

PAULO ROBERTO CELIDONIO CAROLI

**UMA METODOLOGIA DE PROJETO DE
SOFTWARE ORIENTADO A OBJETOS**

DISSERTAÇÃO DE MESTRADO

DEPARTAMENTO DE INFORMÁTICA

PUC-Rio

Rio de Janeiro, 22 de dezembro de 1999

PAULO ROBERTO CELIDONIO CAROLI

**UMA METODOLOGIA DE PROJETO DE
SOFTWARE ORIENTADO A OBJETOS**

Dissertação apresentada ao Departamento de
Informática da PUC-Rio como parte dos
requisitos para a obtenção do título de
Mestre em Ciências em Informática.

Orientador: Carlos Jose Pereira de Lucena

Co-orientador: Firmo Freire

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 22 de dezembro de 1999

“A melhor forma de resolver um problema é apresentar a solução mais simples possível, mas não mais simples que esta. “

Albert Einstein

Ao professor Sérgio Carvalho

Agradecimentos

A Carlos José Pereira de Lucena, Firmo Freire, Arndt von Staa e Maurício José Vianna e Silva, participantes da Comissão Julgadora deste trabalho.

A Firmo Freire, Clécio Radler dos Guarany's e Bruno Rodrigues de Mello, amigos e colegas de trabalho envolvidos diretamente no desenrolar desta dissertação. Agradecimentos especiais ao Firmo, quem confiou e participou do desenvolvimento e evolução deste trabalho.

Ao professor Sérgio Carvalho e seus alunos, em especial Toacy e Sylvia, com os quais tive a oportunidade de aprender a qualidade do paradigma de Orientação a Objetos.

Ao professor Carlos José Pereira de Lucena e seus alunos. A qualidade deste grupo e suas reuniões apresentam um alicerce importante para a evolução dos trabalhos de pesquisa.

Aos doutores e doutorandos Marcus Fontoura (Mafé), Cristiano Braga, Elvira Maria Antunes Uchôa e Renato Cerqueira pelas conversas e trocas de idéias nas áreas de pesquisa que precederam este trabalho.

À Viviane Torres da Silva pelo carinho, amizade e ajuda, fundamentais no momento de conclusão deste trabalho.

À Fernanda Victal Mesquita, amiga e companheira do mestrado.

A Rodrigo Lemos de Assis e Claudia Mesquita, amigos e instrutores das ferramentas Microsoft Word e Microsoft Power Point.

Ao Departamento de Informática da PUC-Rio, seus professores e funcionários.

À CAPES pelo auxílio financeiro.

Ao LES (Laboratório de Engenharia de Software) por proporcionar o ambiente de integração indústria/universidade, ideal para a realização deste trabalho, dando suporte a toda sua evolução.

À minha família, em especial meu irmão, Luis Eduardo Celidonio Caroli, por ser sempre este exemplo de força e perseverança.

Aos amigos de longa data Renata Vargas e Rafael d'Angelo, os quais fizeram parte desta conquista, ajudando-me a vencer momentos difíceis.

Aos meus amigos, pessoas especiais sempre presentes e participantes na minha vida.

Ao Half (meu cachorro e filho), ouvinte paciente de todas minhas dúvidas, problemas e indagações; amigo sempre fiel e companheiro.

Resumo

Apesar da crescente utilização do paradigma de Orientação a Objetos, ainda existe uma impedância na plenitude de sua aplicação no desenvolvimento de sistemas, como por exemplo em Sistemas de Informação para a Web. Nesses, Interfaces são, geralmente, orientadas a eventos, e a maioria das Bases de Dados utilizadas são Relacionais.

O trabalho apresentado propõe uma Arquitetura em 3 Camadas para o desenvolvimento de sistemas que se dividem naturalmente em Interface, Negócio e Persistência, além de propor uma Metodologia para o desenvolvimento da Camada de Negócio.

Os objetivos deste trabalho são:

- auxiliar no acompanhamento e no gerenciamento de desenvolvimento de sistemas;
- garantir a continuidade de sistemas grandes e suscetíveis a alterações e evoluções;
- auxiliar na comunicação entre as diferentes pessoas envolvidas no projeto de sistemas;
- dar suporte ao desenvolvimento Orientado a Objetos coexistindo com diferentes paradigmas;
- apresentar uma documentação efetiva do projeto.

O trabalho foi validado a partir de sua aplicação em projetos reais.

Abstract

In spite of the increasing utilization of the Object Oriented paradigm, there still exists reluctance on its application in the development of systems as, for example, Web Information Systems. In these type of applications, User Interfaces are in general event oriented and the majority of the Databases are Relational.

This work proposes a Three-Tier Architecture for the development of systems that can be naturally divided in User Interface, Business and Persistence layers. It also proposes a Methodology for the development of the Business layer.

The objectives of this work are:

- to help management to keep up with the project development;
- facilitate the maintenance of large systems subject to modification and evolution;
- help communication among the persons in the development team;
- provide Object Oriented development support coexisting with other paradigms;
- serve as an effective project documentation .

Applying the concepts here put forth in real projects validated the work.

Sumário

1	INTRODUÇÃO	1
2	ARQUITETURAS DE SOFTWARE	4
2.1	ESCOLHA	4
2.2	ARTEFATOS	6
3	A ARQUITETURA EM CAMADAS	4
3.1	ARQUITETURA EM 2 CAMADAS	10
3.2	ARQUITETURA EM 3 CAMADAS	12
4	MOTIVAÇÃO	14
4.1	O PROBLEMA	14
4.2	A SOLUÇÃO PROPOSTA	16
5	A ARQUITETURA	17
5.1	ELEMENTOS DA ARQUITETURA	19
5.1.1	<i>Camada de Interface (CI)</i>	19
5.1.2	<i>Intercamada Interface/Negócio (IIN)</i>	20
5.1.3	<i>Camada de Negócio (CN)</i>	21
5.1.4	<i>Intercamada Negócio/Persistência (INP)</i>	21
5.1.5	<i>Camada de Persistência (CP)</i>	22
5.1.6	<i>Módulo de Acesso (MA)</i>	24
5.2	VISIBILIDADE DA ARQUITETURA	24
5.3	COLABORAÇÕES NA ARQUITETURA	28
5.4	AVALIAÇÃO DA ARQUITETURA	31
5.4.1	<i>Aspectos de Desenvolvimento</i>	31
5.4.2	<i>Aspectos de Gerência</i>	35
6	A METODOLOGIA	37
6.1	PRÉ-REQUISITO	39
6.2	ETAPAS	40
6.2.1	<i>Definição do Módulo de Acesso</i>	40
6.2.2	<i>Organização da Camada de Negócio</i>	42
6.2.3	<i>Detalhamento das Unidades de Negócio (UNs)</i>	43
6.3	RESULTADOS	48
6.4	AVALIAÇÃO	50
6.4.1	<i>Tamanho</i>	50
6.4.2	<i>Complexidade</i>	53
6.4.3	<i>Acoplamento</i>	54
6.4.4	<i>Qualidade de Abstração</i>	56
6.4.5	<i>Simplicidade</i>	58
6.4.6	<i>Similaridade</i>	59
6.4.7	<i>Volatilidade</i>	61
7	HEURÍSTICAS	63
7.1	RELACIONAMENTOS	63
7.1.1	<i>Associação</i>	63
7.1.2	<i>Agregação e Composição</i>	68
7.1.3	<i>Herança</i>	72
7.1.4	<i>Visão Global</i>	76

7.2	CONCORRÊNCIA	77
7.2.1	<i>Concorrência baseada em Dados</i>	77
7.2.2	<i>Concorrência baseada em Funcionalidades</i>	78
8	TRABALHOS FUTUROS	82
8.1	FERRAMENTA.....	82
8.2	CONTROLE DE TRANSAÇÃO.....	87
8.3	SEPARAÇÃO FÍSICA.....	88
9	CONCLUSÕES	89
APÊNDICE A		93
A.1	PROJETO WANDIRTEL.....	93
A.1.1	<i>Gerência de Materiais</i>	93
A.1.2	<i>Gerência de Configuração</i>	93
A.1.3	<i>Gerência de Clientes</i>	94
A.2	SISTEMA - CONTROLE DE ACESSO.....	94
A.3	UM PROCESSO DE CONSTRUÇÃO DE GERADORES.....	94
APÊNDICE B.....		96
B.1	DIAGRAMAS DE CLASSE.....	96
B.2	CÓDIGO GERADO.....	98
B.2.1	<i>Camada de Interface</i>	99
B.2.2	<i>Intercamada Interface/Negócio</i>	101
B.2.3	<i>Camada de Negócio</i>	102
B.2.4	<i>Intercamada Negócio/Persistência</i>	107
B.2.5	<i>Camada de Persistência</i>	108
B.2.6	<i>Módulo de Acesso</i>	109
B.3	“DUMPS” DE TELA.....	110

