

Um Método Orientado a
Objetos de Desenvolvimento de
Sistemas de Informação
Distribuídos

Henrique Wiermann Paques

UNIDADE	BC
N.º CHAMADA	T/UNICAMP
V.	Ex
TOMBO BC	31156
PROG.	281197
C	<input type="checkbox"/>
D	<input checked="" type="checkbox"/>
PREÇO	R\$ 11,00
DATA	24/07/97
N.º CPD	

CM-0009324-5


**FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA
DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO
CIENTÍFICA - UNICAMP**

Si38i	<p>Paques, Henrique Wiermann Um método orientado a objetos de desenvolvimento de sistemas de informação distribuídos / Henrique Wiermann Paques. — Campinas, SP : [s.n], 1997. Campinas, SP : [s.n], 1997.</p> <p align="center">Orientador : Geovane Cayres Magalhães Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.</p> <p>1. Engenharia de software. 2. Análise e projeto de software. 3. Programação orientada a objetos (computação).</p> <p>I. Magalhães, Geovane Cayres . II. Universidade Estadual de Campinas - Instituto de Computação. III. Título.</p>
-------	---

Um Método Orientado a Objetos de Desenvolvimento de Sistemas de Informação Distribuídos

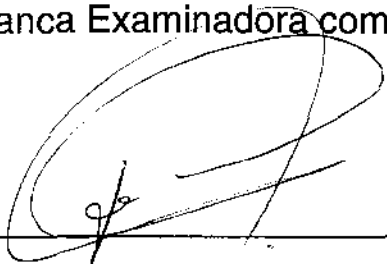
Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por Henrique Wiermann Paques e aprovada pela Banca Examinadora.

Campinas, 28 de Maio de 1997.

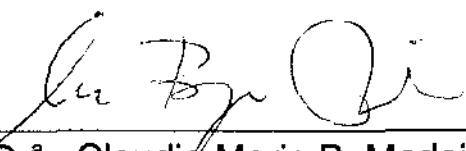

Geovane Cayres Magalhães
Orientador_

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Tese de Mestrado defendida e aprovada em 05 de maio de 1997
pela Banca Examinadora composta pelos Professores Doutores



Prof. Dr. José Carlos Maldonado



Prof.^a. Dr.^a. Cláudia Maria B. Medeiros



Prof. Dr. Geovane Cayres Magalhães

Agradecimentos

Agradeço ao meu amigo e orientador prof. Dr. Geovane Cayres Magalhães pela orientação e incentivos recebidos.

À profa. Dra. Claudia Maria Bauzer Medeiros pela sua ajuda.

À minha esposa Sonia por ter tido paciência e estado sempre ao meu lado.

Aos meu pais, Antonio e Otilia, por tudo que fizeram por mim.

Ao meu amigo e companheiro Nevtton Cereja pelo apoio e incentivo.

Este trabalho foi financiado pela FAPESP, processo 94/4131-0.

Parte dos resultados descritos neste trabalho foram desenvolvidos no CPqD/TELEBRÁS através do convênio de Cooperação Técnica TELEBRÁS/UNICAMP número P&D/DAI/126/95-JDPqD

Resumo

Com a viabilização de um número cada vez maior de tecnologias distribuídas (tanto no nível de hardware como no de software), a procura pelo desenvolvimento de sistemas de informação distribuídos, por parte de empresas de grande porte, tem aumentado consideravelmente nos últimos tempos. Porém, há uma carência de métodos que tratem o aspecto de distribuição desde sua fase inicial (análise de requisitos) até sua fase final (implementação). O que existem são especificações de arquiteturas distribuídas (por exemplo, CORBA do OMG) que provêem suporte apenas às atividades relacionadas ao processo de implementação do software (o processo de análise não é contemplado). Neste trabalho é apresentado um método orientado a objetos (OO) de desenvolvimento de sistemas de informação distribuídos que integra conceitos utilizados em modelos conceituais de métodos OO de desenvolvimento de software com conceitos utilizados na especificação de arquiteturas distribuídas. Como resultado disto, promove-se, então, uma melhor utilização das tecnologias atuais de distribuição (por exemplo, bancos de dados distribuídos, internet, intranet, etc.). Durante a fase de análise deste método, objetos são agrupados em subsistemas segundo a afinidade que existe entre eles. Este agrupamento é conduzido de maneira que o sistema de informação distribuído venha apresentar um bom desempenho. Finalmente é proposta uma ferramenta para automatizar o agrupamento dos objetos em subsistemas.

Abstract

With the increasing availability of distributed technologies, large companies have been pursuing more and more the development of distributed information system. However, there is a lack of methods that consider the distribution aspect from its initial phase (requirement analysis) to its final phase (implementation). Indeed distributed architecture specifications (e.g., OMG's CORBA) provide support only to the activities related to the software implementation process (the analysis process is not considered). This work presents an object-oriented (OO) method for developing distributed information systems which integrates concepts used on conceptual models of OO methods with concepts used in distributed architecture specifications. This integration provides a better usage of nowadays distributed technology (e.g., distributed databases, internet, intranet, etc.). During the analysis phase of this method, objects are grouped in subsystems based on the affinity that exists among them. This grouping process is conducted in order to induce better performance for the distributed information system. Finally, the work proposes a tool that automates the object grouping process.

Conteúdo

Capítulo 1 - Introdução.....	1
1.1 Por que Orientado a Objetos	2
1.2 Conceitos Básicos	3
1.3 Organização dos Capítulos.....	4
Capítulo 2 - Métodos Orientados a Objetos.....	5
2.1 Conceitos.....	5
2.2 Técnica de Modelagem de Objetos	6
2.2.1 Modelo de Objetos	7
2.2.2 Modelo Dinâmico	10
2.2.3 Modelo Funcional	12
2.2.4 Projeto de Sistema.....	14
2.2.5 Projeto de Objetos.....	14
2.3 Projeto Orientado à Responsabilidade.....	15
2.3.1 Fase Exploratória.....	16
2.3.1.1 Classes	17
2.3.1.2 Responsabilidades	18
2.3.1.3 Colaborações	19
2.3.2 Fase de Análise	20
2.3.2.1 Análise da Hierarquia	20
2.3.2.2 Análise de Subsistemas.....	21
2.3.3 Protocolos.....	24
2.4 Casos de Uso	25
2.4.1 Análise.....	26
2.4.1.1 Modelo de Requisitos.....	26
2.4.1.1.1 Modelo de Casos de Uso	27
2.4.1.2 Modelo de Análise.....	28
2.4.1.3 Subsistemas	32
2.4.2 Construção	33
2.4.2.1 Modelo de Projeto.....	33
2.5 Conclusão	35
Capítulo 3 - Arquiteturas Distribuídas	38
3.1 DOMS	38
3.1.1 Modelo de Objetos Comum	40
3.2 CORBA	41
3.2.1 A Arquitetura ORB.....	42
3.3 OSCA.....	44
3.3.1 Camada de Dados.....	44
3.3.2 Camada de Processamento.....	45
3.3.3 Camada de Usuário.....	45
3.3.4 Software de Comunicação	45
3.3.5 Blocos de Construção.....	46
3.3.6 Contratos.....	46

3.4 Conclusão	47
Capítulo 4 - Análise de Ferramentas CASE	50
4.1 Engenharia da Informação	50
4.1.1 Planejamento Estratégico da Informação	51
4.2 Ferramentas CASE.....	53
4.2.1 ADW.....	54
4.2.2 IEF	54
4.3 Conclusão	55
Capítulo 5 - O Método Proposto.....	57
5.1 Descrição do Método.....	57
5.1.1 Fase de Análise.....	59
5.1.1.1 Modelo de Casos de Uso Estendido	60
5.1.1.2 Modelo de Objetos.....	64
5.1.1.3 Modelo Dinâmico.....	67
5.1.1.4 Subsistemas.....	69
5.1.2 Fase de Projeto	74
5.1.2.1 Classes Clientes e Classes Servidoras.....	74
5.1.2.2 Blocos de Construção	76
5.1.2.3 Distribuindo os Blocos de Construção	78
5.1.3 Fase de Implementação	79
5.2 Proposta de uma Ferramenta CASE	80
5.2.1 Matriz de Colaboração.....	82
5.3 Conclusão	84
Capítulo 6 - Exemplo de Aplicação do Método	86
6.1 Motivação e Descrição	86
6.2 Identificação dos Requisitos	88
6.2.1 Informações Geográficas	88
6.2.2 Módulos da Aplicação	89
6.3 Aplicação do Método.....	92
6.3.1 Fase de Análise.....	92
6.3.2 Fase de Projeto.....	102
6.4 Conclusão	103
Capítulo 7 - Conclusões	105
7.1 Contribuições.....	105
7.2 Extensões.....	106
Referências Bibliográficas	108

Capítulo 1 - Introdução

Com a viabilização de um número cada vez maior de tecnologias distribuídas (tanto no nível de hardware como no de software), a procura pelo desenvolvimento de sistemas de informação distribuídos, por parte de empresas de grande porte, tem aumentado consideravelmente nos últimos tempos. Com o surgimento de soluções distribuídas, essas empresas vêem a possibilidade de poder integrar todas suas informações (normalmente localizadas em diferentes lugares) de maneira a melhor aproveitá-las. Entretanto, existe uma carência de métodos que tratem o aspecto de distribuição desde sua fase inicial (análise de requisitos) até sua fase final (implementação). O que existem são especificações de arquiteturas distribuídas (por exemplo, CORBA do Object Management Group) que provêem suporte apenas às atividades relacionadas ao processo de implementação do software (o processo de análise não é contemplado).

De um modo geral, os métodos de desenvolvimento de software (estruturados ou orientados a objetos) prescrevem o seguinte processo de construção: primeiro cria-se um modelo conceitual (ou abstrato) da aplicação e em seguida implementam-se as informações contidas nesse modelo em algum ambiente computacional alvo. Esse processo talvez fosse ideal há alguns anos quando existiam apenas máquinas de grande porte onde todas as aplicações eram instaladas. As informações do modelo conceitual eram, então, implementadas, sem levar em conta aspectos de distribuição.

Porém, quando se aplica esse processo na construção de sistemas de informação distribuídos, a passagem da fase de análise para a fase de projeto representa um grande passo a ser dado. Os projetistas devem ser responsáveis por todas as tomadas de decisões de como melhor distribuir e implementar as informações contidas no modelo conceitual. A questão que surge, então, é: quais aspectos de distribuição devem ser considerados no modelo conceitual?

O objetivo principal do modelo conceitual é promover um entendimento do funcionamento da aplicação sem considerar aspectos do ambiente computacional alvo. Todavia, é possível trabalhar os conceitos envolvidos na construção deste modelo de maneira a identificar a melhor forma de agrupar (distribuir) as informações do sistema. Tal agrupamento deve auxiliar o projeto e a implementação do sistema num ambiente distribuído.

O objetivo da dissertação é, então, fazer um estudo dos modelos e conceitos utilizados pelos principais métodos orientados a objetos (OO) de desenvolvimento de software ([RUMB91], [WIRF90a] e [JACO92]) e um estudo de arquiteturas distribuídas (DOMS, CORBA e OSCA), e a partir desses estudos, propor um método. É proposta uma maneira de suavizar esta transição da fase de análise para a fase de projeto no sentido de que o aspecto de distribuição do sistema já seja considerado na fase de análise.

As principais contribuições deste trabalho são:

1. apresentação de um método que integra conceitos utilizados em modelos conceituais de métodos OO de desenvolvimento de software com conceitos utilizados na especificação de arquiteturas distribuídas. Como resultado disto, promove-se, então, uma melhor utilização das tecnologias atuais de distribuição (por exemplo, bancos de dados distribuídos, internet, intranet, etc.).

2. proposta de um método orientado a objetos para desenvolver sistemas de informação distribuídos que venham apresentar um bom desempenho. Durante a fase de análise deste método, objetos são agrupados em subsistemas segundo a afinidade que existe entre os mesmos. O bom desempenho desejado para o sistema é obtido através deste agrupamento.
3. proposta de uma ferramenta para automatizar o agrupamento dos objetos em subsistemas. Existem no mercado dois programas (apresentados no Capítulo 4) que implementam ferramentas que agrupam processos e entidades de dados segundo a afinidade que existe entre cada um desses. A ferramenta proposta nesta dissertação estende as ferramentas implementadas por estes dois programas em dois sentidos. Em primeiro lugar, ela é construída sob o paradigma de orientação a objetos enquanto que as outras duas estão baseadas na decomposição funcional. Em segundo lugar, ela promove uma avaliação dos agrupamentos a fim de garantir o bom desempenho do sistema, o que não ocorre nas ferramentas do Capítulo 4.

1.1 Por que Orientado a Objetos

O método proposto neste trabalho foi elaborado sob o paradigma de orientação a objeto pelos seguintes motivos [LOW96]:

Representação

Objetos podem ser tomados como unidade de distribuição, migração e comunicação de um sistema de informação distribuído. Um objeto pode ser considerado unidade de distribuição devido à sua similaridade com a forma que processos são tratados: processos encapsulam variáveis locais e funcionalidades relacionadas a estas variáveis, e objetos encapsulam estados e funcionalidades. A migração de objetos pode representar o movimento de processos ao redor do sistema. A troca de mensagem entre os objetos pode representar a comunicação inter-processos.

Particionamento

Facilidade de particionar o sistema em componentes (subsistemas) a serem distribuídos numa rede. Este particionamento pode ser feito de maneira fácil conforme apresentado no item anterior.

1.2 Conceitos Básicos

Em engenharia de software, um método compreende um conjunto de regras que inclui os seguintes componentes [RUMB95b]:

- Um conjunto de conceitos utilizados na modelagem do software a ser desenvolvido.
- Um conjunto de notações utilizadas para representar os modelos construídos a partir dos conceitos do item anterior.
- Um processo de desenvolvimento iterativo que descreve, passo a passo, como e quando construir e implementar os modelos do método. Este processo, normalmente, é dividido em três fases: análise, projeto e implementação. Na fase de análise é construído um modelo conceitual do software a ser desenvolvido. Na fase de projeto é estabelecida uma maneira de implementar as informações do modelo conceitual no ambiente computacional alvo escolhido. Na fase de implementação é feita a codificação do software.

Os métodos existentes podem ser classificados basicamente em ([JACO92]):

- *método baseado na decomposição funcional*: neste método, funções e dados são tratados separadamente. O funcionamento do sistema é dividido em funções, e os dados trafegam entre as funções. Essas funções são decompostas gradativamente ao longo das fases do processo de desenvolvimento até chegar às funções atômicas (funções a serem implementadas). Exemplo deste método é a Engenharia da Informação [MART89], descrita brevemente no Capítulo 4.
- *método orientado a objetos*: neste método, funções e dados estão altamente integrados. O processo de desenvolvimento desse método está baseado na identificação dos objetos que irão compor o sistema. Três exemplos desse tipo de método são [JACO92], [WIRF90a] e [RUMB91], cujas descrições são apresentadas no Capítulo 2.

Ferramentas oferecem suporte automático ou semi-automático aos métodos. Diagramadores de fluxogramas, editores de texto, editores gráficos, etc., são exemplos deste tipo de ferramenta. Quando as ferramentas são integradas de maneira que informações criadas por uma ferramenta possam ser utilizadas por outras ferramentas, o que se tem é um sistema de suporte ao desenvolvimento de software, chamado ferramenta CASE (Computer-Aided Software Engineering) [PRES92]. No Capítulo 4 são apresentadas duas ferramentas CASE para o método Engenharia da Informação. Exemplos de ferramentas CASE para métodos orientados a objetos são StP [IDE96], Rational Rose [RATI96] e Objectory [JACO92]. No Capítulo 5 é proposta uma ferramenta CASE que automatiza as atividades da fase de análise do método proposto neste trabalho.

Segundo [JACO92], a arquitetura, no desenvolvimento de software, representa um conjunto de características que descrevem o ambiente computacional alvo sobre o qual será desenvolvido o software. No método apresentado neste trabalho, a arquitetura utilizada descreve um ambiente distribuído.

1.3 Organização dos Capítulos

O método proposto neste trabalho está dividido em três fases: análise, projeto e implementação. Os conceitos e modelos utilizados na elaboração destas fases foram retirados dos conceitos e modelos apresentados nos capítulos 2 e 3.

No Capítulo 2 é feita uma análise de três métodos OO de desenvolvimento de software. São apresentados os processos de desenvolvimento propostos por cada método, dando uma ênfase maior na construção dos modelos conceituais elaborados nas fases iniciais desses métodos. Na conclusão desse capítulo são identificados os conceitos e modelos utilizados na elaboração do método proposto.

No Capítulo 3 é apresentada uma discussão de três arquiteturas distribuídas. A partir dessas arquiteturas é identificado um ambiente distribuído em que o sistema de informação, construído segundo o método proposto, pode ser implementado.

No Capítulo 4 é feito um estudo de duas ferramentas CASE que apresentam um mecanismo de agrupamento de processos e entidades de dados segundo a afinidade que existe entre cada um destes. Tais ferramentas automatizam as fases da Engenharia da Informação.

No Capítulo 5 é feita a apresentação do método proposto. Ainda nesse capítulo é proposta uma ferramenta que automatiza o processo de agrupamento de objetos em subsistemas segundo a afinidade que existe entre eles. Conforme já mencionado, esta ferramenta proposta representa uma evolução no processo de agrupamento implementado pelos programas descritos no Capítulo 4.

No Capítulo 6 é apresentada a descrição de um sistema de informação (geográfica) distribuído sobre o qual é aplicado o método.

Finalmente, no Capítulo 7 são apresentadas as contribuições e extensões deste trabalho.

Capítulo 2 - Métodos Orientados a Objetos

Os conceitos e modelos utilizados na elaboração do método apresentado no Capítulo 5 estão baseados em conceitos e modelos propostos pelos seguintes métodos:

- Técnica de Modelagem de Objetos (James Rumbaugh et al.) [RUMB91]
- Projeto Orientado a Responsabilidades (Rebecca Wirfs-Brock et al.) [WIRF90a]
- Casos de Uso (Ivar Jacobson et al.) [JACO92]

Neste capítulo é feita uma breve apresentação de como cada um desses métodos modela os objetos que deverão compor a aplicação. Sobre esta apresentação, na conclusão do capítulo é feita uma análise sobre os pontos positivos e negativos de cada método.

De um modo geral, os modelos construídos por esses três métodos, durante as fases iniciais, estão baseados nos conceitos apresentados na próxima seção. A ilustração dos conceitos apresentados pelos métodos é feita segundo a descrição da aplicação no Capítulo 6.

2.1 Conceitos¹

Um software orientado a objetos é formado por uma coleção de objetos que incorporam estruturas de dados e operações.

Objetos que possuem o mesmo tipo (estrutura de dados) e o mesmo comportamento são agrupados em classes, chamadas de *classes concretas*. O comportamento de um objeto é definido em termos das operações que ele é capaz de realizar. Essas operações constituem a interface do objeto, através da qual outros objetos interagem com ele. Objetos se comunicam através de troca de mensagens. Uma mensagem é formada pelo nome da operação desejada acompanhado de eventuais argumentos que possam existir. Um objeto é uma instância de uma classe concreta. Classes concretas podem ser consideradas como sendo *templates* a partir dos quais objetos são definidos. Cada objeto criado deve possuir um identificador próprio que permite que ele seja referenciado de maneira única.

Os relacionamentos entre as classes concretas são representados pelas associações. Uma associação entre duas classes implica a existência de regras de integridade sobre os valores dos atributos de suas respectivas instâncias. As regras devem ser aplicadas de forma a garantir a consistência entre esses valores.

Uma forma especial de associação é a *agregação*. Este tipo de associação representa a idéia de "todo-parte" ou "é-parte-de": objetos representando componentes de um conjunto são associados a um objeto que representa o conjunto.

Classes concretas que possuem definições (atributos, operações e associações) em comum podem ser agrupadas em torno de uma *classe abstrata* através do relacionamento de *generalização*. A classe abstrata (superclasse), formada pelas definições em comum, representa a generalização das classes concretas (subclasses). Subclasses representam especializações da respectiva superclasse. Uma subclasse é formada por suas definições próprias

¹Os conceitos apresentados nesta seção foram extraídos de [RUMB91], [JACO92], [WIRF90a] e [WIRF90b].

(especialidades) e pelas definições herdadas de sua superclasse. Classes abstratas também podem ser identificadas a partir de um conjunto de classes concretas e abstratas. A diferença entre uma classe concreta e uma abstrata é que a primeira pode ser instanciada, enquanto que a segunda não pode.

De modo geral, classes abstratas são utilizadas para criação de classes genéricas, as quais podem trazer alguns benefícios, tais como:

- facilitar o entendimento das classes concretas uma vez que possibilita o agrupamento delas em torno de uma classe abstrata mais genérica e portanto, de fácil compreensão;
- facilitar a criação de novas classes concretas (ou mesmo abstratas) por meio de especialização (ou generalização) e
- facilitar a manutenção das subclasses a partir de mudanças na superclasse.

O último benefício listado acima está relacionado com o conceito de herança: mudanças na superclasse são herdadas pelas subclasses. Herança é nome dado ao mecanismo de compartilhamento de atributos e operações utilizando o relacionamento de generalização. Todos os atributos, operações e associações de uma superclasse são herdados por suas subclasses.

É possível que uma subclasse apresente definições provenientes de mais de uma superclasse (*herança múltipla*). Porém, recomenda-se que esse tipo de construção seja evitado. Se os conjuntos de definições das superclasses que devem ser herdadas pela respectiva subclasse forem disjuntos, então, não há problemas. No entanto, se esses conjuntos não forem disjuntos, ficará difícil determinar de quais superclasses as definições devem ser herdadas. Uma herança múltipla sempre pode ser organizada em várias heranças não múltiplas [COAD90].

2.2 Técnica de Modelagem de Objetos

A Técnica de Modelagem de Objetos (Object Modeling Technique - OMT), de James Rumbaugh et al. [RUMB91], prescreve um processo de desenvolvimento de software baseado nas seguintes fases:

- Análise
- Projeto de Sistema
- Projeto de Objetos
- Implementação

O processo tem seu início quando desenvolvedores, gerentes e usuários elaboram um pedido que resulta no enunciado do problema. Esse enunciado é, então, entregue aos analistas, que modelam o problema de maneira mais precisa. Essa modelagem caracteriza as atividades da fase de análise do método (Figura 2.1) e tem como resultado os seguintes modelos:

- *Modelo de Objetos*: descreve as classes componentes da aplicação;

- *Modelo Dinâmico*: mostra a seqüência na qual as operações são executadas em resposta a eventos externos;
- *Modelo Funcional*: descreve as operações das classes de objetos.

Na *fase de projeto de sistema* é identificado o ambiente computacional em que a aplicação deverá ser implementada. Na *fase de projeto de objetos* os três modelos da fase de análise são utilizados na identificação da melhor implementação para os objetos. Decisões tomadas nessa fase devem ser, então, concretizadas na fase de implementação através de uma linguagem de programação específica.

Nas seções que seguem são descritos os três modelos da fase de análise e os principais aspectos inerentes às fases de projeto de sistema e de objetos. Na fase de implementação são apresentadas várias técnicas para se implementar a aplicação desenvolvida segundo o método OMT. A descrição dessas técnicas de implementação foge do escopo desta dissertação.

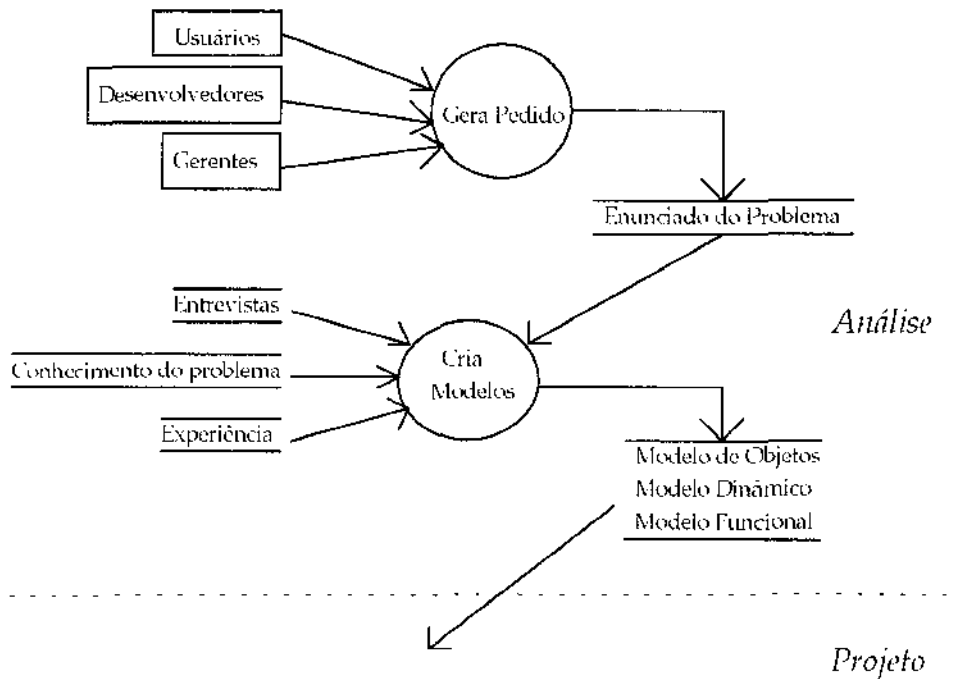


Figura 2.1: Visão geral do processo de análise.

2.2.1 Modelo de Objetos

O objetivo da construção do modelo de objetos é capturar os conceitos do mundo real que são importantes para uma aplicação. Neste modelo são descritos:

- classes e objetos componentes da aplicação; e
- os relacionamentos (associações) que existem entre tais classes.

No modelo de objetos são registradas as informações estáticas da aplicação, a partir das quais são construídos os modelos dinâmico e funcional.

Classes e Objetos

Classes e objetos são representados pelos seguintes diagramas [RUMB95a]:

- *Diagrama de Classes*: descreve as classes de objetos;
- *Diagrama de Objetos*: descreve as instâncias das classes de objetos

No diagrama de classes, uma classe é representada por um retângulo dividido em três partes:

- nome da classe
- lista de atributos
- lista de operações

É possível representar uma classe com apenas o seu nome, omitindo as duas outras partes. No entanto, essa omissão não implica que a classe não possua atributos e operações. Para que isto seja indicado no diagrama é necessário representar a classe com as partes de atributos e operações vazias.

No diagrama de objetos, um objeto é representado por um retângulo dividido em duas partes:

- nome do objeto seguido pelo nome da classe a que pertence
- lista de atributos seguido de seus respectivos valores

Na representação de um objeto não é necessário listar suas operações, uma vez que estas são as mesmas de sua classe. A instanciação de uma classe também pode ser representada por uma flecha pontilhada, apesar do nome da classe no objeto já ser suficiente.

A Figura 2.2 ilustra a notação utilizada pelo método para representar classes e objetos.

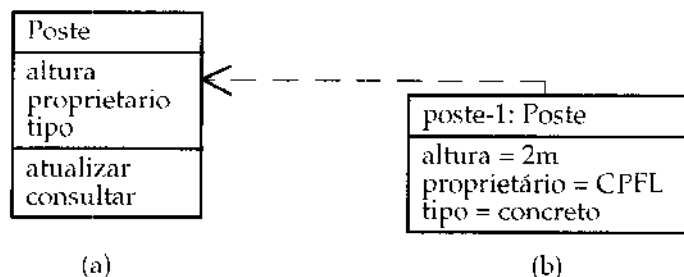


Figura 2.2: (a) Diagrama de classe: classe com atributos e operações; (b) Diagrama de objetos: objetos com os valores dos atributos.

Associações

Associações representam relacionamentos entre classes. As instâncias de uma associação, chamadas *conexões*, representam relacionamentos entre objetos e são formadas por tuplas de referências a objetos.

A grande maioria das associações são binárias, representadas por linhas entre pares de classes (Figura 2.3). A notação utilizada para indicar a multiplicidade (cardinalidade) das classes envolvidas na associação (isto é, quantas instâncias de uma classe podem ser associadas com uma instância da outra classe) é mostrada na Figura 2.4.

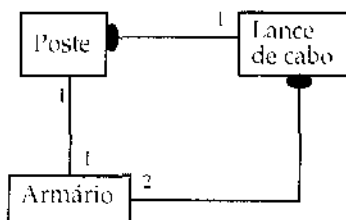


Figura 2.3: Associação

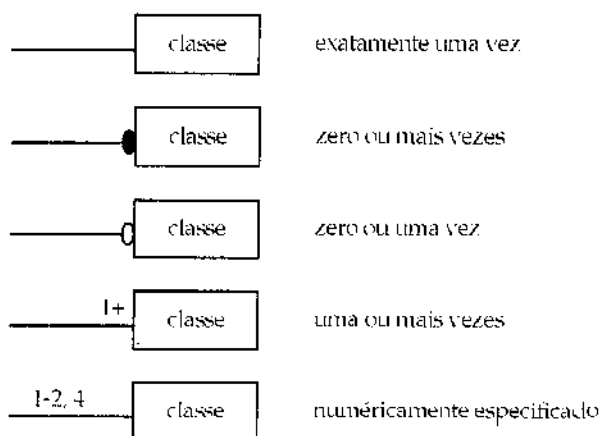


Figura 2.4: Notação de cardinalidade em associações

As classes de objetos desse modelo também podem estar relacionadas segundo as associações de agregação (Figura 2.5.a) e generalização (Figura 2.5.b).

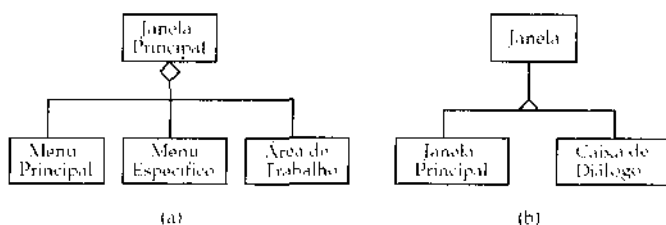


Figura 2.5: (a) Agregação e (b) generalização.

Módulos

Um módulo representa um subconjunto do modelo de objetos. Esse subconjunto é formado por classes (e respectivas associações e generalizações) que representam uma determinada funcionalidade da aplicação (por exemplo, classes de objetos responsáveis pela interface com o usuário). Em [RUMB91] não é apresentada uma maneira formal de se criar módulos. No entanto, é apontado que devem existir poucas conexões inter-módulos e muitas intra-módulo. Isto implica que deve-se criar módulos que sejam fortemente acoplados. Cada classe pertence a um único módulo onde são descritos seus detalhes internos (operações, atributos, etc.), mas pode estar presente em outros módulos, desde que seja referenciada por estes.

2.2.2 Modelo Dinâmico

O modelo dinâmico descreve o comportamento dos objetos identificados no modelo de objetos. Este comportamento é modelado em termos das mudanças de estados que os objetos sofrem como resposta a interações com outros objetos dentro e fora do sistema. Um comportamento trivial é um comportamento cuja modelagem não é relevante para o desenvolvimento da aplicação. É um comportamento que pode ser facilmente deduzido a partir da descrição da classe. Por exemplo, a classe Caixa de Diálogo, utilizada para leitura dos atributos de um elemento de rede e envio de mensagens para o usuário, possui um comportamento trivial.

O modelo dinâmico é definido por *diagramas de estado*. Para cada classe de objetos que possui um comportamento não trivial, é construído um diagrama de estado que modela todos os possíveis *estados* em que uma instância dessa classe pode se encontrar desde o momento de sua criação até sua destruição. O estado de um objeto é definido em termos dos valores que seus atributos e conexões podem assumir. Um objeto muda de um estado para outro através de estímulos de um objeto a outro (por exemplo o objeto Caixa de Diálogo envia os valores dos atributos lidos para o objeto elemento de rede), chamados de *eventos*.

Para auxiliar a construção dos diagramas de estados são construídos *cenários*. Um cenário descreve uma seqüência de interações entre objetos durante uma dada execução do sistema. Essa seqüência pode ser representada de duas maneiras: ou através de uma descrição textual, ou através de um *diagrama de fluxo de eventos*. Para ilustrar estas duas formas de representações, considere um cenário para inserir um poste na rede externa. A Figura 2.6 apresenta a descrição do cenário na forma textual e a Figura 2.7 mostra o cenário descrito segundo um diagrama de fluxo de eventos. No diagrama as linhas verticais representam os objetos (e não as classes) envolvidos no cenário. Cada uma das flechas horizontais representa um evento em cuja origem está o objeto emissor. Um estado, neste diagrama, corresponde ao intervalo de tempo que existe entre dois eventos recebidos. Normalmente, a um estado existe uma *atividade* associada. No diagrama da Figura 2.7, entre os eventos "inserir poste" e "ponto fornecido", é realizada a atividade "obter ponto de inserção".

- O usuário do cadastro seleciona a operação de inserir poste através do menu específico
- O menu específico solicita a operação inserir do objeto poste
- O objeto poste cadastro manda uma mensagem para o usuário, através da caixa de diálogo, para que esse indique a posição de inserção
- O usuário, com a ajuda do mouse, indica a posição de inserção
- O objeto poste cadastro solicita, através da caixa de diálogo, que o usuário forneça os valores dos atributos do poste
- O usuário indica os valores dos atributos
- O objeto poste cadastro envia os valores dos atributos lidos para o objeto de banco de dados para que seja feita a inserção do novo elemento
- O objeto de banco de dados avisa que a inserção foi feita com sucesso ou não
- O objeto poste cadastro transmite o sucesso da operação para o usuário através de uma caixa de diálogo

Figura 2.6: Descrição textual de um cenário para uma inserção de poste na rede externa.

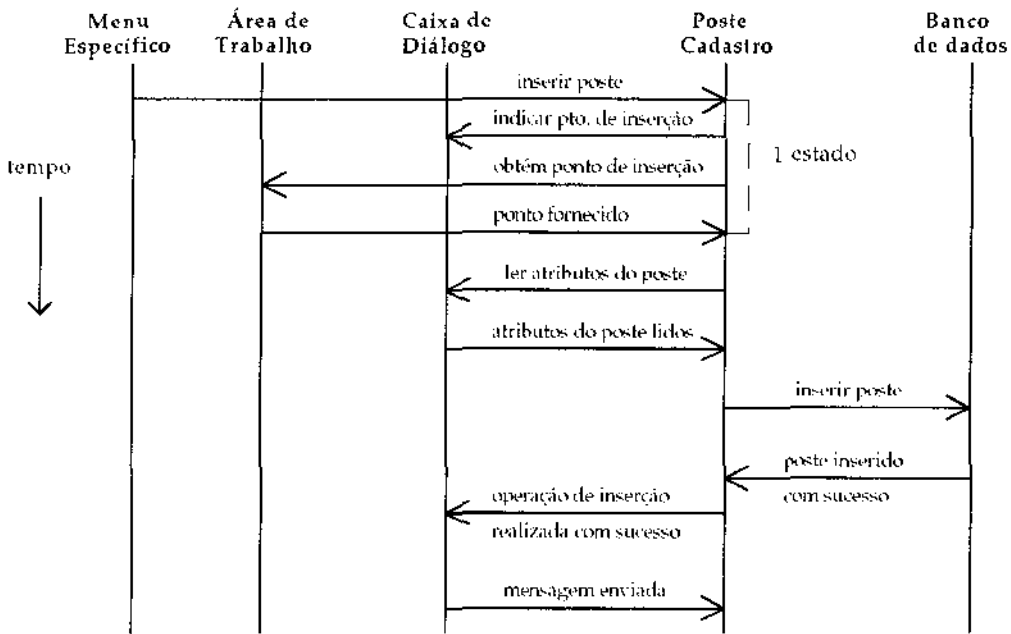


Figura 2.7: Diagrama de fluxo de eventos para uma inserção de poste na rede externa.

