FEDERAL DE ENGENHARIA DE ITAJUBÁ

Um Modelo para um Mercado Virtual Atacadista de Energia

ROGER DANIEL FRANCISCO FERREIRA

ITAJUBÁ - Fevereiro
2002

ESCOLA FEDERAL DE ENGENHARIA DE ITAJUBÁ

Programa de Pós-Graduação em Engenharia Elétrica

UM MODELO PARA UM MERCADO VIRTUAL ATACADISTA DE ENERGIA

Dissertação apresentada à Escola Federal de Engenharia de Itajubá, para obtenção do título de Mestre em Engenharia Elétrica.

ROGER DANIEL FRANCISCO FERREIRA

Itajubá, Fevereiro de 2002

ROGER DANIEL FRANCISCO FERREIRA

UM MODELO PARA UM MERCADO VIRTUAL ATACADISTA DE ENERGIA

Dissertação apresentada à Escola Federal de Engenharia de Itajubá, para obtenção do título de Mestre em Engenharia Elétrica.

Área de concentração:
Automação e Sistemas Elétricos Industriais

Orientador:
Germano Lambert Torres

Co-orientador:
Fábio Gavião

Itajubá, Fevereiro de 2002

A minha noiva e
minha família por todo
incentivo, amor e carinho.

Agradecimentos

Aos professores Germano Lambert Torres e Fábio Gavião pela orientação, dedicação e amizade que resultaram neste trabalho.

À todos, que direta ou indiretamente, contribuíram para a realização desse trabalho.

À Deus que sempre nos protege e nos guia.

Sumário

RESUMO

Este trabalho busca introduzir, essencialmente, a aplicação de novas tecnologias de software na engenharia. Por se tratar de um modelo fundamentalmente distribuído, composto por diversos elementos, o MAE foi escolhido como cenário ideal para o emprego destas novas tecnologias de software.

Particularmente para esta aplicação, optou-se pela utilização de uma tecnologia baseada em sistemas distribuídos, empregada em diversos ramos da indústria, comércio e empresas de uma maneira geral. Agregando-se o poder da programação orientada a objetos com a consolidação das redes de computadores, será apresentada nesta dissertação uma solução conhecida como Objetos Distribuídos, que viabiliza o rápido desenvolvimento de complexos sistemas naturalmente distribuídos, como é o caso do ambiente do MAE. Mais especificamente, o modelo de programação sugerido é o da OMG e seu padrão CORBA.

Complementarmente para o projeto do mercado virtual, será apresentado o Processo Unificado, usado basicamente para auxiliar e facilitar o processo de desenvolvimento de sistemas de software grandes e complexos.

ABSTRACT

This work deals, essentially, with the use of new software technologies in the engineering field. The scenario chosen to show the usage of these new technologies was the MAE – the Brazilian new Energy Wholesale Market, because it has many different kinds of components that characterize a real distributed system. We then present a suggestion of software architecture to support the needs demanded for the new MAE system.

The ideas for this application, specifically, will use a Distributed System technology already employed in many segments of industry, commerce and companies in general. Known as Distributed Objects, this technology, allied with the power of the Object Oriented Programming, allows the development of complex systems intrinsically distributed, such as the MAE system environment. More specifically, the programming model suggested is the CORBA standard, from OMG.

In addition to the virtual market design, it will be presented the Unified Process, used basically to support the development process of large and complex software systems.

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

A partir da reestruturação sofrida nos últimos anos pelo mercado brasileiro de energia elétrica, com a criação de novos agentes que atuam em conjunto com o MAE - Mercado Atacadista de Energia, para formar o novo modelo do mercado de energia elétrica brasileiro, a comercialização de energia elétrica passa a ter um caráter muito mais dinâmico.

Para se adaptar às essas novas características de comercialização da energia elétrica, torna-se importante a integração desses agentes para possibilitar a troca de informações entre os mesmos. Contudo, para assegurar a perfeita operação do MAE, é necessário criar um sistema eficiente e seguro.

A tecnologia de Objetos Distribuídos abordada neste trabalho, baseada na arquitetura proposta pelo OMG, ou seja, o padrão CORBA, em função de suas próprias características e utilização, destaca-se, a nosso ver, como uma solução apropriada para o desenvolvimento de um sistema computacional distribuído que atenda os requisitos funcionais e de segurança do ambiente do MAE.

O modelo de sistema distribuído proposto concentra-se essencialmente no emprego da tecnologia que suporta o padrão CORBA, considerando somente algumas idéias básicas do atual modelo de mercado de energia elétrica.

1.2 OBJETIVOS

Este trabalho tem como objetivo principal apresentar tecnologias de software modernas que podem ser usadas para a solução do problema de integração dos agentes do MAE em um mercado virtual.

Complementarmente, serão apresentadas também técnicas e ferramentas que auxiliam o desenvolvimento de sistemas de software complexos.

Como tecnologia de software, será abordado o uso da tecnologia de Objetos Distribuídos como solução para o desenvolvimento de sistemas distribuídos. O padrão CORBA, mais especificamente, será o modelo de objetos distribuídos de interesse.

A técnica de desenvolvimento usada será o Processo Unificado, que utiliza a Linguagem de Modelagem Unificada (UML) para a criação dos modelos. A ferramenta empregada para auxiliar o processo de desenvolvimento e documentação do projeto do sistema de software será o UML Suite 4.5 da Telelogic.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

A divisão deste trabalho é feita em cinco capítulos, conforme as sínteses a seguir.

- *Capítulo 2*

  O capítulo 2, "Computação Distribuída", faz uma breve introdução dos principais conceitos de software que redundaram no surgimento da tecnologia de Objetos Distribuídos.

  O capítulo começa com a descrição dos objetivos, vantagens e desvantagens dos sistemas distribuídos, apresentando as características que um projeto de tal sistema deve atender para ser bem sucedido.

  Em seguida, é feita uma apresentação que destaca as características de comunicação entre computadores, começando com os protocolos de comunicação, passando pelo modelo cliente-servidor, a API de soquete, o conceito de RPC, até chegar no conceito de *middleware*. Alguns tópicos de segurança são apresentados também.

  O capítulo continua com a apresentação da Orientação a Objetos, destacando os pontos mais relevantes. O resultado final é o advento da tecnologia de Objetos Distribuídos, apresentada por último.

- *Capítulo 3*

  O capítulo 3, "Padrão CORBA", é um compilado do modelo de Objetos Distribuídos CORBA, da OMG.

O capítulo inicia destacando os conceitos e características gerais da solução proposta pela OMG, descrevendo em seguida o mecanismo que permite a comunicação entre os objetos distribuídos em um sistema distribuído. O modelo é apresentado na seqüência.

O capítulo termina com a apresentação dos serviços de repositório e serviços padronizados, que permitem desenvolver aplicativos CORBA mais eficientes e completos para a solução de sistemas distribuídos complexos.

- *Capítulo 4*

O capítulo 4, "Modelo de Simulação", apresenta as linhas gerais para o projeto do modelo de simulação do MAE.

O capítulo começa com a apresentação do novo modelo de mercado de energia elétrica brasileiro, destacando seus agentes e sua operação.

O modelo de simulação, propriamente dito, é detalhado na seqüência, com a definição dos agentes considerados, a descrição do seu funcionamento, a descrição do Sistema Elétrico usado na simulação e a apresentação do algoritmo de programação linear usado para a solução do despacho ótimo do sistema elétrico considerado. As características do sistema distribuído são apresentadas no fim, destacando as plataformas e linguagens de programação que serão usadas no projeto do modelo de simulação.

- *Capítulo 5*

O capítulo 5, "Processo de Desenvolvimento de Software OO", apresenta o Processo Unificado como solução usada para o projeto do modelo de simulação do MAE.

O capítulo inicia com uma breve apresentação da estrutura do Processo Unificado. A descrição do processo contempla as etapas de captura de requisitos, análise, projeto e implementação. A arquitetura resultante de cada etapa é apresentada numa seção à parte. O sistema resultante do projeto do modelo de simulação do MAE é apresentado no final do capítulo, enfatizando algumas considerações iniciais referentes à implementação e destacando as interfaces gráficas dos aplicativos desenvolvidos.

- *Capítulo 6*

  O capítulo 6, "Conclusão", apresenta as conclusões do trabalho desenvolvido e as sugestões de trabalhos futuros que podem orientar novas pesquisas que podem dar continuidade aos resultados obtidos nessa dissertação.

CAPÍTULO 2

COMPUTAÇÃO DISTRIBUÍDA

2.1 INTRODUÇÃO

A solução para o projeto de grandes aplicações continua sendo uma árdua tarefa para desenvolvedores de sistemas de software. Apesar da existência de diversas tecnologias, o conhecimento para o emprego das melhores técnicas, bem como dos melhores procedimentos para o desenvolvimento de projetos, representam ainda fatores de grande discussão e dúvida. Em função da própria diversidade de tecnologias, como as diferentes plataformas (Sistema Operacional e Hardware) [1] que compõem um ambiente totalmente heterogêneo, os projetos de sistemas de software se tornaram ainda mais complexos.

Aliado aos requisitos de alta confiabilidade e alto desempenho exigidos pelas novas aplicações, o desenvolvimento dos sistemas modernos de software exige rapidez ao menor custo possível, consequência direta das próprias necessidades de mercado.

Na busca de uma solução para a complexidade envolvida no desenvolvimento de tais sistemas, pensou-se num conceito já existente e que parecia ser promissor para a criação de software, que era o conceito da Orientação a Objetos (OO), gerado no final da década de 60, início de 70, e aperfeiçoado por diversos autores, entre eles, Grady Booch.

Baseando-se nos próprios objetos existentes no mundo real com os quais devemos lidar, a OO fornece um modelo mais adequado com técnicas que facilitam e melhoram o controle da complexidade dos sistemas e a estruturação de seus componentes.

Assim como os objetos do mundo real, os objetos da computação estão distribuídos em redes e entre redes. Surgiu então o conceito de Objetos Distribuídos (OD), que são unidades de código distribuídas em diversas máquinas, comunicando-se apenas através da troca de mensagens. Essas unidades são objetos desenvolvidos de acordo com a filosofia de OO que agregam um novo componente que os difere dos objetos convencionais, tornando-os distribuídos.

Além de se valer do poder da Orientação a Objetos, como a reutilização de objetos, os Objetos Distribuídos possibilitam ainda a integração de ambientes heterogêneos, grandes responsáveis pela complexidade no desenvolvimento de projetos.

2.2 SISTEMAS DISTRIBUÍDOS

No início da era da computação (ano de 1945), os computadores existentes eram grandes e caros. Mesmo com o advento dos minicomputadores, o custo de uma máquina ainda era muito alto, o que impedia a compra de várias unidades pelas organizações.

A partir do avanço de duas tecnologias, a situação mudou drasticamente. Primeiro foi o desenvolvimento de poderosos microprocessadores, que passaram a ter o mesmo poder de computação dos antigos computadores de grande porte *(mainframes)*, só que a um custo bem inferior.

Em seguida, o surgimento de redes locais de alta velocidade (LANs) permitiu que dezenas e mesmo centenas de máquinas pudessem ser conectadas para troca de informações.

Dessa forma, os sistemas antes centralizados, passaram a ser conectados de forma a obter o que se denomina de Sistemas Distribuídos.

Entretanto, resta um ponto importante: software. Apesar dos avanços obtidos, existe muita coisa a ser feita ainda para que as exigências dos Sistemas Distribuídos sejam satisfeitas. A evolução de tecnologias de software, como a de Objetos Distribuídos, representa um avanço importante rumo à concretização de um Sistema Distribuído.

2.2.1 VANTAGENS E DESVANTAGENS

Como toda tecnologia, os Sistemas Distribuídos possuem vantagens e desvantagens.

Entretanto, será possível observar que os Sistemas Distribuídos têm muito mais a oferecer para a solução de diversos problemas da indústria e de organizações em geral do que barreiras. A própria evolução dos softwares, dos sistemas de segurança e o desenvolvimento de projetos de redes cada vez mais velozes e seguras são exemplos que poderão mitigar os empecilhos existentes.

Comparando-se com os Sistemas Centralizados, pode-se destacar as seguintes vantagens:

- ***Relação custo/benefício:*** considerando-se o poder de processamento de centenas de máquinas comuns operando cooperativamente através de uma rede, não existiria, mesmo teoricamente, a possibilidade de construção de um único *mainframe* com tal capacidade. Mesmo se houvesse tal possibilidade, os custos de construção, operacionalização e manutenção não justificariam o investimento.
- ***Velocidade:*** o poder de processamento total da rede pode superar o poder de um *mainframe*.
- ***Distribuição Inerente:*** alguns sistemas são intrinsecamente distribuídos, pois apresentam diversas máquinas, como sistemas bancários e sistemas de automação industrial.
- ***Confiabilidade:*** a falha de uma ou mesmo algumas máquinas em um Sistema Distribuído bem projetado não resulta em uma perda de *performance* considerável, pois o sistema como um todo continua operando. No caso de um *mainframe*, uma falha no mesmo significaria uma pane total no sistema, o que é inadmissível, principalmente no caso de aplicações críticas, como o controle de tráfego aéreo em aeroportos, por exemplo.
- ***Crescimento Gradual:*** sempre que houver a necessidade de aumento do poder de computação, basta adquirir um ou mais microcomputadores que propiciam um aumento gradual da capacidade do sistema. Em contrapartida, para Sistemas Centralizados, a aquisição de um *mainframe*, além de dispendiosa, é exagerada, pois não é possível obter um aumento gradual, gerando com isso capacidade ociosa.

No caso de computadores pessoais independentes, existem ainda vários outros fatores que justificam a interligação dessas máquinas, a saber:

- ***Compartilhamento de Dados (Data Sharing):*** permite que vários usuários acessem uma base de dados comuns.
- ***Compartilhamento de Equipamentos (Device Sharing):*** possibilita que equipamentos caros, como impressoras e discos rígidos, sejam compartilhados pelos usuários.
- ***Comunicação:*** facilita a comunicação entre os usuários, como a troca de e-mails e documentos.

- ***Flexibilidade:*** torna-se possível distribuir tarefas através das diversas máquinas conectadas à rede, aliviando a carga de determinadas máquinas a partir da utilização de outras máquinas ociosas ou menos carregadas.

A grande desvantagem, conforme abordado anteriormente, refere-se aos softwares disponíveis. Apesar da dificuldade de definição sobre o que o usuário deve e pode fazer e o tanto que o sistema deve fazer, existem atualmente diversas tecnologias, como a de objetos distribuídos, que possibilitam e facilitam o projeto de Sistemas Distribuídos. Os próprios Sistemas Operacionais (OS) estão se desenvolvendo para tornar o ambiente o mais distribuído possível, da mesma forma que os softwares de rede, compiladores, gerenciadores de banco de dados relacional, etc.

Outro grande problema é representado pelas redes de comunicação, que podem saturar ou causar diversos problemas, como perdas de dados. Por isso, para não comprometer o funcionamento do sistema, é imprescindível uma rede bem projetada.

Por fim, existe o problema de segurança. Se por um lado, o compartilhamento de dados e a troca de informações são essenciais em uma rede, a segurança e privacidade dos mesmos tornam-se ainda mais importantes. Por isso, medidas de segurança como criptografia das informações trocadas, uso de autenticação entre os componentes que trocam mensagens pela rede, e uso de senhas, representam o mínimo necessário para garantir a segurança dos dado.

2.2.2 SISTEMAS DISTRIBUÍDOS REAIS

O objetivo de um Sistema Distribuído real é criar a ilusão aos seus usuários de que toda a rede de computadores é na verdade um único computador, ao invés de uma coleção de máquinas distintas. Com isso, pode-se definir um Sistema Distribuído, segundo [2], como sendo:

> *Um sistema que roda em um conjunto de máquinas sem memória compartilhada, embora pareça um único computador a seus usuários.*

2.2.3 CARACTERÍSTICAS PARA O PROJETO DE UM SISTEMA DISTRIBUÍDO

A criação de um sistema único a partir da interligação de diversas máquinas e redes pode parecer, a primeira vista, uma idéia extremamente atraente e simples. Contudo,

as exigências para se atingir tal objetivo tornam o projeto de um sistema distribuído uma tarefa um tanto quanto complexa.

Por isso, o desenvolvimento de um bom projeto deve levar em consideração alguns pontos importantes, que representam os desafios a serem alcançados pelos Sistemas Distribuídos:

- *Transparência*

  Um sistema que faça com que o conjunto de máquinas da rede se pareça com uma única máquina para o usuário é dito ser transparente.

  O conceito de transparência pode ser aplicado a vários aspectos de um sistema distribuído.

  - ***Transparência de localização:*** não é possível aos usuários dizer aonde se encontram os recursos.
  - ***Transparência de migração:*** os recursos disponíveis na rede podem se deslocar à vontade sem a necessidade de mudança de identificação.
  - ***Transparência de replicação:*** os usuários não podem determinar o total de cópias existentes de cada serviço.
  - ***Transparência de concorrência:*** os usuários não percebem que estão concorrendo entre si por algum recurso. Neste caso, o próprio sistema se encarregaria de controlar o acesso simultâneo aos recursos usados.
  - ***Transparência de paralelismo:*** as atividades de um determinado usuário podem ser distribuídas, ou seja, podem ocorrer em paralelo, sem o conhecimento do usuário.

- *Flexibilidade*

  De uma forma geral, um sistema flexível é aquele que está preparado e apto a mudanças, ou seja, a atualização ou mesmo a criação de novos recursos pode ser implementada facilmente. Contudo, as coisas não são tão simples assim.

  Particularmente para os Sistemas Operacionais Distribuídos, existem duas escolas de pensamento referente à estrutura de um sistema distribuído. Uma que defende o uso de um *kernel* tradicional, monolítico, que forneça praticamente todos os serviços necessários, e outra que defende a criação dos

chamados *microkernel*, ou seja, um *kernel* que forneça o mínimo possível de serviços. Neste último caso, serviços como sistemas de arquivo, diretórios, gerenciamento completo de processos dentre outros, seriam disponíveis através de servidores ao nível de usuário. O *kernel* representa o núcleo do SO. Não é difícil perceber que para os sistemas baseados em um *microkernel*, a habilidade de adicionar, excluir e modificar serviços é muito maior, o que torna sua abordagem muito mais flexível e prática.

Um ponto que poderia ser levantado em questão é a troca de mensagens necessária para que um cliente acesse um serviço. Contudo, apesar de um *kernel* monolítico apresentar maior facilidade e rapidez de acesso a esses serviços, bastando desviar o controle para o *kernel* do sistema, estudos mostram que outros fatores dominantes acabam negligenciando o tempo gasto entre a troca de mensagens. Por isso, é provável que os sistemas operacionais baseados em um *microkernel* dominarão gradualmente os sistemas distribuídos.

- *Confiabilidade*

É muito comum relacionar confiabilidade com disponibilidade, ou seja, um sistema que consiga permanecer o maior tempo possível em funcionamento possui uma alta disponibilidade. Seria, portanto, um sistema confiável.

Contudo, *disponibilidade* representa um dos aspectos de um sistema confiável, podendo ser aprimorada através de um projeto mais apurado que leve em conta a distribuição de tarefas críticas, ou ainda através do uso de replicação. No entanto, o próprio uso de replicação deve ser cuidadosamente analisado, pois quanto maior o número de cópias, maior a chance de inconsistência dos dados.

Um segundo aspecto importante é a questão da *segurança*. Enquanto em um sistema isolado a simples entrada de uma senha já é suficiente para autenticar um usuário, em um sistema distribuído não é possível, a priori, confiar nos dados, pois a origem dos mesmos não pode ser seguramente determinada.

Outro aspecto relacionado à confiabilidade é a Tolerância à Falhas. Um sistema que consiga esconder dos usuários a perda ou falha de serviços e recursos, sem ocasionar um custo computacional elevado (*overhead*), oferece tolerância à falhas, ainda que ocorra uma perda de performance como um todo.

- *Desempenho*

  Como um sistema distribuído depende invariavelmente da rede para a troca de mensagens, surge um paradoxo.

  Ao mesmo tempo em que se deve colocar o máximo possível de atividades rodando simultaneamente, através da distribuição dos processos nas máquinas que compõem o sistema, é necessário também diminuir ao máximo o *overhead* causado pelas trocas de mensagens.

  Por isso, é preciso um balanço entre a quantidade de mensagens trocadas e os processos que devem rodar em paralelo.

  A dificuldade consiste em determinar quais processos são mais ou menos apropriados para rodar em paralelo em outra máquina.

- *Escalabilidade*

  Um sistema é dito escalável quando ele se adapta facilmente ao crescimento e mudança da rede. Neste caso, a aquisição de novas máquinas e equipamentos, mesmo com tecnologias diferentes, não deve apresentar qualquer empecilho.

É importante salientar que todas essas características apresentadas anteriormente são igualmente importantes, o que significa que de nada adiantaria possuir um sistema altamente confiável em detrimento da performance, por exemplo.

2.2.4 CONSIDERAÇÕES FINAIS

É importante observar que os Sistemas Distribuídos são diferentes dos Sistemas Paralelos.

Um Sistema Paralelo é um sistema formado por diversos microprocessadores montados em paralelo com o único objetivo de alcançar a máxima velocidade possível para a resolução de um único problema.

O Sistema Distribuído, por sua vez, permite, através da conexão de diversas máquinas a uma rede, que os microprocessadores trabalhem em conjunto, distribuindo tarefas e funções.

2.3 COMUNICAÇÃO

Para que a interação entre as diversas máquinas de um Sistema Distribuído seja possível, é necessário definir métodos e mecanismos de comunicação que viabilizem a troca de informações, independente da localização dos programas e tecnologias de hardware e software existentes.

Os protocolos constituem sem dúvida a principal peça responsável pela comunicação propriamente dita, pois o protocolo utilizado define os padrões necessários para que diferentes sistemas conversem entre si. Em outras palavras, o protocolo escolhido especifica a linguagem empregada para a troca de informações.

Dentre os protocolos existentes, o TCP/IP é de longe o padrão mais utilizado. Além de ser aberto e amplamente documentado pelos RFCs (*Requests For Comments*), o TCP/IP adapta-se facilmente às mais diversas condições de software e hardware, característicos de um ambiente distribuído heterogêneo. Além disso, o TCP/IP é o protocolo que permitiu a criação e a evolução da Internet.

Da mesma forma que a maioria dos protocolos de comunicação existentes, o TCP/IP define os mecanismos básicos empregados para a transferência de informações. Particularmente, o TCP/IP estabelece a comunicação entre dois programas aplicativos, caracterizando uma comunicação ponto a ponto (*peer-to-peer*).

Contudo, apesar de tratar os detalhes de comunicação, o TCP/IP não define uma organização e operação dos programas em um ambiente distribuído. Neste caso, a arquitetura cliente-servidor é o método organizacional preferido para a construção de sistemas usando o TCP/IP.

2.3.1 PROTOCOLOS

Um protocolo de comunicação é um acordo que especifica o formato e o significado das mensagens trocadas entre os computadores [3].

As aplicações que utilizam uma rede não interagem diretamente com o hardware da rede. Ao invés disso, uma aplicação interage com o software de protocolo que por sua vez interage com o hardware. O software de protocolo fornece uma interface de alto nível para as aplicações, manipulando os detalhes e os problemas de comunicação de baixo nível automaticamente.

Devido à complexidade inerente ao projeto de um software de comunicação, no lugar de se criar um único e gigantesco protocolo que especifique todos os detalhes para todas as possíveis formas de comunicação, o problema de comunicação foi dividido em partes. Assim, cada parte passa a ter um protocolo separado que trata somente de alguns detalhes, facilitando o projeto, análise, implementação e teste.

A divisão em múltiplos protocolos aumenta ainda a flexibilidade, pois permite a escolha dos subconjuntos que serão usados conforme a necessidade.

2.3.1.1 Modelo OSI de Sete Camadas

O modelo OSI apresenta um modelo de divisão em camadas que auxilia o desenvolvimento do projeto de protocolos.

Cada camada do modelo corresponde a cada parte da divisão do problema de comunicação. Especificando-se um protocolo para cada camada, é possível projetar um conjunto de protocolos (*suite*) que resolva o problema de comunicação.

A figura 2.1 a seguir apresenta o modelo OSI de sete camadas.

Figura 2.1 – Modelo OSI de Sete Camadas

Neste modelo, a camada mais baixa corresponde ao hardware, enquanto as camadas sucessivas correspondem ao *firmware* ou *software* que utiliza o hardware, mostrando a interação entre os complexos componentes de hardware e o software de protocolo de maneira simples através da interface entre as camadas.

Como o modelo OSI é representado através de retângulos, o software de protocolo construído a partir de tal modelo recebe a denominação de *Pilha de Protocolo*. Como cada pilha de protocolo é projetada de forma independente, a interação entre protocolos de diferentes pilhas de protocolo não é possível.

Contudo, para poder comunicar-se com diferentes protocolos de rede, um computador pode rodar mais de uma pilha de protocolos utilizando o mesmo hardware.

Cabeçalhos Aninhados

O modelo de abstração de camadas é resultado de um princípio conhecido como *Princípio de Camadas*:

A camada N do software de protocolo no computador de destino deve receber exatamente a mesma mensagem enviada pela camada N do software de protocolo do computador de origem.

Para que cada camada possa receber a mensagem exata, o software de protocolo correspondente a cada camada acrescenta um cabeçalho a mensagem antes de passá-la para a próxima camada.

Dessa forma, quando o software de protocolo do computador de destino receber uma mensagem, cada camada poderá extrair o cabeçalho correspondente a camada do computador de origem, verificando a consistência ou não dos dados enviados.

Os cabeçalhos são aninhados a partir da camada de *Enlace* até a camada de *Aplicação*, conforme a figura 2.2.

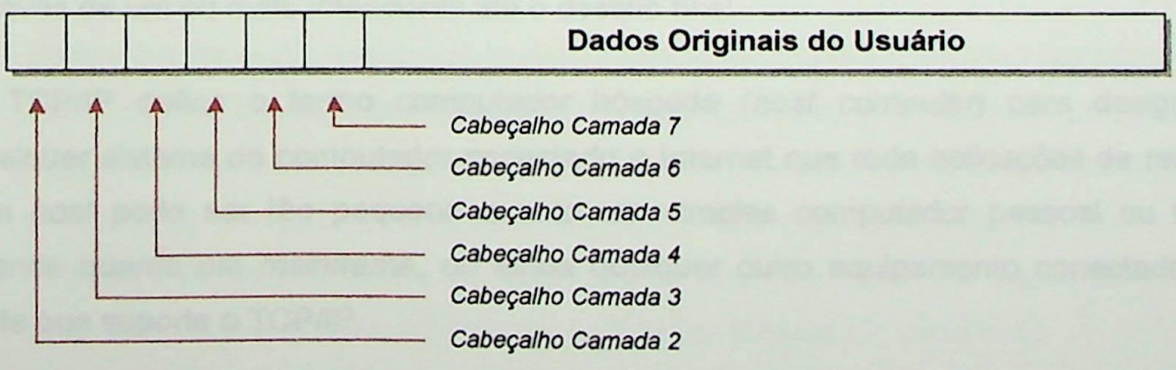

Figura 2.2 – Cabeçalhos Aninhados Acrescentados a Mensagem

2.3.1.2 Camadas e Protocolos TCP/IP

Os protocolos que compõem o padrão TCP/IP [4] possibilitaram, sem dúvida, a grande explosão da interconexão de redes que deu origem a *Internet*.

Juntamente com alguns sistemas de hardware utilizados para interconectar um grupo de redes físicas heterogêneas, o software de rede baseado no TCP/IP provê um sistema de comunicação que permite a comunicação entre pares arbitrários de computadores em redes arbitrárias.

O sistema resultante dessas redes físicas conectadas é conhecido como *Internetwork* ou *Internet*.

Devido às próprias características da *Internet*, o modelo de camadas do TCP/IP baseado no modelo OSI foi adaptado apresentando somente cinco camadas, conforme a figura 2.3 a seguir.

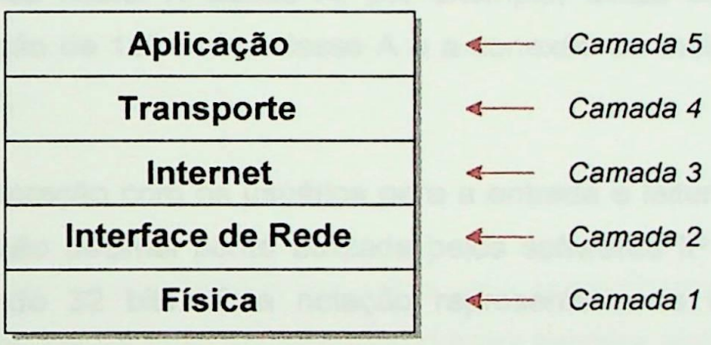

Figura 2.3 – Modelo de Camadas do TCP/IP

A nova camada da Internet especifica o formato dos pacotes enviados pela Internet bem como as máquinas usadas para transmitir os pacotes de um computador através de um ou mais roteadores até o destino final.

O TCP/IP define o termo computador hóspede (*host computer*) para designar qualquer sistema de computador conectado a Internet que rode aplicações de rede. Um *host* pode ser tão pequeno quanto um simples computador pessoal ou tão grande quanto um *mainframe*, ou ainda qualquer outro equipamento conectado à rede que suporte o TCP/IP.

De qualquer forma, os protocolos TCP/IP possibilitam a comunicação entre qualquer par de *hosts*, independentemente das diferenças de hardware.

Endereçamento IP

Para fornecer um endereçamento uniforme para a Internet, o software de protocolo TCP/IP define um esquema de endereçamento abstrato que atribui a cada *host* um endereço único.

Conhecido como *Internet Protocol*, o padrão IP especifica que a cada *host* é atribuído um número único de 32 bits denominado de *Endereço IP*, ou *Endereço Internet.* Dessa forma, o IP é utilizado por todos os usuários, programas de aplicação e camadas de protocolo superiores para comunicarem-se.

O endereço IP é dividido em duas partes: um prefixo e um sufixo. O prefixo identifica a rede física na qual o computador está conectado, enquanto o sufixo identifica um computador individual na rede. Como um endereço IP é único, para otimizar o seu uso, o IP divide os endereços de *host* em três classes primárias: A, B e C. Cada classe define um determinado tamanho de prefixo que permite a criação de várias classes com vários *hosts*. A classe A, por exemplo, utiliza sete bits de prefixo, permitindo a criação de 128 redes classe A e a conexão de mais de 16 milhões de *hosts* a cada rede.

Para facilitar a interação com os usuários para a entrada e leitura de endereços IP, foi criada a notação decimal ponto utilizada pelos softwares IP para expressar os valores binários de 32 bits. Esta notação representa cada *octeto* em decimal separados por um ponto.

Além dos endereços atribuídos a cada computador, o IP define um conjunto de endereços especiais reservados que nunca podem ser atribuídos a um *host*, que são:

- ***Endereço de rede:*** designa uma rede específica com endereço de *host* zero.
- ***Endereço de Difusão Direcionada:*** usado para enviar uma cópia de um pacote para todos os *hosts* de uma rede física identificada pelo endereço de rede. Este endereço é formado pela adição de um sufixo com todos os bits em 1 ao endereço da rede de destino. O endereço de *host* que contêm todos os bits em 1 deve ser reservado para garantir a difusão direcionada em cada rede.
- ***Endereço de Difusão Limitado:*** usado em uma rede física local durante a inicialização de um sistema por um computador que ainda não conhece o número da rede. Este endereço tem todos os bits em 1.

- ***Endereço deste Computador:*** consiste no endereço com todos os bits zeros usado durante *bootstrap* por um computador que não conhece seu endereço.
- ***Endereço de Loopback:*** usado para testar aplicações de rede. O IP reserva o endereço de rede classe A 127 para uso como *Loopback*, independente do endereço de *host*. O endereço 127.0.0.1 é normalmente o mais utilizado.

2.3.2 MODELO CLIENTE-SERVIDOR

O modelo cliente-servidor resolve o problema de *rendez-vous* definindo que em qualquer par de aplicações que estão se comunicando, um lado deve iniciar a execução e aguardar indefinidamente até que o outro lado entre em contato.

Como o TCP/IP não fornece qualquer mecanismo que permita a criação ou ativação automática de programas quando uma mensagem é recebida, um programa deve estar pronto para aceitar uma comunicação antes da chegada de qualquer pedido.

Dessa forma, os programas servidores devem ser executados de preferência logo após a inicialização de uma máquina para garantir o atendimento dos pedidos que chegam dos clientes.

Cliente

De uma forma geral, uma aplicação que inicia uma comunicação é denominada de *cliente*. A aplicação cliente entra em contato com um servidor para enviar um pedido, aguardando uma resposta que será processada em seguida.

As aplicações cliente podem ser padronizadas ou não. As aplicações padronizadas são definidas pelo TCP/IP, que designa uma porta de protocolo bem conhecida e um protocolo específico para cada aplicação. Por exemplo, um cliente de terminal remoto utiliza o protocolo padrão TELNET na porta 23, enquanto um *browser* utiliza o protocolo padrão HTTP na porta 80 para acessar documentos na rede. As demais aplicações são consideradas como não padronizadas ou como localmente definidas.

Para as aplicações não padronizadas deve-se evitar na medida do possível a dependência de serviços disponíveis localmente, pois se a aplicação for distribuída por outras redes é provável que a mesma não funcione, caso esses recursos já estejam sendo utilizados. Por isso, é importante compreender a distinção entre os tipos de aplicações.

Da mesma forma, a aplicação cliente deve permitir sempre a entrada de parâmetros para que o usuário possa especificar a máquina de destino e a porta de protocolo utilizada, tornando o aplicativo mais flexível.

Servidor

Tecnicamente, um servidor é um programa que fica continuamente à espera de pedidos dos clientes. Não é a máquina onde reside o programa. Toda vez que uma nova requisição chega, o servidor executa o serviço solicitado retornando o resultado ao cliente.

