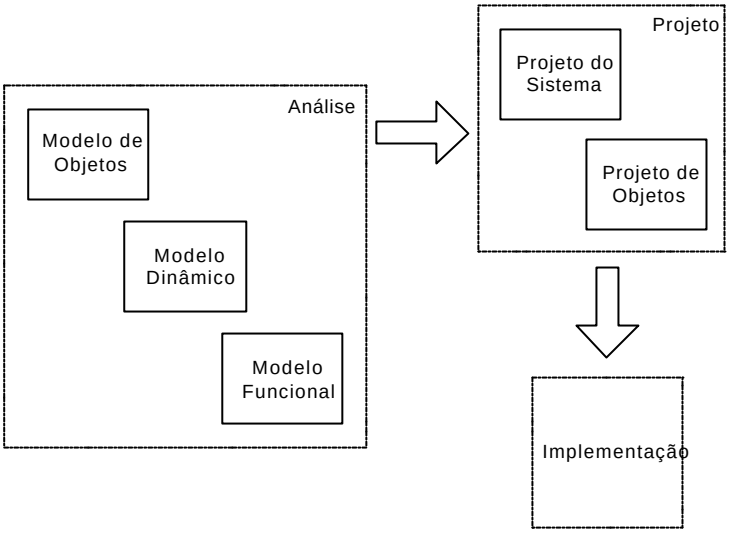


Figura 2-1 – Etapas do OMT

O Projeto do Sistema estabelece a definição da arquitetura geral do sistema computacional sob projeto. Decisões gerais sobre comunicação entre processos, armazenamento de dados e implementação do Modelo Dinâmico são tomadas. Nesta fase são definidas as interfaces com os usuários, seqüência de telas do programa, forma de entrada e saída de dados, etc.

Na fase de Implementação, é escolhida uma linguagem de programação, preferencialmente uma que suporte de forma eficiente o paradigma da MOO, e o projeto é traduzido em termos de códigos-fonte. O processo torna-se mecânico, uma vez que a grande maioria das decisões importantes do projeto já foram tomadas nas etapas anteriores.

Apesar de dividido em fases, o processo de desenvolvimento é contínuo. Uma vez que a abordagem baseada em objetos define um conjunto de objetos logo no início do projeto, e continua a usar e estender esses objetos através do ciclo de desenvolvimento, a separação entre as fases é um tanto tímida. No OMT, o modelo de objetos desenvolvido durante a análise é utilizado no projeto e na implementação, e o trabalho é conduzido rumo ao refinamento do modelo em níveis progressivamente mais detalhados. Assim, o processo não possui descontinuidades, não havendo também troca de notação de uma fase para outra.

Neste trabalho de tese, o método OMT é tomado como base teórica para o desenvolvimento das estruturas de classes, seus relacionamentos e funcionalidades.

2.2.2. Unified Modeling Language

A UML - Unified Modeling Language (BOOCH et al., 2000) extrapola o conceito de técnica de projeto de software OO, definindo um padrão de notação para a modelagem visual de problemas reais. Ela tenta capturar as partes essenciais de um sistema real, modelando-as visualmente de forma eficiente, com o objetivo final de traduzir o problema do domínio do mundo real para o domínio de um sistema computacional, onde o problema será estudado.

A UML combina, além da *modelagem de objetos*, onde entram especificamente os conceitos da MOO, conceitos de *modelagem de dados*, *modelagem de componentes de sistemas* e *modelagem de negócios*. Assim, além dos sistemas de software, cobre também uma ampla faixa de outros problemas reais passíveis de modelagem, tais como sistemas de informação, sistemas técnicos, sistemas de tempo real, sistemas distribuídos e sistemas gerenciais de negócios.

A seguir são apresentados brevemente alguns aspectos que compõe a UML.

· **Perspectivas ou Visões:** mostram diferentes aspectos do sistema sob modelagem. Perspectivas não são gráficos, mas sim abstrações formadas por diversos diagramas. Elas conectam a linguagem de modelagem ao método de projeto utilizado. As perspectivas da UML são mostradas na Figura 2-2.

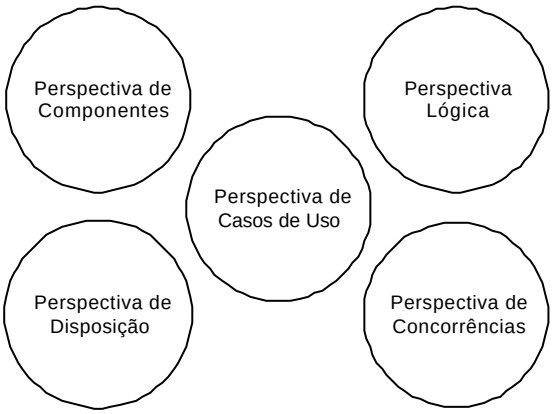


Figura 2-2 – Perspectivas da UML

A Perspectiva de Casos de Uso descreve a funcionalidade do sistema vista externamente a ele. Ela especifica o que o sistema deverá fazer, de uma forma geral. A Perspectiva Lógica, ou Visão de Projeto, descreve a funcionalidade do sistema, olhando para dentro dele. Ela descreve tanto a sua estrutura estática (classes e suas relações) quanto a dinâmica (mensagens entre objetos). A Perspectiva de Componentes, ou Visão de Imple-

mentação, mostra a organização dos componentes do código. Esta perspectiva dá uma descrição dos módulos (blocos de código) e suas dependências. A Perspectiva de Concorrências, ou Visão do Processo, trata da divisão do sistema em processos e processadores; permite assim um eficiente uso dos recursos computacionais disponíveis. Finalmente, a Perspectiva de Disposição, ou Visão de Implantação, mostra a disposição física do sistema, os processadores, os dispositivos, e as conexões entre eles, e é representada por Diagramas de Disposição.

· **Diagramas:** são os gráficos que mostram os elementos do sistema, arranjados de forma a representar aspectos particulares do projeto. Quando agrupados, os diagramas dão origem às diversas perspectivas comentadas anteriormente. Alguns diagramas podem fazer parte de várias perspectivas, dependendo do conteúdo do diagrama. Os tipos de diagramas são divididos em dois grupos: estáticos e dinâmicos. Os estáticos são os *Diagramas de Classes*, *Diagramas de Objetos*, *Diagramas de Componentes* e *Diagramas de Implantação*. Os dinâmicos são os *Diagramas de Casos de Uso*, *Diagramas de Estado*, *Diagramas de Seqüência*, *Diagramas de Colaboração* e *Diagramas de Atividades*. Um determinado projeto não necessita ter todos estes diagramas explicitados, pois alguns deles trazem visões redundantes do sistema. O uso de um diagrama ou outro depende da natureza do sistema sob projeto, do tipo de visão que se quer apresentar do projeto, e da experiência do(s) projetista(s).

· **Elementos do Modelo:** os elementos do modelo são as unidades fundamentais formadoras dos diagramas, e representam desde as classes e objetos, passando por possíveis estados do sistema, até relações (associações) existentes entre classes.

· **Mecanismos:** A UML utiliza alguns mecanismos nos seus diagramas, para representar informações adicionais que tipicamente não poderiam ser representadas com os elementos do modelo. Alguns tipos são: *adornos*, como diferenciação no tipo de letra, sublinhados, etc., os quais procuram adicionar semântica aos elementos; *notas*, adicionando quaisquer informações adicionais aos diagramas, sob forma de comentários; *especificações*, mostrando propriedades específicas de determinados elementos do modelo; etc.

A UML, sendo uma linguagem de modelagem visual, não tem o intuito de estipular etapas específicas para o processo de desenvolvimento do projeto, característica específica dos métodos de projeto. Porém, ainda assim é possível identificar cinco fases (ou etapas) principais no desenvolvimento de um sistema, baseadas nas etapas do OMT:

· **Análise dos Requisitos:** primeira fase do projeto; procura identificar *o que* o sistema deverá realizar. Já são montados nesta etapa alguns diagramas de casos de uso, dando forma à Perspectiva de Casos de Uso.

· **Análise:** nesta fase identificam-se as classes (e objetos) do projeto, suas relações, e os mecanismos presentes no domínio do problema. Diagramas de classes são montados, e colaborações entre as classes são descritas através de diagramas dinâmicos.

· **Projeto:** nesta fase os produtos da análise são acrescidos das soluções técnicas. Novas classes são adicionadas, provendo suporte técnico: interfaces com o usuário, bancos de dados, etc. Esta é a fase do desenvolvimento propriamente dito do projeto.

· **Implementação:** aqui as classes são programadas em termos de códigos-fonte, em uma linguagem de programação específica; preferencialmente escolhe-se uma linguagem com suporte eficiente à MOO, tais como C++ ou Java.

· **Testes:** na fase de testes são provadas as várias partes do projeto, tanto individualmente como em conjunto. Testes em unidades usam os diagramas de classes e especificações das classes. Testes de integração entre módulos usam tipicamente diagramas de colaboração, e testes no sistema completo utilizam os diagramas de casos de uso para validar o sistema, de acordo com os requerimentos iniciais do projeto.

Um projeto é mais bem desenvolvido sob a forma de um conjunto de iterações, evoluindo-se os modelos a cada iteração. Assim, cada iteração adiciona características novas, mais detalhes aos diagramas. Uma iteração anterior fornece a entrada para a posterior, tanto em termos de diagramas, quanto em termos de realimentação no próprio processo de desenvolvimento, sendo que o entrosamento da equipe de desenvolvimento com o projeto tende a aumentar a cada iteração. Assim, o desenvolvimento do projeto passa a ser iterativo e incremental, como mostra a Figura 2-3.

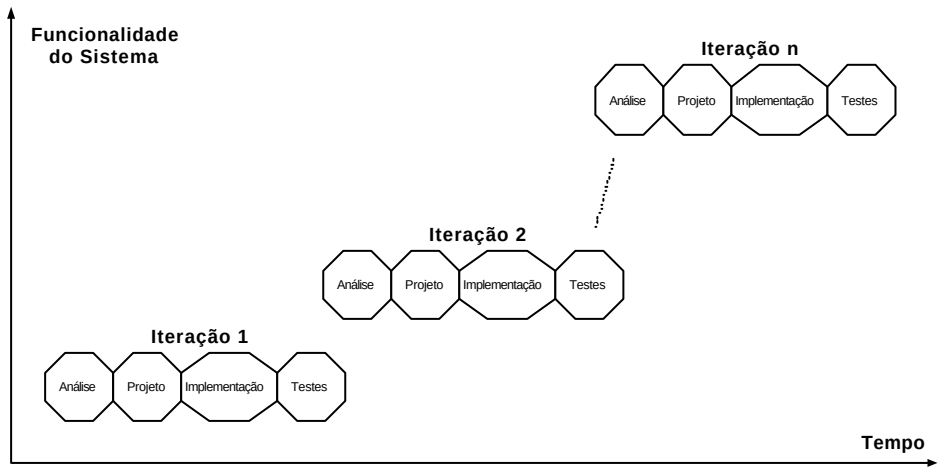


Figura 2-3 – Desenvolvimento Iterativo e Incremental de um Projeto

Existem pacotes computacionais para suporte ao desenvolvimento de projetos utilizando a UML. Pode-se citar o Rational Rose, da Rational (http://www.rational.com), e também o Together, da TogetherSoft (http://www.togethersoft.com). Esses pacotes facilitam todo o processo de desenvolvimento do projeto, desde sua concepção até a implementação dos códigos-fonte.

Neste trabalho de tese, a UML é utilizada para a notação gráfica das estruturas concebidas.

2.3. Padrões de Projeto

Recentemente, um novo conceito vem ganhando destaque em projetos orientados a objetos: os *Padrões de Projeto* (ou *Design Patterns*) (GAMMA et al., 2000). Conceito herdado da construção civil, os padrões de projeto, como o próprio nome sugere, são estruturas padrões para a solução de problemas genéricos, com os quais normalmente os projetistas de sistemas orientados a objetos deparam-se.

Em qualquer projeto de software, a maioria dos problemas a serem resolvidos (isto é, modelados) são comuns a outros projetos, e muitas vezes já foram solucionados de maneira eficiente em outra oportunidade. Quando um grupo de problemas semelhantes pode ser solucionado por uma determinada estrutura padrão, tem-se aí um padrão de projeto. Um padrão não necessariamente é imutável; pelo contrário, pode sofrer pequenos ajustes, para adaptar-se às especificidades de cada problema.

Um bom projeto orientado a objetos possui certamente uma grande quantidade de padrões em suas estruturas de classes. Muitas vezes, estes padrões podem não estar explícitos, ou mesmo podem ter sido usados de forma não intencional; mas se o projeto segue de maneira correta os princípios da orientação a objetos, padrões podem ser identificados em suas estruturas, resolvendo problemas genéricos.

Em GAMMA et al. (2000), os autores identificam 23 padrões de projeto, os quais vêm sendo testados e utilizados pelos autores e outros desenvolvedores ao longo do tempo. Estes padrões são classificados quanto a sua finalidade, e quanto ao seu escopo. Os padrões podem ter basicamente três tipos de finalidade: criação, estrutural ou comportamental. Os padrões de criação identificam e resolvem problemas relacionados à criação de objetos. Os estruturais tratam do agrupamento entre classes ou objetos, formando novas classes ou objetos. Os padrões comportamentais lidam com as maneiras como classes e objetos interagem entre si. Quanto ao escopo, os padrões podem ter aplicação em classes ou objetos. Os padrões de classes têm como mecanismo base a herança; são

construções estáticas no tempo, ou seja, são definidas em tempo de compilação, sem a possibilidade de mudança na estrutura em tempo de execução do software. Os padrões de objetos (mais numerosos) são definidos por arranjos de ligações entre objetos (ligações simples, agregações, composições, etc.). São dinâmicos, sendo possível alterações em suas estruturas em tempo de execução.

O uso de padrões em projetos orientados a objetos vem ao encontro de um preceito chave da MOO: a reutilização de estruturas (ARAÚJO et al., 2000). Sua aplicação traz modularidade e extensibilidade aos projetos, uma vez que enfatizam a definição adequada das interfaces entre diferentes abstrações do sistema sob modelagem. Quando se utiliza um padrão na solução de um determinado problema, sabe-se que esta solução já foi devidamente testada e aprovada por outros desenvolvedores, nas mais diversas situações de projeto.

Os principais padrões utilizados neste trabalho de tese são apresentados a seguir, juntamente com suas estruturas de classe.

2.3.1. Padrão Adapter

Adapter é um padrão estrutural que pode ser aplicado tanto em classes como em objetos. Seu objetivo básico é converter a interface de uma classe já existente, através de uma nova classe; permite, assim, que classes, em princípio incompatíveis, trabalhem em conjunto.

Uma determinada classe, projetada hoje para ser uma classe reutilizável no futuro, terá uma determinada interface de comunicação. Porém, é possível que uma aplicação futura que necessite reutilizar a classe não seja compatível com sua interface. O padrão Adapter pode ser utilizado com a finalidade de “adaptar” a classe ao projeto que a está reutilizando, através de uma nova interface. Esta nova interface será encapsulada em uma nova classe, que estará ligada à classe sob reutilização. A Figura 2-4 mostra a estrutura do padrão. A classe *Adapted*, projetada anteriormente para ser reutilizada por futuras aplicações (ou clientes), está tendo sua interface adaptada pela nova classe *Adapter*, de forma que ela possa ser reutilizada por *Client*. Neste exemplo utilizou-se o padrão em objetos, pois a interface de *Adapted* está sendo convertida por um objeto do tipo *Adapter*.

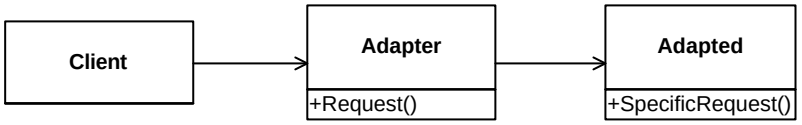


Figura 2-4 – Padrão Adapter

Aplicando-se este padrão, é possível projetar-se hoje classes que serão reutilizadas no futuro por outras aplicações, sem maiores preocupações sobre como serão estas futuras aplicações, ou sobre como as aplicações irão utilizar a classe. Projeta-se neste momento uma classe devidamente encapsulada, com uma interface básica de comunicação, abstraindo-se questões externas a ela. Aplicações que por ventura venham a reutilizar a classe, definirão classes "adaptadoras", compatibilizando suas interfaces.

O padrão Adapter é utilizado neste trabalho no projeto das interfaces funcionais dos elementos físicos do SEE, conforme será visto no Capítulo 6.

2.3.2. Padrão Strategy

O padrão Strategy é um padrão comportamental de objetos, cujo objetivo é encapsular em classes distintas, diferentes formas de realizar uma mesma tarefa, tornando-as intercambiáveis. A Figura 2-5 mostra o diagrama de classes do padrão.

Em uma estrutura de classes, pode-se ter uma classe realizando uma determinada tarefa, a qual pode ser realizada de diferentes formas (solução de sistemas lineares, por exemplo). O padrão Strategy encapsula em classes distintas (chamadas *strategy*), as diferentes formas de se realizar as tarefas, classes estas que são parte da classe responsável pela tarefa (chamada *context*). Normalmente utilizam-se associações do tipo composição para conectar as *strategies* ao seu *context.* Um cliente que venha a trabalhar em conjunto com o *context,* solicitando a este a execução da tarefa, em princípio cria e informa uma determinada estratégia para a realização da tarefa ao contexto, e a partir daí interage exclusivamente com o contexto.

Esta construção apresenta algumas características importantes, dentre as quais ressaltam-se duas:

- A estrutura apresenta facilidade de expansão; novas maneiras de se realizar determinada tarefa podem ser implementadas futuramente, sem que alterações sejam necessárias nas classes já existentes;

- A troca entre diferentes estratégias de execução de uma mesma tarefa pode ser realizada dinamicamente, em tempo de execução do software.

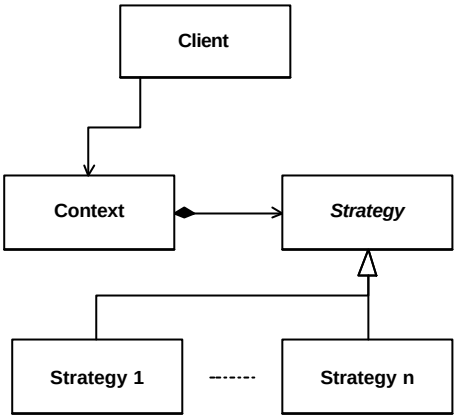


Figura 2-5 – Padrão Strategy

O padrão Strategy é utilizado em vários momentos neste trabalho, como por exemplo no encapsulamento das diversas estratégias de solução de sistemas lineares da estrutura para tratamentos de matrizes esparsas, ou no encapsulamento das diferentes estratégias de leitura de dados de SEE, de acordo com o tipo de banco de dados utilizado (ver Capítulo 7).

2.3.3. Padrão Singleton

Sendo um padrão de criação de objetos, o padrão Singleton garante que uma determinada classe tenha uma única instância durante o tempo de execução do software. Além disso, o uso deste padrão permite que essa instância única seja acessível de qualquer parte do software (tornando-a um *objeto global*). Ainda, os objetos do tipo *singleton* são criados *sob demanda*, isto é, somente são criados quando realmente são necessários.

A utilização do padrão é realizada através da implementação de um atributo e um método na classe, ambos estáticos (tipo *static*). Este método deverá ser utilizado para a criação do objeto único da classe. Quando o método estático é acionado, retorna o atributo estático, o qual contém um apontador para o objeto único. Caso o objeto ainda não tenha sido criado, o método o faz, armazena um apontador para o endereço do objeto no atributo estático da classe, e retorna o endereço. Garante-se que apenas o método estático será usado na tarefa de criação do objeto fazendo-se com que o construtor da classe seja *privado*. A estrutura do padrão pode ser observada na Figura 2-6.

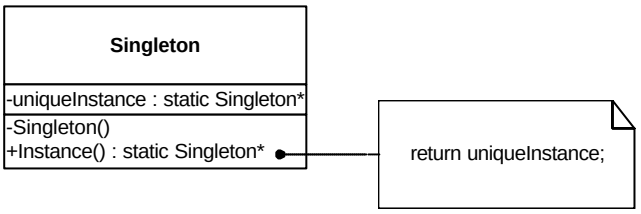


Figura 2-6 – Padrão Singleton

O padrão Singleton é utilizado na criação do objeto *Power System* (ver Capítulo 5), e na criação do objeto principal do software (objeto ferramenta – ver Capítulo 7).

2.3.4. Padrão Composite

O padrão Composite permite que se componham objetos em estruturas de árvore, para a representação de hierarquias do tipo "partes-todo". Neste padrão, objetos podem ser compostos por outros objetos de mesmo nível hierárquico do composto, permitindo que os clientes da estrutura tratem de maneira uniforme tanto objetos individuais (*leaf*), quanto uma composição de objetos (*composite*). Um diagrama de classes do padrão pode ser observado na Figura 2-7.

Nesta figura, a classe *Composite*, que está em um mesmo nível hierárquico que *Leaf*, pode ser formada por *n* instâncias de objetos tipo *Component*. As classes *Leaf* e *Composite* são equivalentes, quer dizer, são tratadas da mesma forma pelos clientes, através da interface declarada em *Component*. Toda mensagem tipo *Operation()* dada a um objeto tipo *Composite* é repassada por este aos seus componentes.

O padrão permite ainda algumas variações. Pode ser utilizada a composição em vez de agregação, fazendo com que os componentes de um *composite* pertençam exclusivamente a este. Variações podem ocorrer na forma de gerenciamento dos componentes de um *composite*, delegando a este tarefas de adicionar, remover, etc. componentes (dessa forma não é necessário declarar métodos do tipo *Add()* e *Remove()* na classe base *Component*).

Este padrão pode ser utilizado no projeto de metodologias de aplicação para SEE mais complexas, as quais podem ser compostas de outras metodologias mais simples, já implementadas na estrutura (ver Capítulo 6).

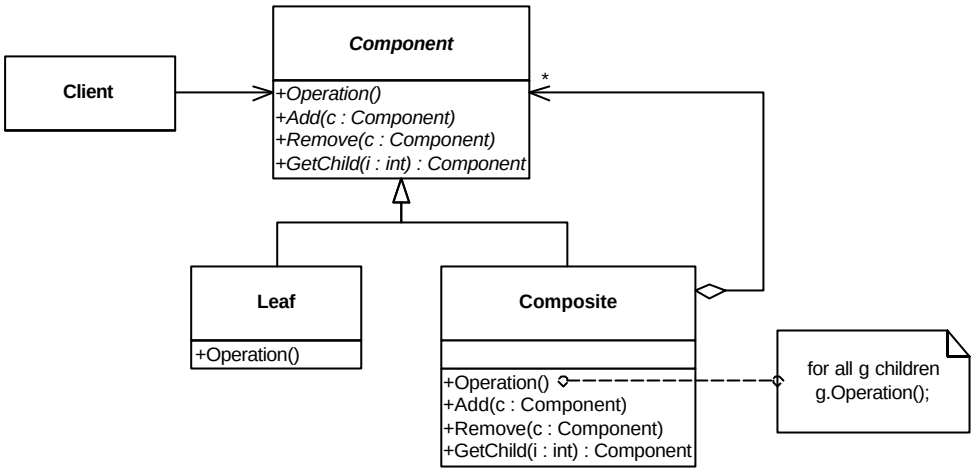


Figura 2-7 – Padrão Composite

2.3.5. Padrão Template Method

Template Method é um padrão comportamental de classes, o qual define o esqueleto de um algoritmo em uma operação na classe base (método *template*), postergando a implementação de alguns passos do algoritmo (operações primitivas) para as classes derivadas. Cada classe derivada redefine certas etapas do algoritmo, sem alterar seu conceito original. O padrão concretiza uma técnica fundamental para a fatoração de comportamentos semelhantes em um grupo de classes, permitindo concentrar etapas comuns desses comportamentos em uma classe base abstrata. A Figura 2-8 apresenta sua estrutura básica.

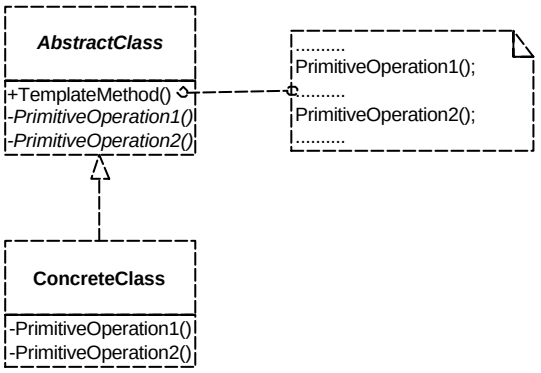


Figura 2-8 – Padrão Template Method

Observa-se na figura que o método *TemplateMethod()* de *AbstractClass* define a estrutura principal do método, realizando certas etapas que são comuns a todas as classes derivadas (representadas na figura por "....."), e fazendo chamadas para métodos primitivos (*PrimitiveOperation1()* e *PrimitiveOperation2()*). Estes métodos primitivos são decla-

rados virtuais na classe base *AbstractClass*, e redefinidos por cada classe derivada *ConcreteClass*. Dessa forma, um objeto criado através de uma classe qualquer *ConcreteClass* possui o método *TemplateMethod()* completo, sendo que sua declaração e esqueleto principal são herdados de *AbstractClass*, e etapas específicas são redefinidas na classe *ConcreteClass* em questão.

No presente trabalho de tese, utiliza-se o padrão Template Method na implementação do sistema genérico para leitura de dados de SEE (em conjunto com o padrão Strategy – ver Capítulo 7).

CAPÍTULO 3

3. APLICAÇÃO DE MOO NO DESENVOLVIMENTO DE SOFTWARE PARA SEE

Neste capítulo são abordados aspectos gerais relativos à aplicação de técnicas de Modelagem Orientada a Objetos (MOO) tanto para a representação e modelagem de Sistemas de Energia Elétrica (SEE), quanto para o desenvolvimento de softwares para o setor elétrico. Apresentam-se alguns aspectos gerais a respeito do tema, tais como a adaptabilidade dos SEE à modelagem orientada a objetos e dificuldades na definição de uma estrutura eficiente para sua representação. Questões como desempenho de códigos orientados a objetos e flexibilidade das estruturas de classes frente a expansões são abordadas. Comenta-se também o tema "reutilização de códigos", preceito fundamental do paradigma da MOO. Ao final do capítulo é apresentada uma revisão bibliográfica de trabalhos publicados na área.

3.1. Adaptabilidade dos SEE à Modelagem Orientada a Objetos

As técnicas de projeto de software baseadas no paradigma da MOO procuram modelar um sistema físico real no âmbito computacional onde se vai estudar o sistema, representando-se internamente no software as características estáticas e dinâmicas dos elementos do mundo real. Esta representação deve ser o mais fiel possível, para facilitar o entendimento do projeto.

Em um primeiro momento no projeto de um sistema orientado a objetos (até mesmo antes de se começar o projeto propriamente dito), convém se fazer uma análise geral do

que se pretende modelar, com o objetivo de se identificar a viabilidade deste projeto. Alguns sistemas mais simples, ou extremamente funcionais, podem não se adaptar eficientemente a MOO; ou, em alguns casos, pode-se constatar que a aplicação da MOO não trará benefícios ao projeto.

Fazendo-se esta análise com relação à área de SEE, observa-se que a aplicação da MOO dá-se de forma natural, com benefícios em muitos aspectos. Os SEE têm uma estrutura física bastante adaptável a uma estrutura de classes. A forma com que os elementos se conectam formando a rede elétrica sugere o formato geral das estruturas (ZHU et al., 1997; FUERTE-ESQUIVEL et al., 1998; MANZONI et al., 1999).

Toma-se como exemplo o esquema de uma unidade de geração, mostrado na Figura 3-1, ilustrando sua modularidade. Distingue-se o objeto unidade de geração, o qual está conectado a um objeto barra. A unidade de geração é formada por um conjunto (composição) de outros elementos do sistema. Sua comunicação com a barra dá-se através das variáveis *tensão terminal* e *corrente de armadura.* Distinguem-se também os elementos internos à unidade de geração, os quais apresentam também uma comunicação (ligações) clara entre si. Vê-se que todos os elementos conectam-se à máquina síncrona. O regulador de tensão (RAT) e o estabilizador de sistema de potência (ESP) estão conectados entre si, assim como o regulador de velocidade e a turbina.

Assim, a unidade de geração poderia ser modelada segundo o diagrama de classes[1] da Figura 3-2. O objeto unidade de geração está representado por um composto, formado por uma máquina síncrona equivalente, um regulador de tensão, um estabilizador (ESP), um regulador de velocidade, e uma turbina. Dependendo de como se quer modelar a unidade, pode-se ainda expandir a estrutura, tendo-se outros elementos no composto, tais como caldeira, sensores, etc. A unidade está conectada a uma única barra, que pode servir de conexão para diversas unidades de geração. As conexões entre os elementos formadores da unidade estão intrínsecas ao composto, ou seja, a classe *Unidade de Geração* é responsável por gerenciar as trocas de informações entre seus elementos componentes.

[1] Os diagramas de classes desenvolvidos neste trabalho são apresentados em suas formas completas nos capítulos posteriores.

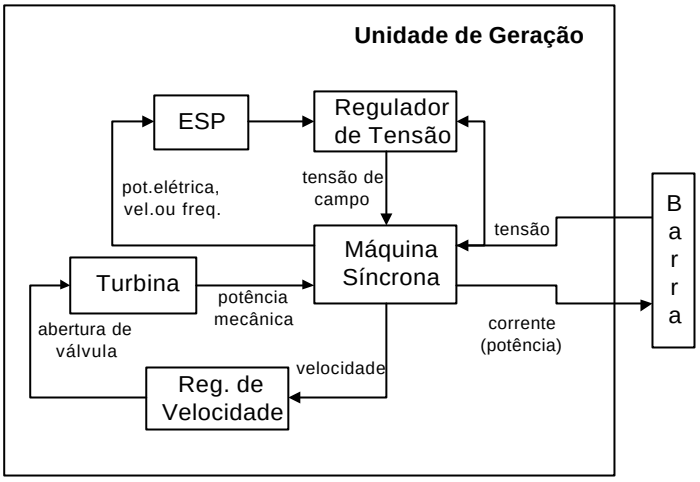


Figura 3-1 – Esquema Estrutural de uma Unidade de Geração

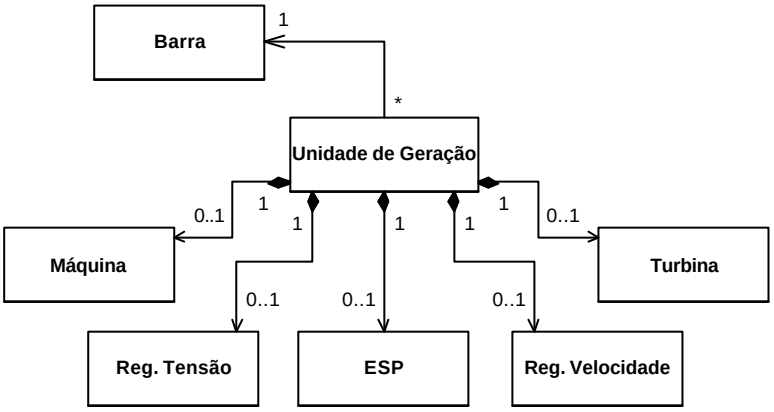


Figura 3-2 – Diagrama de Classes para uma Unidade de Geração

3.2. Modelagem das Metodologias como Classes

Não somente os elementos do SEE sinalizam a construção de uma boa estrutura de classes, como também as metodologias de análise e síntese relativas ao sistema sugerem estruturas bem definidas. Seria natural pensar que os métodos de análise e síntese relativos aos SEE deveriam ser implementados como métodos na estrutura representativa do sistema. Porém, essa visão limita demasiadamente a flexibilidade da estrutura, uma vez que se estaria misturando em uma mesma estrutura dois grandes conceitos: a modelagem dos elementos físicos do sistema elétrico em si, e as metodologias de análise e síntese aplicadas sobre esses elementos.

RUMBAUGH et al. (1994) afirmam que "*...uma operação que tem características próprias deve ser modelada como classe...*". Assim, quando uma determinada operação, primeiramente imaginada como um método pertencente a alguma classe da estrutura, as-

sume importância relevante no projeto, adquirindo uma identidade, esta deve ser modelada como uma classe.

Seguindo-se essa idéia, alguns trabalhos procuram representar tanto os elementos do SEE quanto as suas metodologias de análise através de estruturas de classes (GAING et al., 1996; HANDSCHIN et al., 1998; FUERTE-ESQUIVEL et al., 1998). Neste trabalho esboça-se a construção de duas grandes estruturas: uma representativa dos elementos dos SEE, e outra representando as metodologias de análise e síntese nestes sistemas. A primeira trata exclusivamente de representar os elementos que compõe a estrutura física dos SEE. A segunda estrutura representa uma hierarquia de metodologias de análise e síntese, modeladas independentemente da estrutura representativa dos SEE. Essa segunda estrutura atuará sobre a estrutura representativa dos SEE, utilizando-a como entrada e saída de dados para os seus métodos.

A representação das metodologias de análise e síntese por meio de uma estrutura de classes independente da estrutura do sistema traz flexibilidade à modelagem. Na construção da estrutura dos elementos físicos abstraem-se características específicas das metodologias a serem implementadas sobre o sistema, e vice-versa. O projeto passa a ter uma característica mais modular, pois se separa o que é elemento físico real, do que é metodologia computacional de análise e síntese.