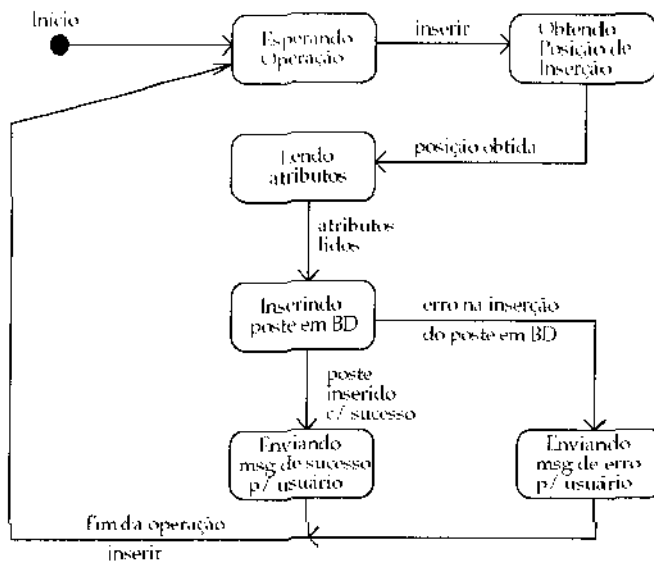


Figura 2.8: Diagrama de Estados da classe *Poste Cadastro*.

A construção dos diagramas de estados que compõem o modelo dinâmico é feita, basicamente, a partir de diagramas de fluxo de eventos. Para cada objeto representado nesses diagramas é construído um diagrama de estados onde são relacionados os estados e eventos associados aos mesmos. Um objeto muda de um estado para outro quando ele recebe um evento gerado por algum objeto: por exemplo, na Figura 2.8, o objeto poste muda do estado "esperando operação" para o estado "obtendo posição de inserção" quando ele recebe o evento "inserir" gerado pelo objeto menu específico (Figura 2.7). À mudança de um estado para outro é dado o nome de *transição*. Uma transição é representada no diagrama de estados por uma flecha que vai do estado origem para o estado destino. De um mesmo estado podem sair várias transições para diferentes estados. Por exemplo, na Figura 2.8, do estado "Inserindo em BD" o objeto pode ir ou para o estado "Enviando msg. de sucesso para usuário" ou para "Enviando msg. de erro para usuário". A mudança de estados é dependente de dois aspectos: o estado atual do objeto e o tipo de evento recebido.

Analisando ainda o diagrama da Figura 2.8, nota-se que existe uma situação de "erro na inserção do poste no BD" que não está representada no diagrama de fluxo de eventos da Figura 2.7. Isto porque cada diagrama de fluxo de eventos reflete apenas a seqüência de eventos listada no cenário correspondente. Poder-se-ia ter um outro cenário, com os mesmos objetos, representando essa situação de erro. No diagrama de estados são representadas todas as possíveis seqüências de eventos (cenários) aos quais os objetos de uma classe podem estar associados.

2.2.3 Modelo Funcional

O Modelo Funcional descreve as operações que devem ser executadas pela aplicação. Este modelo é representado por um ou mais *diagramas de fluxo de dados* (DFD). Cada um desses diagramas contém:

- *processos* que transformam os dados;
- *fluxos de dados* que indicam as movimentações dos dados;
- *atores* que produzem/consomem dados;
- *depósitos de dados* que armazenam os dados;
- *fluxos de controle* que indicam a ativação ou não de certos processos.

Os processos representam as operações (ou parte das operações) da aplicação. Os atores representam as entidades externas (por exemplo, usuários) que interagem com o DFD e os depósitos de dados representam os objetos cujas estruturas de dados estão descritas no modelo de objetos.

O DFD da Figura 2.9 descreve a função de abrir um banco de dados para projeto. O projetista fornece o nome e a senha de acesso ao banco de dados. O banco de dados só será aberto se a senha fornecida pelo projetista estiver correta.



Figura 2.9: DFD.

No DFD, temos:

- ator: *Projetista*.
- depósito de dados: *Banco de Dados*.
- processos: *verifica senha* e *abrir BD*.
- fluxo de dados: *nome do BD*, *identificador de acesso ao BD*, *senha* e *senha codificada*.
- fluxo de controle: *senha OK*.

No exemplo, o papel do fluxo de controle é permitir que o processo de abrir BD seja ativado somente se a senha fornecida pelo projetista estiver correta. Caso contrário nada será feito.

Cada um dos processos pode ser expandido em outros DFDs onde serão representados os detalhes de funcionamento do mesmo. O mesmo pode ser feito para cada um dos processos desses novos DFDs e assim sucessivamente, até que se obtenha as operações chamadas de *atômicas*. Estas operações poderão ser especificadas em linguagem natural, linguagem estruturada, tabela de decisão, etc. Esta especificação compreende:

- *assinatura*: define a interface da operação, isto é, argumentos e valores de retorno
- *transformação*: define o efeito da operação (os valores de saída em função dos valores de entrada e os efeitos colaterais das operações sobre seus objetos)

Essas operações são normalmente listadas no modelo de objetos para efeito de herança.

2.2.4 Projeto de Sistema

O objetivo da fase de projeto de sistema é identificar meios para a implementação da aplicação. Essa identificação deve ser feita com base nas informações contidas nos três modelos construídos durante a fase de análise.

Duas atividades principais compreendem essa fase: particionar a aplicação em subsistemas e alocar recursos de software e hardware a esses subsistemas. Subsistemas compreendem aspectos do sistema que possuem propriedades em comum (por exemplo, mesmas funcionalidades, mesma localização física, ou mesma execução sobre um mesmo tipo de hardware). Um subsistema não é um objeto ou uma operação. Um subsistema é um conjunto de classes, associações, operações e eventos fortemente inter-relacionados. É interessante que existam poucos relacionamentos inter-subsistemas e muitos intra-subsistema (coesão). Subsistemas interagem entre si através de interfaces bem definidas. A idéia é encapsular ao máximo os elementos componentes de um subsistema de maneira que esse possa evoluir (através de manutenções) de maneira independente dos demais subsistemas. Isto permite que o projeto de tais subsistemas seja concorrente.

A alocação de recursos para os subsistemas deve envolver algumas perguntas que os projetistas desses devem responder. Essas perguntas envolvem: qual é o desempenho esperado para o subsistema? Qual deve ser, então, o hardware e o software necessários? Como deve ser a comunicação inter-subsistemas e intra-subsistema? Respostas para essas perguntas devem ser obtidas a partir da especificação de requisitos da aplicação e dos três modelos da fase de análise. Por exemplo, a identificação do desempenho esperado para o subsistema deve ser obtida a partir da especificação de requisitos, do modelo dinâmico (o projetista determina quais operações (dos objetos) podem ser executadas concorrentemente e quais não podem) ou até mesmo a partir de uma entrevista com os futuros usuários da aplicação; o software e hardware a serem empregados no subsistema podem ser inferidos a partir da resposta à pergunta anterior e a comunicação entre os subsistemas é obtida através da análise dos três modelos da fase de análise. Neste último caso, a partir do modelo de objetos, os projetistas identificam quais são as operações que estão sendo solicitadas entre os objetos (dentro e fora de um subsistema). Tendo conhecimento de como essas operações devem ser implementadas (a partir do modelo funcional) e qual é o comportamento dos objetos que possuem tais operações (a partir do modelo dinâmico) o projetista pode determinar como deve ser feita a comunicação inter-subsistemas e intra-subsistema.

Ao final dessa fase, obtém-se uma proposta de ambiente computacional sobre o qual deve ser conduzida a implementação da aplicação.

2.2.5 Projeto de Objetos

Durante a fase de análise foram descritas as classes de objetos que compõem a aplicação através de três modelos. Durante a fase de projeto de sistema foi identificado um ambiente computacional para a implementação da aplicação. Durante a fase de projeto de obje-

to é feito um mapeamento das informações contidas nos modelos da fase de análise para o ambiente proposto na fase de projeto de sistema. Considerando que classes de objetos são formadas por tipos e operações, as principais atividades dessa fase são:

- determinar a representação dos atributos: em termos de tipos primários (inteiros, reais, strings, etc) ou em termos de outras classes; e
- descrever os algoritmos que implementam as operações: novas classes podem ser necessárias (classes específicas do domínio da solução) além daquelas especificadas na análise (classes específicas do domínio do problema).

No entanto, antes de realizar a segunda atividade acima, projetistas devem combinar os três modelos da fase de análise de forma a completar as operações das classes: associar uma operação a cada processo do modelo funcional; converter atividades do modelo dinâmico em operações.

O documento resultante desse projeto é formado pelos três modelos da fase de análise mais detalhados. Esse documento deve servir como guia para a implementação do sistema.

2.3 Projeto Orientado à Responsabilidade

O método Projeto Orientado à Responsabilidade (Responsability Driven Design - RDD) apresentado por Rebecca Wirfs-Brock et al. [WIRF90a], [WIRF90b] está baseado nos conceitos de *responsabilidades* e *colaborações* entre objetos. Cada objeto é responsável por executar suas operações. A fim de garantir a execução de suas operações, um objeto pode solicitar a colaboração de outros objetos.

O processo de desenvolvimento proposto por este método está dividido em duas grandes fases (Figura 2.10):

- Fase Exploratória
- Fase de Análise

Durante a *fase exploratória* são identificados os objetos que irão compor a aplicação, quais serão as responsabilidades que cada objeto deverá ter e quais serão as colaborações que deverão existir entre os objetos para que suas responsabilidades sejam garantidas. Na *fase de análise* são realizadas três atividades principais: análise de hierarquias, identificação de subsistemas e criação de protocolos. A análise de hierarquia envolve uma possível reorganização das estruturas hierárquicas de superclasses/subclasses, construídas durante a fase exploratória. O objetivo é poder promover um maior re-uso das definições das classes. A identificação de subsistemas envolve o agrupamento das classes em subsistemas, para oferecer aos desenvolvedores um melhor entendimento funcional da aplicação. A criação de protocolos identifica como as classes serão implementadas através da definição das assinaturas das operações de cada classe. Também é feita a formalização das especi-

ficações das classes e subsistemas componentes da aplicação a partir das quais a implementação da aplicação é conduzida.

Com relação ao processo de implementação, [WIRF90a] só considera aspectos de implementação das classes de objetos a partir de uma linguagem orientada a objetos.

2.3.1 Fase Exploratória

A Fase Exploratória tem como entrada um documento com a especificação dos requisitos da aplicação a ser projetada. A partir dessa especificação são desenvolvidas as seguintes atividades:

- identificação das classes de objetos necessárias para modelar a aplicação;
- identificação das responsabilidades que o comportamento da aplicação implica e a distribuição dessas responsabilidades entre as classes de objeto e
- identificação das colaborações que devem ocorrer entre as classes de objetos para que suas responsabilidades sejam garantidas.

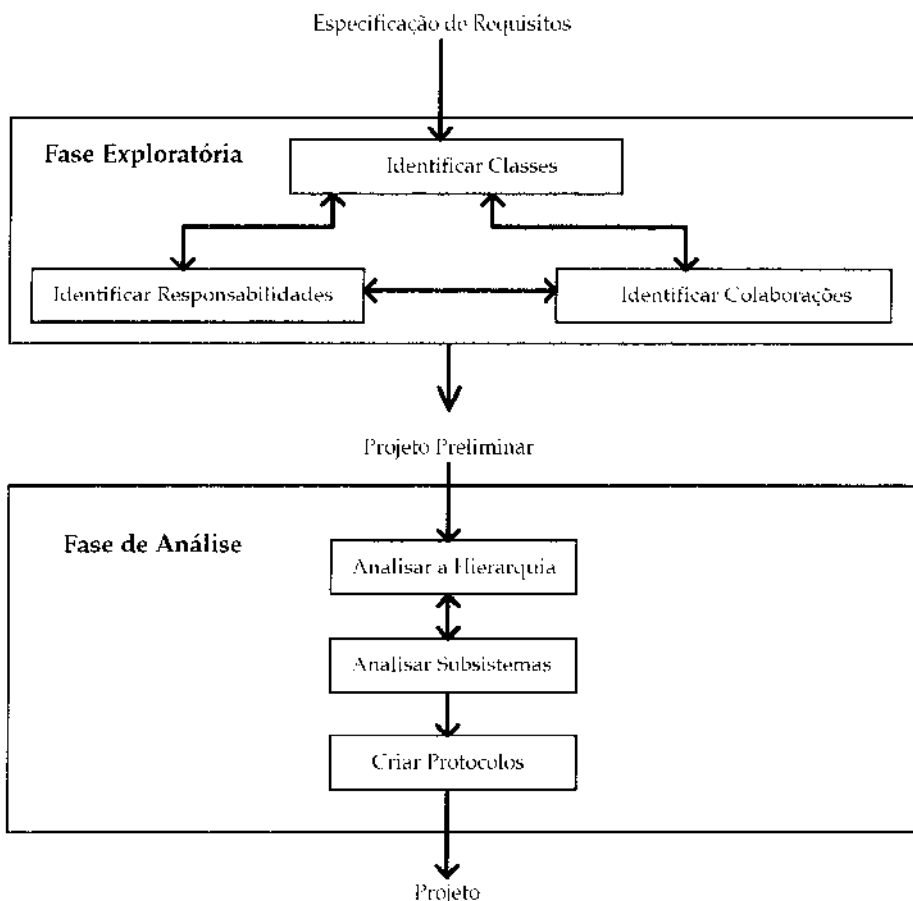


Figura 2.10: Fases do Projeto: Exploratória e Análise

2.3.1.1 Classes

As classes de objetos são identificadas a partir do texto da especificação de requisitos. Para cada classe identificada, chamada, então, de *classe concreta*, é feito um cartão onde são registradas suas informações. Nesse primeiro momento, é registrado apenas o nome da classe. Conforme o processo de desenvolvimento da aplicação vai evoluindo, outras informações vão sendo acrescentadas a esse cartão. Na Figura 2.11 é apresentado o cartão feito para a classe Poste Projeto.

Classe: Poste Projeto	

Figura 2.11: Cartão da classe Poste Projeto.

A partir das classes concretas são identificadas as classes abstratas. Para cada classe abstrata identificada é feito também um cartão com suas informações. No entanto, nesse momento, além do nome da classe, também deve-se colocar em cada cartão quais são as superclasses e as subclasses (se existirem) a que estão relacionadas (Figura 2.12). Esse relacionamento de superclasse/subclasse pode ser melhor visualizado a partir do *grafo de hierarquia* (Figura 2.13). Todos os nós folha do grafo de hierarquia devem ser classes concretas.

Classe: Poste Controle	
Elemento de Controle	
Poste Projeto, Poste Cadastro	

Classe: Poste Projeto	
Poste Controle	

Figura 2.12: Cartão de Classe com superclasse(s) e subclasse(s).

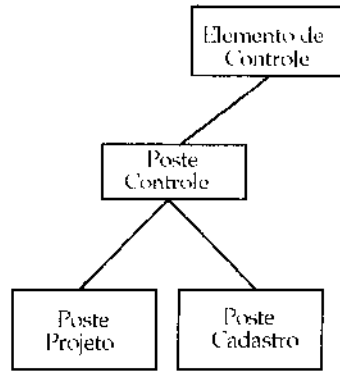


Figura 2.13: Grafo de Hierarquia.

2.3.1.2 Responsabilidades

Responsabilidades de uma classe representam as operações que essa deve ser capaz de realizar. Responsabilidades são identificadas a partir da análise da especificação das classes (abstratas e concretas) e da especificação de requisitos. A distribuição das responsabilidades entre as classes deve ser feita de maneira que os seguintes requisitos sejam contemplados:

Distribuir uniformemente a inteligência da aplicação

A *inteligência* de uma aplicação é representada pelas informações mantidas pelos atributos de seus objetos. O que se propõe é que essa inteligência seja distribuída da forma mais uniforme possível. Classes com muitas responsabilidades são normalmente classes com muita inteligência. Deve-se evitar ter poucas classes com muita inteligência e muitas com pouca. No entanto, uma distribuição perfeitamente uniforme é impossível, pois, dependendo do papel que possuem dentro da aplicação, algumas classes terão que ter necessariamente mais inteligência que outras.

Manter o comportamento com a informação relacionada

Um objeto que possui uma determinada informação deve ter a responsabilidade de mantê-la. Desta forma, responsabilidades que manipulam uma dada informação em particular devem ser atribuídas ao objeto que detém tais informações.

Compartilhar responsabilidades

Responsabilidades que envolvem muitas tarefas que podem ser realizadas por vários objetos devem ser divididas em responsabilidades menores e mais específicas. Essas responsabilidades menores serão atribuídas às classes mais apropriadas.

As responsabilidades atribuídas às classes devem ser registradas em seus respectivos cartões (Figura 2.14).

Classe: Poste Projeto	
Poste Controle	
Inserir Poste	
Remover Poste	
Alterar Poste	
Consultar Poste	
Associar Poste	

Figura 2.14: Responsabilidades da classe Poste Projeto.

2.3.1.3 Colaborações

Para garantir suas responsabilidades os objetos:

- executam todas as computações necessárias, ou
- solicitam colaborações de outros objetos.

As colaborações envolvem sempre um cliente e um servidor. Toda colaboração solicitada pelo cliente está associada a alguma responsabilidade implementada pelo servidor. Uma classe que não possui nenhuma interação com outras classes, isto é, não participa de nenhuma colaboração, deve ser descartada. Colaborações são identificadas a partir da análise das responsabilidades atribuídas a cada uma das classes, que consiste em verificar como cada responsabilidade é resolvida por cada classe. Por exemplo, responsabilidades compartilhadas resultam em colaborações (classes com responsabilidades menores deverão colaborar entre si para garantir a execução da responsabilidade maior). As colaborações identificadas devem ser registradas nos cartões com as informações das classes (Figura 2.15). Nem todas as responsabilidades de uma classe requerem colaborações.

Classe: Poste Projeto	
Poste Controle	
Inserir Poste	Area de Trabalho Projeto; Caixa de Dialogo Projeto; Poste
Remover Poste	Caixa de Diálogo Projeto; Poste
Alterar Poste	Caixa de Diálogo Projeto; Poste
Consultar Poste	Caixa de Diálogo Projeto
Associar Poste	Area de Trabalho Projeto; Caixa de Dialogo Projeto; Poste

Figura 2.15: Colaborações da classe Poste Projeto.

2.3.2 Fase de Análise

A fase exploratória gera um projeto preliminar no qual tem-se uma primeira especificação das classes que formam a aplicação. A partir desse projeto, na fase de análise são desenvolvidas as seguintes atividades:

- fatoração das responsabilidades na hierarquia de classes, a fim de se obter o máximo de re-usabilidade das especificações das classes;
- modelagem das colaborações entre os objetos de forma mais detalhada, a fim de poder encapsular esses objetos em subsistemas;
- construção dos protocolos para cada classe (assinaturas para as operações), completando a especificação das classes e subsistemas e permitindo a especificação dos contratos.

2.3.2.1 Análise da Hierarquia

O objetivo da análise da hierarquia de classes é obter maior re-usabilidade das especificações das classes que formam a aplicação. Isso pode ser alcançado em duas etapas: na primeira etapa deve-se obter o maior número possível de classes abstratas e tentar fazer com que a maioria das classes concretas sejam subclasses de alguma superclasse abstrata (através da análise dos grafos de hierarquia construídos na fase exploratória); na segunda etapa são identificados os *contratos*. Um contrato é um conjunto de responsabilidades que uma classe *servidora* oferece para um conjunto de classes *clientes*. A maneira como deve ocorrer a interação entre clientes e servidores é determinada pelo contrato: clientes só podem fazer requisições das responsabilidades disponíveis nos contratos e servidores devem atender às requisições solicitadas de maneira apropriada (Figura 2.16).

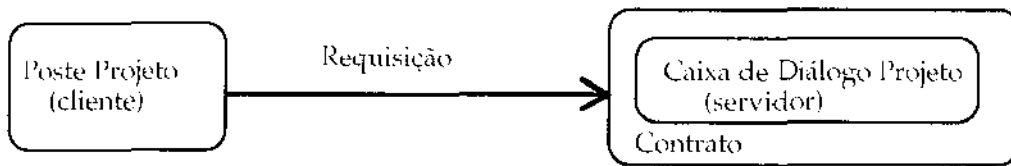


Figura 2.16: Contrato Cliente Servidor.

Normalmente, uma classe suporta apenas um contrato. Entretanto, quando uma classe tem múltiplos papéis dentro do sistema, ou quando suas responsabilidades podem ser fatoradas em conjuntos que são utilizados por clientes distintos, tal classe deve suportar múltiplos contratos.

O agrupamento de responsabilidades em contrato tem por objetivo promover um melhor entendimento dos serviços que são oferecidos pela classe. Na tentativa de criar tais agrupamentos, desenvolvedores avaliam a distribuição de responsabilidades entre as classes realizada na fase anterior.

Uma maneira simples de se definir contratos é começar definindo contratos para as classes que estão no topo da hierarquia. A definição de novos contratos só será necessária se as subclasses adicionarem novas funcionalidades significantes, a serem utilizadas por clientes distintos daqueles que utilizam o contrato herdado. Cada responsabilidade adicionada a uma subclasse representa uma nova funcionalidade. Caso seja simplesmente uma maneira mais específica de expressar uma responsabilidade herdada, a responsabilidade será incorporada no contrato herdado.

2.3.2.2 Análise de Subsistemas

O objetivo da análise de subsistemas é simplificar os padrões de comunicação entre as classes. Tal simplificação é feita em duas etapas: *identificação* dos subsistemas e *simplificação das colaborações* dentro de cada subsistema. A organização das classes em subsistemas deve promover um melhor entendimento da aplicação.

Identificação de subsistemas

Subsistemas representam abstrações sobre as colaborações que existem entre as classes. Um subsistema é formado por um grupo de classes que interagem entre si para garantir alguma funcionalidade da aplicação (por exemplo, as operações de interface da aplicação). Normalmente tais classes possuem muitas colaborações entre si e poucas com as classes fora do subsistema. Um subsistema também pode ser formado por subsistemas.

As colaborações entre as classes são representadas no *grafo de colaboração* (Figura 2.17) a partir do qual são identificados subsistemas. Os contratos das classes são representados por semicírculos enumerados. Na Figura 2.17, a classe Elemento de Controle (superclasse de Poste Controle, Lance de Cabo Controle, Armário Controle, etc.) solicita colaborações da classe Caixa de Diálogo através do contrato 2. As solicitações de colaborações feitas pela classe Elemento de Controle são herdadas pelas suas subclasses.

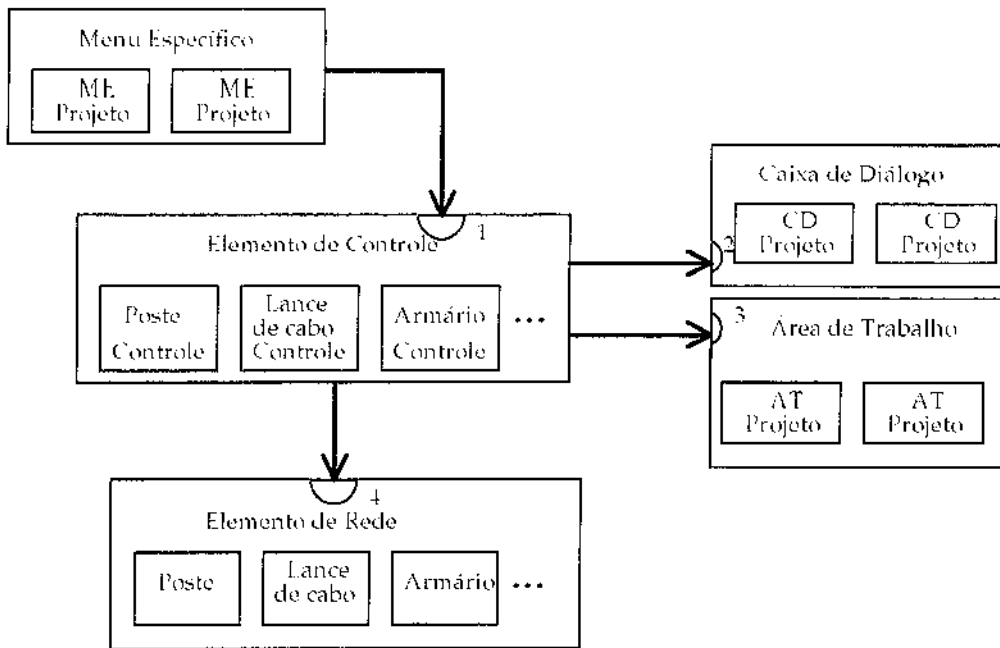


Figura 2.17: Grafo de Colaboração.

Um exemplo de subsistema é apresentado na Figura 2.18. Um subsistema é representado por um retângulo com bordas arredondadas (para diferenciá-lo das classes que são representadas por um retângulo). Para cada subsistema identificado deve ser feito um cartão onde são registradas as seguintes informações: o nome do subsistema e uma lista de contratos oferecidos pelo mesmo.

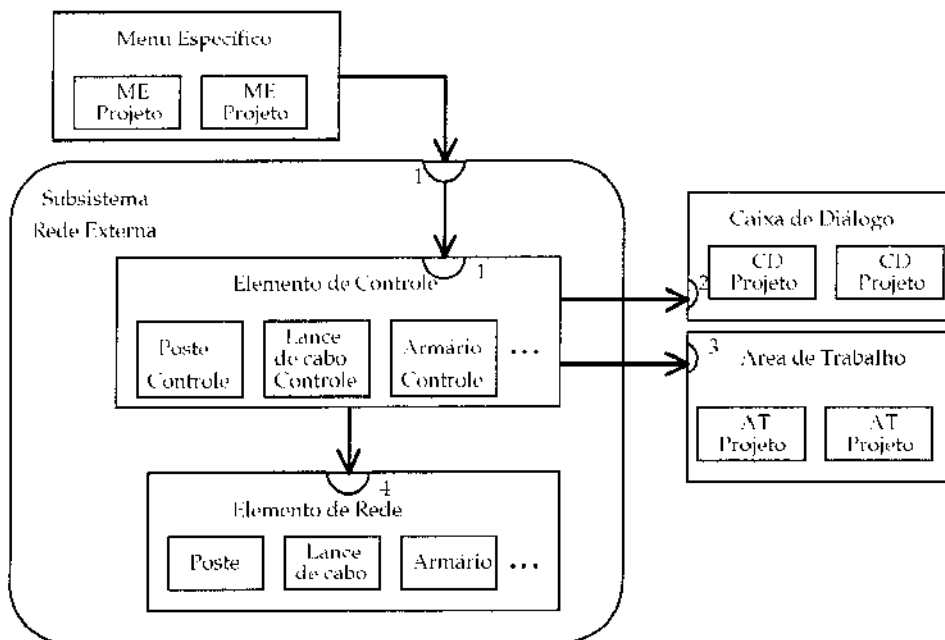


Figura 2.18: Subsistema Rede Externa

A lista de contratos de um subsistema é formada pelos contratos que seus elementos (classes e subsistemas) oferecem para os clientes fora do subsistema. Para cada contrato listado é indicada a classe/subsistema responsável por garanti-lo (Figura 2.19). Quando um subsistema recebe um pedido para um determinado contrato, este delega o cumprimento do mesmo para a classe/subsistema responsável. Caso seja um subsistema o responsável por tratar do pedido, este por sua vez irá repassar o pedido para a sua classe responsável (ou para o subsistema responsável, e assim sucessivamente até chegar à classe responsável de fato). Um subsistema é apenas uma entidade conceitual, não tendo a capacidade de cumprir nenhuma responsabilidade.

Subsistema: Rede Externa	
operações sobre elemento de rede	Elemento de Controle
.....
.....
.....
.....
.....

Figura 2.19: Cartão do subsistema Rede Externa com delegações.

Simplificação das colaborações dentro de cada subsistema

As seguintes medidas devem ser tomadas na simplificação das colaborações dentro de cada subsistema:

Minimizar o número de colaborações entre clientes e servidores

Dentro de um subsistema, tentar fazer com que apenas uma classe seja responsável pelo controle das atividades internas. Desta maneira, clientes deste subsistema têm que solicitar seus pedidos apenas para esta classe principal. Isto acaba reduzindo o número de colaborações que são necessárias entre clientes e servidores (subsistemas), além de promover um melhor encapsulamento das atividades desenvolvidas pelos subsistemas.

Minimizar o número de delegações de responsabilidades feitas pelo subsistema

Evitar que todos os subsistemas e classes de um subsistema sejam responsáveis diretamente pelos contratos que este oferece para seus clientes. Seguindo a idéia de centralização apresentada no item anterior, tentar encontrar uma classe que responda pelos contratos do subsistema e que repasse as respectivas responsabilidades para as

classes e subsistemas internos. O ideal é ter poucas colaborações entre clientes e servidores, e ter muitas dentro dos servidores.

Minimizar o número de diferentes contratos suportados por um subsistema.

Evitar que um subsistema suporte muitos contratos. Isto pode ser um sinal de que a inteligência da aplicação está muito concentrada num mesmo lugar. Se um subsistema possui muitos contratos, deve-se tentar dividi-lo em outros subsistemas, cada qual com um ou dois contratos.

Identificados todos os subsistemas, é recomendado que seja feita uma revisão de todos os cartões das classes. O objetivo desta revisão é verificar se as responsabilidades das classes que necessitavam da colaboração de outras classes não precisam agora de colaborações de subsistemas. Isto deve-se ao fato de que classes que estavam colaborando podem agora estar fazendo parte de um subsistema. Por exemplo, a classe Menu Específico, antes da criação do subsistema Rede Externa solicitava a colaboração da classe Elemento de Controle. Agora deverá solicitar a colaboração para o subsistema Rede Externa.

2.3.3 Protocolos

A criação de protocolos envolve:

- a construção de protocolos para cada classe: especificação das assinaturas de cada método implementado pelas classes;
- a especificação final para cada classe e subsistema e
- a especificação final para cada contrato.

A primeira tarefa envolve a atribuição de nomes significativos para as responsabilidades (operações) de cada classe e a identificação de quais serão seus respectivos parâmetros. O tipo de cada parâmetro deve ser identificado, bem como o tipo de retorno da operação (caso ela retorne algum resultado).

Na segunda tarefa, para cada classe, devem ser especificados:

- o tipo da classe (concreta ou abstrata);
- suas respectivas subclasses e superclasse;
- quais grafos de hierarquia e colaborações em que a classe está presente;
- o propósito da classe;
- o(s) contrato(s) oferecido(s) pela classe. Para cada método do contrato indicar quais suas respectivas colaborações;
- eventuais comentários relacionados à implementação dessa classe que devem ser lembrados.

De maneira semelhante deve ser feita a especificação dos subsistemas, a qual deve apresentar:

- o nome do subsistema;
- a lista dos nomes de todas classes e subsistemas que o compõem;
- a descrição do propósito da classe;
- a listagem dos contratos do subsistema e seus respectivos responsáveis.

A última tarefa na criação de protocolos é fazer a especificação final de cada contrato, que deve conter:

- o número do contrato seguido do nome
- que classe de objetos oferece o contrato
- quais são os clientes daquele contrato
- uma descrição do contrato

2.4 Casos de Uso

O método proposto por Ivar Jacobson et al. [JACO87], [JACO92], [JACO95] está baseado no conceito de *casos de uso*. Um caso de uso descreve uma seqüência de atividades que a aplicação, a ser desenvolvida, deve realizar como resposta a um pedido feito pelo usuário. O objetivo principal é identificar todos os casos de uso da aplicação. Com base nesses casos de uso são conduzidas as atividades das fases desse método. O método é composto por três fases (Figura 2.20):

- Análise
- Construção
- Teste

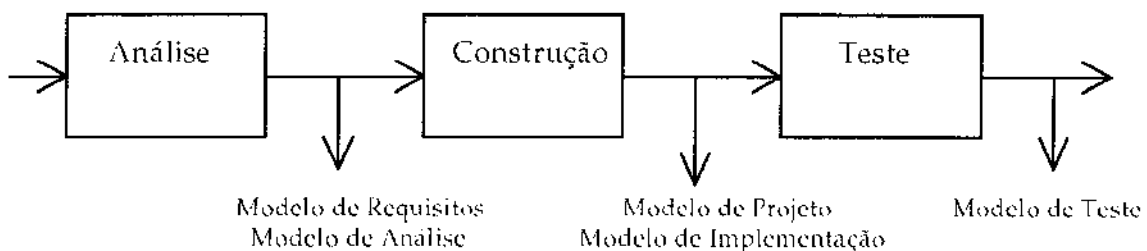


Figura 2.20: Fases do método de Jacobson.

A partir dos casos de uso, identificados no início da fase de análise, são identificados:

- os objetos (fase de análise);
- as interações entre os objetos (fase de projeto); e
- os cenários para testes (fase de teste).

A Figura 2.21 ilustra a abrangência dos casos de uso no método em questão.

Das três fases do método, são apresentadas apenas as duas primeiras. A última fase não é tratada aqui por fugir do propósito desta dissertação.

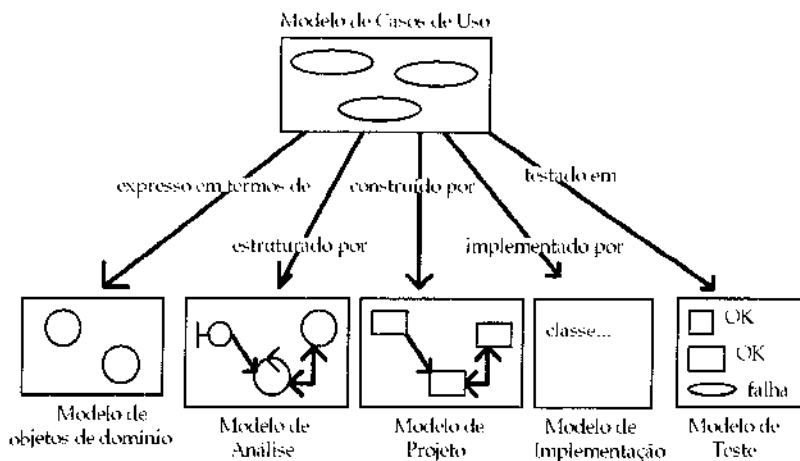


Figura 2.21: O modelo de casos de uso é utilizado na construção de todos os outros modelos do método.

2.4.1 Análise

A fase de análise envolve duas atividades principais que são:

- identificar casos de uso (modelo de requisitos) e
- identificar os objetos a partir dos casos de uso (modelo de análise).

2.4.1.1 Modelo de Requisitos

A construção do modelo de requisitos envolve:

- a identificação de casos de uso (e a construção do respectivo modelo de casos de uso);
- a descrição da interface;
- a identificação de objetos de domínio da aplicação (modelo de objetos de domínio).