Para ter acesso a diversos recursos de um sistema operacional, um servidor exige privilégios especiais sobre o sistema operacional, sobrepondo-se ao mesmo. Conseqüentemente, um servidor passa a ser responsável pela:

- *Autenticação*, verificando a identidade do cliente.
- *Autorização*, determinando se um dado cliente tem direito de acesso ao serviço oferecido.
- *Segurança dos Dados*, mantendo as informações particulares fora do alcance de usuários não autorizados.
- *Privacidade*, garantindo que as informações particulares não sejam reveladas.
- *Proteção*, restringindo o acesso aos recursos do sistema para as diferentes aplicações da rede.

O controle destas funções acaba aumentando consideravelmente a complexidade do projeto de um servidor.

Outro aspecto importante para o projeto de um servidor é o tratamento das informações dos clientes. A informação mantida pelo servidor sobre os clientes é denominada de *Informação de Estado*. Um servidor que mantém informações a respeito dos clientes pode aumentar a eficiência da comunicação, pois as mensagens trocadas podem ser reduzidas, agilizando o processamento dos pedidos enviados pelos clientes. Por outro lado, a consistência do estado das informações depende da confiabilidade do protocolo usado e também da capacidade de um servidor contornar problemas como a queda e reinício de uma máquina.

Em um ambiente distribuído heterogêneo, como a Internet, onde máquinas falham e reiniciam, e mensagens podem ser perdidas, atrasadas, duplicadas, ou entregues

fora de ordem, o projeto de um servidor que mantém as informações de estado resulta em protocolos de aplicação complexos que são difíceis de implementar. Neste caso, o projeto de um servidor que não mantém as informações de estado é mais indicado.

Para tanto, o protocolo de aplicação utilizado deve garantir necessariamente uma operação idempotente, ou seja, o servidor deve gerar a mesma resposta não importa quando ou quantas vezes um pedido é feito.

Interação Cliente-Servidor

A informação trocada entre as aplicações pode fluir em qualquer ou ambas direções entre um cliente e um servidor. Embora muitos serviços restrinjam-se simplesmente a um cliente enviando pedidos e um servidor retornando respostas, outras interações são igualmente possíveis.

A figura 2.4 a seguir mostra a interação entre um cliente e servidor usando o TCP/IP.

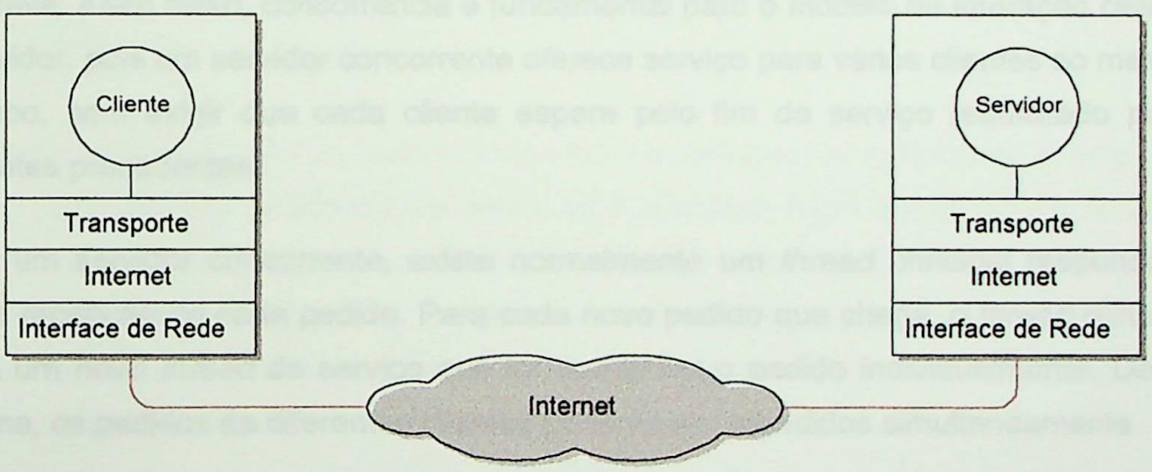

Figura 2.4 – Interação cliente-servidor

Pela figura, pode-se observar que tanto o cliente quanto o servidor necessitam de uma pilha de protocolos completa para comunicarem-se, interagindo diretamente com a camada de transporte que utiliza as camadas mais baixas para enviar e receber as mensagens.

Uma mesma máquina pode oferecer ainda diversos serviços ao mesmo tempo, aproveitando melhor seus recursos disponíveis enquanto um outro servidor estiver à espera de pedidos. Neste caso, é necessário um programa servidor separado para cada serviço.

Para distinguir um servidor do outro, o protocolo de transporte designa um identificador único para cada serviço conhecido como *porta de protocolo*.

Um servidor especifica a porta de protocolo para o serviço que ele oferece, aguardando pela comunicação dos clientes. O cliente, por sua vez, especifica a porta de protocolo para o serviço desejado quando envia um pedido. Quando uma mensagem de entrada chega na máquina do servidor, o software TCP/IP utiliza então a porta de protocolo para direcionar cada pedido para o servidor correspondente, garantindo a entrega correta das mensagens.

Contudo, quando um servidor for acessado por diversos clientes simultaneamente, não será possível atender a todas as chamadas simultaneamente, pois cada cliente deverá esperar até que o servidor finalize os pedidos precedentes. Visivelmente, esta é uma situação totalmente indesejável, pois inviabiliza o projeto de um servidor de grande escala.

A programação concorrente, através de *threads* de controle, permite resolver este entrave. Além disso, concorrência é fundamental para o modelo de interação cliente-servidor, pois um servidor concorrente oferece serviço para vários clientes ao mesmo tempo, sem exigir que cada cliente espere pelo fim do serviço requisitado pelos clientes precedentes.

Em um servidor concorrente, existe normalmente um *thread* principal responsável pela recepção de cada pedido. Para cada novo pedido que chega, o *thread* principal cria um novo *thread* de serviço que irá manipular o pedido individualmente. Dessa forma, os pedidos de diferentes clientes poderão ser atendidos simultaneamente.

Para identificar a cópia correta do servidor, ou seja, o *thread* de serviço de um cliente, o protocolo de transporte designa também uma porta de protocolo para cada cliente. Através da combinação dos identificadores do cliente e do servidor, juntamente com os endereços IP de cada um, é possível para o software de protocolo na máquina do servidor identificar a cópia correta de um servidor concorrente.

Sem Conexão vs Orientado a Conexão

Um programa pode optar entre dois tipos de interação: sem conexão ou orientado a conexão.

O TCP/IP oferece exatamente dois protocolos de transporte que se enquadram perfeitamente nesses dois tipos de interação: o TCP, para uma interação orientada a conexão, e o UDP, para uma interação sem conexão. O TCP é o protocolo mais indicado, pois ele provê uma comunicação confiável e orientada a conexão. Os programas só devem utilizar o UDP se o protocolo de aplicação controlar a confiabilidade, se a aplicação exigir *broadcast* ou *multicast* de hardware, ou se a aplicação não puder tolerar o *overhead* do circuito virtual.

Servidores como Clientes

Apesar das diferenças existentes entre um programa cliente e um programa servidor, é possível que um programa servidor torne-se cliente de outro servidor para acessar um determinado serviço que complete o pedido do cliente.

Assim, o papel tanto de um cliente quanto de um servidor, depende da condição em que o programa se encontra em determinado momento, o que mostra que a classificação cliente-servidor não é rígida.

2.3.3 SOCKETS

Quando uma aplicação de rede interage com o software de protocolos, a aplicação deve especificar os detalhes para definir se a aplicação é um cliente ou um servidor. A interface que uma aplicação utiliza para interagir com o software de protocolo de transporte é conhecida como Interface para Programas Aplicativos – API. A API define um conjunto de operações que uma aplicação pode realizar para interagir com o software de protocolo.

A API *socket* [5] do BSD Unix é sem dúvida o padrão mais utilizado. Para utilizar a API *socket*, uma aplicação deve escolher um protocolo de transporte, definir o endereço de protocolo IP da máquina remota, e especificar se a aplicação é cliente ou servidor.

A figura 2.5 apresenta a seqüência de funções básicas da API *socket* utilizada por uma aplicação cliente e outra servidora.

O cliente começa chamando a função *gethostbyname* para converter o nome de um computador para um endereço IP e *getprotobyname* para converter o nome de um protocolo para a forma binária interna usada pela função *socket*.

Em seguida, o cliente chama a função *socket* para criar um *socket* e *connect* para conectar o *socket* ao servidor. Uma vez que a conexão tenha sido estabelecida, o cliente passa a chamar repetidamente a função *recv* para receber os dados enviados pelo servidor. Quando todos os dados forem recebidos, o cliente chama a função *close* para encerrar a comunicação, fechando o *socket*.

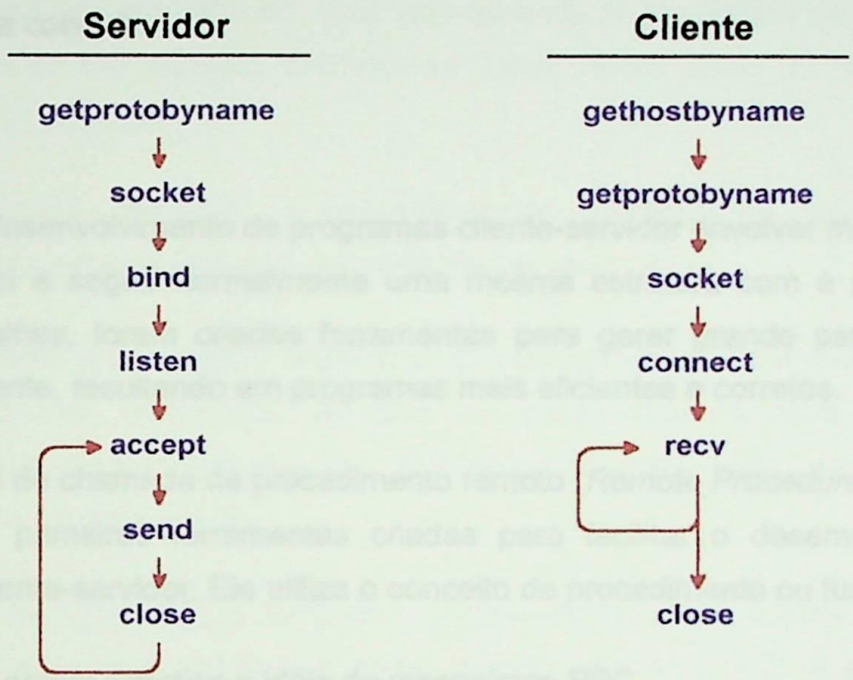

Figura 2.5 – Chamadas de Funções *Socket* de um Cliente e Servidor

O servidor, por sua vez, inicia chamando a função *getprotobyname* para gerar um identificador binário interno necessário para o protocolo antes da criação do *socket*. Uma vez que um *socket* tenha sido criado, o servidor chama a função *bind* que designa uma porta de protocolo local para o *socket* seguida da função *listen* que coloca o *socket* no modo de "espera por dados". O servidor entra então em um *loop* infinito e fica escutando a porta a procura de um pedido de conexão gerado por algum cliente. Quando um pedido chega na porta que o servidor está "escutando", ele chama a função *accept* para aceitar ou recusar a requisição de entrada. A função *send* é usada para enviar uma mensagem ao cliente, e a função *close* para finalizar a conexão. Após encerrar uma conexão, o servidor inicia um novo ciclo chamando novamente a função *accept*, seguindo a seqüência indefinidamente.

Para aplicações concorrentes, existe um *socket* de entrada criado no *thread* principal do programa responsável pela recepção de todas as requisições de entrada. Toda vez que uma nova requisição chega, o *thread* principal cria um novo *socket* em um

thread de serviço que passa a ser responsável pela nova conexão. Dessa forma, o *socket* do *thread* principal fica liberado para aceitar novos pedidos de entrada, atendendo de forma concorrente vários clientes simultaneamente. O *socket* que um servidor concorrente utiliza para aceitar as novas conexões de entrada existe enquanto o *thread* principal do servidor estiver ativo. Um *socket* usado para uma conexão específica existe somente enquanto o *thread* de serviço criado existir para manipular esta conexão.

2.3.4 RPC

Pelo fato do desenvolvimento de programas cliente-servidor envolver muitos detalhes de baixo nível e seguir normalmente uma mesma estrutura com a repetição dos mesmos detalhes, foram criadas ferramentas para gerar grande parte do código automaticamente, resultando em programas mais eficientes e corretos.

O mecanismo de chamada de procedimento remoto (*Remote Procedure Call* – RPC), foi uma das primeiras ferramentas criadas para facilitar o desenvolvimento de programas cliente-servidor. Ele utiliza o conceito de procedimento ou função.

A figura 2.6 a seguir sintetiza a idéia do mecanismo RPC.

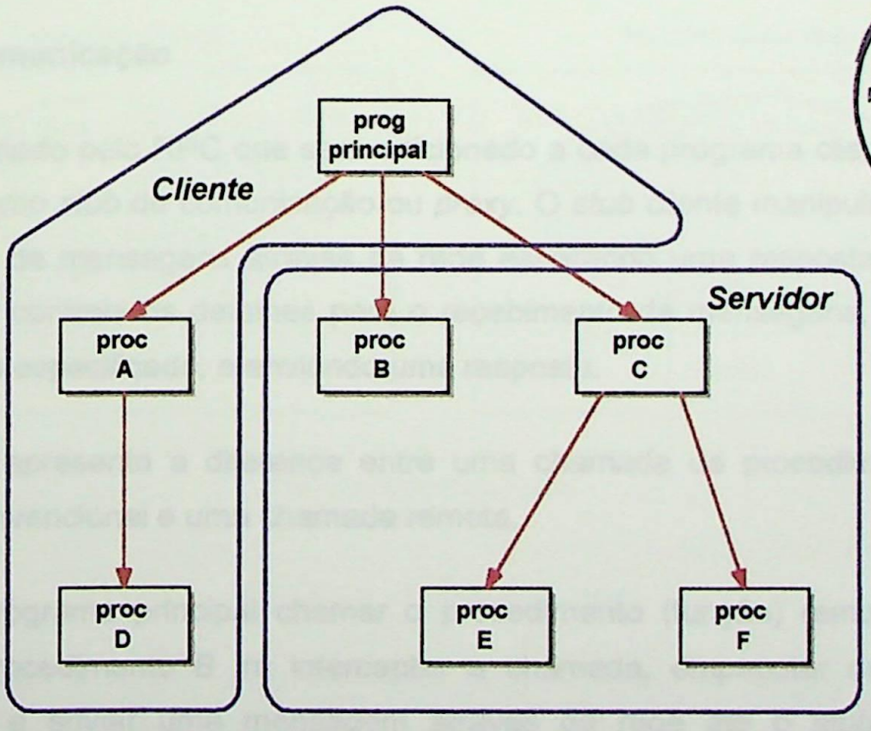

Figura 2.6 – Procedimentos Distribuídos

Partindo da linguagem estruturada baseada em funções, o RPC permite que linguagens de programação convencionais façam chamadas de funções residentes em computadores situados em outros pontos da rede.

O desenvolvimento do programa é feito normalmente como se o programa fosse rodar em uma única máquina, sem tratar os detalhes sobre a rede de computadores ou os protocolos de comunicação. Uma vez resolvido os problemas de programação, o programa pode ser dividido, definindo-se quais partes serão do cliente e quais partes serão do servidor.

Finalmente, através do RPC, o mecanismo de chamada de funções é então estendido para permitir a inclusão dos detalhes de rede.

Para usar o RPC é necessário especificar o conjunto de procedimentos que serão remotos – servidor, bem como os tipos dos parâmetros de cada procedimento remoto. A partir dessas informações, a ferramenta pode então criar o software responsável pela comunicação.

No exemplo da figura 2.6, o programa original dividido deu origem a um cliente formado pelo programa principal e pelos procedimentos A e D, e a um servidor constituído pelos demais procedimentos.

Stubs de Comunicação

O software criado pelo RPC que será adicionado a cada programa cliente-servidor é conhecido como *stub* de comunicação ou *proxy*. O *stub* cliente manipula os detalhes para o envio de mensagens através da rede esperando uma resposta, enquanto o *stub* servidor controla os detalhes para o recebimento de mensagens, chamando o procedimento especificado, e enviando uma resposta.

A figura 2.7 apresenta a diferença entre uma chamada de procedimento em um programa convencional e uma chamada remota.

Quando o programa principal chamar o procedimento (função) remoto B, o *stub* cliente do procedimento B irá interceptar a chamada, empacotar os parâmetros (*marshaling*) e enviar uma mensagem através da rede até o *stub* servidor do procedimento B. O *stub* servidor, por sua vez, usa os mecanismos de chamada de procedimento convencionais para invocar o procedimento especificado, enviando os

resultados de volta ao *stub* cliente. Finalmente, a resposta recebida pelo *stub* cliente é devolvida para o programa principal exatamente como se a chamada tivesse sido feita a um procedimento local.

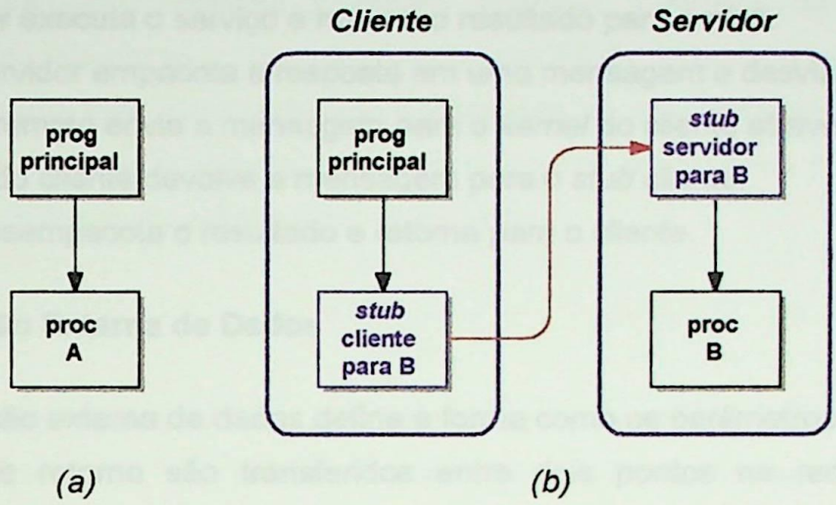

Figura 2.7 – (a) Chamada Convencional (b) Chamada Remota

A figura 2.8 permite observar melhor a seqüência resultante da interação que ocorre entre o *stub*, o aplicativo e o *kernel* de uma máquina cliente e de uma máquina servidora.

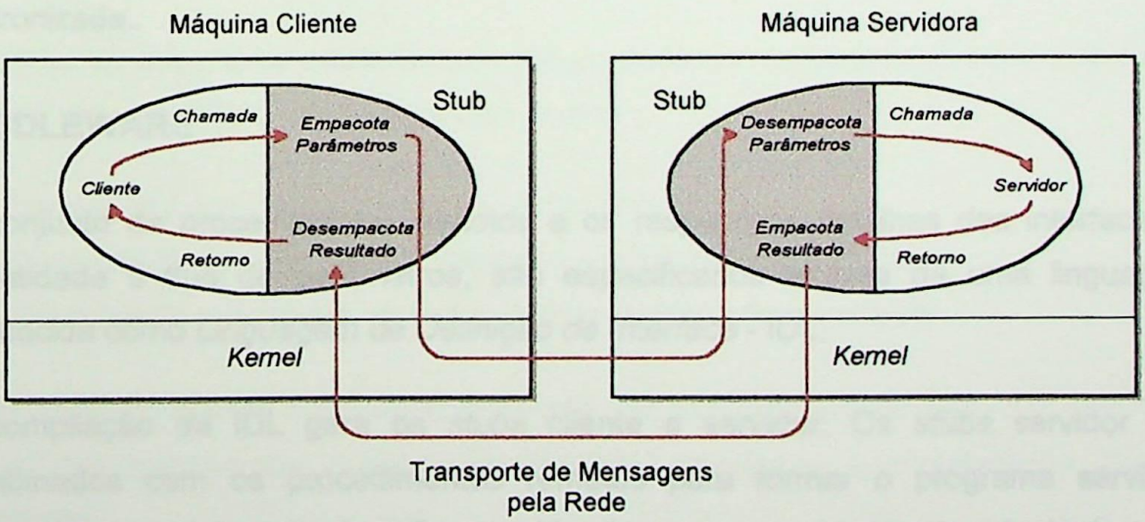

Figura 2.8 – Mecanismo RPC

De acordo com a figura, uma chamada de procedimento remoto (RPC) segue basicamente os seguintes passos:

1. O procedimento do cliente invoca o *stub* cliente no modo normal.

2. O *stub* cliente constrói uma mensagem e desvia para o *kernel*.
3. O *kernel* envia a mensagem para o *kernel* remoto pela rede.
4. O *kernel* remoto entrega a mensagem para o *stub* servidor.
5. O *stub* servidor desempacota os parâmetros (*unmarshaling*) e chama o servidor.
6. O servidor executa o serviço e retorna o resultado para o *stub*.
7. O *stub* servidor empacota a resposta em uma mensagem e desvia para o *kernel*.
8. O *kernel* remoto envia a mensagem para o *kernel* do cliente através da rede.
9. O *kernel* do cliente devolve a mensagem para o *stub* cliente.
10. O *stub* desempacota o resultado e retorna para o cliente.

Representação Externa de Dados

A representação externa de dados define a forma como os parâmetros das funções e os valores de retorno são transferidos entre dois pontos na rede. Devido às diferentes representações internas de dados existentes em um sistema distribuído heterogêneo, no qual os clientes e servidores podem executar em plataformas distintas, o RPC deve ser capaz de manipular a conversão de um tipo de representação de dados para outro.

Um método largamente utilizado define uma representação externa padronizada e exige que cada lado faça a conversão entre as representações locais e a externa padronizada.

2.3.5 MIDDLEWARE

O conjunto de procedimentos remotos e os respectivos detalhes das interfaces – quantidade e tipo de parâmetros, são especificados através de uma linguagem conhecida como *Linguagem de Definição de Interface* - IDL.

A compilação da IDL gera os *stubs* cliente e servidor. Os *stubs* servidor são combinados com os procedimentos remotos para formar o programa servidor, enquanto os *stubs* cliente são combinados com o programa principal e os procedimentos locais para formar o programa cliente.

A ferramenta responsável pela compilação da IDL para a geração dos *stubs* é usualmente conhecida como *middleware*, pois ela fornece o software necessário que se encaixa entre um programa de aplicação convencional e um software de rede.

Contudo, atualmente, o termo *middleware* pode ser melhor entendido como a camada de software que permite a comunicação entre as partes componentes de um sistema distribuído abstraindo a parte do sistema operacional, hardware e rede.

Os *middlewares* mais recentes suportam sistemas baseados na Orientação a Objetos e desempenham a mesma função do RPC. Enquanto a programação estruturada usa o RPC para estender a chamada de procedimentos pela rede, a programação orientada a objetos dispõe dos objetos distribuídos para estender a invocação de métodos de objetos através da rede.

2.3.6 SEGURANÇA

Segurança é com certeza um dos aspectos mais importantes no projeto de sistemas distribuídos. Apesar de não existir uma definição absoluta de segurança de rede, qualquer organização pode definir a sua própria política de segurança, de acordo com os itens que devem ser protegidos.

Contudo, a criação de uma política de segurança de rede pode tornar-se complexa, pois muitas vezes é difícil determinar o valor da informação. A política de segurança deve aplicar-se tanto para as informações armazenadas em computadores como também para as informações que trafegam pela rede.

A política de segurança adotada deve atingir ainda um compromisso entre segurança e eficiência, considerando-se aspectos como:

- *Integridade dos Dados:* proteção contra alterações.
- *Disponibilidade dos Dados:* proteção contra falhas de serviço.
- *Privacidade dos Dados:* proteção contra acesso não autorizado.

Além disso, o controle e a responsabilidade das informações precisam ser definidos, como autorizações de acesso, modificação de dados e monitoração dessas operações.

2.3.6.1 Mecanismos de Segurança

Embora existam técnicas para assegurar a integridade dos dados contra danos acidentais de hardware, como bits de paridade, *checksums* e *cyclic redundancy*

checks (CRCs), é impossível garantir que os mesmos não tenham sido adulterados ou manipulados por um intruso.

Para garantir a integridade de mensagens contra alterações intencionais, foram criados diversos mecanismos de segurança. Em geral, um código de autenticação de mensagem (MAC) que não pode ser quebrado ou falsificado é utilizado para codificar os dados transmitidos. Este esquema de segurança é conhecido como criptografia.

Através do uso de uma chave secreta, por exemplo, os dados transmitidos são codificados e decodificados, assegurando a proteção e autenticidade das mensagens trocadas, pois somente os usuários que possuem a chave podem decifrar os dados.

De uma forma geral, a encriptação embaralha os bits da mensagem de tal forma que somente o receptor esperado possa desembaralhar os bits para recuperar a mensagem original.

O processo para encriptar uma mensagem *M* através de uma chave de encriptação *K* pode ser expresso como:

$$E = encripta\ (K,M)$$

A mensagem *E* é decifrada pela mesma chave *K* através da função:

$$M = decifra\ (K,E)$$

Matematicamente, a função *decifra* é o inverso da *encripta*, ou seja:

$$M = decifra\ (K,\ encripta\ (K,M)$$

Uma técnica de encriptação muito utilizada atribui duas chaves a cada usuário: uma chave secreta mantida pelo usuário, e uma chave pública distribuída para o público de interesse.

Uma mensagem encriptada com a chave pública *pub* só pode ser decifrada através da chave privada *prv*, enquanto que uma mensagem encriptada com a chave privada só pode ser decifrada com a chave pública. Matematicamente, o processo pode ser expresso como:

$$M = decifra\ (pub,\ encripta\ (prv,M))$$

Uma outra técnica conhecida como *Assinatura Digital* é usada juntamente com o mecanismo de encriptação para autenticar a origem de uma mensagem.

Primeiro, a mensagem é assinada através da chave privada do remetente usada para encriptar a mensagem. Em seguida, a mensagem encriptada é novamente encriptada através da chave pública do destinatário. Matematicamente, o processo de encriptação dupla pode ser expresso como:

$$X = encripta\ (\ pub\text{-}u2,\ encripta\ (\ prv\text{-}u1, M))$$

onde: pub-u2 – chave pública do destinatário
prv-u1 – chave privada do remetente

Uma vez recebida a mensagem *X*, o destinatário usa sua chave privada para decifrar a mensagem, removendo um nível da encriptação que deixa a mensagem assinada digitalmente. Em seguida, o destinatário utiliza a chave pública do remetente para decifrar a mensagem novamente, recuperando finalmente a mensagem *M* original. O processo pode ser expresso como:

$$M = decifra\ (\ pub\text{-}u1,\ decifra\ (\ prv\text{-}u2, X))$$

onde: pub-u1 – chave pública do remetente
prv-u2 – chave privada do destinatário

Dessa forma, o usuário garante ao mesmo tempo a privacidade e autenticidade de qualquer mensagem trocada.

No caso do mecanismo de senha usado para o controle de acesso a recursos, devem ser tomados cuidados especiais para prevenir a captura de senhas na rede.

FireWall

Para prevenir que um computador de uma rede acesse computadores ou serviços arbitrários, é empregada uma técnica conhecida como *Filtragem de Pacotes*.

Um filtro de pacotes é um software que permite examinar os campos dos cabeçalhos de cada pacote transmitido pela rede. O software roda em um roteador, equipamento responsável pela interligação de diferentes redes. A partir da configuração do filtro, é possível especificar quais pacotes podem ou não passar através de um roteador.

Um *firewall* é um filtro de pacotes configurado para proteger uma organização contra o tráfego indesejável da Internet.

Conforme ilustrado pela figura 2.9, o filtro é colocado no roteador que conecta a organização ao resto da Internet.

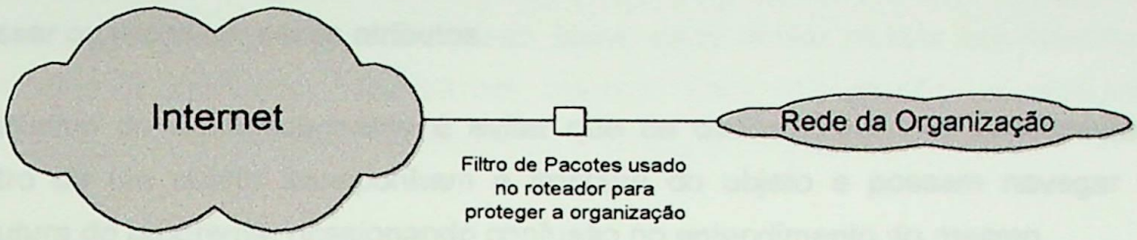

Figura 2.9 – Filtro de Pacotes usado em um *Firewall*

Os Firewalls são as ferramentas de segurança mais importantes usadas para controlar as conexões de rede entre diferentes organizações, pois criam um perímetro de segurança que previne o acesso externo de intrusos que poderiam sondar os computadores de uma organização, congestionar a rede com tráfego indesejável ou mesmo atacar os computadores.

2.4 ORIENTAÇÃO A OBJETOS

Apesar de ser um conceito antigo, somente a partir dos anos 90 é que a orientação a objetos ganhou força e explodiu no mercado de desenvolvimento de software.

O objetivo da orientação a objetos é basicamente representar em software os objetos que se pressupõe existir no mundo real, descrevendo os mesmos através de dados (informações) e ações que manipulam estes dados. Ela oferece técnicas avançadas de programação que facilitam a elaboração, o desenvolvimento e a análise de sistemas complexos.

Algumas das principais propriedades de software [6] que fazem parte da orientação a objetos são apresentadas a seguir.

2.4.1 ENCAPSULAMENTO

Encapsulamento consiste no empacotamento de operações e atributos que representam o estado de um determinado tipo de objeto, de modo que o seu estado

possa ser acessado ou modificado somente através de uma interface bem definida que é fornecida pelo encapsulamento.

Um objeto consiste assim em um conjunto de operações e atributos. Os atributos representam as informações a respeito do objeto, enquanto as operações são os procedimentos ou funções acessíveis que podem ser usadas por outros objetos para acessar ou modificar esses atributos.

O objetivo do encapsulamento é evitar que os dados envolvidos (encapsulados) dentro de um objeto transponham a fronteira do objeto e possam navegar pela estrutura do programa, ocasionando confusão no entendimento do mesmo.

Além desta grande finalidade, o encapsulamento também força quem precisa dos dados a usar a interface do objeto, padronizando a forma de acesso, o que tem a vantagem de tornar o programa mais claro.

2.4.2 OCULTAMENTO DE INFORMAÇÃO/IMPLEMENTAÇÃO

Através do ocultamento de informações, é possível esconder os detalhes da implementação das operações de um objeto, melhorando consideravelmente a inteligibilidade do software. Dessa forma, um observador externo enxerga um objeto como sendo uma caixa preta. Em outras palavras, o usuário sabe o que o objeto pode fazer, mas não tem a menor idéia de como o objeto pode fazê-lo, ou seja, como o objeto é construído internamente.

Com isso, surgem duas conseqüências diretas. Primeiro, estando a interface do usuário bem definida, qualquer atualização ou modificação da implementação da mesma terá um impacto mínimo ou mesmo nenhum impacto no resto do sistema como um todo.

Finalmente, como o conteúdo das informações é independente da sua forma de representação, não é possível interferir no objeto e nem mesmo introduzir artifícios de programação, garantindo assim a consistência e robustez do código.

2.4.3 RETENÇÃO DO ESTADO

A capacidade que um objeto têm de manter seu estado, ou seja, de reter suas informações por um tempo indefinido consiste na retenção do estado.

É importante salientar que as idéias apresentadas até então já eram bem conhecidas sob o termo ADT – *Abstract Data-Type*. Contudo, a orientação a objetos vai muito além do ADT, conforme mostram as próximas propriedades.

2.4.4 IDENTIDADE DO OBJETO

Independente da classe ou do estado atual, cada objeto possui um identificador único que o distingue dos demais objetos. Este identificador é comumente denominado de referência ou manipulador (*handle*) do objeto.

2.4.5 MENSAGENS

Mensagem é o mecanismo empregado por um objeto **obj1** para requisitar, a um segundo objeto **obj2**, a execução de um determinado serviço que é implementado por um dos métodos da interface do objeto **obj2**.

A estrutura básica de uma mensagem compreende o manipulador do objeto, a operação desejada e os possíveis argumentos necessários para a execução da mesma, como a seguir:

```
Obj2.AjustaHora(in Hora, in Minutos, out Modificado);
```

Os parâmetros **in** e **out** indicam a direção ou fluxo da informação.

Apesar de simples, tal estrutura possibilita o uso de polimorfismo, sobrecarga e ligação dinâmica, recursos avançados e extremamente poderosos que serão abordados na seqüência.

2.4.6 CLASSES

O modelo a partir do qual os objetos são criados é definido como classe.

Cada objeto possui a mesma estrutura e comportamento da classe da qual ele é instanciado. Assim, pode-se dizer que se um objeto **obj1** pertence à classe **C**, ele é uma instância de **C**.

O que diferencia um objeto do outro é exatamente o manipulador do objeto e o seu estado, conforme discutido anteriormente.

2.4.7 HERANÇA

A capacidade que uma classe **D** tem de herdar os atributos e operações de uma classe **C** é conhecida como herança. É como se os atributos e as operações tivessem sido definidos na própria classe **D**. Neste caso, a classe **C** é definida como sendo uma superclasse de **D**, e **D** passa a ser uma subclasse de **C**.

O uso de herança torna possível, a partir de uma classe geral, a criação de subclasses específicas, facilitando enormemente o desenvolvimento de softwares.

Um exemplo de herança é a forma:

```
class Carro inherits from Automovel;
```

Este código mostra o uso de herança simples, ou seja, a partir de uma única superclasse. Herança múltipla é quando uma classe herda de duas ou mais superclasses.

2.4.8 POLIMORFISMO

De uma maneira geral, é possível definir polimorfismo como a capacidade de assumir diferentes tipos. Com isso, é possível criar uma variável polimórfica que aponta para objetos de classes diferentes em momentos diferentes.