3.3. MOO e Desempenho de Códigos OO

A definição das estruturas de classes é o ponto de partida em um projeto orientado a objetos. O modelo de objetos que representa estaticamente os elementos do mundo real e suas relações é a base para as próximas etapas do projeto (RUMBAUGH et al., 1994). Por isso, especial atenção deve ser despendida para essa fase do projeto.

Essa definição não é uma tarefa fácil (ZHU, 1999). Depende fundamentalmente de um perfeito entendimento por parte da equipe, do domínio da aplicação que se está modelando. Além disso, em problemas complexos, como é o caso de SEE, existe a possibilidade de se ter não somente uma, mas várias formas de se representar a hierarquia dos elementos. No caso de SEE, onde os métodos de análise e síntese tendem a ser computacionalmente dispendiosos, a escolha de uma forma ou de outra para a representação da estrutura, mesmo estando todas de acordo com a filosofia da MOO, pode implicar em diferenças de desempenho para determinadas aplicações específicas.

De maneira geral, o projeto de um bom software orientado a objetos não é uma tarefa fácil (ZHU, 1999; GAMMA et al., 2000). Entende-se por um *bom software orientado a*

objetos um programa com todas as características já mencionadas anteriormente, quais sejam flexibilidade para manutenções e atualizações, que possua uma estrutura de dados robusta, facilidades de projeto em equipes, etc. Além disso, um bom software deve ser elegante em seu projeto, ou seja, deve utilizar-se de técnicas modernas e claras de programação, inclusive em nível de códigos-fonte.

Porém, um bom software deve também ter um bom desempenho computacional. De nada adianta um programa ser orientado a objetos, com projeto e códigos extremamente limpos e claros, se seu desempenho em termos de tempo de execução for pouco atrativo.

Discussões vêm sendo travadas a respeito do desempenho computacional de programas orientados a objetos. NEYER et al. (1990), em um trabalho pioneiro na área, relatam problemas de desempenho de programas orientados a objetos. Os autores relatam a necessidade de *pagar* as vantagens da MOO com *overheads* adicionais devido às trocas de mensagens entre os objetos, e também com maiores requerimentos de memória. Toma-se como comparativo para essa análise linguagens baseadas em sub-rotinas, já consolidadas há algum tempo, as quais contam com compiladores com capacidade de gerarem códigos extremamente eficientes (Fortran 77, por exemplo).

Trabalhos mais recentes (ZHOU, 1996; FUERTE-ESQUIVEL et al., 1998; MANZONI et al., 1999; AGOSTINI et al., 2000) têm apresentado melhores resultados para implementações orientadas a objetos, utilizando-se normalmente a linguagem de programação C++. Atribuem-se essas melhoras de desempenho principalmente aos avanços no desenvolvimento dos compiladores disponíveis para linguagens com suporte a orientação a objetos, os quais evoluíram bastante nos últimos tempos, incorporando técnicas para a geração de códigos bastante eficientes (FOLEY et al., 1995). Características como *substituições inline*, *polimorfismo* e *templates* são suportadas em tempo de compilação, tornando a execução dos códigos mais eficiente.

Outra questão importante a ser observada nesse aspecto é o constante avanço no hardware disponível para a utilização dos novos programas. O poder de processamento das máquinas disponíveis no mercado aumenta constantemente. Por exemplo, adventos como o sistema operacional LINUX, sistema com código aberto, permite a implementação de *clusters de PCs* com grande capacidade de processamento, a um baixo custo. E nota-se que isto não é apenas uma situação, mas uma tendência, sinalizando muitos avanços que ainda estão por vir.

Apesar de ser um item relevante, o desempenho computacional não deve mais ser a prioridade principal no desenvolvimento de software para o setor elétrico. Este item cede

a prioridade para as questões já descritas anteriormente, tais como clareza de projeto, flexibilidade para manutenções, robustez de estruturas de dados, etc.

3.4. Multiplicidade e Expansibilidade das Estruturas de Classes

Conforme abordado na seção anterior, a definição das estruturas de classes representativas do problema é uma tarefa difícil, uma vez que diferentes possibilidades podem ocorrer. Dependendo de como são identificadas as abstrações no momento da modelagem dos objetos, a estrutura pode seguir diferentes hierarquias, todas elas corretas frente a filosofia da MOO, mas implicando em diferentes alocações de características e funcionalidades ao longo da estrutura.

É falso o pensamento de que uma determinada estrutura hierárquica, projetada tendo-se em mente um conjunto limitado de aplicações, servirá para a implementação de qualquer método de análise ou síntese aplicado a SEE. Inclusive pelo fato de que novos métodos, e também novos equipamentos e modelos são constantemente desenvolvidos, frutos de pesquisas em desenvolvimento, ou mesmo a serem ainda desenvolvidas para o setor elétrico.

Dessa forma, quando se propõe um trabalho de *Aplicação de MOO em SEE*, pretende-se desenvolver uma idéia de representação do mundo real, a qual modele o sistema de forma que possíveis expansões sejam viáveis. A estrutura não deve ser fechada, e sim prever essas expansões, mesmo que alterações na estrutura já existente se façam necessárias. Nota-se que tais alterações não devem propagar-se pelo código inteiro, causando problemas em outros pontos do programa; a estrutura deve estar preparada para alterações dessa natureza, encapsulando seus objetos e definindo adequadamente suas interfaces de comunicação para a troca de mensagens dentro da hierarquia. Segue-se o pensamento do que “um trabalho bem feito pode ser sempre melhorado”; quer dizer, um projeto caracteriza-se como bem realizado não por ser cabal, mas sim por atender a um conjunto de requisitos presentes, e dar margem a melhorias futuras, de acordo com requisitos que venham a surgir.

Sendo assim, este trabalho não tem a pretensão de apresentar uma estrutura de classes cabal, fechada, e sim propor idéias para a representação de SEE na forma de estruturas orientadas a objetos, procurando definir interfaces com um alto grau de expansibilidade.

3.5. Complexidade de Projetos Orientados a Objetos

Afirmações tais como as listadas abaixo são seguidamente encontradas no âmbito de grupos de profissionais envolvidos diretamente com MOO:

- *"O uso da MOO torna simples o projeto de um sistema computacional."*
- *"O projeto de um software, quando realizado utilizando-se a MOO, pode ser analisado de forma mais fácil que um projeto que não utiliza o paradigma."*
- *"A implementação dos códigos em um projeto OO é simples e rápida."*

Deve-se tomar cuidado quanto a essas afirmações. Inicialmente, MOO é fácil para projetistas familiarizados com o paradigma. Um projeto OO, mesmo devidamente documentado, seguindo fielmente os preceitos do paradigma, pode parecer a primeira vista complicado para um projetista acostumado a trabalhar com metodologias formais. Da mesma maneira, um projetista especialista em linguagem Fortran, por exemplo, pode ficar sem ação frente a uma manutenção ou atualização no melhor projeto orientado a objetos, implementado em linguagem C++.

Em se tratando do desenvolvimento de sistemas computacionais para o setor elétrico, estas questões ficam evidenciadas. Isto porque o uso de metodologias baseadas nas funcionalidades dos sistemas, associadas a linguagens de programação tais como Fortran, é prática comum no setor.

Assim, quando a questão em destaque é a complexidade de diferentes metodologias adotadas no desenvolvimento de sistemas computacionais, uma premissa básica a ser observada é que se tenha o mesmo nível de conhecimento sobre as diferentes instâncias em discussão. E, sob esta premissa, a MOO apresenta suas vantagens, principalmente pelo fato de permitir que se construam modelos computacionais que traduzem de forma cognitiva conceitos existentes no mundo real sob estudo.

Deve-se considerar também que o paradigma da MOO é adequado ao projeto de sistemas computacionais de médio e grande porte, naturalmente complexos (que é o caso dos SEE). A modelagem destes sistemas traz complexidades intrínsecas, independentes da metodologia que se utiliza para tanto. Nestas situações também se evidenciam as vantagens da MOO, uma vez que, desde que corretamente utilizada, permite obterem-se projetos que facilitam futuros processos de manutenção e atualização.

Além disso, o uso de MOO implica em uma mudança expressiva na forma de se pensar o desenvolvimento de um software ou sistema computacional, ainda mais se levando em consideração as práticas usuais adotadas no setor elétrico. Este fato, por si só,

tende a fazer com que o desenvolvimento utilizando MOO não seja trivial. Porém, as dificuldades são certamente diluídas quando se considera o processo como um todo, tendo em vista os bons resultados que o paradigma permite que sejam obtidos.

3.6. Reutilização de Códigos

Muitos projetos de software que utilizam MOO tendem a desenvolver novos códigos para o projeto completo, confrontando-se com códigos e rotinas já estabelecidos em determinada área, com comprovada eficiência, tanto numérica quanto em termos de desempenho computacional.

Contrapondo esta idéia, a filosofia proposta no presente trabalho prevê a reutilização de códigos e rotinas já consolidadas na área de SEE, independente das suas formas de projeto e linguagens utilizadas na sua implementação. Entende-se neste trabalho que a reutilização de códigos está relacionada ao *domínio de aplicação* do projeto. Os esforços da equipe devem ser dirigidos para questões de real interesse no escopo do projeto, considerando-se a existência de códigos especializados para a realização de determinadas tarefas pontuais. Procura-se evitar, assim, o *reprojeto*, reduzindo-se custos. Projetos paralelos podem dedicar esforços para o desenvolvimento de novos códigos OO para a substituição de códigos antigos; porém isto não impede o prosseguimento do projeto principal. Desde que uma adequada interface de comunicação entre o projeto OO e os códigos sob reutilização seja devidamente definida, o paradigma da MOO não é violado, e a troca de um código por outro se torna trivial (HAKAVIK et al., 1994; AGOSTINI et al., 2002a; AGOSTINI et al., 2002d).

No presente trabalho realiza-se a reutilização de códigos em diversos momentos. No desenvolvimento de aplicações, especificamente na aplicação de simulação dinâmica detalhada, códigos desenvolvidos anteriormente em MANZONI (1996), em linguagem C++, são reutilizados na solução das equações algébricas e diferenciais dos modelos dinâmicas de determinados elementos. A estrutura proposta para manter e tratar matrizes esparsas de grande porte, e para a solução de sistemas lineares que as envolvam, também faz uso do conceito de reutilização de códigos, reutilizando bibliotecas já consolidadas para estes fins.

3.7. Aplicações da MOO em Sistemas Elétricos (Revisão da Literatura)

Alguns trabalhos têm surgido na literatura registrando aplicações da MOO na área de SEE. A maioria trata de aplicações em fluxo de potência (NEYER et al., 1990; HAKAVIK et al., 1994; FOLEY et al., 1995; ZHOU, 1996; FUERTE-ESQUIVEL et al., 1998), encontrando-se também aplicações na área de banco de dados para SEE (FLINN et al., 1992), sistemas de distribuição (BRITTON, 1992; ZHU et al., 1997; LEE et al., 1998; LOSI et al., 2000; SILVA et al., 2000), interfaces gráficas homem - máquina (FOLEY et al., 1993), estabilidade transitória (VANTI, 1994; MANZONI, 1996; MANZONI, 1999; BRADLEY et al., 1999), diagnóstico de falhas e restauração do sistema (GAING et al., 1996; MIAO et al., 1996), planejamento (OLIVEIRA et al., 1996; HANDSCHIN et al., 1998), configuração de subestações (ATANACKOVIC et al., 1998), e solução de sistemas lineares para SEE (HAKAVIK et al., 1994; ARAUJO et al., 2000; PANDIT et al., 2001b; AGOSTINI et al., 2002). Mais recentemente alguns trabalhos surgiram relatando a modelagem computacional genérica de SEE usando MOO, com o objetivo de desenvolver aplicações integradas para o setor (AGOSTINI et al., 2000; BECKER et al., 2000; PANDIT et al., 2000; PANDIT et al., 2001a).

NEYER et al. (1990) aplicaram MOO ao problema de fluxo de potência, onde foram avaliadas as potencialidades da MOO para aplicações a SEE. A estrutura proposta parte de uma classe única *Objetos*, subdividindo-se em *Físicos* e *Conceituais*. Registram-se neste trabalho 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 1,5 vezes maior que linguagens tradicionais (Fortran 77).

FLINN et al. (1992) propuseram um banco de dados orientado a objetos para SEE. Neste trabalho relata-se que uma tentativa de se usar bancos de dados relacionais para armazenar dados de SEE não obteve sucesso, devido à complexidade dos dados a serem modelados; as exigências de complexidade justificam a aplicação de MOO. Detalha bastante a estrutura do banco de dados em si, não esclarecendo, porém, a hierarquia de classes utilizada.

FOLEY et al. (1993) apresentaram uma interface homem - máquina utilizando MOO para aplicações gráficas em SEE. Neste trabalho é proposta uma estrutura hierárquica para os elementos baseada no número de nós que compõem o elemento. Posteriormente

(FOLEY et al., 1995), os autores aplicaram esta estrutura em um programa de fluxo de potência.

HAKAVIK et al. (1994) propuseram uma nova estrutura hierárquica para os elementos dos SEE, classificando os elementos em dois grandes grupos: *Conexões* e *Barramentos*. Também avaliaram a viabilidade da utilização da MOO para manipulações de matrizes esparsas, sendo relatados problemas de desempenho em função da manutenção da flexibilidade computacional do programa; apresentam comparativos do tipo velocidade de execução *versus* flexibilidade. Os autores mostraram a possibilidade de se reutilizar códigos já consagrados para a resolução de problemas particulares, como a solução de sistemas lineares, mesmo que codificados em outras linguagens; tais códigos foram encapsulados no interior dos objetos da estrutura. Mostram também que a MOO não causa necessariamente aumento do tempo computacional. A estrutura é aplicada em um fluxo de potência.

VANTI (1994) aplicou a MOO no desenvolvimento de um módulo de simulação da dinâmica de SEE, utilizando modelagem simplificada para os elementos do SEE. O trabalho usa uma estrutura hierárquica de classes semelhante à proposta em HAKAVIK et al. (1994).

FOLEY et al. (1995) aplicaram a MOO para desenvolver uma aplicação de análise de redes. Propõe-se no trabalho 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 gráfica das estruturas.

ZHOU (1996) aplicou MOO em um programa 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) os autores desenvolveram um simulador da dinâmica de SEE, utilizando MOO. O trabalho apresenta uma estrutura de classes bastante *horizontal*, eliminando-se os vários níveis hierárquicos normalmente utilizados nas estruturas para representação de SEE. Aplica MOO também na resolução de sistemas lineares esparsos, obtendo bons desempenhos computacionais. Os autores apresentam tempos de processamento inferiores aos tempos da dinâmica simulada para sistemas de até 730 barras, demonstrando a possibilidade de simulações em tempo real para estes sistemas.

GAING et al. (1996) aplicaram MOO no projeto de uma ferramenta para restauração do sistema após a ocorrência de contingências. Justificam a MOO pela flexibilidade e

mantenabilidade do software projetado. O trabalho é uma verdadeira aula sobre o método OMT, seguindo-o detalhadamente, e apresentando todas as fases do projeto. A estrutura de classes mostrada representa não somente o sistema em si, representado de forma simplificada por um banco de dados, mas também o problema da restauração, apresentando diagramas de classes hierarquizando o problema. Representa etapas do procedimento de restauração como classes. Para as implementações foi utilizada a linguagem C++.

ZHU et al. (1997) aplicaram a MOO para desenvolver uma estrutura representativa dos SEE para análise de sistemas de distribuição. Este trabalho é apresentado com base no método de projeto OMT, descrevendo sua metodologia de uma forma geral, e utilizando sua notação gráfica para a apresentação das estruturas. A estrutura define *Elementos Shunt, Ramos* e *Barras*, e separadamente *Elementos de Proteção*. Tais elementos são agregados de forma a dar origem a objetos como *Subestação*.

HANDSCHIN et al. (1998) aplicaram a MOO em um problema de planejamento do sistema de transmissão em um ambiente desregulamentado, modelando tanto os elementos físicos dos SEE quanto o seu mercado. A estrutura proposta classifica os elementos de acordo com o seu número de nós, descendendo daí os elementos do sistema. O trabalho propõe também que os métodos de análise do sistema, que por sua vez contêm os algoritmos numéricos propriamente ditos, sejam separados da representação dos elementos do sistema, modelados como *objetos de aplicação*. Utiliza o OMT como metodologia de projeto.

FUERTE-ESQUIVEL et al. (1998) desenvolveram um programa de fluxo de potência utilizando MOO na modelagem do sistema. São modelados também dispositivos FACTS. A estrutura de classes proposta, a exemplo de MANZONI 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. Apresenta resultados de desempenho do código orientado a objetos implementado em linguagem C++, da ordem de 17% mais lento em relação a linguagens tradicionais (Fortran 77).

LEE et al. (1998) propuseram uma nova estratégia para a restauração de sistemas de distribuição, utilizando MOO no projeto do software para facilitar futuras manutenções e reutilizações dos seus códigos. A estrutura de classes apresentada é específica para o problema, modelando alimentadores, chaves e áreas de alimentação, utilizando derivação e agregação. O trabalho segue o método de projeto OMT, apresentando os três modelos (objetos, dinâmico e funcional) para o software.

ATANACKOVIC et al. (1998) apresentaram uma metodologia para a configuração de subestações, desenvolvida utilizando MOO. A estrutura hierárquica desenvolvida é específica para o problema em questão, apresentando diagramas de classes para modelar elementos internos de uma subestação, tais como chaves seccionadoras, transformadores e equipamentos compensadores. É utilizado como base para o trabalho o método de Booch (BOOCH, 1994; BOOCH, 1998).

ZHU et al. (1999) introduziram conceitos de *padrões de projeto* (*design patterns*) na modelagem de SEE. Neste trabalho os autores enfatizam a dificuldade de se chegar a uma estrutura *ótima* para a modelagem do sistema, a qual atenderia uma ampla gama de aplicações. O trabalho mostra alguns conceitos de padrões de projeto, assim como aplicações destes conceitos na modelagem de SEE, utilizando notações do método OMT.

BRADLEY et al. (1999), em um trabalho onde desenvolveram um sistema para a definição de seqüências de contingências para análise de estabilidade transitória, ressaltam as dificuldades de se projetar estruturas de dados complexas em linguagens tradicionais. Observam ainda que diferentes níveis de abstração no momento do desenvolvimento das estruturas levam a diferentes níveis de modelagem. O trabalho propõe o uso de agregação para se representar esses diferentes níveis. Para driblar possíveis conflitos na modelagem, propõem uma hierarquia de classes baseada na funcionalidade dos elementos do sistema, e não na sua estrutura física real. O trabalho não utiliza notações claras para essa hierarquia, o que dificulta sua compreensão.

No ano de 2000 alguns trabalhos foram publicados utilizando MOO para desenvolvimentos na área de sistemas de distribuição de energia elétrica. LOSI et al. (2000) desenvolveram um fluxo de potência para sistemas de distribuição utilizando conceitos da MOO. Neste trabalho os elementos dos sistemas de distribuição são representados por uma estrutura de classes baseada em *conexões* e *nós*. Os autores ressaltam a importância de se utilizar um banco de dados único, orientado a objetos, para um conjunto de aplicações para SEE.

Em SILVA et al. (2000), os autores descrevem uma aplicação de gerenciamento de sistemas de distribuição, na qual utilizou-se a MOO para a implementação do projeto. Novamente neste trabalho ressaltam-se características do método OMT, relativas aos seus três modelos: Objetos, Dinâmico e Funcional. Os autores afirmam que o modelo de objetos normalmente tem uma maior relevância na documentação de um projeto OO, pois representa os elementos do sistema que está sendo modelado e suas conexões, servindo de base para os outros dois modelos. Ressalta-se também a necessidade de se desenvolver bancos de dados orientados a objetos para as aplicações em SEE, e que isto se

dará naturalmente, acompanhando o crescente uso da MOO no projeto e desenvolvimento de ferramentas computacionais para o setor elétrico. Quanto a estrutura de classes, organiza-a em três níveis: elétrico, topológico e geográfico. Os elementos físicos do sistema são alocados no nível de dispositivos elétricos, classificados pelo número de terminais. Um módulo, chamado *processador topológico* é acionado sobre o nível elétrico, originando o nível topológico, de acordo com a posição dos disjuntores do sistema; assim são criados objetos do tipo nó e ramo, definindo a conexão do sistema. No nível geográfico são representadas as interconexões entre os módulos que integram o gerenciamento dos sistemas de distribuição.

Recentemente, alguns trabalhos surgiram propondo a modelagem genérica de SEE usando conceitos de MOO, no intuito de possibilitar o desenvolvimento de diversas ferramentas computacionais para o setor, de forma integrada. AGOSTINI et al. (2000) propuseram uma estrutura de classes genérica para a representação dos elementos físicos de um sistema elétrico, classificando-os de acordo com seu tipo de conexão (nó ou ramo). Diagramas de classe, de estado e funcionais são apresentados no trabalho, modelando uma aplicação de fluxo de potência linearizado e uma de simulação da dinâmica de SEE. O método OMT foi utilizado para as notações gráficas. Este trabalho concentrou os estudos realizados para o exame de qualificação ao doutorado do autor do presente trabalho de tese.

BECKER et al. (2000) tratam do problema da integração entre aplicações. O artigo apresenta os esforços que estão sendo realizados no Electric Power Research Institute (EPRI) sobre o tema. Uma *task force* foi montada nesta instituição, Control Center Application Program Interface (CCAPI), que vem desenvolvendo o *Common Information Model* (CIM), um conjunto de classes para a representação de SEE. O CIM está dividido em 5 submodelos, *wires model, SCADA model, load model, energy scheduling model* e *generation model*. A UML é utilizada para a documentação do projeto. Neste trabalho defende-se o conceito de *metadados* para uma melhor integração entre aplicações já existentes, ou mesmo entre novas aplicações. Dessa forma interfaces de comunicação são desenvolvidas, encapsulando módulos prontos, e a troca de informação entre estes módulos passa a ser realizada através destas novas interfaces.

Em PANDIT et al. (2000), os autores defendem a divisão entre a modelagem dos elementos físicos de um SEE e suas aplicações. Na modelagem dos elementos físicos, relatam que esta 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, divide 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 do sistema). Também da classe *network* derivam as aplicações, tais como fluxo de potência e análise de curto-circuito. O artigo utiliza a notação gráfica do método de Booch. No ano seguinte, outro trabalho dos mesmos autores (PANDIT et al., 2001) detalha a etapa do processador de topologia de rede, projetado segundo os conceitos da MOO.

Questões mais específicas, mas ainda relacionadas à aplicações na área de SEE, tal como a solução de sistemas lineares esparsos de grande porte, também têm tido destaque na literatura, em se tratando de modelagem orientada a objetos. Artigos mais antigos, como HAKAVIK et al. (1994) já dedicavam atenção a esta questão. Neste trabalho os autores falam da reutilização de códigos especializados nesta tarefa, através do seu encapsulamento em classes específicas nos novos códigos. Em AGOSTINI et al. (2002a) os autores também defendem esta idéia, desenvolvendo uma estrutura de classes genérica para armazenar e tratar matrizes esparsas de grande porte. A estrutura concebida permite encapsular diferentes métodos para a solução de sistemas lineares, através de classes identificadas como *estratégias*; utilizou-se para isso o padrão de projeto *Strategy* (GAMMA et al., 2000). O assunto é tratado também em FUERTE-ESQUIVEL et al. (1998) e MANZONI et al. (1999).

Em ARAUJO et al. (2000) e PANDIT et al. (2001b) os autores desenvolvem novas metodologias para a solução de sistemas lineares esparsos e de grande porte, as quais têm seus projetos e implementações realizados segundo o paradigma da MOO.

Em suma, a grande maioria dos trabalhos existentes trata de aplicações específicas, desenvolvendo estruturas de classes para aplicações particulares em um ou outro tema relacionado a área de SEE. Mesmo trabalhos mais recentes, que têm tratado de modelagem genérica de SEE, não apresentam ainda uma proposta efetiva, suficientemente ampla e de fácil aplicação para a modelagem das mais diversas instâncias destes sistemas. Assim, neste trabalho de tese apresenta-se uma proposta neste sentido, buscando-se com isso o desenvolvimento integrado de ferramentas computacionais de apoio ao planejamento e à operação dos sistemas elétricos.

CAPÍTULO 4

4. MODELAGEM DO SISTEMA: ABSTRAÇÕES

De acordo com os conceitos da MOO, pode-se definir uma *abstração* como sendo uma visão particular de uma parte de um conjunto mais amplo, onde alguns aspectos (aqueles relevantes para o problema em questão) são ressaltados, e outros negligenciados. Deixa-se de lado momentaneamente o conjunto completo, para que os esforços de projeto sejam voltados exclusivamente para a modelagem das entidades que fazem parte daquele escopo. O uso deste conceito na modelagem de sistemas orientados a objetos dá subsídios a uma maior modularidade do sistema, uma vez que diferentes abstrações trocam informações entre si através de interfaces devidamente identificadas, e o funcionamento interno de uma abstração em particular é irrelevante para as outras. A abstração pode ser utilizada em diferentes níveis, desde a divisão do sistema completo em alguns pacotes específicos, segundo um critério qualquer, até a modelagem dos atributos e métodos internos (ou privados) de uma determinada classe, quando esta é abstraída do restante do sistema.

Neste trabalho, o processo de criação das estruturas de classes representativas das diversas entidades do sistema como um todo, é dividido em abstrações distintas, representadas por pacotes. A Figura 4-1 mostra estas abstrações, e suas dependências.

Observa-se na nesta figura as três principais abstrações identificadas: Sistema Elétrico, Aplicações e Facilidades Computacionais, englobadas pela abstração mais geral Ferramentas Computacionais. A abstração do Sistema Elétrico congrega as classes que representam os elementos físicos formadores de um SEE. As estruturas de classes pertencentes a esta abstração contêm classes como *Barra, Linhas de Transmissão, Transformadores, Cargas, Unidades de Geração*, etc., as quais estão representadas na figura

pela sua classe base *C_Device*. Esta classe serve de base para todos os elementos físicos do SEE. Nesta abstração ainda aparece a classe *C_Power_System*, que representa o sistema elétrico em si; ela é formada pela composição dos seus diversos elementos físicos.

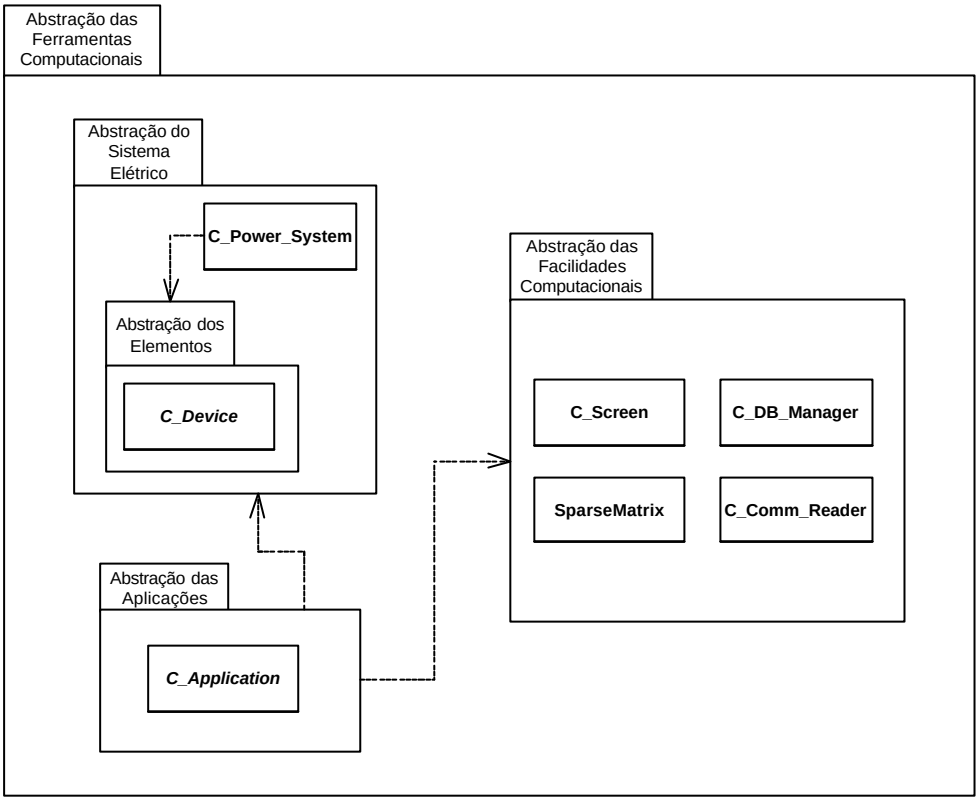


Figura 4-1 – Abstrações na Modelagem Computacional de SEE

A Abstração das Aplicações concentra as classes que representam metodologias de análise ou síntese (tratadas neste trabalho como sinônimo de *aplicações*), sendo todas derivadas de uma classe base chamada *C_Application*. As aplicações *dependem* do sistema elétrico, pois os algoritmos de análise e síntese serão executados sobre uma base de dados representativa do sistema sob estudo.

Ainda na Figura 4-1 aparece a Abstração das Facilidades Computacionais. No âmbito desta abstração modelam-se as funcionalidades que devem existir no sistema computacional, para facilitar o funcionamento da ferramenta computacional em si. Observam-se aí classes responsáveis pelo gerenciamento de telas (*C_Screen*), leituras de comandos para o software (*C_Comm_Reader*) e leitura de dados dos SEE (*C_DB_Manager*), e representação de matrizes (*SparseMatrix*). As aplicações dependem também das facilidades computacionais.

No nível mais amplo do processo identifica-se a Abstração das Ferramentas Computacionais. Nesta abstração concentram-se esforços na obtenção das ferramentas finais

para o setor elétrico, construídas utilizando-se como base as estruturas de classes das abstrações anteriores. Basicamente, uma ferramenta computacional é formada por um objeto SEE, uma ou mais aplicações sendo executadas sobre o sistema elétrico, e alguns objetos realizando tarefas essencialmente matemáticas ou computacionais.

Assim como as próprias estruturas que as compõem, as abstrações não são *fechadas*, isto é, admitem a inclusão de novas entidades em seus respectivos limites. Principalmente no caso da abstração das facilidades computacionais, caso futuros desenvolvimentos identifiquem funcionalidades que possam ser consideradas como facilidades computacionais de uso geral em ferramentas orientadas a objetos, suas classes podem e devem vir a fazer parte desta abstração.

Neste capítulo cada abstração é discutida em seus aspectos gerais. O detalhamento específico das classes que compõem cada pacote, e suas respectivas estruturas, são apresentadas e discutidas nos capítulos subseqüentes.

4.1. Abstração do Sistema Elétrico

A Abstração do Sistema Elétrico delimita a modelagem dos elementos físicos dos SEE. A abstração pode ser vista com mais detalhes na Figura 4-2.

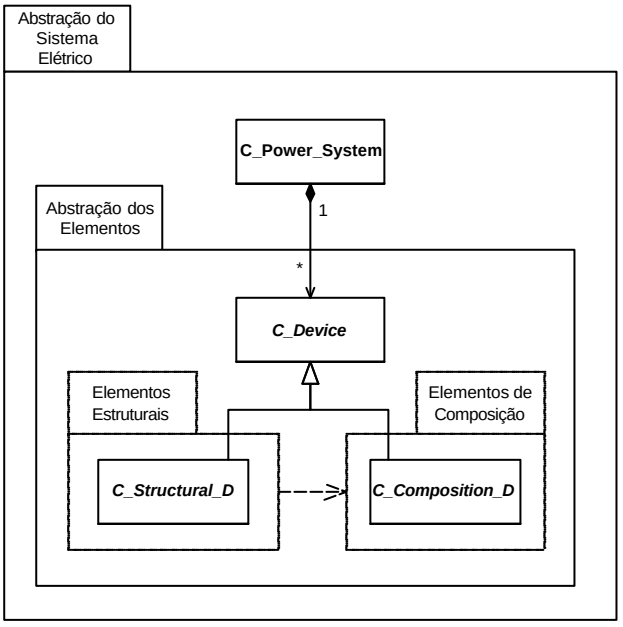


Figura 4-2 – Abstração do Sistema Elétrico

No âmbito desta abstração identificam-se duas entidades: a classe *C_Power_System* e a Abstração dos Elementos. A classe *C_Power_System* modela o SEE em si. Ela é for-

mada pela composição de diversos elementos, os quais derivam todos de uma classe base *C_Device*. O detalhamento da classe *C_Power_System*, assim como de todas as outras classes relacionadas à abstração do sistema elétrico, é apresentado no Capítulo 5 (seção 5.3).

A segunda entidade identificada é a Abstração dos Elementos do SEE. Nesta abstração estão situadas as estruturas de classes que representam os diversos elementos formadores dos sistemas elétricos. A classe abstrata *C_Device*, classe de mais alto nível nesta abstração, serve de base para todas as outras classes que modelam elementos físicos das mais diversas naturezas.

4.1.1. Modelagem dos Elementos Físicos

Um ponto chave na concepção de uma estrutura genérica para a representação de SEE é o critério 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.

Conforme já descrito no Capítulo 3 (seção 3.7), alguns critérios já foram utilizados para esta classificação em trabalhos disponíveis na literatura. No presente trabalho de tese a classificação dos elementos físicos é baseada na estrutura real do sistema elétrico. Esta classificação leva em conta em primeira instância o número de conexões a 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).