Dentre estas três atividades, a mais importante é a primeira, quando comparada com as outras duas. Essa atividade será descrita na próxima seção.

A atividade de descrição da interface deve ser realizada quando se verifica que a aplicação a ser desenvolvida envolve muita interação com usuários. Nesse caso, com base nos casos de uso, desenvolvedores podem criar protótipos que simulam o funcionamento da aplicação. O objetivo é poder fazer a especificação da interface de forma precisa e condizente com as expectativas do usuário.

A atividade de identificação de objetos de domínio deve ser realizada quando a especificação de requisitos apresentada é muito vaga, dificultando o entendimento do domínio

da aplicação. É sugerido, então, que alguns objetos básicos do domínio sejam identificados para auxiliar a elaboração dos respectivos casos de uso.

2.4.1.1.1 MODELO DE CASOS DE USO

A construção do modelo de casos de uso consiste na identificação dos casos de uso da aplicação a partir da especificação de requisitos apresentada pelo usuário. Todo caso de uso é iniciado por um *ator*. Atores representam os papéis que os usuários desempenham nas atividades que solicitam à aplicação. Por exemplo, o caso de uso da Figura 2.22, que descreve a associação de um poste a um lance de cabo no módulo de cadastro, é iniciado pelo ator *Usuário do Cadastro* (UC).

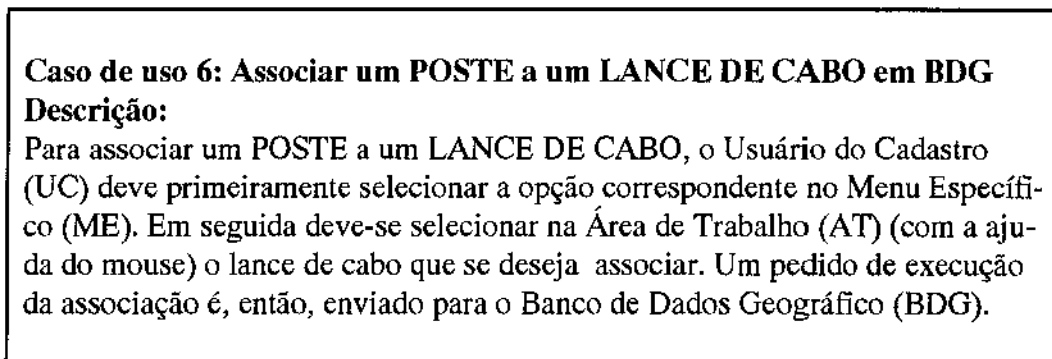


Figura 2.22: Caso de uso

É importante que as descrições dos casos de uso sejam feitas de forma que os futuros usuários possam compreendê-las. Dessa forma, esses usuários também poderão apresentar suas opiniões, contribuindo para que a aplicação seja desenvolvida de forma consistente e coerente com seus propósitos iniciais.

A Figura 2.23 mostra a representação do modelo de caso de uso para o caso de uso da Figura 2.22. A flecha bidirecional ligando o ator ao caso de uso representa o fluxo de controle que existe entre ambos: atores solicitam operações e a aplicação retorna os resultados das operações.

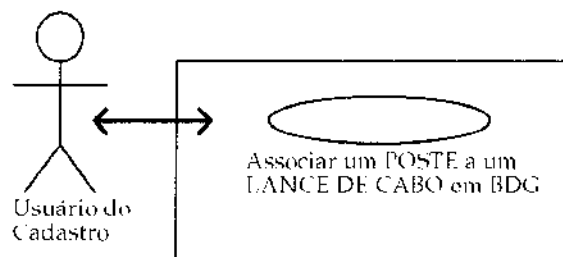


Figura 2.23: Modelo de caso de uso.

Normalmente, na descrição de um caso de uso não são consideradas situações alternativas (por exemplo, situações de erro). No caso de uso da Figura 2.22 não é considerada a possibilidade do poste que foi associado ao lance de cabo estar participando de algum projeto. A representação de execuções alternativas relacionadas a um caso de uso é feita através da associação de *extensão*. Na Figura 2.24 é apresentada a descrição de uma execução alternativa relacionada em o caso de uso da Figura 2.22. Nessa descrição é necessário indicar exatamente em que momento do caso de uso a execução alternativa deverá ocorrer.

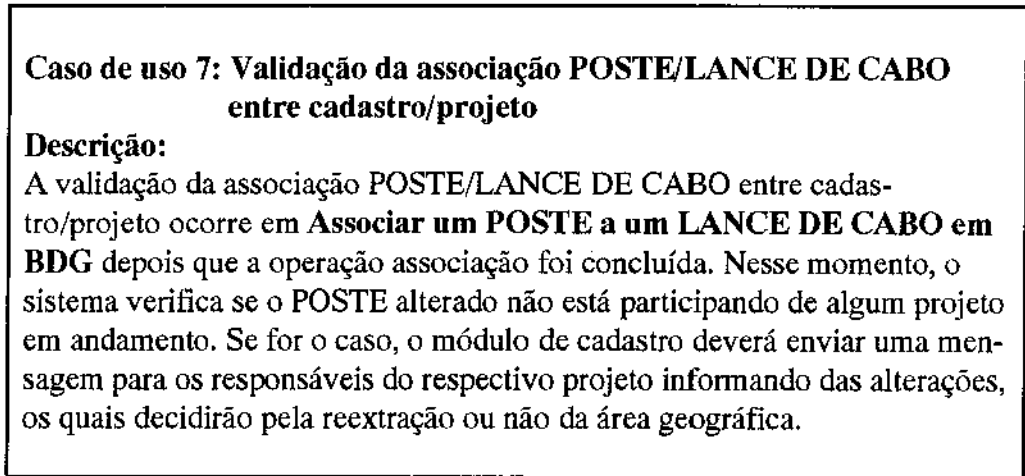


Figura 2.24: Extensão.

Execuções alternativas são representadas no modelo de caso de uso com uma flecha pontilhada. Essa flecha sai da execução alternativa e termina no respectivo caso de uso.

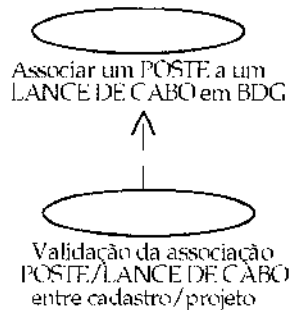


Figura 2.25: Representação da extensão.

2.4.1.2 Modelo de Análise

No modelo de análise são identificados os objetos que compõem a aplicação e as associações que existem entre esses objetos.

Classes e Objetos

Os objetos são identificados a partir dos casos de uso identificados na construção do modelo de requisitos e são classificados em três categorias: *objetos de interface*, *objetos entidade* e *objetos de controle* (Figura 2.26). Os objetos identificados para o caso de uso da Figura 2.22 estão listados na Figura 2.27.

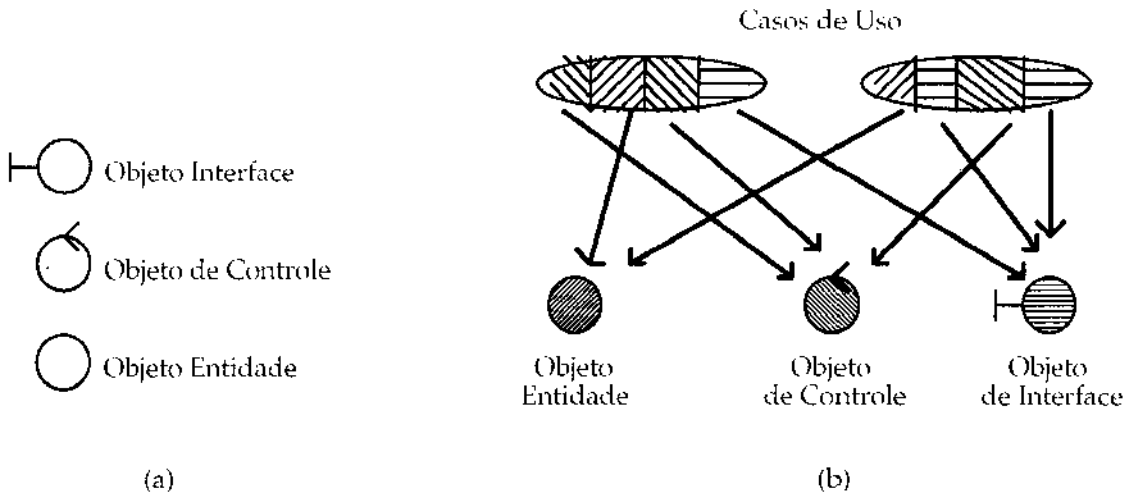


Figura 2.26: (a) Notação utilizada para representar objetos de interface, controle e entidade; (b) Identificação dos objetos a partir dos casos de uso.

# Caso de Uso	Objeto de Interface	Objeto de Controle	Objeto Entidade
6	- Menu Específico - Área de Trabalho	- Poste Controle	- Poste

Figura 2.27: Lista de objetos identificados no caso de uso da Figura 2.22.

Objetos de interface são objetos responsáveis por garantir a comunicação entre os atores e a aplicação. Objetos entidade representam as informações que a aplicação deve manter. Esses objetos são normalmente mantidos por bancos de dados.

Objetos de controle são responsáveis pelas atividades que não são exclusivas de objetos de interface ou de objetos entidade. São funções que normalmente envolvem algum processamento interno à aplicação. Por exemplo, o objeto de controle Poste Controle deve coordenar as atividades que envolvem a associação de um poste com um lance de cabo. Esta tarefa não deve ser realizada nem por objetos de interface e nem por objetos entidades.

Um objeto identificado em um caso de uso pode estar presente em outro caso de uso. No entanto, em cada caso de uso que o objeto participa, este deve ser responsável por apresentar determinado comportamento, o qual pode ser descrito em termos das responsabilidades que o objeto deve garantir para cada caso do qual participa.

Associações

Existem dois tipos de associações: associações entre objetos (representadas por flechas) e associações entre classes de objetos (representadas por flechas pontilhadas).

As associações entre os objetos são as seguintes:

- *associação de conhecimento*: representa o relacionamento que existe entre dois objetos. Este relacionamento é construído a partir da idéia de que um objeto *tem conhecimento* da existência de outro objeto. No exemplo da Figura 2.28, o objeto Poste tem o conhecimento da existência do objeto Armário. Note que a este relacionamento é unidirecional. A cardinalidade é indicada entre colchetes: [1] significa exatamente um. Para o caso em que se tem um intervalo de valores possíveis, a cardinalidade é representada da seguinte forma: [0...N], onde 0 e N indicam os limites do intervalo.

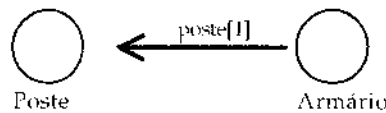


Figura 2.28 Associação de conhecimento.

- *associação de comunicação*: representa a comunicação (troca de mensagem) que existe entre dois objetos, necessária para garantir suas responsabilidades em cada caso de uso. A comunicação entre os objetos é representada por uma flecha que vai do objeto que solicitou a informação (ou a execução de uma operação) para o objeto que detém a informação (ou a operação). Um exemplo desse tipo de associação é mostrado na Figura 2.29, onde o objeto de controle Elemento de Controle solicita que o objeto de interface Caixa de Diálogo faça a leitura dos atributos do elemento de rede selecionado pelo usuário. Nenhum nome ou cardinalidade é associada a esse tipo de associação. Trata-se da associação de um para um entre instâncias de classes, onde informações são trocadas entre as mesmas.

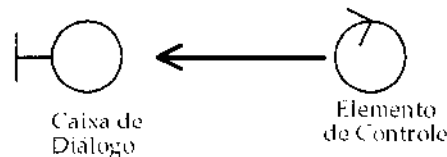


Figura 2.29 Associação de comunicação.

- *associação de agregação*: representa o relacionamento de agregação entre os objetos. Um exemplo desse tipo de associação é mostrado na Figura 2.30, onde o objeto janela é composto por dois menus. A representação da cardinalidade é a mesma utilizada para a associação de conhecimento.

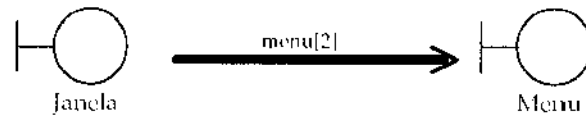


Figura 2.30: Associação de agregação.

Os atributos de um objeto também são representados pela associação de agregação. O tipo de um atributo pode ser considerado como sendo uma classe. A representação dos atributos é feita conforme mostra a Figura 2.31.

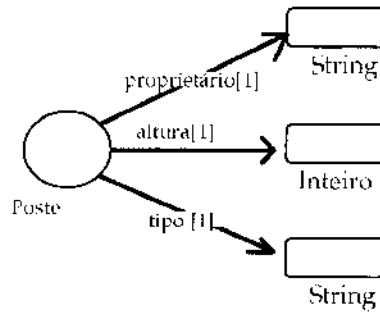


Figura 2.31: Atributos associados ao objeto.

A associação que existe entre as classes de objetos é a *associação de herança* (generalização). Um exemplo desse tipo de associação é mostrado na Figura 2.32 onde a classe de objetos Elemento de Rede representa uma generalização das definições de todos os elementos de rede (poste, armário, lance de cabo, etc.) que compõem a Rede Externa.

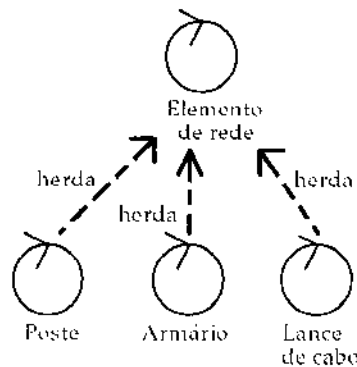


Figura 2.32: Associação de herança entre classes de objetos.

A Figura 2.33 ilustra as associações entre os objetos identificados na Figura 2.27. É representado também o respectivo ator.

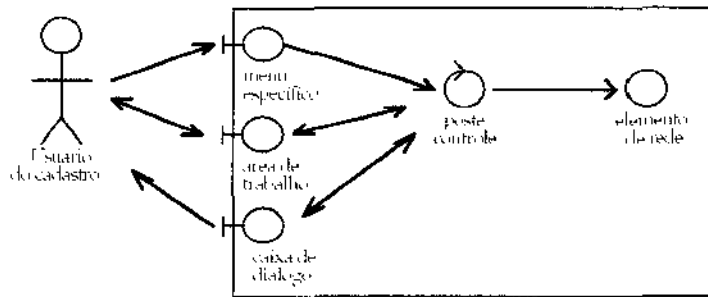


Figura 2.33: Associação entre os objetos.

2.4.1.3 Subsistemas

O agrupamento de objetos em subsistemas é normalmente aplicado no desenvolvimento de aplicações de grande porte onde existe um grande volume de objetos. O objetivo principal desse agrupamento é reduzir a complexidade da aplicação de forma a facilitar seu desenvolvimento e futuras manutenções. É interessante que os objetos componentes de um subsistema possuam uma grande afinidade entre si. Tal afinidade deve estar baseada em dois critérios principais:

- O primeiro critério está relacionado com as futuras atividades de manutenção da aplicação. É interessante que tais atividades, quando ocorrerem, se restrinjam a apenas um subsistema. Como normalmente atividades de manutenção são originadas pelos atores da aplicação, é interessante que um subsistema mantenha comunicação com apenas um único ator.
- O segundo critério propõe que as funcionalidades da aplicação sejam divididas entre subsistemas. Nesse caso, objetos que são fortemente acoplados funcionalmente devem ficar juntos.

Na identificação dos subsistemas é recomendado que existam poucas associações (de comunicação) inter-subsistemas e muitas intra-subsistema. Um subsistema pode ainda ser formado por outros subsistemas segundo o conceito de agregação. Nos subsistemas do nível mais baixo dessa hierarquia de agregação estariam os objetos identificados no modelo de análise.

Uma maneira de identificar subsistemas é atribuir para cada subsistema um objeto de controle. Em seguida adiciona-se todos os objetos de entidade e de interface que são fortemente acoplados a esse objeto de controle. No entanto, nessa distribuição de objetos entre subsistemas, é possível que algum objeto fique de fora uma vez que as funcionalidades que este oferece se encaixam em mais de um subsistema. Sugere-se que esse objeto tenha suas funcionalidades distribuídas entre vários objetos de maneira que cada um desses objetos participe de algum subsistema.

2.4.2 Construção

Durante a fase de construção duas atividades principais são realizadas:

- construção do modelo de projeto
- construção do modelo de implementação

Apenas a primeira será descrita. A segunda atividade representa a implementação da aplicação (código, linguagem de programação) e foge do escopo desta dissertação.

2.4.2.1 Modelo de Projeto

Em [JACO92] é apontado que a principal característica a ser contemplada na construção do modelo de projeto é a *rastreabilidade*². O objetivo é tentar construir um modelo de projeto cujas informações estejam baseadas nas informações contidas no modelo de análise. Dessa forma, é possível identificar como as classes de objetos do modelo de análise foram projetadas no modelo de projeto. Essa ligação entre modelos deverá auxiliar em muito quando da manutenção da aplicação. Apesar de tais atividades normalmente ocorrerem ao nível de implementação, a possibilidade de poder identificar como as mudanças afetam o modelo de análise pode permitir uma manutenção mais precisa e consistente.

A construção do modelo de projeto envolve, basicamente, três etapas:

- *Identificar o ambiente de implementação*: nessa etapa são verificadas todas as seqüências que a escolha do ambiente de implementação irá implicar sobre o projeto da aplicação. Devem ser considerados aspectos do tipo: como serão implementados os processos e como será feita a comunicação entre estes; como será feito o tratamento de erro; se for utilizar algum Sistema de Gerenciamento de Banco de Dados (SBGD), como este será incorporado à aplicação, etc. É sugerido que as atividades dessa etapa sejam conduzidas em paralelo às atividades da fase de análise de forma que quando o projeto tiver seu início, todos esses aspectos de ambiente de implementação já tenham sido levados em consideração.
- *Incorporar os aspectos de implementação identificados na etapa anterior no modelo de projeto*: esta etapa envolve tradução um para um dos objetos do modelo de análise para os objetos do modelo de projeto, chamados de *bloco*. No entanto, em virtude das características do ambiente de implementação, nem sempre essa tradução é possível. Talvez alguns objetos do modelo de análise devam ser mapeados em um ou mais blocos. As atividades desta etapa estão relacionadas diretamente com a questão de rastreabilidade.
- *Identificar a interação entre os objetos*: nessa etapa as interações entre os objetos são modeladas num *diagrama de interação*. Tais diagramas são derivados das descrições dos casos de uso identificados durante a fase de análise: para cada caso de uso é feito

²O termo rastreabilidade é uma tradução do termo em inglês *traceability*.

um diagrama de interação, a partir do qual são identificadas as interfaces dos objetos, e as operações que estes oferecem para os outros objetos.

Soluções para as duas primeiras etapas são muito dependentes do ambiente computacional escolhido para a implementação da aplicação. Em [JACO92] são apresentados vários aspectos que devem ser considerados no momento de executar as atividades dessas etapas. No entanto, as atividades envolvidas na terceira etapa estão relacionadas com o entendimento da interação dos objetos que compõem a aplicação, o que se aproxima mais do propósito da dissertação.

Diagramas de interação são, então, construídos a partir da descrição de casos de uso. A Figura 2.34 mostra o diagrama de interação para o caso de uso da Figura 2.22.

As barras verticais representam os objetos que participam do caso de uso. A barra achurada no canto esquerdo do diagrama representa o ambiente externo à aplicação (usuário ou um outro sistema). As flechas horizontais que saem dessa barra representam os pedidos feitos pelo ambiente externo. As demais flechas indicam as trocas de mensagens entre os objetos e correspondem, de certa forma, às associações de comunicação identificadas no modelo de análise. Cada uma das flechas recebe um nome que especifica o tipo de operação que está sendo solicitado.

O tempo de execução das operações é representado por uma barra mais cheia sobreposta à barra do objeto. As flechas pontilhadas indicam respostas condicionais relacionadas com a execução da operação. No exemplo, a operação de associação pode ou não ser completada.

Apesar não estar representado na Figura 2.34, pode-se reproduzir a descrição do caso de uso no lado esquerdo da barra achurada de maneira a documentar a sequência das interações que ocorrem entre os objetos.

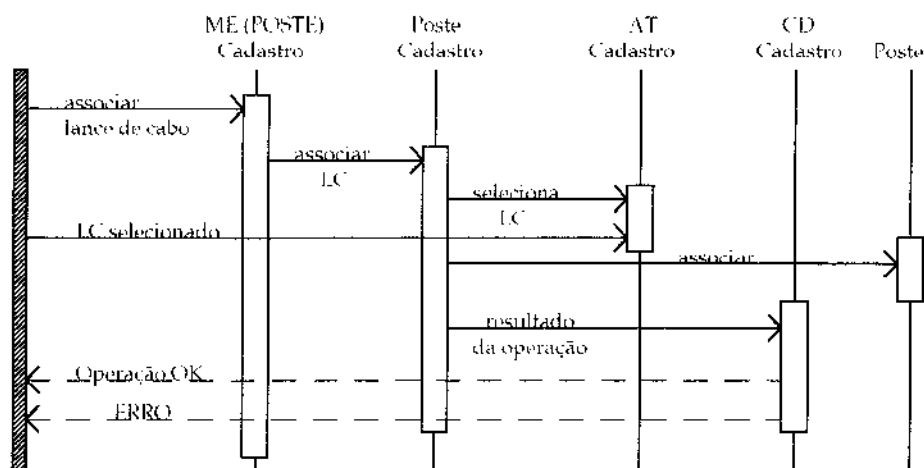


Figura 2.34: Diagrama de interação do caso de uso da Figura 2.22.

Feitos todos os diagramas de interação, é possível, então, determinar a interface de cada um dos objetos. Cada interface é formada por todas as operações que um objeto oferece em cada um dos diagramas de interação de que participa.

Uma vez definida as interfaces dos objetos, o próximo passo é definir o comportamento que cada um deve ter. Tal comportamento é modelado segundo o diagrama de estados de Mealy (Mealy machines). Não será feita uma descrição da construção deste diagrama.

2.5 Conclusão

Neste capítulo foram apresentados três métodos OO de desenvolvimento de aplicações. O primeiro, Técnica de Modelagem de Objetos, está baseado na construção de três modelos: modelo de objetos, modelo dinâmico e modelo funcional. Objetos modelados no primeiro modelo têm seus comportamentos descritos no segundo modelo e suas operações detalhadas na terceiro modelo. O segundo método, Projeto Orientado à Responsabilidade, está baseado na identificação das responsabilidades e colaborações entre os objetos componentes da aplicação. E o terceiro, Casos de Uso, está baseado na identificação de casos de uso da aplicação.

Os pontos positivos e negativos destes três métodos estão sumarizados na Tabela 2.1. O método Casos de Uso apresenta o conceito de casos de uso que auxilia em muito a análise dos requisitos apresentados pelo usuário. Uma preocupação que existe neste método é a questão da rastreabilidade: construir modelos de maneira que elementos pertencentes a um modelo possam ser facilmente identificados nos demais modelos. Esta característica torna o processo de desenvolvimento mais consistente e preciso, uma vez que as informações modeladas se encontram mais integradas. Tal aspecto não acontece no método Técnica de Modelagem de Objetos onde não existe uma integração clara e objetiva entre os modelos de Objetos, Dinâmico e Funcional. Porém, a notação utilizada neste último método, quando comparada com os outros dois métodos, se apresenta mais precisa (completa).

O método Casos de Uso propõe que os objetos identificados a partir dos casos de uso sejam classificados em interface, controle e entidade. A partir desta classificação analisistas podem identificar, com maior precisão, qual é a finalidade de cada objeto.

O método Projeto Orientado à Responsabilidade, apesar de não tratar nem do comportamento dos objetos e nem das associações/agregações que existem entre os mesmos, apresenta dois pontos positivos:

1. *conceitos de responsabilidade e colaboração*: as responsabilidades (atividades) que a aplicação deve garantir são distribuídas entre os objetos que compõem a mesma. A fim de garantir a execução de suas responsabilidades um objeto pode pedir a colaboração de outros objetos. Estes dois conceitos representam uma forma diferente de identificar as operações (responsabilidades) dos objetos e as interações (colaborações) que devem existir entre os mesmos.
2. *construção de subsistemas*: no método de Casos de Uso, o agrupamento de objetos em subsistemas está mais relacionado com futuras atividades de manutenção da aplicação (se alterações sobre objeto X afeta o objeto Y, então X e Y devem ficar num mesmo subsistema). No método Técnica de Modelagem de Objetos, subsistemas compreendem aspectos da aplicação que possuem propriedades em comum (por exemplo, mesmas

funcionalidades, mesma localização física, ou mesma execução sobre um mesmo tipo de hardware). Já no método Projeto Orientado à Responsabilidade, a construção de subsistemas representa uma maneira de simplificar os padrões de comunicação que existem entre os objetos. Esta simplificação promove uma otimização sobre as interações (colaborações) que devem existir entre os objetos. A maneira de construir subsistemas apresentada por este método é mais precisa e consistente quando comparada com os demais métodos, onde subsistemas são construídos de forma subjetiva.

O relacionamento que existe entre os conceitos apresentados por cada um dos três métodos é apresentado na Tabela 2.2.

Tabela 2.1: Pontos positivos e negativos dos métodos apresentados neste capítulo.

Método	Pontos Positivos	Pontos Negativos
Casos de Uso	- conceito de casos de uso - rastreabilidade - classificação dos objetos em interface, controle e entidades	- notação imprecisa
Técnica de Modelagem de Objetos	- notação precisa	- fraca integração entre os modelos
Projeto Orientado à Responsabilidade	- conceitos de responsabilidade e colaboração - construção de subsistemas	- notação fraca - falta de modelagem do comportamento dos objetos - falta de tratamento de associações/agregações entre os objetos

Tabela 2.2: Comparação entre os conceitos dos métodos apresentados neste capítulo.

Técnica de Modelagem de Objetos	Projeto Orientado à Responsabilidade	Casos de Uso
classe/objeto	classe/objeto	classe/objeto (interface, controle, entidade)
associação	-	associação de conhecimento
agregação	-	associação de agregação
generalização	generalização (grafo de hierarquia)	associação de herança
DFD (fluxo de dados)	colaborações	associação de comunicação (interação)
cenário	-	caso de uso
estados/eventos (Harel)	-	estados/eventos (Mealy)
-	contratos (responsabilidades)	-
subsistema	subsistema	subsistema

Os modelos e conceitos utilizados no método do Capítulo 5 estão relacionados com os pontos positivos de cada método apresentados na Tabela 2.1 (Tabela 2.3). Do método de Casos de Uso é utilizado o Modelo de Casos de Uso e o Diagrama de Interação (através do qual são modeladas as colaborações entre os objetos). A necessidade da classificação dos objetos em interface, controle e entidade é justificada na conclusão do próximo capítulo (Seção 3.4).

Do método Técnica de Modelagem de Objetos é utilizado o Diagrama de Estado e a notação empregada na representação do modelo de objetos. O modelo de funcional deste método não foi utilizado uma vez que sua abordagem está mais voltada para a decomposição funcional (Seção 1.2) do que para a filosofia de orientação a objetos. Neste modelo, funções (processos) são decompostas até atingirem um ponto em que podem ser implementadas (chamadas de funções atômicas) e atribuídas às definições das classes de objetos. Segundo o paradigma de orientação a objetos, as funções pertencentes aos objetos são identificadas a partir das responsabilidades (Seção 2.3) que estes possuem, e não derivadas da decomposição funcional das funções que a aplicação deve oferecer ao usuário.

Do método Projeto Orientado a Responsabilidade, além dos conceitos de responsabilidade e colaboração, são utilizadas, também, as medidas empregadas na simplificação das colaborações dentro dos subsistemas. Tais medidas devem contribuir na construção dos blocos de construção componentes do sistema de informação distribuído.

Tabela 2.3: Modelos e conceitos utilizados no método do Capítulo 5.

Método	Modelo/Diagrama	Conceito
Casos de Uso	- Modelo de Casos de Uso - Diagrama de Interação	- classificação de objetos em interface, controle e entidade. - rastreabilidade
Técnica de Modelagem de Objetos	- Diagrama de Estado	- (notação)
Projeto Orientado a Responsabilidade	-	- responsabilidade/ colaboração - construção de subsistemas

Capítulo 3 - Arquiteturas Distribuídas

Hoje em dia existem arquiteturas distribuídas sobre as quais sistemas de informação distribuídos podem ser implementados. Algumas dessas arquiteturas são:

- Distributed Object Management System (DOMS) dos Laboratórios GTE [MANO92]
- Common Object Request Broker Architecture (CORBA) do Object Management Group (OMG) [OMG92a]
- Operations Systems Computing Architecture (OSCA) do Bellcore [BELL92], [MILL92a]

Neste capítulo é feita uma breve apresentação da especificação e dos conceitos utilizados em cada uma destas arquiteturas. Sobre esta apresentação, na conclusão do capítulo é feita uma análise das principais características da três arquiteturas.

Os conceitos apresentados pela especificação dessas arquiteturas em conjunto com os conceitos e modelos apresentados no Capítulo 2 formam a base do ciclo de vida de desenvolvimento de sistemas de informação distribuídos descrito no Capítulo 5.

3.1 DOMS

A arquitetura DOMS consiste num ambiente em que recursos computacionais *heterogêneos, autônomos e distribuídos* (HAD) podem ser integrados num único sistema distribuído global. Os requisitos necessários para este ambiente podem ser alcançados em duas fases:

- A primeira é possibilitar a interconectividade que consiste, basicamente, em possibilitar que dois ou mais recursos consigam trocar mensagens. No entanto, para que isto aconteça, é necessário que exista uma considerável generalização das funcionalidades das interfaces entre os sistemas de informação.
- A segunda fase consiste em possibilitar a *interoperabilidade*³ entre os recursos.

Para atingir estes dois requisitos foi proposto um Gerenciador de Objetos Distribuído (DOM - Distributed Object Management) onde os objetos são os recursos HAD e o DOM é a camada que possibilita a interação entre os objetos. Um Sistema de Gerenciamento de Objetos Distribuído (DOMS) é formado pelos recursos HAD e por um ou mais DOMs.

A Figura 3.1 mostra um esquema da arquitetura DOMS.

³Dois ou mais recursos são interoperáveis se conjuntamente interagem para executar tarefas.

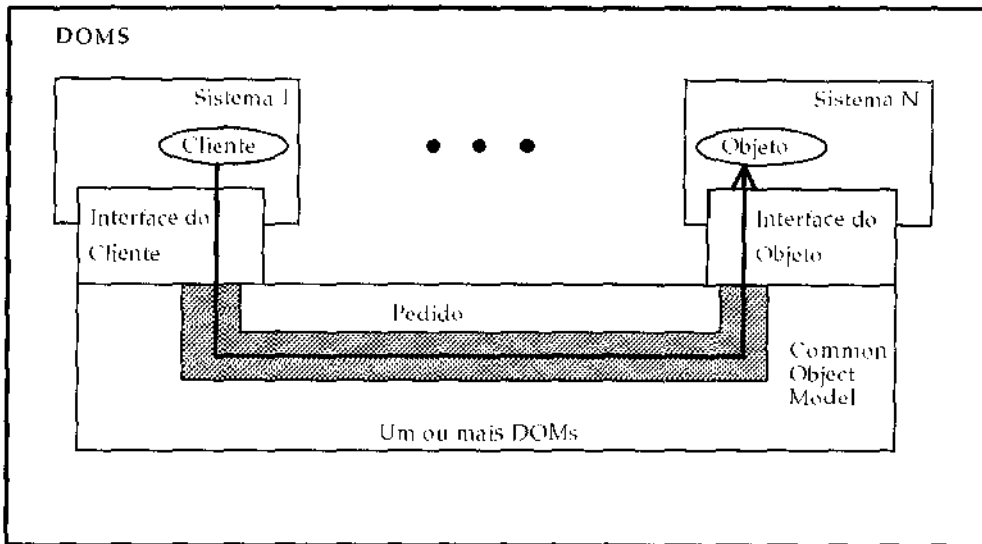


Figura 3.1: Esquema de Arquitetura DOMS

Nesta arquitetura existem três elementos básicos: os clientes, os objetos (recursos HAD) e um ou mais DOMs. A função dos DOMs é atuar como intermediadores entre clientes e objetos, permitindo que clientes solicitem operações sobre objetos instalados em qualquer ponto da rede, de maneira transparente. A solicitação desses pedidos é feita por meio de uma interface - *Interface do Cliente* que pode ser especificada ou como uma biblioteca de chamadas a subrotinas ou através de uma linguagem de definição de interface (Interface Definition Language - IDL). Neste último caso existe um processador que, a partir das definições das interfaces escritas em IDL, gera *stubs*. Esses *stubs*, por sua vez, fazem as chamadas apropriadas.

As implementações de objetos devem atender às interfaces dos objetos suportados pelos DOMs. Essas implantações podem ser módulos arbitrários desenvolvidos a partir das especificações dessas interfaces, ou podem ser softwares já existentes. Neste último caso deve existir um adaptador que serve de intermediador entre o software e a especificação da interface, mascarando, assim, as possíveis incompatibilidades que possam existir entre esses dois elementos.

A princípio, os componentes que serão integrados nesta arquitetura são, em grande parte, componentes que não foram projetados para funcionar num ambiente orientado a objetos. Como consequência, a integração deve ser algo difícil e ter baixo desempenho. No entanto, com o aumento do uso da tecnologia de orientação a objetos no desenvolvimento dos mais diversos recursos (por exemplo linguagens de programação, sistemas operacionais, sistemas de banco de dados, etc.), os limites entre os recursos desaparecem progressivamente. A integração desses poderá ser mais forte e mais fácil através do desenvolvimento de padrões de modelos de objetos, de implementações baseadas nestes modelos e bibliotecas de objetos (re-usabilidade).

3.1.1 Modelo de Objetos Comum

A fim de lidar com a heterogeneidade dos objetos, tanto a interface do cliente quanto a do objeto devem suportar mapeamentos dos pedidos de operações (argumentos e resultados) para as diferentes representações. Apesar disto poder ser feito de maneira casada entre clientes e implementações de objetos (Figura 3.2), o uso de um modelo de objetos comum reduz a quantidade de mapeamentos necessários (Figura 3.3).