Com o polimorfismo, aparecem dois conceitos interessantes:

- ***Ligação dinâmica ou Ligação Tardia (Dynamic Binding ou Late Binding):*** técnica através da qual o correto pedaço de código a ser executado é determinado somente em tempo de execução, e não em tempo de compilação, como na programação comum.
- ***Redefinição (Overriding):*** consiste na redefinição de um método da superclasse em suas subclasses, possibilitando diferentes implementações.

2.4.9 PROGRAMAÇÃO GENÉRICA

Programação genérica (*Generic Programming*) se refere à capacidade de construir uma classe **C** que contem uma ou mais variáveis indefinidas cujos tipos só serão conhecidos quando um objeto da classe **C** for instanciado. Em outras palavras,

significa que pelo menos uma das variáveis usadas dentro de **C** não precisa ser atribuída a um tipo específico até o instante da instanciação.

A classe **C**, neste caso, é formalmente conhecida como uma classe genérica, ou ainda como uma classe parametrizada. Em C++ o termo usado é classe *Template*.

O exemplo a seguir ilustra bem esta idéia:

```
template<class T>
class Soma
{
  …
  T arg1,
    arg2;
  …
  void print(T var);
  …
}
```

Neste exemplo, T representa a variável indefinida passada como parâmetro que será utilizada para personalizar o objeto criado. Dessa forma, pode-se instanciar uma soma de inteiros, de reais, ou de strings, por exemplo.

2.5 OBJETOS DISTRIBUÍDOS

O grande avanço dos sistemas de comunicação, notadamente as redes de computadores com seus sistemas cliente-servidor e posteriormente a Internet, provocou o surgimento de uma nova tecnologia de softwares orientados para a comunicação entre diversas máquinas. Esta tecnologia, aliada ao poder da programação orientada a objetos, deu origem ao conceito de Objetos Distribuídos.

Portanto, objetos distribuídos são objetos construídos e ligados segundo as técnicas e a filosofia da orientação a objetos, só que acrescidos de um componente novo responsável por todos os detalhes de comunicação. É exatamente este componente novo que permite a interação de objetos remotos através da troca de mensagens que são transmitidas pela rede. Em outras palavras, os objetos distribuídos estendem o conceito de um objeto comum. É a evolução do objeto convencional.

Mais especificamente, a tecnologia de objetos distribuídos estende um sistema de programação orientado a objeto, permitindo que objetos sejam distribuídos através de uma rede heterogênea. Dessa forma, os objetos distribuídos passam a interoperar como um sistema unificado. Além disso, estes objetos podem ser

distribuídos em diferentes máquinas de uma rede, funcionando independentemente de uma aplicação, apesar de parecerem locais à mesma.

Conseqüentemente, qualquer objeto da rede pode solicitar um determinado serviço oferecido por outro objeto, sem se preocupar com a localização, linguagem ou mesmo a plataforma na qual esse objeto foi criado. A própria estrutura de programação permanece a mesma, com a chamada de método convencional.

O componente que possibilita a interação entre os objetos distribuídos é conhecido como *middleware*, já visto anteriormente. É um conceito interessante que expressa bem a idéia da ponte criada para permitir a comunicação entre os diversos objetos de um sistema distribuído. A diferença em relação ao modelo RPC é que agora, no lugar de chamar um procedimento remoto, o objeto cliente invoca um método remoto de um objeto distribuído, caracterizando o mecanismo de invocação de métodos remotos (*Remote Method Invocation* – RMI).

O *middleware* compreende diversas partes, como protocolos de comunicação, serviços básicos e outros recursos que dependem da tecnologia de objetos distribuídos utilizada.

As linguagens de programação disponíveis para o desenvolvimento de objetos distribuídos dependem da tecnologia escolhida. De uma maneira geral, as linguagens de programação orientadas a objetos C++ e Java são de longe as mais empregadas. Contudo, a tecnologia de *middleware* atual permite a ponte entre objetos escritos em linguagens orientadas a objeto com linguagens não orientadas a objeto como Pascal, Cobol, Ada.

Esta é inclusive uma grande vantagem para o uso de Objetos Distribuídos no MAE, por que o *middleware* vai permitir a interoperabilidade entre as linguagens existentes em seus sistemas mais antigos, que rodam em *mainframes*, com os novos sistemas que rodam em plataformas Unix e Windows e já usam a tecnologia de Orientação a Objetos.

2.5.1 VANTAGENS E DESVANTAGENS

Sem dúvida a grande contribuição da tecnologia de Objetos Distribuídos é a possibilidade de integração de um sistema heterogêneo, viabilizando o desenvolvimento de sistemas distribuídos em ambientes dessa natureza.

Dessa forma, processos de máquinas sobrecarregadas podem ser convenientemente distribuídos, através da rede, em outras máquinas que estejam ociosas ou menos sobrecarregadas, aproveitando-se ao máximo todos os recursos oferecidos pela rede.

A orientação a objetos, em particular, possui características que permitem um desenvolvimento mais sólido e rápido das aplicações, resultando principalmente na reutilização de código que garante a flexibilidade do sistema frente a mudanças e atualizações.

O *middleware*, por sua vez, elimina toda a complexidade envolvida na comunicação entre plataformas diferentes que fazem parte de um ambiente distribuído e heterogêneo.

A grande desvantagem reside na principal parte de um sistema distribuído, a rede, pois qualquer perda ou falha de comunicação pode inviabilizar completamente a operação do sistema como um todo. Por isso, é indispensável o uso de uma rede segura e bem projetada.

A própria perda de uma máquina, por exemplo, representa outro grande problema, pois os objetos presentes nesta máquina não estarão mais disponíveis. Neste caso, o uso adequado de replicação pode aumentar sensivelmente a disponibilidade de serviços distribuídos pela rede. Um outro recurso muito importante também é a implementação de tolerância à falhas, que pode garantir a estabilidade do sistema mesmo diante de falhas de comunicação. Com isso, a confiabilidade do sistema como um todo aumenta consideravelmente.

2.5.2 ARQUITETURAS

As arquiteturas de objetos distribuídos disponíveis atualmente que mais se destacam para o desenvolvimento de sistemas distribuídos são as seguintes:

- CORBA – *Common Object Request Broker Architecture*, padrão do OMG .
- DCOM – *Distributed Component Object Model*, da Microsoft.
- Jini, da Sun MicroSystems.

De uma maneira geral, a tecnologia DCOM [7] [8] se destaca para aplicações em ambiente Windows, pois é uma tecnologia inerente da Microsoft. Jini [9] [10] [11] [12]

é uma solução da Sun para sistemas heterogêneos, só que depende do ambiente Java. O padrão CORBA, por sua vez, se destaca como a única solução disponível para o desenvolvimento de sistemas distribuídos realmente heterogêneos, ou seja, sistemas com hardwares diferentes, sistemas operacionais diversos, aplicativos desenvolvidos em diferentes linguagens de programação, redes de comunicação de variadas tecnologias e diferentes protocolos de comunicação, como o ambiente do MAE, caso de estudo deste trabalho.

CAPÍTULO 3

PADRÃO CORBA

3.1 INTRODUÇÃO

O problema de desenvolver aplicações para sistemas distribuídos e heterogêneos pode ser tratado, de uma maneira geral, a partir de duas regras chaves:

- Encontrar modelos e abstrações independentes de plataformas que possam ser usados na solução de uma grande variedade de problemas.
- Esconder o máximo possível a complexidade de baixo nível sem sacrificar muito o desempenho.

O padrão CORBA (*Common Object Request Broker Architecture*) desenvolvido pelo OMG (*Object Management Group*), destaca-se como a única solução disponível atualmente para possibilitar a criação mais produtiva de sistemas que funcionam em ambientes desse tipo. Sua especificação fornece um conjunto balanceado de abstrações flexíveis e serviços concretos necessários para a solução prática de problemas associados com sistemas distribuídos e heterogêneos.

3.2 OMG

O OMG [13], fundado em 1989, é uma organização internacional suportada por mais de 800 membros, entre usuários e indústrias produtoras de bens e serviços de informática.

A primeira especificação chave criada pelo OMG é a OMA – *Object Management Architecture*, que define dois modelos relacionados para descrever como objetos distribuídos e a interação entre os mesmos pode ser especificada independente de plataforma:

- ***Modelo de Objeto*** *(Object Model)*, que define um objeto como sendo uma *entidade encapsulada* com *identidade distinta e imutável* cujos serviços são acessados somente através de *interfaces* bem definidas.

- ***Modelo de Referência*** *(Reference Model)*, que fornece categorias de interface que são agrupamentos gerais para interfaces de objetos.

Todas as categorias de interface estão conceitualmente ligadas através de um despachante de pedidos de objetos (*Object Request Broker* – ORB). O ORB, de uma forma geral, habilita a comunicação entre clientes e objetos, ativando de forma transparente os objetos solicitados que não estão em execução. Além disso, o ORB fornece uma interface que pode ser usada diretamente tanto por clientes quanto por objetos. As categorias de interface que usam os recursos do ORB são as seguintes:

- ***Serviços de Objetos - SO*** *(Object Services)*: são interfaces que oferecem serviços básicos usados por diversas aplicações de objetos distribuídos. Estas interfaces são consideradas, normalmente, parte da infraestrutura básica do núcleo de computação distribuída.
- ***Interfaces de Domínio - ID*** *(Domain Interfaces)*: são interfaces específicas, que a exemplo dos serviços de objeto, oferecem determinados serviços para as aplicações, só que específicos para cada área de interesse, como telecomunicações, por exemplo.
- ***Interfaces de Aplicação - IA*** *(Application Interfaces)*: representam as interfaces desenvolvidas especificamente para uma dada aplicação, que podem ser padronizadas e transformadas nas categorias anteriores caso se verifique sua larga utilização.

Os programas baseados em CORBA são compostos de componentes formados por vários objetos que suportam uma ou mais dessas categorias de interface da OMA, criando a estrutura (*framework*) da figura 3.1 a seguir.

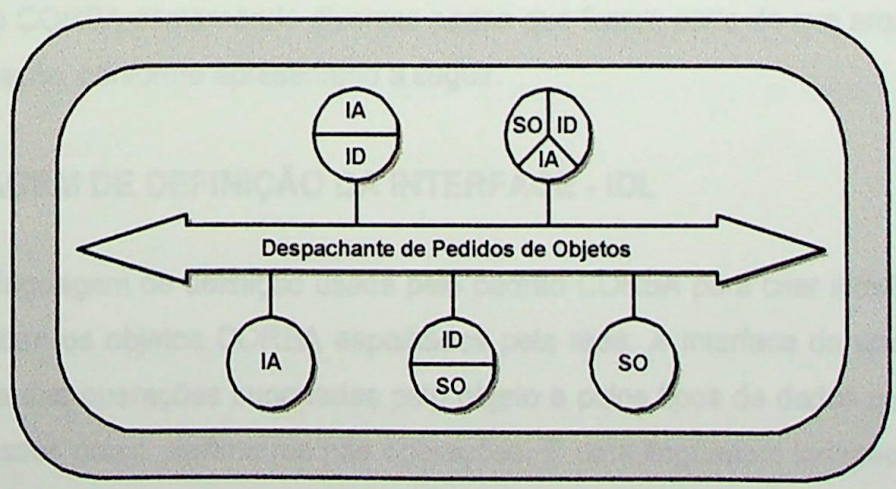

Figura 3.1 – Estrutura de Objetos OMA

3.3 CONCEITOS

Como toda tecnologia, CORBA [14] [15] possui sua própria terminologia, definindo conceitos e termos que são importantes para o entendimento do modelo. Os principais conceitos e termos do padrão CORBA são os seguintes:

- ***Objeto CORBA:*** é uma entidade virtual que pode ser localizada pelo ORB a partir de solicitações feitas por clientes.
- ***Objeto de Destino:*** é um objeto CORBA que é o alvo (destino) da invocação de um pedido feito por um cliente a partir de uma referência de objeto.
- ***Cliente:*** é uma entidade que invoca um pedido para um objeto CORBA.
- ***Servidor:*** é uma aplicação na qual um ou mais objetos CORBA existem para atenderem os pedidos de um ou mais clientes.
- ***Pedido:*** é a invocação de uma operação de um objeto CORBA feita por um cliente ou outro objeto CORBA.
- ***Referência de Objeto (OR):*** é um manipulador usado para identificar, localizar e endereçar um objeto CORBA, referindo-se exclusivamente a um único objeto CORBA.
- ***Servant:*** é uma entidade de uma linguagem de programação que implementa um ou mais objetos CORBA. Os *servants* fornecem a implementação de um objeto CORBA, transformando a entidade virtual em uma entidade concreta. Por isso, diz-se que os *servants* encarnam os objetos CORBA. Em C++, os *servants* correspondem a instâncias de objetos de uma determinada classe.

3.4 CARACTERÍSTICAS

O padrão CORBA compreende diversas partes que fazem parte da sua arquitetura e especificação, conforme apresentado a seguir.

3.4.1 LINGUAGEM DE DEFINIÇÃO DA INTERFACE - IDL

IDL é a linguagem de definição usada pelo padrão CORBA para criar interfaces que representam os objetos CORBA espalhados pela rede. A interface de um objeto é composta das operações suportadas pelo objeto e pelos tipos de dados que podem ser passados como parâmetros nas operações. É uma linguagem independente de qualquer outra linguagem de programação, possibilitando a interoperabilidade entre

aplicações desenvolvidas em diferentes linguagens. A implementação das interfaces fica a cargo dessas aplicações.

A interface a seguir apresenta um exemplo de interface CORBA definida de acordo com a linguagem IDL:

```
interface Funcionario {                    // Objeto CORBA
   string get_Nome();
   void set_Nome(in string nome);          // Operação da interface
   …

}

interface Gerente : Funcionario {
   …

}
```

A IDL suporta diversos tipos primitivos, como inteiros, strings, assim como tipos construídos, como estruturas.

Os argumentos das operações da IDL devem ter a direção declarada para que o ORB possa determinar o sentido das informações trocadas entre o cliente e o objeto de destino. O sentido é definido pelas seguintes palavras-chave (*keyword*):

- ***in***: o argumento é passado do cliente para o objeto de destino.
- ***out***: assim como para os valores de retorno, o argumento é passado do objeto de destino para o cliente.
- ***inout***: indica que o argumento é inicializado pelo cliente e enviado ao objeto de destino que pode modifica-lo retornando o novo valor para o cliente.

Uma característica das interfaces da IDL é a possibilidade do uso de herança, conforme a interface definida anteriormente. Além disso, todas as interfaces IDL derivam implicitamente da interface *Object* que fornece operações comuns a todos os objetos CORBA.

3.4.2 MAPEAMENTO DE LINGUAGEM

Por se tratar de uma linguagem de definição, a IDL não pode ser compilada ou interpretada em um programa executável. A IDL tem um tradutor que é quem transforma os códigos fontes escritos em IDL em uma outra linguagem de programação.

O trabalho do tradutor não é o de compilar e sim de mapear o arquivo fonte IDL para outro arquivo fonte em C++, por exemplo. Atualmente, existem mapeamentos para as linguagens C++, Java, Cobol, ADA, etc.

Por exemplo, em C++, as interfaces da IDL são mapeadas para classes e as operações para métodos dessa classe. Em Java, as interfaces são mapeadas para interfaces públicas do Java.

A compilação (tradução) da IDL, de acordo com o mapeamento de linguagem usado, gera os *stubs* e *skeletons* para cada interface definida. Os *stubs* e *skeletons* são códigos específicos ligados ao código do cliente e do servidor para a construção do aplicativo final. Eles são usados para esconder os detalhes e implementar a comunicação entre objetos locais e remotos.

A existência de múltiplos mapeamentos de linguagem possibilita o desenvolvimento de aplicativos de um sistema distribuído em diferentes linguagens. Dessa forma, enquanto um servidor pode ser desenvolvido em C++ para aumentar a eficiência do programa, os clientes podem ser desenvolvidos como *applets* do Java, de tal forma que eles possam ser carregados pela rede.

Esta é uma característica que demonstra o poder do padrão CORBA como uma tecnologia de integração para sistemas heterogêneos.

3.4.3 INVOCAÇÃO E DESPACHO DE OPERAÇÕES

Existem duas formas gerais para a invocação e o despacho de pedidos:

- *Estática*

  A forma estática utiliza *stubs* e *skeletons* que são compilados nas respectivas aplicações, de modo que a aplicação tenha conhecimento durante a compilação dos tipos e funções mapeadas a partir da descrição dos objetos remotos na IDL.

  O *stub* é uma função cliente que permite que uma invocação de pedido seja feita tal como uma simples chamada de função local. Em C++, um *stub* CORBA é um método de uma classe. O objeto C++ que suporta os métodos *stubs* é denominado procurador (*proxy*), pois ele representa o objeto remoto para a aplicação local.

Analogamente, o *skeleton* é uma função do lado servidor que permite que uma invocação de pedido recebida por um servidor seja despachada para o respectivo *servant* que implementa o objeto CORBA de destino.

- *Dinâmica*

  Esta forma constrói os *stubs* e *skeletons* em tempo de execução, a partir de informações de interfaces e tipos obtidas de serviços disponíveis na rede CORBA. O serviço de Repositório de Interfaces é um exemplo de serviço CORBA que fornece acesso em tempo de execução às definições da IDL.

3.4.4 ADAPTADOR DE OBJETOS

Os adaptadores de objeto, em CORBA, servem de ligação entre os *servants* e o ORB, adaptando a interface de um objeto para uma interface diferente esperada pelo solicitante. Em outras palavras, o adaptador de objetos permite que um solicitante faça invocações em um objeto sem conhecer a verdadeira interface do objeto.

Os adaptadores de objetos em CORBA preenchem três requisitos básicos que aliviam as responsabilidades do ORB, tornando-o mais leve e simples:

- Eles criam as referências de objetos, usadas pelos clientes para localizar os objetos.
- Eles garantem que cada objeto de destino seja encarnado por um *servant*.
- Eles direcionam os pedidos despachados pelo ORB servidor para os *servants* que encarnam os respectivos objetos de destino.

Em C++, os *servants* derivam das classes *skeleton* produzidas pela compilação das definições de interfaces da IDL. As operações são implementadas a partir da redefinição (*override*) das funções virtuais da classe base *skeleton*. Os *servants* são então registrados com o adaptador de objetos para que o mesmo possa direcionar os pedidos feitos pelos clientes para o respectivo *servant* que encarna o objeto de destino.

POA

Para resolver os problemas de portabilidade do lado servidor que existiam até então nas primeiras versões do padrão CORBA, foi introduzido um novo recurso a partir da

revisão 2.2 do CORBA: o Adaptador de Objetos Portável (*Portable Object Adpator*), ou POA.

O Adaptador de Objetos Portável fornece serviços fundamentais como a criação de objetos, registro de *servants* e despacho de pedidos. Sua especificação compreende diversas funções que permitem o desenvolvimento de aplicações servidoras escaláveis de alta performance, representando uma parte fundamental de toda aplicação CORBA.

3.4.5 PROTOCOLOS INTER-ORB

Para possibilitar a comunicação remota de diferentes ORBs através de um protocolo de transporte orientado à conexão, foi criado o Protocolo Geral Inter-ORB (GIOP), um protocolo abstrato que especifica as sintaxes de transferência e um conjunto padrão de formatos de mensagens usados na comunicação.

O Protocolo Internet Inter-ORB (IIOP), em especial, especifica como o protocolo GIOP é implementado através do padrão TCP/IP.

Além disso, o próprio formato das referências de objetos deve seguir um padrão também. O formato padrão, conhecido como Referência de Objeto Interoperável (IOR), permite armazenar informações sobre qualquer tipo de protocolo Inter-ORB suportado.

No caso do IIOP, uma IOR contém o nome do *host*, o número da porta TCP/IP, e uma chave de objeto que identifica o objeto de destino para a respectiva combinação de *host* e número de porta.

3.5 INVOCAÇÃO DE PEDIDOS

Para que um cliente possa acessar os serviços oferecidos por um determinado objeto, o cliente precisa obter antes uma referência de objeto para o objeto em questão. A referência de objeto atua como um manipulador que identifica unicamente o objeto de destino, encapsulando todas as informações exigidas pelo ORB para enviar a mensagem para o destino correto.

Sempre que um cliente invoca uma operação a partir de uma referência de objeto, o ORB realiza as seguintes tarefas:

- Localiza o objeto de destino
- Ativa a aplicação servidora caso o servidor não esteja rodando
- Transmite os argumentos da chamada para o objeto
- Ativa um *servant* para o objeto se necessário
- Aguarda o término do pedido
- Retorna qualquer parâmetro definido como *out* e *inout* e o valor de retorno para o cliente caso a chamada complete com sucesso
- Retorna uma exceção ao cliente caso a chamada falhe

O mecanismo de invocação de pedidos do CORBA é completamente transparente para o cliente, que executa a chamada a um objeto remoto de forma parecida com a invocação usual de método de um objeto local, como no caso de C++.

Dessa forma, a invocação de pedidos em uma rede CORBA atende vários requisitos para o desenvolvimento de aplicativos em um sistema distribuído real, como:

- Transparência de Localização
- Transparência do Servidor
- Independência de Linguagem
- Independência de Implementação
- Independência de Arquitetura
- Independência de Sistema Operacional
- Independência de Protocolo
- Independência de Transporte ou Tecnologia de Rede Utilizada

3.5.1 SEMÂNTICAS DA REFERÊNCIA DE OBJETO

As referências de objetos CORBA possuem características similares a ponteiros de objetos em C++, a não ser pela capacidade de endereçamento distribuído que possibilita identificar objetos implementados em processos diferentes (possivelmente em outras máquinas) assim como objetos implementados no próprio espaço de endereço do cliente.

- *Cada referência identifica exatamente um único objeto*

  Enquanto o objeto referenciado existir, a referência de objeto irá denotar sempre o mesmo objeto CORBA. Quando o objeto de destino for destruído, sua

referência não tem mais utilidade, o que garante que a mesma não pode ser usada mais tarde para acessar outro objeto CORBA.

- *Várias referências distintas podem referenciar o mesmo objeto*

  Apesar de duas ou mais referências possuírem conteúdos diferentes, é possível que as mesmas denotem o mesmo objeto, já que uma referência de objeto não é a mesma coisa que uma identidade de objeto, pois o conteúdo diferente inclui outras informações além da própria identidade do objeto.

- *Referências podem ser nulas*

  Uma referência nula aponta para lugar nenhum, da mesma forma que um ponteiro nulo (*null*) em C++.

  Este tipo de referência é útil para indicar que algo não existe ou não foi encontrado, como um objeto ou parâmetro.

- *Referências podem ficar pendentes*

  Como o padrão CORBA não define mecanismos para o controle das referências de objetos criadas, quando um objeto CORBA é destruído, sua referência fica perdida, pois o objeto de destino não existe mais. Contudo, os mecanismos de controle necessários podem ser implementados pelas aplicações, como no caso da operação *non_existent*( ), suportada por todos os objetos, que pode ser invocada pelos clientes para verificar se o objeto referenciado ainda existe.

- *Referências são opacas*

  Para possibilitar a interoperabilidade entre ORBs de diferentes fornecedores, o padrão CORBA criou um tipo de estrutura para a OR contendo informações padronizadas e proprietárias, de tal forma que as aplicações não possam verificar a representação de uma OR, a não ser através de uma interface padronizada que manipula as informações básicas.

  Dessa forma, o padrão CORBA permite que novas informações sejam agregadas a uma OR, como novos protocolos de comunicação, sem comprometer os códigos fonte existentes.

- *Referências são fortemente tipadas*

  Para forçar a segurança de tipo, o padrão CORBA inclui em cada OR uma indicação da interface suportada pela referência, de tal forma que o ORB possa verificar, em tempo de execução, se a operação invocada é suportada ou não pelo objeto de destino, lançando uma exceção no caso de falha.

- *Referências suportam ligação tardia*

  Polimorfismo e ligação tardia (*late binding*) em CORBA funcionam de forma transparente através do fio para objetos remotos da mesma forma que para objetos locais em C++.

  Com isso, um cliente pode tratar uma referência de um objeto derivado como se fosse uma referência para o objeto base.

- *Referências podem ser persistentes*

  Uma referência de objeto em CORBA pode ser transformada em uma *string* e salva em disco, para mais tarde ser recuperada e usada normalmente.

- *Referências podem ser interoperáveis*

  O formato padrão das referências de objetos especificado pelo padrão CORBA permite a interoperabilidade entre ORBs de diferentes fornecedores. Por esta razão, estas referências de objetos padrão são também conhecidas como Referências de Objetos Interoperáveis (IOR).

3.5.2 AQUISIÇÃO DE REFERÊNCIAS

Como a referência de objeto é o único meio possível para localizar um objeto remoto, o cliente precisa obter, de alguma forma, a referência do objeto CORBA desejado. Basicamente, o servidor que implementa o objeto remoto pode:

- Retornar uma referência como o resultado de uma operação.
- Publicar uma referência em algum serviço bem conhecido, tal como o Serviço de Nomes (*Naming Service*) ou o Serviço de Negócios (*Trading Service*).
- Espalhar uma referência como uma *string* através de um arquivo.

- Transmitir uma referência através de outras vias, como e-mail ou página da *Web*.

O meio mais utilizado pelos clientes é o retorno de uma referência como o resultado de uma operação. A operação, neste caso, é invocada em um objeto que funciona como um objeto *Web*, já que o cliente pode navegar pelo objeto para obter a referência desejada.

De qualquer forma, independente da origem das referências de objetos, elas são sempre criadas pelo ORB servidor em tempo de execução em favor do cliente. Esta abordagem permite esconder do cliente a representação interna da referência.

3.5.3 CONTEÚDO DE UMA REFERÊNCIA DE OBJETO

As informações contidas em uma IOR são divididas em três partes principais:

- *ID de Repositório*

  O ID de repositório é uma *string* usada pelo ORB para localizar o Repositório de Interfaces que contem uma descrição detalhada da interface suportada pela referência de objeto correspondente.

- *Informação de Conexão*

  Todas as informações exigidas pelo ORB para estabelecer uma conexão com o servidor que implementa o objeto de destino estão contidas neste campo. São informações específicas sobre o tipo de protocolo a ser usado e endereçamento físico apropriado para um determinado transporte.

  Particularmente para o IIOP, a informação de conexão contém um nome de domínio da Internet ou endereço IP e um número de porta TCP. No caso de existirem várias opções, o ORB escolhe de modo transparente o protocolo mais apropriado.

  Contudo, existem casos em que este campo pode conter o endereço de um repositório de implementação usado pelo ORB para localizar o servidor correto.

  Dessa forma, o processo servidor pode migrar de máquina em máquina na rede CORBA sem comprometer as referências existentes em posse dos clientes.

- *Chave do Objeto*

  A chave do objeto, ao contrário das informações anteriores, é proprietária. Portanto, as informações nela contida só podem ser interpretadas pelo ORB e pelo adaptador de objetos do lado do servidor que criou a mesma, de modo que o objeto de destino possa receber os pedidos dos clientes.

A figura 3.2 a seguir ilustra a estrutura básica de uma referência de objeto.

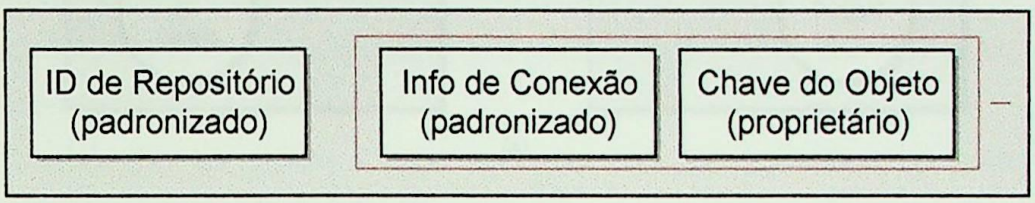

Figura 3.2 – Conteúdo da Referência de Objeto

A combinação da informação de conexão (Info de Conexão) com a chave do objeto, denominada de perfil (*profile*), pode aparecer diversas vezes em uma IOR, permitindo que o ORB escolha dinamicamente o protocolo mais apropriado de acordo com as características do cliente e do servidor.

3.5.4 REFERÊNCIAS E *PROXIES*

Quando um cliente obtém uma referência de objeto CORBA, uma instância de um objeto *proxy* é criada no espaço de endereço do processo cliente. O objeto *proxy* fornece ao cliente uma interface para o objeto de destino, que é específica para o tipo de objeto que está sendo acessado.

A classe *proxy* é gerada a partir da respectiva interface definida na IDL, implementando o *stub* usado pelo cliente para despachar os pedidos. Esta abordagem garante a segurança de tipo: o cliente só pode invocar uma operação quando ele tiver um *proxy* do tipo correto, pois somente este *proxy* possui o método necessário para enviar o pedido.

A figura 3.3 a seguir ilustra o mecanismo de interação que ocorre entre aplicações locais e remotas.

O pedido do cliente será enviado sempre pelo *proxy* para o *servant* apropriado. A transparência de localização é garantida pelo ORB, que decide entre a transmissão

da mensagem pela rede (a) ou diretamente (b) através da chamada de um método (*stub*) C++, por exemplo.

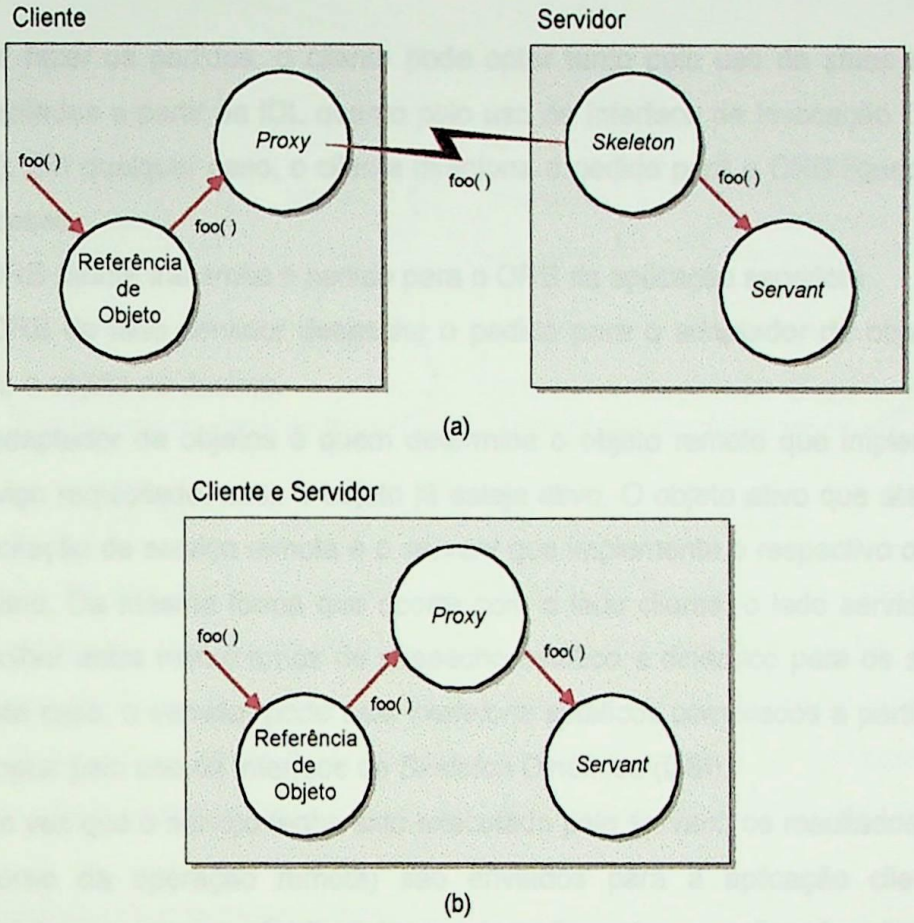

Figura 3.3 – (a) *Proxy* local com Objeto Remoto (b) *Proxy* e Objeto locais

3.6 MODELO

A figura 3.4 a seguir apresenta o modelo de interação entre um cliente e um servidor no padrão CORBA.

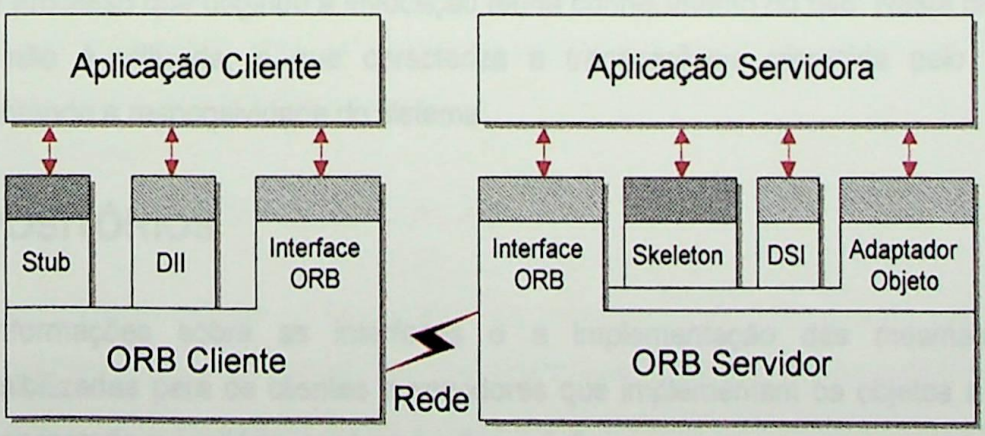

Figura 3.4 – Modelo CORBA

A aplicação cliente faz os pedidos à aplicação servidora que executa os serviços requisitados. A interação entre as aplicações através dos diversos componentes CORBA ocorre da seguinte forma:

1. Para fazer os pedidos, o cliente pode optar tanto pelo uso de *stubs* estáticos compilados a partir da IDL quanto pelo uso da Interface de Invocação Dinâmica (DII). Em qualquer caso, o cliente direciona o pedido para o ORB ligado ao seu processo.
2. O ORB cliente transmite o pedido para o ORB da aplicação servidora.
3. O ORB do lado servidor despacha o pedido para o adaptador de objetos que criou o objeto de destino.
4. O adaptador de objetos é quem determina o objeto remoto que implementa o serviço requisitado, caso o objeto já esteja ativo. O objeto ativo que atenderá a solicitação de serviço remota é o *servant* que implementa o respectivo objeto de destino. Da mesma forma que ocorre com o lado cliente, o lado servidor pode escolher entre mecanismos de despacho estático e dinâmico para os *servants*. Neste caso, o servidor pode usar *skeletons* estáticos compilados a partir da IDL ou optar pelo uso da Interface de *Skeleton* Dinâmica (DSI).
5. Uma vez que o serviço tenha sido executado pelo *servant*, os resultados obtidos (retorno da operação remota) são enviados para a aplicação cliente que requisitou o serviço, finalizando a interação entre a aplicação cliente e a servidora.