Este critério falha, porém, no momento de se classificar elementos físicos que não possuem conexão(ões) direta(s) a alguma barra, mas que, quando agregados de uma certa maneira, originam elementos série ou derivação (ou outro tipo qualquer, com mais de duas conexões). Como exemplo, pode-se citar a unidade de geração, elemento tipicamente conectado em derivação no 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 a barras, e portanto suas classes não podem ser situadas em uma estrutura hierárquica baseada em número de conexões.

Sendo assim, neste trabalho duas estruturas de classes são propostas para a representação dos elementos físicos do sistema, delimitadas respectivamente pelas abstra-

ções Elementos Estruturais e Elementos de Composição, conforme mostrado anteriormente na Figura 4-2. A primeira situa os elementos ditos *estruturais*, por formarem a estrutura básica da rede elétrica, conectados através das barras. Nesta abstração, todos os elementos derivam da classe abstrata *C_Structural_D*. A segunda abstração situa os elementos ditos *de composição* (derivados todos da classe *C_Composition_D*) que, apesar de não possuírem conexões diretas a rede elétrica, formarão elementos estruturais através do mecanismo de composição da MOO (Figura 4-3).

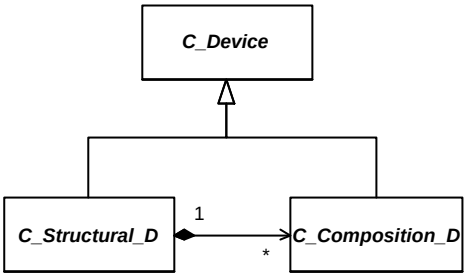


Figura 4-3 – Elementos Físicos Estruturais e de Composição

Observa-se nesta figura 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 dependência entre as abstrações. Esta é a dependência representada na Figura 4-2.

As estruturas de classes, tanto dos elementos estruturais, como dos elementos de composição, são apresentadas e discutidas em detalhes no Capítulo 5 (seções 5.1 e 5.2, respectivamente).

4.2. Abstração das Aplicações

A Abstração das Aplicações define um escopo para a modelagem das classes representativas das mais diversas metodologias de análise e síntese, aplicadas aos SEE. A abstração pode ser vista com mais detalhes na Figura 4-4. Nesta abstração mostra-se uma classe chamada *C_Application*, a qual serve como base para a estrutura hierárquica das classes representativas das aplicações.

Além do critério para a classificação dos elementos físicos, um segundo ponto chave na definição de uma estrutura de classes genérica para SEE é a maneira de se representar 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) podem ser divididas em dois grupos básicos: *características reais* e *características funcionais*. As características reais estão relacionadas à existência dos elementos reais no sistema elétrico, tais como suas conexões às barras ou entre si, seus parâmetros, etc. As características funcionais estão relacionadas às funcionalidades desempenhadas por cada elemento, quando considerada uma determinada metodologia de análise ou síntese 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 os elementos físicos em si, os quais são independentes das inúmeras metodologias de análise e síntese que podem ser executadas sobre o sistema.

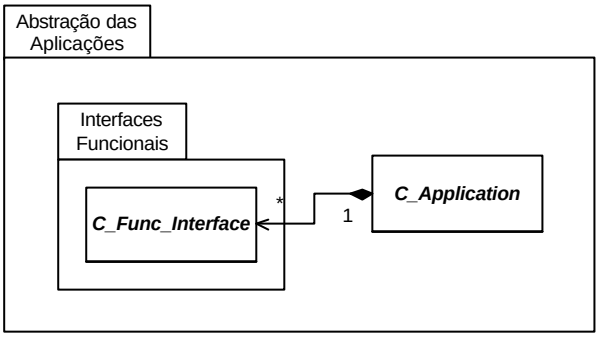


Figura 4-4 – Abstração das Aplicações

Assim, torna-se imprescindível a definição de um arranjo de classes que permita que ambos os conjuntos de características sejam representados, de forma que as características funcionais possam variar independentemente das características reais dos elementos. Neste trabalho de tese, propõe-se o uso de um padrão de projeto orientado a objetos, chamado Adapter, na definição de um 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 (apresentada em detalhes no Capítulo 2), 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 compo-

sição, interfaces funcionais e aplicações podem ser observadas na Figura 4-5. Todo elemento físico (seja estrutural ou de composição) possui um conjunto de interfaces funcionais, cada uma relacionada a uma aplicação em específico. Cada interface funcional pertence ao escopo de uma aplicação, e conecta-se a um elemento físico, complementando-o na aplicação em questão.

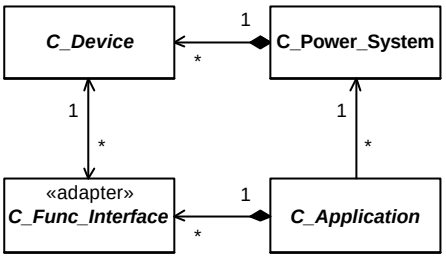


Figura 4-5 – Elementos, Interfaces Funcionais e Aplicações

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 ou 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; em tempo de execução, cada aplicação cria e destrói seu conjunto de interfaces funcionais para os elementos físicos já existentes na memória do computador. Diversas aplicações podem existir ao mesmo tempo, trabalhando de forma sincronizada, pois os elementos físicos podem ter mais de uma interface funcional associada.

O detalhamento das estruturas de classes, tanto das aplicações, como das interfaces funcionais dos elementos físicos, é apresentado no Capítulo 6.

4.3. Abstrações das Ferramentas e das Facilidades Computacionais

Os produtos concretos a serem obtidos com a utilização da filosofia de projeto apresentada neste trabalho são ferramentas computacionais para o setor elétrico, materializadas 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 intrinsecamente computacionais. Pode-se citar tarefas de gerenciamento de telas, leitura e escrita de arquivos de dados, monitoração de desempenhos

computacionais parciais e globais, etc. Incluem-se aí entidades com conotações matemáticas, tais como matrizes, vetores, sistemas lineares, etc.

Todas estas entidades são abstraídas da modelagem do SEE em si, e delimitadas por uma abstração denominada Facilidades Computacionais. Essas facilidades são projetadas e implementadas sob a forma de pacotes computacionais independentes, sempre sob o paradigma da MOO. Os pacotes assim criados são utilizados no projeto das aplicações de análise e síntese, e finalmente na construção das ferramentas computacionais.

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 este o objeto de mais alto nível no código. Neste trabalho as ferramentas computacionais são representadas por uma classe formada por um SEE, uma ou mais aplicações que podem ser executadas sobre o SEE, e um conjunto de objetos realizando tarefas puramente computacionais. A abstração das ferramentas computacionais é apresentada na Figura 4-6. Nesta figura, uma ferramenta genérica é representada pela classe *C_OOTPS* (Ferramenta Orientada a Objetos para Sistemas de Potência).

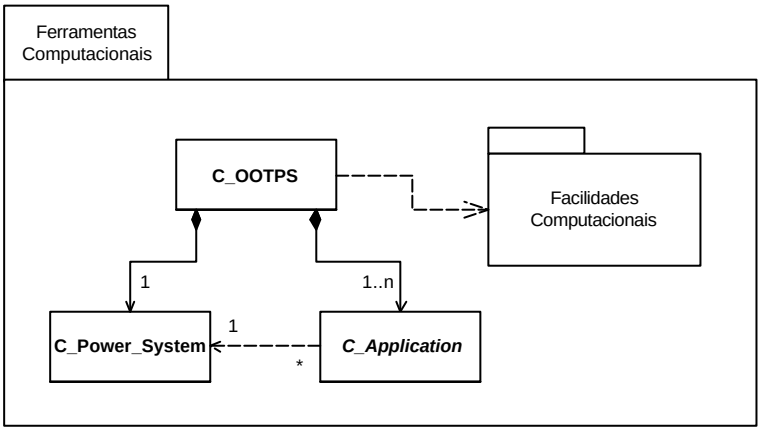


Figura 4-6 – Ferramentas Computacionais

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 preocupa-se em como executar 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 comandos), etc., o que ela faz através das facilidades computacionais.

As ferramentas e pacotes de facilidades computacionais projetadas e implementadas neste trabalho, são apresentadas e descritas em detalhes no Capítulo 7.

CAPÍTULO 5

5. MODELAGEM DOS ELEMENTOS FÍSICOS

Uma estrutura de classes genérica para a representação de SEE deve cumprir alguns requisitos:

- → permitir a representação de todos os elementos de um SEE, desde seus elementos físicos mais básicos, tais como barras ou linhas de transmissão, até elementos conceituais, tais como as metodologias de análise e síntese a serem aplicadas sobre estes sistemas;
- → permitir expansões, objetivando representar elementos que poderão vir a ser criados e modelados no futuro;
- → facilitar estas expansões e manutenções, fazendo com que futuras alterações, tanto em nível de projeto, quanto 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 execuções das metodologias de aplicação.

Frente a estes requisitos, observa-se que a definição de uma estrutura de classes genérica para a representação de SEE é uma tarefa árdua (ZHU et al., 1999). Alguns trabalhos surgiram nos últimos anos propondo formas para classificar os elementos físicos de um SEE, segundo uma hierarquia de classes. Pode-se encontrar hierarquias baseadas na estrutura física do sistema (MANZONI et al., 1999), no número de nós dos elementos

(FOLEY et al., 1995; HANDSCHIN et al., 1998; SILVA et al., 2000), ou estruturas mais horizontais, com os elementos todos em um mesmo nível hierárquico (FUERTE-ESQUIVEL et al., 1998; PANDIT et al., 2000), entre outras.

Neste trabalho duas grandes estruturas são concebidas para a representação dos elementos físicos dos SEE: uma para a representação dos elementos estruturais do sistema, os quais formam a estrutura física da rede elétrica, conectados através das barras; e outra para a representação dos elementos de composição, os quais, apesar de não estarem conectados diretamente em alguma barra, formam por composição alguns elementos estruturais. Na Figura 5-1 estas duas estruturas são representadas pelas suas classes bases *C_Structural_D* e *C_Composition_D*, respectivamente.

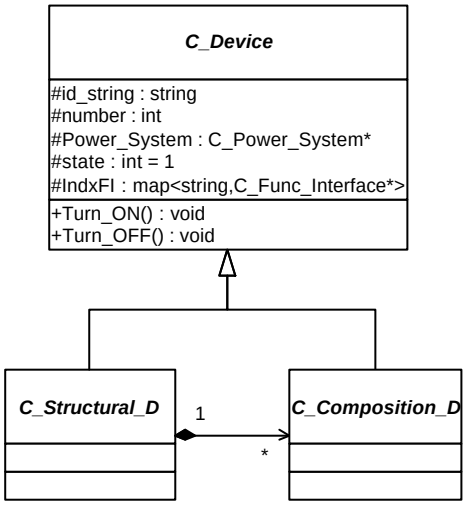


Figura 5-1 – Elementos Físicos dos SEE

Todos os elementos físicos dos SEE derivam de uma classe abstrata *C_Device*. Esta classe define os seguintes atributos e métodos principais:

→ *id_string*: identificador do tipo de elemento;
→ *number*: número de identificação do elemento;
→ *Power_System*: apontador para o objeto Sistema Elétrico, ao qual todos os elementos pertencem;
→ *state*: estado do elemento, ativo ou inativo;
→ *IndxFI*: lista de interfaces funcionais do elemento físico;
→ *Turn_ON()*: habilita o elemento;
→ *Turn_OFF()*: desabilita o elemento.

Cada elemento possui um identificador do tipo *string* (atributo *id_string*), o qual identifica o tipo (ou a classe) do elemento. Este identificador é sempre atribuído pela própria classe do elemento, no momento da sua criação (no método construtor), e deve con-

ter o nome da classe, sem o prefixo “C_”. Com ele é possível identificar cada tipo de elemento em uma lista genérica de elementos (por exemplo, é possível identificar as unidades de geração ou as cargas conectadas em uma barra, percorrendo-se a lista genérica de elementos conectados em derivação na referida barra – *IndxShunt* de *C_Bar*; a classe *C_Bar* é apresentada logo a seguir). Cada elemento recebe ainda um número de identificação (armazenado em *number*); este número não é necessariamente único para cada elemento (por exemplo, diversas unidades de geração podem conter o mesmo número de identificação, desde que conectadas à barras distintas).

O estado do elemento (ativo ou inativo) é controlado pelo atributo *state*, o qual é alterado utilizando-se os métodos públicos *Turn_ON()* e *Turn_OFF()*.

O atributo *IndxFI* é uma lista de apontadores para os objetos tipo interfaces funcionais. Esta é uma lista indexada, onde a chave de indexação é o nome da aplicação a que a interface refere-se. Todos os elementos físicos do sistema possuem este atributo, herdado de *C_Device*. Esta lista indexada permite que seja acessada diretamente uma determinada interface funcional de um elemento físico, relativa a uma aplicação em específico, a partir do próprio elemento físico.

Nas seções seguintes as duas estruturas de classes (estruturais e de composição) são apresentadas e discutidas, juntamente com suas principais classes.

5.1. Elementos Estruturais

Neste trabalho a classificação dos elementos estruturais dos SEE é baseada na sua estrutura real. Esta classificação considera em primeira instância o número de conexões a objetos do tipo barra que cada elemento apresenta. Na Figura 5-2 apresenta-se a hierarquia de classes para os elementos estruturais.

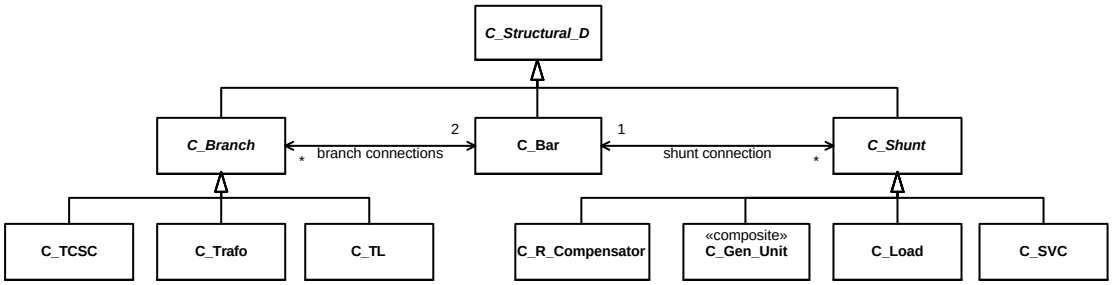


Figura 5-2 – Elementos Físicos Estruturais

A hierarquia descende da classe abstrata *C_Structural_D*, a qual serve como base para todos os elementos estruturais do sistema. Identifica-se na hierarquia três classes principais, que descendem diretamente de *C_Structural_D*: *C_Bar*, representando as barras; *C_Branch*, representando os elementos série (duas conexões); e *C_Shunt*, representando os elementos em derivação (uma conexão). A maioria dos elementos estruturais descende destas duas últimas classes abstratas.

Neste trabalho apenas elementos série e em derivação são modelados, pois normalmente elementos com mais de duas conexões (transformadores de três enrolamentos, *Unified Power Flow Controller* – UPFCs, etc.) são representados por um ou mais elementos série e/ou em derivação, modelando suas etapas assim conectadas. Porém a estrutura prevê a modelagem direta destes elementos. Uma expansão pode ser realizada, acrescentando-se uma classe *C_Nport* no mesmo nível hierárquico de *C_Branch* e *C_Shunt*, derivando-a também diretamente de *C_Structural_D*. Esta classe serve de base para todos os elementos com *n* conexões.

O critério "número de conexões a barras", para a classificação dos elementos na estrutura, considera apenas as conexões das etapas de potência dos equipamentos, desconsiderando as conexões de sinais de controle. Um transformador com troca de derivação automática sob carga, operando como um controlador remoto de tensão, pode estar "conectado" a uma terceira barra, que está tendo sua tensão controlada pelo transformador. Mesmo assim, este transformador ainda é considerado um elemento série.

- **Classe *C_Bar***

A classe *C_Bar* modela as barras de um sistema elétrico. Estes objetos servem de pontos de conexão para a formação da estrutura básica da rede elétrica. A Figura 5-3 mostra a classe em detalhes.

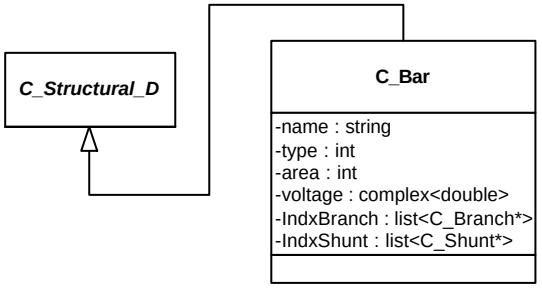


Figura 5-3 – Classe *C_Bar*

Os principais atributos de *C_Bar* são:

→ *name*: *string* com o nome da barra no sistema;
→ *type*: tipo da barra (carga, geração, referência, etc.);
→ *area*: área do sistema a qual pertence a barra;
→ *voltage*: tensão, em pu (coordenadas retangulares);
→ *IndxBranch*: lista de elementos série conectados na barra;
→ *IndxShunt*: lista de elementos em derivação conectados na barra.

As ligações entre os objetos tipo *C_Branch* / *C_Shunt* às barras são bidirecionais, conforme pode ser observado na Figura 5-2. Assim, os atributos *IndxShunt* e *IndxBranch* são listas que armazenam apontadores para os elementos em derivação e série, respectivamente, conectados a uma determinada barra. Isto permite que a estrutura física da rede elétrica seja facilmente percorrida.

5.1.1. Elementos Série

- **Classe *C_Branch***

A classe *C_Branch* é uma classe abstrata, que serve de base para todos os elementos conectados em série na rede elétrica (elementos conectados a duas barras distintas). A classe é mostrada na Figura 5-4.

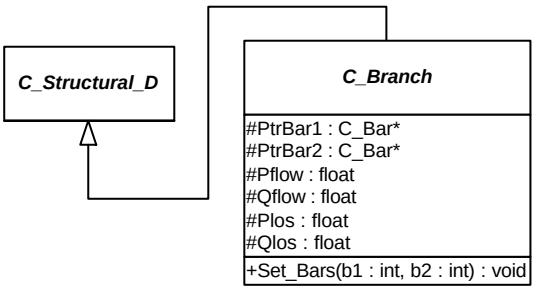


Figura 5-4 – Classe *C_Branch*

Os principais atributos e métodos da classe são:

→ *PtrBar1*, *PtrBar2*: apontadores para os objetos barra, aos quais o elemento está conectado;
→ *Pflow*, *Qflow*: fluxos de potência no elemento série;
→ *Plos*, *Qlos*: perdas no elemento série;
→ *Set_Bars()*: método para conexão do elemento às suas barras terminais.

A conexão dos elementos série às suas barras, assim como no caso dos elementos em derivação, mostrados a seguir na seção 5.1.2, é feita automaticamente no momento da leitura dos dados, através de métodos do tipo *Set_Bars()*. Para isso, a ordem de leitura dos dados é relevante, ou seja, elementos conectados a outros elementos devem ser lidos após seus elementos de conexão (as barras, por exemplo, devem ser lidas antes das linhas de transmissão e dos transformadores).

Os atributos *Pflow* e *Qflow* armazenam os fluxos de potência ativa e reativa, respectivamente, no terminal da barra 1, com sinal positivo para o sentido "da barra 1 para a barra 2" (caso se queira os valores dos fluxos no terminal da barra 2, deve-se diminuir o valor das perdas ativa e reativa, dos valores dos respectivos fluxos na barra 1).

- **Classe *C_TL***

A classe *C_TL* representa as linhas de transmissão do sistema elétrico. A classe é mostrada na Figura 5-5.

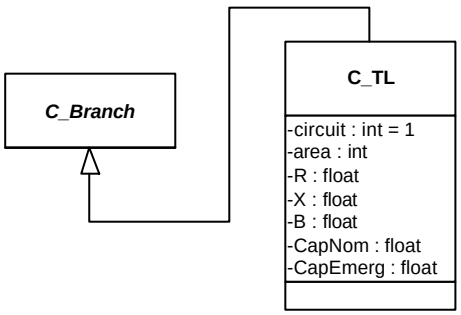


Figura 5-5 – Classe *C_TL*

Os principais atributos da classe são:

→ *circuit*: número do circuito em paralelo (deve ser diferente de 1, caso seja a segunda linha ou superior conectada entre duas mesmas barras);

→ *area*: área do sistema a qual pertence a linha;

→ *R*, *X*, *B*: parâmetros concentrados da linha: resistência, reatância e susceptância total, respectivamente (em pu);

→ *CapNom*, *CapEmerg*: capacidades de carregamento nominal e de emergência da linha, respectivamente (em MVA).

- **Classe *C_Trafo***

A classe *C_Trafo* representa os transformadores (trafos) do sistema elétrico. A classe é mostrada na Figura 5-6.

Os principais atributos da classe são:

→ *circuit*: número do circuito em paralelo (deve ser diferente de 1, caso seja o segundo trafo ou superior conectado entre duas mesmas barras);

→ *area*: área do sistema a qual pertence o trafo;

→ *R*, *X*: resistência e reatância do trafo, respectivamente (em pu);

→ *tap*: derivação de operação do trafo;

→ *tapmim*, *tapmax*: limites de variação da derivação, para trafos com variação de derivação sob carga (LTC);

→ *tapsteps*: número de posições intermediárias entre os limites de derivações, para trafos tipo LTC;

→ *angle*: ângulo de defasagem, para trafos defasadores (em graus);

→ *CapNom*, *CapEmerg*: capacidades de carregamento nominal e de emergência do trafo, respectivamente (em MVA).

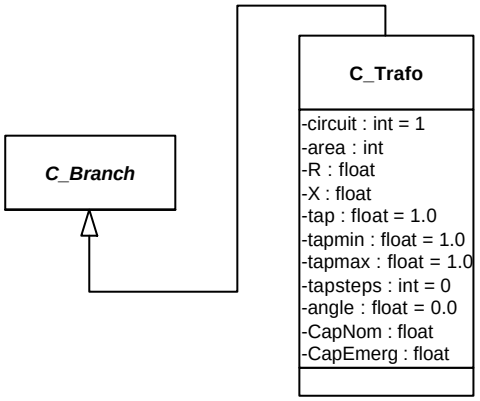


Figura 5-6 – Classe *C_Trafo*

Considerando-se que as diferenças em termos de parâmetros são pequenas entre trafos comuns, LTCs e defasadores, utiliza-se apenas uma classe para a representação destes elementos. Diferenças em termos de comportamentos frente às aplicações são modeladas nas interfaces funcionais para transformadores, conforme será explicitado no Capítulo 6.

- **Classe *C_TCSC***

Esta classe está prevista na estrutura para a representação de equipamentos FACTS, do tipo *thyristor controlled series capacitor* (TCSC). Porém a classe *C_TCSC* não foi desenvolvida neste trabalho, e portanto não será descrita.

A exemplo dos TCSCs, qualquer equipamento FACTS com características de conexão série na rede elétrica deve ser modelado em um mesmo nível hierárquico que as linhas de transmissão e os transformadores, derivando diretamente de *C_Branch*.

5.1.2. Elementos em Derivação

- **Classe *C_Shunt***

A classe *C_Shunt* é uma classe abstrata, que serve de base para todos os elementos conectados em derivação na rede elétrica (elementos conectados a uma barra apenas). A classe é mostrada na Figura 5-7.

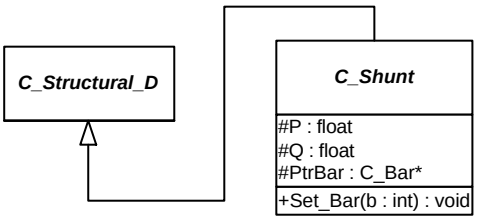


Figura 5-7 – Classe *C_Shunt*

Os principais atributos e métodos da classe são:

→ *P*, *Q*: potências ativa e reativa do elemento em derivação;
→ *PtrBar*: apontador para o objeto barra, ao qual o elemento está conectado;
→ *Set_Bar()*: método para a conexão do elemento à sua barra terminal.

Os valores de P são positivos para potências ativas geradas, e negativos para potências ativas consumidas. Os valores de Q são positivos para potências reativas capacitivas, e negativos para potências reativas indutivas.

- **Classe *C_Load***

Esta classe modela as cargas nas barras do sistema, representadas por um modelo polinomial. A classe é mostrada na Figura 5-8.

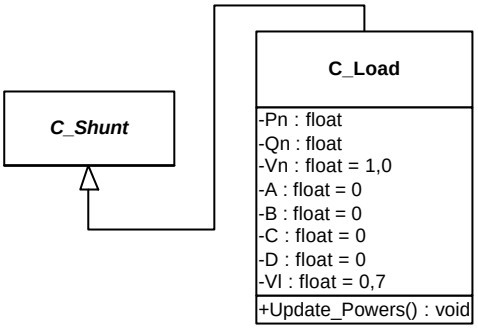


Figura 5-8 – Classe *C_Load*

Os principais atributos e métodos da classe são:

→ *Pn*, *Qn*: potências ativa e reativa nominais, respectivamente;

→ *Vn*: tensão nominal (na qual as potências nominais foram medidas), em pu;

→ *A*, *C*: parcelas de carga ativa e reativa, respectivamente, que variam linearmente com a magnitude da tensão, em % (comportamento de "corrente constante");

→ *B*, *D*: parcelas de carga ativa e reativa, respectivamente, que variam com o quadrado da magnitude da tensão, em % (comportamento de "impedância constante");

→ *Vl*: tensão abaixo da qual as parcelas relativas a comportamento de potência constante são transformadas em impedância constante, em pu;

→ *Update_Powers()*: atualiza os valores de P e Q da carga.

No modelo polinomial, as cargas podem ter três tipos principais de comportamento: potência constante (fazendo-se *A=B=C=D=0,0*), corrente constante (com *A=C=1,0* e *B=D=0,0*) ou impedância constante (com *A=C=0,0* e *B=D=1,0*); ou ainda uma combinação qualquer destes comportamentos, ajustando-se os parâmetros *A*, *B*, *C* e *D*. Por definição, as cargas são modeladas inicialmente com comportamento de potências constantes. Caso se deseje modificar este comportamento, deve-se alterar os parâmetros *A*, *B*, *C* e *D* do objeto (através de métodos específicos de interface da classe, tipo *Set....()*).

Se a tensão da barra onde a carga esta conectada baixar além de um determinado valor limite, as parcelas relativas ao comportamento tipo potência constante são alteradas para impedância constante. Este valor limite de tensão pode ser alterado para cada carga, através do atributo *Vl*. Seu valor inicial é 0,70pu (ou 70%) (CEPEL, 1999a).

O método *Update_Powers()* atualiza os valores de *P* e *Q* da carga (potências de carga, representadas por atributos herdados de *C_Shunt*), de acordo com suas potências e

tensão nominal, seus parâmetros do modelo polinomial, e a tensão terminal. Este método deve ser sempre chamado antes de se solicitar as potências de uma carga.

- **Classe *C_R_Compensator***

Esta classe modela as compensações reativas que podem ser conectados nas barras do sistema, produzidas por bancos de capacitores e/ou reatores. A classe é mostrada na Figura 5-9.

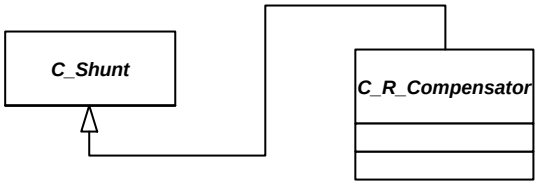


Figura 5-9 – Classe *C_R_Compensator*

O principal atributo de *C_R_Compensator* é sua potência reativa, representada pelo atributo *Q*, herdade de *C_Shunt*. Seu valor será positivo se o compensador é um banco de capacitores, ou negativo caso seja um reator. Este valor de potência reativa refere-se a potência injetada na barra na qual o compensador está conectado, quando a barra estiver operando na sua tensão nominal de 1pu.

- **Classe *C_Gen_Unit***

Uma unidade de geração, elemento tipicamente conectado em derivação no sistema, é responsável pela geração de potência na barra onde está conectada. Considera-se neste trabalho que uma unidade de geração (UG) é um objeto formado por um conjunto de outros objetos, através do mecanismo de composição da MOO. Pode ser formada por diversos elementos, tais como máquina síncrona, regulador de tensão, estabilizador de sistema de potência, regulador de velocidade, turbina e caldeira (no caso de unidades térmicas). Neste trabalho são considerados apenas os dois primeiros, sendo que a estrutura proposta permite expansões futuras para a adição de outros componentes. Uma UG é representada pela classe *C_Gen_Unit*, a qual é mostrada na Figura 5-10.

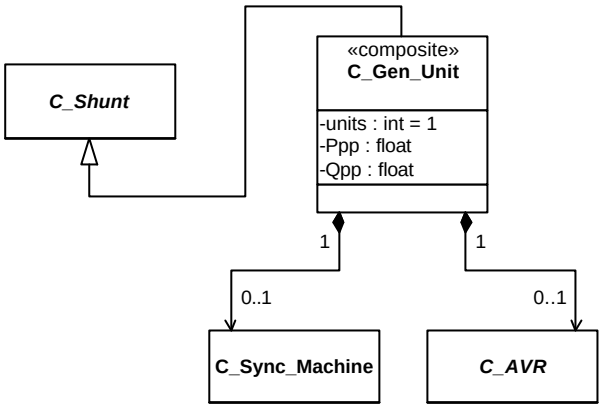


Figura 5-10 – Classe *C_Gen_Unit*

Os principais atributos de *C_Gen_Unit* são:

→ uni*t*s: número de máquinas síncronas iguais, no caso de uma unidade de geração formada por um equivalente de diversas máquinas;
→ *Ppp*: contribuição de geração de potência ativa na barra terminal (%);
→ *Qpp*: contribuição de geração de potência reativa na barra terminal (%);
→ *PtrSM*: apontador para o objeto máquina síncrona da unidade;
→ *PtrAVR*: apontador para o objeto regulador de tensão da unidade.

Um objeto tipo *C_Gen_Unit* pode representar uma máquina síncrona apenas, juntamente com seus controles, ou um equivalente de várias máquinas. Esta situação é definida pelo atributo *units*. Ainda, várias unidades de geração podem estar conectadas em uma mesma barra, atuando de forma independente. Os atributos *Ppp* e *Qpp* representam as parcelas de geração de potência ativa e reativa, respectivamente, com as quais a unidade de geração participa, em relação às potências totais geradas na barra (valores em porcentagem das potências totais). A soma dos valores de *Ppp* de todas as unidades de geração conectadas em uma mesma barra deve ser 100% (o mesmo se aplica para *Qpp*).

Observam-se na Figura 5-10 as classes que fazem parte do composto. As associações referentes à composição são representadas por apontadores na classe *C_Gen_Unit.*

- **Classe *C_SVC***

Esta classe está prevista na estrutura para a representação dos compensadores estáticos de reativos (CER). Porém a classe *C_SVC* não foi desenvolvida neste trabalho, e portanto não será aqui descrita.

5.2. Elementos de Composição

Os elementos de composição são elementos que não possuem conexões estruturais diretas a alguma barra do sistema, mas que, quando organizados de uma certa maneira, formam elementos estruturais. A estrutura de classes para sua representação é mostrada na Figura 5-11.

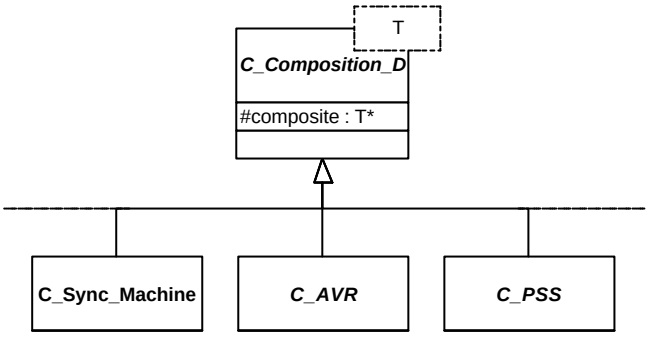


Figura 5-11 – Elementos de Composição

A estrutura é horizontal, identificando-se dois níveis principais. No primeiro nível aparece a classe abstrata *C_Composition_D*, que serve de base para todos os elementos de composição. Esta classe possui um atributo protegido, *composite*, cujo tipo é um apontador parametrizado por *T*. Utilizando-se esta construção, cada classe derivada de *C_Composition_D* "escolhe" o tipo de elemento para o qual seu apontador *composite* aponta, identificando assim o elemento estrutural que forma. Exemplificando, as classes *C_Sync_Machine* (máquina síncrona) e *C_AVR* (regulador de tensão), formadoras da classe *C_Gen_Unit* (unidades de geração), herdam a classe *C_Composition_D* fazendo o parâmetro *T* valer *C_Gen_Unit*; sendo assim, o atributo *composite* naquelas classes é do tipo "apontador para *C_Gen_Unit*", apontando para a unidade de geração que formam.

No nível seguinte encontram-se todos os elementos classificados como "de composição", derivados diretamente de *C_Composition_D*. O diagrama da Figura 5-11 mostra apenas três elementos de composição (máquina síncrona - MS, regulador de tensão - RAT e estabilizador de sistema de potência - ESP); porém a estrutura prevê sua expansão, permitindo acomodar os mais diversos elementos de composição que venham a ser modelados no futuro (turbina, regulador de velocidade, caldeira, controladores em geral, etc.), sempre derivando-os diretamente de *C_Composition_D*.