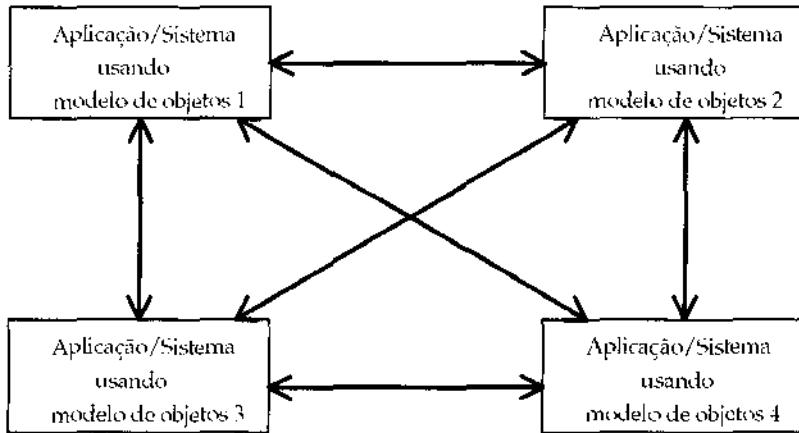


Figura 3.2: Mapeamento dois a dois entre os modelos

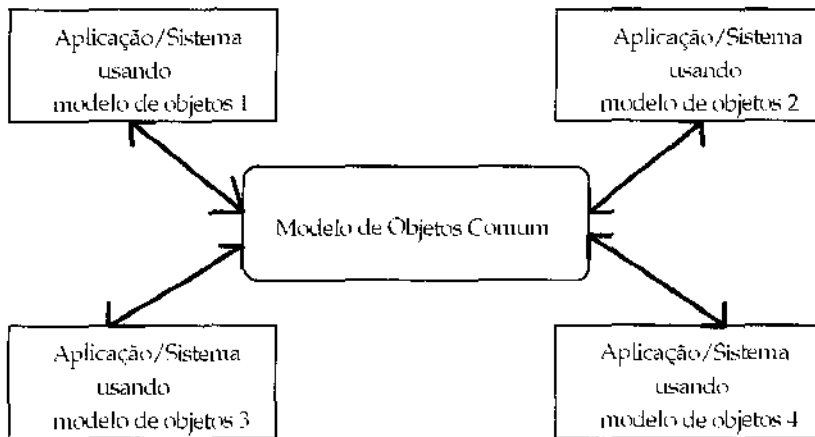


Figura 3.3: Um modelo de Objetos em Comum.

O modelo de objetos comum proposto para o DOMS, chamado de FROOM (Functional/Relational Object-Oriented Model) [MANO90], está baseado em três conceitos básicos: objeto, função e tipo. Os *objetos* representam abstrações das aplicações (por exemplo entidades do mundo real, tais como pessoas, carros), valores (por exemplo inteiros ou strings) e componentes do próprio modelo (por exemplo funções e tipos).

O comportamento dos objetos é modelado em termos das *funções* que podem ser aplicadas a esses. Objetos com mesmo comportamento são agrupados em *tipos*.

3.2 CORBA

Fundado em 1989, o Object Management Group (OMG) é uma organização formada por vendedores e usuários de produtos de hardware e software que se reuniram com o objetivo de adotar um padrão para possibilitar a interoperação entre softwares - especificamente, os orientados a objetos - de diferentes tecnologias e de diferentes fornecedores.

A arquitetura ORB (Common Object Request Broker Architecture - CORBA) proposta por este grupo é uma especificação de arquitetura e interface que permite aplicações fazerem pedidos de operações sobre objetos de maneira transparente (independentemente de implementação e localização). A implementação desses objetos (dados e métodos) é responsabilidade dos vendedores e usuários finais. O OMG se preocupa apenas com a especificação da interface desses componentes. Desta forma, é garantido que os produtos que resolverem adotar o padrão estabelecido pelo OMG poderão intercomunicar-se, uma vez que a interface será a mesma.

Este padrão está baseado em um modelo (ou arquitetura) de referência geral chamado de Arquitetura de Gerenciamento de Objetos (Object Management Architecture - OMA) [OMG92b] (Figura 3.4).

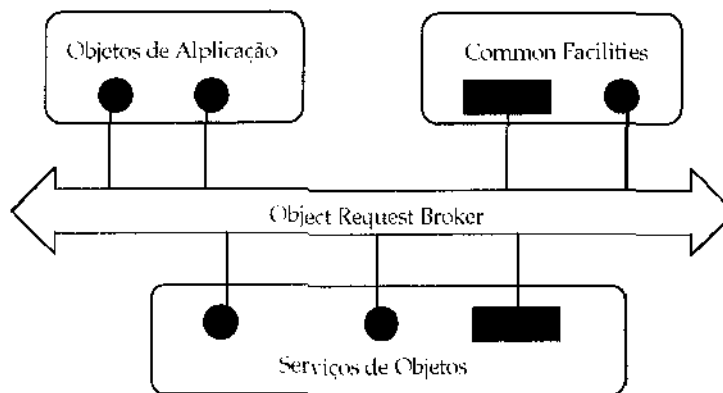


Figura 3.4: *Object Management Architecture.*

Os componentes da OMA são os seguintes:

- *Serviços de Objetos*: coleções de serviços com interfaces que fornecem funções básicas para realizar e manter objetos de qualquer categoria.;
- *Common Facilities*: coleções de classes e objetos que fornecem funcionalidades de propósitos gerais úteis em muitas aplicações;
- *Objetos de Aplicação*: objetos específicos das aplicações particulares de usuários finais;
- *Object Request Broker*: infraestrutura que permite que objetos façam pedidos e recebam respostas de operações sobre outros objetos num ambiente heterogêneo e distribuído.

O modelo de referência OMA representa uma visão completa que o OMG tem do ambiente distribuído enquanto a CORBA trata apenas da interação entre os objetos, provido mecanismos para que esta ocorra. A CORBA representa o componente central da pa-

dronização proposta pelo OMG no sentido de que a OMA é construída a partir dos serviços oferecidos pelo ORB.

3.2.1 A Arquitetura ORB

O ORB se propõe a possibilitar a interoperabilidade entre aplicações em ambientes heterogêneos. De maneira geral podemos esquematizar este problema conforme a Figura abaixo:

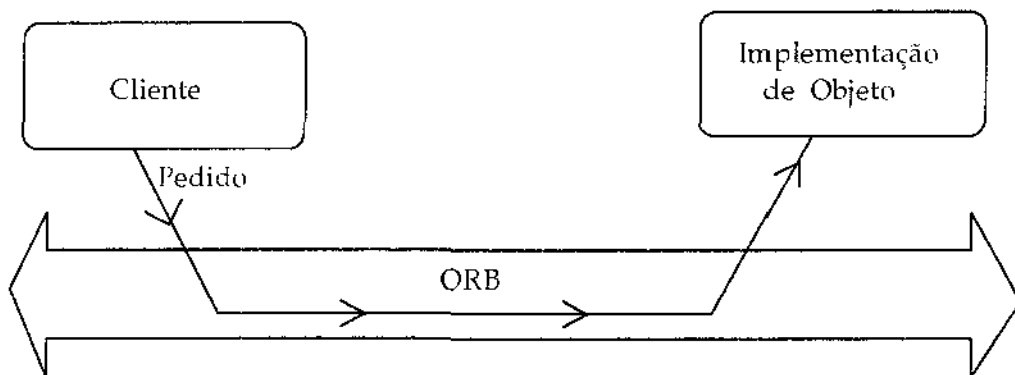


Figura 3.5: Um pedido sendo enviado através do ORB.

Um *Cliente* faz um pedido de operação sobre um objeto cuja implementação está no componente *Implementação de Objeto*. No entanto, o cliente tem apenas conhecimento da operação que o objeto disponibiliza através de sua interface. A maneira como esta operação está implementada e onde está localizada é transparente para o cliente. A responsabilidade de fazer com que esta operação seja realizada (com sucesso ou não) é do ORB, que oferece o mecanismo para localizar a implementação de objeto desejada, preparar esta implementação para receber o pedido de operação e passar os respectivos parâmetros (se existir algum).

As interfaces dos objetos são definidas por uma IDL especificada pelo OMG. Através destas definições é possível mapear os objetos CORBA para a linguagem de programação dos clientes de maneira que estes possam fazer os pedidos de operações. O mapeamento de uma determinada linguagem deve ser o mesmo para qualquer implementação ORB. Isto garante a portabilidade dos clientes para qualquer implementação ORB. Atualmente existem mapeamentos para as linguagens C, C++ e Smalltalk, estando em andamento mapeamentos para Java [OMG96].

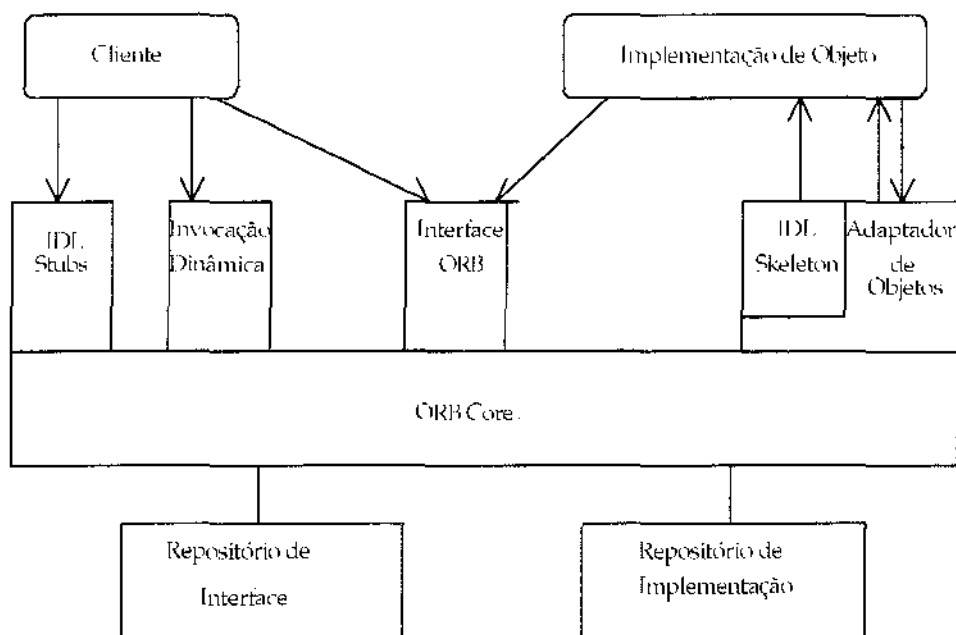


Figura 3.6: Estrutura de Interfaces do ORB.

Os pedidos dos clientes podem ser feitos através de (Figura 3.6):

- *IDL Stubs*: stubs são funções resultantes do mapeamento das interfaces (definidas em IDL) dos objetos para uma determinada linguagem de programação. Desta forma, para que o cliente solicite um pedido, basta saber qual é o stub correspondente à operação desejada;
- *Invocação Dinâmica*: a solicitação da operação é especificada em tempo de execução. Informações sobre a operação e seus respectivos parâmetros são obtidas através do Repositório de Interface.

Os pedidos de operações são recebidos pelas implementações de objetos através do IDL Skeleton. Os Skeletons são funções resultantes do mapeamento das interfaces dos objetos (definidas em IDL) para as linguagens de programação nas quais estes estão implementados. Essas funções representam chamadas aos métodos que implementam as operações dos objetos e são invocadas pelo ORB.

A existência de um Skeleton não implica na existência de um stub respectivo, uma vez que pedidos de operações podem ser feitos pelos clientes de maneira dinâmica (Invocação Dinâmica).

Ao executar um pedido a implementação do objeto pode necessitar de alguns serviços do ORB, o que implica na existência de uma interface entre esses dois elementos. No entanto, devido a grande variedade de características distintas que os objetos possuem, fica difícil especificar uma única interface que seja conveniente a todos os objetos. Esse problema é, então, resolvido através de Adaptadores de Objetos, os quais fornecem, para as implementações de objetos, uma interface através da qual essas implementações podem solicitar os serviços do ORB.

3.3 OSCA

A arquitetura OSCA foi projetada pela companhia de telecomunicações norte-americana Bellcore. O objetivo desta arquitetura é possibilitar que as companhias clientes do Bellcore possam combinar seus produtos de software (aplicações) da maneira que melhor atenda às suas necessidades. Considerando que essas aplicações, na grande maioria das vezes, serão heterogêneas entre si e possivelmente distribuídas, pode-se dizer, então, que o objetivo principal desta arquitetura é permitir a interoperabilidade entre esses produtos.

A arquitetura se baseia nas funcionalidades de cada uma das aplicações envolvidas, as quais são agrupadas em camadas: camada de dados, camada de processamento e camada de usuário (Figura 3.7). Cada camada é formada por *blocos de construção* (BCs).

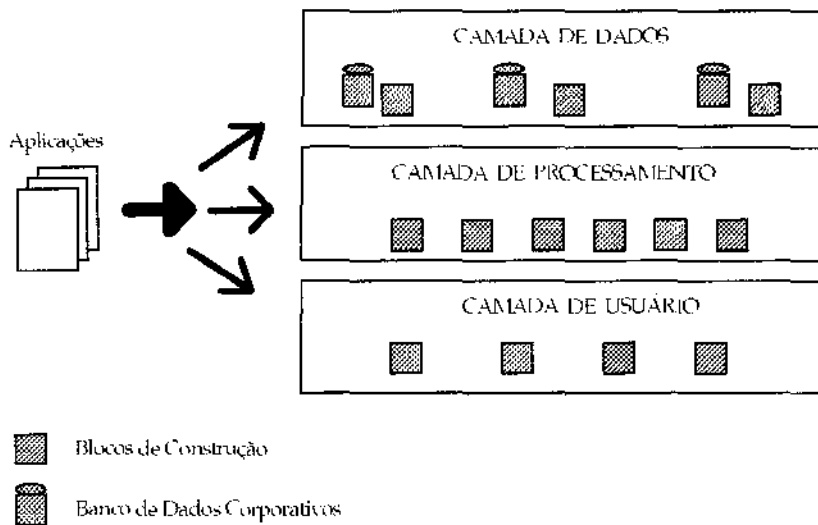


Figura 3.7: Agrupamento das aplicações em camadas.

Um *bloco de construção* (BC) é a unidade fundamental pela qual as aplicações são formadas. Nessa arquitetura, o desenvolvimento (projeto, implantação e manutenção) de software de aplicação ocorre no nível de BC.

3.3.1 Camada de Dados

A camada de dados administra os dados corporativos e fornece funcionalidades que são responsáveis pelo gerenciamento desses dados, sem levar em conta quais processos estão fazendo uso deles. Estas funcionalidades envolvem operações de criação, remoção, atualização e consulta sobre esses dados corporativos de maneira semanticamente consistente, garantindo assim a integridade dos mesmos. Esta camada ainda suporta o gerenciamento de redundância e consultas *ad-hoc*.

Apenas clientes autorizados podem ter acesso a esses dados. A cada banco de dados corporativo existe apenas um único BC associado que gerencia o acesso aos seus dados, fornecendo todas as funcionalidades descritas acima.

3.3.2 Camada de Processamento

A camada de processamento está basicamente relacionada com os processos de negócios. Os processos nesta camada manipulam dados, constroem relatórios, fazem consultas complexas e controlam o fluxo dos processos, devido ao fato de que um Bloco de Construção da Camada de Processamento (BCCP), ao executar alguma função, talvez necessite de funções localizadas em outro BCCP.

Processos podem obter ou enviar dados para a camada de dados e, eventualmente, podem manter dados locais, se necessário. A responsabilidade de gerenciar os dados corporativos é da camada de dados. Quando um BCCP necessita de algum dado corporativo, ele faz um pedido para a camada de dados solicitando as informações necessárias.

3.3.3 Camada de Usuário

A camada de usuário apresenta funcionalidades que possibilitam que usuários finais tenham acesso ao ambiente heterogêneo de maneira amigável. Esta camada possibilita que o usuário possa enxergar os vários produtos de software envolvidos no ambiente como sendo um único sistema. Através dela, os usuários têm acesso aos processos oferecidos pela camada de processamento e podem fazer consultas *ad-hoc* sobre os dados corporativos mantidos pela camada de dados.

A camada de usuário permite que projetos de aplicações sejam simplificados, uma vez que ela encapsula as complexidades envolvidas nas interações com os usuários.

3.3.4 Software de Comunicação

Para que os BCs que constituem cada uma das três camadas possam interagir, é necessário a existência de algum meio de comunicação (Figura 3.8). Este meio é fornecido através de um software de comunicação, cujas funcionalidades podem ser divididas em três diferentes classes:

- *Serviços de comunicação*, os quais possibilitam o envio de dados entre BCs localizados remotamente e entre BCs e usuários finais;
- *Serviços de diretório* de rede, que oferecem suporte aos serviços de comunicações fornecendo nomes lógicos aos mapeamentos dos endereços físicos e acesso às informações dos usuários finais tais como *profiles*, *scripts de logon*, entre outros.
- *Serviços de gerenciamento* de BCs, que fornecem informações sobre disponibilidade e desempenho de BCs.

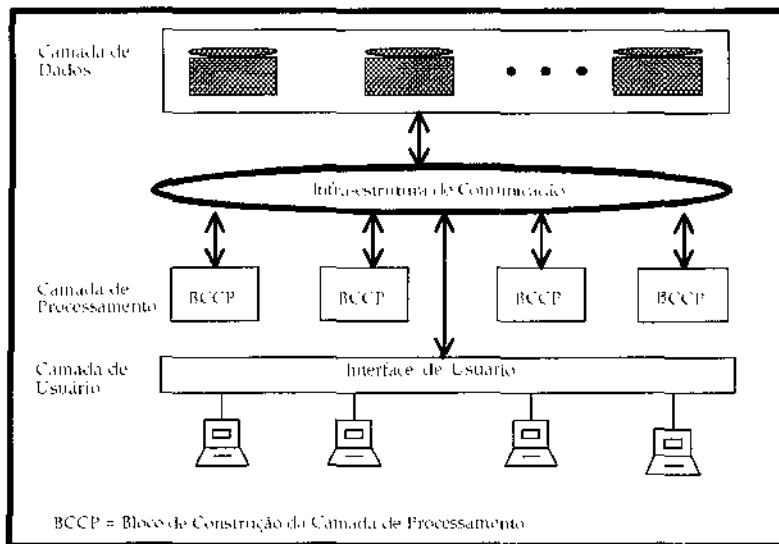


Figura 3.8: Interação entre as camadas de blocos de construção.

3.3.5 Blocos de Construção

Blocos de construção são módulos de software bem definidos contendo um conjunto de funcionalidades coerentes e que são independentes em termos de instalação e execução. Cada BC gerencia o seu próprio estado, podendo ser trocado, atualizado ou realocado sem qualquer impacto no resto do sistema, desde que seus contratos sejam preservados.

Cada BC deve ser altamente coeso e fracamente acoplado em relação às funcionalidades dos demais. No entanto, cada BC deve interoperar com outros BCs para que, cooperativamente, possam realizar as funções das aplicações. As interações entre os BCs se dá através de *contratos* (Seção 3.3.6).

Um BC pertence a apenas uma das três camadas. Não existe BC que, por exemplo, esteja parte na camada de dados e parte na camada de processamento.

3.3.6 Contratos

Um BC disponibiliza suas funcionalidades a outros BCs através de uma ou mais interfaces, chamadas de contratos (Figura 3.9). O termo contrato é utilizado a fim de mostrar que um BC deve ser responsável por fornecer suas funcionalidades de maneira adequada. Para que um BC tenha acesso às funcionalidades de outro BC basta saber apenas quais são os contratos que lhe interessam. Não é preciso que ele tenha conhecimento de como estas funcionalidades estão implementadas.

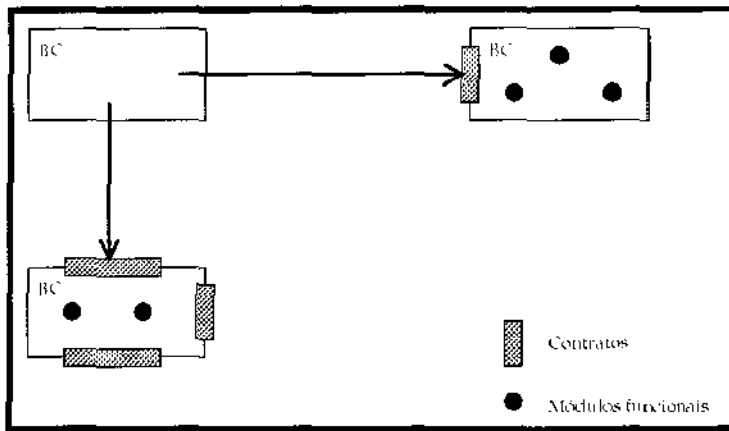


Figura 3.9: Blocos de Construção e seus contratos.

3.4 Conclusão

Neste capítulo foram apresentadas três arquiteturas distribuídas: DOMS, CORBA e OSCA. O principal objetivo das arquiteturas DOMS e CORBA é, dado um conjunto de aplicações HAD, fazer com que os elementos desse conjunto consigam se comunicar de maneira transparente a fim de realizar alguma atividade. Em outras palavras, o que estas duas arquiteturas se propõem é promover a interoperabilidade entre aplicações HAD. No entanto isto não é uma tarefa fácil de ser alcançada, uma vez que cada elemento desse conjunto tem uma implementação própria e proprietária: cada qual com uma interface e um protocolo de comunicação próprio. Porém esse problema pode ser resolvido em duas etapas, a saber:

Primeira Etapa: Padronização da Interface:

Se tivermos duas aplicações, de diferentes fabricantes, que querem se comunicar, o que pode ser feito é um mapeamento das informações entre elas. Mas se tivermos várias aplicações de diferentes fabricantes que desejam se comunicar, fazer um mapeamento dois a dois das informações talvez não seja a melhor solução (veja Figuras 3.2 e 3.3). Uma boa solução seria a padronização da interface, o que possibilitaria uma maior portabilidade para as aplicações. Esta padronização pode ser feita através de uma IDL. Desta maneira as aplicações já conseguem trocar mensagens entre si.

Segunda Etapa: Padronização de Protocolos:

Para as aplicações trocarem mensagens é necessário uma padronização do protocolo de comunicação utilizado pelas aplicações. Essa padronização irá permitir que as aplicações interoperem. A interoperação ocorre quando um programa de um sistema consegue ter acesso aos programas e dados de um outro sistema. E isto só é possível se os dois sistemas usam o mesmo protocolo, ou seja, os mesmos formatos e seqüências de mensagens. Uma maneira de possibilitar essa padronização é através do mecanismo de Remote Procedure Call (RPC).

As arquiteturas DOMS e CORBA oferecem serviços distribuídos que possuem uma IDL e um protocolo padronizado. Esses serviços são chamados de *serviços de middleware* (Figura 3.10) uma vez que se encontram numa camada sobre os Sistemas Operacionais e softwares de rede e sob as especificações das aplicações [BERN96].

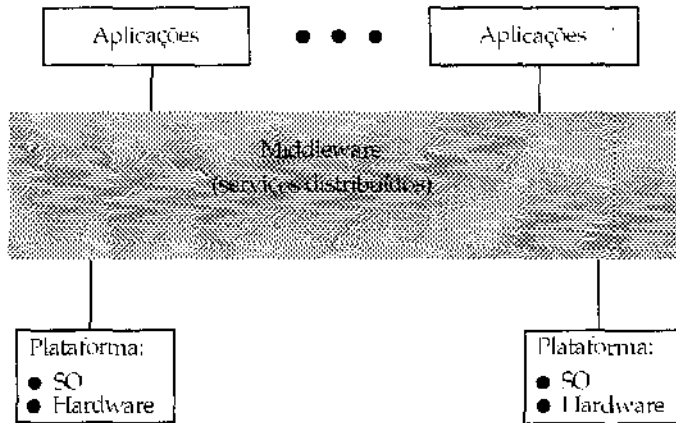


Figura 3.10: Middleware

A arquitetura OSCA apresenta uma forma um pouco diferente de tratar a questão interoperabilidade quando comparada com DOMS e CORBA. Na arquitetura OSCA é abordado mais a questão das funcionalidades dos recursos computacionais envolvidos no processo de interoperação. Não é dada muita ênfase na forma como deve ser feita a comunicação entre estes. Os recursos são agrupados em camadas de acordo com suas funcionalidades. Cada camada é formada por BCs entre os quais ocorre a interoperabilidade.

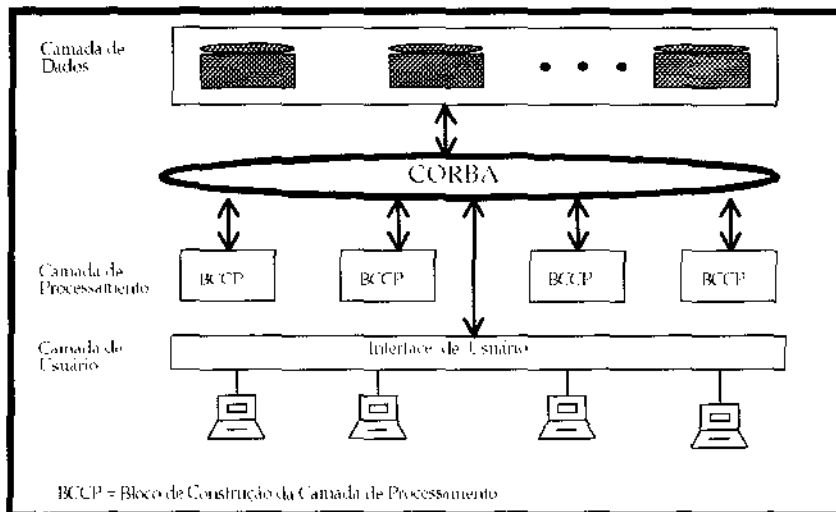


Figura 3.11: Esquema da arquitetura distribuída a ser proposta.

Os conceitos apresentados pelas arquiteturas DOMS, CORBA e OSCA são utilizados nas fases do método apresentado no Capítulo 5 da seguinte forma:

- *durante a fase de análise* são identificados os objetos de interface, de controle e entidade que irão compor, respectivamente, as camadas de usuário, de processamento e de dados propostas pela arquitetura OSCA. Nesta fase é construído também um modelo de objetos que define a semântica dos objetos componentes da aplicação (arquitetura DOMS);
- *durante a fase de projeto* os BCs são identificados e distribuídos segundo o paradigma cliente/servidor. Os BCs da camada de dados são colocados em máquinas servidoras, os da camada de usuário em máquinas clientes e os da camada de processamento podem ser colocados tanto em máquinas clientes como em máquinas servidoras. A localização dos BCs desta última camada depende da afinidade que estes possuem com os BCs das demais camadas. São definidos, também, os contratos dos BCs, mapeados para IDL durante a fase de implementação;
- *durante a fase de implementação* BCs são implementados na arquitetura distribuída apresentada na Figura 3.11. Esta arquitetura representa uma abstração das arquiteturas apresentadas neste capítulo.

Capítulo 4 - Análise de Ferramentas CASE

Neste capítulo é apresentada uma filosofia de agrupamento de objetos. Esta filosofia faz parte de um método de desenvolvimento de sistemas de grande porte prescrito pela disciplina de Engenharia da Informação [MART89], [MART90], [KIPP93]. O método é baseado na decomposição funcional. Sendo assim, seus objetos são entidades de dados e processos.

A filosofia a ser apresentada aqui mostra uma forma de agrupar processos e entidades de dados segundo a afinidade que existe entre esses. Este agrupamento é feito durante a primeira fase da Engenharia da Informação (Seção 4.1). O objetivo deste capítulo é verificar como este agrupamento é realizado e implementado por ferramentas CASE que automatizam as fases da Engenharia da Informação. Hoje em dia, existem no mercado dois programas que implementam tais ferramentas: o Application Development Workbench (ADW) da KnowledgeWare Inc. ([KNOW91], [SERC95]) e o Information Engineering Facility (IEF) da Texas Instruments ([TEXA90], [TEXA93]). Uma breve apresentação de como estes dois programas implementam a filosofia de agrupamento é apresentada na Seção 4.2.

4.1 Engenharia da Informação

A Engenharia da Informação prescreve um método de desenvolvimento de sistemas abrangente que auxilia no entendimento das necessidades de informação da empresa. Esse entendimento é feito através da construção de uma representação abstrata da empresa, chamada de *Modelo da Empresa*, formado basicamente pelos dados (informações) e atividades do negócio da empresa e pelas interações entre estes dois. Esses três componentes (dados, atividades e interações), identificados durante a fase inicial da Engenharia da Informação, auxiliam no entendimento dos processos envolvidos na empresa num nível abstrato. A partir desse nível, cada componente é incrementalmente refinado até o ponto em que a combinação dos três forma uma aplicação executável. Esse refinamento ocorre ao longo das fases que compõem a Engenharia da Informação. Essas fases são as seguintes:

- *Planejamento Estratégico da Informação*: estabelece uma visão ampla das necessidades de informação da empresa.
- *Análise de Áreas de Negócio*: é feita uma análise mais detalhada sobre um determinado segmento particular do negócio, denominado Área de Negócio.
- *Projeto de Sistema*: especifica detalhadamente um sistema de informação que suporta um segmento específico da Área de Negócio sem considerar o ambiente computacional.
- *Construção*: implementação de todos os componentes executáveis: programas, banco de dados, instruções de controle de tarefa, formatos de tela, etc. Em resumo, todas as partes necessárias para que a aplicação projetada execute no ambiente computacional selecionado.

Em cada uma das fases são utilizadas técnicas específicas. Essas técnicas geram resultados bem determinados que são utilizados como entrada para a fase seguinte. O grau de detalhamento aumenta conforme se avança pelas fases (Figura 4.1).

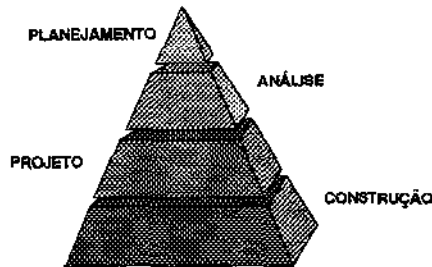


Figura 4.1: Fases da Engenharia da Informação.

O desenvolvimento de sistemas segundo a Engenharia da Informação está baseado no conceito de "dividir para conquistar". Partes menores, componentes do conjunto de requisitos de informação do negócio como um todo, são gradualmente refinadas e detalhadas. Cada subconjunto deve apresentar um tamanho suficientemente pequeno de forma a permitir seu gerenciamento.



Figura 4.2: Interligação das Fases da Engenharia da Informação.

4.1.1 Planejamento Estratégico da Informação

Durante a fase de Planejamento [MART90] é analisado quais são as necessidades de informações da empresa. Como resultado desta análise têm-se os seguintes modelos:

- *Modelo Funcional*: onde são mapeadas as funções/processos que a empresa deve realizar para suportar o negócio.
- *Modelo de Dados*: representa o patrimônio de dados da empresa e os relacionamentos entre eles.
- *Modelo Organizacional*: contém os objetivos da empresa, problemas, estrutura gerencial, fatores críticos de sucesso, necessidades de informações não descritas nos modelos anteriores.

Os modelos de Dados e Funcional representam os componentes *dados e atividades* do Modelo da Empresa (Seção 4.1), respectivamente. O terceiro componente, *interações*, é obtido a partir da associação entre estes dois modelos (Figura 4.3). O resultado desta associação é uma matriz cujas linhas são entidades de dados e as colunas, processos. Esta matriz é chamada de *Matriz de Associação*. Suas células (intersecção de linhas e colunas) são marcadas com um **X** caso uma entidade de dados seja utilizada por um processo.

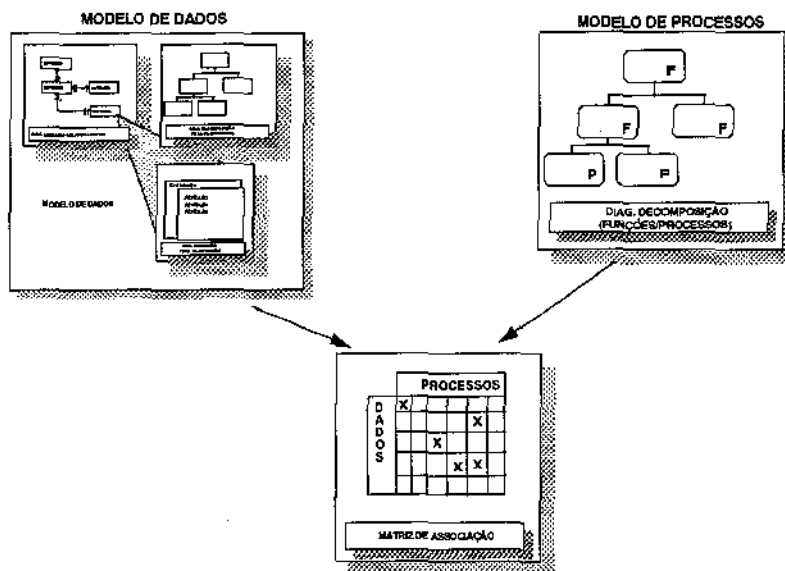


Figura 4.3: Associação entre Dados e Processos.

A Matriz de Associação irá auxiliar a equipe de projeto a elaborar as metas a serem alcançadas no desenvolvimento do sistema. Essas metas serão traduzidas em projetos de sistemas de informação e projetos de bancos de dados. Os projetos são determinados a partir das *Matrizes de Afinidades* entre processos e entre entidades de dados. Essas matrizes são obtidas a partir da Matriz de Associação.

A Figura 4.4 mostra uma Matriz de Afinidades entre processos. As células dessa matriz contém percentuais que mostram as afinidades que existem entre os processos. Na Figura 4.4, os processos P1 e P2 têm 60% de afinidade. Dois processos têm 100% de afinidade se ambos utilizam as mesmas entidades de dados, e 0% se utilizam nenhuma entidade de dados em comum. Processos com afinidades entre 0 e 100% utilizam algumas entidades de dados em comum.

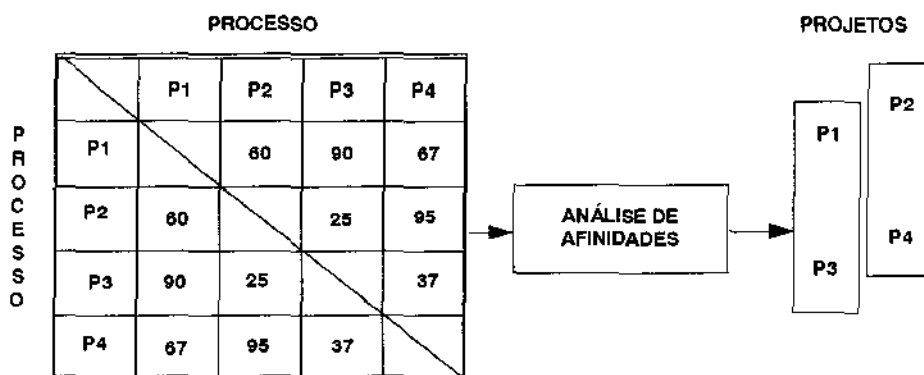


Figura 4.4: Análise de Afinidades.

A partir de uma análise da Matriz de Afinidades entre processos, os projetistas podem identificar projetos de sistemas de informação. Cada projeto de sistema de informação será formado por um grupo de processos. A equipe de projeto deve estabelecer valores de percentuais mínimos para esses agrupamentos (por exemplo, os sistemas de informação serão formados por processos com pelo menos 70% de afinidade entre si).

Uma vez identificados os projetos de sistemas de informação, os projetistas devem identificar os projetos de bancos de dados. Assim como os projetos de sistemas de informação, esses projetos são também identificados a partir de uma Matriz de Afinidades. Porém, a Matriz de Afinidades utilizada agora é entre entidades de dados. A afinidade entre entidades de dados é medida em termos dos processos que as utilizam (duas entidades de dados com 100% de afinidade são utilizadas pelo(s) mesmo(s) processo(s)).