Lista de Figuras

FIGURA 3-1 ARQUITETURA EM CAMADAS.....	9
FIGURA 3-2 SISTEMAS EM 2 CAMADAS.....	11
FIGURA 3-3 SISTEMAS EM 3 CAMADAS.....	13
FIGURA 5-1 A ARQUITETURA.....	17
FIGURA 5-2 ARQUITETURA EM TEMPO DE EXECUÇÃO.....	18
FIGURA 5-3 SISTEMA LEGADO TRATADO COMO CP.....	23
FIGURA 5-4 CLASSE DE ACESSO USUÁRIO.....	24
FIGURA 5-5 INTERFACE DE UMA CAMADA.....	26
FIGURA 5-6 VISIBILIDADE ENTRE CAMADAS E MÓDULO DE ACESSO.....	27
FIGURA 5-7 EXEMPLO DE ESPECIALIZAÇÃO DE UMA CLASSE DE ACESSO NA CAMADA DE INTERFACE.....	28
FIGURA 5-8 COLABORAÇÕES INTER-CAMADAS E INTRA-CAMADA.....	29
FIGURA 5-9 COLABORAÇÃO MA-CAMADAS.....	29
FIGURA 5-10 COMUNICAÇÃO INTERFACE ↔ NEGÓCIO.....	30
FIGURA 5-11 COMUNICARÃO NEGÓCIO ↔ PERSISTÊNCIA.....	31
FIGURA 6-1 RESULTADO OBTIDO NA UTILIZAÇÃO DA METODOLOGIA.....	38
FIGURA 6-2 EXEMPLO DE DIAGRAMA DE CLASSES.....	39
FIGURA 6-3 CLASSE DO DIAGRAMA DE CLASSES – CLASSE USUÁRIO.....	41
FIGURA 6-4 CLASSE DO MÓDULO DE ACESSO –CLASSE DE ACESSO USUÁRIO.....	41
FIGURA 6-5 DIAGRAMA DE CLASSES E MÓDULO DE ACESSO.....	41
FIGURA 6-6 UNS DA CAMADA DE NEGÓCIO, GERADAS A PARTIR DO MODELO DE ANÁLISE.....	42
FIGURA 6-7 UNIDADE DE NEGÓCIO.....	43
FIGURA 6-8 INTERMEDIÁRIOS DE INTERFACE PARA USUÁRIO DENTRO DA UN_USUÁRIO.....	45
FIGURA 6-9 ASSOCIAÇÃO ENTRE GERENTES DAS UNS.....	46
FIGURA 6-10 UNIDADE DE NEGÓCIO USUÁRIO (UN_USUÁRIO).....	48
FIGURA 6-11 TRATAMENTO DE DADOS X TRATAMENTO DE FUNCIONALIDADES.....	49
FIGURA 6-12 MEDIDA DO TAMANHO DE UM PROJETO.....	51
FIGURA 6-13 MEDIDA DO TAMANHO DO PROJETO, RESULTADO DA APLICAÇÃO DA METODOLOGIA.....	52
FIGURA 6-14 ATRIBUTOS SIMILARES.....	60
FIGURA 6-15 OPERAÇÕES SIMILARES.....	61
FIGURA 7-1 EXEMPLO DE ASSOCIAÇÃO : FORNECEDOR-PRODUTO.....	63
FIGURA 7-2 EXEMPLO FORNECEDOR-PRODUTO (CN-COMPORTAMENTO).....	64
FIGURA 7-3 EXEMPLO FORNECEDOR-PRODUTO (MA-DADOS).....	65
FIGURA 7-4 EXEMPLO PROFESSOR-TURMA.....	65
FIGURA 7-5 EXEMPLO PROFESSOR-TURMA (CN-COMPORTAMENTO).....	66
FIGURA 7-6 EXEMPLO PROFESSOR-TURMA (MA-DADOS).....	66
FIGURA 7-7 CLASSE SUBSTITUINDO RELACIONAMENTO.....	67
FIGURA 7-8 EXEMPLO FORNECEDOR-PRODUTO (CN-COMPORTAMENTO).....	68
FIGURA 7-9 EXEMPLO FORNECEDOR-PRODUTO (MA-DADOS).....	68
FIGURA 7-10 EXEMPLO DE AGREGAÇÃO : COMPUTADOR-PLACA.....	69
FIGURA 7-11 EXEMPLO COMPUTADOR-PLACA (CN-COMPORTAMENTO).....	69
FIGURA 7-12 EXEMPLO COMPUTADOR-PLACA (MA-DADOS).....	70
FIGURA 7-13 EXEMPLO DE COMPOSIÇÃO : PLACA-PORTA.....	70
FIGURA 7-14 EXEMPLO PLACA-PORTA (CN-COMPORTAMENTO).....	71
FIGURA 7-15 EXEMPLO PLACA-PORTA (MA-DADOS).....	71
FIGURA 7-16 HERANÇA.....	72
FIGURA 7-17 CN - HERANÇA.....	73
FIGURA 7-18 MA - HERANÇA.....	74
FIGURA 7-19 EXEMPLO DE HERANÇA : ALUNO,PROFESSOR,PESSOA.....	75
FIGURA 7-20 EXEMPLO ALUNO, PROFESSOR ->PESSOA (CN-COMPORTAMENTO).....	75
FIGURA 7-21 EXEMPLO ALUNO, PROFESSOR, PESSOA (MA-DADOS).....	76
FIGURA 7-22 MÓDULO DE ACESSO, COMO REPOSITÓRIO DE CLASSES DE ACESSO.....	78
FIGURA 7-23 CLASSES II E GERENTE COMO SINGLETON.....	79

FIGURA 7-24 DIVISÃO DE TRABALHO (INSTANCIÇÃO DE UM NOVO GERENTE)	80
FIGURA 8-1 EXEMPLO DE DIAGRAMA DE CLASSES NA FERRAMENTA RATIONAL ROSE.....	82
FIGURA 8-2 EXEMPLO NA FERRAMENTA VISUAL AGE FOR JAVA.....	83
FIGURA 8-3 CAMADA DE NEGÓCIO E SUAS UNs NO VISUAL AGE FOR JAVA	84
FIGURA 8-4 O MÓDULO DE ACESSO (MA) NO VISUAL AGE FOR JAVA.....	85
FIGURA B. 1 DIAGRAMA DE CLASSES 1	96
FIGURA B. 2 DIAGRAMA DE CLASSES 2	97
FIGURA B. 3 DIAGRAMA DE CLASSES 3	97
FIGURA B. 4 DIAGRAMA DE CLASSES 4	98
FIGURA B. 5 ESTRUTURA DE PACOTES.....	99
FIGURA B. 6 INTERFACE E SEUS PACOTES.....	100
FIGURA B. 7 PACOTE INTEFACE_NEGOCIO	101
FIGURA B. 8 PACOTE NEGOCIO E SEUS SUBPACOTES (UNs).....	104
FIGURA B. 9 HIERARQUIA DE HERANÇA DA UN_EQUIPAMENTO MODULAR (1-2)	105
FIGURA B. 10 HIERARQUIA DE HERANÇA DA UN_EQUIPAMENTO MODULAR (2-2)	106
FIGURA B. 11 PACOTE NEGÓCIO_PERSISTENCIA.....	108
FIGURA B. 12 "MÓDULO DE ACESSO".....	110
FIGURA B. 13 GERENCIA DE MATERIASI - CADASTRAR EQUIPAMENTO.....	111
FIGURA B. 14 GERENCIA DE MATERIASI - CADASTRAR PLACA.....	112
FIGURA B. 15 GERENCIA DE MATERIASI – EDITAR EQUIPAMENTO	113

1 Introdução

À medida que sistemas crescem em tamanho e em complexidade, aumenta-se a necessidade de se realizar um desenvolvimento organizado no qual os objetivos a serem atingidos sejam decididos *a priori*, permitindo que busquem-se as soluções que melhor satisfaçam suas necessidades.

Diferentemente de desenvolvimento de sistemas menores, em sistemas maiores, verificamos que, em geral, mais pessoas interagem para realizar seu desenvolvimento. Essas pessoas são gerentes, analistas, projetistas, arquitetos, testadores, clientes e desenvolvedores de sistemas de Software. Visando organizar o desenvolvimento, as necessidades de todos os envolvidos neste devem ser consideradas.

Todo sistema possui uma vida útil. O tempo de vida de um sistema pode estar diretamente relacionado ao tamanho do mesmo. Sistemas maiores, por estarem relacionados a custos maiores, tendem a persistir mais que sistemas menores; por conseguinte, podem sofrer mais reparos e manutenções ao longo do tempo. O sucesso do sistema não depende apenas do seu desenvolvimento inicial, mas também se o sistema desenvolvido garante os aspectos de evolutibilidade e manutenibilidade.

Gerentes da área de Engenharia de Software, por muitas vezes, deparam-se com a árdua tarefa de acompanhar o desenvolvimento de um sistema. Para realizar este trabalho, é necessário entender como este desenvolvimento é composto, tentar organizá-lo e adequá-lo às equipes de desenvolvimento. O conhecimento técnico, por parte do gerente, torna-se mais necessário para compreender as necessidades do sistema, uma vez que esse apresenta uma organização pouco definida. Esse problema também ocasionará dificuldade na realização de cronogramas de execução do sistema, e conseqüentemente, imprecisão na medição dos custos envolvidos no mesmo.

Os gerentes também precisam atentar para a independência do seu sistema em relação à equipe que realizou o desenvolvimento. O ideal é que o sistema desenvolvido possa ter continuidade, mesmo mudando-se toda a equipe original. Este fato é de extrema

importância, por exemplo, para laboratórios universitários que contam com uma mão de obra especializada que dispõe, no entanto, de prazos curtos. Projetos precisam ser duradouros e o tempo gasto para realizar a transferência de conhecimento entre a equipe que sai e a que entra deve ser minimizado.

Dentro da Engenharia de Software, vem ocorrendo um aumento do nível de especialização dos desenvolvedores. Em um mesmo sistema, podemos ter partes sendo tratadas por diferentes desenvolvedores. Um Banco de Dados será melhor estruturado por um DBA (*Database Administrator*), enquanto que um especialista em interfaces tratará dos aspectos da interface com o usuário, talvez até passando as tarefas de projeto gráfico e de programação visual para algum profissional mais específico na área. Com uma especialização da mão de obra, consegue-se realizar uma melhor organização e divisão do trabalho entre as equipes envolvidas com o desenvolvimento do sistema, e com isso atinge-se uma maior produtividade.

Apesar da crescente utilização do paradigma de Orientação a Objetos, ainda existe uma impedância na plenitude de sua aplicação para o desenvolvimento de sistemas. Isso ocorre pois nem todas as partes do desenvolvimento são melhor estruturadas com Orientação a Objetos. A situação ideal para um sistema Orientado a Objetos puro seria uma interface Orientada a Objetos, com um modelo Orientado a Objetos, e a persistência em uma Base de Dados Orientada a Objetos. Todavia, esta não é a composição comumente utilizada no desenvolvimento de sistemas. Atualmente, interfaces são, geralmente, orientadas a eventos, e a maioria das Bases de Dados utilizadas são Relacionais. Necessita-se, então, de uma organização que seja de um nível tal que possibilite alcançar um bom resultado no projeto e na implementação de sistemas independente das tecnologias envolvidas.

Um mapeamento entre modelos e diagramas, os quais representam as necessidades que serão resolvidas pelo sistema, e sua codificação, permite que um desenvolvimento seja realizado de forma evolutiva, seguindo-se fases de desenvolvimento apropriadamente, desde a coleta de requisitos, até a implementação e testes do sistema. Este mapeamento possibilita a ida e volta entre o código de implementação e seus modelos e diagramas,

sendo também uma documentação exata do desenvolvimento do sistema, útil não apenas durante esta etapa de desenvolvimento, mas também para a documentação do Projeto.

Este trabalho propõe uma Arquitetura em 3 Camadas para o desenvolvimento de sistemas em 3 camadas (Interface/Negócio/Persistência) e uma Metodologia para o desenvolvimento da Camada de Negócio. Para tal, este trabalho divide-se em mais 8 capítulos. O capítulo 2 apresenta uma elaboração sobre Arquiteturas de Software, seguido do capítulo 3, que trata de Arquiteturas em Camadas. O capítulo 4 apresenta a motivação para o desenvolvimento do âmbito deste trabalho, o qual é apresentado nos capítulos 5 e 6, respectivamente, A Arquitetura e A Metodologia propostas. O capítulo 7 apresenta Heurísticas de desenvolvimento, de acordo com a proposta apresentada nos capítulos anteriores. Os capítulos seguintes apresentam, respectivamente, Trabalhos Futuros e Conclusões.

2 Arquiteturas de Software

Com o crescimento em tamanho e em complexidade dos sistemas a serem desenvolvidos, as tarefas de Especificação e de Projeto tornam-se mais significativas que a escolha de algoritmos e de estruturas de dados.

As arquiteturas expressam esquemas de organização básicos para a estruturação de sistemas de software. Elas fornecem um conjunto de elementos organizados, especificam suas responsabilidades, e determinam regras e heurísticas para os relacionamentos entre esses elementos.

Um dos aspectos fundamentais para o sucesso do sistema a ser desenvolvido baseia-se na escolha de uma arquitetura apropriada, cuja descrição serve como um esqueleto sobre o qual o sistema será implementado.

A seguir, serão apresentadas as seções Escolha e Artefatos, elucidando, respectivamente, elaborações relativas a escolha de uma arquitetura, e aos artefatos proporcionados ao Projeto pela utilização de uma arquitetura.

2.1 Escolha

O entendimento detalhado de arquiteturas de software permite que desenvolvedores realizem escolhas específicas e justificadas dentre as diferentes alternativas de Projeto. A utilização de notações afins possibilita que desenvolvedores percebam paradigmas comuns, fazendo com que os relacionamentos de alto nível entre sistemas possam ser compreendidos, e novos sistemas possam ser construídos como variações de sistemas anteriormente desenvolvidos. Este fato é de grande importância, pois a maior parte dos sistemas desenvolvidos não são sistemas inovadores, mas sim sistemas similares aos desenvolvidos anteriormente.