Alguns elementos de composição possuem diversos modelos internos disponíveis no mundo real, com variações no número de parâmetros, funções de transferência, etc. Nestes casos, as classes mostradas na Figura 5-11 podem ser especializadas através de he-

rança, originando uma família de diversos modelos de um determinado elemento. Não se deve confundir “diversos modelos de dispositivos reais” com diferentes níveis de detalhamento na consideração de um determinado elemento. Por exemplo, a máquina síncrona pode ser representada em simulações dinâmicas com diferentes níveis de detalhamento matemático, desde um modelo simplificado, no qual é representada por uma fonte de tensão constante e sua reatância transitória de eixo direto (conectados em série), até modelos detalhados, onde se consideram inclusive efeitos subtransitórios no estator da máquina. Esta distinção entre diferentes comportamentos de um mesmo elemento é considerada na definição das interfaces funcionais dos elementos físicos, uma vez que diz respeito às metodologias de análise e síntese, e não especificamente ao dispositivo real (as interfaces funcionais são tratadas no Capítulo 6, seção 6.2).

Neste trabalho, dois elementos de composição são modelados: máquina síncrona e regulador de tensão. As classes para estes elementos são descritas a seguir.

- **Classe *C_Sync_Machine***

Esta classe representa as máquinas síncronas do sistema elétrico. Uma máquina síncrona, quando agrupada com seus equipamentos de controle (tensão, velocidade, etc.), origina um elemento estrutural unidade de geração, responsável pela geração de potência nas barras. A classe é apresentada em detalhes na Figura 5-12.

Os principais atributos e métodos da classe são:

→ *units*: número de máquinas síncronas iguais, no caso de uma unidade de geração formada por um equivalente de diversas máquinas;
→ *model*: modelo a ser utilizado para a representação da máquina em estudos dinâmicos;
→ *delta, wr, Pm, Pe, Vfd, Ifd*: grandezas internas da máquina (posição e velocidade angulares, potências mecânica e elétrica, tensão e corrente de campo, respectivamente);
→ *freq, wo, ra, H, D, MVA, xd, xq, xld, xlq, xlld, xllq, xl, Tldo, Tlqo, Tlldo, Tllqo*: parâmetros standard da máquina síncrona;
→ *Aggregate()*: calcula os parâmetros para uma máquina síncrona equivalente.

A classe *C_Sync_Machine* representa qualquer máquina síncrona no sistema, independente do modelo a ser utilizado para representar o seu comportamento dinâmico. Diferenciações entre estes comportamentos devem ser modeladas nas interfaces funcionais da máquina síncrona, dependentes de cada aplicação. Mais detalhes são mostrados no capítulo seguinte (seção 6.4.4).

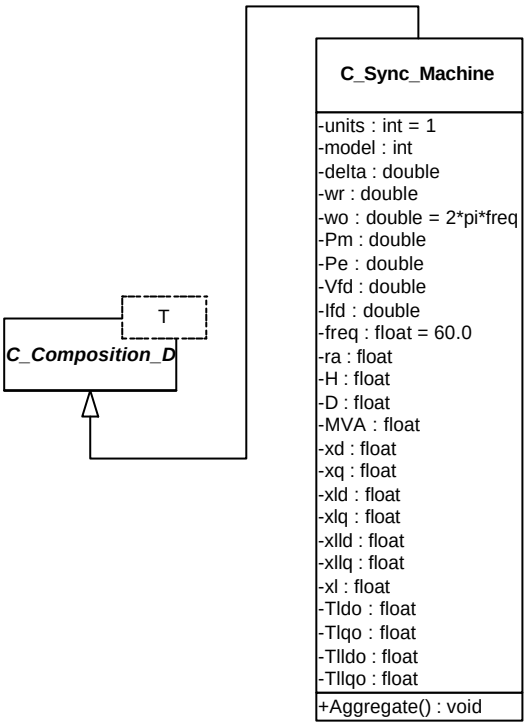


Figura 5-12 – Classe *C_Sync_Machine*

- **Classe *C_AVR***

Diferentemente das máquinas síncronas, os sistemas de excitação (formados pelo conjunto regulador de tensão + excitatriz) possuem diferentes modelos internos, com parâmetros e características de funcionamento diferenciados. Para sua representação, criou-se uma classe abstrata *C_AVR*, que serve como base para classes específicas, representantes de cada modelo interno de sistema de excitação. A classe *C_*AVR é apresentada na Figura 5-13.

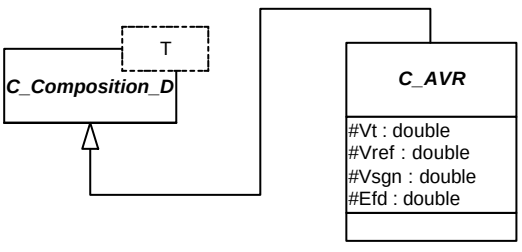


Figura 5-13 – Classe *C_AVR*

Os principais atributos da classe, que representam características comuns a todos os sistemas de excitação, são:

→ *Vt*: tensão terminal da máquina síncrona sob controle;
→ *Vref*: referência de tensão;
→ *Vsgn*: sinal adicional (sinal de saída de um estabilizador de sistema de potência - ESP);
→ *Efd*: excitação (tensão de campo) a ser aplicada na máquina síncrona.

- **Classe *C_AVR_I***

Esta classe representa sistemas de excitação (regulador de tensão e excitatriz) conforme o modelo IEEE ST-1. A classe é mostrada na Figura 5-14.

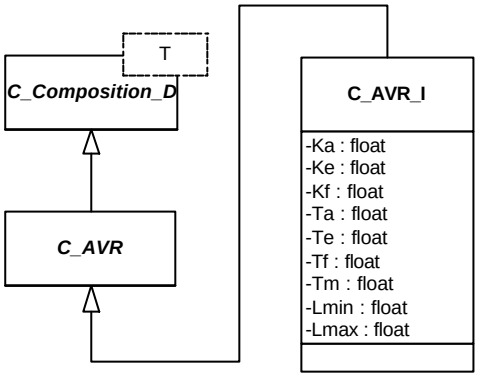


Figura 5-14 – Classe *C_AVR_I*

Seus atributos representam os parâmetros do modelo, e podem ser observados diretamente na figura acima.

- **Classe *C_AVR_KaTa***

Esta classe representa sistemas de excitação (regulador de tensão e excitatriz) conforme um modelo simplificado *ganho + constante de tempo*. A classe é mostrada na Figura 5-15.

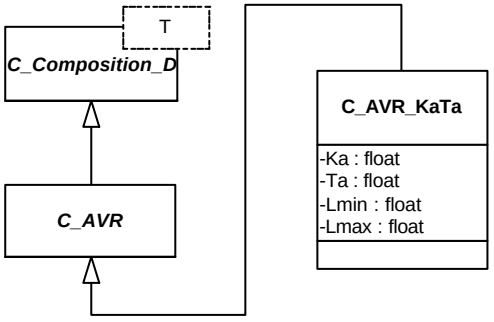


Figura 5-15 – Classe *C_AVR_KaTa*

Seus atributos representam os parâmetros do modelo, e podem ser observados diretamente na figura acima.

5.2.1. Comunicação entre Elementos de um Composto

A comunicação entre os objetos formadores de um composto pode ser vista basicamente de duas formas: comunicação direta entre os objetos, através de associações implementadas entre suas classes; ou comunicação indireta, gerenciada pelo objeto composto.

Na primeira maneira faz-se com que o objeto composto gerencie todo o processo de troca de informações entre seus componentes. Neste caso os componentes não possuem ligações entre si, apenas ligações com o objeto que compõem. Este objeto de mais alto nível conhece informações sobre suas conexões, tanto externas (barras, por exemplo), quanto internas (quais elementos o compõe, e como se comunicam). Compostos assim formados são mais facilmente adaptados a novas situações, uma vez que uma classe apenas (o composto) concentra informações sobre conexões e troca de informações. A desvantagem é que o processo de funcionamento do composto como um todo passa a ser mais "burocrático", uma vez que toda troca de informações necessariamente passa pelo objeto composto.

No segundo caso criam-se associações entre as classes dos objetos formadores, de forma que esses tenham acesso direto uns aos outros, sem necessariamente terem que se dirigir ao objeto composto. Com isso a troca de informações e mensagens entre os objetos do composto, executando determinados comportamentos do objeto composto, torna-se mais ágil, e visualmente mais perceptível nos diagramas de classes do projeto. Porém esta maneira apresenta a desvantagem de que os objetos de composição passam a ser mais específicos, por possuírem certas ligações que o caracterizam como formador de um determinado composto específico, e sendo, portanto, difícil utilizar um mesmo tipo de objeto em diferentes compostos. Além disso, torna difícil adaptar uma certa estrutura já existente a outra situação de utilização, uma vez que se delegam informações sobre a conectividade interna de um composto para seus elementos formadores.

Neste trabalho, a estrutura de classes dos elementos físicos dos SEE utiliza a primeira maneira. Dessa forma as classes que representam os elementos físicos estruturais caracterizados como compostos tornam-se mais genéricas, centralizando-se as atividades relativas às trocas de mensagens entre elementos formadores, no próprio objeto composto.

5.3. Representação do Sistema de Energia Elétrica

O SEE em si é representado por uma classe chamada *C_Power_System*, mostrada na Figura 5-16.

Os principais atributos e métodos da classe são:

→ *title*: título (nome) de identificação do sistema elétrico sob estudo;
→ *Sbase*: base de potência do sistema (MVA);
→ *IndxBranch*: lista de elementos série do sistema;
→ *IndxShunt*: lista de elementos em derivação no sistema;
→ *IndxBar*: lista de barras do sistema;
→ *Indx...*: listas de elementos diversos.

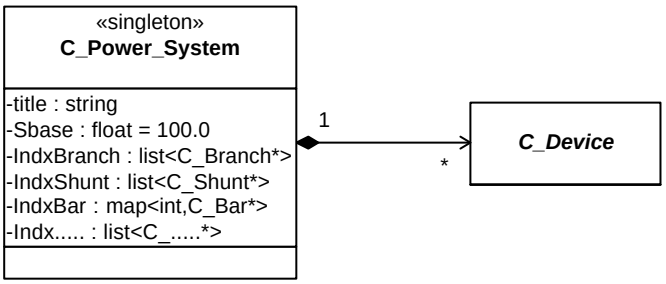


Figura 5-16 – Classe *C_Power_System*

O objeto SEE, do tipo *C_Power_System*, contém por composição todos os elementos físicos do sistema sob análise. Os elementos são armazenados sob a forma de listas, representadas na classe por atributos do tipo *IndxXXX*, onde *XXX* identifica cada tipo de elemento. As três listas principais armazenam as barras do sistema (*IndxBar*), os elementos conectados em série (*IndxBranch*) e em derivação (*IndxShunt*). Todos os elementos físicos (estruturais ou de composição) devem ser armazenados no objeto SEE através de listas; assim, tem-se também listas de linhas de transmissão (*IndxTL*), transformadores (*IndxTrafo*), compensadores reativos (*IndxRC*), cargas (*IndxLoad*), unidades de geração (*IndxGen_Unit*), máquinas síncronas (*Indx_Sync_Machine*), reguladores de tensão (*IndxAVR*), etc. Estas listas estão representadas de forma genérica na Figura 5-16 pelo atributo simbólico *Indx.....* . Qualquer novo tipo de elemento modelado futuramente na estrutura deve ser acompanhado de uma nova lista específica na classe *C_Power_System*.

As listas armazenam *apontadores* para cada elemento (ponteiros, no caso da linguagem C++), e não diretamente os elementos. Portanto, alguns elementos podem ser (e efetivamente o são) armazenados em mais de uma lista; um transformador, por exemplo,

terá um apontador tanto na lista de elementos série (*IndxBranch*) quanto na de transformadores (*IndxTrafo*). Isto é feito para que seja possível acessar os elementos de um SEE de diferentes formas, seja percorrendo diretamente uma lista específica de um determinado tipo de elemento (*IndxTrafo* para transformadores, por exemplo), seja percorrendo uma lista mais genérica, para acessar todo um conjunto de elementos com certas características comuns (*IndxBranch* para acessar todos os elemento série de um sistema, por exemplo – acessar-se-á, através desta lista, tanto transformadores, como linhas de transmissão, TCSCs, ou qualquer outro elemento série).

As listas utilizadas na implementação deste trabalho são do tipo *list*<> (lista simples) e *map*<> (lista indexada), que são *containers* disponíveis na Standard Template Library (STL) da linguagem de programação C++.

A classe *C_Power_System* ainda contém uma série de métodos de interface do tipo *Get_...()* e *Set_...()*, os quais fornecem acesso aos atributos e listas de elementos do objeto.

Utilizou-se no projeto desta classe o padrão Singleton (GAMMA et al., 2000), pois apenas um objeto SEE deve existir em tempo de execução de uma determinada ferramenta computacional. Conforme abordado adiante, no Capítulo 7, esta restrição é considerada neste trabalho apenas por questão de simplicidade, não caracterizando uma restrição definitiva ao desenvolvimento de futuras ferramentas computacionais para o apoio ao planejamento e análise de sistemas elétricos.

CAPÍTULO 6

6. MODELAGEM 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 real (física). Este fato permite que sua representação seja mais flexível, pois não existe uma estrutura de conexão bem definida entre as metodologias delineando as conexões entre suas classes representativas. Além disso, pequenas variações na forma de se classificar as metodologias segundo uma hierarquia de classes trarão pouco impacto no projeto das ferramentas computacionais.

A exemplo dos elementos físicos, algumas formas para se representar metodologias de análise e síntese já foram propostas. Entretanto, na maioria dos trabalhos publicados na área encontram-se poucos detalhes que permitam esclarecer a estrutura de classificação das metodologias. Até pelo fato de essa maioria de trabalhos apresentarem focos específicos em determinadas metodologias, desenvolvendo estruturas de classes voltadas para uma metodologia em particular (ou um grupo restrito de metodologias). Ainda assim é possível identificar desde estruturas completamente independentes dos elementos físicos para acomodar metodologias (HAKAVIK et al., 1994, HANDSCHIN et al., 1998), até conjuntos de classes para metodologias derivando de classes de elementos físicos (FUERTE-ESQUIVEL et al., 1998), principalmente em estruturas que possuem classes do tipo *Rede Elétrica* (PANDIT et. al., 2000).

Neste trabalho, as metodologias de análise e síntese são representadas em uma estrutura hierárquica de classes independente das estruturas que representam os elementos físicos do SEE. As metodologias, também chamadas no presente trabalho de aplica-

ções, possuem uma ligação ao objeto SEE (tipo *C_Power_System*), sobre o qual atuam. De uma forma geral, cada aplicação solicita uma série de informações ao SEE e seus elementos formadores, para assim montar estruturas matemáticas (matrizes, sistemas lineares, conjuntos de equações algébricas e diferenciais, problemas de otimização, etc.). Essas estruturas matemáticas são então resolvidas, obtendo-se a solução do problema em questão. E caso seja pertinente, grandezas referentes ao estado do SEE podem ser atribuídas novamente ao objeto SEE, e aos objetos que o compõe.

Na Figura 6-1 pode ser observada a estrutura proposta para a representação das aplicações. 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*.

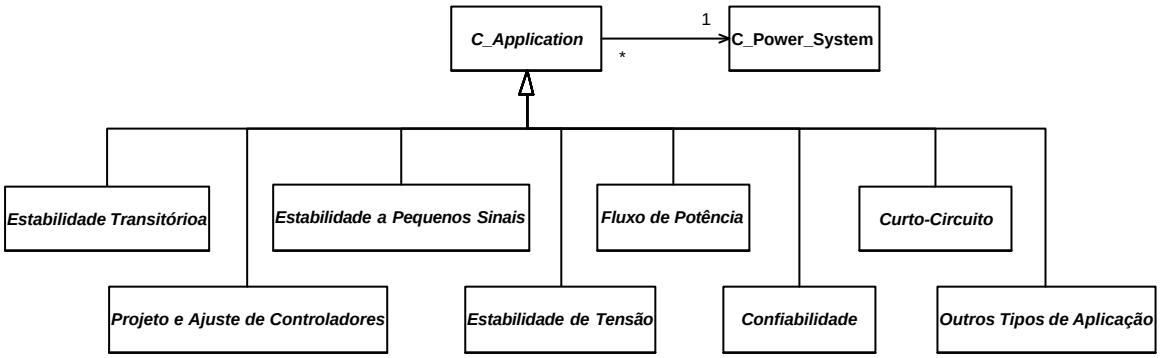


Figura 6-1 – Diagrama de Classes para as Aplicações

Cada conjunto de aplicações representado na estrutura acima pode ser detalhado, de acordo com as diversas metodologias que o integram. Na Figura 6-2, por exemplo, apresentam-se as classes que representam as diversas metodologias para cálculo de fluxo de potência, tais como fluxo de potência linearizado (*C_Flow_DC*), via método de Newton-Raphson (*C_Flow_NR*), desacoplado rápido (*C_Flow_FD*), e fluxo de potência ótimo (*C_Flow_OPF*). Como outro exemplo, a Figura 6-3 apresenta classes para representar duas diferentes metodologias de análise da estabilidade transitória: "simulação dinâmica com modelagem detalhada" (*C_SIMSP*), e "avaliação global da estabilidade transitória via método da superfície limite de energia potencial" (*C_SLEP*).

O padrão de projeto *Composite* é utilizado na modelagem das aplicações, facilitando o reaproveitamento de metodologias já implementadas. Dessa forma, aplicações existentes podem ser agrupadas originando uma nova aplicação. Exemplificando o uso do padrão, a Figura 6-4 apresenta a aplicação "Avaliação e Melhoria da Segurança Dinâmica" (*C_AMSD*) como sendo um *composite*, formado pelas aplicações "fluxo de potência via mé-

todo de Newton-Raphson" (*C_Flow_NR*), "simulação dinâmica com modelagem detalhada" (*C_SIMSP*), e "avaliação global da estabilidade transitória via método da superfície limite de energia potencial" (*C_SLEP*).

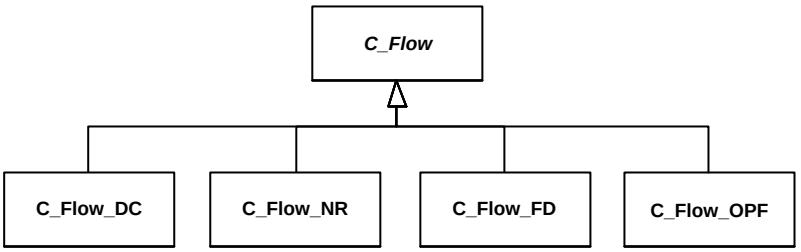


Figura 6-2 – Metodologias de Fluxo de Potência

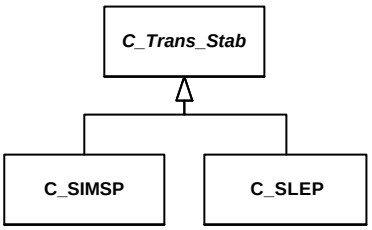


Figura 6-3 – Metodologias de Análise da Estabilidade Transitória

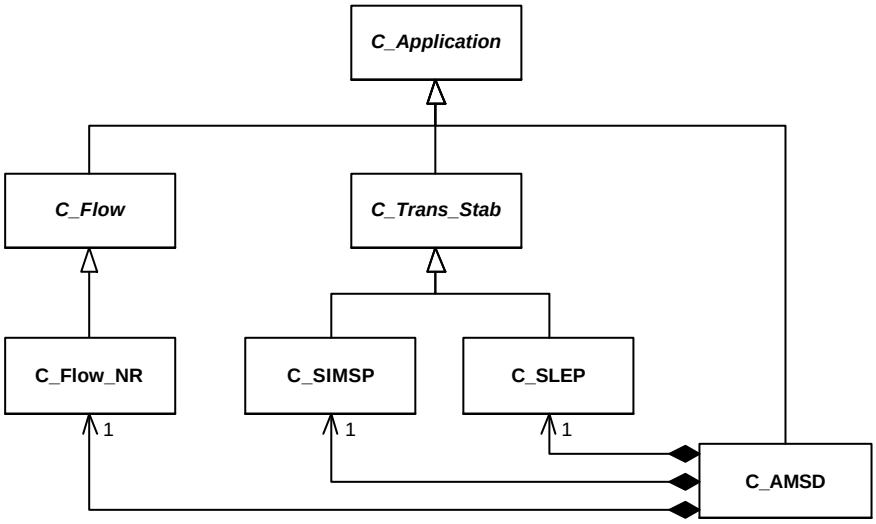


Figura 6-4 – Avaliação e Melhoria da Segurança Dinâmica como um *Composite*

Nesta aplicação, as metodologias de análise automática das curvas de simulação e cálculo de margens de estabilidade, assim como a melhoria da segurança, representada por ações de controle a serem tomadas sobre o sistema, podem ser modeladas como funcionalidades na classe *C_AMSD*, ou como novas aplicações independentes. A escolha por

uma forma ou outra depende do escopo que se deseja dar ao projeto da aplicação. Em um primeiro momento essas metodologias podem ser modeladas como métodos internos da classe *C_AMSD*, sendo posteriormente modeladas como classes independentes, inclusive por outra equipe de projeto. A aplicação de Avaliação e Melhoria da Segurança Dinâmica está sendo modelada em um trabalho de dissertação de mestrado, no âmbito do tema geral enfocado no presente trabalho de tese.

Algumas das metodologias listadas acima são aqui projetadas e implementadas. A seguir descrevem-se as classes representantes destas metodologias.

6.1. Comportamento Geral de uma Aplicação – Classe *C_Application*

A classe *C_Application* serve de base para todas as metodologias de análise e síntese que são aplicadas a um SEE. Ela implementa características comuns a todas as metodologias. A classe é mostrada na Figura 6-5.

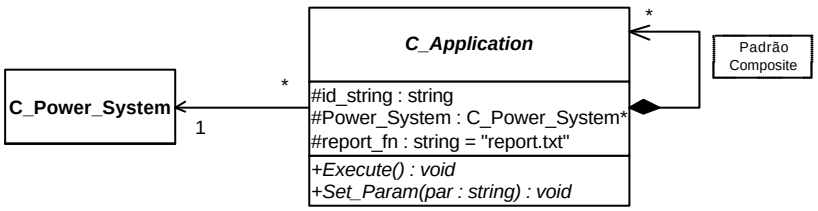


Figura 6-5 – Classe *C_Application*

Os principais atributos e métodos de *C_Application* são:

→ *id_string*: identificador do tipo de aplicação;
→ *Power_System*: apontador para o SEE, sobre o qual a aplicação será executada;
→ *report_fn*: nome do arquivo onde a aplicação escreve seu relatório padrão;
→ *Execute()*: método virtual, responsável pela execução da aplicação;
→ *Set_Param()*: método virtual, para a atribuição dos parâmetros de aplicação.

Uma ferramenta computacional implementada sob a filosofia apresentada neste trabalho pode conter diversas aplicações, operando em um ambiente integrado. Para que seja feita a identificação e a diferenciação entre diversas aplicações em uma mesma ferramenta, cada aplicação tem um identificador, do tipo *string* de caracteres, armazenado no atributo *id_string*. Este identificador é utilizado para localizar também as interfaces funcionais pertencentes a cada aplicação em um determinado elemento físico. Ele é atri-

buído no momento da criação da aplicação, contendo o nome da classe, sem o prefixo "C_".

Todas as aplicações geram um relatório padrão, relatando informações a respeito da sua execução. O nome deste arquivo é armazenado em *report_fn*. Caso uma aplicação em específico necessite escrever mais de um relatório, atributos devem ser especificados na própria classe, identificando os diversos outros relatórios.

A classe declara os métodos virtuais *Set_Param()* e *Execute()*, os quais são herdados publicamente e definidos por todas as aplicações. O primeiro método é o responsável pela atribuição de parâmetros ao objeto de aplicação; o segundo, pela execução da aplicação. Estes métodos são acionados pelo objeto da ferramenta computacional, mediante a recepção de comandos (o funcionamento das ferramentas é descrito no Capítulo 7).

O método *Execute()* deve coordenar o processo completo de execução da aplicação. O controle de como a aplicação será executada é feito através dos seus parâmetros, que devem ser completamente configurados antes da execução. Cada aplicação contém parâmetros previamente inicializados (normalmente no método construtor da classe), que podem ser alterados de acordo com a situação em que a análise será feita.

6.2. Interfaces Funcionais

Conforme apresentado no Capítulo 4, os diferentes comportamentos dos elementos físicos dos SEE, para cada metodologia de análise e síntese, são encapsulados em classes específicas, denominadas neste trabalho *interfaces funcionais*. Cada aplicação contém um conjunto de interfaces funcionais, que implementam comportamentos dos elementos físicos relativos especificamente à aplicação em questão. As interfaces funcionais são implementadas juntamente com a aplicação, definindo-se comportamentos apenas para elementos físicos considerados ativos em cada aplicação.

A comunicação entre uma interface funcional e o elemento físico ao qual ela se refere é modelada utilizando-se o padrão de projeto orientado a objeto Adapter. Dessa forma torna-se eficiente a expansão das estruturas de classes, modelando-se mais facilmente novas aplicações, e conseqüentemente novas interfaces funcionais. Esta construção possibilita ainda a troca dinâmica de comportamentos dos elementos físicos, ou seja, diversas aplicações podem ser executadas no âmbito de uma mesma ferramenta computacional, trocando-se as funcionalidades dos elementos em tempo de execução.

Na Figura 6-6 é apresentado o diagrama de classes para a representação das interfaces funcionais. Observa-se na figura que as associações entre as classes seguem a mesma estrutura que representa os elementos físicos do sistema. Isto caracteriza uma redundância de informações, pois estas associações podem ser obtidas através dos objetos dos elementos físicos propriamente ditos. Esta redundância aumenta a eficiência da base computacional desenvolvida. Permite-se, assim, que se percorra a estrutura física da rede elétrica diretamente pelas interfaces funcionais, sem a necessidade de se ter que recorrer aos objetos físicos. Esta redundância não caracteriza uma quebra de conceito, pois as interfaces funcionais nada mais são do que extensões dos elementos físicos, fazendo parte deles. Assim, nada impede que elas tenham conhecimento local das conexões estruturais que os elementos físicos possuem.

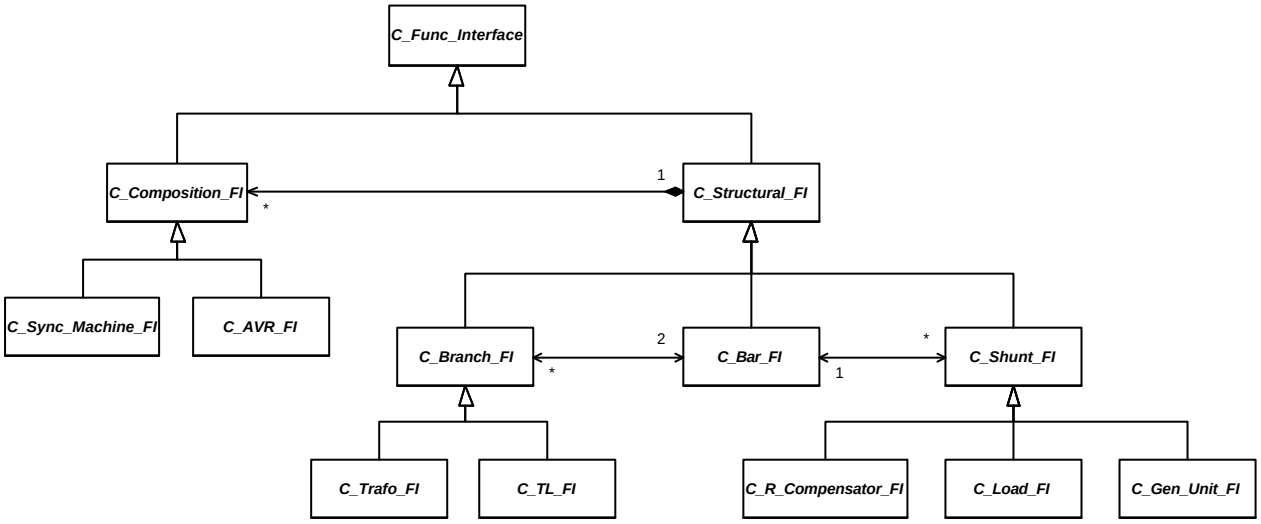


Figura 6-6 – Diagrama de Classes para as Interfaces Funcionais

As classes na estrutura da Figura 6-6 são todas abstratas, caracterizando que não devem originar objetos diretamente. Estas classes servem de base para a implementação de classes concretas, modelando comportamentos específicos de cada elemento físico, em cada aplicação específica.

Uma vez que as interfaces funcionais pertencem diretamente a uma aplicação, o processo para sua criação normalmente é feito pelo método construtor da aplicação, percorrendo as listas de elementos físicos do objeto *Power System*, e criando interfaces para aqueles elementos que possuem funcionalidades na aplicação corrente. As conexões das interfaces aos seus elementos físicos são realizadas internamente pelo próprio método construtor de cada interface. Para isso, o endereço do elemento físico ao qual a interface que esta sendo criada refere-se, deve ser passado como parâmetro no momento da criação desta interface. As conexões estruturais entre interfaces funcionais são realizadas

por métodos tipo *Connect()*, implementados nas classes abstratas; estes métodos devem ser acionados pelas aplicações, logo após a criação de cada interface. As interfaces são destruídas juntamente com a aplicação a qual pertencem.

A seguir são descritas as classes principais da estrutura da Figura 6-6.

- **Classe *C_Func_Interface***

A classe *C_Func_Interface* serve como base para todas as interfaces funcionais na base computacional desenvolvida. Ela é apresentada em detalhes na Figura 6-7, tendo seus principais atributos descritos a seguir.

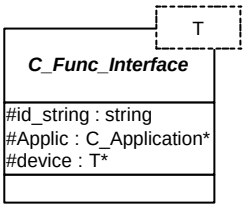


Figura 6-7 – Classe *C_Func_Interface*

→ *id_string*: nome que identifica o tipo de elemento físico ao qual a interface refere-se;
→ *Applic*: apontador para a aplicação à qual a interface pertence;
→ *device*: apontador para o elemento físico ao qual a interface refere-se.

Esta classe é parametrizada pelo parâmetro T, que modela o tipo do apontador *device* da classe. O parâmetro é definido nas classes bases das interfaces funcionais de cada elemento físico. Por exemplo, a classe *C_Load_FI*, que serve como base para todas as interfaces funcionais de cargas, herda a classe *C_Func_Interface* atribuindo a classe *C_Load* ao parâmetro *T*. Isto faz com que o atributo *device* na classe *C_Load_FI* (herdado de *C_Func_Interface*), e por conseqüência em todas as classes derivadas de *C_Load_FI*, aponte para objetos tipo *C_Load*. Todas as classes intermediárias na estrutura hierárquica, desde *C_Func_Interface* até as interfaces específicas para cada elemento físico, são também parametrizadas, para que seja possível repassar o parâmetro *T* estrutura acima, até a classe *C_Func_Interface*.

Além do apontador para seu elemento físico (atributo *device*), parametrizado por *T*, a classe possui um atributo tipo *string* de caracteres, chamado *id_string*, que armazena o nome do tipo de elemento físico do qual a interface implementa funcionalidades. Este atributo é formado pelo nome da classe, menos o prefixo "*C_*", adicionado do sufixo "*_FI*". O atributo é imputado pelos construtores das classes bases das interfaces de cada tipo

de elemento. Por exemplo, todas as interfaces funcionais de barras possuem o atributo *id_string* contendo “Bar_FI”, que é imputado pelo construtor da classe *C_Bar_FI.*

- **Classe *C_Composition_FI***

A classe *C_Composition_FI* é base para todas as interfaces funcionais de elementos físicos de composição. A classe é mostrada na Figura 6-8.

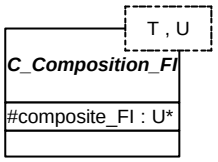


Figura 6-8 – Classe *C_Compositon_FI*

Seu principal atributo é a variável parametrizada *composite_FI*, que aponta para objetos tipo *U*. Este parâmetro representa o tipo de elemento estrutural que cada elemento de composição forma. As classes bases das interfaces funcionais de cada elemento de composição herdam a classe *C_Composition_FI*, atribuindo ao parâmetro *U* a classe correspondente ao composto formado. Isto faz com que o apontador *composite_FI*, pertencente a classe *C_Composition_FI*, e herdado por cada interface funcional de elementos de composição, aponte para o elemento estrutural correto. Por exemplo, no caso da classe - *C_AVR_FI*, ela herda a classe *C_Composition_FI* atribuindo a classe *C_Gen_Unit_FI* ao parâmetro *U*. Dessa forma, todas as interfaces funcionais de sistemas de excitação (que derivam necessariamente de *C_AVR_FI*) possuem o atributo *composite_FI* apontando para interfaces funcionais de unidades de geração (*C_Gen_Unit_FI*), pois sistemas de excitação são partes formadoras de unidades de geração. Ainda, atribui *C_AVR* ao parâmetro *T*, de forma que seu atributo *device* (herdado de *C_Func_Interface*) aponta para elementos físicos sistemas de excitação.