Finalmente, após a análise das Matrizes de Afinidades, os projetistas têm uma lista de projetos de sistemas de informação e bancos de dados os quais devem ser priorizados. A necessidade dessa priorização deve-se ao fato de que a empresa dificilmente terá recursos (humanos e materiais) suficientes para investir na realização de todos esses projetos dentro de prazos adequados. A priorização deve ser feita com base nas informações contidas no Modelo Organizacional.

É importante ressaltar que esta lista de projetos deve ser elaborada cuidadosamente. É com base nesta lista que o Modelo da Empresa será segmentado em Áreas de Negócio que serão submetidas às demais fases da Engenharia da Informação.

4.2 Ferramentas CASE

As ferramentas CASE apresentadas nesta seção são implementadas pelos programas ADW e IEF. Ambos estão baseados na existência de uma enciclopédia a qual é construída inicialmente durante a fase de Planejamento Estratégico da Informação. Nesse momento são colocadas nessa enciclopédia o Modelo de Dados, o Modelo Funcional e o Modelo Organizacional.

Conforme o desenvolvimento do sistema avança para as demais fases da Engenharia da Informação, essa enciclopédia vai sofrendo alterações. Um exemplo dessas alterações é a questão dos atributos das entidades de dados. Durante a fase de planejamento não é conside-

rado quais atributos as entidades de dados devem ter. No entanto esta preocupação surge na fase seguinte (Análise de Áreas de Negócio).

A seguir são apresentadas as ferramentas do ADW e IEF aplicadas a fase de Planejamento Estratégico da Informação.

4.2.1 ADW

Para a fase de Planejamento, o programa ADW [KNOW91] possui as seguintes ferramentas:

- *Entity Relationship Diagrammer*: permite a construção do Modelo de Dados.
- *Decomposition Diagrammer*: permite a construção do Modelo Funcional.
- *Association Matrix Diagrammer*: permite a construção da Matriz de Associação e a derivação das respectivas Matrizes de Afinidades.
- *Property Matrix Diagrammer*: permite a definição de certas características das informações (dados e atividades) coletadas.

A Matriz de Associação é gerada automaticamente com base nas informações cadastradas para cada um dos processos. No momento que se inclui um processo no Modelo Funcional, é possível indicar quais são as entidades de dados que este utiliza. Desta forma, a ferramenta que gera a Matriz de Associação pode identificar, para cada entidade de dados, quais são os processos que a utilizam.

Depois de gerada, o conteúdo das células da Matriz de Associação pode ser editado pelo usuário. Para cada X marcado, este pode especificar que tipo de operação o processo executa sobre a entidade de dados: criação, leitura, atualização, remoção ou uma combinação destes. Esta informação fornecida pelo usuário é conhecida como *propriedade CRUD* (Create, Read, Update e Delete).

A partir da Matriz de Associação é possível gerar automaticamente as Matrizes de Afinidades entre processos e entre entidades de dados. Essa geração de matrizes é acompanhada por uma sugestão de como agrupar os processos e as entidades de dados em sistemas de informação e bancos de dados, respectivamente. Este agrupamento é considerado uma sugestão pois se baseia apenas nas afinidades entre processos e entre entidades de dados, não considerando, por exemplo, a existência de eventuais dependências funcionais. A decisão de qual deve ser o melhor agrupamento a ser feito é responsabilidade dos projetistas.

4.2.2 IEF

Para a fase de Planejamento, o programa IEF [TEXA93] apresenta praticamente as mesmas ferramentas que aquelas apresentadas no ADW diferindo na forma como são implementadas. Entre tais diferenças, a mais importante diz respeito à maneira como é feita a análise da Matriz de Associação.

A ferramenta de construção da Matriz de Associação apresentada pelo IEF possui duas opções para a análise dessa matriz. A primeira é a geração automática das Matrizes de

Afinidades, assim como existe na ferramenta do ADW. Porém, no IEF, essa opção não gera nenhuma sugestão de agrupamento de processos e entidades de dados como ocorre no ADW. Essa decisão fica a cargo dos projetistas.

A segunda opção (que não existe na ferramenta do ADW) permite que a equipe de projeto identifique, a partir da Matriz de Associação, Áreas de Projetos. Estas áreas são identificadas com base nas propriedades CRUD contidas nas células da Matriz de Associação. A ferramenta agrupa processos e entidades de dados em um *cluster*⁴. Esse agrupamento irá auxiliar os projetistas a:

- identificar grupos de processos que atuam sobre as mesmas entidades de dados;
- identificar grupos de entidades de dados que são utilizadas pelos mesmos processos;
- verificar se cada entidade de dados é criada por pelo menos um processo;
- determinar se existe alguma dependência que gere algum impacto na seqüência da implementação do sistema; e
- verificar quais processos e entidades de dados pertencem a que Área de Negócio.

Através desses agrupamentos, projetistas e futuros usuários podem identificar processos, entidades de dados e interações que possam estar faltando. No entanto, esses agrupamentos são apenas sugestões de como trabalhar com os processos e as entidades de dados. A maneira como deverá ser feito realmente os agrupamentos é responsabilidade dos projetistas. Os agrupamentos identificados serão transformados em Áreas de Projetos (ou Áreas de Negócio) as quais serão submetidas às demais fases da Engenharia da Informação.

4.3 Conclusão

Neste capítulo foi apresentada uma filosofia de agrupamento de objetos. Esta filosofia, que faz parte da Engenharia da Informação, esta baseada nas afinidades que existem entre os objetos (entidades de dados e processos). Com base nessas afinidades são identificados projetos de sistemas de informação e bancos de dados, os quais poderão ser, eventualmente, distribuídos ao longo de uma rede.

O objetivo é poder aplicar esta filosofia no método proposto no Capítulo 5. Através da análise de afinidade entre os objetos, obtém-se projetos de subsistemas. A afinidade entre os objetos é medida em termos dos relacionamentos de associação, agregação e colaboração que existem entre os mesmos.

Na segunda parte deste capítulo foram apresentados dois programas, ADW e IEF, que implementam ferramentas CASE que automatizam as fases da Engenharia da Informação. Em particular, foi analisado como estes programas implementam a filosofia de agrupamento dos objetos baseada na análise da Matriz de Associação. A ferramenta implementada pelo IEF se mostra mais efetiva na análise da Matriz de Associação do que a do ADW, que se limita a gerar as Matrizes de Afinidades, as quais são acompanhadas por sugestões de como agrupar entidades de dados em bancos de dados e processos em sistemas de informação. A ferramenta do IEF, além de gerar as Matrizes de Afinidades, possui uma outra opção

⁴O processo de agrupamento é chamado de *Interaction Clustering*.

que faz uma análise da Matriz de Associação a partir das propriedades CRUD contidas nas células da mesma. Esta análise auxilia a equipe de projetos a identificar as Áreas de Projetos (ou Áreas de Negócio).

Finalmente, com a lista de projetos de sistemas de informação e bancos de dados e com a lista de Áreas de Negócio, os projetistas podem decidir que projeto vai em que Área de Negócio. Essas áreas serão, então, submetidas as demais fases da Engenharia da Informação.

Capítulo 5 - O Método Proposto

Neste capítulo, em primeiro lugar é apresentada a descrição do método orientado a objetos de desenvolvimento de sistemas de informação distribuídos, objeto desta dissertação, e em seguida é feita uma proposta de ferramenta para automatizar o processo de agrupamento de objetos em subsistemas. O capítulo é concluído com um resumo do método e uma breve comparação entre a ferramenta proposta e as ferramentas apresentadas no Capítulo 4.

5.1 Descrição do Método

O método aqui proposto foi elaborado a partir dos conceitos e modelos apresentados nos capítulos 2 (métodos) e 3 (arquiteturas). O que se propõe é um processo de desenvolvimento de Sistemas de Informação Distribuídos (SIDs) dividido em três fases: análise, projeto e implementação (Figura 5.1). Um esquema da evolução deste processo é apresentado na Figura 5.2.

A partir da especificação de requisitos apresentada pelos usuários, os analistas constroem, primeiramente, o Modelo de Casos de Uso Estendido. O objetivo deste modelo é promover um melhor entendimento do funcionamento do sistema que se pretende desenvolver (Seção 2.4 - Casos de Uso). A partir desse modelo são construídos outros dois modelos:

- *Modelo de Objetos*: onde são definidas as classes (em termos de atributos e responsabilidades) e os relacionamentos (associações) que existem entre as mesmas;
- *Modelo Dinâmico*: onde são descritos os comportamentos que as classes deverão apresentar durante a execução do sistema.

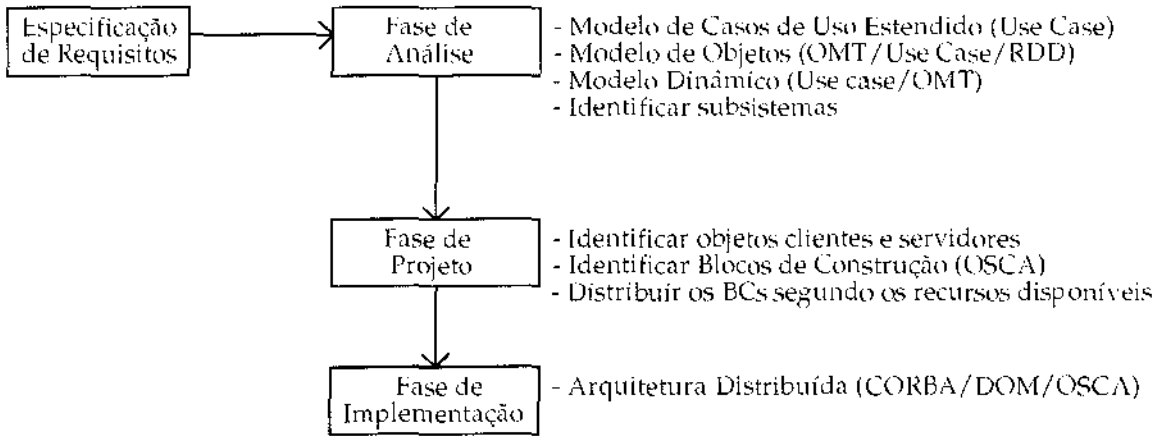
A fim de facilitar o projeto e implementação do SID, classes são agrupadas em subsistemas segundo a afinidade que existe entre elas. Esta afinidade é determinada a partir de regras aplicadas às informações contidas nos modelos de objetos e dinâmico. Os objetivos deste agrupamento são dois:

1. identificar quais classes devem ficar juntas no momento de distribuir o sistema, e
2. propor uma distribuição das classes de objetos que proporcione um bom desempenho para o sistema.

Para garantir o segundo objetivo é proposta uma métrica a qual está baseada na frequência com que as colaborações inter-subsistemas ocorrem: para que o SID apresente um bom desempenho, a frequência das colaborações intra-subsistemas deve ser maior do que a frequência das colaborações inter-subsistemas. Isto deve garantir que o tráfego de dados na rede seja o menor possível.

Neste método é considerado que o SID é implementado numa arquitetura distribuída cliente/servidor. Sendo assim, é necessário identificar quais classes devem ser clientes e quais devem ser servidoras. Uma classificação nesse sentido é conduzida em cada subsiste-

ma durante a fase de projeto. A partir dessa classificação, blocos de construção são identificados e implementados (fase de implementação).



Legenda

Use Case: Método baseado em Casos de Uso proposto por Jacobson et al.
 OMT: Object Modeling Technique proposto por James Rumbaugh et al.
 RDD: Responsibility Design Driven, método proposto por Rebecca Wirfs-Brock et al.

Figura 5.1: Fases do desenvolvimento do SID.

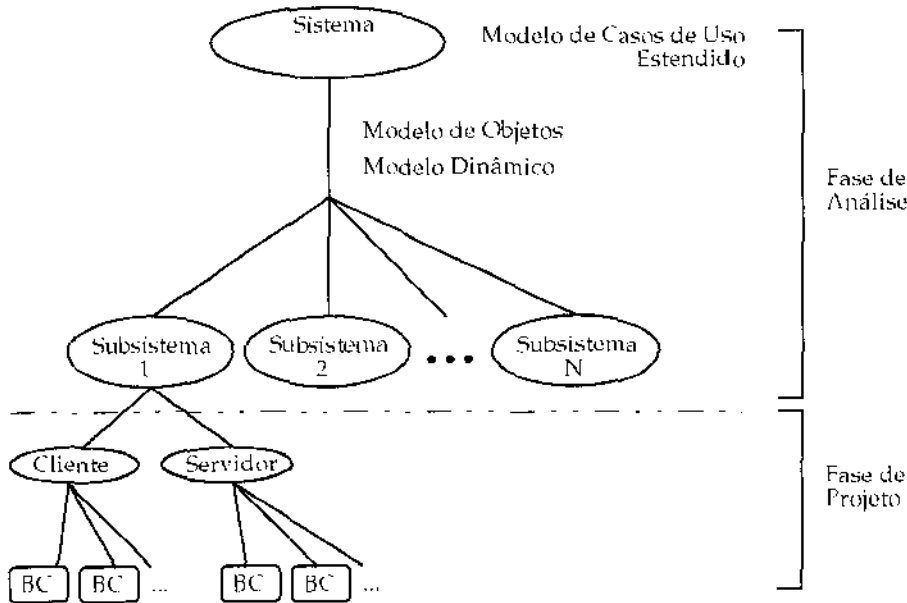


Figura 5.2: Distribuição do sistema de informação.

5.1.1 Fase de Análise

A partir da especificação de requisitos os seguintes modelos são construídos durante a fase de análise:

- *modelo de casos de uso estendido*: descreve o funcionamento da aplicação em termos de casos de uso da mesma [JACO92];
- *modelo de objetos*: descreve as classes de objetos segundo suas responsabilidades [WIRF90a] e atributos; também são identificados eventuais relacionamentos (associações) que possam existir entre essas classes [RUMB91]. Os objetos identificados são classificados em objetos de interface, de controle e entidade [JACO92]. Na construção deste modelo é utilizada a notação apresentada por [RUMB91];
- *modelo dinâmico*: descreve o comportamento dinâmico dos objetos a partir dos diagramas de interação [JACO92] e de estado [RUMB91].

Esses modelos deverão promover uma visão conceitual da aplicação sem considerar os aspectos de implementação. Com base nas informações contidas nesses modelos, será feita uma primeira distribuição dos objetos em subsistemas. Essa distribuição deverá ser conduzida de maneira que a aplicação venha apresentar um bom desempenho.

A Figura 5.3 apresenta como esses modelos estão relacionados. A partir do modelo de casos de uso são construídos os modelos de Objetos e Dinâmico. A construção dos subsistemas é feita a partir de duas regras e uma métrica aplicadas sobre informações contidas nestes dois últimos modelos.

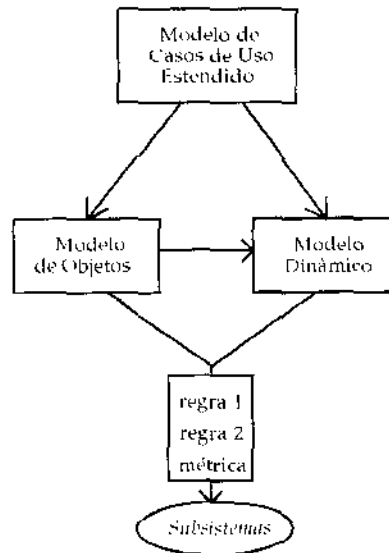


Figura 5.3: Relacionamento entre os modelos da fase de análise.

5.1.1.1 Modelo de Casos de Uso Estendido

O *modelo de casos de uso estendido* é formado pelo modelo de casos de uso descrito na Seção 2.4.1.1.1 acrescido das frequências com que cada caso de uso ocorre durante um determinado intervalo de tempo. Estas frequências devem ser utilizadas pela métrica aplicada na construção dos subsistemas. A apresentação da construção do modelo está dividida em duas etapas:

- *Primeira Etapa*: identificar os casos de uso.
- *Segunda Etapa*: identificar a frequência de cada caso de uso.

Primeira Etapa: Identificar os casos de uso

Nesta primeira etapa é feita a construção do modelo de casos de uso conforme apresentado na Seção 2.4.1.1.1. Esta construção envolve a identificação dos seguintes componentes a partir da especificação de requisitos:

- atores;
- casos de uso e
- extensões (dos casos de uso).

Os **atores** representam os usuários que irão interagir com o SID. Na aplicação descrita no Capítulo 6, os atores são os *projetistas* e *usuários do cadastro*. A partir das operações que esses atores devem solicitar para a aplicação, **casos de uso** são identificados. Um caso de uso representa uma seqüência de tarefas que o SID deve ser responsável por realizar a fim de atender a um pedido feito por um ator. No exemplo de caso de uso da Figura 5.4, o ator usuário do cadastro solicita a alteração do elemento de rede poste.

Caso de uso 5: Alterar o elemento de rede POSTE em BDG

Descrição:

Para alterar um elemento de rede POSTE existente na rede externa o Usuário do Cadastro (UC) deve selecionar a operação de alteração do Menu Específico (ME). Será apresentada para o UC uma Caixa de Diálogo (CD) com os valores dos atributos do POSTE para que sejam alterados. Uma vez encerradas as alterações, o sistema deverá enviar os novos valores dos atributos para o Banco de Dados Geográfico a fim de que sejam modificados.

Figura 5.4: Caso de uso.

As **extensões** dos casos de uso representam as situações alternativas não descritas nas seqüências de tarefas dos mesmos. Por exemplo, na descrição do caso de uso da Figura 5.4 não é considerada a possibilidade do elemento de rede poste, que foi alterado, estar par-

ticipando de algum projeto em andamento. Na descrição de uma situação alternativa, deve-se indicar exatamente em que momento do caso de uso a mesma deve ocorrer (Figura 5.5).

Caso de uso 8: Validação da alteração do elemento POSTE em BDG entre cadastro/projeto

Descrição:

A validação da alteração do elemento POSTE em BDG entre cadastro/projeto ocorre em **Alterar o elemento de rede POSTE em BDG** depois que a operação alteração foi concluída. Nesse momento, o sistema verifica se o POSTE alterado não está participando de algum projeto em andamento. Se for o caso, o módulo de cadastro deverá enviar uma mensagem para os responsáveis do respectivo projeto informando das alterações. Esses deverão decidir pela reextração ou não da área geográfica.

Figura 5.5: Extensão.

Atores, casos de uso e extensões são representados pelo modelo de caso de uso conforme mostra a Figura 5.6. A flecha bidirecional ligando ator e caso de uso representa a interação que existe entre ambos.

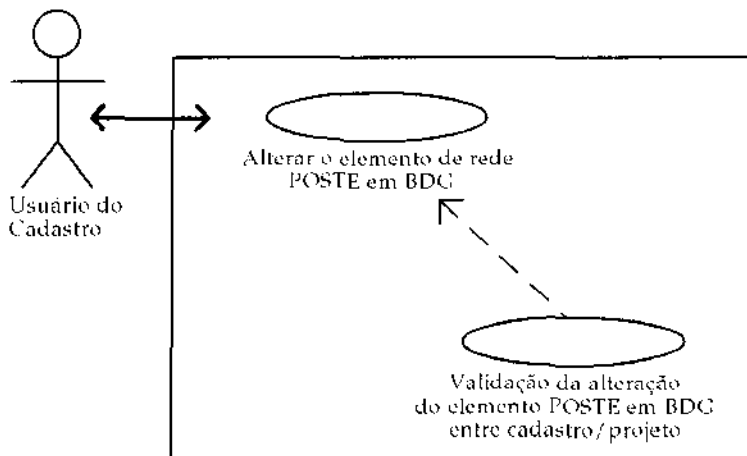


Figura 5.6: Exemplo de modelo de casos de uso.

Segunda Etapa: Identificar a freqüência de cada caso de uso

A freqüência de um caso de uso é dada pela fórmula apresentada na Figura 5.7. Esta fórmula envolve duas variáveis: o número de vezes que o caso de uso ocorre em um determinado intervalo de tempo (n_{CSU}) e o intervalo de tempo T . A primeira variável deve ser determinada pelo usuário conforme apresentado a seguir. A segunda variável, intervalo de tempo, é determinada da seguinte forma:

- para cada tipo de ator (usuário), identificar qual é o intervalo de tempo mínimo para que todos os casos de uso iniciados por ele sejam exercitados pelo menos uma vez.
- o intervalo de tempo T é o máximo de todos os intervalos obtidos (Figura 5.8).

$$f_{CsU} = \frac{n_{CsU}}{T}$$

onde
 f_{CsU} : frequência do caso de uso
 n_{CsU} : número de vezes que o caso de uso ocorre em T .
 T : intervalo de tempo

Figura 5.7: Definição da frequência de um caso de uso.

$$T = \max\{T_{A1}, T_{A2}, \dots, T_{AN}\}$$

onde
 T : intervalo de tempo
 T_{Am} : intervalo de tempo do ator m ($1 \leq m \leq N$)

Figura 5.8: Definição do intervalo de tempo.

A identificação do número de vezes que cada caso de uso ocorre em T deve ser conduzida ao nível de atores. Para cada ator deve-se:

1. listar todos os casos de uso iniciados por ele, e
2. solicitar que ele identifique o número de vezes que cada caso de uso ocorre em T (n_{CsU}).

Para obter a frequência de cada caso de uso aplica-se a fórmula da Figura 5.7. Porém, se a quantidade de casos de uso iniciados por um mesmo ator for muito grande, a execução do segundo item listado acima pode se tornar impraticável. Por exemplo, o número de casos de uso que o usuário do cadastro inicia é superior a 100. Para resolver este problema, casos de uso podem ser agrupados de acordo com as tarefas que representam. O ator deve, então, identificar o número de vezes que a funcionalidade representada pelo agrupamento ocorre em T (n_{G-CsU}). Para a situação do usuário do cadastro, os casos de uso poderiam ser agrupados da seguinte forma:

- casos de uso de manipulação dos bancos de dados;
- casos de uso de manipulação do elemento de rede poste

- casos de uso de manipulação do elemento de rede armário;
- etc.

Quando estes agrupamentos se fazem necessários, é dito que a identificação dos n_{CSU} s é conduzida em vários níveis (Figura 5.9). No nível ZERO cada agrupamento abrange uma grande quantidade de casos de uso. No nível UM, cada agrupamento do nível ZERO é expandido em uma nova lista de casos de uso. Se neste nível a quantidade de casos de uso ainda for muito grande, novos agrupamentos podem ser feitos. Listas (para identificação de n_{CSU} e eventuais n_{G-CSU}) e agrupamentos devem ser feitos até que todos os casos de uso sejam devidamente avaliados.

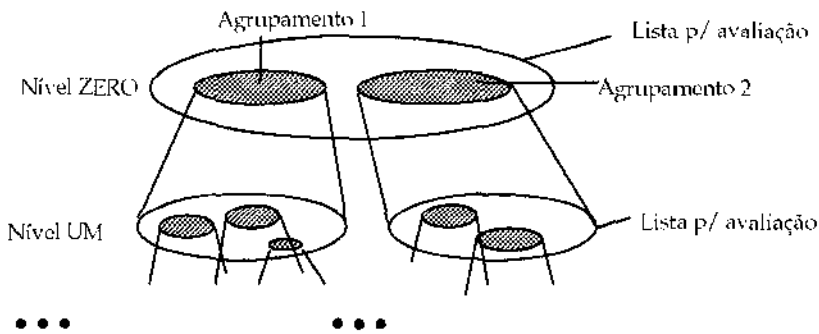


Figura 5.9: Organização em níveis das listas de casos de uso.

É possível que o ator não seja capaz de fazer a identificação dos n_{CSU} e n_{G-CSU} de todas as listas. Neste caso, os casos de uso das listas não avaliadas recebem a frequência atribuída ao agrupamento do qual foram derivados. Por exemplo, se o usuário do cadastro não for capaz de avaliar a lista derivada do agrupamento de casos de uso de *manipulação do elemento de rede poste*, todos os casos de uso dessa lista terão a mesma frequência atribuída ao agrupamento.

A frequência de cada caso de uso deve ser documentada no modelo de casos de uso conforme apresentado na Figura 5.10. Neste exemplo, o intervalo de tempo T foi identificado como sendo quatro dias. Como já apontado, essas frequências são utilizadas durante a construção dos subsistemas.

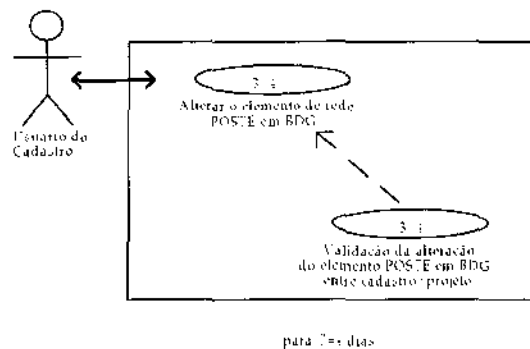


Figura 5.10: Representação dos casos de uso e suas respectivas frequências.

5.1.1.2 Modelo de Objetos

A construção do modelo de objetos é feita após a modelagem dos casos de uso e envolve as seguintes atividades:

- identificar as classes de objetos;
- identificar as associações que existem entre essas classes e
- organizar as definições dessas classes segundo o relacionamento de generalização.

Classes de Objetos

Os objetos que irão compor a aplicação são identificados a partir dos casos de uso listados no respectivo modelo e classificados em objetos de interface, objetos de controle e objetos entidades. No caso de uso da Figura 5.4, os seguintes objetos foram identificados:

- *Objetos de interface*: Menu Específico de Cadastro e Caixa de Diálogo;
- *Objeto de controle*: Poste Controle;
- *Objeto entidade*: Poste (Cadastro).

Como já apresentado na Seção 2.4.1.2, os papéis desses três tipos de objetos são os seguintes:

- *Objetos de interface*: responsáveis por garantir a interação que deve existir entre os usuários e o SID.
- *Objetos entidade*: representam aquelas informações que são normalmente armazenadas em bancos de dados.
- *Objetos de controle*: responsáveis pelas atividades que não são exclusivas de objetos de interface ou de objetos entidade. De um modo geral, são objetos que coordenam as atividades que a aplicação deve realizar a fim de cumprir os pedidos solicitados pelos usuários.

Um mesmo objeto pode estar presente em mais de um caso de uso. Por exemplo, o objeto de controle Poste Controle participa de todos os casos de uso que envolvem alguma operação sobre o elemento de rede Poste (Capítulo 6). Em cada caso de uso que o objeto participa este possui responsabilidades que devem ser cumpridas. As responsabilidades que os objetos listados anteriormente devem cumprir durante a execução do caso de uso da Figura 5.4 são as seguintes:

Objeto	Responsabilidades
Menu Específico de Cadastro	- oferecer a opção de alterar poste
Caixa de Diálogo	- apresentar os valores antigos do elemento de rede POSTE - ler os novos valores dos atributos do elemento de rede POSTE
Poste Controle	- coordenar a execução da alteração do elemento POSTE
Poste (Cadastro)	- fazer todas as consistências necessárias antes de alterar os valores dos atributos do elemento POSTE

Figura 5.11: Exemplo de responsabilidades de objetos.

Os atributos dos objetos são identificados a partir da especificação de requisitos. Nesta fase, uma definição precisa dos atributos não se faz necessária. Porém é interessante que para cada objeto, além de uma lista de responsabilidades, seja feita também, uma lista de características. A partir desta última lista, os atributos são derivados.

Os objetos de um mesmo tipo (interface, controle ou entidade) que tenham as mesmas responsabilidades e as mesmas características são agrupados em uma classe concreta de tipo correspondente. Tem-se, então, três tipos de classes: *classes de interface*, *classes de controle* e *classes entidades*. Essas classes formam a base para a construção das três camadas componentes da arquitetura distribuída cliente/servidor utilizada durante a fase de implementação: camada de dados (classes entidade), camada de processamento (classes de controle) e camada de usuário (classes de interface).

Associação

Associações representam os relacionamentos que existem entre as classes. Na Figura 5.12 a classe Poste está associada à classe Armário. A representação da cardinalidade segue a notação apresentada por [RUMB91]. No exemplo, uma instância da classe Poste pode estar relacionada com exatamente uma instância da classe Armários. De maneira oposta, uma instância da classe Armário pode estar relacionada com exatamente uma instância da classe Poste.



Figura 5.12: Associação.

Um tipo especial de associação é a associação de agregação. Agregação representa a idéia de "todo-parte" ou "é-parte-de": objetos representando componentes de um conjunto são associados a um objeto que representa o conjunto todo (Figura 5.13).

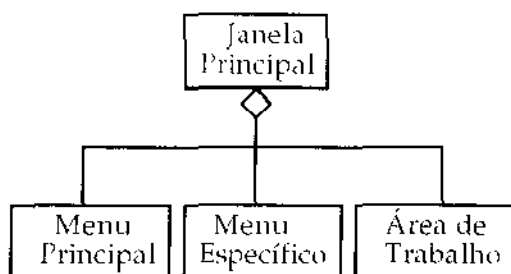


Figura 5.13: Agregação.

Normalmente, associações são identificadas entre objetos entidade. Porém, nem sempre isto é verdade: na Figura 5.13 a associação de agregação foi identificada entre objetos de interface. As associações/agregações identificadas entre as classes de objetos são utilizadas na construção de subsistemas.

Generalização

Generalização representa um relacionamento entre uma classe e uma ou mais versões refinadas da mesma. A classe que foi refinada é chamada de *superclasse* e seus refinamentos (especializações), *subclasses*. Na Figura 5.14, a superclasse Elemento de Rede possui várias subclasses (refinamentos) associadas: Poste, Armário, Lance de Cabo, etc. Logo, generalização é um relacionamento n-ário (para $n \geq 2$), representado por um triângulo ramificado, entre uma superclasse e duas ou mais subclasses.

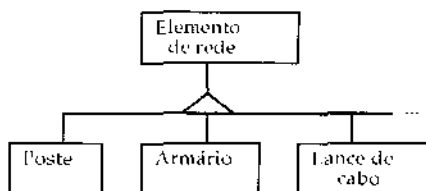


Figura 5.14: Generalização.

O conceito de generalização está relacionado com a construção de *classes abstratas*. Classes abstratas são construídas a partir de um conjunto de classes que possuem atributos e responsabilidades em comum. As classes desse conjunto são associadas à classe abstrata da seguinte forma: a classe abstrata será a superclasse e as classes do conjunto, as subclasses.

Classes abstratas (não instanciáveis) normalmente são construídas a partir de classes concretas (instanciáveis). Na árvore gerada pelo mecanismo de generalização, apenas classes concretas podem ser classes folhas. Uma classe concreta pode ter uma subclasse abstrata desde que esta subclasse abstrata tenha alguma subclasse concreta.

A generalização promove maior re-usabilidade das definições das classes. Para tanto deve-se obter o maior número possível de classes abstratas e tentar fazer com que a maioria das classes concretas sejam subclasses de alguma superclasse abstrata. Isto auxilia os desenvolvedores a identificar classes que estejam faltando e classes que estão a mais no contexto.

5.1.1.3 Modelo Dinâmico

A construção do modelo dinâmico envolve duas atividades principais:

- construção de diagramas de interação [JACO92], e
- construção de diagramas de estado [RUMB91].

Diagramas de interação descrevem as colaborações que devem existir entre os objetos a fim de que os casos de uso sejam realizados. Para cada caso de uso é construído um diagrama de interação.

Diagramas de estado são derivados de diagramas de interação. Para cada classe concreta é feito um diagrama de estado onde é identificado o comportamento que suas instâncias deverão apresentar desde sua criação até sua destruição.

Diagrama de Interação

Cada caso de uso pressupõe uma seqüência de atividades as quais devem ser garantidas pelos objetos que participam da mesma. Diagramas de interação são utilizados para descrever tais seqüências de maneira mais formal. A partir desses diagramas são identificadas as colaborações entre os objetos. Um objeto solicita uma colaboração de outro objeto a fim de poder cumprir com suas responsabilidades [WIRF90a].

O diagrama de interação da Figura 5.15 corresponde ao caso de uso da Figura 5.4. Este diagrama é o mesmo apresentado na Seção 2.4.2.1 acrescido da freqüência do caso uso atribuída a todas as colaborações.

Caso uma colaboração participe de mais de um caso de uso, a sua freqüência será determinada pela maior freqüência atribuída à mesma. A partir desta atribuição de freqüências às colaborações os analistas são capazes de determinar com que freqüência as operações dos objetos são solicitadas. Esta informação é utilizada na construção dos subsistemas.

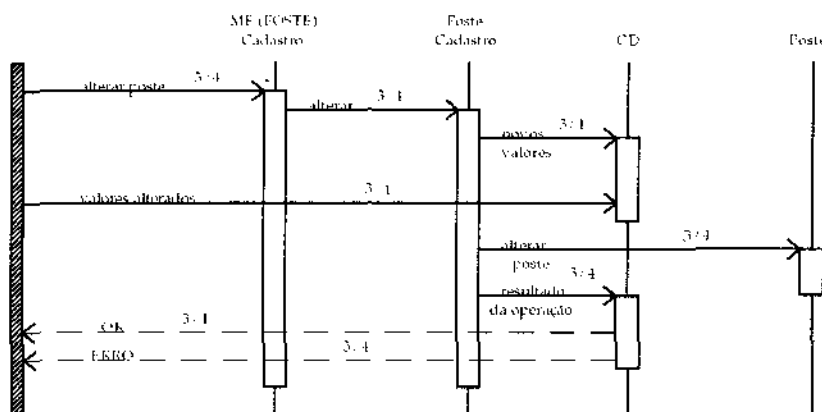


Figura 5.15: Diagrama de Interação da Figura 5.4.

A partir de diagramas de interação pode-se construir a interface de cada objeto. Para tanto, basta considerar todos os diagramas de interação em que determinado objeto participa

e verificar quais são as operações que este é responsável por executar. Essas operações devem ser comparadas com a lista de responsabilidades atribuídas ao objeto (modelo de objetos). Os objetivos dessa comparação são dois:

1. verificar se todas as responsabilidades previstas para cada objeto foram atendidas, e
2. verificar se algum objeto está sendo solicitado para executar operações que foram consideradas, inicialmente, como não sendo responsabilidades dele.

Ou seja, essa comparação deve promover uma reavaliação das responsabilidades de cada objeto.

As colaborações e suas respectivas frequências são utilizadas na construção dos subsistemas.

Diagrama de Estado

Diagrama de estado é utilizado para descrever o comportamento que um objeto apresenta desde sua instanciação até sua destruição. Para cada classe de objeto que possui um comportamento não trivial é construído um diagrama de estado. O diagrama de estado utilizado neste método é o mesmo apresentado na Seção 2.2.2.

O comportamento de um objeto é:

- definido a partir das operações que ele é capaz de realizar [WIRF90a], e
- modelado em termos de estados e eventos [RUMB91].

Diagramas de interação são utilizados na construção de diagramas de estado uma vez que a partir desses é possível obter todas as operações que cada objeto deve ser responsável por realizar (primeiro item listado acima). Estados e eventos são definidos da seguinte forma:

- O *estado* de um objeto é definido pelos valores que seus atributos podem assumir e corresponde ao intervalo de tempo que existe entre dois eventos recebidos por ele;
- Um *evento* recebido por um objeto pode ser de dois tipos:
 1. uma solicitação de colaboração que o objeto recebe de outro objeto, ou
 2. uma resposta que o objeto recebe como resultado de uma colaboração que ele solicitou a outro objeto.