Uma grande preocupação para gerentes de projetos de desenvolvimento de Software é em como fornecer sistemas de alta qualidade e durabilidade em um curto espaço de tempo

utilizando uma tecnologia atual. Nessa era de velocidade, de incerteza e de evolução das tecnologias, maximizar a eficiência da equipe de desenvolvimento é crucial para o sucesso da organização. O processo de escolha da arquitetura representa um dos pontos estratégicos para o nível gerencial do desenvolvimento de um sistema.

Diferentes arquiteturas são apropriadas para diferentes situações. A escolha da arquitetura mais apropriada para um problema ou domínio de problema ainda é uma questão em aberto. [Shaw96].

Para realizar a escolha de uma arquitetura adequada ao desenvolvimento de um sistema, devem ser analisados diferentes aspectos estruturais do sistema em questão. Exemplos destes aspectos são: comunicação de componentes e seus protocolos, possíveis aspectos de sincronização, aspectos de acesso a dados, associação de funcionalidades a componentes específicos do Projeto, composição de elementos de Projeto, distribuição física, escalabilidade, eficiência e aspectos de evolutibilidade e manutenibilidade. Após a análise destes e de outros aspectos pertinentes, deve ser realizada a escolha dentre várias alternativas de Projeto.

Muitos sistemas de software não podem ser estruturados com uma única arquitetura. Devem ser estruturados utilizando-se diferentes arquiteturas, nas quais partes do sistema são adequadamente resolvidas por arquiteturas específicas.

Arquiteturas podem ser organizadas de acordo com seu estilo, sendo que um estilo de arquitetura define uma família de um tipo de sistemas em termos de um padrão de organização estrutural. Mais especificamente, um estilo de arquitetura define um vocabulário de elementos e um conjunto de restrições para a combinação destes. Para muitos estilos, podem existir um ou mais modelos semânticos permitindo especificar como determinar as propriedades gerais do sistema em termos das propriedades específicas de suas partes. [Shaw96]

Como exemplos conhecidos de arquitetura de software, podemos destacar Arquitetura em Camadas, *Pipes and Filters*, *Blackboard*, *Broker*, *Model View Controller*, *Presentation Abstraction Control*, *Microkernel* e *Reflection*. [Buschman96]

2.2 Artefatos

Documentação

Além de especificar a estrutura e a topologia de um sistema, uma arquitetura mostra a correspondência entre os requerimentos do sistema e os elementos que, efetivamente, compõem o sistema a ser implementado, provendo, assim, uma documentação relativa ao processo de Projeto. Essa documentação inclui, por exemplo, aspectos de comunicação, estrutura do fluxo de controle, escalabilidade e possíveis pontos de evolução.

Tal documentação será de grande utilidade para futuras manutenções do sistema. A maior parte do tempo gasto ao realizar manutenção em sistemas é na compreensão do seu funcionamento. Esse esforço reduz-se, substancialmente, se a estrutura de Projeto do sistema estiver claramente explicada. Além disso, por estarem claras as decisões organizacionais realizadas pelo arquiteto do sistema, aqueles que realizarão a manutenção têm mais facilidade para preservar as características de Projeto, garantindo, assim, a integridade do sistema após as necessárias modificações.

Elementos e Interações

Modelos arquiteturais clarificam a separação estrutural e semântica entre seus elementos e suas interações. Elementos podem ser entidades computacionais como clientes, servidores, Base de Dados, filtros e camadas. Interações entre elementos podem ser simples como chamadas a procedimentos, funções e compartilhamento de variáveis, ou complexas e semanticamente ricas como protocolos de comunicação entre clientes e servidores, ou APIs de acesso a Bases de Dados.

Modelos arquiteturais podem ser compostos para definir sistemas maiores e mais complexos. Idealmente, os elementos que compõem uma arquitetura são definidos independentemente, possibilitando seu uso em diferentes contextos. As diferentes arquiteturas estabelecem especificações para seus elementos, os quais podem ser envolvidos em outras arquiteturas.

Notação

Uma arquitetura, em geral, é representada por uma notação gráfica, que provê um meio eficiente de comunicação entre desenvolvedores, sendo eles de uma mesma equipe de desenvolvimento ou não. Do mesmo modo que existe uma linguagem específica para projetos de Engenharia Civil e Elétrica, a Engenharia de Software pode beneficiar-se do uso de uma linguagem comum, na qual decisões relativas à construção de um sistema podem ser analisadas, discutidas, documentadas, e até formalizadas pela equipe de desenvolvimento.

3 A Arquitetura em Camadas

A Arquitetura em Camadas ajuda a estruturar aplicações que podem ser decompostas em grupos de subtarefas, nas quais cada grupo encontra-se em um determinado nível de abstração. [Buschman96]

A Arquitetura em Camadas estrutura o sistema em um número apropriado de camadas, sendo que uma camada é colocada sobre outra até que seja alcançado o maior nível de abstração do sistema. A Figura 3-1, abaixo, ilustra uma arquitetura em camadas na qual o menor nível de abstração é representado pela camada 1, e o maior nível de abstração é representado pela camada N. A camada J representa uma camada intermediária do sistema que oferece serviços a camadas superiores e requer serviços de camadas inferiores.

Esta arquitetura apresenta uma importante característica: os elementos de uma camada trabalham dentro de um mesmo nível de abstração, e um elemento de uma camada não pode pertencer a outras.

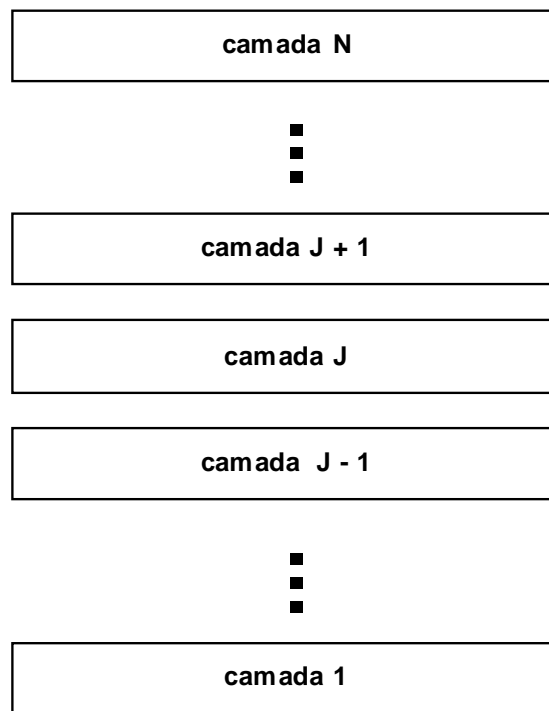


Figura 3-1 Arquitetura em Camadas

Um caso particular da Arquitetura em Camadas ocorre quando cada camada intermediária provê serviço a sua camada imediatamente superior e requer serviço da imediatamente inferior. Neste caso, mudanças em uma camada poderão afetar somente suas camadas adjacentes.

O desenvolvimento de sistemas em camadas utiliza um projeto baseado em níveis de abstração, permitindo que desenvolvedores repartam um problema complexo em uma seqüência de etapas incrementais e de menor complexidade que o problema original. Esses sistemas suportam futuras melhorias e aprimoramentos dentro de cada nível de abstração, os quais podem ser realizados sem causar interferência nos diferentes níveis de abstração. Essa independência entre os níveis de abstração é atingida com a organização proposta pela arquitetura.

Sistemas em camadas facilitam o reuso. Como tipos abstratos de dados, permitem que diferentes implementações de uma mesma camada sejam utilizadas em momentos diferentes, para uma mesma posição, desde que sejam respeitadas as interfaces com as

camadas adjacentes. Tal fato leva a possibilidade de definição de interfaces padrão para camadas, possibilitando o desenvolvimento de diferentes implementações. (um bom exemplo disso é o padrão ISO na arquitetura de redes [ISO92]).

Sistemas em camadas também possuem desvantagens. Nem todos os sistemas são estruturados em camadas de forma coerente. Mesmo sendo o sistema estruturado logicamente em camadas, considerações de desempenho podem levar a um acoplamento maior entre o alto nível de definição do sistema e o baixo nível de implementação.

Assim como Orientação a Objetos apresenta o benefício de encapsular a implementação de suas classes através de métodos que são executados sobre dados internos às classes, a Arquitetura em Camadas, em um nível superior de abstração, realiza o encapsulamento da implementação de suas camadas através de interfaces que operam sobre entidades internas às camadas.

3.1 Arquitetura em 2 Camadas

Uma arquitetura amplamente utilizada na Engenharia de Software é a Arquitetura em 2 Camadas, cujo desenvolvimento é separado em duas partes: Interface e Base de Dados.

Em um Sistema Cliente/Servidor em 2 Camadas, a lógica da aplicação encontra-se dividida entre o Cliente, embutida na Interface, e o Servidor, embutida na Base de Dados (Figura 3-2). A interface, executada no Cliente, por exemplo, envia comandos SQL, chamadas do sistema, ou comandos HTTP para o Servidor. O Servidor processa o pedido e retorna o resultado. Nessa configuração, para o Cliente acessar os dados, ele precisa saber como eles estão organizados e armazenados no lado Servidor. Uma alternativa para este cenário será o Cliente invocar “*stored procedures*”, funções que serão executadas no Servidor dentro da Base de Dados, diminuindo o processamento no lado Cliente.

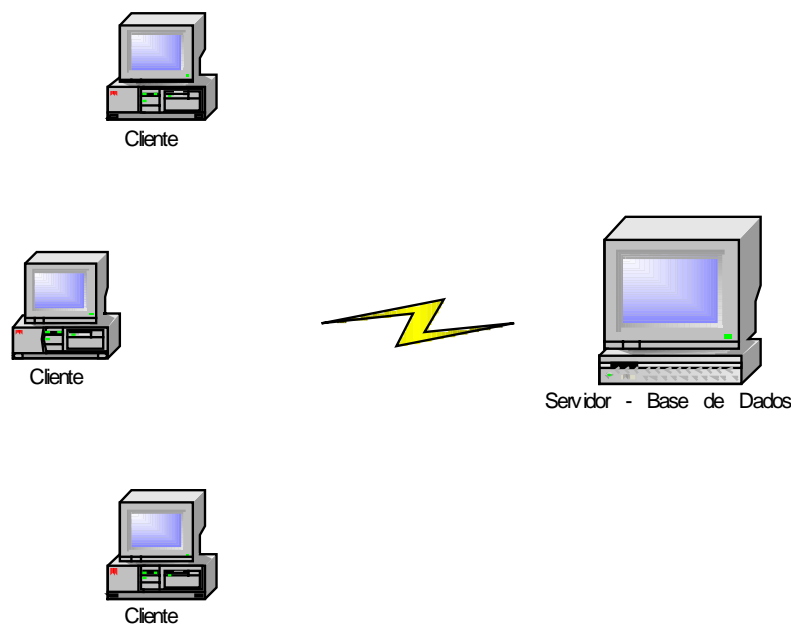


Figura 3-2 Sistemas em 2 Camadas

O principal fator para a popularidade da Arquitetura em 2 Camadas é a simplicidade de seu desenvolvimento. O surgimento de linguagens de quarta geração e seus ambientes de construção visual, como *Delphi*, *Visual Basic*, *Power Builder*, permitem o rápido desenvolvimento de pequenas aplicações Cliente/Servidor. Entretanto, quando estas aplicações crescem em tamanho e em complexidade, a Arquitetura em 2 Camadas deixa a desejar, especialmente se usada com um modelo de objetos.