- **Classe *C_Bar_FI***

Esta classe, mostrada na Figura 6-9, é base para todas as interfaces funcionais de barras. Ela possui dois atributos principais (*IndxBranch_FI* e *IndxShunt_FI*), os quais armazenam em listas, apontadores para as interfaces funcionais de elementos série e em derivação (respectivamente) conectados à barra relativa à interface.

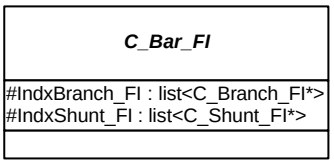


Figura 6-9 – Classe *C_Bar_FI*

- **Classe *C_Branch_FI***

A classe *C_Branch_FI*, mostrada na Figura 6-10, generaliza as interfaces funcionais de todos os elementos conectados em série no sistema elétrico. Ela possui dois atributos principais (*PtrBar1_FI* e *PtrBar2_FI*), que apontam para as interfaces das barras de conexão do elemento série do qual implementa funcionalidades.

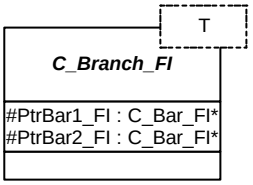


Figura 6-10 – Classe *C_Branch_FI*

- **Classe *C_Shunt_FI***

Semelhante a classe anterior, *C_Shunt_FI* generaliza as interfaces funcionais dos elementos conectados em derivação, e seu atributo *PtrBar_FI* aponta para a interface da barra de conexão do respectivo elemento em derivação.

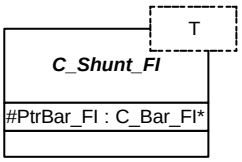


Figura 6-11 – Classe *C_Shunt_FI*

- **Classe *C_Gen_Unit_FI***

C_Gen_Unit_FI serve de classe base para as interfaces funcionais das unidades de geração do sistema. Mostrada na Figura 6-12, ela possui atributos que são apontadores para as interfaces funcionais dos elementos de composição que formam a unidade de geração relativa à interface.

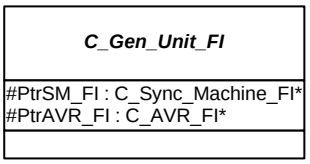


Figura 6-12 – Classe *C_Gen_Unit_FI*

6.3. Aplicação 1: Fluxo de Potência Linearizado – Classe *C_Flow_DC*

Uma aplicação de fluxo de potência linearizado foi implementada neste trabalho, representada pela classe *C_Flow_DC*. A classe é mostrada na Figura 6-13.

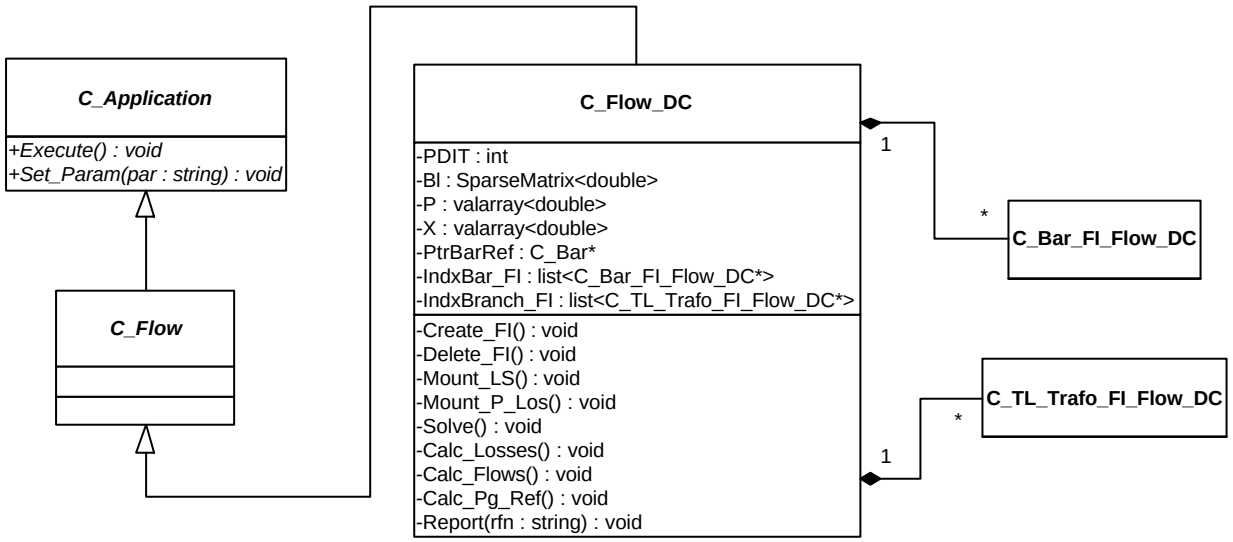


Figura 6-13 – Classe *C_Flow_DC*

Os principais atributos e métodos de *C_Flow_DC* são:

→ *PDIT*: número de iterações para estimação de perdas;

→ *Bl*: matriz B';

→ *P*: vetor de potências de barra – vetor independente do sistema linear a ser solucionado;

→ *X*: vetor de ângulos de tensão nas barras – vetor solução do sistema linear a ser solucionado;

→ *PtrBarRef*: ponteiro para a barra de referência do sistema;

→ *IndxBar_FI*: lista de interfaces funcionais de barras;

→ *IndxBranch_FI*: lista de interfaces funcionais de elementos série (linhas de transmissão e transformadores);

→ *Create_FI()*: cria as interfaces funcionais dos elementos físicos;
→ *Delete_FI()*: elimina as interfaces funcionais;
→ *Mount_LS()*: calcula a matriz B' e o vetor de potências de barras;
→ *Mount_P_Los()*: calcula o vetor de potências de barras corrigido com o valor estimado das perdas nos elementos série;
→ *Solve()*: solicita a solução do sistema linear;
→ *Calc_Losses()*: estima as perdas ativas nos elementos série;
→ *Calc_Flow()*: calcula os fluxos de potência ativa nos elementos série;
→ *Calc_Pg_Ref()*: calcula a potência ativa gerada na barra de referência;
→ *Report()*: imprime o relatório de resultados.

A aplicação soluciona o sistema linear $\vec{P} = B'.\vec{?}$, onde *B'* é uma matriz tipo admitância nodal, cujos elementos são formados por:

$$B'_{km} = -x_{km}^{-1} \qquad B'_{kk} = \sum_{m \in \Omega_k} x_{km}^{-1}$$

sendo x_{km} as reatâncias dos elementos série (linhas de transmissão e transformadores) do sistema, e Ω_k o conjunto de barras que compartilham elementos série conectados com a barra *k*. O vetor $\vec{P}$ contém as injeções líquidas de potência ativa nas barras, e o vetor $\vec{?}$ armazena, após a solução do sistema linear, os ângulos aproximados das tensões nas barras; o módulo da tensão em cada barra é considerado igual a 1pu, uma vez que os fluxos de potência reativa são desprezados. A dimensão do sistema linear é dada pelo número de barras do SEE, menos uma, que é a barra de referência do sistema ($\theta = 0.0$). O modelo projetado e implementado neste trabalho possui um parâmetro chamado *PDIT*, o qual armazena o número de iterações para a estimação das perdas ativas nos elementos série do sistema (linhas de transmissão e transformadores). Mais detalhes sobre a formulação matemática do modelo de fluxo de potência linearizado podem ser encontrados em MONTICELLI (1983).

Ilustrando a seqüência de atividades que um objeto do tipo *C_Flow_DC* realiza, é montado um Diagrama de Atividades para o cálculo de um fluxo de potência linearizado, mostrado na Figura 6-14. Uma vez acionado o método *Execute()*, o objeto do tipo *C_Flow_DC* verifica em primeiro lugar se existe um SEE para que a análise possa ser executada. Caso exista, é montado e solucionado o sistema linear $\vec{P} = B'.\vec{?}$, obtendo-se os ângulos das tensões nas barras. Caso seja solicitada a estimação de perdas ativas nos elementos série, *PDIT* iterações são realizadas, calculando as novas injeções de potências ativas nas barras. Considera-se as perdas estimadas como sendo cargas adicionais nas barras de conexão dos elementos série (MONTICELLI, 1983). Posteriormente são calcula-

dos os fluxos de potência ativa nos elementos série, calcula-se a potência ativa gerada na barra de referência, imprime-se o relatório de resultados, e a execução é encerrada. A criação das interfaces funcionais é realizada no construtor da classe, antes mesmo do acionamento do método *Execute()*.

O relatório fornecido pela aplicação consiste em uma listagem das barras do SEE, apresentando os seus ângulos de tensão, em graus elétricos, em relação à barra escolhida como referência (com módulos de tensão unitários, em pu), e uma listagem dos elementos série (transformadores e linhas de transmissão), apresentando seus fluxos de potência ativa e estimação de perdas.

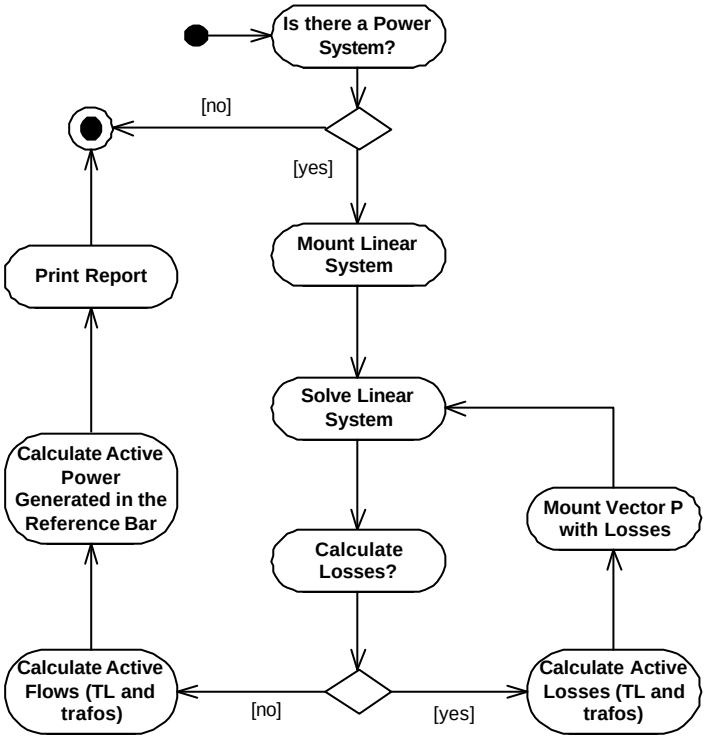


Figura 6-14 – Diagrama de Atividades para a Execução de um Fluxo de Potência Linearizado

6.3.1. Representação dos Elementos Físicos – Interfaces Funcionais para o Fluxo de Potência Linearizado

A definição das interfaces funcionais de elementos físicos para uma determinada aplicação depende basicamente de se identificar quais destes elementos são considerados para o cálculo da aplicação. Assim, dois tipos de interfaces funcionais são definidos para compor um fluxo de potência linearizado: interfaces funcionais para barras (classe

C_Bar_FI_Flow_DC) e para elementos série (classe *C_TL_Trafo_FI_Flow_DC*). Estas classes são apresentadas a seguir.

- **Classe *C_Bar_FI_Flow_DC***

A classe *C_Bar_FI_Flow_DC* implementa o comportamento das barras do SEE, mediante o cálculo de um fluxo de potência linearizado. A classe é mostrada na Figura 6-15.

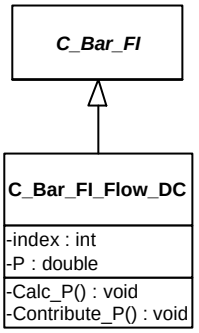


Figura 6-15 – Classe *C_Bar_FI_Flow_DC*

Os principais atributos e métodos da classe são:

→ *index*: índice de localização da barra na matriz de admitâncias B';
→ *P*: injeção líquida de potência ativa na barra;
→ *Calc_P()*: calcula a injeção líquida de potência ativa na barra;
→ *Contribute_P()*: insere a injeção líquida de potência ativa da barra no vetor independente do sistema linear a ser resolvido.

Uma barra, considerando-se um fluxo de potência linearizado, deve "saber" realizar duas tarefas básicas: calcular a sua injeção líquida de potência ativa, e contribuir no cálculo do vetor de injeções de potência ativa, necessário à solução do sistema linear $\vec{P} = B'.\vec{?}$. Para calcular a injeção de potência, a barra percorre os elementos em derivação a ela conectados, acumulando as injeções individuais de potência ativa de cada elemento. Se for uma barra de referência, percorre todas as outras barras do sistema e acumula o balanço de potência, considerando sua carga local e a estimação das perdas ativas nos elementos série. Caso exista mais de uma unidade de geração conectadas na barra de referência, esta distribui igualmente a geração de potência ativa entre as diferentes unidades.

- **Classe *C_TL_Trafo_FI_Flow_DC***

O modelo de fluxo de potência linearizado implementado considera a existência de dois tipos de elementos série no sistema: linhas de transmissão (LT) e transformadores (trafo). O comportamento destes dois tipos de elementos é exatamente o mesmo na aplicação, que desconsidera as derivações (*taps*) e as defasagens angulares em transformadores com estes recursos. Assim, uma classe única é criada para a modelagem do comportamento destes elementos no fluxo de potência linearizado, chamada *C_TL_Trafo_FI_Flow_DC*. A classe é mostrada na Figura 6-16.

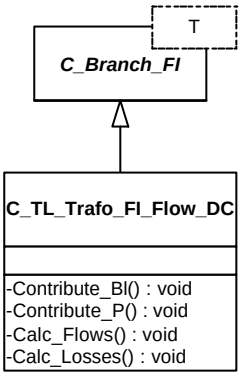


Figura 6-16 – Classe *C_TL_Trafo_FI_Flow_DC*

Os principais métodos da classe são:

→ *Contribute_Bl()*: insere a contribuição do elemento série no cálculo da matriz de admitâncias B';

→ *Contribute_P()*: insere a contribuição do elemento série (sua perda estimada) no cálculo do vetor independente do sistema linear a ser resolvido;

→ *Calc_Losses()*: estima a perda ativa no elemento;

→ *Calc_Flows()*: calcula o fluxo de potência ativa no elemento.

Observa-se na Figura 6-16 que a classe *C_TL_Trafo_FI_Flow_DC* deriva diretamente de *C_Branch_FI*. A herança é implementada definindo-se o parâmetro *T* como sendo do tipo *C_Branch*, o que torna o atributo *device* de *C_TL_Trafo_FI_Flow_DC* (herdado de *C_Func_Interface*) um apontador para qualquer tipo de elemento série. Assim, a interface pode definir funcionalidades tanto para linhas de transmissão (LT) quanto para transformadores (trafos).

Duas funcionalidades são implementadas para elementos série (trafo ou LT), em um fluxo de potência linearizado: estimação das perdas ativas do elemento e cálculo do seu fluxo de potência ativa. A formulação matemática destes procedimentos pode ser encontrada em MONTICELLI (1983).

6.4. Aplicação 2: Simulação da Dinâmica com Modelagem Detalhada – Classe *C_SIMSP*

Foi projetada também neste trabalho uma aplicação para a Simulação da Dinâmica de SEE, utilizando modelagem detalhada para os elementos do sistema elétrico. A aplicação é representada pela classe *C_SIMSP*, mostrada em detalhes na Figura 6-17.

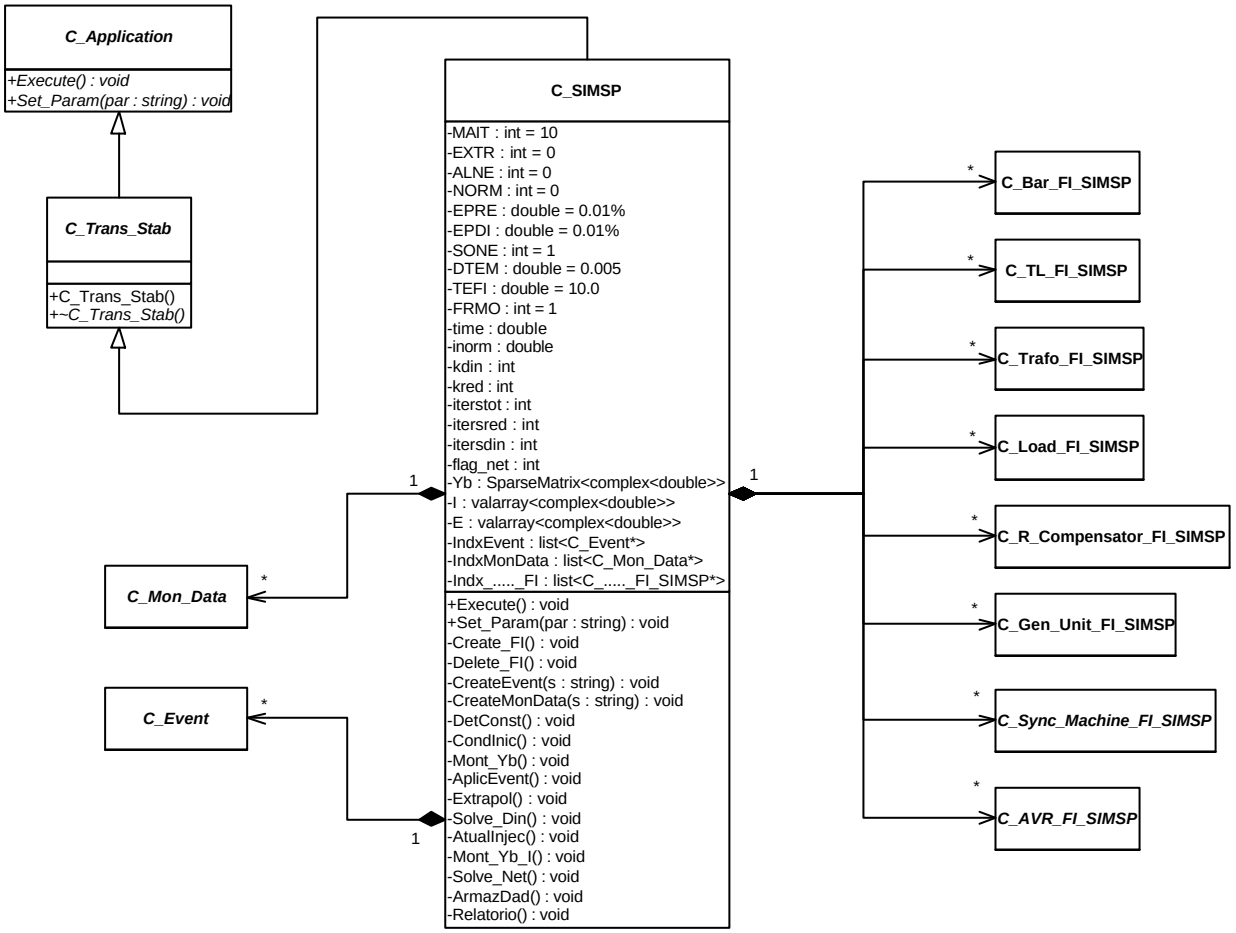


Figura 6-17 – Classe *C_SIMSP*

Os principais atributos e métodos desta classe são os seguintes:

→ *MAIT*: número máximo de iterações por passo de integração;

→ *EXTR*: habilita (1) / desabilita (0) a extrapolação das variáveis de estado;

→ *ALNE*: habilita método não entrelaçado (1) ou método entrelaçado (0);

→ *NORM*: define o tipo de norma a ser utilizada no cálculo dos resíduos das variáveis calculadas nos processos iterativos;

→ *EPRE*: tolerância para convergência das equações da rede elétrica;

→ *EPDI*: tolerância para convergência das equações diferenciais;

→ *SONE*: freqüência de solução das equações da rede elétrica;

→ *DTEM*: passo de integração numérica das equações diferenciais;

→ *TEFI*: tempo final de simulação;

→ *FRMO*: freqüência de monitoração dos dados;

→ *time*: tempo corrente de simulação;

→ *inorm*: norma calculada em cada passo de integração;

→ *kdin*: número de iterações realizadas para a convergência das equações diferenciais algebrizadas dos elementos, em cada passo de integração;

→ *kred*: número de iterações realizadas para a convergência das equações algébricas da rede elétrica, em cada passo de integração;

→ *iterstot*: número de passos de integração realizados;

→ *itersred*: número total de iterações realizadas para a convergência das equações algébricas da rede elétrica;

→ *itersdin*: número total de iterações realizadas para a convergência das equações diferenciais algebrizadas;

→ *flag_net*: indicador de mudanças na rede elétrica após um evento qualquer;

→ *Yb*: matriz de admitâncias nodais do sistema (sistema linear $\vec{I} = Y_b.\vec{E}$);

→ *I*: vetor de injeções de corrente nas barras (sistema linear $\vec{I} = Y_b.\vec{E}$);

→ *E*: vetor de tensões nas barras (sistema linear $\vec{I} = Y_b.\vec{E}$);

→ *IndxEvent*: lista de eventos a serem simulados;

→ *IndxMonData*: lista de dados a serem monitorados, para impressão ao final do processo;

→ *Indx_....._*FI: listas de interfaces funcionais dos elementos físicos;

→ *Create_FI()*: cria as interfaces funcionais;

→ *Delete_FI()*: elimina as interfaces funcionais;

→ *CreateEvent()*: cria os eventos a serem simulados no SEE;

→ *CreateMonData()*: cria os dados a serem monitorados durante a simulação;

→ *DetConst()*: comanda o cálculo dos termos constantes dos modelos dos elementos;

→ *CondInic()*: comanda o cálculo das condições iniciais das equações dinâmicas;

→ *Mont_Yb()*: calcula a matriz de admitâncias Y_b;

→ *AplicEvent()*: aplica os eventos em cada passo de integração;

→ *Extrapol()*: extrapola as variáveis de estado dos elementos dinâmicos;

→ *I*: comanda a solução das equações diferencias;

→ *AtualInjec()*: comanda o cálculo das injeções de corrente dos elementos, e o atualiza nas barras;

→ *Mont_Yb_I()*: calcula o vetor de injeção de correntes nas barras;

→ *Solve_Net()*: comanda a solução das equações da rede elétrica (sistema linear);

→ *ArmazDad()*: armazena os dados de monitoração em cada passo de integração;

→ *Relatorio()*: imprime o relatório com os dados monitorados.

Observa-se na Figura 6-17 que a classe *C_SIMSP* deriva de *C_Trans_Stab*, que por sua vez deriva de *C_Application*. A classe *C_Trans_Stab* generaliza todas as metodologias de análise da estabilidade transitória, tais como simulação dinâmica com modelagem detalhada (aqui descrita), avaliação global da estabilidade transitória com modelagem simplificada, etc.

A classe *C_SIMSP* é formada por diversas outras classes, tais como dados de monitoração e eventos a serem simulados, e também as interfaces funcionais dos elementos físicos. As interfaces modelam comportamentos específicos dos elementos físicos frente à metodologia de simulação dinâmica via Esquema Alternado Implícito. A matriz de admitâncias Y_b é representada através da estrutura *SparseMatrix*, apresentada em detalhes no Capítulo 7 (seção 7.1.1).

Ilustrando a seqüência de atividades que um objeto do tipo *C_SIMSP* realiza, através do seu método *Execute()*, é apresentado na Figura 6-18 um Diagrama de Atividades para a execução do processo de simulação dinâmica via Esquema Alternado.

6.4.1. Eventos Simulados no SEE

Os eventos modelam as ocorrências que podem ser aplicadas no SEE durante uma simulação dinâmica. Informações a respeito dos eventos que devem ser aplicados ao SEE são definidas na classe *C_SIMSP*, através da atribuição de parâmetros (método *Set_Param()*). Os eventos modelados neste trabalho são mostrados na Figura 6-19.

Os eventos derivam todos da classe abstrata *C_Event*, a qual contém os seguintes atributos e métodos principais:

→ *time*: tempo no qual o evento deve ser aplicado ao SEE;
→ *SIMSP*: apontador para o objeto de simulação (aplicação);
→ *Apply()*: aplica o evento ao SEE.

Os eventos modelados são: abertura total de circuito série (classe *C_Ev_ABCI*), fechamento total de circuito série (classe *C_Ev_FECI*), aplicação de curto-circuito em barra (classe *C_Ev_APCB*), e remoção de curto-circuito em barra (classe *C_Ev_RMCB*). As classes possuem atributos (apontadores) que identificam o elemento sobre o qual atuam (através das suas respectivas interfaces funcionais). Possuem também os atributos e métodos herdados de *C_Event*; o método *Apply()*, declarado como virtual puro na classe *C_Event*, é definido em cada classe derivada, e realiza a aplicação do evento. A conexão do(s) apontador(es) é realizada nos construtores das respectivas classes derivadas.

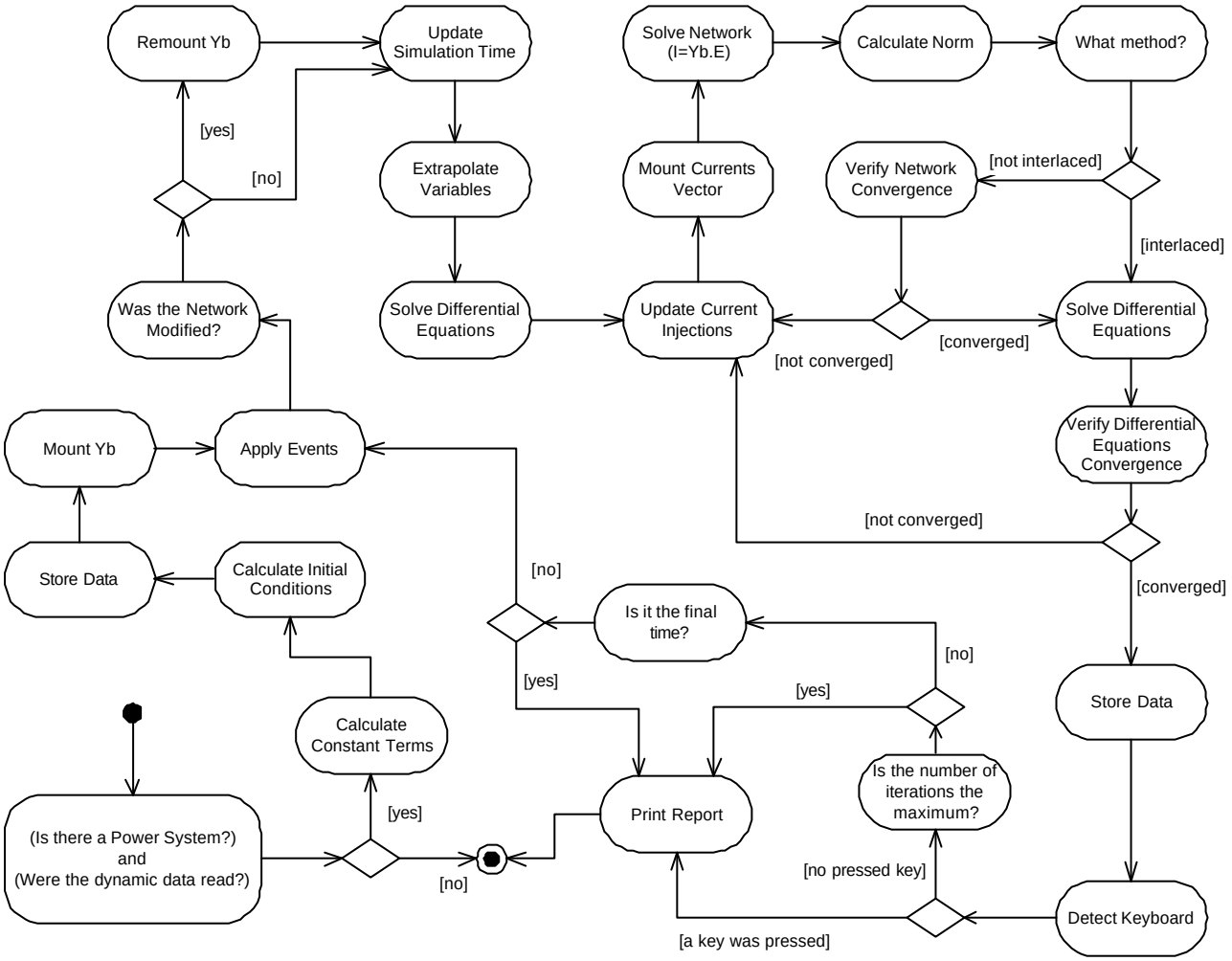


Figura 6-18 – Diagrama de Atividades para a Execução de uma Simulação da Dinâmica via Esquema Alternado.

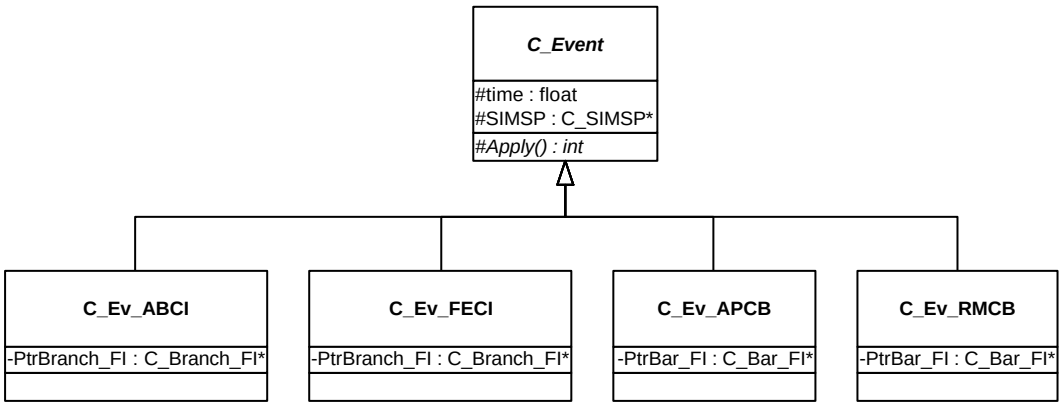


Figura 6-19 – Eventos em SEE Modelados

6.4.2. Dados de Monitoração

Os dados de monitoração representam as variáveis que serão monitoradas ao longo da simulação. A exemplo dos eventos, as variáveis que devem ser monitoradas são informadas ao objeto de simulação através da atribuição de parâmetros. As variáveis de monitoração representadas neste trabalho são mostradas na Figura 6-20.

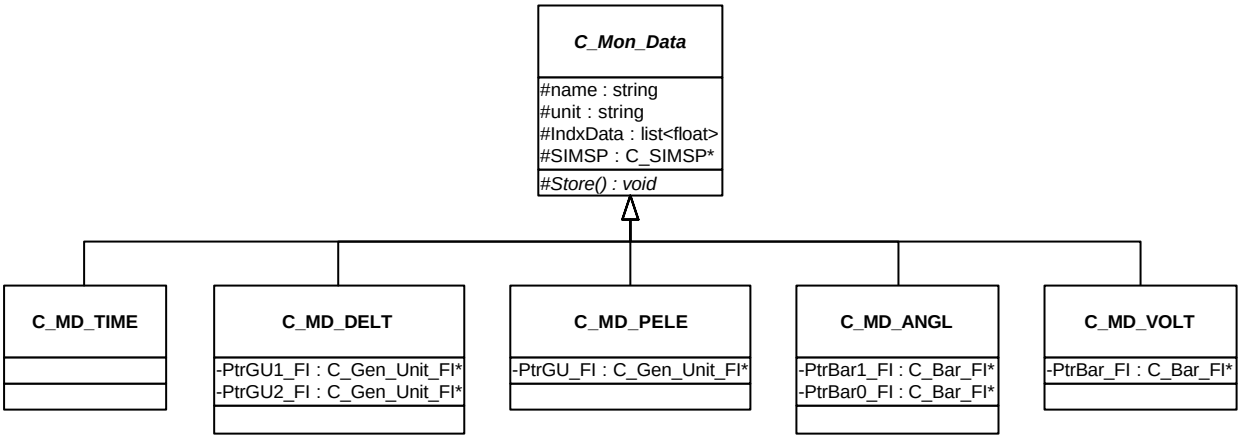


Figura 6-20 – Dados de Monitoração Modelados

A classe *C_Mon_Data* generaliza todos os dados de monitoração. Seus principais atributos e métodos são:

→ *name*: nome do dado monitorado;
→ *unit*: unidade de medição do dado;
→ *IndxData*: lista para armazenamento seqüencial dos valores monitorados;
→ *SIMSP*: apontador para o objeto de simulação (aplicação);
→ *Store()*: armazena o valor instantâneo da variável monitorada.

Os atributos tipo *string* de caracteres *name* e *unit* armazenam, respectivamente, o nome da variável monitorada e a unidade de medição da variável. Estes atributos são definidos no construtor de cada classe derivada.

As variáveis de monitoração modeladas são: tempo de simulação, em segundos (classe *C_MD_TIME*), ângulo do eixo em quadratura da máquina síncrona, medido em graus, relativo a uma máquina de referência (classe *C_MD_DELT*), potência elétrica ativa gerada por uma unidade de geração, medida em MW (classe *C_MD_PELE*), ângulo de tensão em uma barra, medido em graus (classe *C_MD_ANGL*), e módulo de tensão em uma barra, medido em pu (classe *C_MD_VOLT*). As classes possuem atributos (apontadores) que identificam o elemento físico do qual monitoram variáveis, através das suas respectivas interfaces funcionais. Possuem também os atributos e métodos herdados de

C_Mon_Data; o método *Store()*, declarado como virtual puro na classe *C_Mon_Data* e definido em cada classe derivada, realiza o armazenamento do valor instantâneo da variável sob monitoração. A conexão do(s) apontador(es) é realizada nos construtores das respectivas classes derivadas.