A construção do diagrama de estado de uma classe de objeto, envolve, basicamente, dois passos:

1. Identificar todos os diagramas de interação dos quais a classe participa, e
2. Identificar e organizar cronologicamente todos estados e eventos relacionados com a classe.

O segundo passo corresponde às atividades de construção do diagrama de estado apresentado na Seção 2.2.2 com a única diferença que, ao invés de ser derivado de cenários, o diagrama é derivado de diagramas de interação. Um exemplo de diagrama de estado é apresentado na Figura 5.16. Diagramas de estado são utilizados por ocasião da implementação das classes de objetos.

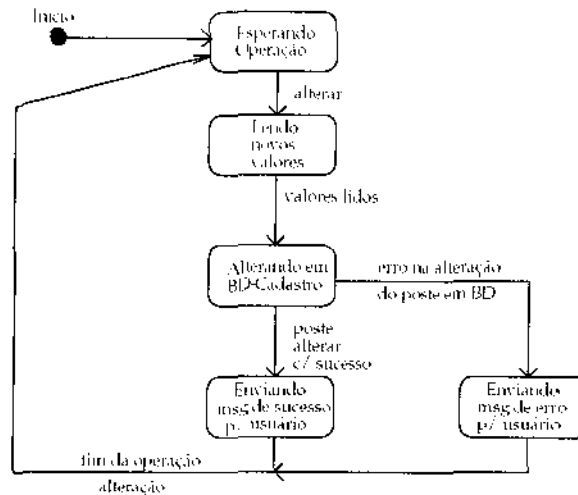


Figura 5.16: Diagrama de estado para a classe de objeto de controle POSTE.

5.1.1.4 Subsistemas

A construção de subsistemas compreende duas etapas:

- *Primeira Etapa:* agrupar em subsistemas classes de objetos que possuem uma grande afinidade entre si; e
- *Segunda Etapa:* avaliar a distribuição dos objetos entre os subsistemas, para verificar se a distribuição de objetos apresentada proporcionará um bom desempenho para a aplicação.

Primeira Etapa: Identificar subsistemas

A construção de subsistemas representa um primeiro passo na distribuição dos objetos componentes do SID. Subsistemas são formados por classes concretas que possuem uma grande afinidade entre si. A afinidade entre classes é medida a partir da análise das associações/agregações (modelo de objetos) e colaborações (modelo dinâmico) que existem entre as mesmas. Como resultado dessa análise são apresentadas duas regras a serem aplicadas na construção de subsistemas.

A ocorrência de uma associação entre duas classes A e B implica na existência de regras de integridade (Seção 2.1). Isto significa que quase todas as operações sobre uma instância de A (exceto a operação de consulta) implica na consulta de valores dos atributos da instância de B (assumindo que estas instâncias já foram associadas) a fim de que a consistência seja garantida. Por exemplo, considere a associação que existe entre as classes Poste e

Armário (Figura 5.12) e considere as instâncias *Poste1* e *Armário1*. Assumindo que *Poste1* e *Armário1* estão associados, antes de efetuar, por exemplo, uma alteração nos valores de seus atributos, *Poste1* deve consultar *Armário1* a fim de verificar a validade das alterações. O mesmo deve ocorrer com a operação de exclusão e associação. É interessante, então, que as classes *Poste* e *Armário* estejam localizadas num mesmo subsistema. A mesma idéia poder ser aplicada a todas as classes que possuam associações entre si.

A agregação representa um tipo especial de associação. Logo, classes que participam de uma agregação também devem estar presentes num mesmo subsistema. Assim a primeira regra que deve ser aplicada na construção de subsistemas é:

- **regra 1:** Classes que possuem associações/agregações entre si devem estar presentes num mesmo subsistema.

Aplicando esta regra nas classes identificadas para a aplicação do Capítulo 6, tem-se os seguintes subsistemas:

- *Subsistema 1:* formado pelas subclasses entidade de Elemento de Rede que compõem o banco de dados do cadastro;
- *Subsistema 2:* formado pelas subclasses entidade de Elemento de Rede que compõem o banco de dados de projeto;
- *Subsistema 3:* formado pelas classes de interface Janela Principal de Cadastro, Menu Principal de Cadastro, Menu Específico de Cadastro e Área de Trabalho de Cadastro.
- *Subsistema 4:* formado pelas classes de interface Janela Principal de Projeto, Menu Principal de Projeto, Menu Específico de Projeto e Área de Trabalho de Projeto.

A segunda regra a ser aplicada na construção de subsistemas está baseada nas colaborações que existem entre os objetos descritas nos diagramas de interação. A fim de garantir a execução de suas responsabilidades, um objeto pode solicitar colaborações de um ou mais objetos. Neste sentido, a necessidade do objeto *A* estar próximo do objeto *B* depende da quantidade de colaborações que *A* precisa de *B* para garantir suas responsabilidades, isto é:

- **regra 2:** Dado dois objetos *A* e *B*, *A* deve ficar no mesmo subsistema de *B* se *B* atende a $N\%$ ($N > 0$) de todas as colaborações que *A* necessita para garantir suas responsabilidades.

O valor de *N* deverá ser estabelecido pelos analistas. Quanto maior o valor de *N*, maior será o acoplamento funcional entre os objetos componentes do subsistema. Porém *N* não deve ser 100 pois isto implica que todas colaborações de um objeto *A* devem ser atendidas por apenas um único objeto *B*.

Para ilustrar a aplicação desta regra, na aplicação descrita no Capítulo 6, o objeto *Poste Cadastro* (subclasse de *Poste Controle*) necessita da colaboração dos seguintes objetos: *Área de Trabalho Cadastro*, *Caixa de Diálogo* e *Poste* (subclasse de *Elemento de Rede*). Cada um destes objetos atende às colaborações que *Poste Cadastro* necessita na seguinte proporção:

Tabela 5.1: Porcentagem de atendimento das colaborações que o objeto Poste Cadastro necessita feito pelos demais objetos.

Objetos	%
Área de Trabalho Cadastro	28,6
Caixa de Diálogo	28,6
Poste	42,8

Se N fosse igual a 40%, o objeto Poste Cadastro deveria ficar no mesmo subsistema do objeto Poste. Uma aplicação exata desta regra sobre as colaborações entre os objetos identificados no Capítulo 6 não é possível uma vez que não são apresentados todos os possíveis casos de uso que devem existir. No entanto, para efeito de ilustração, considere a seguinte distribuição como resultado da aplicação da regra 2:

- *Subsistema 1:* as classes de controle Poste Cadastro, Armário Cadastro, Lance de Cabo Cadastro, etc são incorporadas a este subsistema;
- *Subsistema 2:* as classes de controle Poste Projeto, Armário Projeto, Lance de Cabo Projeto, etc. são incorporadas a este subsistema;
- *Subsistema 3:* as classes Controle Cadastro (controle) **Caixa de Diálogo** (interface) são incorporadas a este subsistema;
- *Subsistema 4:* as classes Controle Projeto (controle) **Caixa de Diálogo** (interface) são incorporadas a este subsistema;

Note que a classe Caixa de Diálogo está participando de dois subsistemas ao mesmo tempo: subsistemas 3 e 4. Porém este tipo de construção deve gerar problemas por ocasião de eventuais atividades de manutenção. Para ilustrar esta situação, considere que apenas a definição da classe Caixa de Diálogo, participando do subsistema 3, deva sofrer alterações. O que se tem, então, é uma definição de uma classe com duas implementações diferentes: uma no subsistema 3 e outra no subsistema 4. Do ponto de vista de manutenção, esta situação é algo indesejável, uma vez que deve atrapalhar futuras manutenções sobre esta classe.

Uma solução para evitar que uma mesma classe participe de mais de um subsistema é a seguinte:

- Se uma classe participa de K ($K > 1$) subsistemas, especialize esta classe K vezes e atribua cada especialização a um subsistema.

No caso da classe Caixa de Diálogo, deve-se especializá-la duas vezes (Figura 5.17): Caixa de Diálogo Cadastro (subsistema 3) e Caixa de Diálogo Projeto (subsistema 4).

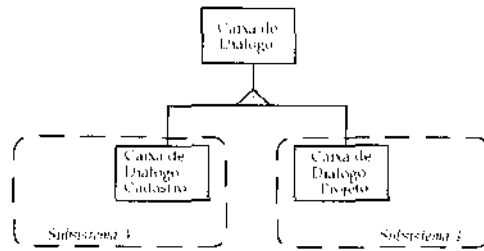


Figura 5.17: Distribuindo a definição de uma mesma classe entre subsistemas.

A aplicação das duas regras apresentadas afeta as classes do SID da seguinte forma:

- a primeira regra promove a distribuição das classes entidades em subsistemas: associações/agregações ocorrem, normalmente, entre classes entidades, as quais devem estar agrupadas, em virtude da grande quantidade de colaborações que existe entre as mesmas classes, necessárias para garantir as regras de integridade.
- a segunda regra promove a distribuição das classes de controle em subsistemas: objetos de controle normalmente não apresentam associações/agregações. São objetos que possuem um processamento próprio, não mantendo nenhum relacionamento de associação/agregação com outras classes. Para distribuir tais objetos em subsistemas é necessário, então, fazer uma análise de suas colaborações..

Classes de interface são distribuídas tanto a partir da primeira regra quanto da segunda regra. Em ordem de prioridade, na construção de subsistemas, deve-se primeiro considerar as associações/agregações que existem entre as classes e em seguida as colaborações que estas mantêm entre si, isto é, a primeira regra tem prioridade sobre a segunda.

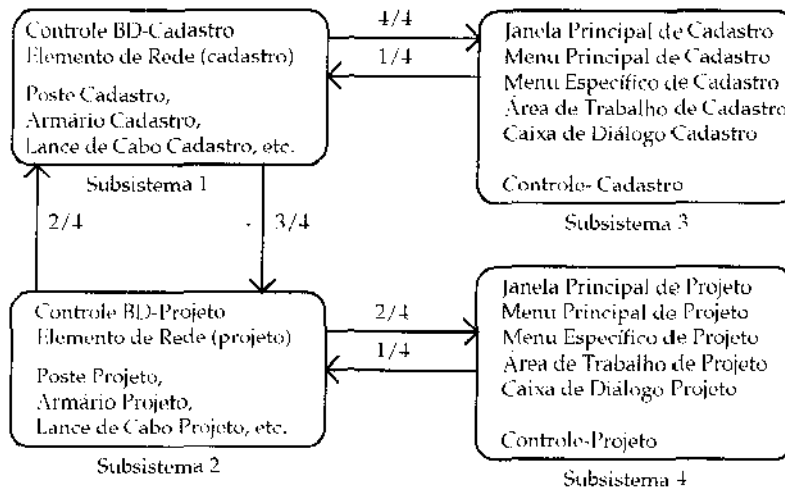


Figura 5.18: Diagrama de dependências.

Os subsistemas identificados para a aplicação do Capítulo 6 estão esquematizados no diagrama de dependências da Figura 5.18. As flechas entre os subsistemas indicam a existência de dependências entre os mesmos. Uma dependência entre dois subsistemas ocorre quando classes de objetos de um subsistema necessitam de uma ou mais colaborações de

uma ou mais classes de objeto localizadas em outro subsistema. A flecha vai do subsistema que precisa de colaborações para o subsistema que oferece as colaborações. A cada uma destas flechas é associada uma frequência determinada pela maior frequência de todas as colaborações que representam.

Segunda Etapa: Avaliar a distribuição dos objetos

A avaliação da distribuição dos objetos em subsistemas, conduzida a partir do diagrama de dependências, tem por objetivo verificar se a aplicação irá apresentar um bom desempenho. Um bom desempenho, neste contexto, está relacionado com a garantia da existência de dependências entre subsistemas que tenham uma baixa frequência.

Neste sentido a métrica apresentada a seguir propõe que, se um objeto interage solicitando colaborações a algum outro objeto que esteja fora do subsistema a que pertence, a frequência desta interação deve ser menor que todas as frequências de suas demais solicitações. Se isto não ocorrer, então o objeto deve ser remanejado para o subsistema em que suas solicitações de colaborações ocorrem com maior frequência.

Métrica

Considere dois objetos *A* e *B* onde *A* pertence ao subsistema 1 e *B* ao 2. Considere também *CI* como sendo uma colaboração que *A* solicita a *B*. Se a frequência com que *CI* ocorre for maior do que as demais frequências com que *A* solicita colaborações de outros objetos (diferentes de *B*), então *A* deve ser transferido para o subsistema 2.

Esta métrica deve ser aplicada a partir do diagrama de dependências da seguinte forma: para cada dependência que existe, aplica-se a métrica. Para ilustrar a aplicação desta métrica, considere a situação do objeto de controle Poste Cadastro ilustrada na Tabela 5.1. Pela regra 2 (N=40%), este objeto deveria ficar no subsistema 1. No entanto, a frequência com que o objeto Poste Cadastro solicita a colaboração do objeto Caixa de Diálogo Cadastro (ou do objeto Área de Trabalho Cadastro) é maior do que a frequência com que ele solicita a colaboração do objeto Poste. Portanto, o objeto Poste Cadastro deve ser transferido para o subsistema 3.

O objetivo desta métrica é garantir que as colaborações que devem existir entre subsistemas ocorram com uma frequência menor do que as colaborações que ocorrem dentro dos subsistemas.

Uma vez minimizadas as frequências das dependências intra-subsistemas, os analistas podem identificar quais subsistemas devem ficar próximos uns dos outros através de uma árvore hierárquica de agregação: dois ou mais subsistemas que devem estar localizados próximos uns dos outros são agrupados em torno de um subsistema abstrato (Diagrama de Proximidade). Por exemplo, o subsistema 1 deve estar próximo do subsistema 3 uma vez a frequência das dependências que existem entre os mesmos é alta quando comparada com as demais dependências em que cada um está envolvido (Figura 5.19).

Pode-se ainda agrupar subsistemas abstratos em torno de subsistemas abstratos. Porém tal tarefa não é tão simples de realizar uma vez que envolve analisar as eventuais dependências que devem existir entre os subsistemas abstratos.

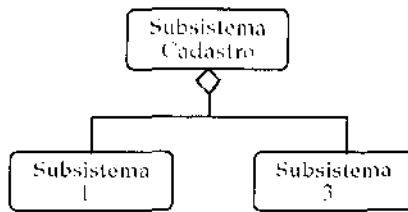


Figura 5.19: Diagrama de Proximidade..

5.1.2 Fase de Projeto

As atividades que devem ser executadas nesta fase seguem o esquema apresentado na Figura 5.20:

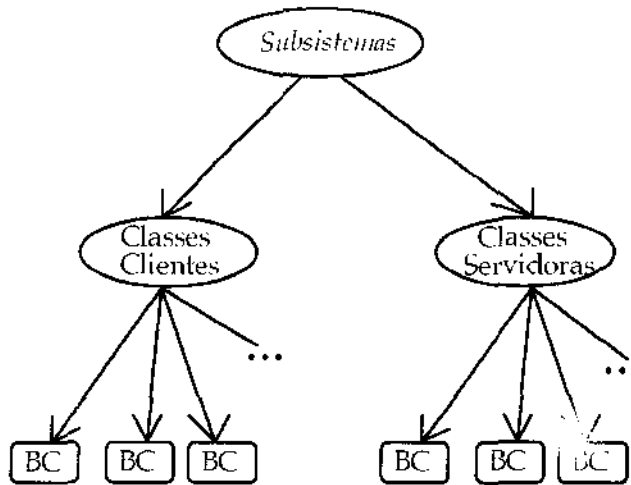


Figura 5.20: Atividade envolvidas na fase de projeto.

O ambiente computacional distribuído alvo deste método é baseado no paradigma cliente/servidor. Assim sendo, a primeira atividade a ser realizada durante a fase de projeto é a identificação de quais classes de objetos ficam em máquinas clientes e quais ficam em máquinas servidoras. Em seguida, blocos de construção devem ser projetados e distribuídos entre as máquinas disponibilizadas para a implementação da aplicação, relacionadas na especificação de requisitos.

5.1.2.1 Classes Clientes e Classes Servidoras

Um subsistema pode ser formado por:

- apenas classes clientes;
- apenas classes servidoras; ou
- classes clientes e classes servidoras.

A classificação das classes em clientes e servidoras deve ser feita no nível de subsistemas. De imediato é possível verificar que classes de interface são classes clientes e classes entidade são classes servidoras. As classes de controle podem ser tanto clientes como servidoras, dependendo da afinidade que estas possuem com as classes de interface e as classes entidade (veja regra 1 e 2 - construção de subsistemas).

Da maneira como os subsistemas foram identificados na fase anterior, a grande maioria deve ser, inevitavelmente, cliente ou servidor. A terceira formação listada acima (subsistema formado por classes clientes e classes servidoras) é menos comum, uma vez que implica a existência de grande afinidade entre classes de interface e classes entidade. No desenvolvimento de SIDs, normalmente as atividades entre esses dois tipos de classes devem ser intermediadas por classes de controle [JACO92].

As classes que formam os subsistemas identificados para o exemplo do Capítulo 6 são classificadas da seguinte maneira (Figura 5.21):

- *Clientes*: todas as classes dos subsistemas 1 e 2
- *Servidoras*: todas as classes dos subsistemas 3 e 4

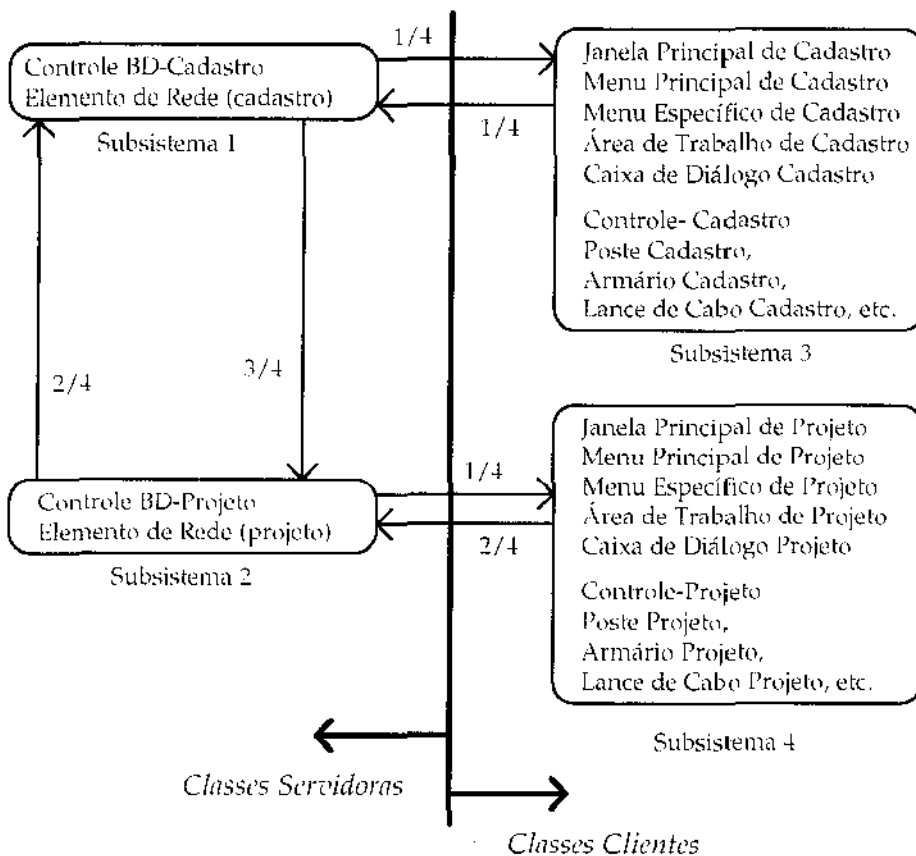


Figura 5.21: Classificação das classes em clientes e servidoras.

5.1.2.2 Blocos de Construção

Depois de terem sido classificadas em clientes e servidoras, as classes de cada subsistema são agrupadas em blocos de construção. Um bloco de construção é formado por um grupo de classes de mesmo tipo (interface, controle ou entidade) e mesma localização (cliente ou servidor) que juntas oferecem um conjunto de funcionalidades coerentes (e.g., um grupo de classes de interface que oferecem funcionalidades de manipulação de janelas e menus). Blocos de construção são classificados em:

- Blocos de construção de dados (BCD): formados por classes entidade;
- Blocos de construção de processamento (BCP): formados por classes de controle; e
- Blocos de construção de usuário (BCU): formados por classes de interface.

Os blocos de construção para os subsistemas identificados para o exemplo do Capítulo 6 são os seguintes (Figura 5.22):

Subsistema 1:

- **BCD1:** formado pelas subclasses entidade de Elemento de Rede que compõem o banco de dados de cadastro e pela classe de controle Controle BD-Cadastro;

Subsistema 2:

- **BCD2:** formado pelas subclasses entidade de Elemento de Rede que compõem o banco de dados de projeto e pela classe de controle Controle BD-Projeto;

Subsistema 3:

- **BCU1:** formado pelas classes de interface Janela Principal de Cadastro, Menu Principal de Cadastro, Menu Específico de Cadastro, Área de Trabalho de Cadastro e Caixa de Diálogo Cadastro;
- **BCP1:** formado pelas classes de controle Controle-Cadastro, Poste Cadastro, Armário Cadastro, Lance de Cabo Cadastro, etc.;

Subsistema 4:

- **BCU2:** formado pelas classes de interface Janela Principal de Projeto, Menu Principal de Projeto, Menu Específico de Projeto, Área de Trabalho de Projeto e Caixa de Diálogo Projeto;
- **BCP2:** formado pelas classes de controle Controle-Projeto, Poste Projeto, Armário Projeto, Lance de Cabo Projeto, etc.

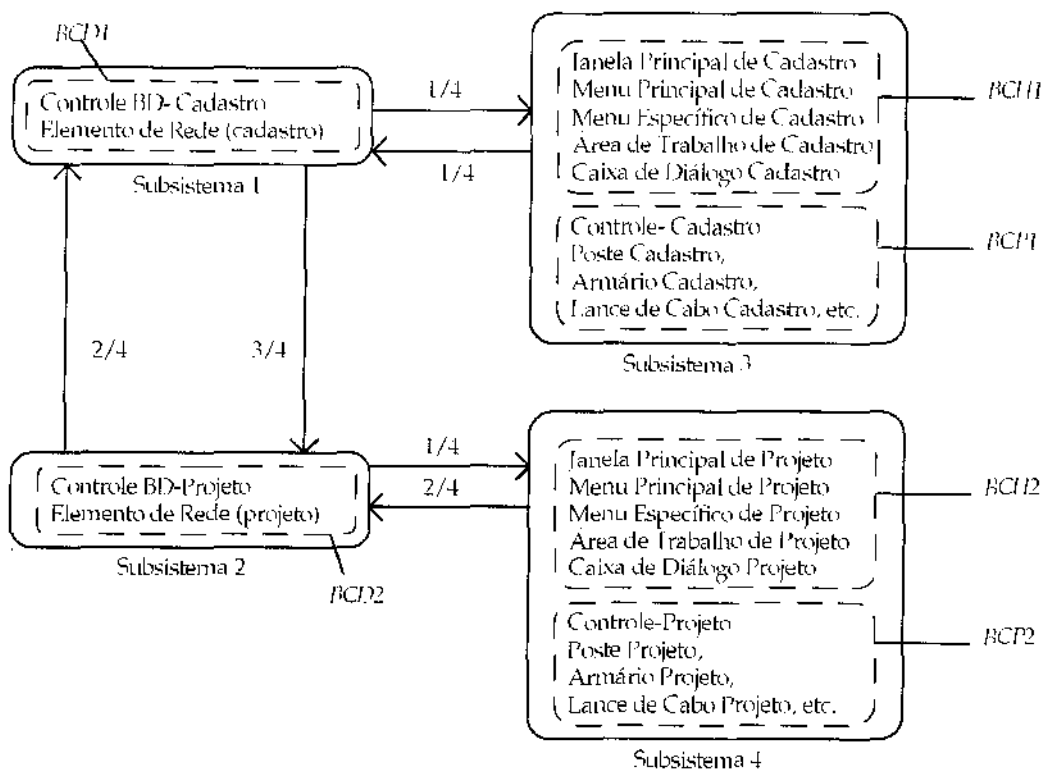


Figura 5.22: Agrupamento das classes em blocos de construção.

As interações entre os blocos de construção ocorrem através de interfaces bem definidas chamadas de contratos (Seção 3.3.6 - Contratos). A fim de tornar mais consistente a formação de um bloco de construção, pode ser feita uma simplificação nas colaborações que existirem entre as classes componentes do mesmo. Para tanto as seguintes medidas devem ser tomadas:

Minimizar o número de colaborações entre blocos de construção

Dentro de um bloco de construção tentar fazer com que apenas uma classe seja responsável pelo controle das atividades internas. Dessa maneira, clientes deste bloco de construção terão que solicitar seus pedidos apenas para essa classe principal. Isto deve reduzir o número de colaborações que são necessárias entre blocos de construção, além de promover um melhor encapsulamento das atividades desenvolvidas pelos mesmos.

Minimizar o número de diferentes contratos suportados por um bloco de construção

Evitar que um bloco de construção suporte muitos contratos. Isto pode ser um sinal de que a inteligência da aplicação⁵ está muito concentrada num mesmo lugar. Deve-

⁵A inteligência de uma aplicação é representada pelas informações que esta mantém por meio dos atributos de seus objetos (Seção 2.3.1.2).

se analisar bem o bloco de construção em questão e verificar se ele não poderia ser subdividido em blocos de construção menores com menos contratos.

A primeira medida listada acima nem sempre se aplica, uma vez que é possível que um bloco de construção possa ser formado por classes que não mantêm nenhum tipo de colaboração entre si. Isto deve-se ao fato de que o projeto de blocos de construção é feito a partir do tipo das classes e das funcionalidades que estas oferecem, ao contrário de subsistemas que são construídos a partir da forma como as classes se relacionam (associações e colaborações).

Na segunda medida, o que se propõe é a que inteligência do SID seja distribuída da forma mais uniforme possível. Blocos de construção com muitas responsabilidades (contratos) são normalmente blocos com muita inteligência. Deve-se evitar ter poucos blocos de construção com muita inteligência, e muitos com pouca. No entanto, uma distribuição perfeitamente uniforme é impossível, pois, dependendo do papel que as classes componentes de um bloco de construção possuem dentro da aplicação, alguns blocos de construção devem possuir necessariamente mais inteligência que outros.

Nos blocos de construção listados anteriormente, a primeira medida poderia ser aplicada sobre o BCD1 e BCD2. Uma simplificação nas colaborações entre as classes componentes dos mesmos poderia ser feita no sentido de tornar mais eficiente a execução das regras de integridade. Quanto a segunda medida, um exemplo de aplicação da mesma requer maior e mais aprofundado conhecimento sobre todas as atividades que o SID deve apresentar.

5.1.2.3 Distribuindo os Blocos de Construção

Uma vez identificados todos os blocos de construção que devem implementar o SID, o próximo passo é propor uma primeira distribuição dos mesmos sobre as máquinas disponibilizadas. Nessa distribuição os seguintes aspectos devem ser observados:

Identificar quais serão as máquinas clientes e máquinas servidoras

Identificar quais serão as máquinas que estarão disponíveis para a implementação do SID e qual será a provável localização dessas máquinas. Essas informações podem ser obtidas a partir da especificação de requisitos.

Distribuir blocos de construção entre máquinas clientes e máquinas servidoras

Blocos de construção que participam de um mesmo subsistema devem ser alocados em máquinas que estejam próximas umas das outras. Se um subsistema for composto de apenas classes clientes ou classes servidoras, os blocos de construção que o formam devem ser alocados todos numa mesma máquina. Verificar também a árvore hierárquica de subsistemas (Figura 5.19). Por exemplo os subsistemas 1 e 3 devem ser alocados em máquinas próximas umas das outras uma vez que devem existir muitas interações entre os blocos construção desses subsistemas a fim de garantir a execução da funções do módulo de cadastro.

Algumas características que devem ser contempladas pelos BCs clientes e servidores são as seguintes:

- BCs clientes podem ter suas definições repetidas em mais de uma máquina cliente;
- BCs clientes não podem desempenhar o papel de servidores ao contrário do que é proposto pela arquitetura OSCA apresentada no Capítulo 3, onde um BC pode ser tanto um cliente como um servidor;
- BCs servidores não devem ter suas definições repetidas em mais de uma máquina servidora: isto implicaria na replicação de informações;
- BCs servidores podem desempenhar tanto papel de servidores como de clientes (quando existem trocas de mensagens entre servidores).

Blocos de construção representam a unidade de implementação e manutenção do SID, conforme será apresentado na descrição da fase de implementação.

5.1.3 Fase de Implementação

Esta fase envolve, basicamente, a implementação de cada um dos blocos de construção a qual poderá ser feita de duas formas:

- codificar, partindo do zero, todas as funcionalidades dos blocos de construção; ou
- utilizar recursos já existentes (na empresa) na implementação dos blocos de construção (por exemplo, para um BDG pode-se utilizar um SGBD já implementado).

De maneira geral, o processo de implementação deve ser conduzido como uma combinação dessas duas formas. Para auxiliar nesse processo devem ser utilizados os diagramas de estado construídos para cada classe durante a fase de análise.

Uma vez que os contratos escondem os detalhes de funcionamento das funcionalidades oferecidas por cada bloco de construção, a implementação deste pode ser conduzida de maneira concorrente e proprietária (isto é, cada um pode ser desenvolvido com, por exemplo, um tipo de linguagem diferente). Blocos de construção são independentes em termos de instalação e execução. Cada um gerencia o seu próprio estado. Sendo assim, um bloco de construção pode ser trocado, atualizado ou realocado sem qualquer impacto para o resto do sistema, desde que seus contratos sejam preservados.

Como resultado desta possível heterogeneidade na implementação da aplicação distribuída, é necessário a utilização de algum middleware que garanta a comunicação entre os blocos de construção. Uma proposta neste sentido é o middleware CORBA do OMG. Para tanto será necessário que os contratos de cada bloco de construção sejam mapeados para IDL. É sugerido também que esses blocos de construção sejam organizados em três camadas (dados, processamento e usuário) conforme proposto pela arquitetura OSCA (veja Figura 5.23).

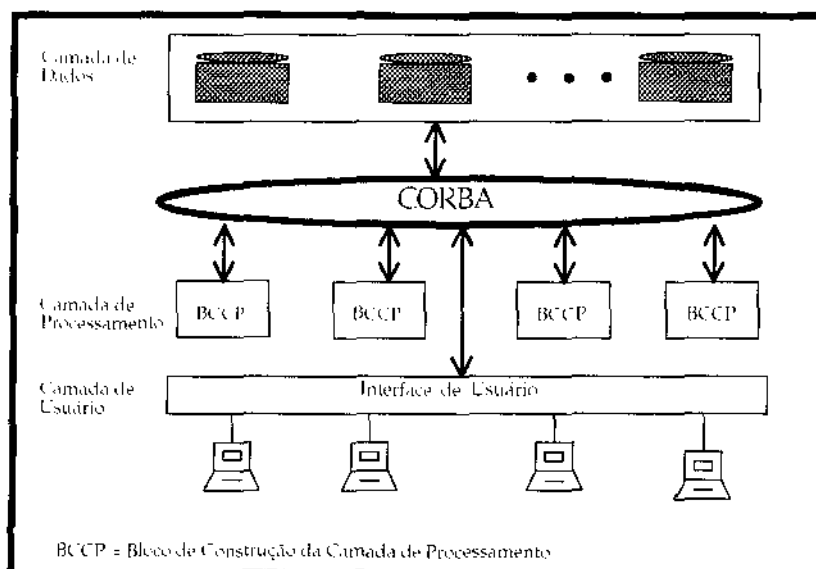


Figura 5.23: Esquema de arquitetura que suporte a implementação SIDs.

5.2 Proposta de uma Ferramenta CASE

Nesta seção é apresentada uma proposta de ferramenta CASE para automatizar o processo de agrupamento de objetos em subsistemas apresentado na seção anterior. A ferramenta é composta pelos seguintes módulos (Figura 5.24):

- *Enciclopédia:* local onde são armazenadas todas as informações manipuladas pela ferramenta.
- *Diagramador de Objetos:* permite a construção do modelo de objetos. Através deste módulo são inseridas as seguintes informações na enciclopédia:
 - Classes com seus atributos e responsabilidades,
 - Associações e agregações existentes entre as classes, e
 - Generalizações.
- *Diagramador de Interação:* permite a construção de diagramas de interação (modelo dinâmico). Através deste módulo são inseridas as seguintes informações na enciclopédia:
 - colaborações que existem entre as classes cadastradas a partir do diagramador de objetos, e
 - frequências das colaborações (obtidas a partir do modelo de casos de uso)
- *Diagramador de Estado:* permite a construção de diagramas de estado (modelo dinâmico). Através deste módulo são inseridas as seguintes informações na enciclopédia:
 - estados (das classes cadastradas a partir do diagramador de objetos) e

- eventos (derivados das colaborações cadastradas a partir do diagramador de interação).
- *Matriz Colaboração*: permite a construção dos subsistemas. A partir deste módulo são gerados dois diagramas:
 - Diagrama de Dependência
 - Diagrama de Proximidade

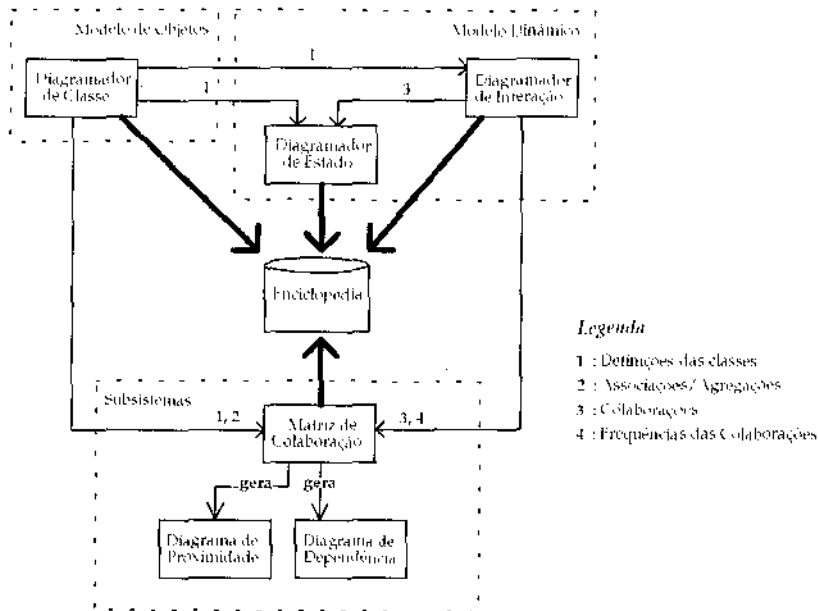


Figura 5.24: Esquema dos componentes da ferramenta proposta.

A construção do modelo de casos de uso pode ser feita a partir de um editor de texto qualquer.