A modelagem Orientada a Objetos enfatiza, naturalmente, aspectos do sistema relacionados a regras de negócio, entretanto, na Arquitetura em 2 Camadas, são implementadas, separadamente, Interface e Base de Dados, ficando a lógica da aplicação dividida entre estas camadas. Com essa separação, toda a modelagem Orientada a Objetos é transformada, por um lado, em modelagem típica de Banco de Dados, como Modelos de Entidades e Relacionamentos em Bases de Dados Relacionais, e por outro lado, em modelos navegacionais para o desenvolvimento de interfaces. Toda informação intrínseca à lógica do sistema modelado é distribuída entre as duas camadas sem um critério bem definido. Futuras evoluções, tanto diretas na implementação do sistema, como no modelo, dificilmente são mapeadas entre si.

O maior problema enfrentado pela Arquitetura em 2 Camadas é causado pela volatilidade das regras de negócio de um sistema. Dada a necessidade de constantes mudanças das regras de negócio, estas merecem um tratamento específico e independente da Interface e da Base de Dados. Como nesta arquitetura este tratamento não é realizado de forma explícita, esta não apresenta uma boa escolha para lidar com aspectos de manutenibilidade e extensibilidade de sistemas.

3.2 Arquitetura em 3 Camadas

O termo Arquitetura em 3 Camadas foi introduzido no início da década de 80 para descrever a partição física de uma aplicação entre terminais (camada 1), micro computadores (camada 2) e *mainframes* (camada 3). [Edwards97]. Atualmente, este termo é utilizado para descrever a arquitetura em camadas de desenvolvimento de sistemas que realiza uma estruturação lógica da divisão de tarefas em 3 partes: clientes executando lógica de interface, a aplicação servidora executando a lógica do sistema, e Base de Dados ou aplicações já existentes responsáveis pelo processamento relativo à persistência (Figura 3-3). Estas 3 partes são denominadas, respectivamente, Camada de Interface, Camada de Negócio e Camada de Persistência.

Na Arquitetura em 3 Camadas, a Interface interage com a Persistência através da Camada de Negócio, na qual reside a lógica da aplicação. Nesta arquitetura, os processos que controlarão a aplicação são executados separadamente da Interface e da Base de Dados. A lógica da aplicação pode ser executada em um ou mais Servidores.

A Arquitetura em 3 Camadas é uma área em evolução para aplicações Cliente/Servidor. Essa arquitetura é particularmente adequada para aplicações Cliente/Servidor na Internet ou Intranets. O gerenciamento é mais simples pela separação de funcionalidades, e a aplicação pode ser facilmente distribuída pela rede de acordo com suas camadas. A divisão de responsabilidades permite a criação de Clientes enxutos, melhorando seu desempenho principalmente ao serem “baixadas” pela Internet.

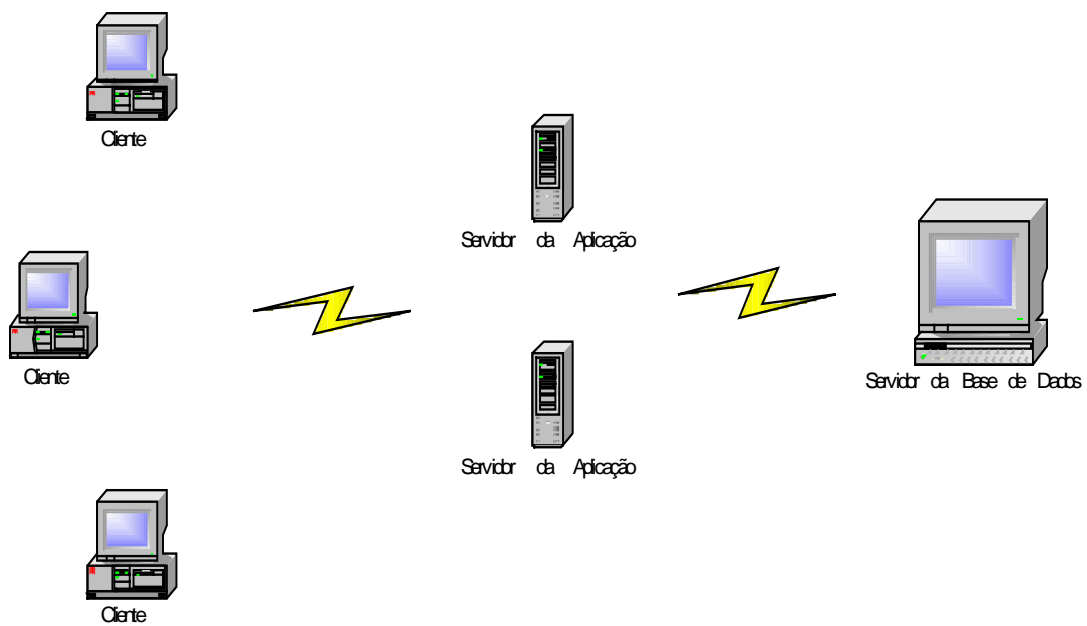


Figura 3-3 Sistemas em 3 Camadas

O aspecto de segurança possui um melhor tratamento na Arquitetura em 3 Camadas, uma vez que o esquema da Base de Dados não é exposto ao Cliente. A Camada de Negócio também pode realizar um melhor tratamento para diferentes níveis de autorização para diferentes Clientes.

A Arquitetura em 3 Camadas realiza uma solução mais satisfatória e complementar a uma modelagem baseada no paradigma de Orientação a Objetos, uma vez que esta modelagem pode ser mapeada para a Camada de Negócio, local específico para o tratamento da lógica da Aplicação.

4 Motivação

Dadas as considerações dos capítulos anteriores, deseja-se aqui explicitar os objetivos desta dissertação. Para tal, este capítulo foi estruturado em duas partes: o problema e a solução proposta.

4.1 O Problema

Dada a importância da utilização de uma arquitetura para o desenvolvimento de sistemas de software, a Arquitetura em 3 Camadas apresenta uma boa solução para sistemas que se dividem naturalmente em Interface, Negócio e Persistência. Entretanto, importantes aspectos devem ser analisados sob a perspectiva da utilização da mesma.

Um problema prático que o desenvolvimento de Sistemas em 3 Camadas enfrenta é que existem boas ferramentas para o desenvolvimento de interfaces gráficas, para o armazenamento de dados, e para pequenas aplicações; entretanto, não existem ferramentas específicas que auxiliem no desenvolvimento da Camada de Negócio. Um provável resultado desta carência seria a migração das regras de negócio para a Camada de interface e/ou de Persistência. Essa migração caracteriza uma desestruturação da Camada de Negócio e poderá reduzir o sistema a um Sistema em 2 Camadas, com todas as desvantagens inerentes a essa arquitetura.

A Arquitetura em 3 Camadas vem sendo utilizada por muitos anos. Apesar da sua longa existência, ainda se mostra necessária a definição de termos homogêneos para descrever cada uma das camadas, explicitando suas responsabilidades. Além da estruturação entre as camadas, os aspectos relacionados à comunicação entre elas precisam ser tratados. Outro problema enfrentado pela Arquitetura em 3 Camadas ocorre quando aparecem entidades que não se adequam a nenhuma das camadas disponíveis, ou que são utilizadas por mais de uma camada. O surgimento destas entidades, que fogem à semântica de um Sistema em Camadas, mostra que a Arquitetura em 3 Camadas deve ser acrescida de outras soluções arquiteturais que solucionem tal problema.

As arquiteturas de software encontradas na literatura [Buschman96], [Shaw96], apresentam em suas descrições um enfoque da sua utilização por parte de projetistas e desenvolvedores. Porém, sistemas grandes e complexos precisam ser gerenciados. A Arquitetura de Software representa uma ferramenta fundamental para o acompanhamento, a organização, a medição e a projeção do desenvolvimento de sistemas. A arquitetura deve apresentar uma solução eficiente não apenas para os desenvolvedores, mas para todos os envolvidos no desenvolvimento do sistema.

A escolha de uma arquitetura faz parte do Projeto de um sistema; entretanto, somente a utilização de uma arquitetura não garante o sucesso do desenvolvimento de Sistemas de Software. Segundo [Johnson95], um terço de todos os projetos de software são cancelados e cinco dentre seis são mal sucedidos. A principal causa destes casos de insucesso são erros de Projeto.

De acordo com [Scott97], Projeto é a segunda atividade mais importante na engenharia de software; sendo a primeira a coleta de requisitos. Enquanto a coleta de requisitos diz o que deve ser realizado, o Projeto contém todas as informações sobre como será realizado. Com um Projeto bom e completo, a tarefa de codificação pode ser simplificada e o acompanhamento do desenvolvimento pode ser mais eficiente.

O Projeto não é apenas uma hipótese sobre como entidades se comportam no mundo real, mas sim uma hipótese sobre como uma organização de componentes comportar-se-á no mundo criado pelo projetista. [Petroski85]. A característica básica para o conjunto de componentes do Projeto é que, juntos, eles implementam todas as propriedades representadas pelas abstrações apresentadas em um Modelo de Análise.

Em [Shaw96], mostra-se que diferentes formas de solução de um mesmo problema levam a diferentes Projetos, os quais utilizam arquiteturas com propósitos significativamente distintos. [Sharble93] mostra esse fenômeno bem claramente quando utiliza duas formas distintas de solução para o mesmo conjunto de requisitos.

Para um sistema em desenvolvimento, sempre haverá objetivos e restrições conflitantes. Por isto, a escolha de um Projeto torna-se uma tarefa de otimização dos objetivos a serem atingidos. Um importante ponto a ser enfatizado é que a utilização de cada solução de Projeto resulta em um modelo completo das dimensões de estrutura, funcionalidades e comportamento de um sistema. Cada solução de Projeto enfatiza diferentes aspectos do problema a ser solucionado.

4.2 A Solução Proposta

A solução proposta tem o objetivo de simplificar e de estruturar a implementação de Sistemas em 3 Camadas, ou seja, sistemas que tenham Interface, Negócio e Persistência. Para atingir tal finalidade, a solução apresentada divide-se, basicamente, em uma arquitetura e uma metodologia.

A Arquitetura propõe uma organização para o desenvolvimento de Sistemas em 3 Camadas, explicitando as entidades envolvidas na Arquitetura, suas responsabilidades, e os aspectos de comunicação entre elas. Esta arquitetura está apresentada no capítulo 5.

A Metodologia propõe uma solução de Projeto para o desenvolvimento da Camada de Negócio em Sistemas em 3 Camadas. Esta metodologia segue o paradigma de Orientação a Objetos e é complementar à utilização da Arquitetura. A Metodologia encontra-se apresentada no capítulo 6.

Além da apresentação da Arquitetura e da Metodologia, serão introduzidas, no capítulo 7, algumas Heurísticas, representando dicas de desenvolvimento para a utilização da Arquitetura e da Metodologia propostas. Considerações finais são discutidas nos capítulos 8 e 9, respectivamente, Trabalhos Futuros e Conclusões.