6.4.3. Discretização no Tempo das Equações Diferenciais e Esquema de Solução Utilizado

Para a discretização no tempo das equações diferenciais dos elementos do SEE foi utilizado o Método Trapezoidal Implícito, devido à sua característica de ser simetricamente A-estável. As equações diferenciais dos elementos, transformadas em equações algébricas a diferenças, são implementadas nas interfaces funcionais dos objetos representantes de cada elemento, sob a forma de métodos chamados *Solve()*, os quais resolvem as equações para um passo de tempo de integração. As interfaces ainda contam com métodos chamados *CondInic()*, os quais calculam as condições iniciais das equações.

O esquema utilizado para a solução das equações do SEE é o Alternado, onde se resolvem separada e alternadamente o conjunto de equações algébricas a diferenças relativas aos elementos dinâmicos do sistema, e o conjunto de equações algébricas da rede elétrica. Opcionalmente o objeto de simulação pode utilizar o método Alternado Entrelaçado, relaxando-se as condições de convergência das equações da rede elétrica, realizando apenas uma iteração para o cálculo dessas equações para cada iteração das equações diferenciais algebrizadas.

Assim, a combinação do método trapezoidal implícito com o esquema alternado dá origem ao esquema geral para simulação do sistema, o Alternado (Entrelaçado ou Não) Implícito. Esse esquema apresenta algumas vantagens em relação a outros disponíveis na literatura, tal como o Esquema Simultâneo, no qual as equações do SEE são todas resolvidas simultaneamente através da montagem de um sistema linear tipo “jacobiano”. Em termos de desempenho computacional, o esquema Alternado mostra-se mais eficiente que o Simultâneo. Outro ponto positivo é a facilidade para manutenções nos algoritmos que utilizam o esquema Alternado. Uma vez que as equações são resolvidas de forma independente, elas podem ser encapsuladas individualmente nos objetos representantes das funcionalidades dos elementos físicos do SEE, permitindo uma boa flexibilidade para inclusão de novos elementos ou modelos. Além, é claro, da robustez numérica do método trapezoidal implícito, utilizado em conjunto com o esquema Alternado.

Mais detalhes sobre o algoritmo de simulação utilizado na classe *C_SIMSP*, sobre o método trapezoidal implícito, ou sobre o esquema Alternado, podem ser encontrados em MANZONI (1996).

6.4.4. Representação dos Elementos Físicos – Interfaces Funcionais para a Simulação Dinâmica

No procedimento de simulação dinâmica de SEE representado pela classe *C_SIMSP*, a rede elétrica é representada pela equação algébrica $\vec{I} = Y_b.\vec{E}$, onde $\vec{I}$ é o vetor de injeção de correntes nas barras, Y_b é a matriz de admitâncias nodais do sistema, e $\vec{E}$ é o vetor de tensões nas barras. Depois de calculado o vetor de injeções de correntes nas barras, resultante da contribuição de corrente de cada elemento do SEE, soluciona-se o sistema linear, obtendo-se assim as tensões em cada barra. Caso esteja ativado o modo não entrelaçado, este processo é repetido até que uma norma, calculada com base nos resíduos de tensão, seja menor que um determinado valor máximo (0,01%, por definição).

A norma pode ser calculada de duas formas: máximo valor entre os resíduos individuais (se o atributo *NORM* valer 0 – norma padrão da aplicação), ou somatório dos quadrados dos resíduos (se *NORM* valer 2). No caso da rede elétrica, a variável considerada para o cálculo dos resíduos é o módulo da tensão nas barras. No caso da solução das equações diferenciais, consideram-se todas as variáveis de estado envolvidas no processo.

O comportamento dinâmico dos elementos físicos é encapsulado em interfaces funcionais, representando o comportamento dinâmico de cada elemento frente à uma simulação dinâmica via esquema Alternado Implícito. Os elementos que têm seu comportamento modelado por equações diferenciais possuem métodos chamados *Solve()*, os quais solucionam individualmente as equações diferenciais algebrizadas de cada elemento. A classe *C_SIMSP*, através do seu método *Solve()*, encarrega-se de percorrer as listas de (interfaces funcionais de) elementos físicos estruturais, ativando os seus métodos *Solve()*.

As interfaces funcionais são criadas juntamente com o objeto tipo *C_SIMSP* ao qual pertencem. Neste momento as interfaces já são conectadas aos seus elementos físicos correspondentes. As mesmas são eliminadas juntamente com a destruição do objeto tipo *C_SIMSP*.

As interfaces funcionais para a simulação dinâmica são apresentadas a seguir.

- **Classe *C_Bar_FI_SIMSP***

A classe *C_Bar_FI_SIMSP* modela o comportamento das barras do SEE em uma simulação dinâmica. A classe é apresentada na Figura 6-21.

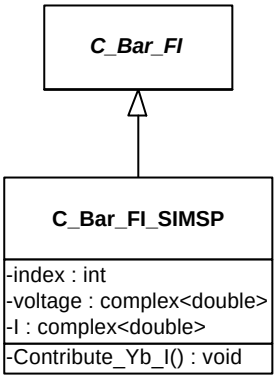


Figura 6-21 – Classe *C_Bar_FI_SIMSP*

Os principais atributos e métodos da classe são:

→ *index*: índice de localização da barra na matriz de admitâncias Yb;

→ *voltage*: tensão na barra;

→ *I*: injeção total de corrente na barra;

→ *Contribute_Yb_I()*: insere a injeção de corrente da barra no vetor independente do sistema linear a ser resolvido.

As barras, frente a uma simulação dinâmica, devem saber como dar sua contribuição no cálculo do vetor de injeções de corrente do sistema linear $\vec{I} = Y_b.\vec{E}$. Isto é feito pelo método *Contribute_Yb_I()*.

- **Classe *C_TL_FI_SIMSP***

A classe *C_TL_FI_SIMSP*, mostrada na Figura 6-22, representa o comportamento das linhas de transmissão em uma simulação dinâmica.

Seus principais atributos e métodos são:

→ *Ysh1*: admitância equivalente em derivação, acoplada à barra 1 da linha;

→ *Ysh2*: admitância equivalente em derivação, acoplada à barra 2 da linha;

→ *Yser*: admitância equivalente em série do circuito;

→ *Contribute_Yb()*: insere a contribuição do elemento no cálculo da matriz de admitâncias Y_b;

→ *Calc_Flows()*: calcula os fluxos de potência no elemento;

→ *Calc_Losses()*: calcula as perdas no elemento.

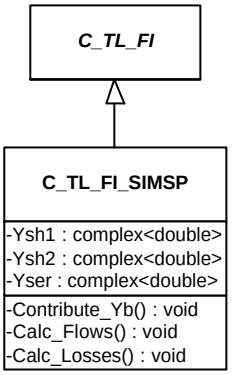


Figura 6-22 – Classe *C_TL_FI_SIMSP*

As admitâncias *Ysh1*, *Ysh2* e *Yser* referem-se ao modelo Π de representação utilizado para linhas de transmissão (e também transformadores). Seus valores são automaticamente calculados com base nos parâmetros do elemento físico, no momento da criação da interface funcional.

O método *Contribute_Yb()*, assim como nas classes apresentadas a seguir, insere a contribuição do respectivo elemento no cálculo da matriz de admitâncias do sistema (Y_b), a qual é utilizada na solução das equações da rede elétrica (sistema linear $\vec{I} = Y_b.\vec{E}$). Os métodos *Calc_Flows()* e *Calc_Losses()*, assim como na classe a seguir, calculam os fluxos de potência (ativa e reativa) na linha de transmissão e as suas perdas de potência, respectivamente, com base nos seus parâmetros e nas tensões das barras de conexão do elemento.

- **Classe *C_Trafo_FI_SIMSP***

A classe *C_Trafo_FI_SIMSP*, apresentada na Figura 6-23, representa o comportamento dos transformadores em uma simulação dinâmica.

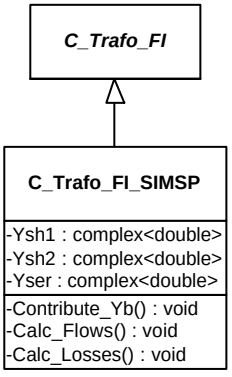


Figura 6-23 – Classe *C_Trafo_FI_SIMSP*

Seus principais atributos e métodos são:

→ *Ysh1*: admitância equivalente em derivação, acoplada à barra 1 da linha;
→ *Ysh2*: admitância equivalente em derivação, acoplada à barra 2 da linha;
→ *Yser*: admitância equivalente em série do circuito;
→ *Contribute_Yb()*: insere a contribuição do elemento no cálculo da matriz de admitâncias Y_b;
→ *Calc_Flows()*: calcula os fluxos de potência no elemento;
→ *Calc_Losses()*: calcula as perdas no elemento.

Assim como no caso da classe *C_TL_FI_SIMSP*, as admitâncias *Ysh1*, *Ysh2* e *Yser* referem-se ao modelo Π de representação utilizado para transformadores e linhas de transmissão. Seus valores são automaticamente calculados com base nos parâmetros do elemento físico, no momento da criação da interface funcional. No caso de estar sendo utilizado uma derivação (*tap*) diferente de 1, as admitâncias são automaticamente ajustadas. Não são modelados por esta classe transformadores com troca de derivação sob carga (LTCs), mantendo-se fixas as derivações de todos os transformadores durante o período de simulação.

- **Classe *C_Load_FI_SIMSP***

Esta classe representa o comportamento das cargas em uma simulação dinâmica via método Alternado. A classe é mostrada na Figura 6-24.

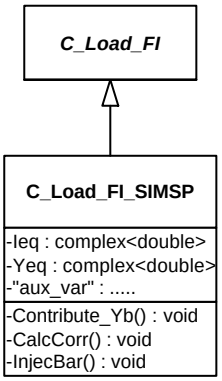


Figura 6-24 – Classe *C_Load_FI_SIMSP*

Os atributos e métodos principais da classe são:

→ *Ieq*: contribuição de corrente da carga na sua barra de conexão;
→ *Yeq*: admitância equivalente da carga;
→ "*aux_var*": conjunto de variáveis auxiliares para o funcionamento interno da classe;

→ *Contribute_Yb()*: insere a contribuição do elemento no cálculo da matriz de admitâncias Y_b;
→ *CalcCorr()*: calcula a contribuição de corrente da carga;
→ *InjecBar()*: injeta na barra a contribuição de corrente da carga.

A admitância equivalente *Yeq* é calculada no momento da criação da classe, com base nos valores nominais de potências e tensão da carga. Este valor de admitância é adicionado na matriz de admitâncias do sistema, pelo método *Contribute_Yb()*.

A cada passo de integração é recalculada a injeção de corrente relativa a cada carga, através do método *CalcCorr()*. Isto é feito com base na admitância equivalente da carga, no seu modelo polinominal, e na sua tensão terminal. A injeção de corrente é então adicionada na barra, através do método *InjecBar()*. As cargas que possuem somente parcelas representadas por modelos de impedância constante não contribuem com injeções de corrente.

- **Classe *C_R_Compensator_FI_SIMSP***

Esta classe representa o comportamento dos compensadores reativos (bancos de capacitores e indutores), conectados em derivação nas barras do SEE. A classe é mostrada na Figura 6-25.

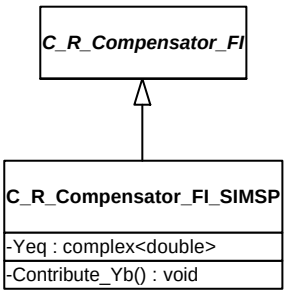


Figura 6-25 – Classe *C_R_Compensator_FI_SIMSP*

O atributo principal da classe é a admitância equivalente do compensador (*Yeq*), a qual é calculada na criação da classe, e adicionada posteriormente na matriz de admitâncias do sistema (método *Contribute_Yb()*).

- **Classe *C_Gen_Unit_FI_SIMSP***

Esta classe representa o comportamento das unidades de geração em uma simulação dinâmica via método Alternado. A classe é mostrada na Figura 6-26.

Os atributos e métodos principais da classe são:

→ *Ieq*: contribuição de corrente da unidade na sua barra de conexão;

→ *Yeq*: admitância equivalente da unidade;

→ *Contribute_Yb()*: insere a contribuição do elemento no cálculo da matriz de admitâncias Y_b;

→ *DetConst()*: comanda o cálculo dos termos auxiliares constantes das equações diferenciais dos elementos formadores da unidade;

→ *CondInic()*: comanda o cálculo das condições iniciais para as equações diferenciais dos elementos formadores da unidade;

→ *Extrapol()*: comanda a extrapolação das variáveis dos seus elementos formadores;

→ *Solve()*: comanda a solução das equações diferenciais dos elementos formadores da unidade;

→ *CalcCorr()*: calcula a contribuição de corrente da unidade;

→ *InjecBar()*: injeta na barra a contribuição de corrente da unidade.

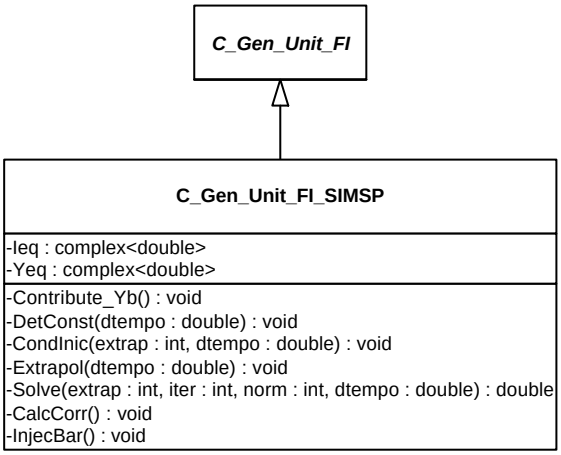


Figura 6-26 – Classe *C_Gen_Unit_FI_SIMSP*

Considerando-se que a unidade de geração é um objeto composto de diversos elementos de composição, a maioria de seus métodos comanda operações em seus elementos componentes. Os métodos *DetConst()*, *CondInic()* e *Solve()* comandam o acionamento de métodos de mesmo nome nas interfaces funcionais de máquinas síncronas e reguladores de tensão. *CalcCorr()* solicita à máquina síncrona integrante da unidade que calcule a sua injeção de corrente na barra de conexão.

DetConst() e *CondInic()* são acionados no início do processo de simulação, comandando o cálculo dos termos constantes e das condições iniciais das equações diferenciais dos elementos de composição formadores da unidade, respectivamente. Os termos constantes são normalmente dependentes do passo de integração utilizado no processo de simulação; estes termos resultam do processo de discretização aplicado às equações diferenciais dos elementos. A classe *C_SIMSP* não contempla controle de passo de integração

variável; porém, caso esta funcionalidade seja implementada futuramente, os termos constantes dos elementos devem ser recalculados a cada mudança de passo.

A solução das equações diferenciais algebrizadas de cada elemento formador da unidade de geração é controlada pelo método *Solve()*. Este método é acionado diversas vezes em cada passo de integração, até que seja alcançada a convergência do conjunto completo de equações.

A injeção de corrente equivalente da unidade de geração na sua barra de conexão é calculada pelo método *CalcCorr()*, e atualizado na barra pelo método *InjecBar()*.

- **Classe *C_Sync_Machine_FI_SIMSP***

As máquinas síncronas podem ter seu comportamento dinâmico modelado com diversos níveis de detalhamento, definindo diferentes modelos de representação. A classe abstrata *C_Sync_Machine_FI_SIMSP* serve como base para estes modelos, e é mostrada na Figura 6-27.

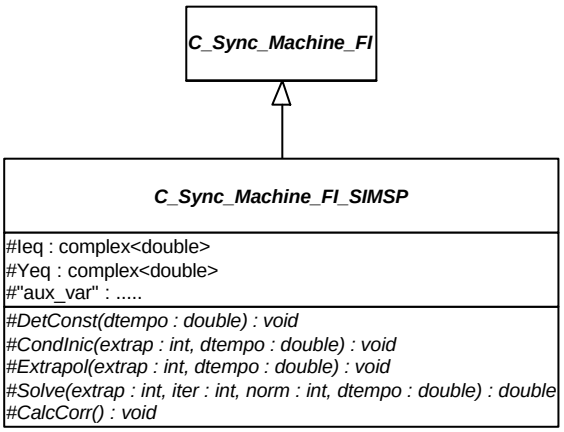


Figura 6-27 – Classe *C_Sync_Machine_FI_SIMSP*

Seus atributos e métodos principais são:

→ *Ieq*: contribuição de corrente da máquina na sua barra de conexão;
→ Yeq: admitância equivalente da máquina;
→ *"aux_var"*: conjunto de variáveis auxiliares para o funcionamento interno da classe;
→ *DetConst()*: calcula os termos auxiliares constantes das suas equações diferenciais;
→ *CondInic()*: calcula as condições iniciais para as suas equações diferenciais;
→ *Extrapol()*: extrapola as variáveis de estado;
→ *Solve()*: soluciona as suas equações diferenciais;
→ *CalcCorr()*: calcula a contribuição de corrente da máquina.

A máquina síncrona é o principal componente de uma unidade de geração, e é o componente que está diretamente conectado em derivação na barra de conexão da unidade. Assim, é a máquina síncrona que realmente produz potência elétrica, e possui uma admitância equivalente na unidade de geração. Estas características são representadas na classe *C_Sync_Machine_FI_SIMSP* pelos atributos *Ieq* e *Yeq*, os quais são numericamente iguais aos atributos de mesmo nome na classe *C_Gen_Unit_FI_SIMSP*.

Os métodos listados acima são declarados como virtuais, devendo ser implementados em cada classe derivada. Eles possuem as mesmas funções já descritas na classe *C_Gen_Unit_FI_SIMSP*.

- **Classes *C_Sync_Machine_I_FI_SIMSP*, *C_Sync_Machine_II_FI_SIMSP*, *C_Sync_Machine_IV_FI_SIMSP* e *C_Sync_Machine_Z_FI_SIMSP***

Neste trabalho, quatro níveis de detalhamento para o comportamento dinâmico de máquinas síncronas foram contemplados: modelo I (modelo simplificado ou clássico), modelo II (máquina de pólos salientes, considerando-se os efeitos transitórios), modelo IV (máquina de pólos salientes, considerando-se os efeitos subtransitórios), e modelo Z (impedância constante). Quatro classes foram criadas para a representação destes modelos: *C_Sync_Machine_I_FI_SIMSP*, *C_Sync_Machine_II_FI_SIMSP*, *C_Sync_Machine_IV_FI_SIMSP*, e *C_Sync_Machine_Z_FI_SIMSP*. Estas classes são mostradas na Figura 6-28.

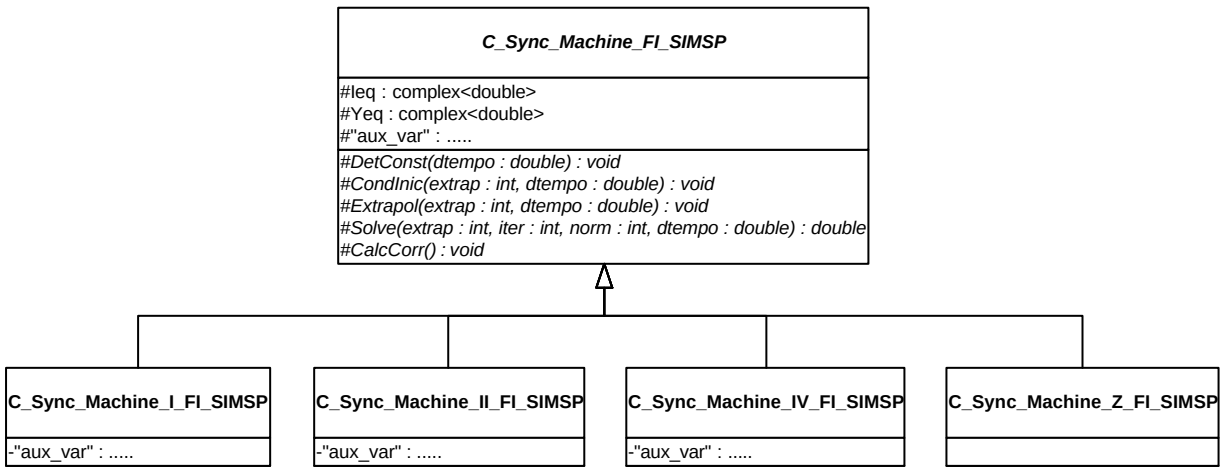


Figura 6-28 – Classes *C_Sync_Machine_I_FI_SIMSP*, *C_Sync_Machine_II_FI_SIMSP*, *C_Sync_IV_FI_SIMSP* e *C_Sync_Machine_Z_FI_SIMSP*

As classes possuem os atributos herdados de suas classes base, além de um conjunto de variáveis auxiliares, que facilitam o cálculo das equações diferenciais de cada modelo. Os métodos implementados em cada classe são aqueles declarados virtuais na classe base *C_Sync_Machine_FI_SIMSP*, que são implementados de acordo com a funcionalidade de cada modelo.

O modelo de impedância constante (classe *C_Sync_Machine_Z_SIMSP*) é utilizado para modelar as máquinas existentes no sistema, mas nas quais não se tem interesse em observar o comportamento dinâmico. A funcionalidade da classe restringe-se a calcular a impedância equivalente da máquina, com base nas suas potências geradas e na tensão da sua barra terminal.

Mais detalhes sobre os modelos de máquina síncrona contemplados neste trabalho, inclusive sobre seus equacionamentos, podem ser encontrados em MANZONI (1996).

- **Classe *C_AVR_FI_SIMSP***

Existem diversos tipos de sistemas de excitação (regulador de tensão + excitatriz), que apresentam comportamentos dinâmicos diferentes. A classe abstrata *C_AVR_FI_SIMSP* serve como base para as classes que representam as funcionalidades destes diversos tipos de sistemas, frente a uma simulação dinâmica via método Alternado Implícito. A classe é mostrada na Figura 6-29.

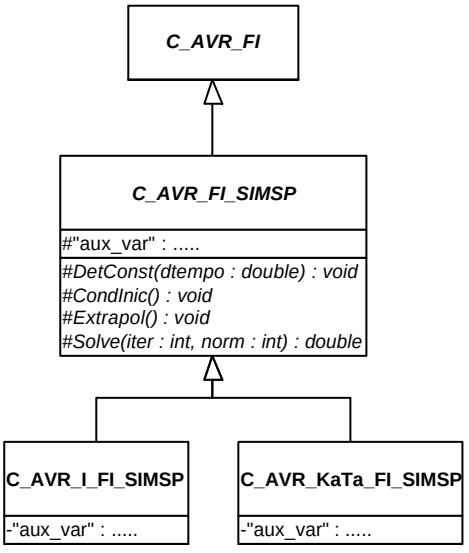


Figura 6-29 – Classe *C_AVR_FI_SIMSP*

A exemplo de *C_Sync_Machine_FI_SIMSP*, a classe *C_AVR_FI_SIMSP* possui um conjunto de variáveis auxiliares para a solução das equações do modelo matemático de cada tipo de sistema de excitação. Declara também quatro métodos virtuais, que deverão ser implementados nas classes derivadas, de acordo com a funcionalidade de cada sistema de excitação. São estes métodos:

→ *DetConst()*: calcula os termos auxiliares constantes das suas equações diferenciais;
→ *CondInic()*: calcula as condições iniciais para as suas equações diferenciais;
→ *Extrapol()*: extrapola as variáveis de estado;
→ *Solve()*: soluciona as suas equações diferenciais.

- **Classes *C_AVR_I_FI_SIMSP* e *C_AVR_KaTa_FI_SIMSP***

Neste trabalho, dois tipos de sistemas de excitação foram contemplados: modelo I e modelo KaTa. O modelo I representa sistemas de excitação estáticos, conforme modelo IEEE ST-1. O segundo representa um modelo simplificado, do tipo *ganho + constante de tempo*. Duas classes foram criadas para a representação das funcionalidades destes modelos de sistema de excitação, frente a uma simulação dinâmica via método Alternado Implícito: *C_AVR_I_FI_SIMSP* e *C_AVR_KaTa_FI_SIMSP*. Estas classes são mostradas na Figura 6-29.

A exemplo das interfaces funcionais para os diferentes níveis de modelagem de máquinas síncronas, as classes acima possuem os atributos herdados de suas classes base, além de um conjunto de variáveis auxiliares próprias, que facilitam o cálculo das suas equações diferenciais. Os métodos implementados em cada classe são aqueles declarados virtuais na classe base *C_AVR_FI_SIMSP*, que são implementados de acordo com a funcionalidade de cada tipo de sistema de excitação.

Mais detalhes sobre os modelos de sistemas de excitação contemplados neste trabalho, inclusive sobre seus equacionamentos, podem ser encontrados em MANZONI (1996).

6.5. Considerações Sobre o Uso de Elementos Físicos e Interfaces Funcionais

As interfaces funcionais traduzem os comportamentos que cada elemento físico do sistema apresenta em cada aplicação considerada. Utilizando-se o conceito de interfaces funcionais, modeladas e implementadas segundo o padrão de projeto Adapter, torna-se possível separar duas abstrações distintas na modelagem computacional de SEE: a re-

presentação física propriamente dita de um elemento, e suas funcionalidades frente a uma determinada metodologia de aplicação.

A representação física nada mais é do que a modelagem sob a forma de classes das características físicas que o elemento apresenta no mundo real. Observa-se o elemento instalado no sistema elétrico, e traduz-se sob a forma de classes (com atributos e métodos) esta observação, independentemente de que tipo de aplicação computacional será estudada sobre o sistema. Todos os diferentes comportamentos que um determinado elemento pode assumir, frente a diferentes aplicações computacionais, são modeladas posteriormente através das interfaces funcionais.

Além de separar dois conceitos distintos, esta construção permite que sejam realizadas trocas dinâmicas de aplicações, sem que seja necessário obter novamente todos os dados do sistema elétrico sob estudo. Ou seja, em tempo de execução de uma determinada ferramenta computacional, diversas aplicações podem ser executadas sobre um determinado sistema elétrico, o qual é adquirido uma única vez de um banco de dados de SEE, e mantido na memória do computador sob a forma de objetos, conforme a estrutura de representação dos elementos físicos. O que muda de uma aplicação para outra são as interfaces funcionais, que são criadas e associadas a cada elemento físico, de acordo com a aplicação em execução no momento. Pode-se entender que em um determinado instante de tempo, no qual uma determinada aplicação está sendo executada, cada elemento do SEE é representado pelo conjunto de dois objetos: seu objeto físico e sua interface funcional relativa à aplicação em execução.

A estrutura facilita também a modelagem e implementação posterior de novas metodologias de análise e síntese para SEE, utilizando-se o ambiente orientado a objetos desenvolvido. Quando do desenvolvimento de uma nova metodologia, projeta-se também as interfaces funcionais para os elementos ativos nesta metodologia, utilizando-se toda a base de classes dos elementos físicos já desenvolvida. Note-se que cada conjunto de interfaces funcionais pertence ao escopo de uma metodologia de aplicação, diferentemente da representação física dos elementos, que tem um caráter genérico dentro do ambiente.

Contudo, o ambiente desenvolvido também permite que aplicações sejam desenvolvidas sem o uso das interfaces funcionais. Desde que a aplicação esteja devidamente encapsulada, definindo um conjunto de métodos de interface adequados ao seu funcionamento, a forma com que isso acontece internamente à aplicação não diz respeito ao restante do projeto. Assim, caso se queira implementar uma determinada aplicação no ambiente proposto neste trabalho reutilizando grande parte de códigos já desenvolvidos para tal aplicação, que não seguem a filosofia proposta neste trabalho, tem-se apenas que

modelar e implementar os métodos de interface adequados, seguindo-se o exposto neste capítulo. Internamente, códigos já desenvolvidos podem ser reutilizados para a execução parcial ou total da aplicação. Dessa forma utiliza-se a estrutura de classes que representa os elementos físicos apenas como um banco de dados do sistema sob estudo, armazenado em memória durante a execução da aplicação.

Observe-se que esta prática não permite obter aplicações modeladas inteiramente sob a filosofia proposta neste trabalho. Isto é, aplicações adicionadas ao ambiente desta maneira poderão trazer problemas futuros, uma vez que não será possível padronizá-las seguindo-se adequadamente as idéias propostas na filosofia. Futuras manutenções internas à aplicação, ou mesmo tentativas de reutilização total ou parcial dos seus códigos, em outras metodologias de aplicação, certamente trarão problemas. Porém o ambiente de desenvolvimento proposto neste trabalho permite esta prática de forma proposital, até mesmo com o objetivo de facilitar desenvolvimentos iniciais utilizando-se o ambiente. Pode-se assim usufruir imediatamente de alguns benefícios do ambiente proposto, tais como as facilidades computacionais implementadas (apresentadas no capítulo seguinte), a medida que se aprimoram os conceitos e se desenvolve a aplicação de forma totalmente integrada à nova filosofia de projeto se software proposta neste trabalho.

CAPÍTULO 7

7. FERRAMENTAS COMPUTACIONAIS

Neste trabalho propõe-se uma nova filosofia para o desenvolvimento de ferramentas computacionais para o setor elétrico. Para isso apresentou-se no Capítulo 5 a modelagem dos elementos físicos dos SEE, e no Capítulo 6 apresentou-se a modelagem das metodologias de análise e síntese (ou aplicações), juntamente com as interfaces funcionais dos elementos físicos para cada aplicação.

Neste capítulo descreve-se a modelagem das ferramentas computacionais propriamente ditas para o setor elétrico, desenvolvidas sob a nova filosofia proposta. Inicialmente são abordadas as facilidades computacionais desenvolvidas no âmbito deste trabalho, sob a forma de *pacotes*. A seguir é apresentada uma visão geral do projeto e implementação de uma ferramenta computacional utilizando-se o ambiente de desenvolvimento proposto. Encerrando, apresenta-se um protótipo de ferramenta computacional, desenvolvido com o objetivo de validar a filosofia proposta.

7.1. Facilidades Computacionais

Em uma proposta de utilização de MOO na representação de SEE, com o objetivo mais amplo de se projetar e implementar um conjunto de ferramentas computacionais para o setor elétrico, em um ambiente integrado de desenvolvimento, outras estruturas devem ser concebidas, além daquelas que representam os elementos físicos do sistema, e suas metodologias de aplicação. Uma série de atividades computacionais auxiliares são executadas em uma ferramenta computacional completa, tais como gerenciamento de telas, entrada e saída de dados (leitura e escrita de arquivos, bancos de dados, etc.), ope-

rações matriciais (solução de sistemas lineares), etc. Estas atividades devem ser também modeladas e implementadas sob o paradigma da MOO, de forma a integrá-las da melhor maneira possível às estruturas representativas das entidades do SEE.

Note-se que não se quer com isso esquecer códigos específicos, já desenvolvidos e com utilização consolidada em diversas áreas do conhecimento. Pelo contrário, este trabalho, conforme já exposto anteriormente, corrobora e faz uso do conceito de reutilização de códigos, pretendendo-se evitar com isso que problemas já (bem) resolvidos sejam novamente abordados, evitando-se com isso o "re-trabalho".

Assim, estas atividades são neste trabalho representadas por um conjunto de classes específicas, e concentradas em uma abstração denominada Facilidades Computacionais. Nas situações em que códigos já desenvolvidos são reutilizados (como no caso da solução de sistemas lineares, por exemplo), classes são desenvolvidas para representar interfaces que sejam adequadas ao uso destes códigos em projetos que seguem a filosofia da MOO.

A seguir são apresentadas as principais facilidades computacionais projetadas e implementadas neste trabalho. Estas facilidades são agrupadas em pacotes independentes, de forma a tornar o mais simples possível sua reutilização, mesmo em protótipos ou programas desenvolvidos à margem da filosofia aqui proposta.

7.1.1. Armazenamento e Tratamento de Matrizes Esparsas

A maioria das metodologias de aplicação na área de SEE está baseada na soluções de sistemas lineares, formados normalmente por matrizes altamente esparsas e de grande porte. Sendo assim, uma estrutura de classes foi criada para o armazenamento e tratamento dessas matrizes. A estrutura é mostrada na Figura 7-1.

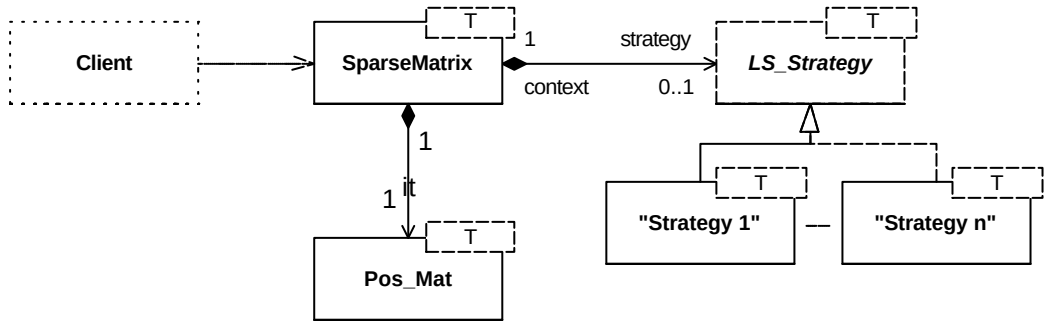


Figura 7-1 – Estrutura para Armazenamento e Tratamento de Matrizes Esparsas e de Grande Porte

Observa-se na figura a classe principal da estrutura, *SparseMatrix*. Esta classe é responsável pelo armazenamento de matrizes esparsas e de grande porte, implementando técnicas computacionais eficientes para tanto. Utilizando o padrão de projeto Strategy, a classe *LS_Strategy* define uma família de estratégias de solução de sistemas lineares, as quais colaboram com a classe *SparseMatrix* para este fim. Uma classe auxiliar *Pos_Mat* é criada para trabalhar em conjunto com *SparseMatrix*, auxiliando no processo de procura por elementos dentro da matriz.