Qualquer pedido requisitado por um objeto local irá seguir sempre a mesma seqüência. Para aplicações que estejam executando na mesma máquina, a invocação dos pedidos locais é feita diretamente através do ORB para o objeto local. O objeto local pode residir no mesmo processo ou em um processo diferente, sem que o processo que originou a invocação tenha conhecimento do fato. Neste caso, a rede não é utilizada, o que caracteriza a transparência oferecida pelo ORB, aumentando a responsividade do sistema.

3.7 REPOSITÓRIOS

As informações sobre as interfaces e a implementação das mesmas são disponibilizadas para os clientes e servidores que implementam os objetos a partir dos serviços de repositório, conforme a figura 3.5.

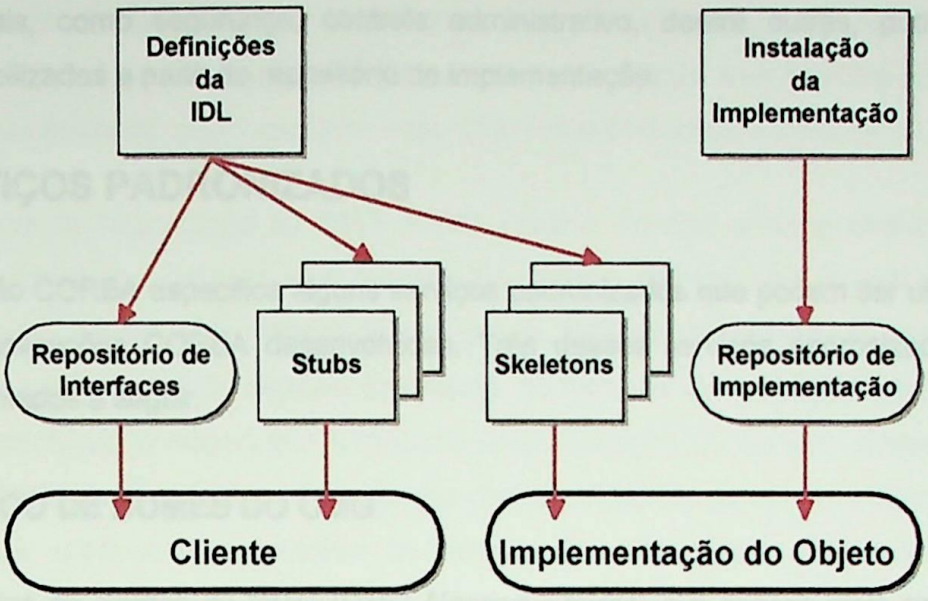

Figura 3.5 – Repositórios de Interface e de Implementação

As informações das interfaces definidas na IDL e/ou no Repositório de Interfaces são usadas para gerar os *stubs* do lado cliente e os *skeletons* do lado servidor, de forma estática ou dinâmica, respectivamente. A informação sobre a implementação dos objetos é fornecida no instante da instalação do servidor e armazenada no Repositório de Implementação para uso durante a entrega de pedidos dos clientes.

3.7.1 REPOSITÓRIO DE INTERFACES

O Repositório de Interfaces é um serviço que fornece objetos persistentes que representam a informação da IDL em uma forma disponível para uso em tempo de execução. A partir dessas informações, o ORB pode encontrar os novos objetos que suportam determinada interface desconhecida até então para os aplicativos compilados, e determinar quais operações são válidas para o objeto, para que a invocação dos pedidos possa ser feita corretamente. Além disso, outras informações relevantes associadas com as interfaces podem ser colocadas no repositório de interfaces, como informações de *debug*.

3.7.2 REPOSITÓRIO DE IMPLEMENTAÇÃO

As informações contidas no Repositório de Implementação são usadas pelo ORB para localizar e ativar as implementações dos objetos CORBA, sendo específicas do ORB empregado. Da mesma forma que para o repositório de interfaces, informações

adicionais, como segurança, controle administrativo, dentre outras, podem ser disponibilizadas a partir do repositório de implementação.

3.8 SERVIÇOS PADRONIZADOS

O padrão CORBA especifica alguns serviços padronizados que podem ser utilizados pelas aplicações CORBA desenvolvidas. Três desses serviços padronizados são apresentados a seguir.

3.8.1 SERVIÇO DE NOMES DO OMG

O Serviço de Nomes do OMG (OMG *Naming Service*) é o serviço mais simples e mais básico dentre todos os serviços padronizados. Ele fornece um mapeamento de nomes para referências de objetos: dado um nome, o serviço retorna uma referência de objeto armazenada para aquele nome. Este serviço é parecido com o Serviço de Nomes de Domínio da Internet (DNS), que traduz nomes de domínios da Internet, tais como pet.efei.br, em endereços IP, tais como 192.168.1.2.

O Serviço de Nomes oferece algumas vantagens para seus clientes, que neste caso, são as próprias aplicações CORBA desenvolvidas:

- Os clientes podem usar nomes significativos para os objetos ao invés de lidarem com referências de objetos convertidas em string (*stringified object references*).
- Modificando o valor de uma referência publicada por um nome, é possível oferecer uma implementação diferente de uma interface aos clientes CORBA sem a necessidade de mudança do código fonte do cliente. Os clientes usam o mesmo nome, mas recebem uma referência diferente.
- O Serviço de Nomes pode ser usado para resolver o problema de acesso as referências iniciais das aplicações. A publicação dessas referências no Serviço de Nomes elimina a necessidade de armazena-las em arquivos como referências convertidas em string.

3.8.2 SERVIÇO DE NEGOCIAÇÃO DO OMG

O Serviço de Nomes permite que um cliente localize uma referência de objeto a partir de um nome simbólico. Este mecanismo é suficiente para o cliente localizar um objeto quando o cliente conhece exatamente o objeto que ele quer usar.

No caso do cliente ter somente uma idéia do tipo de objeto que ele necessita, ou seja, não possuir todas as informações exigidas para fazer uma escolha precisa, o cliente necessita de um mecanismo mais dinâmico e flexível para localizar objetos.

O Serviço de Negociação do OMG (*OMG Trading Service*) oferece exatamente a funcionalidade que permite aos clientes localizarem objetos com a ajuda de um negociante (*trader*). Da mesma forma que o Serviço de Nomes, o negociante armazena referências de objetos. Entretanto, no lugar de armazenar um nome para cada referência, o negociante armazena uma descrição do serviço oferecido por cada referência. Os clientes realizam uma procura dinâmica de serviços baseada em perguntas relativas às descrições do serviço. Este mecanismo, conhecido como ligação dinâmica (*late binding*), possibilita um mapeamento mais dinâmico dos critérios de seleção de referências de objetos.

3.8.3 SERVIÇO DE EVENTOS DO OMG

Aplicações convencionais são baseadas normalmente em invocações de pedidos síncronas. Através de pedidos síncronos, um cliente ativamente invoca pedidos em servidores passivos; após o envio do pedido, o cliente bloqueia a espera da resposta. Neste caso, os clientes estão cientes do destino dos pedidos, pois eles mantêm referências dos objetos de destino, e cada pedido tem um destino único indicado pela referência do objeto usada para invocar o pedido. Se o objeto de destino não existir mais ou por algum outro motivo não puder ser acessado, o cliente que invocou o pedido recebe uma exceção.

Entretanto, para algumas aplicações distribuídas, o modelo de invocações de pedidos síncronas é muito restrito apesar da sua utilidade. Estas aplicações exigem normalmente um meio de separar os fornecedores de informações dos consumidores interessados nelas.

O Serviço de Eventos do OMG (OMG *Event Service*) fornece suporte para comunicações separadas (*decoupled*) entre os objetos. Ele permite que os fornecedores enviem mensagens para um ou mais consumidores com uma única chamada. Além disso, os fornecedores que usam o Serviço de Eventos não precisam ter conhecimento de qualquer dos consumidores interessados em suas mensagens. O Serviço de Eventos atua como um intermediador que separa fornecedores de consumidores. A implementação do Serviço de Eventos protege ainda os

fornecedores das exceções resultantes de qualquer um dos objetos dos consumidores que não podem ser acessados ou que se comportam de maneira estranha.

3.8.4 SERVIÇOS ADICIONAIS

Diferentes implementações podem oferecer serviços adicionais, como o ORBacus, que fornece uma Interface de Comunicações Aberta, OCI, usada, dentre outras coisas, para obter informações sobre uma aplicação CORBA, como o nome do host e a porta de comunicação de um servidor CORBA, por exemplo.

CAPÍTULO 4

MODELO DE SIMULAÇÃO

4.1 INTRODUÇÃO

A partir da reestruturação sofrida nos últimos anos pelo mercado brasileiro de energia elétrica, com a criação de novos agentes que atuam em conjunto com o MAE - Mercado Atacadista de Energia, para formar o novo modelo do mercado de energia elétrica brasileiro, a comercialização de energia elétrica passa a ter um caráter muito mais dinâmico.

Para se adaptar às essas novas características de comercialização da energia elétrica, torna-se importante a integração desses agentes para possibilitar a troca de informações entre os mesmos. Contudo, para assegurar a perfeita operação do MAE, é necessário criar um sistema eficiente e seguro.

A tecnologia de Objetos Distribuídos abordada neste trabalho, baseada na arquitetura proposta pelo OMG, ou seja, o padrão CORBA, em função de suas próprias características e utilização, destaca-se, a nosso ver, como uma solução apropriada para o desenvolvimento de um sistema computacional distribuído que atenda os requisitos funcionais e de segurança do ambiente do MAE [16].

O modelo de sistema distribuído proposto concentra-se essencialmente no emprego da tecnologia que suporta o padrão CORBA, considerando somente algumas idéias básicas do atual modelo de mercado de energia elétrica.

4.2 MERCADO DE ENERGIA ELÉTRICA

A nova estrutura do mercado de energia elétrica no Brasil [17] se caracteriza por um novo modelo de "desverticalização" que implica na separação do produto, dos serviços e da comercialização. O produto, neste caso, é a energia elétrica, disponibilizada pelos serviços de transmissão e distribuição. Com as privatizações, esta nova estrutura de mercado livre, como é o caso de qualquer outro produto, passa a ser controlada pelas próprias leis de mercado. O mercado livre se apresenta como uma estrutura que permite a vendedores e compradores a escolha dos

melhores negócios e produtos, facilitando a realização de transações entre os agentes.

Contudo, por se tratar de um mercado estratégico, existe a necessidade de regulamentação para impor limites e monitorar a atividade como um todo. Para tanto, foram criados pelo governo brasileiro alguns agentes com funções próprias e específicas.

Os principais agentes criados são apresentados a seguir:

- *ANEEL* – Agência Nacional de Energia Elétrica: responsável pela regulação e fiscalização da indústria de energia elétrica.
- *ONS* – Operador Nacional do Sistema: responsável basicamente pela coordenação, supervisão e controle do sistema elétrico brasileiro como um todo.
- *MAE* – Mercado Atacadista de Energia Elétrica: responsável pelas transações de compra e venda de energia elétrica em todo o país.

Além destes, existem ainda outros agentes, como o agente comercializador, a ASMAE, que é a administradora do MAE, as próprias concessionárias de energia elétrica, dentre outros [18].

A figura 4.1 a seguir [19] apresenta o modelo comercial dentro desta nova concepção do mercado de energia elétrica.

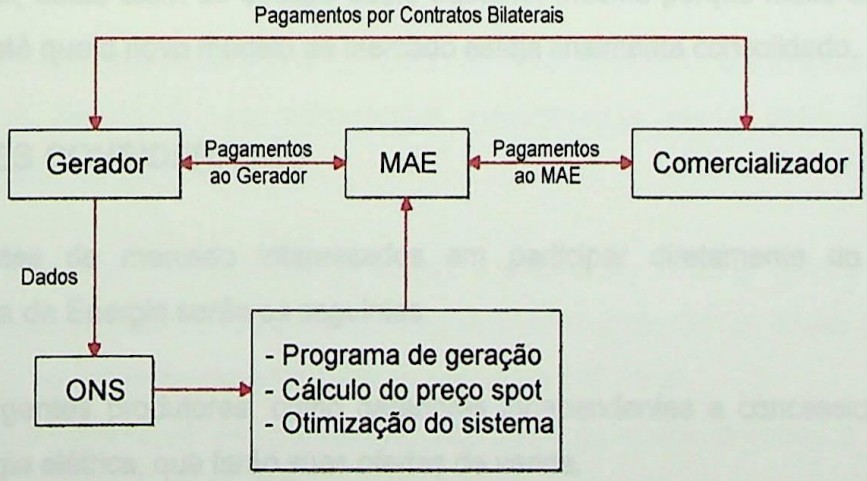

Figura 4.1 – Modelo Comercial

De acordo com esta figura, observa-se a importância do agente Operador Nacional do Sistema (ONS). O ONS calculará um preço que represente o custo marginal do

sistema ou preço *spot*, que implicará no equilíbrio entre a oferta e a demanda. Além disso, o ONS definirá a programação da geração e promoverá a otimização do sistema, de modo a obter o menor custo, dentro de padrões técnicos e critérios de confiabilidade.

Os contratos bilaterais representam os contratos firmados fora do mercado *spot*, pela livre negociação de preços, prazos e quantidades. A energia não coberta pelos contratos bilaterais será a energia negociada no mercado de curto prazo, ou mercado *spot*, de acordo com o preço definido no MAE.

4.3 MODELO DE SIMULAÇÃO PROPOSTO

Para mostrar as vantagens que a tecnologia de objetos distribuídos, utilizando o padrão CORBA, traz para o desenvolvimento e a operação de um sistema distribuído, serão apresentados os pontos considerados mais essenciais para a implementação de um sistema simples, porém completo.

Os principais agentes que compõem o mercado serão representados por simples nós de uma rede. A interação dos agentes será feita a partir de um modelo simplificado baseado em considerações particulares e mesmo hipotéticas. O modelo de mercado atual servirá apenas de base para a definição do modelo de simulação proposto.

Os detalhes de operação do mercado, como a implementação real dos serviços oferecidos, estão além do escopo deste trabalho, mesmo porque muito ainda deve ser feito até que o novo modelo de mercado esteja finalmente consolidado.

4.3.1 AGENTES CONSIDERADOS

Os agentes de mercado interessados em participar diretamente do Mercado Atacadista de Energia serão os seguintes:

- Os agentes produtores, como geradores independentes e concessionárias de energia elétrica, que farão suas ofertas de venda.
- Os consumidores, como indústrias e organizações, interessadas na compra de energia elétrica.

Os agentes transportadores e os comercializadores foram retirados de propósito por questões de simplificação.

Particularmente para o modelo de simulação proposto, os agentes responsáveis pelo funcionamento e controle do Mercado Atacadista de Energia, como o ONS e o MAE, serão considerados da seguinte forma:

- *OSMAE*

  O OSMAE, Operador do Sistema do MAE, fica responsável pelo controle das informações pertinentes que compõem o mercado *spot* (*pregão*) e pela própria liquidação do MAE, notificando os agentes participantes com as respectivas informações. As informações do pregão devem contemplar os dados dos agentes participantes, indicando o nome e a oferta, e o resultado do fechamento do pregão, indicando o valor do despacho de cada gerador (opcionalmente das linhas) e a tarifa de cada carga.

- *ASMAE*

  O ASMAE, Administrador do Sistema do MAE, fica responsável pela administração dos agentes participantes do Mercado Atacadista de Energia, mantendo basicamente uma lista atualizada dos agentes registrados. Ele representa um ponto de acesso comum e conhecido do sistema, usado por todos os agentes, incluindo o OSMAE e o ONS. As informações dos registros de cada agente devem incluir pelo menos o nome e o endereço de rede do agente.

- *ONS*

  O ONS fica responsável pela operação do sistema elétrico, otimizando o sistema elétrico resultante. As informações consideradas neste caso dividem-se basicamente em duas partes:

  - Os dados técnicos dos geradores e cargas do sistema elétrico enviados pelos respectivos agentes produtores e consumidores (os dados das linhas de transmissão deverão estar previamente definidos).
  - O montante de energia despachada pelos geradores e linhas e os custos marginais (preço *spot*) em [R$/MWh], calculados por um algoritmo de programação linear.

A ANEEL, a exemplo dos agentes transportadores e os comercializadores, foi desconsiderada para simplificar o modelo.

É importante ressaltar, mais uma vez, que este modelo se trata de um modelo hipotético, apesar das semelhanças. Os agentes considerados acima foram assim definidos por questão de simplicidade, com o único objetivo de descrever os agentes do modelo de simulação proposto. Embora alguns agentes, como o ONS e o ASMAE, se confundam com os agentes reais de mercado, aqui eles possuem um papel mais específico e bem definido que agrega valor ao modelo de simulação proposto, não servindo de base para comparações.

4.3.2 FUNCIONAMENTO

A simulação do funcionamento do Mercado Atacadista de Energia – MAE, irá envolver 3 empresas geradoras de energia elétrica, 3 consumidores, além do OSMAE, ASMAE e ONS. A figura 4.2 a seguir apresenta um esboço do modelo de simulação proposto, mostrando os principais elementos envolvidos na nova concepção do sistema energético brasileiro.

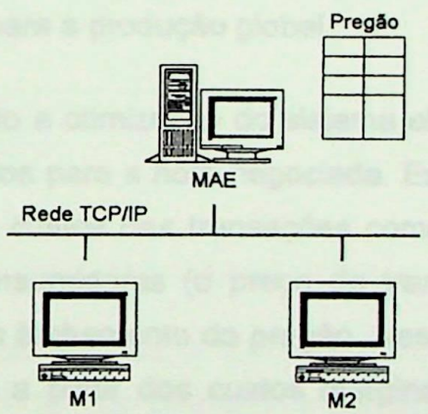

Figura 4.2 – Modelo de Simulação do MAE

Neste modelo, a simulação do ambiente do MAE é feita basicamente a partir de três microcomputadores conectados a uma rede TCP/IP local. O micro M1 representará os produtores. O micro M2 representará os consumidores. O terceiro micro, por sua vez, representará o OSMAE, ASMAE e ONS.

Os produtores terão que elaborar uma programação horária de geração a ser enviada ao ONS. Esta informação a ser passada para o ONS deverá conter a previsão de geração de energia para a hora negociada, em [MW], juntamente com o custo do gerador, em [R$/MWh].

Simultaneamente, enquanto o ONS recebe os dados dos produtores, os consumidores estarão elaborando e enviando as suas respectivas previsões de

demanda horária também. Neste caso, a informação passada ao ONS deverá conter somente a previsão de consumo da hora negociada, em [MW].

Quando o ONS tiver recebido os dados técnicos de todos os geradores e cargas, o ONS realizará então a otimização do sistema elétrico resultante da hora negociada para calcular o fornecimento real de cada gerador e o custo da demanda de cada barra do sistema.

Para esta tarefa de otimização, o ONS usará um algoritmo de programação linear baseado no método *simplex* de otimização. Este algoritmo tem como característica principal encontrar o melhor ponto de operação do sistema como um todo, de forma a minimizar os custos de produção.

Além disso, o algoritmo permite estabelecer um rateio entre os componentes do sistema de forma justa, de acordo com a teoria marginalista [20] que propõe que cada participante de uma associação receba de forma *proporcional* à *importância* dos *recursos* que contribui para a produção global.

Depois que o ONS tiver feito a otimização do sistema elétrico, ele deverá enviar ao OSMAE os resultados obtidos para a hora negociada. Esses dados servem de base para o OSMAE calcular os custos das transações comerciais das ofertas entre as empresas produtoras e consumidoras (o preço do transporte pode ser calculado também) antes de realizar o fechamento do pregão. Neste caso, o preço da energia será calculado diretamente a partir dos custos marginais obtidos no algoritmo de programação linear.

Com o fechamento do pregão, o OSMAE enviará aos produtores a respectiva geração da hora negociada juntamente com o montante a ser faturado, ou seja, o valor a ser recebido pelo produtor.

Cada consumidor, também, deverá receber o respectivo valor de energia negociado juntamente com o custo de fornecimento de energia a ser pago pelo consumidor (para as transportadoras de energia, a informação recebida deve contemplar a potência a ser transmitida juntamente com o montante a ser faturado - pedágio).

O ciclo de simulação reinicia na hora seguinte, quando os novos lances de oferta, contendo a nova programação horária de geração e consumo, forem enviados ao sistema do MAE.

4.3.3 SISTEMA ELÉTRICO

O sistema elétrico considerado para o modelo de simulação é o sistema térmico descrito na figura 4.3 [19] a seguir.

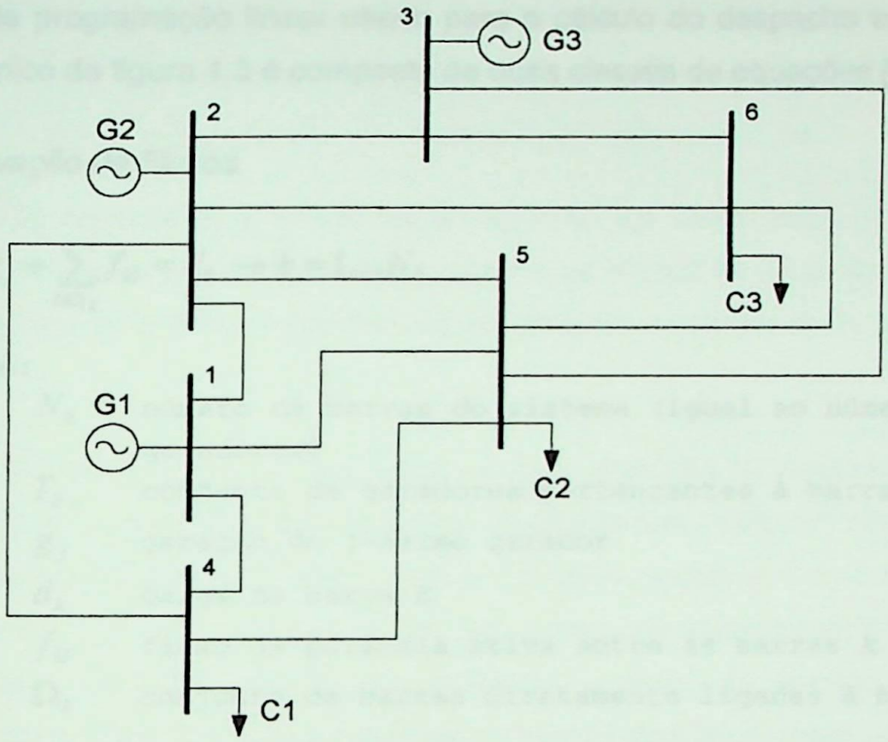

Figura 4.3 – Sistema Elétrico

Neste sistema, os geradores G1, G2 e G3 representam os agentes produtores, enquanto as cargas C1, C2 e C3 representam os agentes consumidores conectados ao sistema elétrico. As barras de 1 a 6 representam os pontos de conexão ou nós do sistema elétrico. As linhas de transmissão, por sua vez, são identificadas a partir das barras conectadas, como $L_{1\text{-}2}$, que representa a linha de transmissão que conecta a barra 1 à barra 2.

Dados dos Componentes do Sistema

Para os geradores, serão considerados três dados básicos: a barra de conexão do gerador, a potência máxima (ou capacidade), em [MW], e o respectivo custo de geração, em [R$/MWh]. No caso das cargas, a barra de conexão e a potência da mesma são suficientes. As linhas de transmissão do sistema, em particular, deverão estar definidas previamente com os dados de reatância e capacidade de cada linha, em [Ω] e [MW], respectivamente, juntamente com as barras de origem e destino.

Os dados de base do sistema, a saber, *Potência de Base* e *Tensão de Base*, deverão estar previamente definidos no sistema também.

4.3.4 PROGRAMAÇÃO LINEAR

O modelo de programação linear usado para o cálculo do despacho econômico do sistema térmico da figura 4.3 é composto de duas classes de equações [6]:

- *Conservação de Fluxos*

$$\sum_{j \in T_K} g_j + \sum_{l \in \Omega_K} f_{kl} = d_k \rightarrow k = 1 \ldots N_b$$

sendo:

N_b número de barras do sistema (igual ao número de geradores)

T_k conjunto de geradores pertencentes à barra K

g_j geração do j-ésimo gerador

d_k carga na barra K

f_{kl} fluxo de potência ativa entre as barras k e l

Ω_k conjunto de barras diretamente ligadas à barra k

- *Representação da Susceptância*

$$f_{kl} = \gamma_{kl} \cdot \Delta\theta_{kl}$$

sendo:

$$\Delta\theta_{kl} = \theta_k - \theta_l$$

θ_k ângulo de tensão na barra k

γ_{kl} susceptância do circuito k-l

O despacho econômico, propriamente dito, para um sistema puramente térmico, é feito a partir da minimização dos custos de produção conforme a representação a seguir:

$$z = \min \sum_{j=1}^{N_t} c_j \cdot g_j \qquad \text{Custo Marginal}$$

s.a.

$$\sum_{j \in T_k} g_j + \sum_{l \in \Omega_k} f_{kl} = d_k \qquad \pi d_k \qquad (1)$$

$$f_{kl} = \gamma_{kl} \cdot \Delta\theta_{kl} \qquad \pi x_{kl} \qquad (2)$$

$$g_j \leq \overline{g}_j \qquad \pi g_j \qquad (3)$$

$$f_{kl} \leq \overline{f}_{kl} \qquad \pi f_{kl} \qquad (4)$$

para $k = 1 \ldots N_b$

$l \in \Omega_k$

sendo:

c_j custo de geração da unidade j

$\overline{g}_j$ capacidade de geração da unidade j

$\overline{f}_{kl}$ capacidade de transmissão da linha kl

A equação (1) representa a conservação de fluxo em cada barra. A equação (2) corresponde à segunda lei de *Kirchoff*, enquanto as equações (3) e (4) representam, respectivamente, os limites na capacidade de geração e nos fluxos de potência ativa dos circuitos.

Os multiplicadores de *Lagrange* associados a cada restrição do problema na solução ótima, conhecidos no método *simplex* como *multiplicadores simplex*, representam os *índices de sensibilidade* do sistema, ou seja, o custo marginal. Como estes índices são expressos em [$/unidade do recurso], eles podem ser interpretados como preços unitários destes recursos, constituindo, portanto, o preço da energia no mercado livre, ou mercado *spot*. Em outras palavras, o método *simplex* fornece uma indicação da importância de cada recurso na produção global.

Dessa forma, o multiplicador simplex πd_k que representa a tarifa em cada barra, e que por sua vez constitui a tarifa da carga conectada à barra, pode ser usado para calcular o valor total da energia elétrica a ser pago pelo consumidor, que nada mais é que o produto da tarifa pela demanda:

$$L(d) = \pi d_k \cdot d_k$$

Analogamente, pode-se obter os custos da geração e transmissão:

$$L(\overline{g}_j) = \pi g_k \cdot \overline{g}_k$$

$$L(l_{kl}) = (\pi d_k - \pi d_l) \cdot f_{kl}$$

No caso da transmissão, o custo indica que a linha *kl* compra f_{kl} MW da barra *k* ao preço de mercado πd_k, e vende os mesmos f_{kl} MW para a barra *l* ao preço de mercado πd_l, ou seja, a diferença entre os custos de compra e a renda da venda constitui a tarifa de transmissão, denominada de pedágio.

Algoritmo

O algoritmo (Anexo I) usado para o cálculo do problema de otimização do modelo de programação linear descrito anteriormente deverá ser desenvolvido em Matlab e integrado ao ambiente de programação Visual C++. A interface usada pelo programa C++ será definida pela seguinte função de chamada do programa m-File:

```
function [x, lambda] = plinear(NB, NG, CG, PG, C, NL, R, CL)
```

Os parâmetros do sistema elétrico passados para o algoritmo são os seguintes:

- *NB*, que define o número total de barras.
- *NG*, que define o número total de geradores.
- *CG*, que define o custo dos geradores.
- *PG*, que define a potência máxima disponível (ou capacidade) dos geradores.
- *C*, que define a potência das cargas.
- *NL*, que define o número total de linhas.
- *R*, que define o valor da reatância das linhas de transmissão.
- *CL*, que define o valor da capacidade máxima das linhas de transmissão.

Os valores de potência e reatância devem estar em p.u., nas bases de 138 kV e 100MVA. O resultado obtido é composto de duas variáveis:

- *x*, que contêm os valores do despacho econômico.
- *lambda*, que contêm os valores dos custos marginais associados às barras, às linhas e aos geradores.

4.3.5 SISTEMA DISTRIBUÍDO

Para simular o modelo do mercado virtual, deve-se criar um sistema distribuído que se assemelhe o máximo possível com a situação prevista de operação real do MAE, ou seja, o mais heterogêneo possível.

Para tanto, as máquinas utilizadas para o sistema devem ter diferentes sistemas operacionais, como Windows 98, Windows NT e Linux [21]. Os aplicativos devem ser desenvolvidos em C++ e Java, utilizando-se ORBs abertos, como o ORBacus [22] [23], e ORBs proprietários, como o Visibroker [24], da Borland.

CAPÍTULO 5

PROCESSO DE DESENVOLVIMENTO DE SOFTWARE ORIENTADO A OBJETO

Engenharia de software pode ser categorizada de acordo com: classes, técnicas e sistemas. As classes representam os blocos básicos de construção que são combinados, a partir de técnicas apropriadas, para produzir um sistema.

5.1 INTRODUÇÃO

A Orientação a Objetos apresentada no capítulo 2 é fruto do trabalho de muitas pessoas ao longo de vários anos. Os conceitos desenvolvidos representam uma mudança radical de paradigma no projeto e desenvolvimento de softwares.

No início, o desenvolvimento de software baseava-se simplesmente na programação. Conforme os sistemas foram tornando-se cada vez maiores e mais complexos, as pessoas perceberam que simplesmente escrever o código para uma aplicação de maneira desordenada não era bom o suficiente. Mesmo se a aplicação viesse a funcionar, qualquer mudança necessária poderia resultar em um verdadeiro pesadelo para os programadores.

A partir da programação orientada a objetos, surge a idéia de análise e projeto orientados a objeto, que permitem planejar as etapas de desenvolvimento de um software de maneira mais organizada antes mesmo que uma única linha de código seja programada.

As ferramentas CASE (*Computer-Aided Software-Engineering*), conhecidas atualmente como ferramentas de modelagem automática, representam um bom exemplo de ferramentas que auxiliam com as disciplinas de análise de requisitos, projeto e construção de software. Estas ferramentas tornam o desenvolvimento e a manutenção de softwares uma tarefa mais simples também.

Apesar de tudo, a orientação a objetos não garante milagres. Se as classes de objetos não forem projetadas cuidadosamente, de acordo com algumas diretrizes apresentadas ao longo deste capítulo, a orientação a objetos não trará os benefícios

oferecidos. Consequentemente, o software desenvolvido não será mais confiável e reutilizável, ou seja, os principais objetivos e vantagens da orientação a objetos não serão alcançados.

5.2 VANTAGENS

A utilidade específica da orientação a objetos depende muito da forma como ela é colocada em uso durante o processo de desenvolvimento do software. Apesar de não ser uma solução definitiva para o problema, a orientação a objetos é um meio poderoso, embora desafiador, para o desenvolvimento de software profissional. O processo exige um trabalho duro no modelo de orientação a objetos para que o mesmo possa ser integrado nos planos de longo prazo do projeto.

As atividades típicas de software [6] que sofrem influência direta da orientação a objetos podem ser percebidas da seguinte forma, onde a orientação a objetos:

- *Requisitos do Usuário*

  Facilita e antecipa a análise do processo e dos dados que fazem parte do sistema desenvolvido.

- *Projeto do Software*

  Oferece características de encapsulamento e herança que permitem esconder os detalhes de programação, criando estruturas mais sólidas quando bem projetado.

- *Construção do Software*

  Compreende os seguintes aspectos:

  - **Reutilização:** promove a reutilização dos códigos das classes no lugar das sub-rotinas.
  - **Confiabilidade:** garante a operação correta do código de forma contínua e consistente.
  - **Robustez:** possibilita uma recuperação sempre que ocorrer uma falha.
  - **Ampliação:** permite acrescentar novos recursos ao sistema de forma simples.

- **Escalabilidade:** suporta um aumento de carga no sistema, tanto de usuários quanto de transações.
- **Distribuição:** estende o modelo de objetos locais para o modelo de objetos distribuídos.
- **Armazenamento:** permite salvar objetos de classes arbitrárias, mantendo as características da OO, como herança e polimorfismo.

➢ *Manutenção do Software*

Melhora e realça cinco qualidades importantes apresentadas anteriormente: reutilização, confiabilidade, robustez, escalabilidade e ampliação, reduzindo os custos de manutenção.

➢ *Utilização do Software*

Oferece aplicações gráficas de fácil utilização, como menus que se adaptam a diferentes situações e estados.

➢ *Gerenciamento de Projetos de Software*

Proporciona uma mudança na organização do projeto como um todo, redistribuindo e criando novos papéis.

É importante ressaltar mais uma vez que o desenvolvimento de softwares orientados a objeto só será bem sucedido se gerido de forma inteligente, com uma introdução suave e bem acompanhada. O Processo Unificado descrito a seguir apresenta justamente as técnicas que devem ser empregadas para se alcançar o sucesso de um projeto.

5.3 PROCESSO UNIFICADO

Um processo de desenvolvimento de software compreende um conjunto de atividades necessárias para transformar os requisitos de um usuário em um sistema de software.