5 A Arquitetura

A Arquitetura é uma fusão de arquiteturas ou de padrões arquiteturais. Ela é uma combinação da Arquitetura em 3 Camadas, *Brokers* [Buschman96][Shaw96]. e um Repositório de Classes.

Sua estrutura básica é a Arquitetura em 3 Camadas, contendo as camadas de Interface, Negócio e Persistência. *Brokers* encapsulam as comunicações entre camadas, modelando as intercamadas Interface/Negócio e Negócio/Persistência. O Repositório de Classes, denominado Módulo de Acesso, agrupa entidades que não podem ser naturalmente alocadas a uma única camada.

A figura, abaixo, ilustra a Arquitetura:

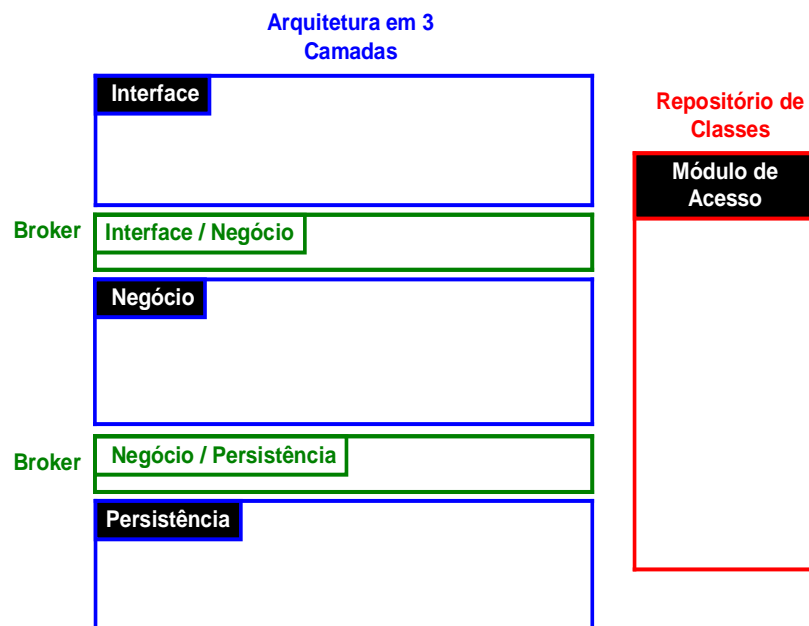


Figura 5-1 A Arquitetura

Para fins desta dissertação, a Arquitetura é utilizada dentro de um paradigma de Orientação a Objetos. Assim sendo, as camadas, as intercamadas e o Módulo de Acesso são compostos por classes. Para utilizar esta arquitetura em outros paradigmas, é necessário realizar uma adaptação de sua estrutura para tal paradigma.

A Figura 5-2, abaixo, objetiva ilustrar o fluxo de dados na Arquitetura a partir da entrada de dados na interface. Os dados submetidos à Camada de Interface são transformados em objetos na Intercamada Interface/Negócio; desta seguem para a Camada de Negócio, onde regras de negócio os manipulam. Caso seja necessário realizar a persistência desses objetos, eles seguem para a Intercamada Negócio/Persistência, onde são transformados nos respectivos tipos de dado persistentes, e armazenados, fisicamente, na Camada de Persistência. O caminho oposto ocorre quando a Camada de Interface necessita alguma informação persistente.

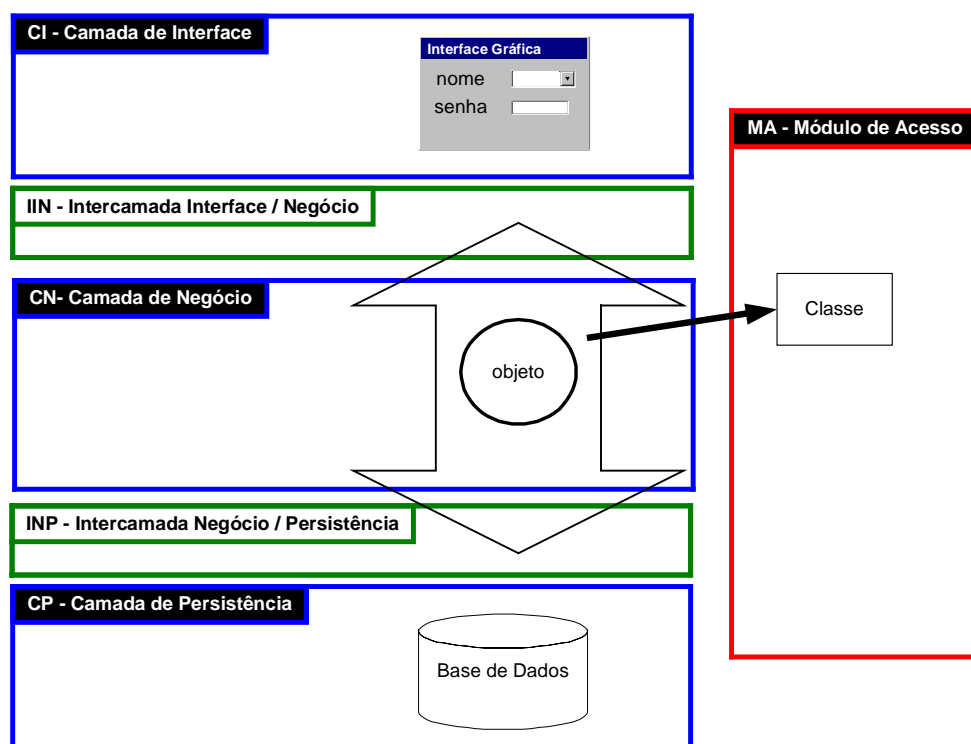


Figura 5-2 Arquitetura em tempo de execução

Este capítulo está estruturado em quatro seções : Elementos, Visibilidade, Colaborações e Avaliação da Arquitetura.

5.1 Elementos da Arquitetura

Esta seção apresenta uma descrição de cada elemento participante da Arquitetura, conforme ilustrado na Figura 5-1 acima. São eles :

- Camada de Interface (CI);
- Intercamada Interface/Negócio (IIN);
- Camada de Negócio (CN);
- Intercamada Negócio/Persistência (INP);
- Camada de Persistência (CP);
- Módulo de Acesso (MA).

5.1.1 Camada de Interface (CI)

Responsabilidades

A CI é responsável somente pela interface com o usuário. Ela trata, por exemplo, de janelas, menus, botões, fontes e outras entidades da interface propriamente dita. Normalmente, utiliza-se algum *framework* ou biblioteca de classes para a construção de interface, como por exemplo, *MFC*, *MacApp*, ou *Java Servlets*. Essa camada não realiza nenhum cálculo, nem acessa diretamente a CN ou a CP. Ela executa tão somente o processamento típico da interface, por exemplo, o preenchimento de campos obrigatórios e a navegabilidade entre telas.

Colaboradores

A IIN colabora com a CI. Para realizar a interface do sistema, a CI não precisa conhecer as classes da CN ou da CP, como elas estão implementadas, ou são executadas. A CI somente acessa a IIN.

5.1.2 Intercamada Interface/Negócio (IIN)

Responsabilidades

Essa Intercamada é responsável pelo atendimento à CI, sendo responsável também por todos os acessos à CN, e por todo o processamento necessário para permitir a comunicação entre a CI e a CN.

Essa Intercamada possui um *Broker* responsável pela seleção de informações da CN e pela disponibilização no formato requisitado pela CI. Assim sendo, toda manipulação com a CN é transparente para a CI e vice-versa.

A IIN é responsável pela conversão de tipos de dados entre a CI e a CN. Como exemplo da necessidade dessa conversão, podemos citar uma interface para sistemas na Internet que utilize classes *Java servlets*. Essas classes da CI comunicar-se-ão através de um *Broker* da IIN com classes da CN. Ao receber um requerimento da CI, essa intercamada acessa funcionalidades da CN. Os resultados desses acessos podem ser objetos modelados por classes do MA. As classes *Java Servlets* da CI não compreendem esses objetos, somente compreendem *strings*. É responsabilidade dessa intercamada converter esse objeto para os respectivos *strings* requisitados pela CI.

Colaboradores

Colaboram com esta intercamada a CN e o MA.

A CN realiza a implementação de todas as funcionalidades do negócio do sistema que suprirão os requerimentos da IIN, gerados pela CI.

Essa intercamada tem acesso ao MA, do qual ela pode acessar dados comuns à CN, podendo, então, realizar as conversões necessárias entre a CI e a CN.

5.1.3 Camada de Negócio (CN)

Responsabilidades

Esta camada é responsável pela execução da semântica do sistema, independente da localização física, da estrutura dos dados armazenados e de suas interfaces de acesso. Ela é responsável também pela realização de toda a lógica do sistema e pelo controle da execução das regras de negócio.

A CN modela o comportamento das entidades envolvidas na Aplicação. Essa camada modela as regras e os processos que operam sobre estas entidades. O foco está na forma com a qual estas entidades reagem a essas regras, e não na informação sobre cada entidade em particular. Estas entidades manipuladas por esta camada são Classes de Acesso do MA, que serão descritas mais adiante.

Colaboradores

A INP e o MA colaboram com esta camada.

Todos os acessos que a CN precisa realizar para a CP são tratados pela INP, que realiza a independência entre a CN e a CP.

Essa camada tem acesso ao MA, que contém as classes modelando os objetos necessários a CN.

5.1.4 Intercamada Negócio/Persistência (INP)

Responsabilidades

Essa Intercamada, organizada como *Broker*, é responsável pela manipulação de dados da CP de acordo com os requerimentos da CN. Esse *Broker* pode realizar desde simples

comandos SQL em Base de Dados Relacionais [JDBC] até complexos *frameworks* de manipulação de dados em Banco de Dados Heterogêneos [Uchôa99-1] [Uchôa99-2].

A INP permite o desacoplamento entre o modelo físico da CP e o modelo lógico da CN. Para isso, ela realiza a adaptação entre esses dois modelos, ou seja, ela deve realizar toda a conversão de tipo de dados entre a CN e a CP. Um exemplo da solução realizada por essa Intercamada seria o mapeamento de objetos para tabelas de Base de Dados Relacional.

O mapeamento entre objetos e Base de Dados Relacionais é assunto de vários pesquisadores envolvidos na área da Engenharia de Software. Essa intercamada pode compreender desde soluções mais simples [Vianna97] até soluções mais complexas para resolver tal problema [Keller98] [PERSISTENCE BUILDER].

Colaboradores

A CP e o MA colaboram com essa intercamada.

As Bases de Dados encontradas na CP realizam os requerimentos para persistência recebidos por essa intercamada.

Essa intercamada acessa o MA, que contém as classes modelando os objetos comuns a várias camadas.

5.1.5 Camada de Persistência (CP)

Responsabilidades

Essa camada é responsável pelo armazenamento físico dos dados. Esses dados podem estar armazenados em Bases de Dados Relacionais, Bases de Dados Orientadas a Objetos, Sistemas Legados, Arquivos Texto, Banco de Dados Heterogêneos, etc.

No caso de Sistemas Legados, estes representarão uma CP e serão acessíveis através de uma INP. A Figura 5-3 ilustra um sistema que necessita obter dados de um Sistema Legado. Essa solução pode ser utilizada para realizar reengenharia de projetos, no qual um Sistema Legado deve continuar funcionando juntamente com o novo sistema.