Todas as classes são do tipo *template*, o que significa que podem trabalhar com qualquer tipo de dado. Matrizes contendo tipos padrão, tais como inteiros ou ponto flutuante (precisão simples ou dupla), e até mesmo tipos novos, criados por terceiros, tais como blocos matriciais 2_x2, por exemplo, podem ser perfeitamente gerenciadas pela estrutura. Note-se que, para isso, os tipos precisam estar completamente definidos, com suas operações básicas devidamente sobrecarregadas.

A seguir cada uma das classes da estrutura é explicitada.

- **Classe *SparseMatrix***

Esta classe é a responsável pela representação e armazenamento de matrizes. Apesar de trabalhar qualquer tipo de matriz, inclusive retangulares, é especializada em matrizes quadradas esparsas e de grande porte, pois implementa internamente técnicas de armazenamento compacto, guardando apenas elementos diferentes de zero. Ainda, caso a matriz seja simétrica, armazena apenas os elementos triangulares superiores, otimizando o uso de recursos computacionais. A classe pode ser observada em detalhes na Figura 7-2.

Os principais atributos e métodos da classe são os seguintes:

→ *nlin*: número de linhas da matriz;
→ *ncol*: número de colunas da matriz;
→ *nnz*: número de elementos diferentes de zero;
→ *flagsym*: indica se a matriz é simétrica (1) ou assimétrica (0);
→ *diag*: vetor que armazena os elementos diagonais da matriz;
→ *inilin*: estrutura para o armazenamento compacto dos elementos não diagonais diferentes de zero;
→ *it*: apontador para o objeto auxiliar *Pos_Mat*;
→ *strategy*: apontador para a estratégia de solução de sistema linear associada a matriz;
→ *SparseMatrix(...)*: construtores para a classe;
→ *Get_Elem_Next()*: retorna o elemento não diagonal apontado pelos índices (l,c), ou o próximo, caso o elemento seja nulo;
→ *Dim()*: dimensiona a matriz;

→ *Set_strategy()*: associa uma estratégia de solução de sistema linear à matriz;

→ *Solve_LinSys()*: soluciona um sistema linear com a matriz corrente, e o vetor independente *b* informado;

→ *Reset()*: reinicializa a classe, apagando todos os seus elementos;

→ *operator []*: sobrecarga do operador colchetes [ ], para acesso direto aos elementos da matriz;

→ *operator ()*: sobrecarga do operador parênteses ( ), para acesso direto aos elementos da matriz;

→ *operator =*: sobrecarga do operador igualdade, para cópia total dos elementos da matriz (não copia a estratégia).

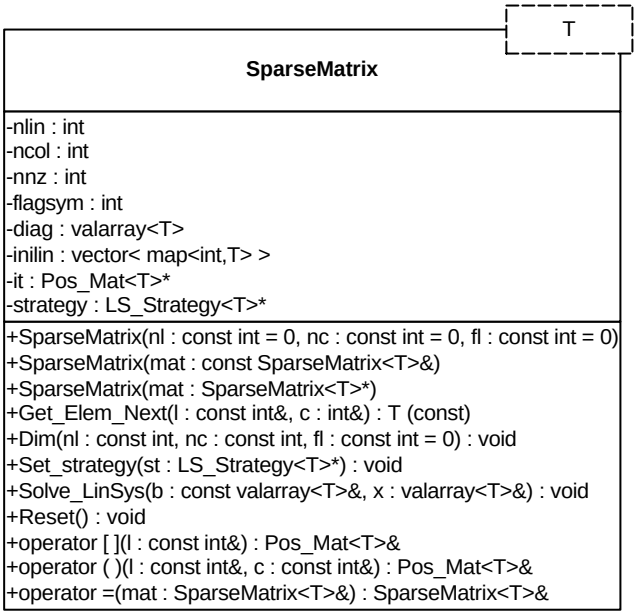


Figura 7-2 – Classe *SparseMatrix*

A estrutura interna de armazenamento da classe é mostrada na Figura 7-3.

A classe armazena os elementos não diagonais diferentes de zero em um vetor de listas indexadas (*inil*). O armazenamento é realizado por linhas, ou seja, para cada linha da matriz existe uma lista indexada, que armazena o valor do elemento, juntamente com sua coluna. O vetor *inil* tem *nlin* posições, sendo *nlin* o número de linhas da matriz. Os elementos pertencentes à diagonal são armazenados em um vetor chamado *diag*.

A classe *SparseMatrix* (em conjunto com *Pos_Mat*) sobrecarrega os operadores colchetes [ ] e parênteses ( ), permitindo acesso direto a elementos da matriz. Por exemplo, para atribuir o valor do elemento da linha l e coluna c da matriz A, na variável x, pode ser utilizada uma das duas formas a seguir:

x = A[l][c] ou x = A(l,c)

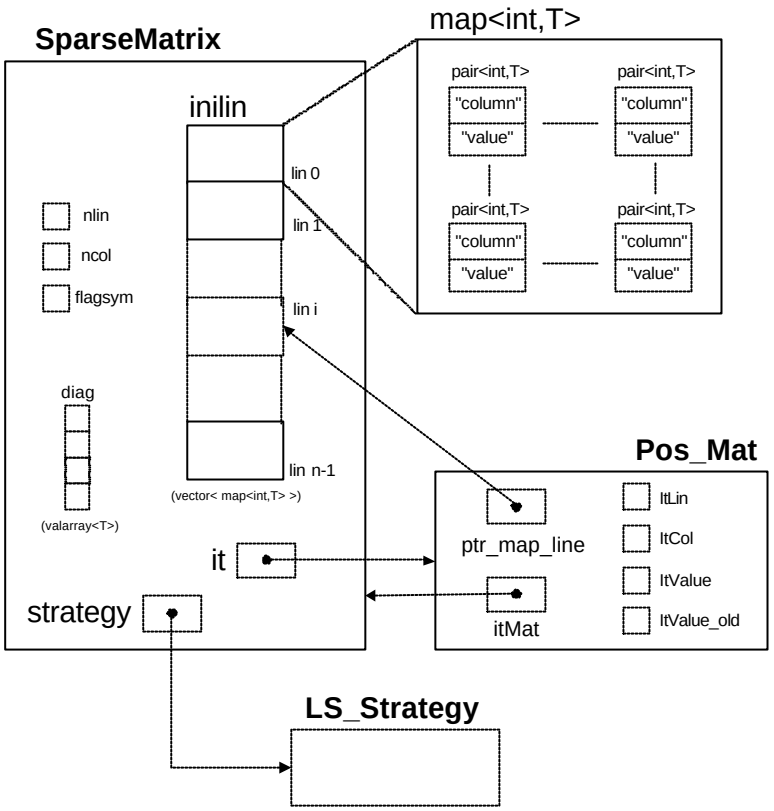


Figura 7-3 – Estrutura Interna de Armazenamento da Classe *SparseMatrix*

Da mesma forma, no caso de uma atribuição a um elemento da matriz, a mesma notação pode ser utilizada:

A[l][c] = x ou A(l,c) = x

A matriz deve ser sempre dimensionada antes do uso. Este dimensionamento pode ser feito no momento da criação da matriz, através de um construtor que recebe o número de linhas e de colunas, ou posteriormente, através do método *Dim()*. Juntamente com o dimensionamento deve-se indicar se a matriz é simétrica (1) ou não (0).

A classe implementa ainda dois outros construtores: um que recebe uma outra matriz através de referência, e outro que recebe um apontador para outra matriz. Em ambos os casos é realizada uma cópia da matriz original, a menos da estratégia de solução de sistema linear, que somente é associada através do método *Set_strategy()*.

O método *Get_Elem_Next()* permite a um cliente ou estratégia percorrer somente os elementos não diagonais diferentes de zero da matriz. Ao se solicitar *Get_Elem_Next(i,j)*, o método tenta retornar o elemento da linha *i* e coluna *j*. Caso este elemento seja nulo, retorna o próximo elemento não diagonal diferente de zero da linha *i*, e atualiza *c* com o valor da coluna do elemento retornado. Caso o final da linha seja alcançado, retorna zero, e

faz *c = ncol* (deve ser lembrado que o sistema é todo baseado em índice zero, ou seja, a maior coluna é a *ncol-1*).

- **Solução de Sistemas Lineares – Classe *LS_Strategy***

A solução de sistemas lineares (SSL) foi implementada na estrutura descrita acima através do padrão de projeto Strategy, encapsulando diferentes métodos de SSL em diferentes classes (estratégias). A classe *LS_Strategy* é mostrada na Figura 7-4.

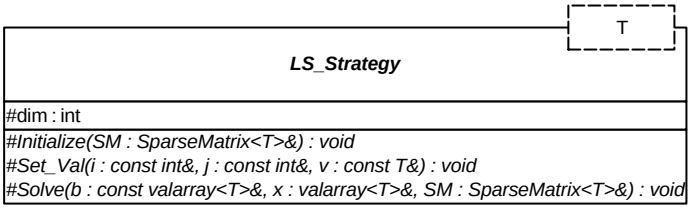


Figura 7-4 – Classe *LS_Strategy*

Esta é uma classe abstrata, que serve como base para todas as estratégias de SLL. Seus principais atributos e métodos são:

→ *dim*: dimensão do sistema linear;
→ *Initialize()*: inicializa uma estratégia;
→ *Set_Val()*: atribui valores na estrutura interna à estratégia, para o armazenamento dos valores da matriz;
→ *Solve()*: soluciona o sistema linear.

A classe declara três métodos virtuais, que devem ser obrigatoriamente definidos em cada estratégia. O método *Initialize()* é responsável por inicializar uma estratégia. Este método é acionado pela classe *SparseMatrix* toda vez que uma nova estratégia é atribuída a matriz. Em seu interior devem constar todos os passos necessários para a inicialização dos atributos internos de uma estratégia.

Cada estratégia de SSL pode conter em seu interior estruturas especializadas para o armazenamento dos valores da matriz do sistema que esta sendo solucionado, permitindo uma maior eficiência da estratégia. Assim, cada vez que um elemento é modificado na matriz tipo *SparseMatrix*, essa classe aciona o método *Set_Val()* de sua estratégia, para que se mantenham atualizadas as estruturas internas da estratégia.

Já o método *Solve()* é o responsável pela solução propriamente dita do sistema linear, formado pela matriz e pelo vetor independente *b*, informado como atributo em *Solve()*. A solução é armazenada em *x*.

Note-se que qualquer método para a SSL deve ser implementado como classe derivada de *LS_Strategy*, seguindo a interface declarada por esta classe. Métodos diretos ou indiretos, especializados em resolver sistemas simétricos ou assimétricos, devem sempre se adaptar à interface declarada em *LS_Strategy*, formada pelos três métodos descritos acima.

O procedimento de SSL propriamente dito de cada estratégia deve estar completamente encapsulado no método *Solve()*. A seqüência de execução de etapas internas em um determinado método (ordenação, fatoração, substituição inversa e direta, por exemplo, em métodos diretos) deve ser completamente controlada pela estratégia, através de *flags* de controle. Por exemplo, a necessidade ou não de uma nova ordenação e/ou fatoração da matriz, após a modificação de um valor numérico qualquer na matriz, deve ser avaliada internamente pela estratégia. Desta forma, o uso da estrutura como um todo por clientes externos torna-se transparente, sendo necessários apenas dois comandos para a solução de um determinado sistema linear, ambos da classe *SparseMatrix* (atribuição de uma estratégia – comando *Set_strategy()* – e solicitação de solução do sistema – e comando *Solve_LinSys()*).

É verdadeira a observação que uma estrutura interna à estratégia para o armazenamento dos elementos da matriz caracteriza uma redundância de dados. Porém esta redundância é de grande utilidade em termos de eficiência computacional da estratégia, uma vez que a operação dos métodos para a SSL são normalmente dependentes da forma com que os dados da matriz são armazenados. Além disso, o par de métodos *Initialize()* e *Set_Val()* permite que o sincronismo entre os elementos armazenados no objeto tipo *SparseMatrix* e na sua estratégia seja mantido. Nestes casos em que existe tal estrutura interna, ela normalmente será inicializada com uma cópia dos elementos da matriz pelo método *Initialize()*, e depois a cada alteração nos valores da matriz original, o método *Set_Val()* encarrega-se de atualizar a estrutura interna. Caso seja implementado um determinado método para a SSL que não necessite de uma estrutura interna, o método *Set_Val()* poderá ser ignorado, definindo-o vazio.

Caso seja realizado qualquer tipo de reordenação nos elementos da matriz, etapa normalmente encontrada em métodos diretos, com o objetivo de minimizar a criação de preenchimentos no processo de fatoração da matriz, esta deve ser realizada na estrutura interna de armazenamento da estratégia, sem se alterar o objeto tipo *SparseMatrix*. Ainda, a seqüência de reordenação deve ser armazenada, de forma a permitir que o vetor solução seja devolvido pelo método *Solve()* na ordem original em que o vetor independente *b* e o objeto tipo *SparseMatrix* foram montados. Deve ser enfatizado que procedimentos tais como a ordenação são conceitos matemáticos, que dizem respeito ao algoritmo matemá-

tico de solução de um sistema linear, e portanto não são pertinentes ao objeto do tipo *SparseMatrix*, muito menos ao cliente externo que está fazendo uso da estrutura para armazenar e manipular matrizes (métodos de análise e síntese de SEE, por exemplo). A utilização da estrutura deve ser o mais transparente possível, abstraindo-se conceitos matemáticos do cliente, encapsulando-os nas estratégias de solução.

- **Considerações Sobre a Criação e Uso da Estrutura *SparseMatrix* e sua Funcionalidade de Solução de Sistemas Lineares**

A estrutura descrita acima para a SSL não propõe um novo método para a SSL propriamente dita. Seu objetivo principal é o de permitir que os mais diversos métodos para este fim possam vir a ser utilizados em projetos orientados a objetos, de forma encapsulada em estratégias de solução, abstraindo-se conceitos matemáticos dos clientes. A estrutura permite inclusive que bibliotecas já desenvolvidas e exaustivamente testadas, disponíveis sob comercialização ou livres, sejam incorporadas nas ferramentas desenvolvidas sob a filosofia proposta neste trabalho. A troca de um método para outro é realizada da forma mais simples possível, ou seja, através de um comando apenas (método *Set_strategy()*, da classe *SparseMatrix*). Códigos e bibliotecas desenvolvidas sob outros enfoques que não a MOO, mesmo em diferentes linguagens de programação, podem vir a ser perfeitamente utilizadas em projetos orientados a objetos, de forma encapsulada.

Neste trabalho foram implementados sob a forma de estratégias três métodos para a SSL, um primeiro baseado em uma fatoração LU para sistemas simétricos (MOROZOWSKI FILHO, 1981), um segundo baseado no método da Bi-Fatoração de Zollenkopf (ZOLLENKOPF, 1971), e um terceiro baseado na biblioteca SPOOLES (ASHCRAFT et al., 1999). O primeiro método está implementado em linguagem Fortran 77, e foi encapsulado em uma classe (*LS_CTRI3*), a qual realiza o acoplamento do código em Fortran com o restante do projeto da estrutura. O segundo método foi implementado diretamente em linguagem C++, e também se encontra encapsulado em uma classe, chamada *LS_Zollen*. SPOOLES é uma biblioteca para a SSL esparsos, disponível on-line na NETLIB (www.netlib.org), a qual foi projetada utilizando-se o paradigma da MOO, e implementada em linguagem C. Além da SSL propriamente dita, a biblioteca realiza uma série de outras operações relacionadas a vetores, matrizes e sistemas lineares. Para efeito de sua utilização na estrutura *SparseMatrix*, reuniu-se um conjunto mínimo de operações que realizam a solução de um sistema linear esparso, e encapsulou-se este conjunto de operações em uma classe chamada *LS_SPOOLES*.

Mais detalhes sobre a estrutura *SparseMatrix*, assim como resultados de ensaios computacionais realizados com a estrutura, e suas estratégias de SSL, podem ser encontrados em AGOSTINI (2002a).

7.1.2. Leitor de Comandos

Neste trabalho desenvolveu-se uma estrutura de classes para controlar o envio de comandos para uma determinada ferramenta computacional. Basicamente, uma ferramenta poderá receber comandos de duas fontes: diretamente do teclado (modo interativo), ou de um arquivo de comandos (arquivo em lote - modo *batch*). Utilizando-se novamente o padrão de projeto Strategy, criou-se uma estrutura de classes que permite abstrair este gerenciamento da ferramenta propriamente dita, encapsulando esta funcionalidade em um conjunto de classes específicas. Esta estrutura é mostrada na Figura 7-5.

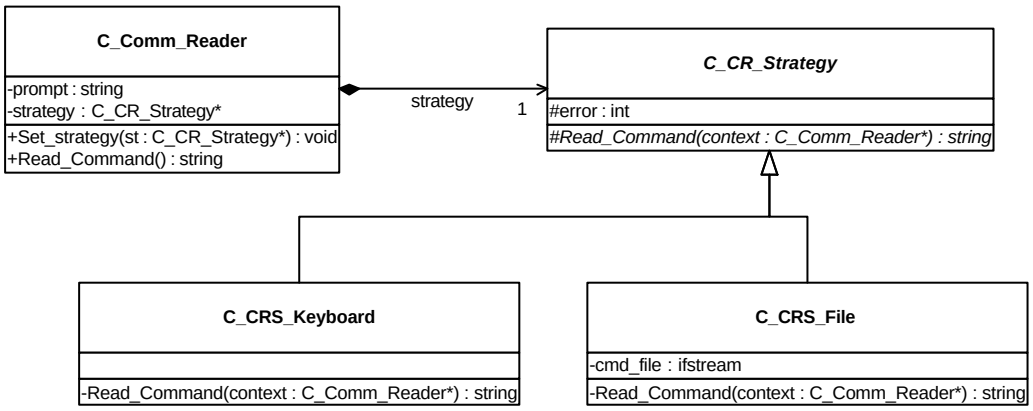


Figura 7-5 – Estrutura de Classes para a Leitura de Comandos

A principal classe da estrutura é a *C_Comm_Reader*, que tem os seguintes atributos e métodos principais:

→ *prompt*: texto que é impresso na tela, quando aguardando comando;
→ *strategy*: apontador para a estratégia de leitura de comandos;
→ *Set_strategy()*: atribui uma estratégia de leitura de comandos ao objeto tipo *C_Comm_Reader*;
→ *Read_Command()*: solicita que um novo comando seja lido.

A classe abstrata *C_CR_Strategy* define uma família de estratégias de leitura de comandos, encapsuladas em classes distintas. Possui o atributo *error*, o qual sinaliza através de códigos inteiros a ocorrência de algum erro durante uma leitura (zero indica que

não foram constatados erros). Declara o método virtual *Read_Command()*, o qual é definido em cada estratégia de leitura, de acordo com a estratégia.

Foram implementadas estratégias para a leitura de comandos diretamente do teclado e através de um arquivo em lote, representadas pelas classes *C_CRS_Keyboard* e *C_CRS_File*, respectivamente. A primeira apenas define o método *Read_Command()* utilizando o teclado como entrada. A classe *C_CRS_File*, além de definir o método *Read_Command()*, possui o atributo *cmd_file*, que identifica o arquivo de comandos. Outras estratégias de comando podem facilmente ser implementadas, como por exemplo a leitura de comandos vindos de um processo mestre, em uma ferramenta paralela; ou leitura de comandos de uma interface gráfica remota via rede, em ferramentas que podem ser executadas remotamente (aplicações remotas via WEB, por exemplo); etc.

O uso do padrão Strategy para a construção desta estrutura novamente aumenta modularidade do projeto como um todo, abstraindo da ferramenta em si o processo de controle da sua seqüência de execução. Este processo é encapsulado em uma estrutura a parte, e esta separação de conceitos facilita procedimentos de manutenção e atualização dos projetos desenvolvidos segundo a filosofia proposta neste trabalho.

7.1.3. Gerenciador de Banco de Dados

Os dados de um SEE podem ser armazenados permanentemente em diversos formatos de banco de dados. Para que uma ferramenta computacional possa ler dados dos SEE nos mais diferentes formatos, é interessante tornar o processo de leitura de dados independente do restante da ferramenta, permitindo que códigos para a leitura de novos formatos sejam facilmente implementados, sem maiores implicações no restante do projeto.

Com este intuito, desenvolveu-se uma estrutura de classes baseada no padrão Strategy, que é responsável pela leitura dos dados dos elementos dos SEE, e seu armazenamento nas estruturas para a representação dos elementos físicos (descritas no Capítulo 5). A estrutura é apresentada na Figura 7-6.

Nesta estrutura, o cliente (normalmente o objeto do tipo ferramenta computacional) cria e atribui uma certa estratégia de leitura de dados para o objeto do tipo *C_DB_Manager*, através do método *Set_strategy()*, e solicita a este que seja realizada a leitura dos dados do SEE, através do método *Read_System()*. O objeto do tipo *C_DB_Manager* repassa então o pedido para a sua estratégia (método *Read_System()*, da

classe *C_DB_Strategy*), que realiza a leitura a partir do arquivo informado através do atributo *fn* (nome do arquivo).

A classe abstrata *C_DB_Strategy* é base para uma família de estratégias de leitura de dados de SEE. Utilizando-se o padrão de projeto Template Method, declara-se e define-se na classe o método *Read_System()*, o qual armazena um apontador para o sistema elétrico que será montado a partir dos dados lidos (atributo *PS*), e chama o método *Primitive()*. Este método é apenas declarado privado na classe *C_DB_Strategy*, e então definido em cada estratégia, de acordo com as operações necessárias para a leitura dos dados na estratégia.

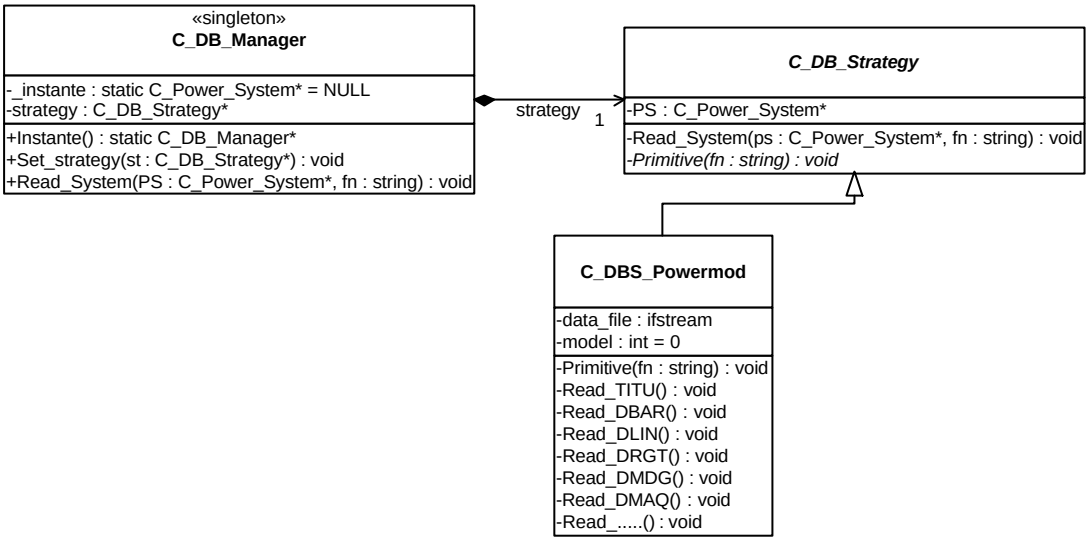


Figura 7-6 – Estrutura de Classes para a Leitura de Dados de SEE

Neste trabalho apenas uma estratégia de leitura de dados de SEE é implementada, encapsulada na classe *C_DBS_Powermod*. Esta estratégia é especializada na leitura de arquivos no formato ASCII, compatíveis com o Programa de Análise de Redes – ANAREDE (CEPEL, 1999a) (padrão POWERMOD, ou PECO). Novas estratégias para a leitura de outros tipos de dados (formato SAVECASE, utilizado também pelos programas do CEPEL; ou novos formatos, tais como bancos de dados orientados a objetos) podem ser facilmente anexadas à estrutura, de forma que as alterações no restante do projeto são mínimas (apenas o comando de criação e atribuição da estratégia de leitura é modificado).

A classe *C_DB_Manager* é projetada para ter uma instância única em tempo de execução (padrão Singleton), uma vez que não é necessária a existência de mais de um objeto deste tipo no âmbito da ferramenta computacional.

7.1.4. Gerenciador de Telas

Este trabalho prevê a existência de uma (ou um conjunto de) classe(s) responsável pelo gerenciamento de telas em uma ferramenta computacional final. Porém, tendo-se em vista o caráter científico do protótipo desenvolvido no escopo do presente trabalho, o qual tem como objetivo principal a validação da filosofia e base computacional propostas, este pacote para o gerenciamento de telas não foi implementado. Seu detalhamento fica sugerido como parte de trabalhos futuros, no desenvolvimento de ferramentas computacionais completas para o setor elétrico.

7.2. Ferramentas Computacionais – Aspectos Gerais

Conforme abordado nos capítulos anteriores, três grandes abstrações são identificadas no escopo da nova filosofia de projeto e implementação de softwares para SEE, proposta neste trabalho: abstração dos elementos físicos do sistema, abstração das metodologias de análise e síntese (aplicações), e finalmente a abstração das ferramentas computacionais de apoio ao planejamento e à operação de SEE.

Uma ferramenta computacional é o produto final de um processo de projeto de software, e deve englobar e gerenciar as diversas entidades identificadas ao longo do projeto. No presente trabalho considera-se uma ferramenta como sendo um objeto composto por diversos outros objetos, os quais representam os elementos físicos de um SEE (juntamente com suas conexões), as metodologias de análise e síntese, e módulos para atividades específicas (chamadas aqui de facilidades computacionais), tais como gerenciamento de telas, realização de operações matemáticas (solução de sistemas lineares), etc. Pode-se dizer que é o objeto de mais alto nível no projeto de um software, possuindo um método do tipo *Execute()*, o qual é responsável por gerenciar todo o processo de execução do programa.

Uma vez implementadas as classes que originam os objetos listados acima, a construção de uma ferramenta não segue necessariamente padrões rígidos. Nesta seção apresentam-se algumas diretrizes para o desenvolvimento dessas ferramentas. São apenas diretrizes, pois o desenvolvimento de cada ferramenta específica pode exigir particularidades nos projetos (e certamente o fará).

Uma primeira diretriz seria a consideração de que cada ferramenta deve conter pelo menos um SEE, o qual será estudado pelo usuário com o auxílio da ferramenta. Uma vez contendo o objeto tipo *C_Power_System*, apresentado no Capítulo 5, a ferramenta contém

todos os elementos físicos formadores deste sistema. Por questão de simplicidade, considera-se neste trabalho a existência de um único SEE por ferramenta (através do uso do padrão Singleton no projeto da classe *C_Power_System*). Porém, esta consideração não é definitiva; futuras ferramentas poderão ser desenvolvidas para análise de mais de um SEE simultaneamente.

Em uma segunda diretriz pode ser considerado que uma determinada ferramenta pode conter um número qualquer de aplicações, dependendo do escopo a que se destina. Exemplificando, uma ferramenta para avaliação da segurança dinâmica de SEE pode conter, além de uma aplicação específica de avaliação da segurança, outras aplicações, tais como uma para cálculo de fluxo de potência, e/ou outra aplicação para análise linear da dinâmica. Estas outras aplicações adicionam funcionalidades à ferramenta hipotética, a qual poderia executar cálculos de fluxo de potência de modo isolado antes da execução de uma avaliação da segurança propriamente dita, definindo pontos de operação para o sistema, assim como poderia, através de sua aplicação de análise linear da dinâmica, identificar modos instáveis no SEE, e suas causas.

É importante observar a diferença entre uma ferramenta que possui mais de uma aplicação, e uma ferramenta que possui uma aplicação que seja formada (por composição) por outras aplicações. Uma aplicação formada por outras aplicações engloba-as, ou seja, não é possível executar uma aplicação componente de forma isolada. Já uma ferramenta contendo diversas aplicações pode comandar a execução independente de cada uma delas.

Uma ferramenta deve conter também objetos que auxiliem a execução de tarefas modulares. Uma ferramenta simples deve conter no mínimo dois destes objetos: um leitor de comandos e um leitor de dados. O primeiro é o objeto responsável por administrar o envio de comandos (ou mensagens) do usuário para a ferramenta, o que pode ser feito de várias formas (via teclado, arquivo em lote, rede de comunicação, etc.). Na seção 7.1.2 foi apresentada uma estrutura de classes desenvolvida neste trabalho para a realização desta tarefa. O leitor de dados de SEE é responsável pela obtenção dos dados do sistema, e também pela criação dos elementos físicos em tempo de execução. Neste trabalho desenvolveu-se uma estrutura de classes baseada no padrão de projeto orientado a objetos Strategy, que encarrega-se desta atividade (apresentada na seção 7.1.2).

Quanto às funcionalidades que cada ferramenta deverá possuir, sob um aspecto mais genérico, considera-se a existência de um método que coordena a sua execução completa. Este método deve ser ativado no início da execução da ferramenta, o qual ge-

rencia o prosseguimento das atividades realizadas pelo objeto que representa a ferramenta, e todos os seus componentes.

Um Diagrama de Atividades simplificado mostra de maneira geral as atividades que uma ferramenta deve realizar, quando em execução. Conforme pode ser observado na Figura 7-7, basicamente uma ferramenta solicita comandos ao seu leitor de comandos, verifica qual é este comando, e o executa.

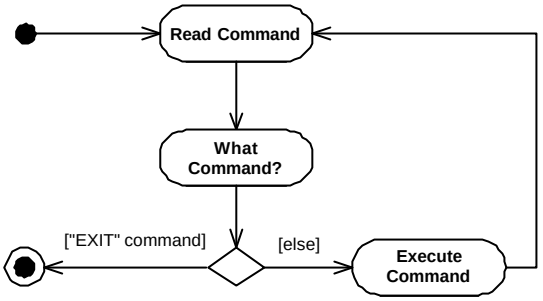


Figura 7-7 – Diagrama de Atividades Simplificado
para uma Ferramenta Computacional

É possível que uma ferramenta emita algum tipo de relatório, reportando alguma(s) informação(ões) a respeito da sua execução. Porém a maioria dos relatórios são gerados diretamente pelos objetos componentes. Por exemplo, cada aplicação gerencia a emissão de relatórios que lhe digam respeito.

A seguir é descrito um protótipo de ferramenta computacional, desenvolvido neste trabalho.

7.3. O Protótipo Implementado – *OOTPS*

Um protótipo de ferramenta computacional foi desenvolvido neste trabalho, 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*). Ele é representado com uma classe concreta, chamada *C_OOTPS*, a qual origina, em tempo de execução, um objeto - ferramenta, chamado *OOTPS*. Esta classe é apresentada na Figura 7-8.

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.

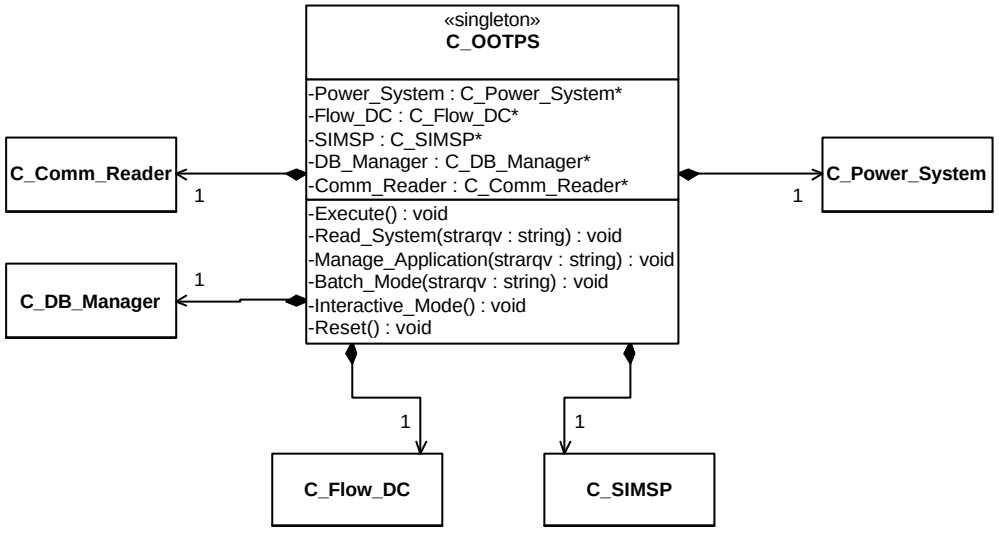


Figura 7-8 – Classe *C_OOTPS*

O protótipo possui duas aplicações: um fluxo de potência linearizado (classe C*_Flow_DC*) e uma simulação dinâmica com modelagem detalhada via método Alternado Implícito (classe *C_SIMSP*). Estas aplicações foram descritas no Capítulo 6.