O *Processo Unificado* [25] é um processo de desenvolvimento de software que:

- Coordena a ordem das atividades de uma equipe de desenvolvimento.

- Indica as tarefas de cada integrante da equipe como um todo.
- Especifica quais partes devem ser desenvolvidas.
- Oferece critérios para a monitoração e medida das atividades e produtos resultantes do projeto.

Mais do que um processo isolado, o Processo Unificado é uma estrutura de desenvolvimento genérica (*framework*) que pode ser especializada para uma vasta classe de sistemas de software, para diferentes áreas de aplicação, diferentes tipos de organizações, diferentes níveis de competência, e diferentes tamanhos de projeto.

O Processo Unificado usa a *Linguagem de Modelagem Unificada* (UML) para o desenvolvimento das partes do sistema de software.

Contudo, os aspectos que realmente distinguem o Processo Unificado e o tornam único são capturadas em três palavras chaves: orientado por casos de uso (*use-case driven*), centrado na arquitetura (*architecture-centric*), e iterativo e incremental.

5.3.1 UM PROCESSO ORIENTADO POR CASOS DE USO

Um sistema de software tem como objetivo servir seus usuários. Portanto, para construir um sistema de sucesso, é necessário conhecer o que os usuários do sistema querem e precisam.

A interação de cada usuário com o sistema é descrita por um *Caso de Uso* (*Use-Case*). Cada caso de uso representa um pedaço de funcionalidade do sistema que retorna um resultado de valor para o usuário. O caso de uso captura os requisitos funcionais do sistema. O conjunto de todos os casos de uso resulta no *Modelo de Caso de Uso* que descreve a funcionalidade completa do sistema.

No entanto, mais do que especificar os requisitos de um sistema, os casos de uso conduzem também todo o processo de desenvolvimento do sistema, desde o projeto e implementação até os testes. Baseado no modelo de caso de uso, os projetistas criam uma série de modelos de projeto e implementação que realizam os casos de uso. Os testes da implementação, por sua vez, são feitos para assegurar a correta implementação dos casos de uso pelos componentes do modelo de implementação. Dessa forma, os casos de uso não iniciam somente o processo de desenvolvimento, mas o mantém coeso também, pois todo o processo prossegue através de uma série de fluxos de trabalho (*workflows*) que derivam deles.

5.3.1.1 Descrição Geral do Desenvolvimento Orientado por Casos de Uso

A captura de requisitos tem dois objetivos: encontrar os verdadeiros requisitos, ou seja, aqueles que quando implementados agregam valor aos usuários, e representa-los de uma forma adequada para os usuários, clientes, e projetistas.

Um sistema tem normalmente muitos tipos de usuários, sendo cada um representado como um ator (*actor*). Os atores interagem com o sistema e os casos de uso (*use-case*) representam esta interação. Um caso de uso é uma seqüência de ações que o sistema desempenha para oferecer alguns resultados de valor para um ator. O conjunto de todos os atores e casos de uso de um sistema compreende o modelo do caso de uso.

O modelo de casos de uso obtido é usado pelos projetistas para criar um modelo de análise (*analysis model*). Neste novo modelo, algumas classes de análise já podem ser identificadas, assim como algumas de suas responsabilidades, formando um conjunto de classes que juntas realizam os casos de uso utilizados pelos usuários.

A partir do modelo de análise, os projetistas projetam as classes e as realizações dos casos de uso para tirarem proveito dos produtos e tecnologias que serão utilizados para implementar o sistema, como kits de interface gráfica, sistemas de gerenciamento de banco de dados, e ORBs, criando o modelo de projeto (*design model*). As classes de projeto são agrupadas em subsistemas, definindo-se as interfaces entre os mesmos. O modelo de implantação (*deployment model*) é preparado paralelamente pelos projetistas. Neste modelo, os projetistas definem a organização física do sistema em nós computacionais, ou seja, máquinas, e verificam que os casos de uso podem ser implementados como componentes que executam nestes nós.

Em seguida, já no modelo de implementação, os projetistas implementam as classes de projeto como um conjunto de componentes do tipo arquivo (código fonte), cujos executáveis, tais como DLLs, e componentes Active X podem ser produzidos, ou seja, compilados e ligados. Os casos de uso, neste caso, ajudam os projetistas a determinarem a ordem em que eles devem implementar e integrar os componentes.

Finalmente, na etapa de testes, verifica-se se o sistema implementa realmente a funcionalidade descrita nos casos de uso e satisfaz os requisitos do sistema.

Contudo, o que foi descrito até aqui representa somente uma iteração. O projeto de desenvolvimento completo será composto por uma série de iterações, conforme discutido a seguir.

5.3.2 UM PROCESSO ITERATIVO E INCREMENTAL

Para facilitar o desenvolvimento de grandes projetos, é prático dividi-lo em pedaços menores ou mini-projetos. Cada mini-projeto representa uma iteração que resulta em um incremento ou crescimento do produto.

Em cada iteração, os projetistas identificam e especificam os casos de uso mais relevantes, criam um projeto usando a arquitetura escolhida como guia, implementam o projeto em componentes, e verificam que os componentes satisfazem os casos de uso selecionados. O processo prossegue com a próxima iteração assim que a iteração atual atingir seus objetivos. A arquitetura fornece a estrutura usada como base para guiar o trabalho nas iterações, enquanto os casos de uso definem os objetivos e conduzem o trabalho de cada iteração.

Um processo iterativo controlado dessa forma possibilita a redução de custos e riscos de uma forma geral, pois os problemas identificados antecipadamente em cada iteração podem ser solucionados rapidamente com uma revisão da própria iteração, e não de todo o produto. Consequentemente, o produto final tem uma qualidade muito melhor para um tempo de liberação ainda menor, o que aumenta as chances de sucesso do projeto.

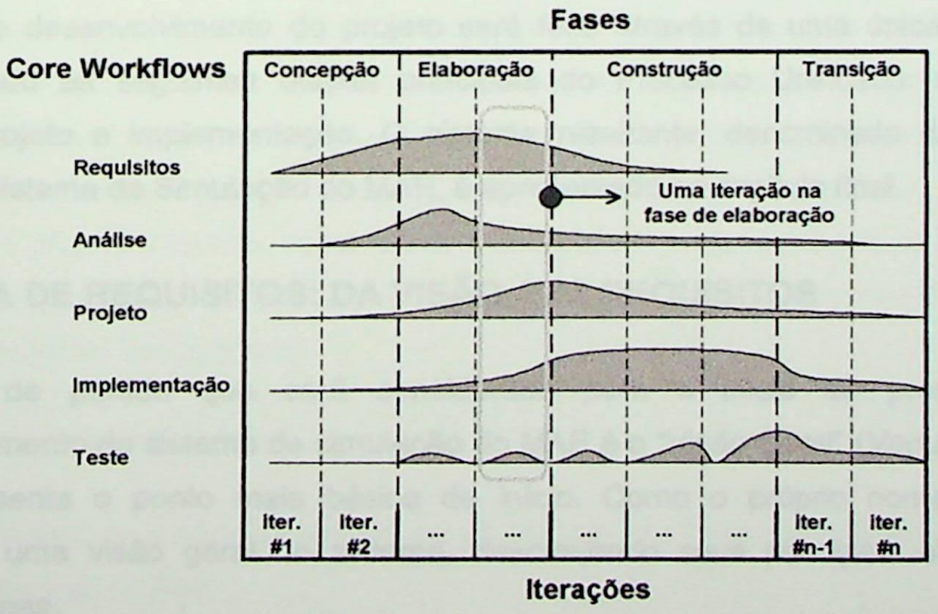

Figura 5.1 – Um ciclo completo do Processo Unificado

O Processo Unificado se repete, assim, ao longo de uma série de ciclos que compõem a vida de um sistema. Cada ciclo consiste de quatro fases: concepção (*inception*), elaboração (*elaboration*), construção (*construction*) e transição (*transition*). Cada fase, por sua vez, é subdividida em iterações, conforme discutido anteriormente. Além disso, cada iteração pode passar pelas cinco etapas principais (*core workflows*) apresentadas anteriormente, conforme ilustrado na figura 5.1: requisitos, análise, projeto, implementação e teste.

5.3.3 CONSIDERAÇÕES PARA O PROJETO DE SIMULAÇÃO DO MAE

O projeto do modelo de simulação proposto no capítulo anterior será feito de acordo com as diretrizes do Processo Unificado.

O modelo de simulação servirá de base para duas etapas fundamentais do Processo Unificado: coleta de requisitos e descrição da arquitetura.

Para simplificar o processo de desenvolvimento, serão feitas duas considerações importantes:

- A etapa de testes não será realizada, visto que o sistema resultante não impõe grandes dificuldades para sua operação.
- O planejamento da divisão do projeto, conforme descrito anteriormente, será ignorado, pois o sistema não é grande e complexo o suficiente para justificar um planejamento rigoroso.

Portanto, o desenvolvimento do projeto será feito através de uma única iteração, considerando as seguintes etapas principais do Processo Unificado: requisitos, análise, projeto e implementação. O sistema resultante, denominado daqui para frente de Sistema de Simulação do MAE, é apresentado no capitulo final.

5.3.4 CAPTURA DE REQUISITOS: DA VISÃO AOS REQUISITOS

O ponto de partida que será considerado para o início do processo de desenvolvimento do sistema de simulação do MAE é o "Visão Geral" (*Vague Notion*), que representa o ponto mais básico de início. Como o próprio nome diz, ele apresenta uma visão geral do sistema, descrevendo seus principais objetivos e características.

Visão Geral

A partir da reestruturação do setor de energia elétrica brasileiro, um novo modelo de mercado de energia passa a existir, com a criação de novos agentes e regras: o Modelo de Mercado Livre.

Para adaptar-se a essa nova estrutura, busca-se desenvolver um sistema de simulação para a integração desses agentes, de acordo com as novas regras do mercado de energia, em uma Intranet. O sistema será responsável, basicamente, pelo controle das ofertas de compra e venda feitas pelos agentes consumidores e produtores, respectivamente, retornando o resultado da liquidação das ofertas para os respectivos agentes.

O sistema deverá atender a característica distribuída, heterogênea e dinâmica do novo modelo, considerando seus principais agentes: o ASMAE, o OSMAE, os Produtores, os Consumidores e o ONS.

Independente do ponto de partida escolhido, o processo de desenvolvimento segue através de atividades importantes que resultarão no primeiro modelo do sistema do MAE: o modelo de caso de uso.

5.3.4.1 Listagem dos Requisitos Possíveis

Todas as boas idéias que surgirem durante o período de desenvolvimento do sistema e que forem consideradas como requisitos possíveis, devem ser mantidas em uma lista, com um nome e uma breve definição com informação suficiente para descrevê-la. Algumas informações extras podem ser adicionadas para caracterizar melhor o possível requisito listado, como Prioridade e Risco de Implementação. Estas informações, usadas em conjunto com outros aspectos, permitem estimar o tamanho do projeto e decidir como dividir o projeto em uma sequência de iterações. Além disso, é possível definir também em qual iteração será implementada cada idéia que se tornar um requisito.

Lista dos Requisitos Possíveis

- *Distribuído: o sistema deverá ser distribuído em 3 máquinas ligadas em rede Ethernet TCP/IP.*

- *Heterogêneo: o sistema deverá ser implementado em diferentes plataformas, usando diferentes linguagens de programação e SO, para mostrar a complexidade da integração de máquinas de diferentes locais e organizações.*
- *Segurança: o sistema deverá oferecer troca segura de informações entre usuários, como ACL´s.*
- *Sistema Elétrico: o sistema elétrico usado será um sistema térmico pré-configurado, com as barras e as linhas definidas.*
- *Cargas: cada consumidor deverá enviar os dados de carga previstos a cada hora. Os dados devem conter pelo menos a potência consumida.*
- *Geração: cada produtor deverá enviar os dados de geração horária, contemplando pelo menos a potência e o custo de geração.*
- *Dinâmico: o sistema deverá usar o ONS para otimizar o sistema elétrico usando Programação Linear, assim que todos os agentes produtores e consumidores tiverem enviado os respectivos dados dos geradores e cargas.*
- *Tarifas: os valores das tarifas de geração e consumo serão definidos a partir dos custos marginais resultantes da otimização.*
- *Liquidação: o sistema realizará o fechamento dos lances de oferta/demanda uma vez que o sistema elétrico esteja otimizado, enviando os resultados dos despachos e fatura para os respectivos agentes produtores e consumidores.*
- *Registro: todos os agentes deverão se registrar com o ASMAE para poderem participar do MAE. Será um ponto de acesso comum e conhecido usado pelos agentes para a localização dos demais agentes, além do uso de serviços gerais oferecidos pelo sistema.*
- *Difusão: toda vez que um novo agente se registrar no MAE, os demais agentes já registrados devem ser imediatamente informados, para tomarem conhecimento do fato.*

5.3.4.2 Contexto do Sistema

O contexto de um sistema pode ser expresso pelo menos de duas maneiras diferentes: *modelagem do domínio* e *modelagem do negócio*. O modelo do domínio descreve os conceitos importantes do contexto como objetos do domínio, ligando estes objetos um ao outro. Este modelo é usado durante o desenvolvimento dos modelos de caso de uso e de análise.

O modelo do negócio especifica quais processos do negócio devem ser suportados pelo sistema, estabelecendo ainda as competências exigidas em cada processo: os trabalhadores, suas responsabilidades, e as operações que eles irão desempenhar. Portanto, o modelo do negócio é um modelo mais completo, que inclui mais do que somente objetos do domínio.

O modelo do domínio para o sistema do MAE encontra-se na figura 5.2 a seguir.

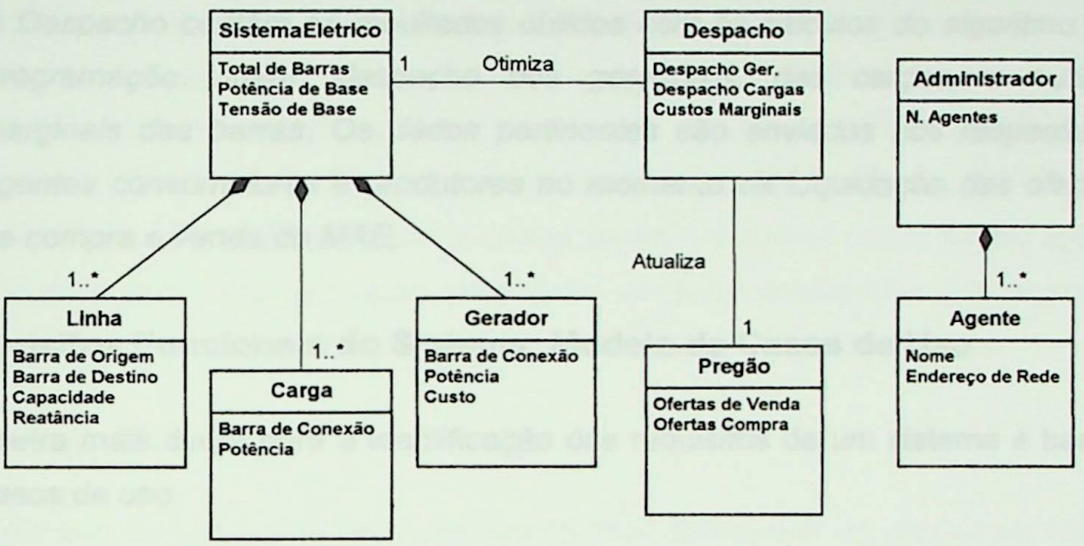

Figura 5.2 – Modelo do Domínio

As classes do Domínio

O sistema do MAE usará uma Intranet para a troca de dados entre os agentes, que devem se registrar previamente com o ASMAE para indicarem sua participação no MAE. O ASMAE funciona, assim, como um ponto comum usado pelos agentes para a localização dos demais agentes, mantendo uma lista dos agentes registrados no sistema que armazena pelo menos o nome e o endereço de rede do agente.

Um Gerador contém atributos como Custo de Geração, Potência e Barra de conexão com o sistema elétrico. São os dados enviados por cada produtor, a cada hora, ao ONS, toda vez que o produtor lançar uma nova oferta de venda no sistema do MAE.

Uma Carga contém atributos como Potência e Barra de conexão. São os dados enviados por cada consumidor, a cada hora, ao ONS, toda vez que o consumidor lançar uma nova oferta de compra no sistema do MAE.

Uma linha contém atributos como barra de origem, barra de destino, Capacidade e Reatância. Ela é criada no ONS.

O SistemaEletrico é um sistema térmico pré-configurado pelo ONS que define o número de barras e as linhas de transmissão do sistema elétrico. Os geradores e cargas são acrescentados ao sistema toda vez que novos produtores e consumidores integrarem o sistema do MAE, respectivamente.

O Despacho contém os resultados obtidos com os cálculos do algoritmo de Programação Linear: despacho dos geradores, das cargas, e custos marginais das barras. Os dados pertinentes são enviados aos respectivos agentes consumidores e produtores no momento da Liquidação das ofertas de compra e venda do MAE.

5.3.4.3 Requisitos Funcionais do Sistema: Modelo de Casos de Uso

A maneira mais direta para a identificação dos requisitos de um sistema é baseada nos casos de uso.

De uma maneira geral, os casos de uso oferecem uma forma sistemática e intuitiva para capturar os requisitos funcionais de um sistema com um foco específico no valor agregado a cada usuário em particular ou, a cada sistema externo que interage com o sistema desenvolvido.

Cada usuário quer que o sistema faça algo para ele, ou seja, realize determinados casos de uso. Para o usuário, um caso de uso é uma maneira de utilizar o sistema. Conseqüentemente, se todos os casos de uso que os usuários necessitam forem descritos, será possível então saber o que sistema deve fazer.

Portanto, cada caso de uso representa uma forma de usar o sistema. Cada usuário, por sua vez, necessita de alguns casos de uso diferentes, cada um representando as diferentes formas que ele usa o sistema.

Contudo, capturar os casos de uso que são realmente exigidos do sistema, tais como aqueles que suportam o negócio e que permitem aos usuários trabalharem de uma maneira confortável, exige um profundo conhecimento das necessidades do usuário e do cliente.

Para isto, é preciso entender o contexto do sistema, entrevistar usuários, discutir propostas, e assim por diante, conforme os passos apresentados anteriormente. Mais especificamente, um modelo de caso de uso é formado pelos seguintes artefatos:

- *Ator*

  Os atores (*Actor*) não representam necessariamente usuários humanos. Os atores podem ser outros sistemas ou hardware externo que interage com o sistema desenvolvido. Cada ator desempenha um determinado conjunto de papéis quando ele interage com o sistema. Os atores se comunicam com o sistema através de mensagens enviadas e recebidas conforme ele realiza os casos de uso. Definindo o que os atores fazem e o que os casos de uso fazem, é possível fazer uma separação clara entre as responsabilidades dos atores e do sistema. Esta separação ajuda assim a delimitar o escopo do sistema.

- *Caso de Uso*

  Um caso de uso (*Use-Case)* representa cada forma que um ator usa o sistema. Eles são pedaços de funcionalidade que o sistema oferece para agregar um resultado de valor aos seus usuários. Mais especificamente, um caso de uso especifica uma seqüência de ações, incluindo alternativas para a seqüência principal, que o sistema pode realizar, interagindo com os atores do sistema. Cada caso de uso pode ser descrito ainda como um texto à parte que detalha o fluxo de eventos do caso de uso. Requisitos não funcionais relativos ao caso de uso podem ser descritos também em um texto à parte que destaca algumas características específicas, tais como desempenho (*performance*), disponibilidade (*availability*), precisão (*accuracy*), e segurança, por exemplo.

- *Descrição da Arquitetura*

  A descrição da arquitetura contém uma vista arquitetônica do modelo de caso de uso, destacando os casos de uso mais significativos do ponto de vista da arquitetura, ou seja, os casos de uso que descrevem alguma funcionalidade importante e crítica ou que envolvem algum requisito importante que deve ser desenvolvido antecipadamente no ciclo de vida do software.

  A arquitetura será explorada por inteiro na seção final deste capítulo.

- *Glossário*

  O glossário pode ser usado para definir alguns termos importantes e comuns específicos do negócio, servindo de referência para projetistas que devem chegar a um consenso quanto à definição de conceitos e noções, de modo a reduzir o risco de interpretações equivocadas que possam comprometer o desenvolvimento do sistema.

- *Protótipo da Interface do Usuário*

  O protótipo de uma interface ajuda a entender e especificar melhor as interações que ocorrem entre atores humanos e o sistema, resultando em interfaces mais amigáveis e fáceis de usar.

A figura 5.3 a seguir apresenta o modelo completo do caso de uso do sistema de simulação do MAE.

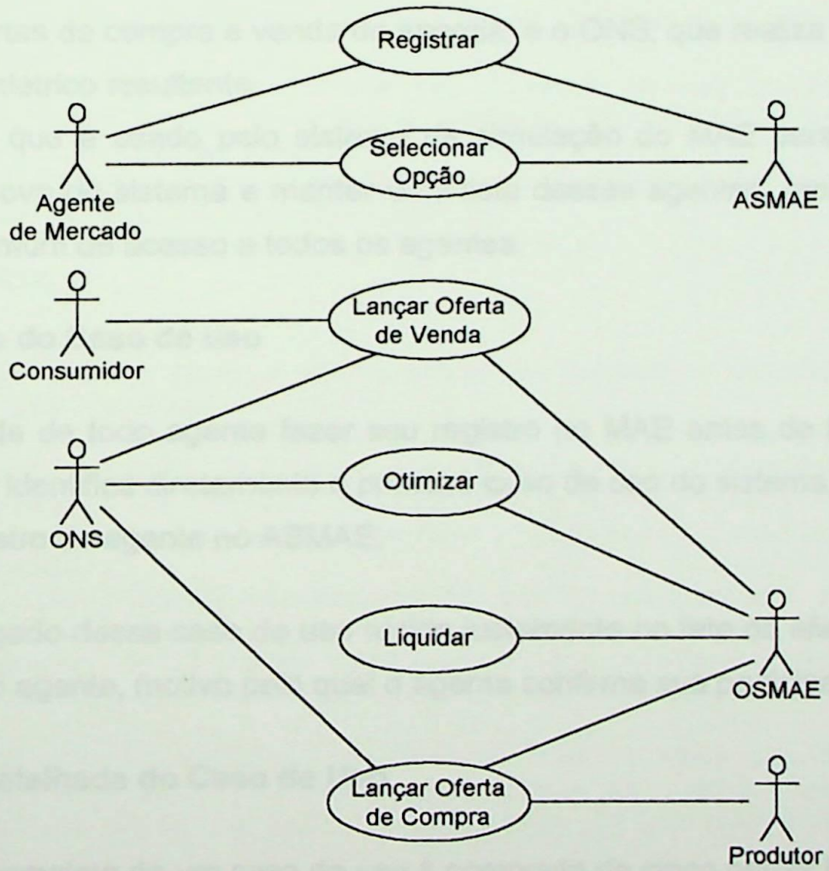

Figura 5.3 – Modelo de Caso de Uso do Sistema do MAE

O processo de desenvolvimento usado para a captura e documentação de cada requisito do sistema que resulta no modelo da figura 5.3 é descrito a seguir.

Os resultados das atividades destacadas referem-se aos resultados obtidos para o caso de uso *Registrar*, em particular, que será considerado daqui para frente como o caso base de estudo para apresentar o Processo Unificado.

O caso de uso *Registrar*, de uma forma sucinta, descreve o requisito do sistema que atesta que todos os agentes devem se registrar primeiramente no ASMAE para poderem participar do mercado atacadista de energia, conforme identificado na lista de requisitos possíveis feita previamente.

Identificação dos Atores

Os usuários que desejam participar do mercado atacadista de energia são todos os *Agentes* do mercado que precisam se registrar previamente no *ASMAE* para confirmar sua participação, o que resulta em dois atores:

- *Agente*, que representa um agente qualquer do mercado que deseja ou precisa fazer parte do sistema, como os agentes consumidores e produtores que fazem suas ofertas de compra e venda de energia, e o ONS, que realiza o despacho do sistema elétrico resultante.
- *ASMAE*, que é usado pelo sistema de simulação do MAE para registrar todo agente novo do sistema e manter uma lista desses agentes, representando um ponto comum de acesso a todos os agentes.

Identificação do Caso de uso

A necessidade de todo agente fazer seu registro no MAE antes de tomar qualquer outra ação já identifica diretamente o primeiro caso de uso do sistema, *Registrar*, que realiza o registro do agente no ASMAE.

O valor agregado desse caso de uso reside justamente no fato da efetivação ou não do registro do agente, motivo pelo qual o agente confirma sua participação no MAE.

Descrição Detalhada do Caso de Uso

A descrição completa de um caso de uso é composta de cinco partes básicas:

- *Descrição breve* (*Brief Description*), que explica sucintamente o caso de uso.
- *Pré-condição* (*Precondition*), que indica uma ou mais condições iniciais que devem ser satisfeitas antes de usar o caso de uso.

- *Caminho básico* (*Basic Path*), que descreve o fluxo normal do caso de uso.
- *Caminhos Alternativos* (*Alternative Paths*), que descreve os fluxos alternativos de ação do caso de uso considerando diferentes cenários possíveis.
- *Pós-condição* (*Postcondition*), que indica as condições necessárias para finalizar o caso de uso.

Para o caso de uso Registrar, a descrição resultante é a seguinte:

Descrição Breve: O caso de uso Registrar é usado pelos agentes de mercado interessados em participar do MAE para realizar o registro do agente no ASMAE que mantém uma lista dos agentes registrados.

Pré-condição: o agente de mercado deve possuir o endereço de rede do ASMAE.

Fluxo de Eventos
Caminho Básico

1. O agente de mercado invoca o caso de uso para participar do MAE, criando uma referência interna necessária para se registrar.
2. O agente, usando o endereço de rede obtido, conecta-se ao ASMAE.
3. O agente solicita seu registro, passando a referência criada.
4. O ASMAE registra o agente, criando um manipulador associado ao agente usado para poder acessar os serviços oferecidos pelo sistema do MAE.
5. O ASMAE informa aos demais participantes já registrados sobre a entrada do novo agente.
6. O ASMAE salva os registros atuais.
7. Fim.

Caminhos Alternativos

A.1. Caso o agente não consiga se conectar ao ASMAE, o caso de uso termina.
B.1. Se o nome usado pelo agente já existir na lista de participantes, o agente deverá optar por outro nome até conseguir se registrar, seguindo o curso básico de ação. Senão, o agente pode cancelar o registro, finalizando o caso de uso.
C.1. Se o ASMAE não conseguir entrar em contato com um ou mais dos agentes registrados, ele deve excluir as respectivas entradas da lista de participantes, informando os demais participantes sobre as exclusões realizadas.

Pós-condição: A instância do caso de uso termina quando o agente tiver se registrado sem problemas, tiver desistido (cancelado o registro) ou tiver falhado na tentativa de conexão com o ASMAE.

Requisitos Especiais

Os requisitos não funcionais identificados que se aplicam ao caso de uso *Registrar* são os seguintes:

- O agente poderá realizar até três tentativas consecutivas de conexão com o ASMAE, em intervalos de 5 min.
- O nome do agente deverá ter no máximo 10 caracteres.
- A lista de participantes deverá ser persistente para oferecer tolerância à falhas.
- O ASMAE deverá excluir um agente da lista de participantes somente após três tentativas consecutivas de conexão, em intervalos de 3 min, com o agente registrado.
- O ASMAE deverá verificar a conexão de todos os agentes registrados a cada 10 min, a fim de atualizar os registros.

Diagrama de Estados

O diagrama de estados do caso de uso *Registrar* é apresentado na figura 5.4 abaixo.

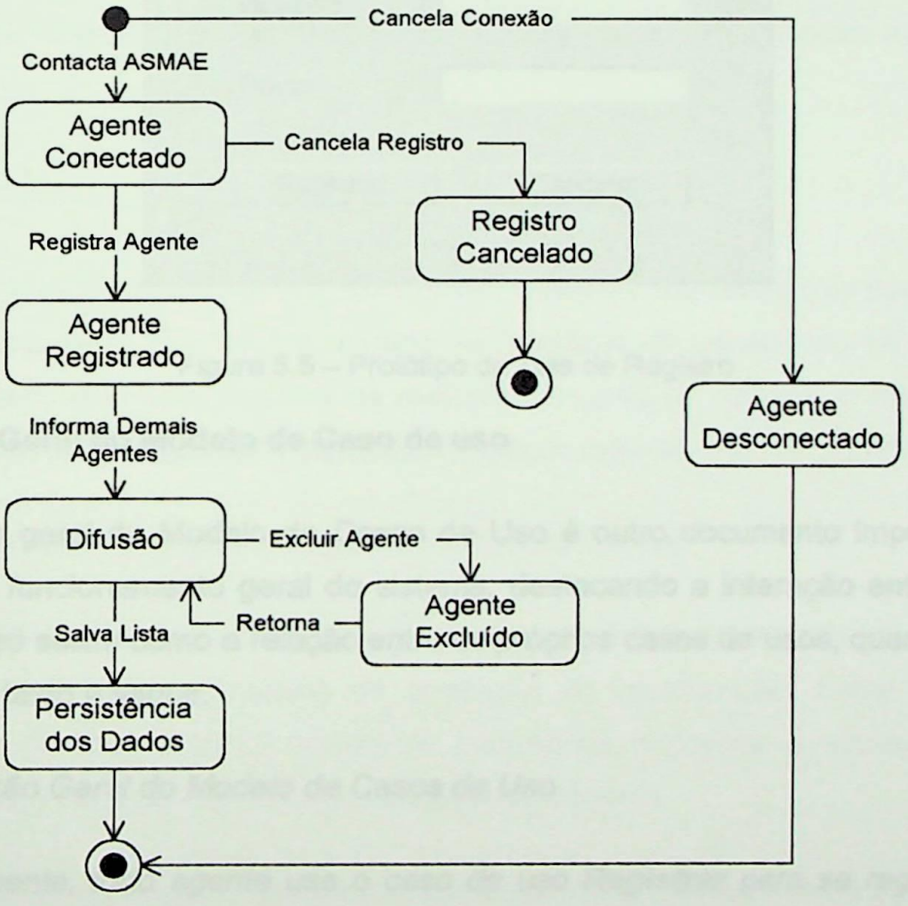

Figura 5.4 – Diagrama de Estados

Este diagrama apresenta todos os estados possíveis do caso de uso e as respectivas transições ou eventos que ocasionam a mudança de um determinado estado atual para um novo estado.

Protótipo da Interface do Usuário

Durante o desenvolvimento dos casos de uso, é interessante especificar também como será a aparência da interface do usuário quando os casos de uso forem realizados. Esta especificação pode ser feita através de esboços ou mesmo através de protótipos que possam ser experimentados pelos usuários.

A figura 5.5 apresenta um exemplo de um esboço feito em *Visio* de uma tela típica do Windows para o caso de uso *Registrar*, mostrando as informações pertinentes para o registro do agente bem como os botões de comando.

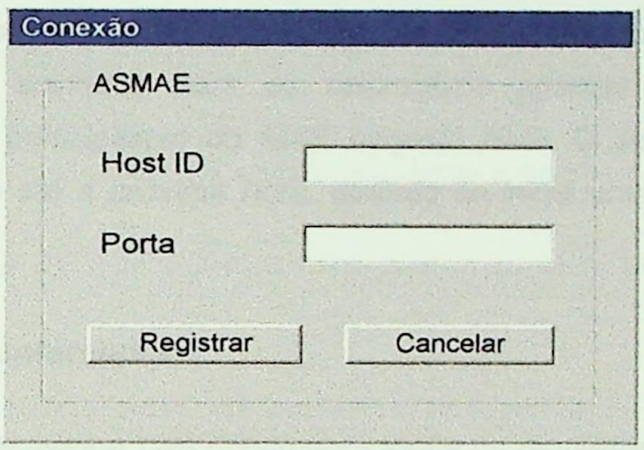

Figura 5.5 – Protótipo da Tela de Registro

Descrição Geral do Modelo de Caso de uso

A descrição geral do Modelo de Casos de Uso é outro documento importante que descreve o funcionamento geral do sistema, destacando a interação entre atores e casos de uso assim como a relação entre os próprios casos de usos, quando houver, conforme o texto a seguir.

> *Descrição Geral do Modelo de Casos de Uso*
>
> *Inicialmente, todo agente usa o caso de uso* ***Registrar*** *para se registrar no sistema de simulação do MAE e indicar sua participação no mercado livre.*
>
> *Cada produtor usa o caso de uso* ***Lançar Oferta de Venda*** *para prever a geração da hora atual e enviar a oferta de venda com os dados técnicos do gerador ao ONS.*

*O consumidor, por sua vez, usa o caso de uso **Lançar Oferta de Compra** para prever a carga da hora atual e enviar a oferta de compra com os dados técnicos da carga ao ONS.*

*Uma vez recebidas todas as ofertas de compra e venda, o ONS usa o caso de uso **Otimizar** para calcular o Despacho Econômico do sistema elétrico na hora atual considerada, usando Programação Linear para os cálculos. O resultado é enviado então ao MAE.*

*O MAE, na sequência, usa o caso de uso **Liquidar** para realizar o fechamento do Pregão, indicando o balanço final das negociações da hora atual no Pregão. As faturas para o pagamento e recebimento da energia negociada são enviadas para os respectivos agentes consumidores e produtores que participaram do MAE naquela hora. O sistema do MAE é então encerrado até a próxima hora, quando se inicia uma nova rodada de negócios.*

5.3.4.4 Especificação Suplementar

A Especificação Suplementar [26] documenta todos os requisitos que não aparecem no modelo de caso de uso, no modelo da interface do usuário ou no modelo do domínio. Estes requisitos incluem as *restrições*, *regras do negócio* e *requisitos não funcionais* que não estão associados com nenhum caso de uso em particular.

Identificação das Regras do Negócio (*Business Rules*)

As regras do negócio compreendem normalmente listas de acesso, avaliações do negócio ou políticas e princípios de operação da organização. Estas regras são identificadas durante o curso normal de captura de requisitos e análise, como na modelagem de caso de uso e na modelagem do domínio.