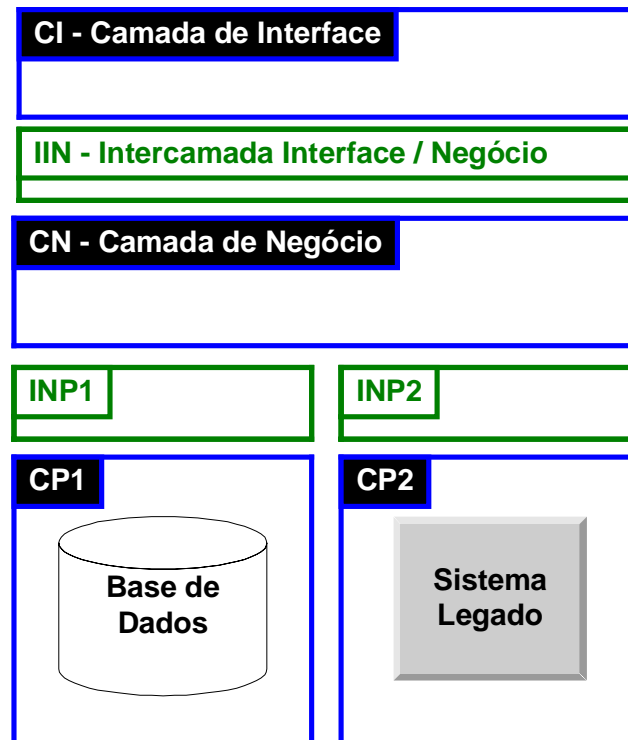


Figura 5-3 Sistema Legado tratado como CP.

Colaboradores

O MA colabora com essa camada. As classes do MA podem ser utilizadas para a geração dos esquemas de armazenamento físico dos dados. Pode ser gerado, por exemplo, um mapeamento entre as classes do MA e as tabelas de um Banco de Dados Relacional [Rumbaugh91] [PERSISTENCE BUILDER].

5.1.6 Módulo de Acesso (MA)

Toda vez que em um Projeto aparecerem classes necessárias a distintas camadas, o repositório destas classes será o MA. As classes encontradas no MA são denominadas Classes de Acesso.

As Classes de Acesso modelam entidades, ou objetos do mundo real, que são representadas nas Aplicações em termos de suas características descritivas e do relacionamento entre elas. O capítulo seguinte, na seção 6.2.1 - Definição do Módulo de Acesso, apresenta um tratamento mais específico em relação à definição dessas classes.

As Classes de Acesso são responsáveis pelo encapsulamento dos dados necessários para a realização da comunicação entre as camadas. Elas apresentam atributos privados que somente são acessíveis através de métodos de acesso (*get* e *set*). A Figura 5-4, abaixo, apresenta um exemplo de uma Classe de Acesso.

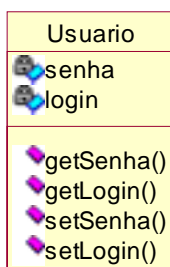


Figura 5-4 Classe de Acesso Usuário

Classes do MA modelam os objetos participantes do fluxo de dados que ocorre entre a CI e a CP, conforme ilustrado na Figura 5-2.

5.2 Visibilidade da Arquitetura

Esta seção apresenta um detalhamento dos aspectos de visibilidade entre os elementos da Arquitetura.

Nessa arquitetura, a visibilidade entre as camadas ocorre da camada superior para a inferior. Essa regra de visibilidade permite definir claramente os protocolos de comunicação entre elas. Cada camada define sua interface de utilização disponível para a camada superior. Dessa forma, cada uma delas, exceto a primeira e a última, presta serviços para a imediatamente superior e realiza pedidos para a imediatamente inferior através da interface disponibilizada por essa camada.

A estrutura interna de implementação de cada camada não é visível para camadas superiores. Alterações internas a uma camada podem ser realizadas desde que a interface disponibilizada seja mantida. Segundo [Brown98], Interfaces estáveis possibilitam o desenvolvimento em paralelo, uma documentação efetiva e reduz a obsolescência do software.

A Figura 5-5, a seguir, exemplifica a comunicação entre duas camadas através da interface disponibilizada pela camada inferior.

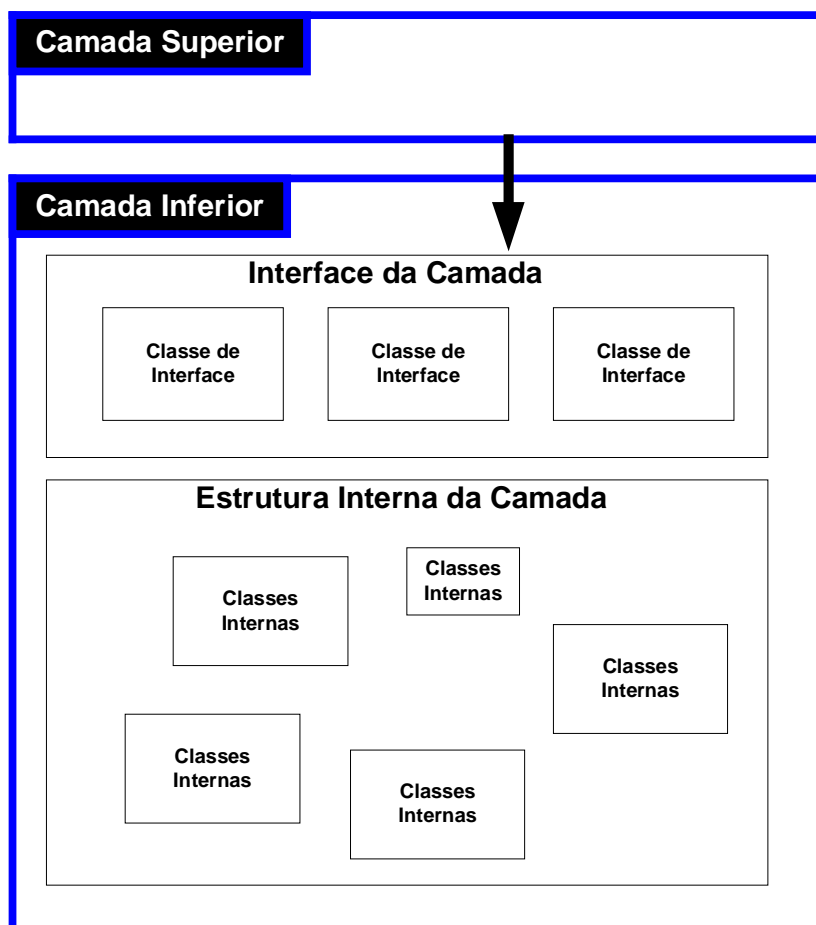


Figura 5-5 Interface de uma Camada

Esse esquema de visibilidade isola a CN da CI através da IIN. A realização da CN é independente das CIs que a utilizam. A IIN realiza a comunicação entre uma CI específica e sua CN. Um exemplo da utilização desse isolamento é um sistema que possui duas interfaces, uma para a *Internet* e outra *Windows*. Nesse caso, a mesma CN, que implementa o negócio do sistema, seria acessada por duas CIs diferentes através de IINs específicas para cada CI. Não ocorre retrabalho por parte da CN para disponibilizar o sistema para uma nova interface.

Analogamente ao isolamento obtido entre a CN e a CI, esse esquema de visibilidade da Arquitetura também isola a CP da CN através da INP. A CP é realizada independente da CN que a utiliza. Uma mesma CP pode ser utilizada por CNs diferentes, através das INPs

específicas. Da mesma forma, uma CN pode utilizar diferentes CPs através das INPs específicas.

O MA é visível para todas as camadas. Cada camada pode utilizar as Classes de Acesso disponíveis no MA. A Figura 5-6, abaixo, ilustra este aspecto de visibilidade entre as camadas e o MA.

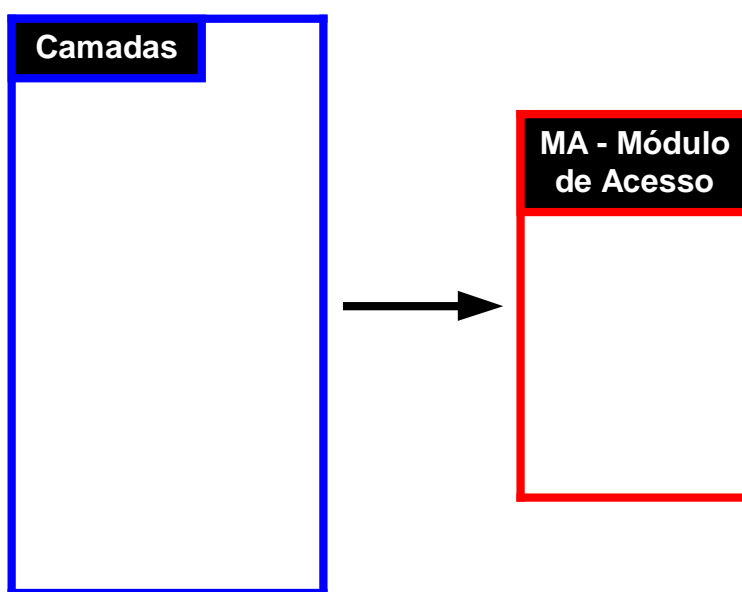


Figura 5-6 Visibilidade entre Camadas e Módulo de Acesso

Caso alguma Classe de Acesso, disponível para todas camadas, precise ser estendida para suprir uma necessidade específica de uma camada, isto deve ser realizado localmente na camada, sem que altere sua utilização em outras camadas.

A Figura 5-7, abaixo, exemplifica essa situação e uma possível solução através de uma especialização por Herança. O exemplo apresenta uma Classe de Acesso Pessoa, para a qual um dos atributos é o número de CPF. A CI pode necessitar de uma verificação simples sobre um número de CPF, como por exemplo, se ele possui a regra de formação correta. Para suprir tal necessidade, a CI pode redefinir a classe Pessoa, inserindo a funcionalidade para verificar a corretude do número de CPF. A Classe de Acesso Pessoa,

necessária para a comunicação com as outras camadas, continua inalterada, contendo apenas as informações necessárias para todas as camadas.

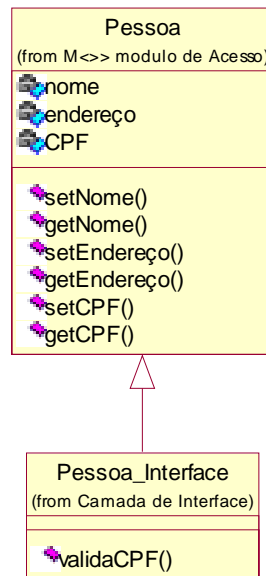


Figura 5-7 Exemplo de especialização de uma classe de Acesso na camada de Interface.

5.3 Colaborações na Arquitetura

Esta seção apresenta uma elucidação das colaborações envolvidas na Arquitetura.

De acordo com a Arquitetura, existem três formas possíveis de colaboração entre seus elementos: a colaboração Inter-Camadas, a Intra-Camada e a MA-Camadas. As figuras (Figura 5-8 e Figura 5-9), abaixo, ilustram as colaborações e os respectivos elementos envolvidos na Arquitetura.