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()*, que gerencia todo o processo de execução do protótipo. Conforme o diagrama apresentado na Figura 7-7, o método recebe comandos através do seu objeto *Comm_Reader*, identifica-os e executa-os.

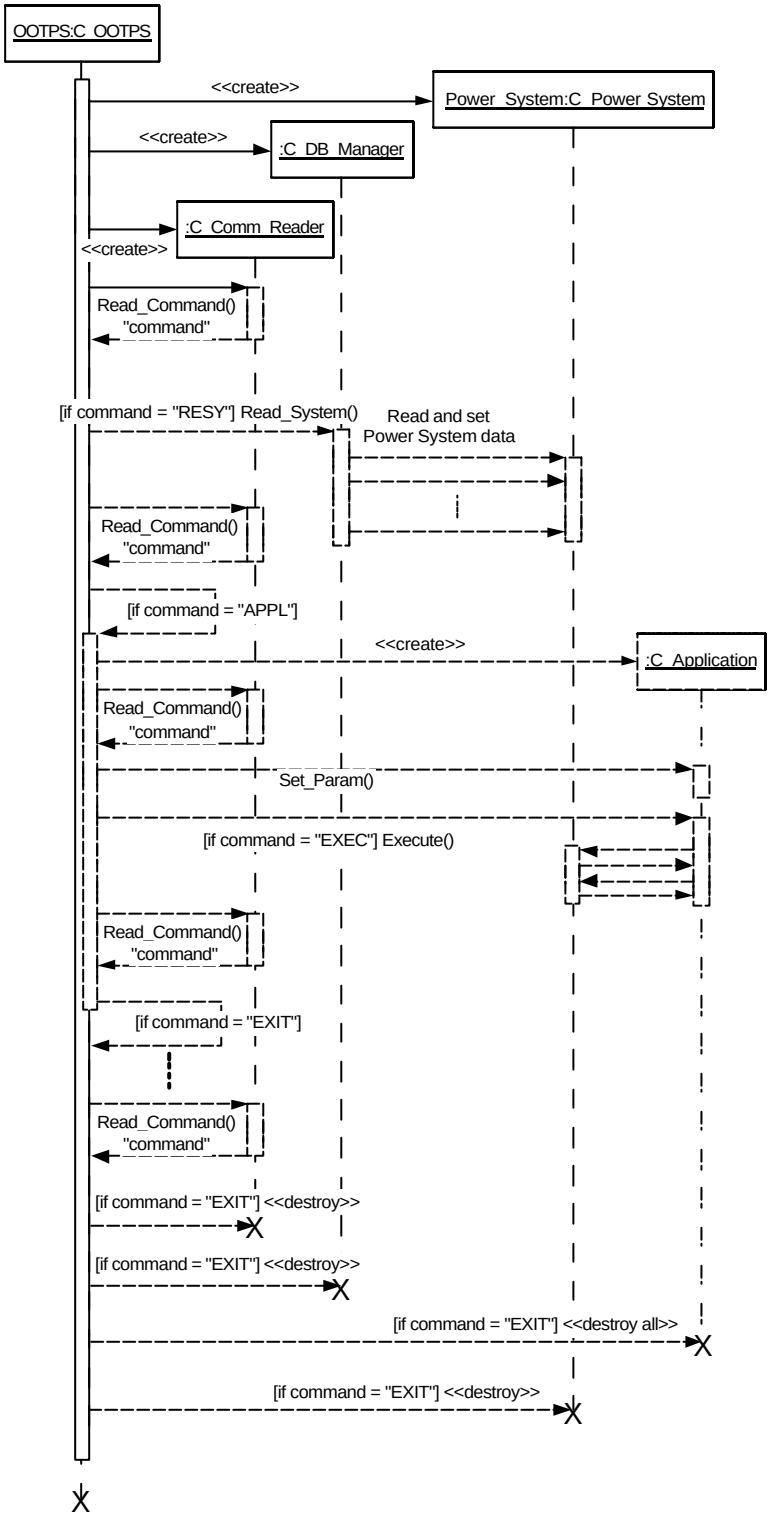


Figura 7-9 – Diagrama de Seqüência para o Processo de Execução Geral de uma Ferramenta Computacional

A Figura 7-9 apresenta um Diagrama de Seqüência para a ferramenta, mostrando uma seqüência básica de atividades do protótipo.

Depois de 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 (representado no diagrama pelo acrônimo "RESY"), a ferramenta solicita ao seu objeto *DB_Manager* que faça a leitura, de acordo com o tipo de arquivo de dados (informado na mesma linha de comando de "RESY"), e monte o objeto *Power_System*. Caso o comando seja para criar uma determinada aplicação (representado no diagrama pelo acrônimo "APPL"), o método *Manage_Applications()* é ativado. Este identifica qual a aplicação (que é informada na mesma linha do comando "APPL"), 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). Depois de 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.

O protótipo, assim como toda a base computacional desenvolvida neste trabalho de tese, foi implementado em linguagem de programação C++ (STROUSTRUP, 1997).

É importante observar que o protótipo aqui descrito não se caracteriza exatamente como uma ferramenta computacional final para o setor elétrico. Seu objetivo principal é ilustrar o desenvolvimento de ferramentas segundo a filosofia proposta neste trabalho, utilizando a base computacional aqui desenvolvida. Ferramentas completas para o apoio ao planejamento e à operação de SEE requerem um detalhamento maior no seu desenvolvimento, que certamente não estiveram presentes no projeto do protótipo. Porém, projetos de futuras ferramentas podem ser balizados pelas diretrizes aqui apresentadas, apenas detalhando-se questões mais específicas.

7.3.1. Resultados de Simulações

Algumas simulações foram realizadas utilizando o protótipo implementado, procurando-se avaliar os desempenhos das metodologias implementadas. Tanto os resultados numéricos quanto os desempenhos computacionais das aplicações foram avaliados.

Estes resultados são apresentados a seguir.

7.3.1.1. Aplicação 1: Fluxo de Potência Linearizado

Diversos SEE representando equivalentes do sistema interligado nacional brasileiro (SIN) foram analisados utilizando a aplicação de fluxo de potência linearizado, implementada no protótipo. Seus resultados numéricos (ângulos de tensão nas barras) foram comparados com resultados do programa ANAREDE (CEPEL, 1999a), software tradicional no setor elétrico brasileiro. Os valores obtidos foram exatamente os mesmos, desde sistemas de menor porte (equivalente do sistema sul brasileiro, com 45 barras), até sistemas mais complexos (equivalente do SIN, com 1916 barras), comprovando a eficiência da aplicação.

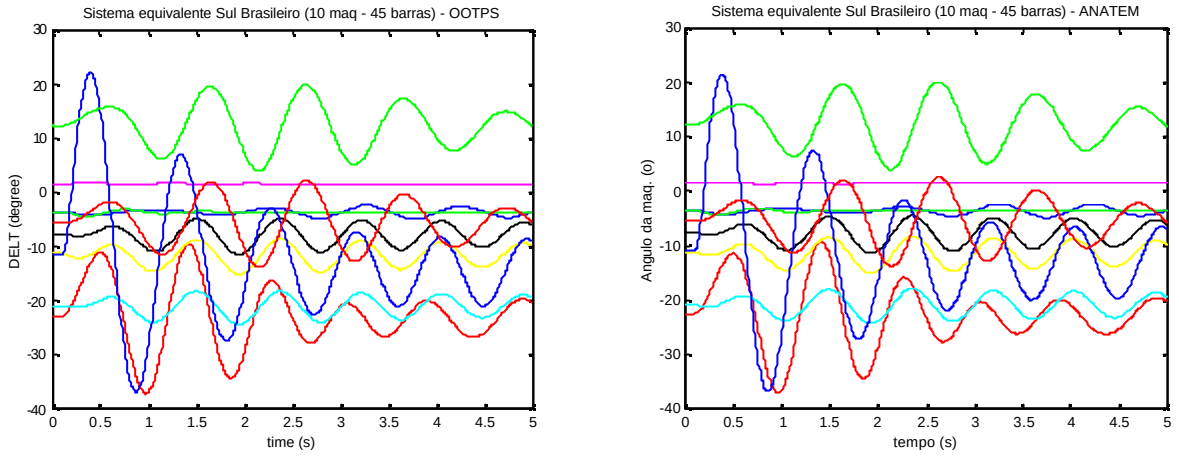
Em termos de desempenho computacional, não foi possível realizar um estudo detalhado, pois a aplicação de fluxo de potência linear exige pouco esforço computacional. Os tempos de execução da aplicação ficaram abaixo de um segundo, assim como no caso do programa ANAREDE, o que não permite comparações mais precisas. Entretanto, tal comparação é apresentada a seguir, para a aplicação de simulação dinâmica.

7.3.1.2. Aplicação 2: Simulação Dinâmica com Modelagem Detalhada

Da mesma forma que a aplicação anterior, diversos sistemas equivalentes do SIN foram simulados utilizando-se a aplicação de simulação dinâmica implementada. Utilizou-se para os estudos comparativos o Programa de Análise de Transitórios Eletromecânicos – ANATEM (CEPEL, 1999b), programa tradicional do setor elétrico brasileiro para a simulação do comportamento dinâmico de SEE.

Apresentam-se a seguir algumas curvas da simulação do sistema equivalente sul brasileiro, com 45 barras, 72 linhas e 10 máquinas. O estudo apresentado considerou a aplicação de um curto-circuito trifásico na barra 408 (Usina de Itaúba – 230kV) em 0,1 segundo de simulação, removendo-o 0,09 segundo depois. Simulou-se o sistema até 5 segundos. Nesta simulação utilizou-se o modelo 4 de máquina síncrona (efeitos subtran-

sitórios) para todas as máquinas, com regulador de tensão associado. As cargas foram modeladas por impedância constante. A Figura 7-10 mostra as curvas de ângulo do eixo q dos geradores, em graus, em relação a máquina de Segredo (barra 397). A Figura 7-10 (a) apresenta as curvas do protótipo *OOTPS*, enquanto a Figura 7-10 (b) apresenta as curvas de saída do programa ANATEM.



(a) Protótipo OOTPS (b) ANATEM

Figura 7-10 – Ângulos do Eixo q das Máquinas

Nota-se que os dois conjuntos de curvas apresentam os mesmos comportamentos dinâmicos, demonstrando a eficiência da aplicação implementada. Esta performance foi obtida também com outros sistemas testes, inclusive sistemas de maior porte.

Para comparações em termos de desempenho computacional utilizou-se um equivalente do SIN, com 1916 barras, 2788 linhas e 250 máquinas. Nesta simulação considerou-se a aplicação de um curto-circuito trifásico na barra 106 (Usina de Adrianópolis – 500kV) em 0,1 segundo de simulação, removendo-o 0,2 segundo depois, juntamente com a retirada da linha de 500kV 106-107 (Adrianópolis – Grajaú). Simulou-se o sistema até 10 segundos. Utilizou-se o modelo 4 de máquina síncrona (efeitos subtransitórios) para todas as máquinas, conectadas a um regulador de tensão. As cargas foram modeladas por impedância constante. Outros parâmetros, tais como tolerância para convergência das equações, passo de integração, freqüência de plotagem, etc. foram ajustados de maneira equivalente em ambos os programas. A plataforma computacional utilizada no teste foi um microcomputador com processador AMD Duron 1GHz, executando o sistema operacional Windows XP. O compilador utilizado para o protótipo foi o Microsoft Visual C++ 6.0, com suas opções padrões de otimização de códigos. Os tempos de simulação são mostrados na Tabela 7-1.

Tabela 7-1 – Tempos Computacionais

Protótipo	ANATEM
7,98 s	5,80 s

Observa-se que o protótipo consome um tempo aproximadamente 37% superior ao programa ANATEM. Este desempenho é considerado bom, demonstrando que tanto a base computacional proposta neste trabalho, quanto as aplicações implementadas apresentam desempenhos computacionais comparáveis a softwares tradicionais do setor elétrico.

Considerando-se o assunto “desempenho de softwares desenvolvidos sob o paradigma da MOO”, comentários sobre alguns aspectos são interessantes. Além daqueles já abordados no Capítulo 3, tais como a influência dos compiladores, avanços na área de hardware, etc., é correto afirmar que um programa orientado a objetos sempre consumirá mais tempo de processamento que programas projetados com base na funcionalidade dos sistemas sob estudo. Esta 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 não somente necessária como imprescindível para se manter a eficiência de uma estrutura de classes, em termos de aspectos como mantenabilidade, reusabilidade, clareza de projeto, etc. Apesar da evolução natural dos compiladores, visando a diminuição dos tempos necessários para as trocas de mensagens, gerando códigos executáveis computacionalmente mais eficientes, estas operações sempre consumirão algum tempo de processamento.

O objetivo central em um projeto de software orientado a objetos é a busca de eficiência em termos mantenabilidade e reusabilidade. Contudo, conforme já abordado no Capítulo 3, uma nova ferramenta computacional desenvolvida sob novos paradigmas não pode apresentar desempenho computacional muito aquém daqueles obtidos por ferramentas já existentes. Neste aspecto, consideram-se bons os resultados obtidos com a base computacional desenvolvida e com o protótipo implementado neste trabalho, pois seu desempenho é da mesma ordem de grandeza que o de ferramentas tradicionalmente utilizadas pelo setor elétrico.

CAPÍTULO 8

8. CONCLUSÕES

Conforme descrito no capítulo introdutório, o presente trabalho trata de um tema amplo, que é a aplicação de Modelagem Orientada a Objetos (MOO) na área de Sistemas de Energia Elétrica (SEE). Especificamente, este trabalho propôs uma nova filosofia de projeto e implementação de software para SEE, juntamente com a concepção de uma base computacional para a representação destes sistemas, na forma de estruturas de classes orientadas a objetos. As mais diversas instâncias de um SEE foram representadas segundo estruturas hierárquicas de classes, desde os elementos físicos do sistema, até suas metodologias de aplicação e ferramentas computacionais finais para apoio ao planejamento e à operação de SEE. Classes foram desenvolvidas formando pacotes e bibliotecas, encapsulando atividades essencialmente computacionais, e facilitando-se com isso o desenvolvimento de metodologias de aplicação e ferramentas finais.

O uso de técnicas modernas de engenharia de software, tal como a MOO, permite uma abordagem sistemática do problema de desenvolvimento de software para sistemas elétricos. A filosofia proposta neste trabalho dá suporte ao desenvolvimento de um novo conjunto ferramentas computacionais para o setor elétrico, permitindo uma representação efetiva das mais diversas instâncias de um SEE.

As duas aplicações projetadas, e implementadas sob a forma de um protótipo de ferramenta, demonstraram a capacidade da base computacional proposta em permitir a incorporação das mais diversas metodologias de análise e síntese, sob um ambiente integrado, e usufruindo as características da MOO. Esta base orientada a objetos mostrou-se flexível e de fácil entendimento e utilização, contendo em sua documentação diversos diagramas em notação UML. Suas estruturas de dados são robustas, “resistentes” a alte-

rações e atualizações nos códigos. A utilização de padrões de projeto orientados a objeto acrescentou modularidade e expansibilidade à base computacional, facilitando a sua reutilização. Comparações em termos de resultados numéricos e desempenho computacional do protótipo foram apresentadas, demonstrando-se a eficiência tanto da base computacional proposta, quanto das aplicações implementadas.

A seguir as contribuições principais do trabalho são apresentadas e descritas.

8.1. Contribuições Principais do Trabalho

- **Proposição da Nova Filosofia**

A principal contribuição deste trabalho de tese é a proposição de uma nova filosofia para o desenvolvimento de softwares para o setor elétrico, fundamentada em preceitos da MOO. Esta filosofia engloba desde aspectos gerais, tais como uma delimitação adequada do escopo dos projetos de desenvolvimento de software para SEE, e uma separação de conceitos através de diferentes abstrações do sistema, até questões específicas de implementação de códigos, passando por princípios como mantenabilidade, expansibilidade e robustez das estruturas de dados (com conseqüente redução de custos nessas tarefas), reutilização de códigos, eficiência computacional, etc.

Suas características principais são:

→ baseada no paradigma da MOO;

→ enfoque principal nas estruturas de dados, reproduzindo o comportamento real do sistema físico;

→ atenção especial ao encapsulamento de abstrações específicas do problema e suas interfaces com o restante do domínio da aplicação, sem prejuízo do desempenho computacional das ferramentas finais;

→ reutilização de códigos nucleares, encapsulados adequadamente;

→ propostas efetivas para a representação de elementos físicos e seus diversos comportamentos (interfaces funcionais);

→ documentação adequada do projeto (UML) e uso de padrões de projeto OO.

- **Desenvolvimento de uma Base Computacional Orientada a Objetos para SEE**

Uma base computacional orientada a objetos para a representação de SEE foi desenvolvida. Esta base servirá como núcleo para futuros desenvolvimentos no âmbito do projeto, e portanto possui mecanismos que permitem futuras expansões, no intuito de se representar as mais diversas instâncias dos SEE. Suas estruturas de classes prevêem a acomodação não somente de elementos físicos que formam o sistema, mas também conceitos abstratos, tais como metodologias de análise e síntese aplicadas a SEE, ferramentas computacionais de apoio ao planejamento e à operação destes sistemas, e também conceitos computacionais e matemáticos, envolvidos na construção destas ferramentas. Todas estas instâncias foram contempladas na construção da base computacional, sempre se mantendo uma clara separação dos conceitos envolvidos (na forma de diferentes abstrações).

As características principais desta base são:

→ Modelagem de Elementos Físicos Estruturais e de Composição: Os elementos físicos do sistema foram classificados em dois grandes grupos: elementos estruturais e elementos de composição. O primeiro grupo representa elementos conectados diretamente às barras do sistema, formando, assim, a sua estrutura básica. O segundo grupo representa elementos que não se conectam diretamente a alguma barra, mas que quando agrupados de uma certa maneira, dão origem a elementos estruturais.

→ Modelagem das Funcionalidades dos Elementos Físicos: As funcionalidades dos elementos físicos, as quais são dependentes da metodologia de análise ou síntese que se está aplicando sobre o sistema, foram encapsuladas em classes paralelas às classes representativas dos elementos físicos, denominadas interfaces funcionais. Dessa forma isolam-se as características puramente físicas dos elementos, válidas em qualquer aplicação considerada, de características comportamentais, que dizem respeito a uma ou outra metodologia de análise ou síntese em específico. Utilizou-se para a montagem deste mecanismo o padrão de projeto orientado a objetos Adapter. Com isso, facilitam-se expansões futuras das estruturas, permitindo-se que novas metodologias e novos elementos sejam facilmente incorporados às estruturas da base computacional.

- **Modelagem de Metodologias e Implementação de um Protótipo de Ferramenta Computacional**

As metodologias de análise e síntese, ou aplicações, foram também organizadas segundo estruturas de classes, as quais conectam-se às estruturas representativas dos elementos físicos do sistema elétrico, atuando sobre elas. Duas aplicações foram projetadas utilizando-se a base computacional desenvolvida, e implementadas em linguagem de programação C++. Estas aplicações, fluxo de potência linearizado e simulação da dinâmica com modelagem detalhada, em conjunto com as facilidades computacionais, foram agrupadas em um protótipo de ferramenta computacional (*OOTPS*), o qual exemplifica a construção e o funcionamento de aplicações para SEE operando sob um ambiente integrado, e desenvolvidas sob a filosofia de projeto de software proposta neste trabalho.

- **Modelagem de Facilidades Computacionais**

Juntamente com as estruturas de classes relativas às mais diversas instâncias de um SEE, classes representando facilidades computacionais foram projetadas e implementadas, sob a forma de pacotes e bibliotecas. Dentre estas figuram classes para leitura e armazenamento de dados, gerenciamento de telas, tomadas de tempo de execução, etc. Destaca-se a estrutura *SparseMatrix*, para o armazenamento e tratamento de matrizes e sistemas lineares esparsos de grande porte. A estrutura permite que os mais diversos métodos de solução de sistemas lineares possam ser utilizados, encapsulados na forma de estratégias de solução (utilizou-se aí o padrão de projeto Strategy). O conceito de reutilização de códigos foi explorado na concepção da estrutura, permitindo que bibliotecas já desenvolvidas e exaustivamente testadas, com desempenho numérico e computacional comprovados, e mesmo que projetadas e codificadas sob paradigmas que não a MOO, sejam incorporadas às metodologias e ferramentas computacionais desenvolvidas sob a filosofia proposta neste trabalho.

- **Formalização Quanto a Aplicação de MOO em SEE**

Diferentemente de outros trabalhos científicos disponíveis na literatura a respeito do tema, o presente trabalho de tese buscou formalizar a aplicação do paradigma da MOO na modelagem de SEE. Para isso, utilizou-se uma notação gráfica padrão para a representação das estruturas e procedimentos desenvolvidos (UML). Também se procurou dedicar atenção especial aos preceitos gerais da MOO, independentemente de um

método de projeto ou outro, fazendo com que as premissas utilizadas na proposição da filosofia apresentada neste trabalho estejam tanto quanto possível em acordo com o paradigma em si, facilitando sua (re)utilização futura.

8.2. Sugestões para Trabalhos Futuros

As sugestões para trabalhos futuros são agrupadas em dois conjuntos: trabalhos específicos e trabalhos gerais. A seguir cada grupo é apresentado.

8.2.1. Trabalhos Específicos

Devido ao caráter científico do presente trabalho, e também à abrangência do tema, e restrições de tempo na realização do mesmo, alguns itens específicos nas classes das estruturas desenvolvidas não foram abordados, adiando-se estes desenvolvimentos e sugerindo-os como trabalhos futuros. Abaixo se listam estes trabalhos:

♦ Melhorias em nível de códigos-fonte: A base computacional desenvolvida constitui-se de estruturas de classes, concretizadas na forma de diagramas e códigos-fonte; quanto a esta segunda forma, um melhor tratamento pode ser dedicado a estes códigos, visando uma melhor separação dos conceitos envolvidos na modelagem computacional dos SEE, e conseqüentemente facilitando-se sua utilização efetiva e futuras expansões.

♦ Melhorias nas funcionalidades dos elementos físicos: O uso das interfaces funcionais depende diretamente do projeto específico de cada metodologia de análise e síntese, ficando a cargo do projetista, conforme suas prioridades, a forma com que as interfaces colaborarão no funcionamento da aplicação, não trazendo reflexos no restante das estruturas; sob este aspecto, as interfaces funcionais dos elementos físicos para cada uma das duas aplicações consideradas neste trabalho podem ser melhoradas, visando uma participação mais efetiva destas classes em cada aplicação.

♦ Melhorias na estrutura *SparseMatrix*: Algumas melhorias podem ser realizadas na estrutura *SparseMatrix* desenvolvida neste trabalho, tais como a implementação de operações matemáticas fundamentais (adição e multiplicação, entre matrizes ou entre matrizes e vetores), e de funções matemáticas específicas para tratamento de matrizes, tais como inversão, transposição, cálculo de condicionamento, etc.

♦ Gerenciamento de telas: No âmbito das facilidades computacionais, a(s) classe(s) para o gerenciamento de telas deve(m) ser desenvolvida(s), facilitando a implementação de ferramentas computacionais finais para o setor elétrico.

8.2.2. Trabalhos Gerais

Diversos trabalhos de caráter mais amplo podem ser identificados, caracterizando novas contribuições referentes ao tema principal enfocado no presente trabalho de tese. Os principais são listados a seguir:

♦ Especialização da base computacional: Novos elementos, tais como dispositivos FACTS, podem ser modelados nas estruturas de classes aqui propostas, incrementando-se a base computacional disponível para o desenvolvimento integrado de ferramentas para o setor elétrico.

♦ Projeto e implementação de outras metodologias: Outras metodologias de análise e síntese (fluxos de potência não-linear e ótimo, análise linear da dinâmica, análise de curto-circuito, métodos para o projeto e o ajuste de controladores, etc.) podem ser projetadas segundo a proposta deste trabalho, visando a construção de ferramentas computacionais.

♦ Desenvolvimento de um banco de dados orientado a objetos para SEE: O desenvolvimento de um banco de dados orientado a objetos para SEE é importante para se fazer o melhor uso possível das características da MOO; atualmente obtém-se os dados dos SEE a partir de arquivos no formato ASCII (padrão POWERMOD, ou PECO), sendo que novas estratégias de leitura de dados estão previstas, e deverão ser encapsuladas na estrutura através do padrão Strategy.

♦ Desenvolvimento de ferramentas computacionais para o apoio ao planejamento e à operação dos SEE: Finalmente, novas ferramentas computacionais deverão ser desenvolvidas segundo a filosofia proposta neste trabalho, utilizando suas estruturas de classes. Estas novas ferramentas representarão um novo paradigma para o desenvolvimento de software para o setor elétrico, caracterizado por menores custos de manutenção e atualização dos códigos, maior nível de integração entre módulos, facilidades no desenvolvimento em equipes, etc.

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.; FERREIRA, J. M. F.; DECKER, I. C. et al.; 2002a. 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.

AGOSTINI, M. N.; DECKER, I. C.; SILVA, A. S.; 2002b. Desenvolvimento e Implementação de uma Base Computacional Orientada a Objetos para Aplicações em Sistemas de Energia Elétrica. *Revista Controle & Automação*, Campinas, v. 13, n. 2, p. 181-189.

AGOSTINI, M. N.; DECKER, I. C.; SILVA, A. S.; 2002c. Discussion of: Design of Generic Direct Sparse Linear System Solver in C++ for Power System Analysis. *IEEE Transactions on Power Systems*, New York, v. 17, n. 3 (Aug.), p. 926-927.

AGOSTINI, M. N.; DECKER, I. C.; SILVA, A. S.; 2002d. Nova Filosofia para o Projeto de Softwares na Área de Sistemas de Energia Elétrica Utilizando Modelagem Orientada A Objetos. In. CONGRESSO BRASILEIRO DE AUTOMÁTICA – CBA (14. : Set. : Natal, RN). *Anais*. Natal. p. 1555-1561.

AGOSTINI, M. N.; SOUZA, A.; SILVA, A. S.; DECKER, I. C.; 2002e. Simulação Dinâmica de Sistemas de Energia Elétrica Utilizando o PSpice. In. CONGRESSO BRASILEIRO DE AUTOMÁTICA – CBA (14. : Set. : Natal, RN). *Anais*. Natal. p. 1200-1205.

ALVARADO, F. L.; 1996. Methods for the Quantification of Ancillary Services in Electric Power Systems. In: SIMPÓSIO DE ESPECIALISTAS EM PLANEJAMENTO DA OPERAÇÃO E EXPANSÃO ELÉTRICA (5. : Maio : Recife, PE). *Anais*. Recife. p. 27-49.

AMBLER, S. W.; 1998. *Análise e Projeto Orientados a Objeto* : Seu Guia para Desenvolver Sistemas Robustos com Tecnologia de Objetos. v. 2. Rio de Janeiro : Infobook.

ARAÚJO, D. L.; REIS, F. G.; 2000. *Desenvolvendo com Java 2 Para Iniciantes*. Rio de Janeiro : Book Express.

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.

ARCINIEGAS, F.; 2002. *C++ XML*. São Paulo : Pearson Education.

ARNHOLM, C. A.; 1997. *Mixed Language Programming Using C++ and Fortran 77* : A Portable Technique for Windows NT/95, UNIX and Other Systems. Version 1.1 (28-May-1997). Disponível on-line em: http:// home.online.no /~arnholm/ cppf77.zip

ASHCRAFT, C.; PIERCE, D.; WAH, D. et al.; 1999. *The Reference Manual for SPOOLES, Release 2.2* : An Object Oriented Software Library for Solving Sparse Linear Systems of Equations. Disponível on-line em: http://www.netlib.org/linalg/spooles/ ReferenceManual.ps.gz.

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.

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.

BOOCH, G.; 1994. *Object-Oriented Analysis and Design* : With Applications. 2. ed. Reading : Addison-Wesley.

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.

BOSSHART, P.; BACHER, R.; 1998. A Domain Architecture for Solving Simultaneous Nonlinear Network Equations. *IEEE Transactions on Power Systems*, New York, v. 13, n. 3 (Aug.), p. 1006-1012.

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; 1999a. *Programa de Análise de Redes – ANAREDE* : Manual do Usuário. Rio de Janeiro, RJ.

CEPEL - Centro de Pesquisas de Energia Elétrica; 1999b. *Programa de Análise de Transitórios Eletromecânicos – ANATEM* : Manual do Usuário. Rio de Janeiro, RJ.

COAD, P.; NORTH, D.; MAYFIELD, M.; 1995. *Object Models* : Strategies, Patterns, and Applications. New Jersey : Prentice Hall.

COLEMAN, D.; ARNOLD, P.; BODOFF, S. et al.; 1996. *Desenvolvimento Orientado a Objetos* : O Método Fusion. Rio de Janeiro : Campus.

DE TUGLIE, E.; DICORATO, M.; LA SCALA, M. et al.; 1999. Dynamic Security Preventive Control in a Deregulated Electricity Market. In: POWER SYSTEMS COMPUTATION CONFERENCE - PSCC (13. : Jun. 1999 : Trondheim). *Proceedings*. Trondheim, 1999. p. 125-131.

ELLIS, M. A.; STROUSTRUP, B.; 1990. *The Annotated C++ Reference Manual.* Reading : Addison-Wesley.

ERIKSSON, H. E.; PENKER, M.; 1998. *UML Toolkit.* New York : Wiley.

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 (May), 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 (Feb.), p. 97-104.

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

FOWLER, M.; 2000. *UML Essencial* : Um Breve Guia para a Linguagem-Padrão de Modelagem de Objetos. 2. ed. Porto Alegre : Bookman.

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.

GAING, Z. L.; LU, C. N.; CHANG, B. S. et al.; 1996. An Object-Oriented Approach for Implementing Power System Restoration Package. *IEEE Transactions on Power Systems*, New York, v. 11, n. 1 (Feb.), p. 483-489.

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.

INTHURN, C.; 2001. *Qualidade e Teste de Software*. Florianópolis : Bookstore.

LARMAN, C.; 2000. *Utilizando UML e Padrões* : Uma Introdução à Análise e ao Projeto Orientados a Objetos. Porto Alegre : Bookman.

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.

LI, S.; SHAHIDEHPOUR, S. M.; 1993. An Object Oriented Power System Graphics Package for Personal Computer Environment. *IEEE Transactions on Power Systems*, New York, v. 8, n. 3 (Aug.), p. 1054-1060.

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.

MOROZOWSKI FILHO, M.; 1981. *Matrizes Esparsas em Redes de Potência* : Técnicas de Operação. Rio de Janeiro : Livros Técnicos e Científicos : Eletrobrás; Florianópolis : Fundação do Ensino da Engenharia em Santa Catarina.

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.

PETERS, J. F.; PEDRYCZ, W.; 2001. *Engenharia de Software* : Teoria e Prática. Rio de Janeiro : Campus.

QUATRANI, T.; 1998. *Visual Modeling with Rational Rose and UML*. Reading : Addison-Wesley.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W. et al.; 1994. *Modelagem e Projetos Baseados em Objetos*. Rio de Janeiro : Campus.

SCHILDT, H.; 1992. *Turbo C++* : Guia do Usuário. São Paulo : Makron Books.

SHIRMOHAMMADI, D.; VOJDANI, A.; 1996. An Overview of Ancillary Services. In: SIMPÓSIO DE ESPECIALISTAS EM PLANEJAMENTO DA OPERAÇÃO E EXPANSÃO ELÉTRICA (5. : Maio : Recife, PE). *Anais*. Recife. p. 1-9.

SILVA, E. L.; 2001. *Formação de Preços em Mercados de Energia Elétrica*. 1. ed. Porto Alegre, RS : Editora Sagra Luzzatto.

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.

SOUZA, A.; 1999. *Avaliação da Segurança Dinâmica Usando Modelos Detalhados e Processamento Distribuído*. Florianópolis. Dissertação (Mestrado em Engenharia Elétrica) - Centro Tecnológico, Universidade Federal de Santa Catarina.

SOUZA, A.; DECKER, I. C.; AGOSTINI, M. N.; et al.; 2002. Sistema Computacional Baseado em Cluster de Microcomputadores para Avaliação e Melhoria da Segurança Dinâmica On-Line. In. SYMPOSIUM OF SPECIALISTS IN ELECTRIC OPERATIONAL AND EXPANSION PLANNING - SEPOPE (8. : Maio : Brasília, DF). *Anais*. Brasília.

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

VANTI, M. R. V.; 1994. *Implementação Orientada para Objeto de um Simulador para Dinâmica Lenta de um Sistema de Energia Elétrica*. Florianópolis. Dissertação (Mestrado em Engenharia Elétrica) - Centro Tecnológico, Universidade Federal de Santa Catarina.

WEINBERG, G. M.; 1993. *Software com Qualidade* : Pensando e Idealizando Sistemas. São Paulo : Makron Books.

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.

ZHU, J.; LUBKEMAN, D. L.; 1997. Object-Oriented Development of Software Systems for Power System Simulations. *IEEE Transactions on Power Systems*, New York, v. 12, n. 2 (May), p. 1002-1007.

ZHU, J.; JOSSMAN, P.; 1999. Application of Design Patterns for Object-Oriented Modeling of Power Systems. *IEEE Transactions on Power Systems*, New York, v. 14, n. 2 (May), p. 532-537.

ZOLLENKOPF, K.; 1971. Bi-Factorization - Basic Computational Algorithm and Programming Techniques. In: Large Sparse Sets of Linear Equations, edited by J. K. Reid, Academic Press.