Como as regras do negócio são freqüentemente referenciadas nos modelos criados, elas podem ser documentadas com um índice para facilitar sua utilização, como nos exemplos a seguir:

```
BR110 Agentes produtores e consumidores devem estar conectados à
rede básica do sistema elétrico para poderem participar do MAE.
BR120.A otimização do sistema elétrico deve ser feita usando um
algoritmo de programação linear.
```

Uma documentação mais completa de cada regra do negócio pode incluir uma descrição mais detalhada, exemplos, fontes (normas, documentos, portarias) e regras relacionadas.

Identificação dos Requisitos Não Funcionais e das Restrições (*Constraints*)

Os requisitos não funcionais compreendem os aspectos técnicos que o sistema deve obedecer, tais como desempenho, confiabilidade e disponibilidade. A exemplo das regras do negócio, os requisitos não funcionais são documentados com um índice:

```
TR40 O sistema não pode ficar indisponível por mais de 10 minutos
durante um período de 24 horas.
```

Uma restrição, por sua vez, influencia o grau de liberdade disponível para prover soluções. Elas podem ser de ordem econômica, política, técnica ou ambiental, tais como recursos limitados, tecnologia a ser empregada e prazos de desenvolvimento.

Os exemplos a seguir ilustram algumas restrições documentadas que se aplicam ao sistema do MAE:

```
C89 O sistema irá operar em servidores Solaris.
C95 O sistema distribuído deverá ser implementado usando o padrão
CORBA.
C100 O algoritmo de programação linear deverá ser desenvolvido em
Matlab.
```

Observe que as iniciais de cada especificação suplementar identificam o tipo de restrição em Inglês, como BR, de *Bussiness Rule*.

5.3.4.5 Identificação de Casos de Mudança

Os casos de mudança são usados para descrever os requisitos que poderão ser acrescentados ou modificados no sistema, ou seja, eles documentam os requisitos que o sistema poderá precisar atender no futuro.

Documentação dos Casos de Mudança

A documentação dos casos de mudança é feita de maneira simples: uma descrição das prováveis mudanças dos requisitos existentes, uma indicação da probabilidade da ocorrência da mudança e uma indicação do possível impacto dessa mudança, como a seguir:

```
Caso de Mudança: O sistema deverá incluir agentes transportadores.
Probabilidade: Dentro de 2 anos.
Impacto:O número de agentes conectados irá aumentar sensivelmente.
```

Vantagens

A identificação dos casos de mudança traz inúmeras vantagens.

De uma maneira geral, os casos de mudança conduzem o desenvolvimento do software para uma aplicação mais fácil de manter, portável, modular, ampliável, de fácil utilização e mais robusta, alcançando os principais objetivos e vantagens da orientação a objetos.

Além disso, os casos de mudança orientam para a escolha mais apropriada da tecnologia que deve ser utilizada para atender as necessidades de longo prazo do projeto, minimizando assim o impacto de mudanças.

5.3.4.6 Análise, Projeto e Implementação: Realizar os Casos de Uso

Durante as etapas de análise e projeto [27], o modelo de caso de uso é transformado em um modelo de projeto através do modelo de análise, ou seja, em uma estrutura de classificadores e realizações de casos de uso (*classifiers and use-case realizations*).

Classificadores, neste caso, são coisas "tipo-classe", como subsistemas, interfaces, atores e casos de uso. O objetivo é realizar os casos de uso de modo que o sistema ofereça um desempenho adequado e possa evoluir no futuro.

5.3.5 ANÁLISE

Na etapa de análise, os requisitos descritos durante a etapa de captura de requisitos são analisados por meio de um refinamento e estruturação. O objetivo é obter um conhecimento mais preciso dos requisitos e uma descrição que seja fácil de manter e que ajude a estruturar o sistema como um todo, incluindo sua arquitetura.

Os diagramas, definições e descrições que se seguem representam o resultado das atividades desenvolvidas na etapa de análise para o sistema de simulação do MAE, mais especificamente para o caso base de estudo adotado: o caso de uso *Registrar*.

5.3.5.1 Análise dos Casos de Uso

O primeiro passo na análise de um caso de uso é identificar as classes de análise necessárias para realizar o caso de uso e esboçar seus nomes, responsabilidades, atributos e relacionamentos. A UML define três estereótipos de classe padrões usados em análise. Estes estereótipos ajudam os projetistas a distinguirem os negócios entre diferentes classes:

- *Classes de Fronteira*

  Uma classe de fronteira (*Boundary Classes*) é usada para modelar a interação entre o sistema e seus atores. Elas representam abstrações de interfaces do usuário, como janelas e caixas de diálogo, interfaces de comunicação, sensores, ou seja, elas modelam as partes do sistema que dependem de seus atores, elucidando e coletando os requisitos nas fronteiras do sistema. A classe Registro UI é um exemplo de uma classe de fronteira usada para suportar a interação entre o ator Agente, o ator ASMAE e o caso de uso *Registrar*. Ela permite que um usuário se conecte ao ASMAE para registrar o agente ou cancelar o registro.

- *Classes de Entidade*

  Uma classe de entidade (*Entity Classes*) é usada para modelar informação que é de longa duração e normalmente persistente, como um indivíduo, um objeto da vida real ou um evento da vida real. Estas classes normalmente derivam diretamente de uma classe correspondente do domínio ou do negócio que fazem parte do modelo do domínio ou do negócio, discutidos anteriormente. Elas mostram geralmente uma estrutura de dados lógica e contribuem para o entendimento das informações usadas pelo sistema. A classe Agente é um exemplo de classe de entidade usada para representar cada agente de mercado. Ela está associada com a classe de fronteira Registro UI usada por um usuário para registrar o agente criado com o ASMAE.

- *Classes de Controle*

  As classes de controle (*Control Classes*) representam coordenação, seqüências, transações e controle de outros objetos, sendo usadas normalmente para encapsular o controle relativo a um caso de uso em particular. Elas podem ser usadas também para representar cálculos complexos, tal como uma lógica do

negócio, que não pode ser relacionada com nenhuma informação específica armazenada pelo sistema, ou seja, com nenhuma classe entidade específica. A classe de controle Administrador de Agentes representa uma classe de controle responsável pela coordenação entre a classe Registro UI e a classe Agente, e pelo controle de todos os agentes registrados no MAE.

O passo seguinte diz respeito à realização do caso de uso propriamente dita.

Uma realização do caso de uso – análise (*use-case realization – analysis*) é uma colaboração dentro do modelo de análise que descreve como um determinado caso de uso é realizado e desempenhado em termos de classes de análise e interações dos seus objetos de análise. Conseqüentemente, uma realização de um caso de uso tem correspondência direta com um determinado caso de uso do modelo de casos de uso, o que demonstra que o processo de desenvolvimento é conduzido pelo caso de uso.

A realização de um caso de uso compreende as seguintes partes:

- *Diagramas de Classe*

  Os diagramas de classe (*Class Diagrams*) são usados para mostrarem as classes de análise que participam da realização de um caso de uso assim como os relacionamentos entre as mesmas.

- *Diagramas de Interação*

  Os diagramas de interação (*Interaction Diagrams*) são usados para descreverem a realização de um determinado fluxo ou cenário de um caso de uso em termos de interações que ocorrem entre objetos da análise. Como o foco principal é encontrar requisitos e responsabilidades dos objetos e não seqüências cronológicas e detalhadas, indica-se utilizar os diagramas de colaboração, que descrevem a interação entre os objetos através de ligações (*link*) identificadas por uma mensagem.

- *Fluxo de Eventos – Análise*

  O fluxo de eventos (*Flow of Events – Analysis*) da mais é que um texto descritivo, escrito em termos de objetos, particularmente objetos de controle, que

interagem para realizarem um determinado caso de uso. O texto ajuda a explicar os diagramas, facilitando sua compreensão.

- *Requisitos Especiais – Realização de um Caso de uso*

  Os requisitos especiais (*Special Requirements*) são descrições em forma de texto que reúnem todos os requisitos não funcionais referentes a uma determinada realização de um caso de uso. Alguns desses requisitos já foram previamente identificados durante a etapa de coleta de requisitos, e são simplesmente transferidos para a realização do caso de uso na análise. O restante corresponde aos novos requisitos que são identificados conforme a etapa de análise progride.

Para o caso de uso *Registrar*, em especial, o diagrama de classe obtido é apresentado na figura 5.6 a seguir.

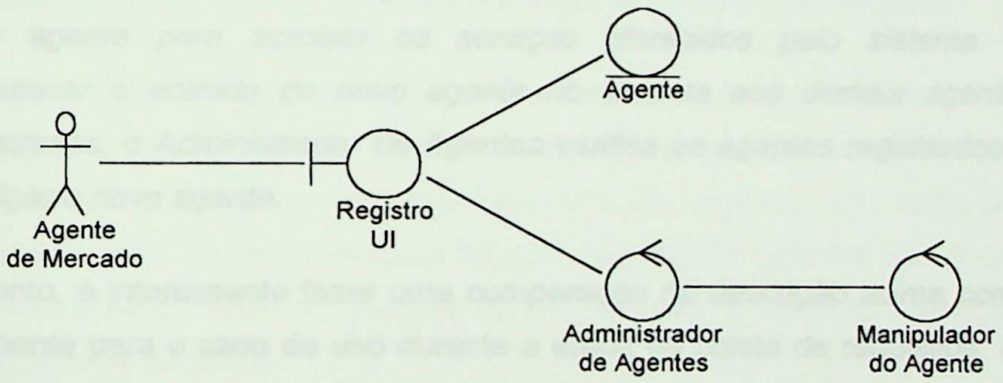

Figura 5.6 – Diagrama de Classe para a realização do Caso de Uso Registrar

A figura 5.7 apresenta o diagrama de colaboração correspondente.

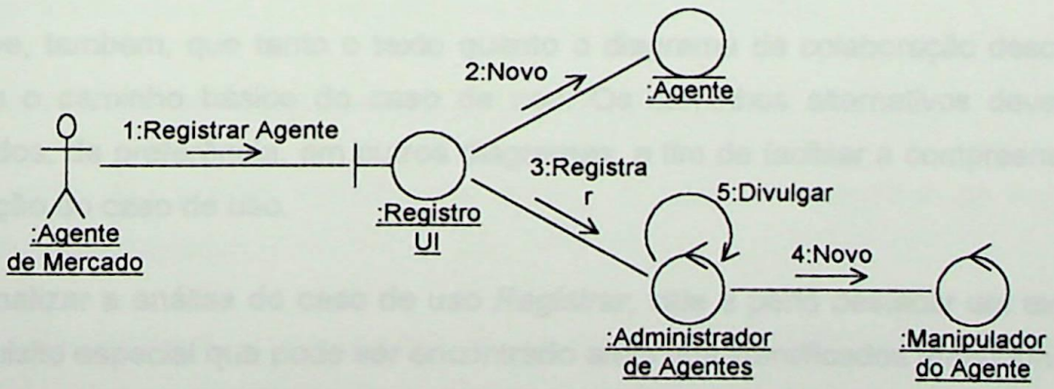

Figura 5.7 – Diagrama de Colaboração para a realização do Caso de Uso Registrar

O fluxo de eventos da análise que explica e complementa o diagrama de colaboração da figura 5.7 está descrito no texto a seguir. Os números no texto identificam as mensagens do diagrama de colaboração correspondentes às ações descritas.

Fluxo de Eventos do Diagrama de Colaboração do Caso de Uso Registrar

O agente de mercado, usando a interface do usuário Registro UI, solicita o registro (1) do respectivo agente no sistema para confirmar sua participação no MAE. Antes de conectar-se com o Administrador de Agentes, o Registro UI cria (2) um objeto Agente usado para registrar o agente. Na seqüência, o Registro UI requisita (3) ao Administrador de Agentes o registro do agente criado. Neste ponto, o Administrador pode usar alguma regra do negócio para verificar se o agente pode ou não participar do MAE (ver exemplo da BR110 na seção Requisitos Suplementares). O Administrador de Agentes cria (4) então um Manipulador do Agente que representa um intermediador usado pelo agente para acessar os serviços oferecidos pelo sistema. Para comunicar a entrada do novo agente no sistema aos demais agentes já registrados, o Administrador de Agentes verifica os agentes registrados para divulgar o novo agente.

Neste ponto, é interessante fazer uma comparação da descrição acima com a feita anteriormente para o caso de uso durante a etapa de coleta de requisitos. Naquela situação, a descrição do caso de uso se baseou no ponto de vista de um observador externo para detalhar os passos ou ações do caso de uso, enquanto a descrição acima foca no modo como o caso de uso é realizado pelo sistema em termos de objetos que colaboram.

Observe, também, que tanto o texto quanto o diagrama de colaboração descrevem apenas o caminho básico do caso de uso. Os caminhos alternativos devem ser retratados, de preferência, em outros diagramas, a fim de facilitar a compreensão da realização do caso de uso.

Para finalizar a análise do caso de uso *Registrar*, vale a pena destacar um exemplo de requisito especial que pode ser encontrado além dos identificados anteriormente:

O tempo total para efetuar o registro do agente não deve ser superior a 1.0 s.

5.3.5.2 Análise das Classes

A análise de uma classe é importante, pois permite:

- Identificar e manter as responsabilidades da classe, baseado no seu papel desempenhado nas realizações dos casos de uso.
- Identificar e manter os atributos e relacionamentos da classe.
- Capturar requisitos especiais na realização da classe de análise.

O resultado da análise da classe de controle *Administrador de Agentes* identificada anteriormente é apresentado na sequência:

- *Papéis da Classe Administrador de Agentes*

  Objetos da classe Agente são criados uma única vez por cada agente de mercado durante o caso de uso Registrar. Cada agente de mercado realiza este caso de uso para solicitar ao ASMAE o registro do respectivo agente.

  O registro é efetuado pelo Administrador de Agentes, que cria um Manipulador de Agente para ser usado posteriormente pelo agente registrado. Logo em seguida, o Administrador de Agentes entra em contato com cada agente já registrado para divulgar a entrada do novo agente. Os agentes não encontrados são excluídos do MAE. No decorrer do caso de uso, o objeto Agente fica sempre no agente de mercado.

  O acesso aos serviços oferecidos pelo sistema é feito por um agente de mercado durante o caso de uso Selecionar Opção. O Administrador de Agentes verifica o registro do agente, executa a opção selecionada e informa os demais agentes registrados com um resumo das ações realizadas.

- *Responsabilidades da Classe Administrador de Agentes*

  A partir dos papéis identificados acima, é possível identificar as seguintes responsabilidades:

  - Registrar novos agentes que se conectam com o ASMAE pela primeira vez
  - Verificar a autenticidade do agente conectado através do seu registro
  - Executar a opção selecionada pelo agente

- Divulgar ações tomadas por um determinado agente aos demais agentes registrados
- Excluir os agentes que não puderam ser contactados

➢ *Atributos da Classe Administrador de Agentes*

Os atributos estão listados abaixo:

- Total de Agentes Registrados
- Lista dos Agentes Registrados
- Opções

Observe que a quantidade de atributos identificados é pequena, em relação às responsabilidades, o que demonstra a característica de uma classe de controle.

➢ *Relacionamentos*

A figura 5.8 a seguir apresenta dois casos de relacionamentos identificados para as classes de análise do caso de uso *Registrar*: uma agregação (a) e uma generalização (b).

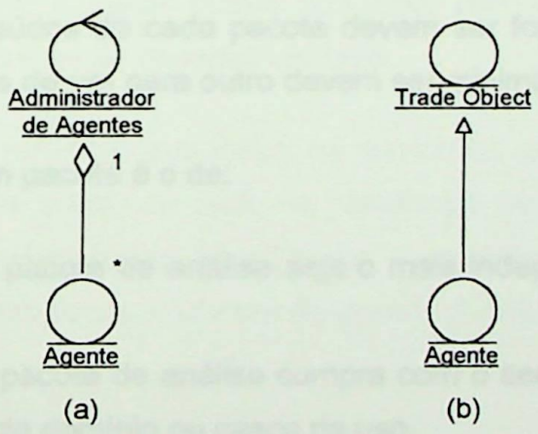

Figura 5.8 – (a) Administrador de Agentes e seus Constituintes
(b) Generalização do Agente

Observe que a generalização, em particular, é uma consequência direta da arquitetura do sistema. Como os agentes estão distribuídos pelo sistema, eles compartilham um comportamento comum: são objetos distribuídos.

Contudo, é importante ressaltar que a generalização na análise deve ser mantida em um nível alto e conceitual, pois o objetivo é simplesmente facilitar o

entendimento do modelo. Somente durante o projeto é que a generalização será adaptada para se encaixar melhor com o ambiente de implementação escolhido, ou seja, de acordo com os requisitos suplementares especificados.

- *Requisitos Especiais – Classe de Análise*

  Neste passo são capturados todos os requisitos de uma classe de análise que são identificados durante a análise, mas que devem ser trabalhados durante o projeto e implementação, ou seja, os requisitos não funcionais. Os requisitos especiais da realização do caso de uso listados anteriormente devem ser considerados neste passo, pois eles podem conter requisitos não funcionais adicionais para a classe de análise. Para a classe de análise *Administrador de Agentes*, pode-se destacar as características do requisito de persistência da classe identificado logo de início, como tamanho do objeto, volume esperado de objetos e freqüência de atualização.

5.3.5.3 Análise dos Pacotes

Os pacotes de análise fornecem uma forma de organizar os artefatos do modelo de análise em partes controláveis. Os pacotes devem ser coesivos e fracamente ligados, ou seja, os conteúdos de cada pacote devem ser fortemente relacionados enquanto as dependências de um para outro devem ser minimizadas.

O propósito de analisar um pacote é o de:

- Assegurar que o pacote de análise seja o mais independente possível dos demais pacotes.
- Assegurar que o pacote de análise cumpra com o seu propósito de realizar algumas classes do domínio ou casos de uso.
- Descrever as dependências para poder estimar os efeitos de mudanças futuras.

A figura 5.9 a seguir mostra dois pacotes que foram criados para a realização do caso de uso *Registrar*.

A separação proposta considera o fato da distribuição dos agentes, isolando a classe *Agente* da classe *Administrador de Agentes*, que é geral. A dependência entre os pacotes é indicada pela seta tracejada, enquanto a seta cheia indica a dependência

direta entre as classes. A direção das setas destaca o pacote e a classe dependente. Os pacotes, neste caso, dependem um do outro. A classe *Administrador de Agentes*, por sua vez, depende da classe *Agente*.

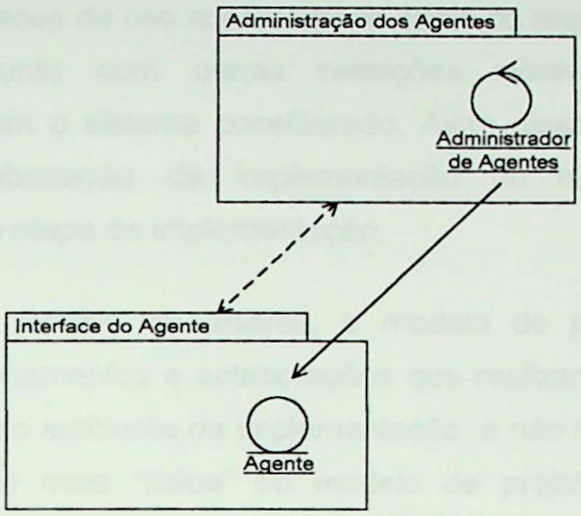

Figura 5.9 – Dependência dos Pacotes do Caso de Uso Registrar

Por se tratar de um modelo conceitual ainda, obtido da análise arquitetônica inicial do sistema do MAE, pode ser difícil enxergar essas dependências no momento. Contudo, o próprio refinamento desses resultados já na próxima etapa de projeto, será suficiente para elucidar qualquer dúvida que ainda possa existir, demonstrando o poder do uso de um processo de desenvolvimento.

Além disso, é importante ressaltar que todos os resultados apresentados até então representam os resultados finais, ou seja, os resultados obtidos após sucessivas atualizações decorrentes das novas abordagens e mudanças feitas nas etapas subseqüentes, característico de um processo de desenvolvimento.

5.3.6 PROJETO

O objetivo da etapa de projeto é modelar o sistema de tal maneira a encontrar uma forma, incluindo uma arquitetura, que atenda a todos os requisitos pertinentes ao sistema, incluindo os requisitos não funcionais e outras restrições.

A modelagem é feita a partir do modelo de análise desenvolvido anteriormente, que impõe uma estrutura do sistema que deve ser preservada na medida do possível durante a modelagem. O modelo de análise serve, portanto, de base para a criação do modelo de projeto, que é então adaptado ao ambiente de implementação

escolhido, tal como um ORB, uma GUI, ou um sistema de gerenciamento de banco de dados.

Mais especificamente, o modelo de projeto é um modelo de objeto que descreve a realização física dos casos de uso considerando como os requisitos funcionais e não funcionais, em conjunto com outras restrições relativas ao ambiente de implementação, atingem o sistema considerado. Além disso, o modelo de projeto serve como uma abstração da implementação do sistema sendo usado, conseqüentemente, na etapa de implementação.

De forma similar ao modelo de análise, o modelo de projeto também define classificadores, relacionamentos e colaborações que realizam os casos de uso, só que agora adaptados ao ambiente de implementação, e não mais conceituais, o que caracteriza a natureza mais "física" do modelo de projeto. Os resultados das atividades desenvolvidas na etapa de projeto para o caso de uso *Registrar* são apresentados na seqüência. A arquitetura, em particular, atividade inicial e principal, será explorada na seção final deste capítulo.

5.3.6.1 Projetar um Caso de uso

O propósito de projetar um caso de uso é para:

- Identificar as classes de projeto e/ou subsistemas cujas instâncias são necessárias para realizar o fluxo de eventos do caso de uso.
- Distribuir o comportamento do caso de uso para objetos de projeto e/ou subsistemas participantes que se interagem.
- Definir requisitos nas operações das classes de projeto e/ou subsistemas e suas interfaces.
- Capturar os requisitos de implementação para o caso de uso.

O diagrama de classes no projeto é útil para mostrar as classes de projeto participantes na realização de um caso de uso juntamente com seus relacionamentos. A interação dos objetos é descrita por um diagrama de seqüência, que contém as instâncias dos atores participantes, os objetos de projeto, e as transmissões de mensagens entre eles.

A figura 5.10 apresenta um diagrama de classes que descreve o caso de uso *Registrar*, conforme o diagrama de classes da análise.

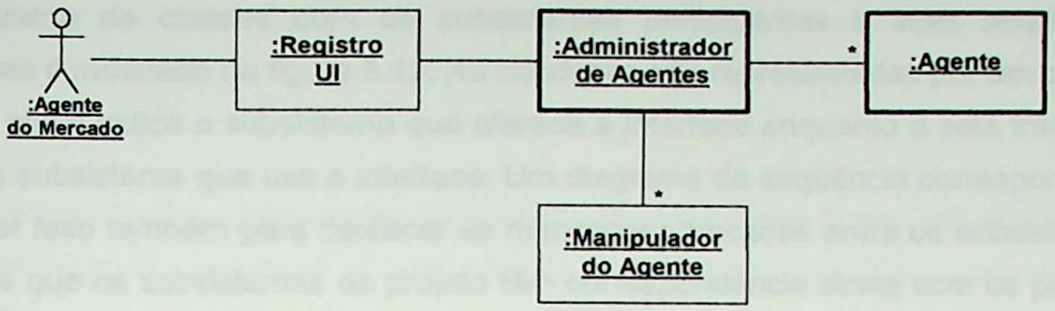

Figura 5.10 – Diagrama de Classes do Caso de Uso Registrar

O diagrama de sequência correspondente é apresentado na figura 5.11.

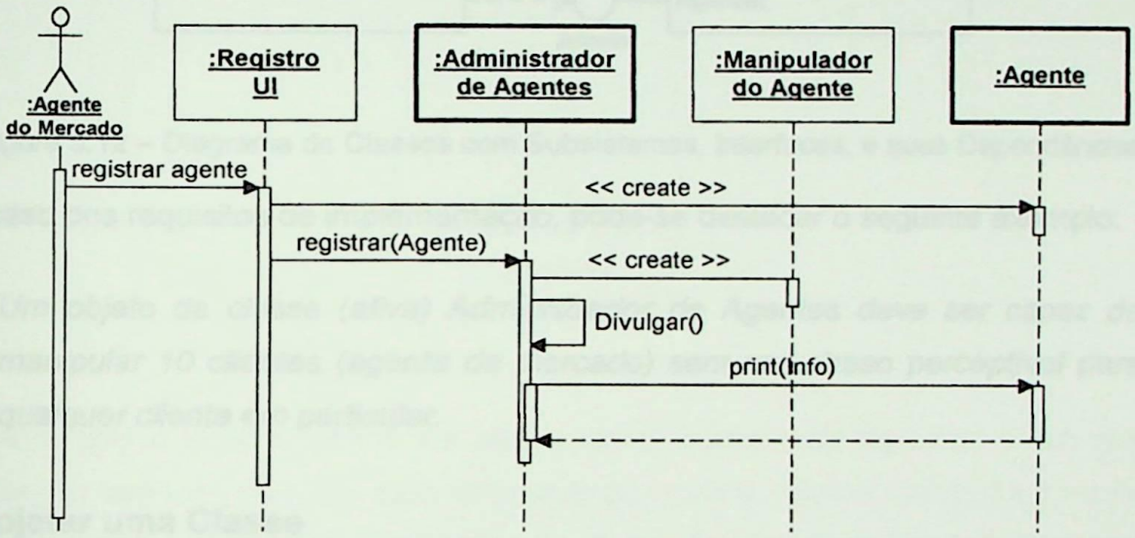

Figura 5.11 – Diagrama de Sequência do Caso de Uso Registrar

As classes destacadas com uma borda mais grossa são as classes ativas. Elas representam as classes principais que mantêm o sistema operando. O fluxo de eventos do diagrama de seqüência é descrito pelo seguinte texto:

Fluxo de Eventos

O agente de mercado usa o sistema, através do diálogo Registro UI e da aplicação do Administrador de Agentes, para registrar o Agente criado. O Administrador de Agentes verifica se não existe um registro com o mesmo nome antes de registrar o novo agente. Uma vez verificado, o Administrador de Agentes cria um objeto Manipulador do Agente e inclui o novo agente na lista de registros. A participação do novo agente no sistema é então divulgada a todos os outros agentes já registrados através da aplicação do Administrador de Agentes, que se conecta com cada aplicação do Agente.

O diagrama de classes com os subsistemas participantes e suas respectivas interfaces é mostrado na figura 5.12. As interfaces são representadas por um círculo. A linha cheia indica o subsistema que oferece a interface enquanto a seta tracejada indica o subsistema que usa a interface. Um diagrama de seqüência correspondente pode ser feito também para destacar as mensagens trocadas entre os subsistemas. Observe que os subsistemas de projeto têm correspondência direta com os pacotes da análise.

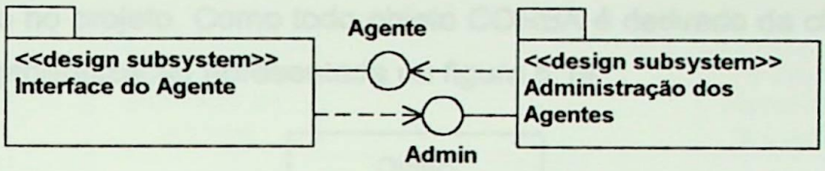

Figura 5.12 – Diagrama de Classes com Subsistemas, Interfaces, e suas Dependências

No caso dos requisitos de implementação, pode-se destacar o seguinte exemplo:

> *Um objeto da classe (ativa) Administrador de Agentes deve ser capaz de manipular 10 clientes (agente de mercado) sem um atraso perceptível para qualquer cliente em particular.*

5.3.6.2 Projetar uma Classe

O objetivo de projetar uma classe é o de criar uma classe de projeto que cumpra com seu papel nas realizações dos casos de uso e atenda os requisitos não funcionais que se aplicam a ela. O resultado parcial obtido para a classe *Administrador de Agentes*, em especial, é descrito na figura 5.13.

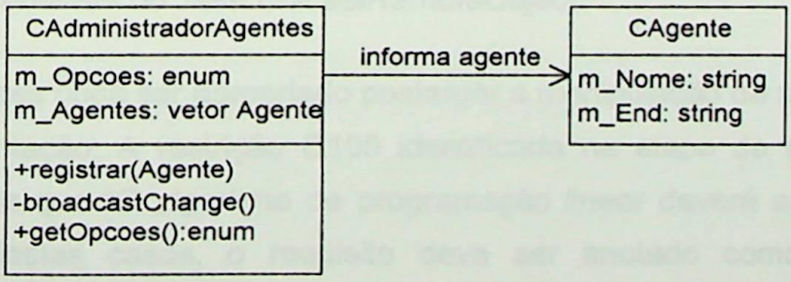

Figura 5.13 – Classe Administrador de Agentes

Os métodos públicos (sinal +) implementam as operações necessárias para suportar a interface Admin do subsistema Administração dos Agentes. O atributo *Agentes* sugerido durante a análise implica na associação entre as classes indicada pela seta.

Observe que os nomes das classes, operações e atributos devem ser especificados de acordo com a sintaxe da linguagem de programação escolhida, que neste caso é C++.

A generalização no projeto, conforme frisado na análise, deve considerar o ambiente de implementação escolhido. Neste caso, a restrição técnica C95 identificada na seção Requisitos Suplementares que diz que "O sistema distribuído deverá ser implementado usando o padrão CORBA", representa o requisito especial que deve ser manipulado no projeto. Como todo objeto CORBA é derivado da classe *Object*, o resultado é a generalização apresentada na figura 5.14.

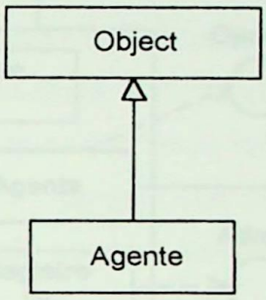

Figura 5.14 – O objeto distribuído Agente

Observe que o próprio objeto da classe *Administrador de Agentes* é um objeto distribuído também, pois ele será acessado por outros objetos distribuídos na rede, além de acessar os objetos distribuídos da classe *Agente*. Portanto, todos os objetos que tiveram que se comunicar com outros objetos distribuídos, deverão ser seguir a generalização da figura 5.14.

Se tivéssemos especificado o mecanismo RMI do Java, por exemplo, para implementar o sistema distribuído, a generalização da figura 5.14 seria feita com base na classe abstrata do Java *UnicastRemoteObject*.

Contudo, às vezes pode ser apropriado postergar a manipulação de alguns requisitos até a implementação. A restrição C100 identificada na etapa de requisitos é um exemplo. Ela diz que "O algoritmo de programação linear deverá ser desenvolvido em Matlab". Nestes casos, o requisito deve ser anotado como Req*uisito de Implementação* para a respectiva classe de projeto.

5.3.6.3 Projetar um Subsistema

O propósito de projetar um subsistema é para:

- Assegurar que o subsistema seja o mais independente possível de outros subsistemas e/ou suas interfaces.
- Assegurar que o subsistema forneça as interfaces apropriadas.
- Assegurar que o sistema cumpra com seu propósito que é oferecer uma correta realização das operações definidas nas suas interfaces.

A figura 5.15 a seguir apresenta o resultado obtido para os subsistemas da figura 5.11.

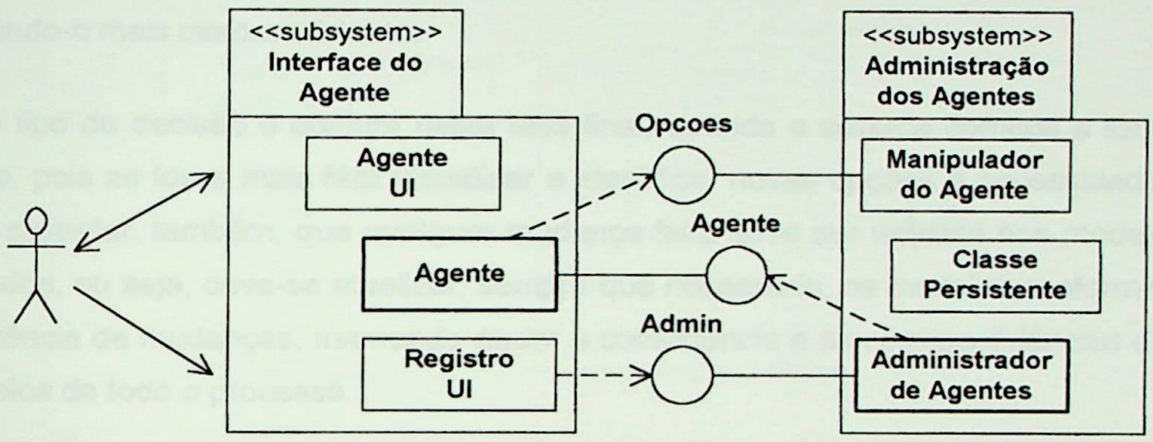

Figura 5.15 –Subsistemas do Caso de Uso Registrar

Observe que os subsistemas agora estão completos, ou seja, com suas respectivas classes, incluindo a identificação da classe que oferece a interface do subsistema. As dependências também estão mais nítidas agora, conforme explicado anteriormente durante a análise.

Neste caso, a linha cheia que sai da classe para a interface significa que a classe fornece a interface. A seta tracejada significa que a classe usa a interface.

A interface *Admin* define as operações necessárias para realizar o registro de um agente de mercado. A interface *Agente* define as operações que permitem informar os eventos relevantes que ocorrem no sistema. A interface *Opções*, por sua vez, define as operações exigidas pelo caso de uso *Selecionar Opção*, incluída somente para mostrar o modelo completo.

Além disso, observe também a nova classe, *Classe Persistente*, pertencente ao subsistema *Administração dos Agentes*. Ela atende justamente o requisito de persistência identificado para o caso de uso *Registrar* logo no início do processo de desenvolvimento.

Por isso é tão importante identificar e documentar cedo os requisitos do sistema, para evitar problemas futuros como a ausência de uma classe que não atenda ou comprometa a realização do caso de uso.