Hoje em dia já existem no mercado ferramentas CASE que implementam os diagramadores listados acima (a única exceção é o diagramador de interação onde, além de informar quais são as colaborações entre os objetos, é preciso também fornecer a frequência com que as mesmas ocorrem). Exemplos destas ferramentas são:

- Rational Rose [RATI96]: diagramadores de objetos, de interação e de estado.
- Software through Pictures (StP) [IDE96]: diagramadores de objetos e de estado.
- Objectory [JACO92]: diagramadores de interação⁶.

Todas essas ferramentas estão baseadas na existência de algum tipo de enciclopédia. Porém nenhuma delas apresenta um mecanismo de agrupar objetos em subsistemas. A seção a seguir mostra como este mecanismo pode ser implementado.

⁶Objectory também possui diagramadores de objetos e de estado. Porém estes utilizam uma notação diferente daquela utilizada pelo método proposto: este método utiliza a notação OMT enquanto que Objectory utiliza a notação proposta por [JACO92]. A ferramenta StP utiliza a notação OMT e a ferramenta Rational Rose utiliza as notações de [RUMB91] e [JACO92].

5.2.1 Matriz de Colaboração

A matriz de colaboração é uma matriz Conjunto de Classes X Conjunto de Classes onde Conjunto de Classes é formado por todas as classes de objetos cadastradas a partir do diagramador de objetos. As células desta matriz são preenchidas da seguinte forma:

- para cada classe de objeto listada nas linhas da matriz, identificar com que porcentagem os objetos listados nas colunas da matriz atendem às suas colaborações. Por exemplo, se o objeto 1 (*ob1*) necessita de oito colaborações e o objeto 2 (*ob2*) atende a duas destas colaborações, então *ob2* atende a 25% das colaborações que *ob1* necessita (Figura 5.25). A soma das porcentagens de cada linha da matriz deve ser sempre igual a 100%.

	ob1	ob2	ob3	ob4
ob1		25%	15%	60%
ob2	20%		80%	—
ob3	10%	80%		10%
ob4	—	100%	—	

Figura 5.25: Exemplo de Matriz de Colaboração

Esta matriz deve ser gerada automaticamente pelo módulo a partir das colaborações cadastradas no diagramador de interação (modelo dinâmico). O processo de construção de subsistemas implementado por este módulo é composto por duas etapas (Figura 5.26). Uma descrição de qual é a entrada, o processo e o resultado gerado por cada etapa é apresentada a seguir. Essas etapas são derivadas do processo de construção de subsistemas descrito na Seção 5.1.1.4:

Primeira etapa:

Entrada: associações/agregações (modelo de objetos);
colaborações (matriz de colaboração).

Processo: aplicar a regra 1 sobre as associações/agregações;
aplicar a regra 2 sobre as colaborações.

Resultado: Lista de subsistemas;
Diagrama de Dependências.

Segunda Etapa:

Entrada: Lista de subsistemas;
Diagrama de Dependências;
Frequência das colaborações (modelo dinâmico).

Processo: para cada objeto que gera uma dependência entre subsistemas, aplicar a métrica. No fim do processo indicar quais subsistemas devem ficar próximos uns dos outros.

Resultado: frequências das dependências entre subsistemas minimizadas;
Lista de subsistemas atualizada (esta lista se faz necessária uma vez que algum objeto pode ter sido transferido para algum outro subsistema durante a execução do processo desta etapa);
Diagrama de Proximidade;

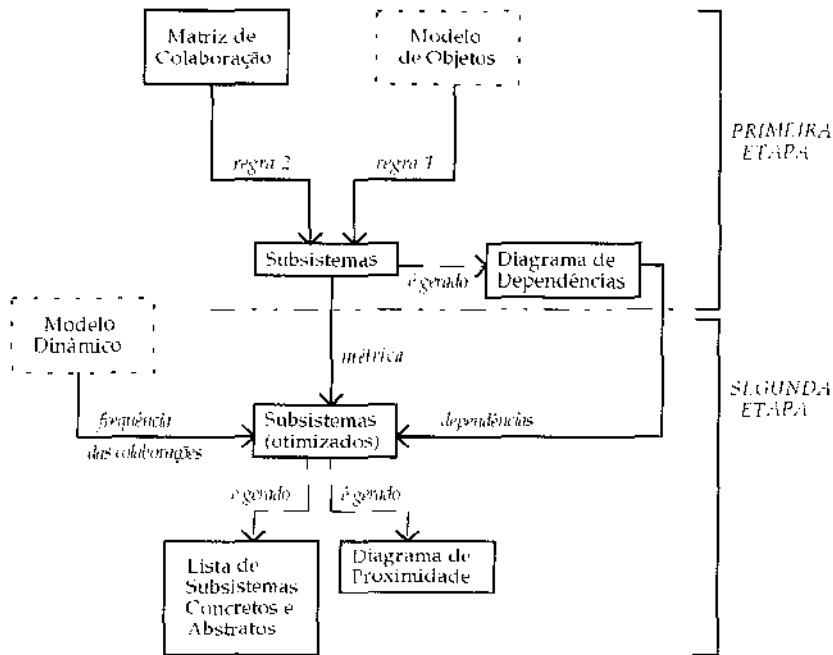


Figura 5.26: Esquema do processo de construção implementado pelo módulo Matriz de Colaboração.

Associado à regra 2 aplicada no processo descrito acima o usuário deve especificar o seguinte parâmetro:

- *regra 2:* informar o valor de N ($0 < N < 100$).

Não existe nenhum parâmetro para a regra 1. Na construção de subsistemas o módulo deve resolver a seguinte situação crítica:

- na primeira etapa, classes de objetos participando de mais de um subsistema: aplicar a solução apresentada na Seção 5.1.1.4.

O módulo gera os seguintes resultados:

- Lista com todos os subsistemas gerados
- Diagrama de dependências
- Diagrama de proximidade

5.3 Conclusão

Neste capítulo foi apresentado um método orientado a objetos de desenvolvimento de sistemas de informação distribuído composto por três fases: análise, projeto e implementação. A principal inovação que este método trás quando comparado com os métodos apresentados no Capítulo 2 é uma construção mais elaborada de subsistemas. Nos outros métodos, subsistemas são utilizados mais como uma forma de facilitar o entendimento funcional da aplicação (Casos de Uso e Projeto Orientado a Responsabilidade) ou facilitar a implementação da mesma (Técnica de Modelagem de Objetos) não apresentando nenhuma influência maior no processo de desenvolvimento. Neste método, o conceito de subsistema representa um ponto chave no desenvolvimento do SID.

A construção de subsistemas é feita a partir da afinidade que existe entre objetos a qual é medida em função de duas regras. A fim de verificar se a distribuição de objetos apresentada proporcionará um bom desempenho para o SID, é proposta uma métrica.

Para automatizar este processo de agrupamento de objetos em subsistemas, foi apresentada uma ferramenta que estende as ferramentas implementadas pelos programas ADW e IEF (Capítulo 4) em dois sentidos. Em primeiro lugar, esta ferramenta foi concebida sob o paradigma de orientação a objetos enquanto que as outras ferramentas são baseadas na decomposição funcional. As vantagens que existem no paradigma de orientação a objetos quando comparado com a decomposição funcional já foram amplamente discutidas na literatura ([JACO92], [RUMB91]).

Em segundo lugar, a ferramenta proposta executa uma avaliação da distribuição dos objetos a fim de verificar se o SID deverá apresentar um bom desempenho. As ferramentas apresentadas no Capítulo 4 se restringem a apresentar o agrupamento dos processos e entidades de dados em sistemas de informação e banco de dados respectivamente.

Um resumo do método proposto é apresentado a seguir:

Fase de Análise

Modelo de Casos de Uso Estendido

- construção do Modelo de Casos de Uso
- identificação da frequência com que cada caso de uso ocorre

Modelo de Objetos

- classificação das classes/objetos em interface, controle e entidade
- identificação das responsabilidades de cada classe
- identificação das associações/agregações entre as classes
- organização das classes em uma hierarquia de generalização (herança)

Modelo Dinâmico

Diagrama de Interação

- Identificação das colaborações que existem entre os objetos
- Atribuição das frequências dos casos de uso às colaborações

Diagrama de Estado

- Modelagem do comportamento de cada classe de objeto em termos de estados e eventos

Construção de Subsistemas (distribuição dos objetos)

- *regra 1*: distribui (agrupa) os objetos a partir das associações/agregações existentes
- *regra 2*: distribui os objetos a partir das colaborações existentes
- *métrica*: avalia a distribuição procurando minimizar as frequências das colaborações (dependências) que existem entre os subsistemas

Fase de Projeto

- Identificação de classes clientes e classes servidoras
- Identificação dos blocos de construção
- Distribuição dos blocos de construção entre as máquinas disponíveis

Fase de Implementação

- Implementação dos blocos de construção

Capítulo 6 - Exemplo de Aplicação do Método

Neste capítulo é apresentada uma aplicação do método descrito no capítulo anterior na construção de SID. O SID escolhido para esta aplicação tem por finalidade a automatização do processo de projeto realizado pelas empresas operadoras do Sistema TELEBRÁS ([POT96], [POTP96]).

Uma breve motivação para o desenvolvimento deste SID é apresentada. Em seguida são descritas algumas atividades que envolvem o processo de projeto. A partir dessas descrições é aplicado o método.

Não é objetivo desta seção apresentar uma solução completa do SID em questão. O que se pretende é apenas ilustrar como os conceitos envolvidos no método proposto podem ser aplicados.

6.1 Motivação e Descrição

Um projeto de rede externa envolve, basicamente, a expansão da rede telefônica a fim de atender à demanda de novos serviços (por exemplo, atender novos assinantes). Com o explosivo avanço tecnológico na área de telecomunicações observado nos últimos tempos (por exemplo, internet, TV a cabo, vídeo sob demanda, comunicação de dados de alta velocidade) a quantidade de projetos exigidos tem sido cada vez maior. Porém os projetistas não têm conseguido atender a essas demandas uma vez que ainda utilizam, como ferramentas de trabalho, basicamente, lápis e papel.

Atualmente, quando um projeto é solicitado, os projetistas devem, primeiramente, obter todas as plantas (em papel) que descrevem a área onde o projeto será conduzido. Apenas esse processo pode, às vezes, levar algumas semanas e até mesmo alguns meses. Também são levantados todos os formulários (em papel) contendo características técnicas dos componentes representados nas plantas.

A demora no levantamento dessas informações deve-se ao fato que essas plantas normalmente encontram-se espalhadas entre as diversas centrais telefônicas (CTs) que existem na cidade. Cada CT possui as plantas que descrevem a região da cidade pela qual é responsável. CTs são interligadas a fim de atender às chamadas telefônicas entre assinantes localizados em regiões diferentes. Chamadas interurbanas são garantidas pela estação telefônica (ET) da cidade a qual é ligada a todas as CTs. A Figura 6.1 ilustra esta situação.

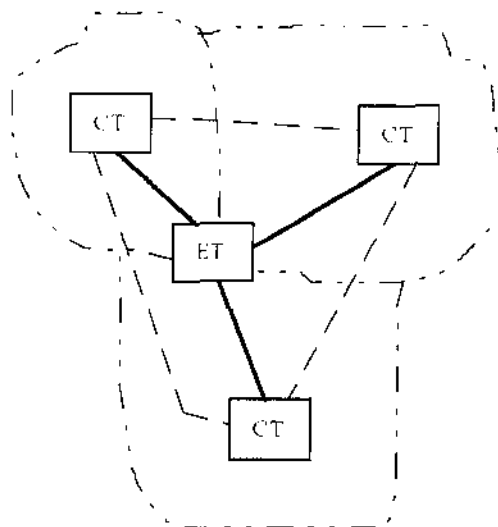


Figura 6.1: Esquema organizacional entre CTs e ET.

Porém, após terem levantado todos os dados necessários para o projeto, os projetistas correm o risco de estarem trabalhando com informações inconsistentes: é possível que um componente representado numa planta tenha sido alterado nesse meio tempo devido a algum problema (por exemplo, um poste foi trocado de lugar, ou até mesmo substituído por outro de material diferente - poste de concreto substituído por poste de madeira). Isso ocorre porque a atualização da planta envolve a correção direta no papel. E como esta estará em posse dos projetistas, a atualização não é feita.

Tendo obtido todas as informações necessárias (plantas e formulários), o projeto tem seu início. Algumas das atividades realizadas são as seguintes:

- **gerência do projeto:** esta atividade é dividida em duas etapas. A primeira, que já foi apresentada, envolve o levantamento das plantas e formulários necessários para o projeto. A segunda envolve o acompanhamento dos trabalhos do projeto;
- **detalhamento do projeto:** esta atividade representa o projeto propriamente dito. Aqui os novos componentes serão desenhados e associados aos outros componentes já existentes na rede telefônica. Para cada novo componente desenhado, também deverão ser preenchidos formulários com suas características técnicas. Componentes já existentes na rede também poderão ser atualizados;
- **implantação:** a última atividade do projeto é a implantação do mesmo. Aqui talvez ocorram alguns problemas em virtude das possíveis inconsistências dos dados utilizados durante o projeto. Atualmente esse problema é resolvido pela equipe que irá fazer a implantação: se existirem inconsistências (por exemplo, a utilização de um poste que não existe mais) a equipe deverá tentar resolver o problema (por exemplo, colocar o poste que falta). Isso pode trazer graves conseqüências como, por exemplo, onerar ainda mais o orçamento previsto para o projeto.

6.2 Identificação dos Requisitos

Nesta seção é apresentada primeiramente uma análise do tipo da informação que a aplicação deverá utilizar e em seguida quais serão os processos que serão aplicados sobre essas informações.

6.2.1 Informações Geográficas

A aplicação que será construída representa um Sistema de Informação Geográfica (SIG). Segundo [CCHMM96], "Sistemas de Informação Geográfica são sistemas automatizados usados para armazenar, analisar e manipular dados geográficos, ou seja, dados que representam objetos e fenômenos em que a localização geográfica é uma característica inerente à informação e indispensável para analisá-la." No caso da aplicação, existirão dois tipos de dados geográficos, ambos representados nas plantas utilizadas durante o projeto. Esses tipos são os seguintes:

- **mapa urbano da cidade:** representando as ruas e avenidas da mesma; e
- **rede externa** (ou rede telefônica): representando a malha de cabos por onde passam as linhas telefônicas. Nessa malha estão representados todos os componentes, chamados de *elementos de rede*, necessários para que as chamadas telefônicas sejam realizadas. Exemplos desses elementos são postes, lances de cabos, armários, etc.

Os dados geográficos contidos na rede externa são mapeados sobre o mapa urbano. A Figura 5.28 ilustra essa situação. Na Figura são representados quatro postes (p1, p2, p3 e p4) por onde está passando o lance de cabo L1. Esses elementos de rede estão localizados na *Av. Francisco Glicério*.

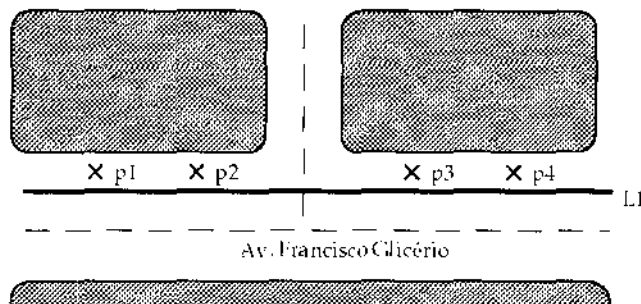


Figura 6.2: Dados geográficos contidos na planta: mapa urbano e rede externa.

Cada CT deverá ser responsável por manter as informações contidas nas plantas que envolvem sua região de atendimento. Em cada CT deverá existir um banco de dados "geográfico" (BDG) onde todas essas informações são armazenadas de forma contínua, isto é, os dados geográficos das plantas são armazenados de forma a representar uma única

planta. Na Figura 6.3, as plantas Pl.1, Pl.2, Pl.3 e Pl.4 representam a região de atendimento de uma dada CT as quais foram armazenadas de forma contínua no respectivo BDG.

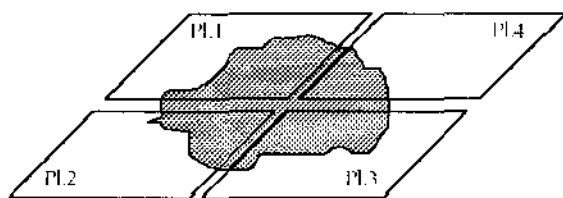


Figura 6.3: Dados geográficos das plantas representadas de forma contínua.

6.2.2 Módulos da Aplicação

A aplicação será composta por dois módulos: *módulo de projeto* e *módulo de cadastro*. O primeiro módulo deverá ser responsável pela automatização das três atividades descritas anteriormente e deverá funcionar na ET da cidade. O segundo módulo deverá permitir que as CTs façam a manutenção das informações contidas nos respectivos BDGs de forma a manter os dados de sua rede de atendimento sempre atualizados.

Apesar de não ser tratado aqui, essa aplicação deveria possuir ainda um terceiro módulo que seria responsável pela conversão dos dados geográficos contidos nas plantas para o BDG. Porém será assumido que esse processo já foi feito de alguma forma e que todos os BDGs já estão devidamente criados.

Módulo de Cadastro

Através do módulo de cadastro, as CTs serão capazes de manter sempre atualizadas as informações contidas nas plantas que envolvem a sua região de atendimento. O módulo deverá ser operado por pessoas que estejam ligadas diretamente com atividades de manutenção da rede externa. Quando alguma mudança é feita, essas pessoas devem ser notificadas para que façam as devidas atualizações. Essas atualizações podem representar uma inclusão, uma exclusão ou uma alteração de algum elemento de rede, ou nova associação entre dois ou mais elementos de rede (por exemplo, associação de um lance de cabo a vários postes). No entanto, antes de efetivar qualquer uma dessas quatro operações, o módulo verifica a validade da operação. Isso deverá garantir a consistência das informações contidas no BDG. Será oferecida também uma operação de consulta aos atributos de um elemento de rede.

Para cada tipo de elemento de rede deverá existir uma implementação própria de cada uma dessas cinco operações (inclusão, exclusão, alteração, associação e consulta). Isso deve-se ao fato de que cada elemento possui regras de integridades próprias as quais devem ser garantidas: a forma como deve ser feita, por exemplo, a exclusão de um poste é diferente da forma como deve ser feita a exclusão de um lance de cabo.

Módulo de Projeto

Uma descrição do funcionamento das atividades desse módulo é apresentada a seguir.

Gerência de Projeto

A gerência de projeto envolve a criação e o acompanhamento do projeto.

Criação de projeto: primeiro os projetistas devem identificar a CT cujo BDG possui as informações da área geográfica sobre a qual será realizado o projeto. Em seguida é feita uma extração dessa área e criado um banco de dados de projeto (BDP) com as informações extraídas (Figura 6.4). O projeto será conduzido sobre esse BDP. Poderá existir mais de um projeto em andamento, cada um com um BDP próprio.

Acompanhamento do projeto: o acompanhamento do projeto está relacionado com a garantia da consistência entre os dados que são comuns ao BDP e ao BDG de onde foi extraída a área geográfica. E a manutenção dessa garantia deverá ser responsabilidade do módulo de cadastro. No módulo de cadastro, os elementos de rede que fazem parte de algum projeto devem possuir um atributo que sinalize essa participação. Desta forma, sempre que houver alguma mudança nos valores dos atributos desses elementos, o módulo deverá informar o projeto correspondente sobre a alteração. Dependendo do tipo de alteração, os projetistas poderão ou não fazer a reextração da área geográfica. Uma alteração que poderia levar à reextração seria o caso em que o elemento de rede é associado a outros elementos. Neste caso é interessante que seja feito a reextração uma vez que essa nova associação pode resultar em algumas mudanças no projeto. Por outro lado, uma simples alteração no desenho de um elemento não implicaria na necessidade de uma reextração.

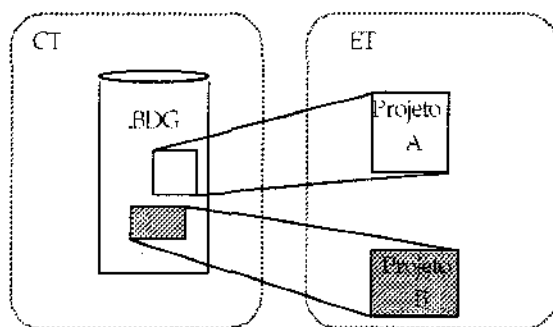


Figura 6.4: Processo de criação de projetos.

Detalhamento do Projeto

O detalhamento do projeto está relacionado com o desenho do projeto em si a partir da região geográfica extraída. Nessa atividade estarão disponíveis para os projetistas

as mesmas operações disponibilizadas pelo módulo de cadastro para a edição da rede externa. Essas operações incluem inserção, exclusão, alteração e associação de elementos de rede (projetados e existentes) além da operação de consulta.

Implantação do Projeto

A implantação do projeto envolve a integração do BDP com o respectivo BDG. No entanto, é possível que nessa integração algumas inconsistências ocorram (por exemplo, um armário de distribuição de lance de cabos que foi utilizado no projeto já atingiu sua capacidade máxima de ocupação e não poderá ser utilizado). Nesse caso, os projetistas terão que fazer uma reextração da área geográfica e fazer as devidas alterações.

Os elementos de rede que foram integrados no BDG irão assumir a situação de projetados e ficarão nessa situação até serem implantados, quando assumirão, então, a situação de existentes (os elementos de rede que já existiam no BDG possuem a situação de existentes).

Características comum aos Módulos

A interface dos módulos será formada por uma janela principal (JP) composta por três partes: *Menu Principal* (MP), *Menu Específico* (ME) e *Área de Trabalho* (AT). No MP serão oferecidas, basicamente, as seguintes opções: *abrir banco de dados*, *fechar banco de dados*, *selecionar um tipo de elemento de rede* e *sair do módulo*. No caso do módulo de projeto existirá ainda uma opção de *criação de projeto*.

A opção *selecionar um tipo de elemento de rede* será necessária para a situação em que se deseja incluir um determinado tipo de elemento de rede. Esta seleção se faz necessária pois, como apontado na descrição do módulo de cadastro, cada elemento de rede possui características próprias que devem ser atendidas. Estas características deverão ser atendidas a partir das operações disponibilizadas no ME. O ME deverá oferecer as cinco opções (inclusão, exclusão, alteração, associação e consulta) específicas ao tipo de elemento selecionado.

Um elemento também poderá ser selecionado a partir da AT. Na AT será carregada a representação da rede externa contida no banco de dados aberto (BDG ou BDP). Neste caso, para o usuário selecionar o elemento, basta clicar (com a ajuda do mouse) sobre o mesmo.

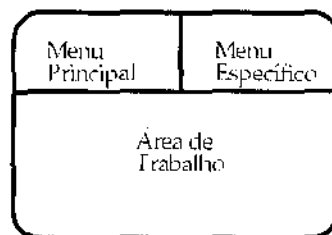


Figura 6.5: Esquema da janela principal dos módulos de cadastro e projeto.

Na interface dos módulos existirá ainda uma caixa de diálogo (CD) através da qual os módulos poderão enviar mensagens para o usuário e solicitar-lhe a entrada de dados.

6.3 Aplicação do Método

6.3.1 Fase de Análise

Modelo de Casos de Uso

Primeira Etapa - Construção do modelo de casos de uso

Atores: *projetistas e usuários do cadastro (UC).*

Caso de uso 1: Criação de Projeto

Descrição:

Para criar um projeto, o projetista deve selecionar no MP a opção correspondente. Uma CD é, então, aberta onde são listados todos os BDGs existentes. Uma vez selecionado o BDG desejado, o projetista deverá fornecer a senha de acesso ao mesmo. O sistema deverá, então, abrir o BDG e fazer uma carga para a AT da representação da rede externa nele contida. Neste momento o projetista deverá definir (com a ajuda do mouse) a região a ser extraída para projeto. Esta região deverá ser armazenada num BDP cujos nome e senha serão fornecidos pelo projetista através de uma CD. O sistema irá, então, fechar o BDG aberto, abrir o BDP recém criado e fazer uma carga deste para a AT.

Caso de uso 2: Selecionar o tipo de elemento de rede POSTE

Descrição:

A seleção de um tipo de elemento de rede POSTE é feita a partir da opção correspondente do MP. Uma CD é, então, aberta onde são listados todos os possíveis tipos de elemento de rede. O usuário (UC ou projetista) deverá selecionar o tipo POSTE. Será associado, então, à JP um ME formado pelas opções específicas de inclusão, exclusão, alteração, associação e consulta do tipo POSTE. Dentre estas opções, apenas a opção de inclusão estará habilitada.

Caso de uso 3: Selecionar o elemento de rede POSTE na rede externa

Descrição:

Para selecionar um POSTE existente na rede externa o usuário (UC ou projetista) deve clicar sobre a representação deste disponibilizada na AT. O sistema deverá ser capaz de reconhecer o tipo do elemento selecionado (POSTE) e obter os valores dos seus atributos através de uma consulta ao banco de dados aberto (BDG ou BDP). Será associado, então, à JP um ME formado pelas opções específicas de inclusão,

exclusão, alteração, associação, e consulta do tipo POSTE. Dentre essas opções, apenas a opção de inclusão estará desabilitada.

Caso de uso 4: Inserir o elemento de rede POSTE em BDG

Descrição:

Para inserir um novo POSTE à rede externa o UC deve, primeiramente, **selecionar o tipo de elemento de rede POSTE** e depois selecionar a operação de inclusão do ME. Nessa opção o UC deve primeiro indicar (com o mouse) a posição de inserção do POSTE na AT e em seguida fornecer os valores dos seus atributos, através de uma CD. O sistema irá, então, enviar as informações do novo elemento para o BDG aberto para que seja incluído.

Caso de uso 5: Alterar o elemento de rede POSTE em BDG

Descrição:

Para alterar um elemento de rede POSTE existente na rede externa o UC deve, primeiramente, **selecionar o elemento de rede POSTE na rede externa** a ser alterado e depois selecionar a operação de alteração do ME. Será apresentada para o UC uma CD com os valores dos atributos do POSTE para que sejam alterados. Uma vez encerradas as alterações, o sistema deverá enviar os novos valores dos atributos para o BDG a fim de que sejam modificados.

Caso de uso 6: Associar um POSTE a um LANCE DE CABO em BDG

Descrição:

Para associar um POSTE a um LANCE DE CABO, o UC deve primeiramente selecionar a opção correspondente no ME. Em seguida deve-se selecionar na AT (com a ajuda do mouse) o lance de cabo que se deseja associar. Um pedido de execução da associação é, então, enviado para o BDG.

Caso de uso 7: Validação da associação POSTE/LANCE DE CABO entre cadastro/projeto

(extensão do caso de uso 6)

Descrição:

A validação da associação POSTE/LANCE DE CABO entre cadastro/projeto ocorre em **Associar um POSTE a um LANCE DE CABO em BDG** depois que a operação associação foi concluída. Neste momento, o sistema verifica se o POSTE alterado não está participando de algum projeto em andamento. Se for o caso, o módulo de cadastro deverá enviar uma mensagem para os responsáveis do respectivo projeto informando das alterações. Estes deverão decidir pela reextração ou não da área geográfica.

Segunda Etapa - Identificação das frequências dos casos de uso

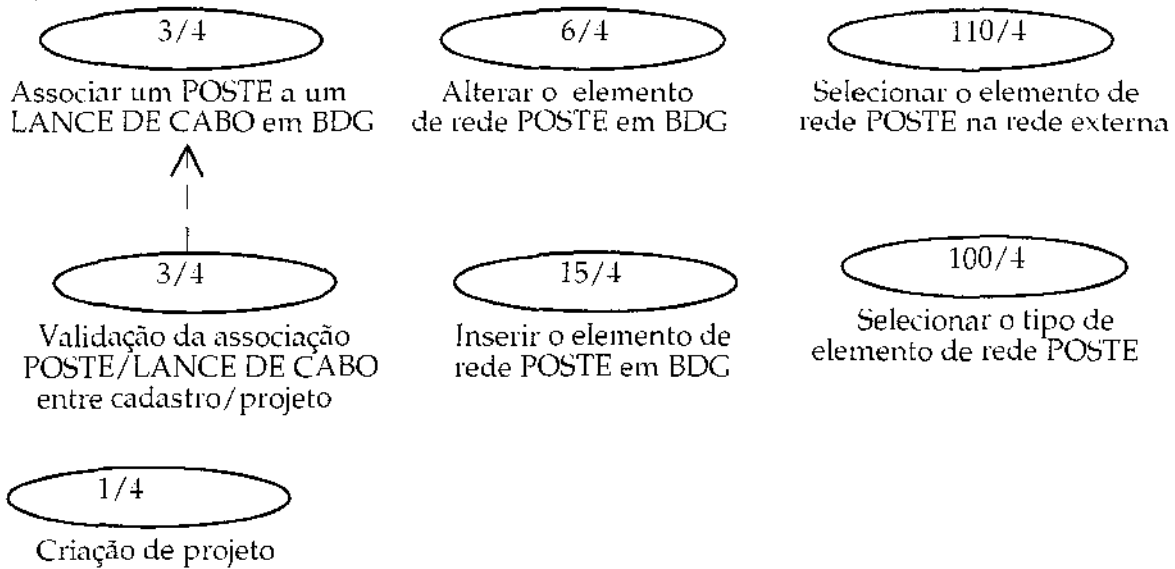


Figura 6.6: Casos de uso, extensões e frequências

Modelo de Objetos

Classes de objetos, atributos e responsabilidades

(A seguir estão descritas as características apenas das superclasses identificadas)

Objetos de Interface

- *Janela:*

atributos: dimensões ((X1, Y1);(X2, Y2))

responsabilidades: dimensionar, maximizar, minimizar, abrir, fechar.

- *Menu*

atributos: lista de operações, lista de chamadas para as operações

responsabilidades: inserir uma nova operação, excluir uma operação, habilitar/desabilitar uma entrada, alterar uma chamada de operação.

- *Área de Trabalho*

descrição: formada apenas pela representação geográfica da planta. Quando um BDG é aberto, faz-se uma carga apenas da representação geográfica da planta armazenada no mesmo. A partir deste objeto Área de Trabalho, o usuário poderá selecionar o componente na planta que desejar trabalhar

atributos: coordenadas da planta

responsabilidades: selecionar um componente, "refresh" (da representação geográfica quando é feita alguma inclusão, remoção ou alteração de algum elemento de rede), zoom in/zoom out

Objetos de Controle

- *Elemento de Controle*

descrição: coordena as operações sobre os elementos de rede armazenados no BDG. Deve-se criar um Elemento de Controle para cada Elemento de Rede.

atributos: depende do elemento de rede que representa. Por exemplo, se for um elemento de rede poste, terá os seguintes atributos: altura, proprietário, material.

responsabilidades: depende do elemento de rede que representa. Se for um elemento de rede poste, terá as seguintes responsabilidades: inserir, excluir, alterar e associar poste (a operação de associação representa relacionar um poste com algum outro elemento de rede).

- *Módulo de Controle*

descrição: coordena as atividades gerais de um módulo. Atividades que não são específicas dos objetos Elemento de Controle.

atributos: nome do banco de dados aberto, elemento de rede corrente (representa o elemento de rede selecionado pelo usuário), lista de operações (disponibilizadas através de um objeto menu).

responsabilidades: depende do módulo que estiver representando. Por exemplo, se for o módulo de projeto, coordena atividades de abertura e criação de projeto.

- *Controle de Banco de Dados*

descrição: representa o gerenciador dos bancos de dados

atributos: nome do banco de dados correntemente aberto, situação deste banco de dados

responsabilidades: abrir e fechar banco de dados, autorizar o acesso de usuários aos bancos de dados.

Objetos Entidade

- *Elemento de Rede*

descrição: representa os componentes (informações geográficas) contidas nas plantas que são armazenadas no BDG (ou BDP)

atributos: representam as características técnicas do componente. Por exemplo, para o elemento de rede poste, tem-se os seguintes atributos: altura, proprietário e material.

responsabilidades: estão relacionadas com a criação e destruição do componente, com a garantia da integridade das associações que o componente mantém com outros elementos de rede.