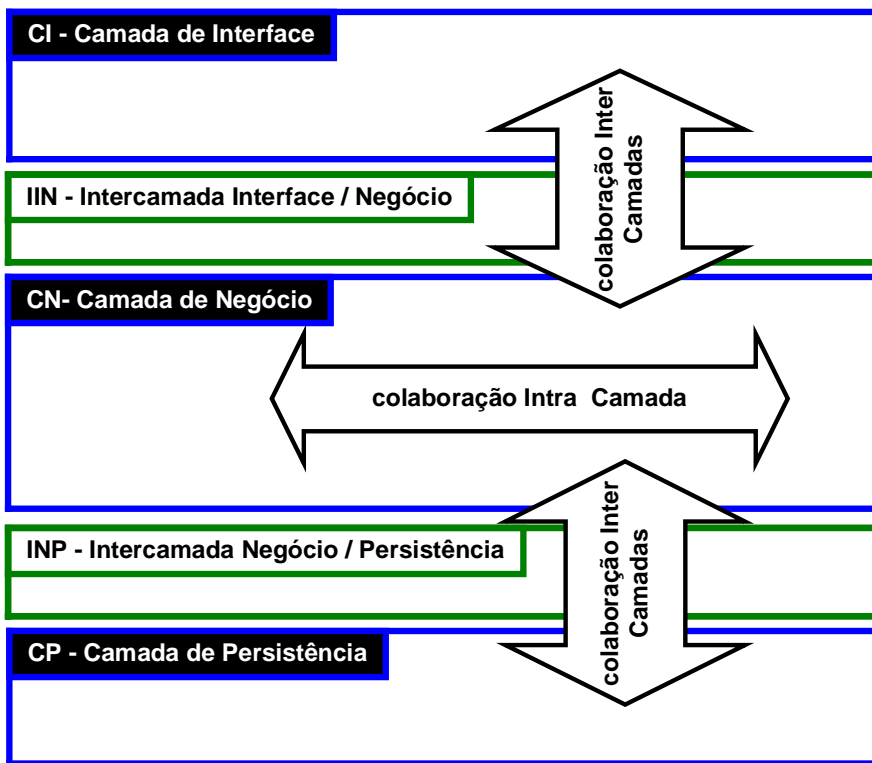


Figura 5-8 Colaborações Inter-Camadas e Intra-Camada.

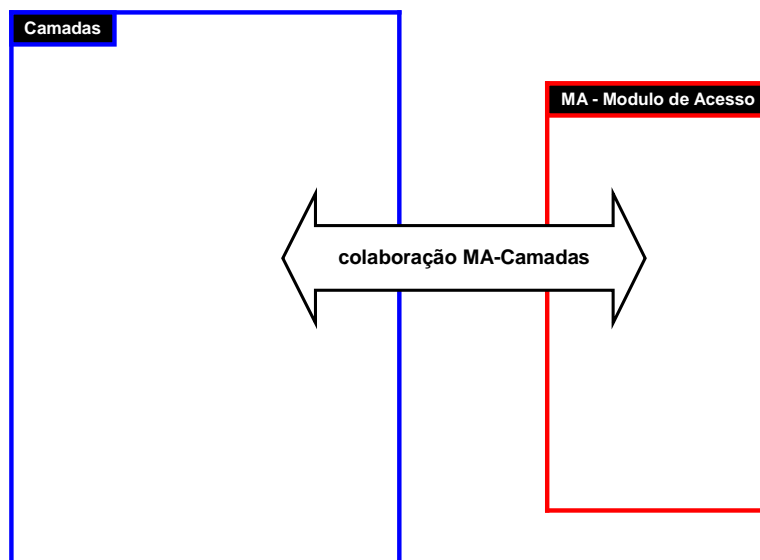


Figura 5-9 Colaboração MA-Camadas.

A colaboração Intra-Camada é aquela que ocorre dentro de cada camada. A especificação da CI e da CP não faz parte do escopo desta dissertação; entretanto, a colaboração que ocorre dentro da CN será assunto do capítulo seguinte.

A colaboração Inter-Camadas é aquela entre camadas e que ocorre em dois momentos distintos. No primeiro momento, ela ocorre entre a CI e a CN, passando pela IIN. No segundo momento, a colaboração ocorre entre a CN e a CP, passando pela INP.

A Figura 5-10, abaixo, ilustra o primeiro momento, no qual um objeto de uma Classe de Acesso, do MA, é instanciado na IIN com os dados coletados da CI. Esse objeto é passado para a CN como parâmetro.

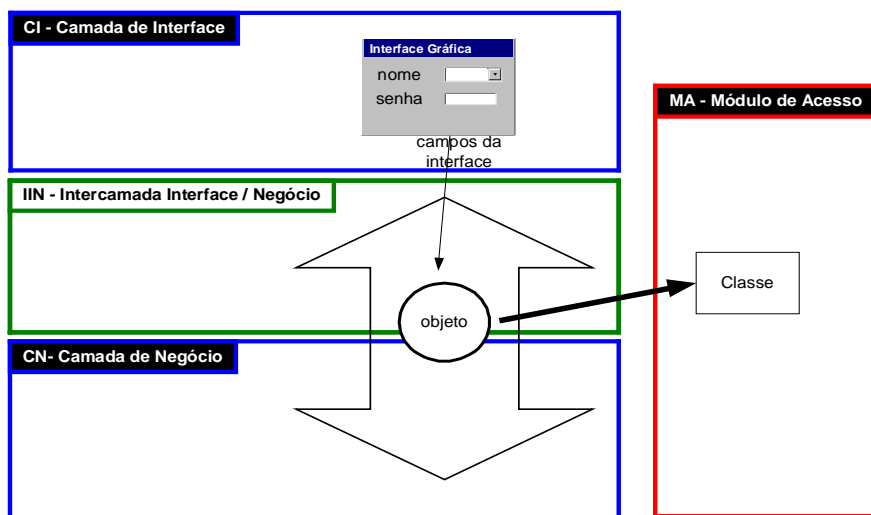


Figura 5-10 Comunicação Interface ↔ Negócio.

A figura 5.11 ilustra o segundo momento, no qual o objeto da Classe de Acesso é passado da CN para a INP, onde é transformado e persistido no esquema de persistência específico da CP.

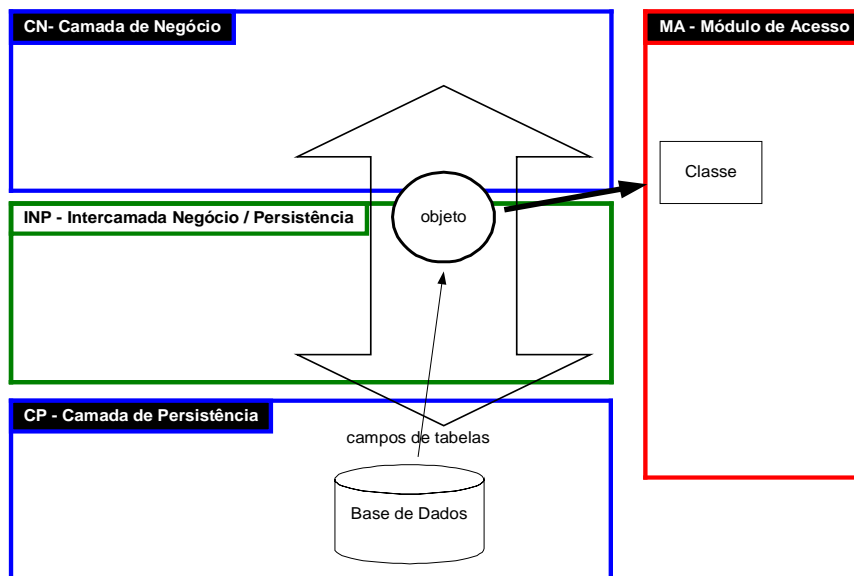


Figura 5-11 Comunicação Negócio ↔ Persistência.

Para ambos os casos descritos, a colaboração entre as camadas pode ocorrer em dois sentidos, seja da CI até atingir a CP, ou no sentido oposto, indo desde a CP até atingir a CI.

A colaboração MA-Camadas é aquela que ocorre entre o MA e as camadas sempre que é necessário que elementos participantes de uma camada precisem fazer referência a dados do MA.

5.4 Avaliação da Arquitetura

Esta seção apresenta uma avaliação da Arquitetura, discutindo aspectos de desenvolvimento e de gerência.

5.4.1 Aspectos de Desenvolvimento

Escalonabilidade. A Arquitetura apresenta, naturalmente, a facilidade de escalonar uma aplicação. A organização em camadas apresenta uma separação lógica entre as entidades envolvidas no funcionamento da Aplicação; e com isso, o problema pode ser

escalonado de acordo com suas funcionalidades específicas de interface, de negócio ou de persistência.

Escalabilidade. A Arquitetura apresenta um alto nível de escalabilidade. Aplicações que possuem interface, negócio e persistência apresentam o mesmo tratamento na Arquitetura, independente de seu tamanho; com este tratamento uniforme, o crescimento em tamanho da aplicação que está sendo desenvolvida representa um aumento linear em cada uma das entidades pertencentes à Arquitetura.

Balanceamento de cargas. Essa arquitetura possibilita uma melhor distribuição do processamento através dos níveis lógicos das camadas e das intercamadas. A separação em camadas possibilita uma melhor distribuição da carga de processamento, o que, atualmente, é de grande importância para o desenvolvimento de sistemas para a *Internet*. O maior problema enfrentado por *Applets*, ao serem executados nos *Browsers*, era o tempo que levavam para serem “baixados”. Uma solução para este problema foi a redução do tamanho do *Applet*, levando parte dessa carga de processamento para a Camada de Negócio.

Banco de Dados Heterogêneos. Uma CN pode utilizar diferentes CPs. Banco de Dados Heterogêneos, representando diferentes CPs, podem ser utilizados pela CN independente de suas particularidades. As INPs são responsáveis pela disponibilização das diferentes CPs para a CN. Uma solução para Banco de Dados Heterogêneos é realizada em [Uchôa99-1][Uchôa99-2]. Seu trabalho apresenta uma solução específica tratando da CN, das INPs e das CPs.

Distribuição. A Arquitetura apresenta um elevado nível de encapsulamento em camadas e em classes dentro de cada camada. A distribuição na aplicação pode ser inserida entre as camadas. Por exemplo, para permitir que clientes distribuídos da CI acessem a CN, os aspectos de distribuição podem ser inseridos na IIN. A Distribuição também pode ser realizada internamente em uma camada, por exemplo, a CP pode apresentar suas Bases de Dados distribuídas, sem afetar o funcionamento da CN. As INPs serão responsáveis pela realização da comunicação com as CPs distribuídas.

Eficiência. Um dos maiores problemas que a divisão em camadas enfrenta é em relação à eficiência. A divisão em camadas apresenta uma solução menos eficiente que um sistema monolítico. Esse é o preço que se paga por melhorar os aspectos relativos à manutenibilidade, à flexibilidade e à organização do desenvolvimento. A utilização da Arquitetura ocasiona um aumento do número de camadas, e conseqüentemente, de classes envolvidas na execução da Aplicação. Esse aumento de classes representa um aumento no caminho que uma funcionalidade deve seguir desde a interface até a persistência. Este aumento do número de classes é um aumento linear e que não compromete a complexidade dos algoritmos envolvidos no sistema [Cormen90].

A eficiência pode ser melhorada através de um otimizador que trabalharia sobre a estrutura organizada em camadas, e geraria um código otimizado. Essa solução manteria a vantagem da separação lógica em camadas, mas, em um segundo momento, a geração de código seria otimizada atingindo níveis próximos a um sistema monolítico.