O subsistema *Administração dos Agentes*, em particular, possui uma nova interface oferecida pelo *Manipulador do Agente*, como resultado de um outro caso de uso, *Selecionar Opção*, usado por todo agente de mercado que deseja acessar serviços especiais oferecidos pelo sistema. De qualquer forma, o objetivo é separar as funcionalidades dos objetos para facilitar a implementação do sistema distribuído, tornando-o mais claro e eficiente.

Esse tipo de decisão é comum nesta fase final, quando o sistema começa a tomar corpo, pois se torna mais fácil visualizar e identificar novas opções e necessidades. Vale salientar, também, que qualquer mudança feita deve ser refletida nos modelos afetados, ou seja, deve-se atualizar, sempre que necessário, os modelos conforme a ocorrência de mudanças, mantendo assim a consistência e as correspondências dos modelos de todo o processo.

5.3.7 IMPLEMENTAÇÃO

Na etapa de implementação realiza-se o desenvolvimento de tudo o que necessário para produzir um sistema executável: os componentes executáveis; os componentes arquivos, contendo os códigos fonte, scripts, etc; os componentes tabelas, contendo os elementos relativos ao banco de dados escolhido; e assim por diante. Um componente, neste caso, é uma parte física e substituível de um sistema que obedece e fornece a realização de um conjunto de interfaces.

O resultado final é o modelo de implementação que descreve como os elementos do modelo de projeto são implementados em termos de componentes. O modelo de implementação descreve também a organização dos componentes de acordo com as características do ambiente de implementação e das linguagens de programação em uso, e como os componentes dependem um do outro.

A figura 5.16 descreve os componentes que implementam as classes de projeto do modelo de projeto apresentado anteriormente.

Neste caso, o componente arquivo *Admin_impl.c* contêm os códigos fonte e, portanto implementa, duas classes do subsistema *Administração dos Agentes*: Administrador

de Agentes e Classe Persistente. O outro componente arquivo implementa a classe restante, *Manipulador do Agente*. Os componentes arquivos são então compilados e ligados para gerar o componente executável *Admin.exe*, que representa o aplicativo final do ASMAE.

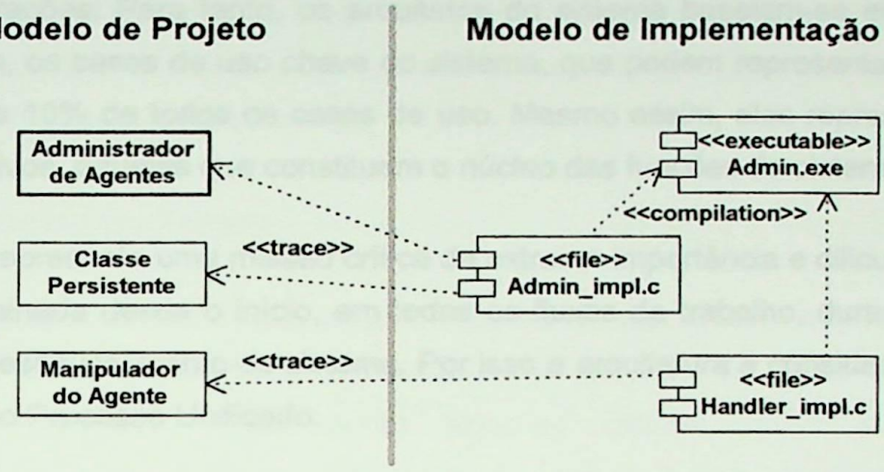

Figura 5.16 – Componentes Implementando Classes de Projeto

Estes três componentes podem ser organizados em um subsistema de implementação correspondente ao subsistema de projeto. O subsistema de implementação é manifestado através de um "mecanismo de empacotamento" real do ambiente de implementação, como *package* em Java [28] [29] e *directory* em C++ [30] [31].

A implementação da classe Persistente usa SQL [32] [33] para acessar um banco de dados simples desenvolvido em Access. A implementação do algoritmo integrado ao Visual C++ usa recursos do MatLab [34] [35].

Sistema Distribuído

A implementação da aplicação distribuída do ASMAE usando o padrão CORBA é descrita no Anexo II. Esta implementação descreve a estrutura básica desenvolvida e aplicada para todo o projeto do sistema distribuído do MAE.

5.3.8 UM PROCESSO CENTRADO NA ARQUITETURA

O conceito de arquitetura de software engloba os aspectos dinâmicos e estáticos mais significativos do sistema. Arquitetura é uma vista de todo o projeto destacando

as características mais importantes que se tornam mais visíveis. Os detalhes são deixados de lado.

A forma do sistema, ou seja, a arquitetura, deve ser projetada de modo a permitir que o sistema evolua, não somente através do seu desenvolvimento inicial, mas através de futuras gerações. Para tanto, os arquitetos do sistema baseiam-se em funções chave, ou seja, os casos de uso chave do sistema, que podem representar somente cerca de 5% a 10% de todos os casos de uso. Mesmo assim, eles representam os mais significativos, aqueles que constituem o núcleo das funções do sistema.

A arquitetura representa uma missão crítica de extrema importância e dificuldade que deve ser trabalhada desde o início, em todos os fluxos de trabalho, durante todo o processo de desenvolvimento do sistema. Por isso a arquitetura é considerada como parte central do Processo Unificado.

Para demonstrar melhor sua importância e apresentar todos os detalhes de seu desenvolvimento, o restante dessa seção (e do capítulo) será dedicado exclusivamente à arquitetura.

Vistas da Arquitetura

A descrição da arquitetura tem cinco seções, uma para cada modelo:

- Vista do Modelo de Caso de uso
- Vista do Modelo de Análise
- Vista do Modelo de Projeto
- Vista do Modelo de Implantação
- Vista do Modelo de Implementação

Nas etapas de análise, projeto e implementação, a descrição da arquitetura é a primeira atividade desenvolvida, já que ela representa a estrutura básica do sistema usada como ponto de partida para facilitar o seu desenvolvimento das demais atividades, demonstrando novamente que o processo de desenvolvimento de software unificado é centrado na arquitetura. Em outras palavras, a arquitetura representa o esqueleto de todo o sistema.

A exceção da vista do modelo de análise que não é mantida normalmente, as demais vistas da arquitetura obtidas para o sistema do MAE são descritas a seguir.

5.3.8.1 Vista do Modelo de Caso de uso

A vista arquitetônica (*architectural view*) do modelo de caso de uso exibe os atores e casos de uso mais importantes. Para fins de apresentação, o caso de uso *Registrar* identificado para o sistema do MAE, foi considerado o mais importante do ponto de vista arquitetônico. Portanto, ele será considerado daqui para frente como o foco principal de toda a arquitetura e do projeto, conseqüentemente, conforme todo o desenvolvimento já feito anteriormente.

5.3.8.2 Vista do Modelo de Projeto

A vista arquitetônica do modelo de projeto exibe os classificadores mais importantes do ponto de vista arquitetônico do modelo de projeto: os mais importantes subsistemas, interfaces, assim como algumas classes muito importantes, principalmente as classes ativas. Na etapa de projeto, foram identificadas duas classes ativas para o caso de uso *Registrar*: *Administrador de Agentes* e *Agente*.

Estas classes são incluídas na vista arquitetônica do modelo de projeto, conforme o diagrama de classes da figura 5.17.

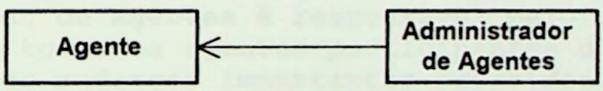

Figura 5.17 – Diagrama de Classes Ativas da Vista Arquitetônica do Modelo de Projeto

Os subsistemas que são exigidos para realizar o caso de uso *Registrar* representam os subsistemas arquitetonicamente significantes, conforme o diagrama de classes da figura 5.18. Embora a interface *Opções* não seja usada para a realização do caso de uso *Registrar*, ela aparece como parte do subsistema, mostrando o subsistema completo.

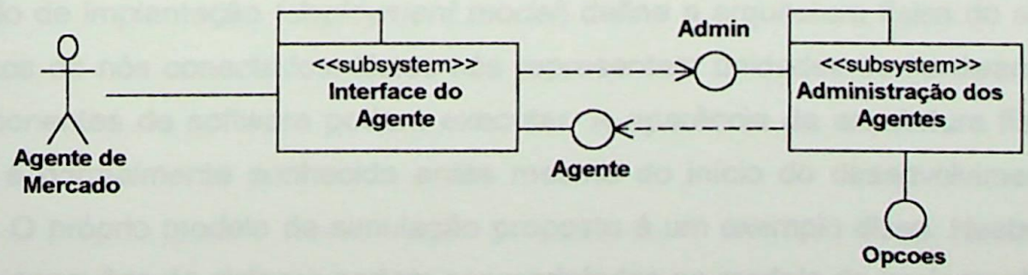

Figura 5.18 – Diagrama de Classes dos Subsistemas e Interfaces da Vista Arquitetônica do Modelo de Projeto

A interação entre os subsistemas para realizar uma instância do caso de uso, ou seja, a troca de mensagens entre os objetos das classes possuídas pelos subsistemas, é descrita pelo diagrama de colaboração da figura 5.19.

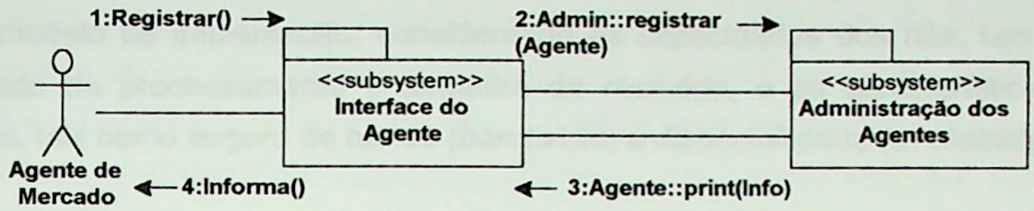

Figura 5.19 – Diagrama de Colaboração dos Subsistemas da Vista Arquitetônica do Modelo de Projeto

O fluxo de eventos correspondente é similar ao texto da realização do caso de uso apresentado na análise, só que agora considerando os subsistemas no lugar de classes.

Pré-Condição: O agente de mercado deve possuir o endereço do ASMAE.

1. O ator agente de mercado quer participar do sistema do MAE e decide se registrar no sistema.
2. A Interface do Agente requisita ao subsistema Administração dos Agentes o registro do Agente criado. O subsistema Administração de Agentes é responsável pelo controle do registro de todos os agentes participantes do sistema, divulgando as mudanças importantes ocorridas no sistema, atualizando os registros, e mantendo a persistência dos agentes registrados.
3. O subsistema Administração dos Agentes confirma o registro e informa ao subsistema Interface do Agente dos demais agentes registrados sobre a entrada do novo agente no sistema, incluindo o próprio agente que originou o registro.
4. O subsistema Interface do Agente de cada agente registrado apresenta as informações atuais do sistema.

5.3.8.3 Vista do Modelo de Implantação

O modelo de implantação (*deployment model*) define a arquitetura física do sistema em termos de nós conectados. Estes nós representam unidades de hardware onde os componentes de software podem executar. A aparência da arquitetura física do sistema é normalmente conhecida antes mesmo do início do desenvolvimento do sistema. O próprio modelo de simulação proposto é um exemplo disso. Neste caso, os nós e conexões do sistema podem ser modelados no modelo de implantação logo cedo durante a etapa de requisitos.

Durante a etapa de projeto, identificam-se quais são as classes ativas, ou seja, threads ou processos. Para cada objeto ativo, define-se o que o objeto deve fazer, qual o ciclo de vida do objeto, e como o objeto ativo deve se comunicar, sincronizar e compartilhar informações. Os objetos ativos identificados são então alocados aos nós do modelo de implantação, considerando as capacidades dos nós, tais como capacidade de processamento e tamanho de memória, e as características das conexões, tais como largura de banda (*bandwidth*) e disponibilidade (*availability*).

Os nós e conexões do modelo de implantação com a alocação dos objetos ativos nos nós podem ser descritos então nos diagramas de implantação. Estes diagramas podem mostrar também a alocação nos nós dos componentes executáveis definidos na implementação.

A vista arquitetônica do modelo de implantação do sistema de simulação do MAE é descrita a seguir. O modelo de implantação é apresentado na figura 5.20.

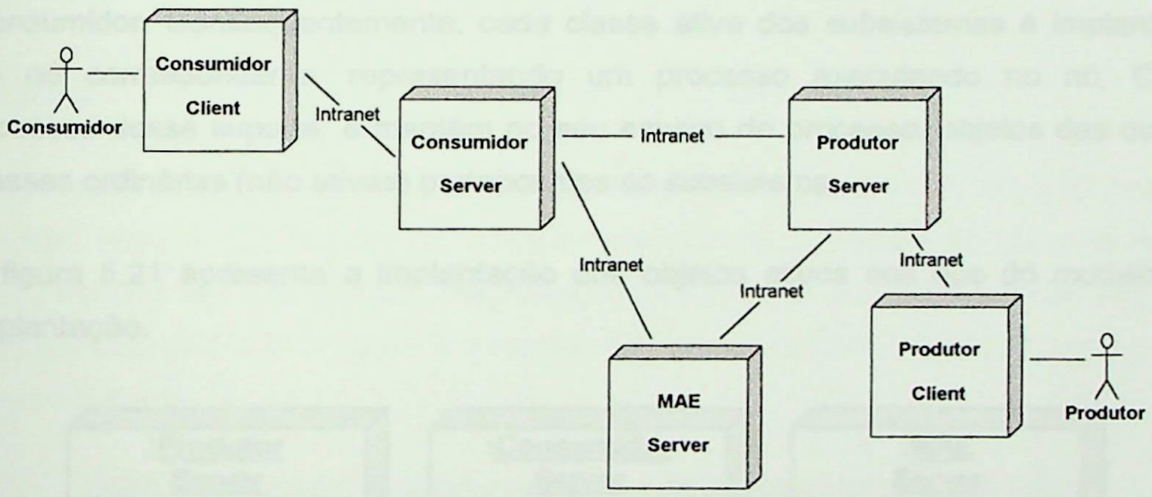

Figura 5.20 – Modelo de Implantação do Sistema de Simulação do MAE

Configuração da Rede para o Sistema de Simulação do MAE

O sistema de simulação do MAE será executado em três nós servidores e um número de nós cliente. Existe um nó servidor para os agentes consumidores e um nó servidor para os agentes produtores, representando o servidor central para os objetos de negócio e o processamento de cada agente consumidor/produtor (para um sistema mais real, o nó servidor representaria o servidor central de cada organização, como uma concessionária, por exemplo). Os usuários finais acessam o sistema através dos nós clientes, tal

como o Consumidor. Os nós se comunicam através de uma rede interna (Intranet) usando protocolo TCP/IP.

O nó servidor principal é o nó do MAE. É neste nó que serão realizados o registro dos agentes, e a cada hora computados os lances de oferta de compra e venda dos agentes consumidores e produtores para otimizar o sistema elétrico resultante e liquidar o pregão (para um sistema mais real, essas responsabilidades estariam divididas em outros nós de outras organizações, como o ONS).

Quando os nós estiverem definidos, é possível implantar funcionalidade neles.

Para o sistema do MAE, considerando somente o mesmo caso de uso *Registrar*, isto será feito com a implantação de cada subsistema completo em um único nó. O subsistema Administração de Agentes é implantado no nó servidor do MAE e o subsistema Interface do Agente é implantado nos nós servidor do Produtor e do Consumidor. Conseqüentemente, cada classe ativa dos subsistemas é implantada no nó correspondente, representando um processo executando no nó. Cada processo desse suporta, e mantém no seu espaço de processo, objetos das outras classes ordinárias (não ativas) pertencentes ao subsistema.

A figura 5.21 apresenta a implantação dos objetos ativos nos nós do modelo de implantação.

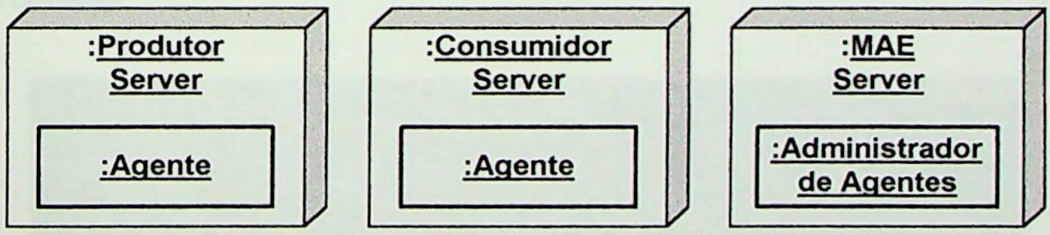

Figura 5.21 – Classes Ativas Distribuídas nos Nós do Sistema de Simulação do MAE

5.3.8.4 Vista do Modelo de Implementação

A vista do modelo de implementação considera os aspectos mais importantes do modelo de implementação, destacando a implementação de componentes críticos que fazem parte do projeto, como os componentes do tipo arquivo necessários para a implementação do sistema distribuído usando o padrão CORBA.

5.4 SISTEMA RESULTANTE

O sistema de simulação do MAE resultante do projeto desenvolvido no capítulo anterior é apresentado na seqüência pelas interfaces gráficas geradas.

5.4.1 CONSIDERAÇÕES INICIAIS

No caso dos aplicativos desenvolvidos em C++, a interface gráfica é a mesma para as plataformas do Windows 98 e NT. A versão Java, por sua vez, pode ser rodada em qualquer plataforma. Neste caso, a plataforma Linux [30] usada é o redHat 7.0. O ORB usado é o ORBacus 4.0.5. O ORB do Visibroker não chegou a ser usado na fase final.

O sistema como um todo opera no modo manual, ou seja, as ações referentes a cada agente são executadas por botões de comando. A única exceção é o ASMAE, que executa automaticamente de acordo com os eventos do sistema.

5.4.2 ASMAE

A aplicação do ASMAE é resultado do trabalhado desenvolvido durante todo o capítulo anterior. Ela representa a implementação do administrador do sistema do MAE, denominado de ASMAE, responsável basicamente pelo controle dos agentes participantes do MAE. A figura 6.1 mostra a interface desenvolvida em Visual C++ , versão console.

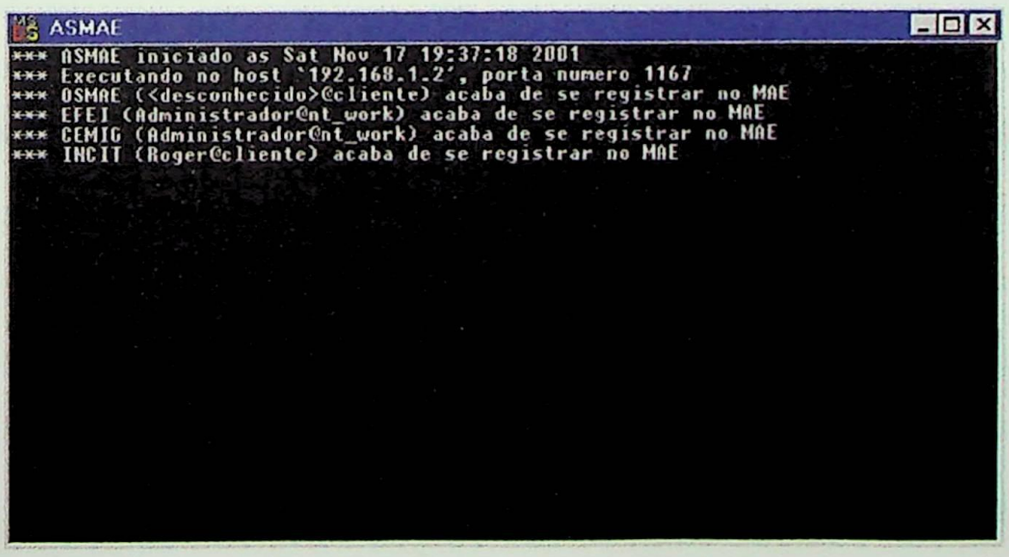

Figura 6.1 – Aplicativo do ASMAE

Como o ASMAE representa o ponto de acesso comum e conhecido do sistema do MAE, seu aplicativo deve ser o primeiro a ser executado, para que os demais aplicativos dos agentes possam se registrar no MAE. Por isso, do ponto de vista da administração do sistema do MAE, como um todo, o ASMAE representa o aplicativo mais importante do sistema.

5.4.3 ASMAE

O aplicativo Agente faz parte de todo agente de mercado, pois juntamente com o aplicativo específico de um agente de mercado deve ser executado inicialmente um aplicativo Agente. Este aplicativo representa o agente de mercado no MAE, criando e mantendo o objeto Agente usado pelo ASMAE para registrar o agente e informa-lo dos acontecimentos do MAE. A figura 6.2 mostra a interface do Agente, no caso o OSMAE, desenvolvida em Visual C++, versão MFC.

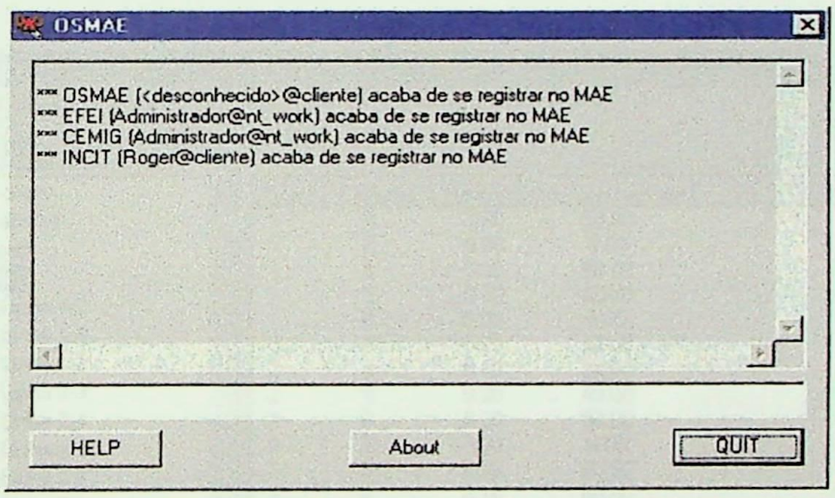

Figura 6.2 – Aplicativo Comum do Agente – versão C++

O diálogo de conexão usado para se registrar no ASMAE é ilustrado na figura 6.3.

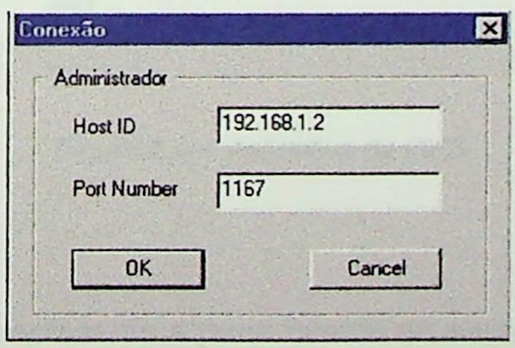

Figura 6.3 – Aplicativo Comum do Agente – Diálogo de Registro

Observe que esta caixa de diálogo é similar ao protótipo da tela de registro apresentado no capítulo anterior, durante a etapa de requisitos. Os dados de conexão são referentes às informações de acesso indicadas na segunda linha do aplicativo do ASMAE, que representa o endereço de conexão usado por todo agente do MAE.

5.4.4 ONS

O aplicativo do ONS é o mais importante do ponto de vista da dinâmica do sistema do MAE, pois ele é responsável pela parte vital de todo o sistema: a otimização do sistema elétrico. O algoritmo de programação linear desenvolvido em MATLAB é integrado com o aplicativo final.

A figura 6.4 apresenta a janela principal do ONS desenvolvida no padrão do Explorer, que facilita e agiliza a visualização das informações pertinentes.

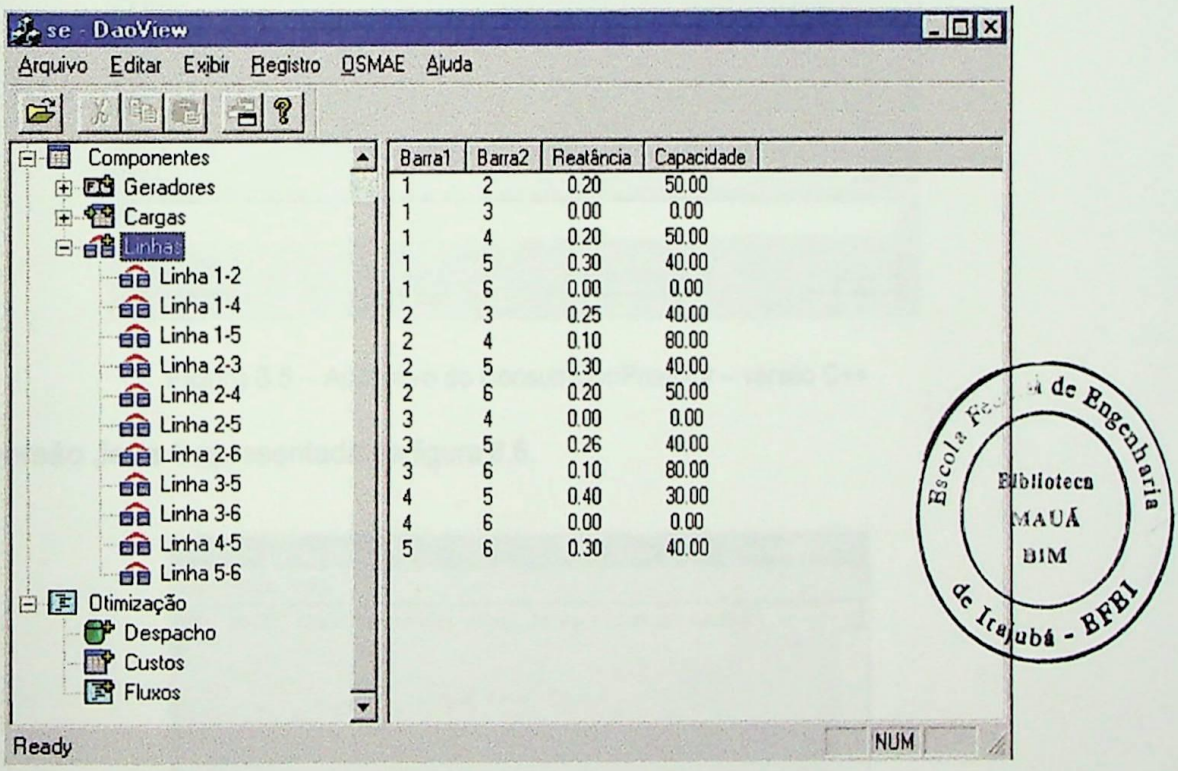

Figura 6.4 – Aplicativo do ONS

5.4.5 OSMAE

O aplicativo do OSMAE representa a parte central do sistema do MAE, do ponto de vista da operação e coordenação do sistema, pois ele é responsável pelo controle das transações do MAE.

Por motivos de simplificação, o OSMAE foi integrado ao aplicativo do ONS. O menu OSMAE da GUI da figura 6.4 contém os comandos de ação usados pelo OSMAE.

5.4.6 AGENTES CONSUMIDORES E PRODUTORES

Os aplicativos dos agentes consumidores e produtores são os aplicativos mais simples que representam a parte externa do sistema. Para simplificar a previsão das cargas e dos geradores, foi criado um vetor com 24 valores diferentes que simulam uma variação real. A figura 6.5 apresenta a janela principal de um agente consumidor/produtor desenvolvida em C++.

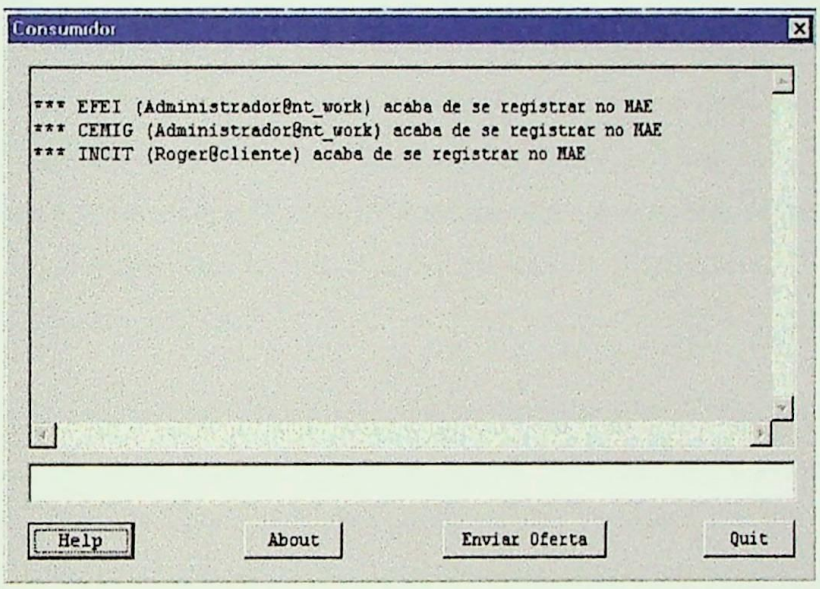

Figura 6.5 – Aplicativo do Consumidor/Produtor – versão C++

A versão Java é apresentada na figura 6.6.

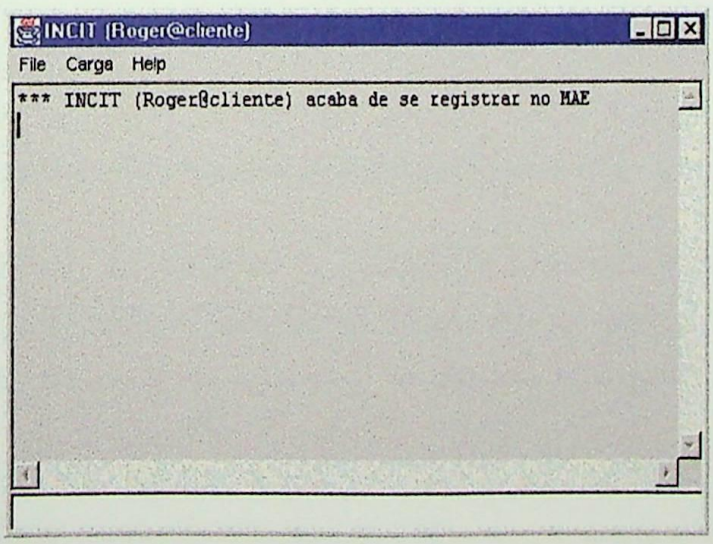

Figura 6.6 – Aplicativo do Consumidor/Produtor – versão Java

Observe que estas janelas são praticamente idênticas as janelas do Agente, a exceção de um ou outro botão de comando. Como eles representam simples clientes do sistema, sua implementação gráfica foi simplificada, mostrando somente as informações estritamente essenciais.

CAPÍTULO 6

CONCLUSÃO

6.1 CONSIDERAÇÕES FINAIS

O sistema de simulação do MAE como um mercado virtual é um sistema distribuído simplificado que não considera alguns aspectos importantes de uma aplicação prática de um sistema distribuído real, como segurança, replicação, balanceamento de carga, tolerância à falhas e pertinência, dentre outros.

Mesmo assim, apesar de não ser um sistema completo, o sistema de simulação resultante demonstra, de maneira relativamente simples, como um processo de desenvolvimento coordenado, aliado a tecnologias de software modernas, pode facilitar a implementação de um sistema dinâmico e heterogêneo tão complexo quanto o ambiente do MAE.

As trocas de informações caracterizadas por simples interfaces CORBA ilustram apenas a idéia do funcionamento de um sistema distribuído real, servindo de base para o desenvolvimento e implementação de um sistema mais completo, que contemple todos os detalhes exigidos para a operação efetiva do ambiente do MAE.

O padrão CORBA, em particular, largamente empregado em diferentes setores da economia [36] [37], destaca-se como uma tecnologia promissora para atender os requisitos de um sistema complexo e heterogêneo como o ambiente do MAE. Além disso, o padrão CORBA permite ainda a integração de outras tecnologias de objetos distribuídos, como Jini e DCOM, que atendem melhor algumas necessidades específicas, resultando em sistemas mais completos e robustos.

Conseqüentemente, as vantagens da utilização de tecnologias de software desse tipo suplantam em muito as dificuldades encontradas por ferramentas tradicionais para a solução de diferentes problemas encontrados na engenharia elétrica, em particular.

Dentre as vantagens encontradas, pode-se citar:

- A facilidade de estender o software de objetos distribuídos, em virtude do ambiente operacional das empresas que estão em constante evolução, produzindo sempre novos requisitos antes impensáveis.
- A própria possibilidade de entrada no cenário do MAE de novos agentes e também de novos recursos sem a necessidade de paralisação do sistema para realizar as devidas modificações.
- A possibilidade de atualizações de softwares sem a paralisação comum nas épocas de trocas de versão.
- A possibilidade de compartilhamento de novos recursos, dentre outras.

Portanto, as aplicações desenvolvidas com objetos distribuídos podem facilitar, muito em breve, a integração de diferentes áreas da indústria e do setor elétrico como um todo.

6.2 TRABALHOS FUTUROS

Como sugestão de trabalhos futuros pode-se destacar as seguintes inovações:

- Implementação de recursos de segurança, tolerância à falhas, replicação, dentre outros.
- Distribuição do sistema em ambientes mais complexos e heterogêneos com máquinas de diferentes tecnologias de hardware conectadas por diferentes tecnologias de redes de comunicação.
- Integração de outras tecnologias de Objetos Distribuídos, como Jini e DCOM.
- Extensão do sistema para suportar operação real-time.
- Aplicação das tecnologias e técnicas apresentadas na solução de outros problemas da engenharia elétrica, em geral.

GLOSSÁRIO DE TERMOS

***7-Layer Reference Model* (Modelo de Referência de 7 Camadas)**

Modelo conceitual inicial criado pela Organização Internacional para Padronização (ISO) para especificar como um conjunto de protocolos opera em conjunto para fornecer serviços de comunicação.

API - *Application Program Interface* (Interface para Programas Aplicativos)

O conjunto de procedimentos que um programa de computador pode chamar para acessar um determinado serviço. Os procedimentos que um programa utiliza para acessar os protocolos de rede é conhecido como uma API de rede.