Associações/Agregações e Generalizações

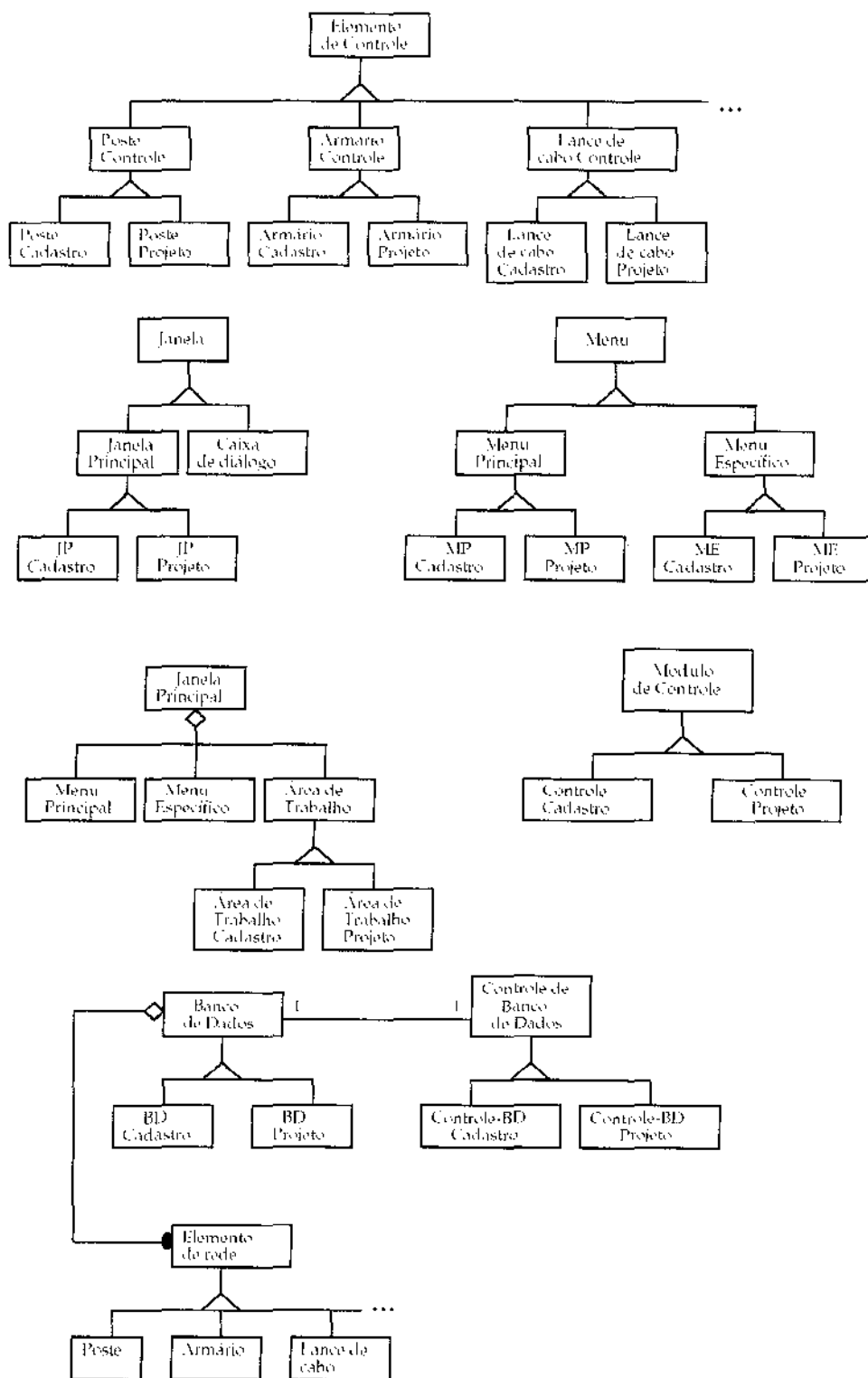
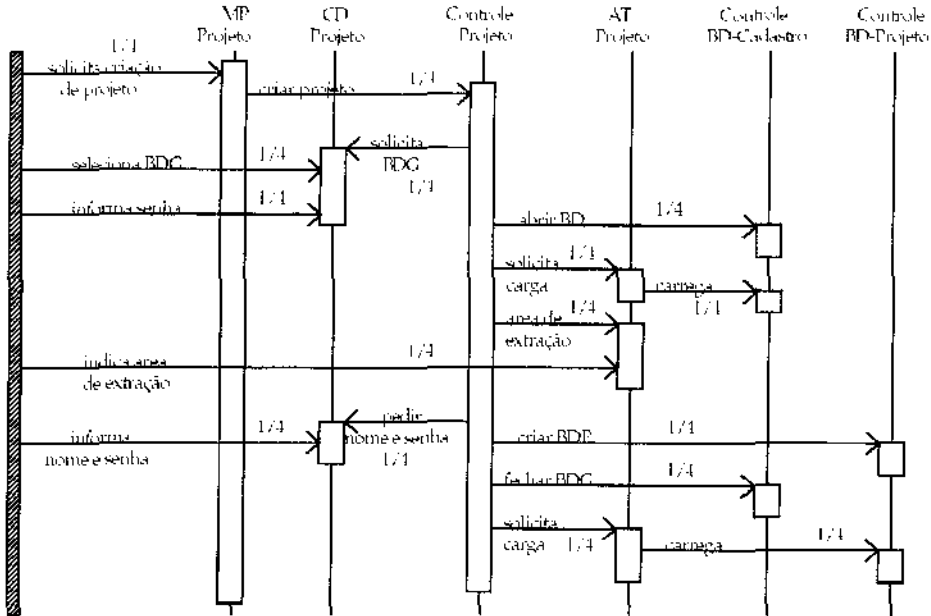


Figura 6.7: Organização das classes de objetos

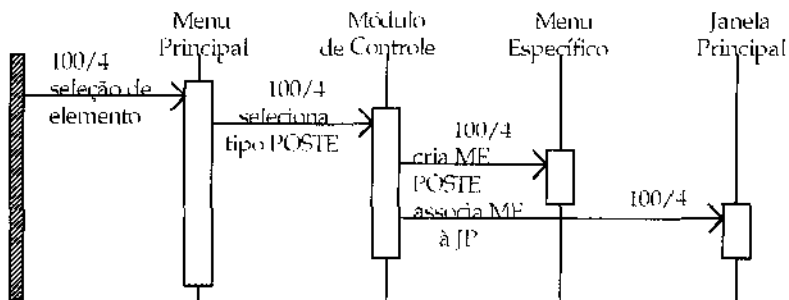
Modelo Dinâmico

Diagramas de Interação

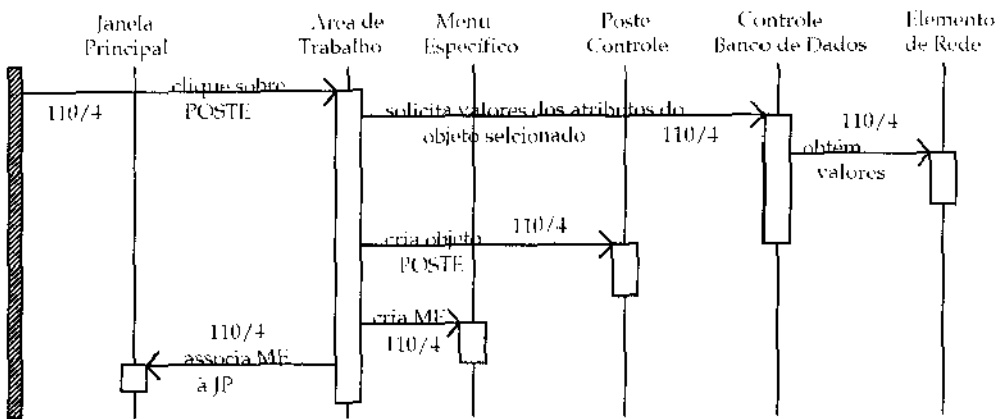
Caso de uso 1: Criação de Projeto



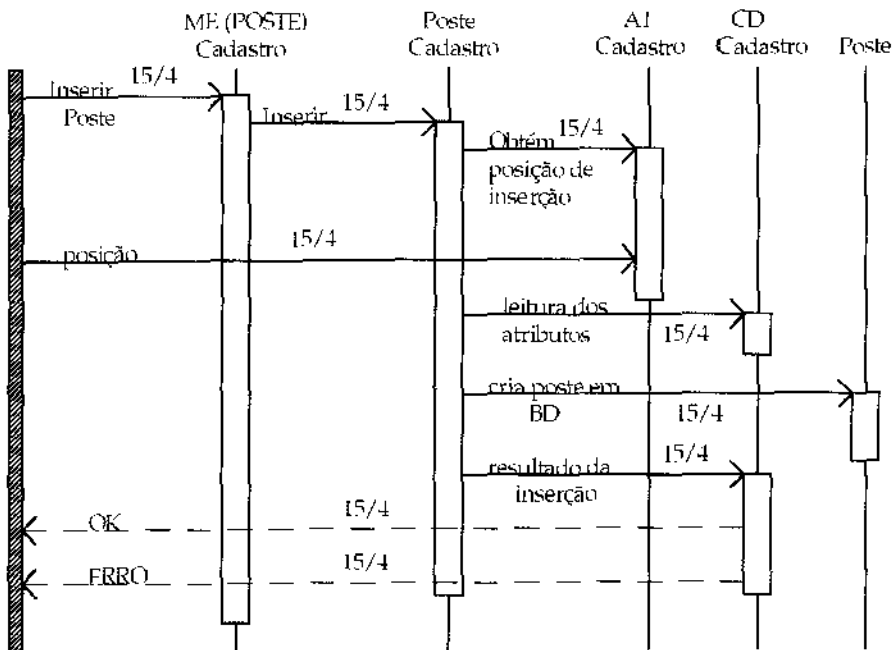
Caso de uso 2: Selecionar o tipo de elemento de rede POSTE



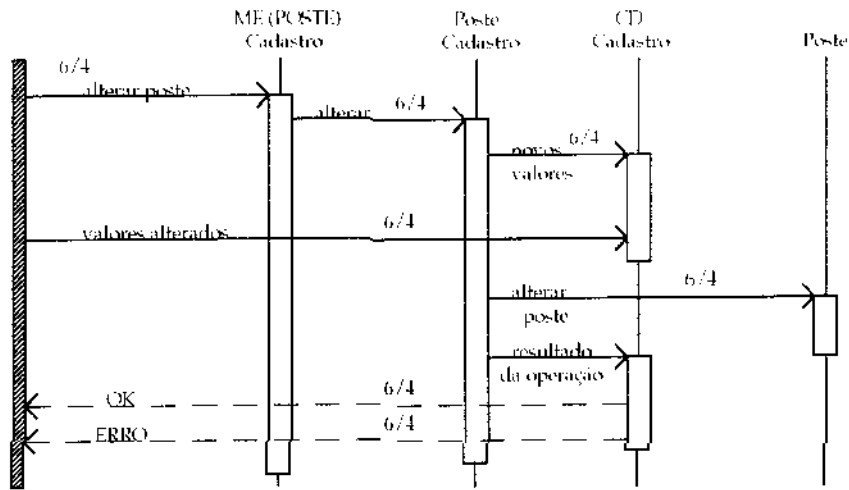
Caso de uso 3: Selecionar o elemento de rede POSTE na rede externa



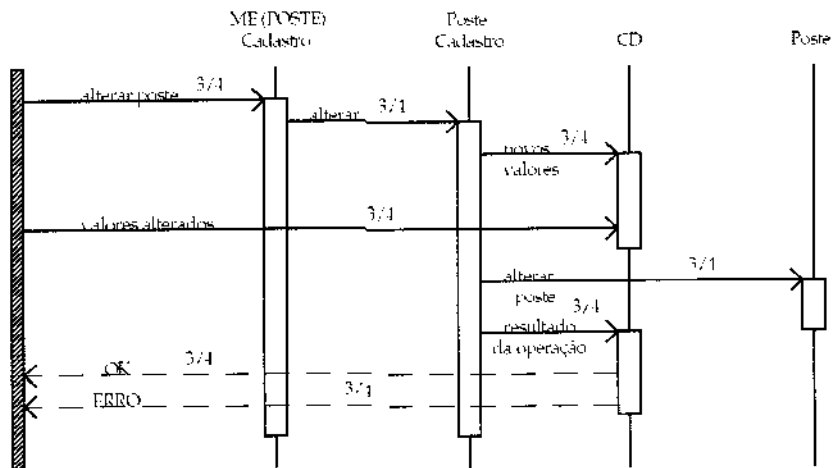
Caso de uso 4: Inserir o elemento de rede POSTE em BDG



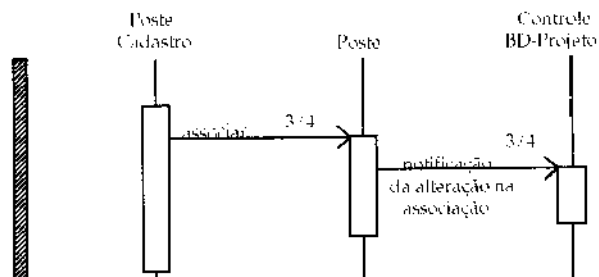
Caso de uso 5: Alterar o elemento de rede POSTE em BDG



Caso de uso 6: Associar um POSTE a um LANCE DE CABO em BDG

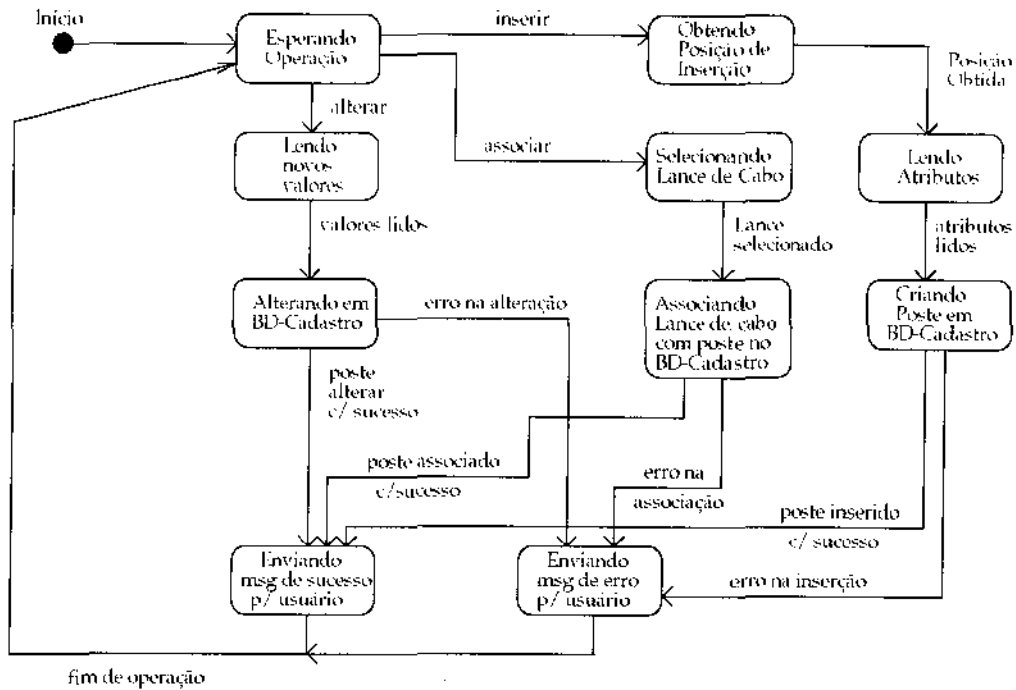


Caso de uso 7: Validação da associação POSTE/LANCE DE CABO entre cadastro/projeto



Diagramas de estados

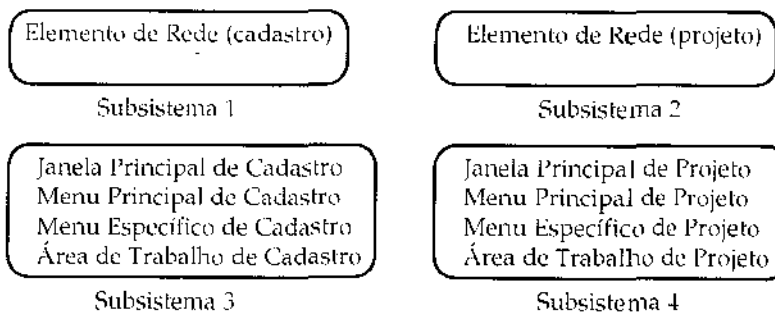
O diagrama abaixo apresentado refere-se à classe de objeto Poste Cadastro. Este diagrama foi derivado dos diagramas de interação apresentado nesta seção dos quais o objeto Poste Cadastro participa.



Subsistemas

Resultado da Primeira Etapa

regra 1: Associações/ Agregações



regra 2: Colaborações

Controle BD-Cadastro
Elemento de Rede (cadastro)
Poste Cadastro,
Armário Cadastro,
Lance de Cabo Cadastro, etc.

Subsistema 1

Controle BD-Projeto
Elemento de Rede (projeto)
Poste Projeto,
Armário Projeto,
Lance de Cabo Projeto, etc.

Subsistema 2

Janela Principal de Cadastro
Menu Principal de Cadastro
Menu Específico de Cadastro
Área de Trabalho de Cadastro
Caixa de Diálogo Cadastro

Controle- Cadastro

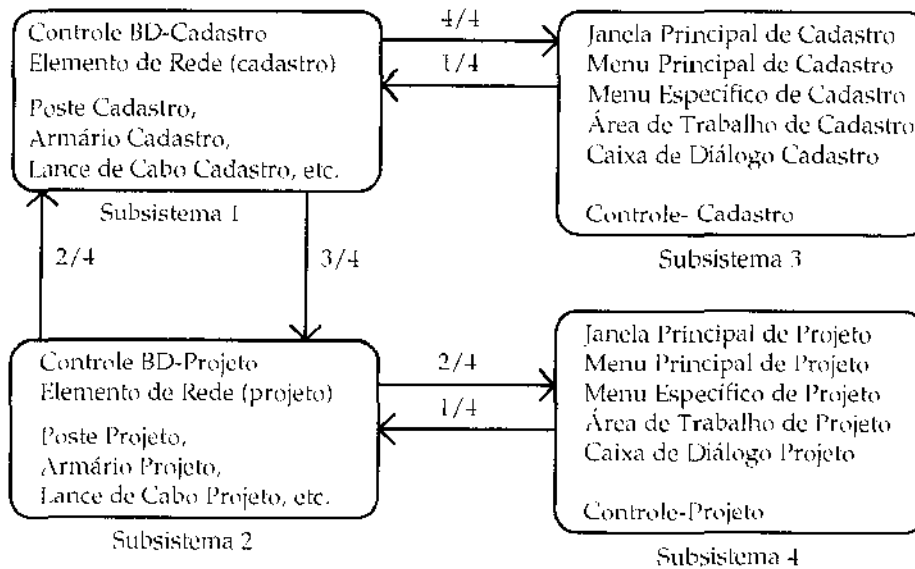
Subsistema 3

Janela Principal de Projeto
Menu Principal de Projeto
Menu Específico de Projeto
Área de Trabalho de Projeto
Caixa de Diálogo Projeto

Controle-Projeto

Subsistema 4

Diagrama de Dependências



Resultado da Segunda Etapa

métrica: frequências

- Objeto Controle-Cadastro transferido do subsistema 1 para o subsistema 3;
- Objeto Controle-Projeto transferido do subsistema 2 para o subsistema 4.

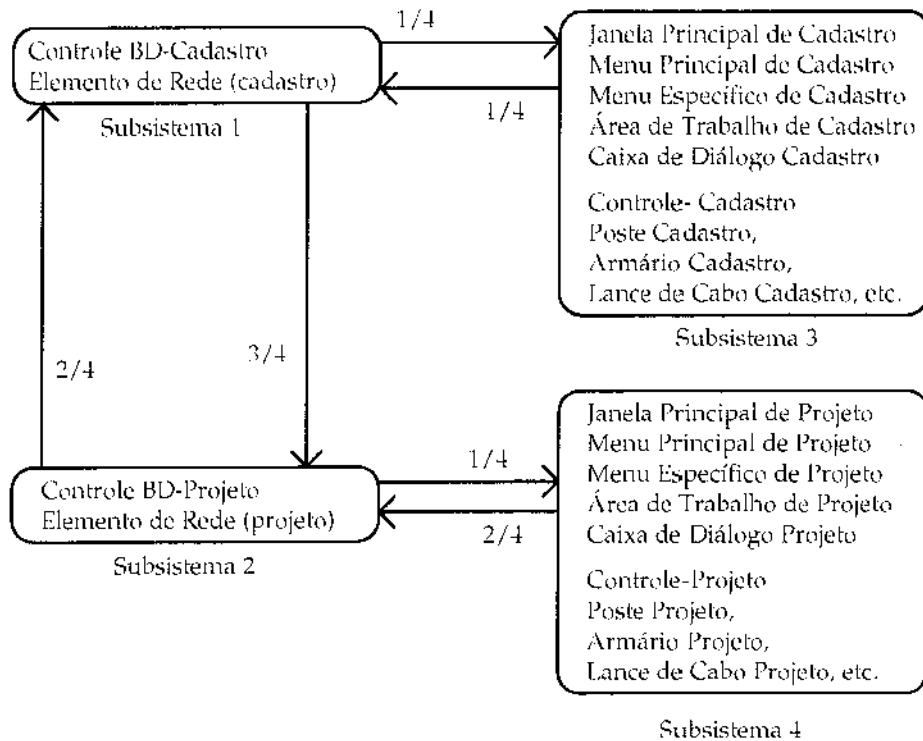
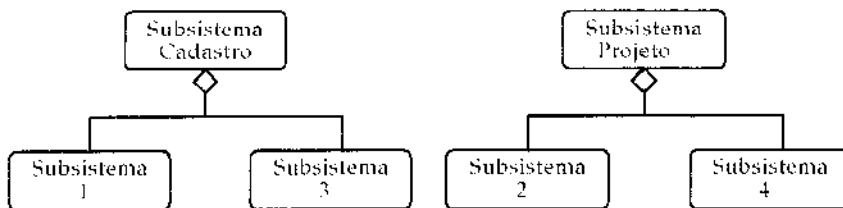


Diagrama de Proximidade



6.3.2 Fase de Projeto

Objetos Clientes

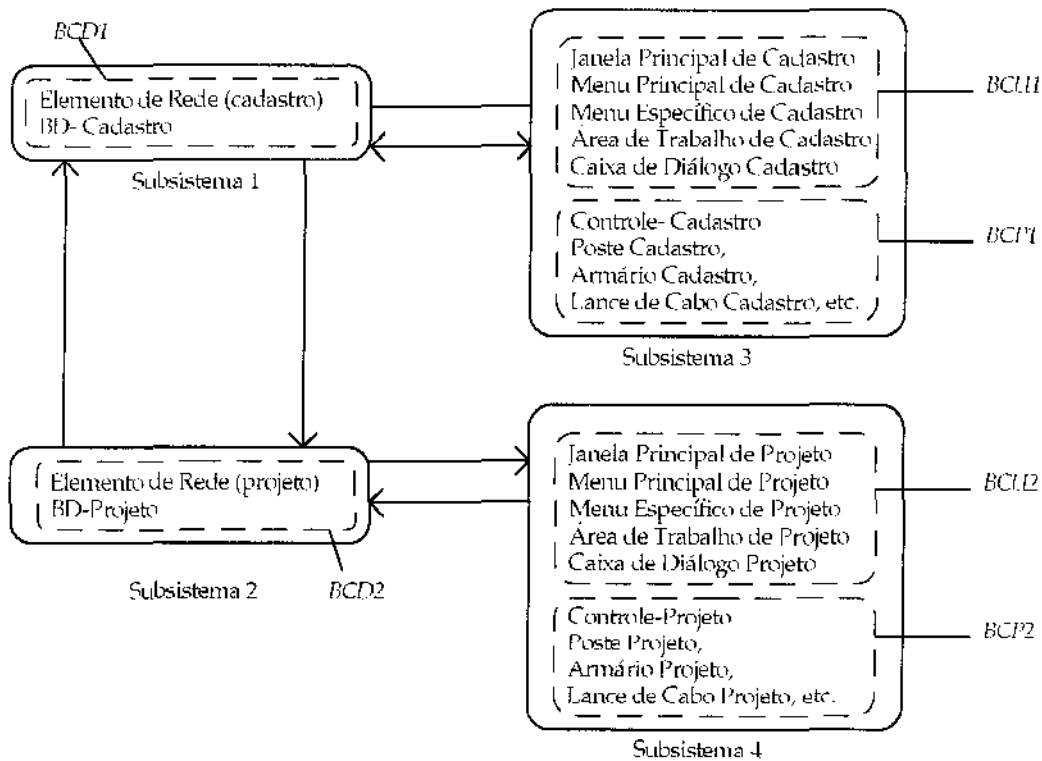
Janela Principal, Menu Principal, Menu Específico, Área de Trabalho, Caixa de Diálogo, Módulo de Controle, Elemento de Controle.

Objetos Servidores

Elemento de Rede (cadastro e projeto), BD-Cadastro e BD-Projeto

Blocos de Construção

- Camada de Dados: BCD1 e BCD2
- Camada de Processamento: BCP1 e BCP2
- Camada de Usuário: BCU1 e BCU2



6.4 Conclusão

Os objetivos deste capítulo eram dois:

1. apresentar uma descrição de um SID, e
2. mostrar como o método proposto no Capítulo 5 pode ser aplicado no desenvolvimento deste SID.

Com relação ao primeiro objetivo listado, foi apresentada uma descrição de um SIG distribuído que automatiza as atividades de projeto de rede externa realizada atualmente, pelas empresas operadoras do Sistema TELEBRÁS, manualmente. A partir desta descrição foram, então, escolhidas algumas situações que retratavam a necessidade de distribuição do SIG e com base nestas situações foram aplicados os conceitos e modelos propostos pelo método apresentado no Capítulo 5 (segundo objetivo).

Na aplicação do método, apresentada na segunda parte do capítulo, foi ilustrado como as informações modeladas durante a fase de análise podem ser utilizadas durante a fase de projeto de maneira a auxiliar o processo de distribuição dos objetos. A fase de implementação não foi tratada uma vez que está mais ligada ao processo de codificação de aplicação, processo este largamente suportado hoje em dia por ferramentas CASE tais como Forté [FORT95] e Dinasty [DINA95]. Tais ferramentas atendem quase que plenamente aos requisitos de ambiente distribuídos apontados na Seção 5.1.3.

Capítulo 7 - Conclusões

Neste trabalho foi apresentado um método orientado a objetos de desenvolvimento de sistemas de informação distribuídos. A elaboração deste método foi feita a partir de estudos de métodos OO (Capítulo 2) e arquiteturas (Capítulo 3). Também neste trabalho foi proposta uma ferramenta que automatiza o processo de agrupamento de objetos em subsistemas a partir da afinidade que existe entre os mesmos. Esta ferramenta representa uma evolução das ferramentas implementadas pelos programas ADW e IEF apresentados no Capítulo 4.

As contribuições e extensões relacionadas ao método e à ferramenta são listados nas duas seções que seguem.

7.1 Contribuições

As contribuições apresentadas neste trabalho são as seguintes:

Com relação ao método proposto

1. A integração dos conceitos utilizados na construção de modelos conceituais dos métodos OMT, RDD e Use Case com os conceitos apresentados na especificação das arquiteturas distribuídas DOM, CORBA e OSCA. Como resultado disto, as equipes de desenvolvimento podem criar e implementar sistemas de informação distribuídos de maneira a fazer um uso pleno dos recursos computacionais distribuídos disponíveis no mercado (por exemplo, banco de dados distribuídos, internet, intranet, etc.). Tendo um bom suporte nas tomadas de decisões de como melhor distribuir os objetos, as pessoas responsáveis pelo desenvolvimento dos SIDs podem, de maneira mais precisa, identificar quais são os melhores recursos computacionais distribuídos que melhor atendem às suas necessidades de implementação. Neste sentido promove-se, então, uma melhor utilização das tecnologias atuais distribuídas.
2. A apresentação de uma forma de agrupar objetos em subsistemas a partir da afinidade que existe entre os mesmos, durante a fase de análise. A afinidade entre objetos é identificada a partir da maneira como os mesmos se relacionam através de associações/agregações (regra 1) e colaborações (regra 2). É apresentada também nesta fase uma métrica que avalia a distribuição dos objetos em subsistemas. O objetivo desta métrica é fazer com que o SID venha a apresentar um bom desempenho.
3. A promoção de uma maior consistência entre as informações modeladas na fase de análise e as informações tratadas durante as fases de projeto e implementação no desenvolvimento de um SID (conceito de rastreabilidade - veja Seção 2.4.2.1). Os projetistas não precisam mais fazer muitas considerações no momento de decidir pela melhor maneira de distribuir os objetos. Uma pré-distribuição já terá sido feita durante a fase de análise.

Com relação a ferramenta proposta

A extensão do processo de agrupamento implementado pelos dois programas apresentados no Capítulo 4 em dois aspectos. Em primeiro lugar a ferramenta proposta está baseada no paradigma de orientação a objetos enquanto que os programas estão baseado na análise estruturada. Em segundo lugar, a ferramenta proposta promove uma avaliação dos agrupamentos a fim de verificar se a distribuição dos objetos em subsistemas deve proporcionar um bom desempenho para o SID. Os programas do Capítulo 4 se restringem a apresentarem os agrupamentos de processos e entidades de dados em sistemas de informação e bancos de dados respectivamente, não sendo feita nenhuma avaliação sobre os mesmos.

7.2 Extensões

Muitas são as extensões relacionadas a este trabalho. Algumas delas são apontadas a seguir:

Com relação ao método proposto

Na fase de análise:

- identificação mais precisa dos aspectos de distribuição que o usuário deve apresentar na especificação de requisitos (por exemplo: onde estão localizadas as informações; qual é o grau de confiabilidade e escalabilidade desejado e outros pontos relacionados com a distribuição do sistema de informação);
- elaboração de uma notação que represente melhor os aspectos de distribuição. Com relação ao modelo de objetos, por exemplo, notações que identifiquem os objetos que podem ser replicados e objetos que não devem ser replicados. Com relação ao modelo dinâmico, notações que diferenciem o envio de mensagens síncrono e assíncrono; que representem a semântica de controle de concorrência, e a migração de objetos [LOW96].
- formalização da métrica e das regras apresentadas para a construção e avaliação dos subsistemas. Tal formalização pode ser alcançada a partir de um estudo de caso completo de desenvolvimento de um SID real, ou a partir de simulações de situações de distribuição de objetos.

Na fase de projeto:

- tratamento de concorrência entre os objetos. Fazer uma análise de quais objetos podem operar de maneira concorrente, e partir disto identificar uma distribuição mais precisa e detalhada dos objetos entre os subsistemas;
- identificação de práticas mais precisas a serem utilizadas na alocação das classes nas máquinas disponíveis para implementação. Estas práticas devem apontar como deve ser conduzida a distribuição dos blocos de construção entre as máquinas; como o diagrama de proximidade pode ser utilizado mais efetivamente nesta distribuição; quais são as regras de distribuição que devem ser seguidas.

Na fase de implementação:

- técnicas de implementação de classes num ambiente distribuído;
- propostas de como conduzir os testes sobre a aplicação resultante.

Com relação a ferramenta proposta

- Implementação de uma ferramenta que monte a matriz de colaboração;

Referências Bibliográficas

- [BELL92] Bellcore, "The Bellcore OSCA Architecture," Technical Advisory TA-ST-000915, Issue 3, Março, 1992.
- [BETZ94] Betz, M., "Interoperable Objects," Dr. Dobb's Journal, pp. 18-39, Outubro, 1994.
- [BETZ95] Betz, M., "Networking Objects with CORBA," Dr. Dobb's Journal, pp. 18-26, Novembro, 1995.
- [BERN96] Bernstein, P.A., "Middleware: A Model for Distributed Services," Communications of ACM, Vol. 39, No. 2, pp. 86-98, Fevereiro, 1996.
- [CCHMM96] Câmara, G., M.A. Casanova, A.S. Hemerly, G.C. Magalhães, C.M.B. Medeiros, "Anatomia de Sistemas de Informação Geográfica", 10a. Escola de Computação organizada pelo Instituto de Computação, UNICAMP, Campinas, SP, 8 a 13 de Julho, 1996.
- [COAD90] Coad, P. and E. Yourdon, "Object-Oriented Analysis," Prentice-Hall, 1990.
- [DINA95] Dynasty Technologies Inc., "Building The Open Enterprise - Product Overview," 1995.
- [FAYA94] Fayad, M.E., W.T. Tsai, R.L. Anthony, M.L. Fulghum, "Object Modeling Technique (OMT): Experience Report," Journal of Object-Oriented Programming, 7(7), pp. 46-58, Novembro-Dezembro, 1994.
- [FORT95] Forté Software Inc., "Forté Application Development," Versão 2.0, Dezembro, 1995.
- [IDE96] Interactive Development Environments Inc., "Creating OMT Models," Software through Pictures, Release 3.2, Junho, 1996.
- [JACO87] Jacobson, I., "Object-Oriented Development in an Industrial Environment," Proceedings OOPSLA'87, pp 183-191, 4 a 8 de Outubro, 1987.
- [JACO92] Jacobson, I., M. Christerson, P. Jonsson, G. Övergaard, "Object-Oriented Software Engineering: A Use Case Driven Approach," Addison-Wesley, ACM-Press, 1992.
- [JACO95] Jacobson, I., M. Christerson, "A Growing Consensus on Use Case," Journal of Object-Oriented Programming, 7(10), pp. 15-19, Março-Abril, 1995.

- [KIPP93] Kipper, E.F., C.A. Müller, E.A. Bastos, J.T.S. Cevallos, J.I. Jaeger, M.A. Resende, "Engenharia de Informações: Conceitos, Técnicas e Métodos," Sagra-De Luzzato Editores, Porto Alegre, 1993.
- [KNOW91] KnowledgeWare Inc., "Planning Working Station - Basics User Guide", Application Development Workbench, Release 1.06.02, Agosto, 1991.
- [LOW96] Low, G.C., G. Rasmussen, B. Henderson-Sellers, "Incorporation of Distributed Computing Concerns into Object-Oriented Methodologies," Journal of Object-Oriented Programming, 9(3), pp. 12-20, Junho, 1996.
- [MANO90] Manola, F. and P. Buchmann, "A Functional/Relational Object-Oriented Model for Distributed Object Management: Preliminary Description", TM-0331-11-90-165, GTE Laboratories Incorporated, Dezembro, 1990.
- [MANO92] Manola, F., S. Heiler, D. Georgakopoulos, M. Hornick, and M. Brodie, "Distributed Object Management," International Journal of Intelligent and Co-operative Information Systems, Vol. 1, No. 1, World Scientific, Março, 1992.
- [MART89] Martin, J., "Information Engineering: Introduction (Book 1)," Prentice-Hall, Englewood Cliffs, New Jersey, 1989.
- [MART90] Martin, J., "Information Engineering: Planning and Analysis (Book 2)," Prentice-Hall, Englewood Cliffs, New Jersey, 1989.
- [MILL92a] Mills, J.A., "OSCA Architecture: Focusing On Functionality," Bellcore Exchange magazine, pp. 26-28, Janeiro-Fevereiro, 1992.
- [MILL92b] Mills, J.A., "Large Scale Interoperability, Distributed Transaction Processing, and Open Systems," Proceedings of the USING'92 - Freedom of Open Systems: Open Solutions for the Telecom Industry; Philadelphia, Pennsylvania, 12 a 14 de Maio, 1992.
- [OMG92a] The Object Management Group Inc., "The Common Object Request Broker: Architecture and Specification," OMG Document no. 93.12.29, Revision 1.2, 1992.
- [OMG92b] The Object Management Group Inc., "Object Management Architecture Guide," OMG TC Document no. 92.11.1. Revision 2.0, 1992.
- [OMG96] The Object Management Group Inc., "The Common Object Request Broker 2.0 / Internet Inter ORB Protocol Specification," OMG Technical Document PTC/96-08-04, 1996.

- [PLAN95] PLANOP - Plano de Operações Integradas - Versão 3.0 - CPqD/TELEBRÁS - Julho, 1995.
- [POT96] Prezzotto, A.L.B., F.I. Obata, D.G. Teijeiro, "Telecommunications Outside Plant: Design Automation", CPqD- TELEBRÁS, Março, 1996.
- [POTP96] Prezzotto, A.L.B., F.I. Obata, D.G. Teijeiro, O. Piton Jr., "SAGRE/Proj: A Nova Geração de Projeto de Rede Externa" , CPqD- TELEBRÁS, Abril, 1996.
- [PRES92] Pressman, R.S., "Software Engineering: A Practitioner's Approach," McGraw-Hill, 1992.
- [RATI96] Rational Rose, "Using Rational Rose 4.0," Rational Software Corporation, Release 4.0, Novembro, 1996.
- [RUMB91] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modeling and Design," Prentice-Hall International, 1991.
- [RUMB95a] Rumbaugh, J., "OMT: The object model," Journal of Object-Oriented Programming, 7(8), pp. 21-27, Janeiro, 1995.
- [RUMB95b] Rumbaugh, J., "What is a method?," Journal of Object-Oriented Programming, 8(6), pp. 10-16, Outubro, 1995.
- [SERC95] Sercheli, A.D., F.J.S. Lopes, J.R. Jaime, S.R. Pereira, W.S. Speltri, "Planejamento Estratégico da Informação Utilizando o Software Application Development Workbench (ADW)", Projeto ARCO, Departamento de Sistemas de Operação, Divisão de Engenharia de Sistemas de Informação, CPqD-TELEBRÁS, 1995.
- [TEXA90] Texas Instruments, "Um Guia para a Engenharia da Informação Utilizando a IEF - Planejamento, Análise e Projetos Assistidos por Computador," Segunda Edição, Janeiro, 1990.
- [TEXA93] Texas Instruments, "Data Modeling and Matrices - Activity Analysis," IEF - Information Engineering Facility, Primeira Edição, Setembro, 1993.
- [WIRF90a] Wirfs-Brock, R., B. Wilkerson, L. Wiener, "Designing Object-Oriented Software," Prentice-Hall, Inc., 1990.
- [WIRF90b] Wirfs-Brock, R., R.E. Johnson, "Surveying: Current Research in Object-Oriented Design," Communication of ACM, Vol. 33, No. 9, pp. 104-124, Setembro, 1990.