***Broadcast* (Difusão)**

Uma forma de entrega na qual uma cópia de um pacote é entregue para cada computador de uma rede.

***Broadcast Address* (Endereço de Broadcast)**

Um endereço especial que faz com que o sistema usado entregue uma cópia de um pacote para todos os computadores de uma rede.

Checksum

Um valor usado para verificar que os dados não foram corrompidos durante uma transmissão. O remetente calcula um *checksum* pela adição dos valores binários dos dados, e transmite o resultado em um pacote com os dados. O destinatário calcula um *checksum* sobre os dados recebidos, e compara o valor com o *checksum* do pacote.

***Client* (Cliente)**

Quando dois programas se comunicam através de uma rede, um cliente é aquele que inicia a comunicação, enquanto o programa que aguarda para ser contactado é o servidor. Um dado programa pode atuar como um servidor para um serviço e um cliente para outro.

***Client-Server Paradigm* (Paradigma Cliente-Servidor)**

O método de interação usado quando dois programas aplicativos se comunicam através de uma rede. O aplicativo servidor aguarda em um endereço conhecido, e um aplicativo cliente entra em contato o servidor.

***Connection-Oriented* (Orientado a Conexão)**

Uma característica de sistemas de rede que exige que um par de computadores estabeleça uma conexão antes de enviar dados. Redes orientadas a conexão são análogas a um sistema de telefonia no qual uma chamada deve ser colocada e respondida antes que a comunicação inicie.

***Connectionless* (Sem Conexão)**

Uma característica de sistemas de rede que permite a um computador enviar dados para qualquer outro computador a qualquer momento. Redes sem conexão são análogas a um sistema de correio no qual cada carta carrega o endereço do destinatário; as cartas podem ser enviadas a qualquer momento.

CRC - Cyclic Redundancy Check

Um valor usado para verificar que os dados não foram corrompidos durante a transmissão. O remetente calcula um CRC e transmite o resultado em um pacote com os dados. Um destinatário calcula o CRC dos dados recebidos, e compara o valor como o CRC do pacote. Um CRC é mais complexo para calcular do que um *checksum*, mas pode detectar mais erros de transmissão.

DNS – *Domain Naming Service* (Serviço de Nomes de Domínio)

O sistema automático usado para traduzir nomes de computadores em endereços IP correspondentes. Um servidor DNS responde a uma pergunta procurando pelo nome e retornando o endereço.

***Domain* (Domínio)**

Uma parte da hierarquia de nome de um computador usada na Internet. Por exemplo, organizações comerciais possuem nomes registrados sob o domínio *.com*.

***Dotted Decimal Notation* (Notação decimal ponto)**

A notação sintática usada para expressar um endereço IPv4 de 32 bits. Cada octeto é escrito em decimal com um ponto separando os octetos.

***Encryption Key* (Chave de criptografia)**

O valor usado para criptografar os dados para garantir segurança. Em alguns esquemas de criptografia, o destinatário deve usar a mesma chave para decifrar os dados. Outros esquemas usam um par de chaves – uma para criptografar e outra para decifrar.

Ethernet

Uma tecnologia de rede de área local (LAN) popular que usa uma topologia de barramento compartilhado e acesso CSMA/CD. A Ethernet básica opera a 10 Mbps.

***Frame* (Quadro)**

A forma de um pacote que o hardware usado aceita e entrega.

Host

Um computador conectado a uma rede. Em uma internet, cada computador é classificado como um *host* ou um roteador.

internet

Um conjunto de redes conectadas por roteadores que são configurados para passarem o tráfego de dados entre computadores conectados a redes do conjunto. A maioria das internets usa protocolos TCP/IP.

Internet

A internet global que usa protocolos TCP/IP.

***Internet Address* (Endereço Internet)**

Ver endereço IP.

Internet Firewall

Um mecanismo de segurança colocado na conexão entre redes de uma organização e redes de uma organização externa. O *firewall* restringe o acesso aos computadores e serviços de uma organização.

***IP – Internet Protocol* (Protocolo Internet)**

O protocolo que define o formato dos pacotes usados um uma Internet TCP/IP e o mecanismo para rotear um pacote até seu destino.

***IP Address* (Endereço IP)**

Um endereço de 32 bits atribuído a um computador que usa protocolos TCP/IP. O remetente deve conhecer o endereço IP do computador de destino antes de enviar um pacote.

ISO – International Organization for Standardization

A organização dos padrões mais conhecidos por ter proposto o modelo de referência de 7 camadas.

Java

Uma linguagem de programação criada pela *Sun Microsystems* para uso principalmente em documentos da *World Wide Web*. Programas Java são compilados em uma representação *bytecode*. Depois que um *browser* carrega um programa Java, o programa executa localmente para controlar a tela.

LAN – *Local Area Network* (Rede de Área Local)

Uma rede que usa tecnologia projetada para cobrir uma pequena área geográfica. Por exemplo, uma Ethernet é uma tecnologia de LAN apropriada para uso em simples edificações.

***Layering Model* (Modelo de Divisão em Camadas)**

Uma estrutura conceitual usada para explicar o propósito e a interação entre um conjunto de protocolos. Criar camadas é apropriado principalmente para projetistas de protocolos; uma vez implementado, os protocolos podem ser usados sem a necessidade de entender a divisão em camadas.

Multicast

Uma forma de endereçamento na qual um único endereço é atribuído a um conjunto de computadores; uma cópia dos dados enviados a este endereço é entregue a cada um dos computadores do conjunto.

***Parit bit* (Bit de Paridade)**

Um bit extra adicionado a uma unidade de um dado, normalmente a cada caracter, para verificar que o dado é transferido sem ser corrompido. O destinatário verifica a paridade de cada unidade de dado.

***Point-to-Point Network* (Rede ponto a ponto)**

Qualquer tecnologia de rede que usa uma tecnologia não compartilhada para conectar pares de computadores. A tecnologia de ponto a ponto é mais popular em redes de áreas extensas (WAN) do que em redes locais (LAN).

***Protocol* (Protocolo)**

Um projeto que especifica os detalhes de como os computadores interagem, incluindo o formato de mensagens que eles trocam e como os erros são tratados.

***Protocol Suite* (Pilha de Protocolos)**

Um conjunto de protocolos que trabalha conjuntamente para fornecer um sistema de comunicação geral. Cada protocolo manipula um subconjunto de todos os possíveis detalhes. A Internet usa a pilha de protocolos do TCP/IP.

***Router* (Roteador)**

O bloco de construção básico de uma internet. Um roteador é um computador conectado a duas ou mais redes que encaminha os pacotes das redes de acordo com a informação encontrada na sua tabela de roteamento. Os roteadores na Internet executam o protocolo IP.

***Server* (Servidor)**

Quando dois programas se comunicam através de uma rede, um cliente é aquele que inicia a comunicação, enquanto o programa que aguarda para ser contactado é o servidor. Um dado programa pode atuar como um servidor para um serviço e um cliente para outro.

***Socket API* (API socket)**

Um conjunto de procedimentos que um programa aplicativo pode usar para se comunicar através de uma rede. Este nome surgiu porque o conjunto inclui um procedimento *socket* que deve ser chamado para estabelecer uma comunicação.

TCP – *Transmission Control Protocol* (Protocolo de Controle de Transmissão)

O protocolo TCP/IP que fornece aos programas aplicativos acesso a um serviço de comunicação orientado a conexão. TCP oferece entrega confiável e de fluxo controlado. Além disso o TCP acomoda condições de mudança na Internet adaptando seu esquema de retransmissão.

TCP/IP

A pilha de protocolos usada na Internet. Embora a pilha contenha muitos protocolos, TCP e IP são dois dos mais importantes.

Token Ring

Uma topologia de rede em anel que usa a passagem de um *token* (bastão) para o controle de acesso à rede. O *token* consiste de uma mensagem especial enviada ao longo do anel. Quando uma estação tem um pacote para enviar, a estação aguarda até o *token* chegar, envia o pacote, e então envia o *token.*

UDP – *User Datagram Protocol* (Protocolo de Datagrama do Usuário)

O protocolo TCP/IP que fornece aos programas aplicativos um serviço de comunicação sem conexão.

WWW – *World Wide Web* (Rede Mundial de Computadores)

O sistema de *hypermedia* usado na Internet no qual uma página de informação pode conter texto, imagens, áudio ou vídeo clips, e referências para outras páginas.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Coulouris, G.; Kindberg, T.; Dollimore, J. "Distributed System – Concepts and Design" – Addison Wesley - 2001

[2] Tanenbaum, Andrew S. "Modern Operating Systems". Prentice Hall, Inc. 1992.

[3] Comer, Douglas E.. "Computer Networks and Internets". 2ª Edição. Prentice Hall, Inc. 1999.

[4] Comer, Douglas E.; Stevens, David L. "Interligação em Rede com TCP/IP Volume I". Editora Campus Ltda.1999.

[5] Comer, Douglas E.; Stevens, David L. "Internetworking with TCP/IP Volume III: Client-Server Programming and Applications with Windows Sockets Version". Prentice Hall, Inc. 1997.

[6] Page-Jones, M. "Fundamentals of Object-Oriented Design in UML". Addison Wesley, 2000.

[7] Microsoft Corporation. "DCOM Technical Overview". White Paper. 1996.

[8] Horstmann, M.; Kirtland, M. "DCOM Architecture". Microsoft Corporation. 1997.

[9] "Jini Technology Core Platform Specification". Sun microsystems. Versão 1.1. 2000.

[10] "Jini Architecture Specification". Sun microsystems. Versão 1.1. 2000.

[11] "Java Remote Method Invocation Specification". Versão 1.3.0. 1999.

[12] Li, Sing. "Professional Jini". Wrox Press. 2000.

[13] OMG : http://www.omg.com

[14] "The Common Object Request Broker: Architecture and Specification". OMG, Revised Edition 1995.

[15] Henning, M.; Vinoski, S. -"Advanced CORBA® Programming with C++"- Addison Wesley, 1999.

[16] Ferreira, R.D.F; Gavião, F.; Torres, G. L. "Um Modelo de Simulação para o Mercado Atacadista de Energia – MAE". CIGRÉ-CIER, Peru. 2000.

[17] ASMAE – http://www.asmae.com.br.

[18] "Perfil do Setor de Energia Elétrica". Revista Eletricidade Moderna, n° 304, Julho de 1999.

[19] Santos A. H. M., Haddad J., Cabral R. S., "A operação de Sistemas Hidrotérmicos e o uso múltiplo das águas". III Congresso Brasileiro de Planejamento Energético, São Paulo, Junho de 1998.

[20] Lima, José W.M. "Curso de Economia do Setor Eletro-Energético". Parte I, 1999.

[21] Matthew, Neil; Stones, Richard. "Programmation Linux". Wrox Press. 2000.

[22] ORBacus : http://www.ooc.com.

[23] "ORBacus For C++ and Java". Versão 4.0.3.

[24] Visibroker :http://www.inprise.com.

[25] Jacobson, I.; Booch, G.; Rumbaugh, J. "The Unified Software Development Process".Addison Wesley. 1999.

[26] Ambler, Scott W. "Building Object Applications That Work" – Cambridge University Press – 1998.

[27] Booch, G. "Object Oriented Analysis and Design with Applications" – 2nd Edition Benjamin/Cummings - Redwood City – CA – 1994.

[28] Eckel, Bruce. "Thinking in Java". 2ª Edição. Prentice Hall, Inc. 2000.

[29] Deitel, H. M.; Deitel, P. J. "Java Como Programar". 3ª Edição. Bookman. 2001.

[30] Lafore, R. "Object-Oriented Programming in C++". 3ª Edição. Sams. 1999.

[31] Bernardi, A. "Curso de Extensão Universitária – Visual C++, Microsoft Foundation Class – Fundamentos". 2001.

[32] Date, C. J. "An Introduction to Database Systems". 7ª Edição. Addison Wesley Longman, Inc. 2000.

[33] Davidson, Louis. "Professional SQL Server 2000 Database Design". Wrox Press. 2001.

[34] "MATLAB C Math Library User's Guide". Versão 2.1. The MathWorks, Inc. 2000.

[35] "MATLAB C++ Math Library User's Guide". Versão 2.1. The MathWorks, Inc. 2000.

[36] Proceedings do 5th Congresso de Objetos Distribuídos – realizado em São Paulo – Centro de Convenção Rebouças – 2000 – Exemplos de Empresas adotando o modelo de sistemas distribuídos. Home Page do OD: www.objetosdistribuidos.com.br.

[37] Ferreira, Maurício G.V.; Nakanishi, T.; Cunha, João B. S. - Distributed Objects and Agents in the Satellite Control Software – INPE – Brazilian National Institute of Space Research - 2001.

ANEXO I – ALGORITMO DE OTIMIZAÇÃO

O arquivo plinear.m listado a seguir contém o algoritmo completo de otimização usado pelo ONS para o cálculo do despacho ótimo do sistema elétrico.

```
%Interface da Funcao de Programacao Linear usada pelo programa C++
function [x,fval,exitflag,output,lambda]=plinear(NB, NG, CG, PG,
C, NL, R, CL, options)

% ********************************************************************
% A funcao precisa dos seguintes parametros de entrada
% Sistema:
%       NB = Numero de Barras
% Gerador:
%       NG = Numero de Geradores
%       CG = Vetor com o custo dos geradores, em R$/MW
%       PG = Vetor Potencia de cada Gerador, em pu
% Carga:
%       C  = Vetor Carga de cada Barra, em pu
% Linha:
%       NL = Numero de Linhas
%       R  = Vetor Reatancia das Linhas, em pu
%       CL = Vetor Capacidade de cada Linha, em p
%
% Resultados:
%       x = Potencia dos Geradores[pu], e angulo das Barras [rad]
%       cb = custo das barras, em R$/MW, ou seja, da carga
%       ce = Custo excedente dos geradores e linhas
%
% ********************************************************************

% ********** Montagem da Matriz Y de Admitancias ***********
Linha = zeros(NB,NB);
Aux = 1;                              % Posiciona a Reatancia atual
for i=1:NB;
  for j=1:NB;
       if ((i ~= j)&(j > i))
       %Monta matriz Y - admitancias. No caso, susceptancia
             Linha(i,j) = R(Aux);    % Base correta ja

             % Se diferente de zero, a linha existe.
             if ((Linha(i,j)~=0)&(j > i) )
                   % Salva susceptancia da linha existente.
                   Linha(i,j)=1/Linha(i,j);
                   Linha(j,i)=Linha(i,j);  % Matriz Simetrica
             end;
             Aux = Aux + 1;
       end;
  end;
end;
% ********************************************************************

% ********************************************************************
% ********** Matrizes para calculo do Despacho Otimo *************
% As variaveis do sistema sao:
% x1 - Gerador 1
% x2 - Gerador 2
```

```
% xn - Gerador n
% a1 - angulo Barra 1
% a2 - angulo Barra 2
% an - angulo Barra n
% num total cd NG + NB variaveis
%
% O sistema se compoem ao todo das seguitnes restricoes:
% - de igualdade (depende da carga-fechar demanda)
% - da capacidade maxima de cada gerador
% - as restantes sao capacidade maxima das linhas de transmissao
% - mais a swing
%
% O fluxo de potencia entre barras sera dado por:
% FP = SxDelta(a)
%*****************************************************************

%*****************************************************************
% Vetor Custo : define os custos referentes a cada variavel do
% sistema. Os primeiros custos referem-se ao custo de cada
% gerador(variaveis xi). Os restantes sao os custos de cada barra.

NV=NG+NB;                    % Define o numero total de variaveis

for i=1:NG;
   if(CG(i)>0)
         c(i)=CG(i); % O vetor custo, com os custos dos geradores
   end;
end;

% Passo - alocacao dos custos das barras. Completa vetor dos
custos.
P=NG+1;
for i=P:(NB+P-1)
   c(i)=0;                   % Completa com os custos das barras que
e zero
end;

% *****************************************************************
% ****************** Determinacao da Swing ************************
Aux=PG(1);
for i=1:NB;
   if PG(i)>=Aux
       Aux=PG(i);            % Salva a barra que tem o maior gerador
       S=i;                  % Determina o numero da barra swing
   end;
end;
% *****************************************************************

% *****************************************************************
% ************** Vetor de Restricoes - Limites ********************
% Define os limites de operacao, ou seja, capacidades das linhas
% (fluxo de potencia) e dos geradores.
% As desigualdades referentes as capacidades das linhas sao em
% modulo. Portanto, existem duas restricoes:uma de limite superior
% e uma inferior.Contudo, deve-se entrar somente com desigualdades
% menor ou igual para o calculo de otimização.
% Restricoes das barras: Somatorio(gj)-Somatorio(fluxos nas
% linhas)=Carga
% O numero de restricoes sera dado pela soma do:
% - NG - numero de geradores. Restricao de geracao (capacidade).
```

```
% - NB - numero de barras. Restricao de demanda(geracao + fluxos).
% - NL - numero de linhas. Restricao das capacidades das linhas.
% Dobrado devido a desigualdade
% - 1 - restricao da swing
% Para as linhas, a restricao de desigualdade negativa e
% transformada em positiva, pois o metodo de otimizacao so resolve
% desigualdades menor ou igual.

NR=NG+NB+(NL*2)+1;  % Numero de Restricoes
for i=1:(2*NB+1);
  %Primeiro as restricoes das barras, que sao as restricoes de
igualdade.
  if i<=NB
       b(i)=C(i);          % Equivale as cargas (demandas) da
     barra.
  elseif i==(NB+1)
       b(i)=0;             % Equivale a barra swing. Teta igual a
     0.
  elseif PG(i-NB-1)>0
       b(i)=PG(i-NB-1);    % Equivale capacidade dos geradores -
                           % desiguladade.
  end;
end;

Aux=1;
NLP=NB*(NB-1)/2;
for i=1:NLP;
  if(CL(i)>0)
       b(2*Aux+NG+NB)=CL(i);   % Equivale a capacidade das
     linhas.
       b(2*Aux+NG+NB+1)=CL(i);% limite superior e inferior.
       Aux = Aux + 1;
  end;
end;
% ********************************************************************

% ********************************************************************
% ********** Matriz dos coeficientes das restricoes **************
% A matriz foi dividida entre as varias restricoes para facilitar
% a logica.
% ******************  Restricoes das barras **********************
% Geradores das barras
A = zeros(NR,NV);
Aux=1;
for i=1:NB;
  % Verifica geradores da barra
  if (PG(i)>0)
       A(i,Aux)=1;  % Existe gerador
       Aux = Aux + 1;
  end;
end;
% Angulos das Barras
for i=1:NB;                % Fixa restricao atual.
  for j=(NG+1):NV;         % Fixa a barra atual de referencia, que
e
                           % a variavel.
       A(i,j)=0;
       if i==(j-NG);       % Se a variavel for igual a restricao
            for h=1:NB;    % Para a barra fixada, varre-se as
                           %susceptancias
```

```
                    A(i,j)=A(i,j)-Linha(i,h);
                end;
            else
                A(i,j)=Linha(i,j-NG);
            end;
    end;
end;
% ************* Restricao da swing ***************
for i=1:NB;
    if i==S
        A(NB+1,i+NG)=1;    % Barra j e a swing
    else
        A(NB+1,i+NG)=0;
    end;
end;
% ************* Restricao dos geradores *************
for i=(NB+2):(NB+1+NG)
    for j=1:NV;
    % Faz-se um offset negativo da linha, como se fosse uma matriz
    ixj
        if j==(i-NB-1)
            A(i,j)=1;
        else
            A(i,j)=0;
        end;
    end;
end;
% ************** Restricao das linhas **************
id=NB+NG+1;                % Deslocamento da linha na matriz
linha=1;
for i=1:NB;                % Fixa barra na matriz de susceptancia
    linha = linha -1;
    for j=1:NB;            % Verifica ligacoes
        % A(i+linha+id,i+NG)=0;
        if ((Linha(i,j)~=0)&(j>=i))

        % Desigualdade limite superior - normal
            A(i+linha+id,i+NG)=Linha(i,j);  %Fluxo da barra
         fixa
            A(i+linha+id,NG+j)=-Linha(i,j); %Fluxo barra
        conectada

        %Desigualdade Limite inferior.Para que seja <=,faz-se o -
            A(i+linha+id+1,i+NG)=-Linha(i,j); %Fluxo da barra
           fixa
            A(i+linha+id+1,NG+j)=Linha(i,j);%Fluxo barra
        conectada

            linha=linha+2; %incrementa a linha para a mesma
           barra
        end;
    end;
end;
% ***********************************************************
% ***********************************************************
% *********** Matriz dos coeficientes das restricoes *********
% ***********************************************************
% Numero de restricoes de igualdade
N=NB+1;              % Numero de barras mais 1 que e a barra swing
% ***********************************************************
```

```
% ********************************************************************
% Programacao Linear. Resolucao da Otimizacao.
% Funcao
% [x,lambda] = LP(f,A,b,VLB,VUB,X0,N)
% sendo : f - vetor dos custos (vetor c). Dimensao n.
% A -matriz dos coeficientes das restricoes (matriz A).Dimensao
nxm
% b - vetor dos limites (vetor b). Dimensao m.
% VLB - vetor dos limites inferiores das restricoes. Dimensao m.
% VUB - vetor dos limites superiores das restricoes. Dimensao m.
% X0 - vetor do chute inicial. Dimensao m.
% N - total das restricoes de igualdade, a partir da primeira.
% x - vetor resposta da solucao das variaveis.
% lambda - vetor dos custos associados - custo marginal.
% n - numero de variaveis
% m - numero de restricoes

% Define limites superiores e inferiores para as variaveis
for i=1:NG;
   VLB(i)= 0;
   VUB(i)= PG(i);
end;

for i=(NG+1):NV;
   VUB(i)= pi;
   VLB(i)=-pi;
end;

% Cria as matrizes de igualdade
for i=1:(NB+1);
   for j=1:(NV);
         Aeq(i,j) = A(i,j);
   end;
   beq(i)=b(i);
end;

% Apaga as linhas da matriz A, eliminando as restricoes de
igualdade
for i=1:(NB+1);
   A(1,:) = [];
   b(1) = [];
end;

%[x,fval,exitflag,output,cx] = linprog(f,A,b,Aeq,beq,VLB,VUB,x0)
[x,fval,exitflag,output,lambda] =
   linprog(c,A,b,Aeq,beq,VLB,VUB,[],options);
```

ANEXO II – IMPLEMENTAÇÃO CORBA

O Administrador do sistema é a aplicação central de todo o sistema de simulação do MAE. Ele representa o único ponto de acesso comum conhecido inicialmente e usado por todo agente de mercado que deseja participar do MAE. Em outras palavras, sem o administrador, não existe sistema.

Como parte central do sistema, a implementação do Administrador do sistema deverá atender o requisito de distribuição exigido, mais especificamente o uso do padrão CORBA apresentado no capítulo 3.

Para criar a aplicação do Administrador do sistema, é necessário implementar o subsistema Administração dos Agentes criado durante o projeto, cujo modelo de implementação correspondente foi apresentado na implementação (figura 5.16).

Neste modelo de implementação foram criados dois componentes do tipo arquivo (*file componente*): Admin_impl.c, correspondente à classe ativa de projeto Administrador de Agentes, e Handler_impl.c, correspondente à classe de projeto Manipulador do Agente. O componente executável Admin.exe é a aplicação final do Administrador do sistema propriamente dita.

Considerações Iniciais

A descrição dos passos necessários para desenvolver uma aplicação CORBA independem da implementação CORBA escolhida.

Contudo, existem diferenças importantes, como os nomes gerados nos códigos, e os próprios códigos em si, que podem confundir quando comparadas com outras implementações.

Para a implementação do sistema de simulação do MAE, o padrão CORBA usado é o ORBacus 4.05.

Portanto, os resultados apresentados daqui para frente se referem às características específicas da implementação do ORBacus, eliminando qualquer dúvida que possa surgir com a comparação dos resultados de outras implementações.

IDL

O primeiro passo para a implementação de qualquer aplicação CORBA, é definir a IDL CORBA, ou seja, definir as interfaces dos objetos CORBA que farão parte do sistema distribuído.

A interface IDL, neste caso, é a própria interface do subsistema Administração dos Agentes identificada na etapa de projeto. As operações da interface são as mesmas definidas no projeto da classe Administrador de Agentes. O arquivo Admin.idl que contém a IDL resultante é o seguinte:

```
#include <Agente.idl>

module MAE                    // Define o escopo do projeto
{
   // Estrutura que contém as informações necessárias para
   // o registro do agente
   struct AgenteDesc
   {
         Agente ag;  // Objeto CORBA do Agente
         string id;
         string host;
         string name;
   };

   exception AgenteExists { };
   exception UnknownAgente { };
   exception AgenteUnreachable { };

   // Interface do Subsistema Administração dos Agentes
   interface Admin
   {
         // Operação usada para registrar o agente
         AgenteHandler registrar(in AgenteDesc agente)
               raises(AgenteExists);
   };

   ...
}
```

Observe que a operação registrar() retorna um valor: AgenteHandler. Este valor, na verdade, representa o objeto AgenteHandler, associado a cada agente registrado, da classe AgenteHandler que implementa a classe de projeto Manipulador do Agente. Este comportamento está de acordo com o caso de uso *Registrar*, realizando a parte específica que relata a criação de um manipulador associado ao agente após a efetivação do seu registro.

Mapeamento da IDL

O passo seguinte consiste na tradução da IDL para a linguagem de implementação escolhida. Os arquivos fonte gerados a partir da compilação da IDL definida anteriormente para a linguagem C++ são os seguintes:

- *Admin.h*: arquivo de cabeçalho incluído no código fonte do cliente. Este arquivo contém as definições de tipo do C++ correspondentes aos tipos usados na IDL.
- *Admin.cpp*: este arquivo contém o código C++ dos *stubs* a serem compilados e ligados com a aplicação cliente. Ele fornece uma API usada pela aplicação cliente para comunicação com os objetos definidos na IDL, ou seja, o objeto CORBA Admin.
- *Admin_skel.h*: arquivo de cabeçalho incluído no código fonte do servidor. Este arquivo contém definições que permitem implementar uma interface de chamada (*up-call*) para os objetos definidos na IDL.
- *Admin_skel.cpp*: este arquivo contém o código C++ dos *skeletons* a serem compilados e ligados com a aplicação do servidor. Ele fornece o suporte em tempo de execução exigido pela aplicação do servidor, para que ela possa receber as invocações das operações enviadas pelos clientes.

Implementação da Classe Servant do Administrador

A implementação da classe *Servant* correspondente ao objeto CORBA *Admin* definido na IDL é dividida em dois arquivos: o arquivo de cabeçalho Admin_impl.h contendo a definição da classe *servant*, e o arquivo do código fonte Admin_impl.cpp que implementa a classe *servant*.

A forma básica do *servant* do administrador é determinada pela classe *skeleton* produzida pelo compilador da IDL: *POA_MAE::Admin*. O *servant* do administrador deve fornecer pelo menos a implementação das operações definidas na interface Admin.

A listagem parcial do arquivo de cabeçalho Admin_impl.h é apresentada a seguir.

```
namespace MAE                    // Define o escopo do Projeto
{
// Classe Servant
class Admin_impl : public POA_MAE::Admin,
                   public PortableServer::RefCountServantBase
{
```

```
        CORBA::ORB_var       orb_;
        PortableServer::POA_var poa_;

        // Definição do formato do registro de cada agente
        struct AgenteListItem
        {
            AgenteDesc agenteDesc;
            CORBA::ULong handlerId;
        };

        typedef OB_STL::list<AgenteListItem> AgenteList;

        // Lista interna contendo os registros de todos os agentes
        // participantes do MAE.
        AgenteList agenteList_;

        // Identificador interno do manipulador associado ao agente
        CORBA::ULong nextHandlerId_;

        // Método interno usado para excluir o registro de um
        // agente da lista
        void remove_agente(CORBA::ULong);

    public:

        // Operações internas
        Admin_impl(CORBA::ORB_ptr orb,
                   PortableServer::POA_ptr poa);

        PortableServer::POA_ptr _default_POA();

        ...                    // Demais operações da classe

        // Operações da IDL
        virtual AgenteHandler_ptr registrar(const AgenteDesc&)
            throw(AgenteExists, CORBA::SystemException);
    }

    } // namespace MAE
```

A listagem parcial do arquivo Admin_impl.cpp contendo o código fonte da implementação da operação registrar() definida na interface Admin da IDL é apresentada a seguir.

```
    AgenteHandler_ptr
    Admin_impl::registrar(const AgenteDesc& agente)
        throw(AgenteExists, CORBA::SystemException)
    {
        // Verifica Se a referência do objeto é nil
        if(CORBA::is_nil(agente.ag)) {
            return AgenteHandler::_nil();
        }

        // Verifica se o agente já existe
        AgenteList::iterator iter;
        for (iter = agenteList_.begin(); iter != agenteList_.end();
                iter++)
            if(strcmp(((*iter).agenteDesc.name), agente.name) == 0)
```

```
                throw AgenteExists();

        // Acrescenta o agente à lista de registros
        AgenteListItem newAgente;

        newAgente.agenteDesc = agente;
        newAgente.broadcasterId = nextBroadcasterId_;
        agenteList_.push_back(newAgente);

        // Cria o servant do manipulador associado ao agente
        AgenteHandler_impl * handlerImpl =
             new AgenteHandler_impl(this, nextHandlerId_);

        // Avança  o identificador do manipulador
        nextHandlerId_++;

        // Cria o objeto CORBA usando o POA padrão (=rootPoa)
        AgenteHandler_var newHandler = handlerImpl->_this();

        // Cria a mensagem a ser divulgada aos demais agentes
        // registrados
        static const char* msg = ") acaba de se registrar no MAE";

        CORBA::String_var message =
            CORBA::string_alloc(4 + strlen(agente.name) +
                                2 + strlen(agente.id) +
                                1 + strlen(agente.host) +
                                strlen(msg));

        strcpy(message.inout(), "*** ");
        strcat(message.inout(), agente.name);
        strcat(message.inout(), " (");
        strcat(message.inout(), agente.id);
        strcat(message.inout(), "@");
        strcat(message.inout(), agente.host);
        strcat(message.inout(), msg);

        // Método interno usado para divulgar o novo agente,
        // excluindo e divulgando os agentes que não puderem ser
        // contactados
        broadcast(message);

        // Retorna o manipulador do agente usado para o agente poder
        // ter acesso aos serviços oferecidos pelo sistema.
        return newHandler._retn();
}
```

Implementação da Aplicação do ASMAE

O arquivo Administrador.cpp contém o código fonte do administrador do sistema, conforme a listagem parcial a seguir. A aplicação resultante é uma versão console.

```
int
run(CORBA::ORB_ptr orb, int argc, char* argv[])
{
     // Passos de Inicialização
     ...
```

```
        // Cria o objeto de implementação, ou seja, o servant
        MAE::Admin_impl* AdminImpl =
                    new MAE::Admin_impl(orb, poa);
        …

        // Imprime mensagem "Administrador Iniciado" com
        // as informações do host e da porta do Administrador
        // do sistema usadas pelos agentes para se conectarem
        …

        // Roda a implementação do Administrador do Sistema
        manager -> activate();
        orb -> run();

        return EXIT_SUCCESS;
}

int
main(int argc, char* argv[], char*[])
{
        int status = EXIT_SUCCESS;
        CORBA::ORB_var orb;

        try {
            orb = CORBA::ORB_init(argc, argv);
            status = run(orb, argc, argv);
        }
        catch(const CORBA::Exception& ex) {
            cerr << ex << endl;
            status = EXIT_FAILURE;
        }

        if(!CORBA::is_nil(orb)) {
            try         {
                    orb -> destroy();
            }
            catch(const CORBA::Exception& ex) {
                    cerr << ex << endl;
                    status = EXIT_FAILURE;
            }
        }

        return status;
}
```

Implementação do Agente de Mercado

O código fonte parcial de um agente de mercado é listado a seguir. O aplicativo resultante é uma versão MFC do Visual C++.

```
// Método de Inicialização de uma aplicação MFC
BOOL CAgenteApp::InitInstance()
{
        // Inicialização Padrão
        …
```

```
// Dialogo de conexão com o Administrador do Sistema
// Corresponde ao protótipo da UI para o caso de uso
// Registrar, ou seja, a classe Registro UI usada pelo
// agente de mercado.
CRegistroDlg registroDlg;
int nResponse = registroDlg.DoModal();

// Se o agente de mercado confirmar o registro
// inicia-se o processo de conexão com o Administrador.
// Senão, a aplicação é finalizada
if (nResponse == IDOK) {
    // Salva as informações do endereço fornecido
    const char* host = registroDlg.m_EditHost;
    const char* port = registroDlg.m_EditPort;
    CORBA::UShort portNum = atoi(registroDlg.m_EditPort);

    OBCORBA::ORB_var orb;

    // Passos de Inicialização
    …

    try {
          …

          // Prepara o endereço para conectar-se com o
          // Administrador
          CORBA::String_var str =
                    CORBA::string_dup("corbaloc::");
          str += host;
          str += ":";
          str += port;
          str += "/Admin";

          // Obtem objeto CORBA do Administrador do sistema
          CORBA::Object_var obj =
                    orb -> string_to_object(str);
          assert(!CORBA::is_nil(obj));

          // Cria o objeto proxy do Administrador
          MAE:: Admin_var server =
                     MAE:: Admin::_narrow(obj);
          assert(!CORBA::is_nil(server));

          …

          // Cria o objeto Agente - servant do Agente
          // usado pelo Administrador do sistema
          …

          // Executa a implementação
          // A operação registrar() é chamada a partir
          // deste ponto em algum lugar.
          // Aqui, ela é a função stub do proxy
          …
    }
    catch(const CORBA::Exception& ex)  {
          CORBA::String_var msg = ex._to_string();
          logger -> error(msg);
    }
```

```
            if(!CORBA::is_nil(orb)) {
                  try {
                        orb -> destroy();
                  }
                  catch(const CORBA::Exception& ex) {
                        CORBA::String_var msg = ex._to_string();
                        logger -> error(msg);
                  }
            }
      } // Fim da Aplicação

      return false;
}
```