A organização em camadas, mesmo aumentando o número de classes e a execução envolvida, pode gerar soluções mais eficientes para casos específicos, como no desenvolvimento de aplicações Cliente/Servidor para sistemas na Internet. A divisão em camadas possibilita um aumento da eficiência de um sistema, já que possibilita a construção de clientes “enxutos”. Em um sistema convencional, sem a divisão em camadas, basicamente, o cliente a ser “baixado” pela Internet apresentava a ineficiência relacionada ao seu tamanho.

Flexibilidade. A Arquitetura apresenta a flexibilidade como um dos aspectos prioritários. As camadas definem suas interfaces de utilização para a camada superior. As camadas são construídas independentemente, e suas estruturas internas podem ser alteradas, desde que sejam mantidas suas interfaces com as camadas superiores.

Evolutibilidade. A Arquitetura permite dois tipos de evolutibilidade: a evolução vertical e a horizontal. Na evolução vertical, alterações ocorrem em todas as camadas. O acréscimo de uma nova funcionalidade no sistema pode representar uma evolução vertical, que determina uma evolução em cada camada específica, ou seja, cada camada

realiza o tratamento relativo a sua abstração para o acréscimo de tal funcionalidade. Na evolução horizontal, esta ocorre dentro de uma mesma camada, sem interferir no funcionamento das outras. A independência entre as camadas permite que a evolução horizontal seja realizada com grande facilidade. Por exemplo, mudanças podem ser realizadas para melhorar aspectos de eficiência da Base de Dados, na CP. Essas alterações são realizadas localmente na CP, sem afetar a CN.

Integridade do sistema. A integridade do sistema pode ser verificada a partir da integridade de cada uma de suas partes. Deve ser verificada a integridade da CI, da IIN, da CN, da INP, da CP e do MA.

Portabilidade. A portabilidade pode ser definida para cada parte da Arquitetura. Uma camada pode ser disponibilizada de diferentes formas para permitir portabilidade. A divisão em partes menores permite uma melhor organização dos elementos básicos formadores da Arquitetura, e com isso, a portabilidade pode ser atingida para o nível desejado.

Segurança. Os aspectos de segurança podem receber um melhor tratamento na Arquitetura em 3 Camadas. Por exemplo, em um sistema monolítico ou em 2 camadas, o esquema da Base de Dados é aberto e precisa ser conhecido para possibilitar interação com o mesmo. Na Arquitetura, a CP encapsula o esquema físico da Base de Dados, o qual somente é acessado através da INP, que pode controlar aspectos relativos a segurança. Regras que controlam segurança podem ser implementadas na CN, garantindo o controle a disponibilização de informações da CP, requeridas por usuários do sistema, a partir da CI.

Tratamento de Falhas. Falhas locais a cada camada podem receber tratamento específico. Falhas que precisem ser refletidas na camada superior são sinalizadas, assim seguindo até que possam atingir o usuário de uma interface.

Reuso. O encapsulamento de camadas em suas interfaces possibilita o reuso no nível de camadas. Por exemplo, uma CP pode ser reutilizada em diferentes sistemas.

Facilidade de desenvolvimento. O desenvolvimento torna-se específico em cada camada. Com a divisão em camadas, as tarefas podem ser agrupadas de acordo com a especialidade de cada equipe de desenvolvimento, atingindo, assim, uma afinidade da equipe com o desenvolvimento específico de cada camada.

Testes. Os testes do sistema podem ser realizados localmente dentro de cada camada e, globalmente, verificando toda a execução, desde a CI até a CP. A divisão do sistema em partes menores possibilita a realização de testes mais específicos. Além do mais, quando um sistema evolui, a realização de novos testes pode ser localizada na camada que sofreu evolução.

Manutenibilidade. A Arquitetura define claramente cada uma de suas partes e suas respectivas responsabilidades; com isso, a realização de manutenção torna-se uma tarefa menos complicada que em sistemas monolíticos.

5.4.2 Aspectos de Gerência

Especialização do trabalho. A divisão em camadas favorece a especialização do trabalho de acordo com o conhecimento necessário para realizar cada camada da Arquitetura. Por exemplo, no desenvolvimento da CI, a programação de interfaces GUI pode ser muito complexa, exigindo um grande conhecimento de *frameworks* de interface GUI e suas formas eficientes de instanciação.

Atualmente, a programação da interface GUI pode ser facilitada por ferramentas de construção de interface visual, como por exemplo, *Delphi*, *Visual Basic*, *Visual Age* e *Power Builder*, entre outras. Por outro lado, a implementação de uma CP e de sua respectiva INP pode ser muito facilitada pela utilização de SGBDs, como *Oracle* ou *SQL Server*. Os conhecimentos necessários para o desenvolvimento do sistema variam de acordo com a parte do desenvolvimento a ser tratada. Essa divisão em camadas permite uma separação na equipe de desenvolvimento de acordo com o conhecimento necessário, permitindo uma maior especialização das equipes e uma melhor divisão de trabalho.

Planejamento e Análise de Custos. A Arquitetura apresenta uma organização do sistema a ser desenvolvido. Essa organização pode ser mapeada para um planejamento de execução do Projeto. Por estar dividida em partes menores e mais especializadas, pode-se conseguir uma melhor averiguação do sistema a ser desenvolvido, e com isso, um planejamento mais realista da execução do projeto. Essa organização também ajuda na realização de uma análise dos custos envolvidos no desenvolvimento. Para cada camada, deve-se analisar o tempo, as ferramentas de trabalho e o perfil de desenvolvedores necessários para realizá-la. Essas três características representam aspectos fundamentais para a Análise dos Custos envolvidos no desenvolvimento do sistema.

Independência entre as equipes. A independência de desenvolvimento das camadas propicia a formação de equipes de desenvolvimento independentes. Com isso, gerentes de projeto podem trabalhar com um escopo mais abrangente, podendo até terceirizar o desenvolvimento de partes de um sistema grande e complexo.

Monitoramento do progresso do sistema. Um gerente consegue acompanhar o desenvolvimento do sistema mesmo sem compreender totalmente a parte técnica envolvida. Cada camada pode apresentar seu próprio cronograma de desenvolvimento, e com isso, o gerente pode perceber algum desbalanceamento da produção esperada.

6 A Metodologia

A Metodologia objetiva guiar o desenvolvimento da Camada de Negócio, de acordo com a Arquitetura apresentada no capítulo anterior. Esta Metodologia objetiva, a partir de um Modelo de Análise, atingir um Projeto da Camada de Negócio. Esse Projeto deve ser coerente com a Arquitetura apresentada e deve enfatizar os seguintes aspectos: organização do código gerado; mapeamento entre Análise, Projeto e Implementação; facilidade de realizar testes para verificar a corretude do código gerado; boa documentação e facilidade de acompanhar o cronograma de desenvolvimento.

Esta Metodologia apresenta um processo de construção de Projeto. De acordo com [Scott97], um processo de Projeto bem definido pode oferecer uma ajuda sobre como realizar decisões sobre Projetos. Um bom processo não especificará uma solução particular em detrimento de outra, mas guiará o projetista a uma forma de estruturar suas decisões que o levará a um resultado positivo. Um processo bem definido nunca vai substituir o talento de um bom projetista. Um bom processo, juntamente com um bom projetista, deve levar a execução de bons projetos sob uma base consistente.

A Metodologia realiza-se a partir de um Modelo de Análise que deve representar as abstrações relativas ao domínio do problema. Não é do escopo deste trabalho definir como obter um bom Modelo de Análise. Entretanto, o Modelo de Análise é fundamental para se chegar a um bom Projeto. Em [Scott97], a tarefa de levantar as abstrações do domínio do problema e de obter um Modelo de Análise é considerada uma das tarefas mais importantes em um desenvolvimento Orientado a Objetos, e a seguinte afirmação é realizada : *“Comece com o conjunto errado de abstrações e você nunca vai obter um Projeto correto”*, ou seja, um bom Projeto somente pode ser atingido a partir de um bom Modelo de Análise. Esta é a premissa básica adotada pela Metodologia aqui apresentada.

A Figura 6-1 apresenta o resultado que a Metodologia deve atingir. Nela, estão representadas as fases de Análise, de Projeto e de Implementação de um sistema. Ao final do capítulo, deve estar claro como esta evolução ocorre, ou seja, como a Metodologia se propõe a guiar o desenvolvimento da Camada de Negócio. A seguir, definimos as etapas

necessárias para atingir o resultado desejado. Nesse momento, esta figura é apenas ilustrativa e serve para adiantar o resultado que desejamos atingir com a aplicação da Metodologia.

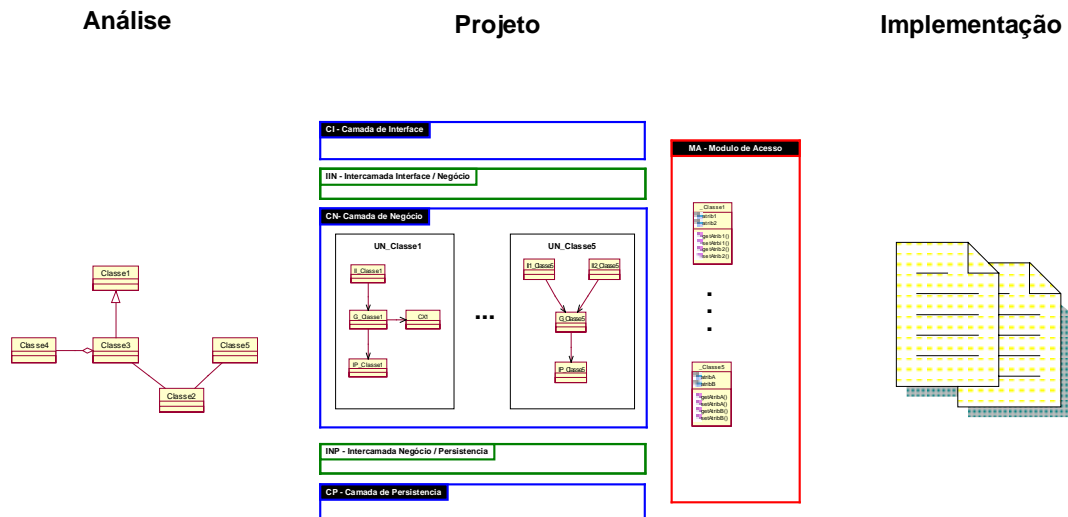


Figura 6-1 resultado obtido na utilização da Metodologia

A Metodologia para o desenvolvimento da Camada de Negócio divide-se em 3 etapas. São elas :

- Definição do Módulo de Acesso;
- Organização da Camada de Negócio;
- Detalhamento das Unidades de Negócio;

Ao final da aplicação da Metodologia, o resultado obtido será o Projeto do Módulo de Acesso e da Camada De Negócio, compatíveis com o Modelo De Análise apresentado.

Este capítulo encontra-se dividido nas seguintes seções: pré-requisito, etapas, resultados e avaliação.

6.1 Pré-Requisito

Conforme mencionado anteriormente, a Metodologia parte de um Modelo de Análise baseado no paradigma de Orientação a Objetos que consiste em alguns diagramas, dentre os quais o Diagrama de Classes.

“Diagramas de classes mostram um conjunto de classes e seus relacionamentos. Esses são os diagramas mais encontrados na modelagem de sistemas orientados a objetos. Diagramas de classe fazem parte de uma visão estática da modelagem do sistema.” [UML99]

A figura, abaixo, exemplifica um Diagrama de Classes.

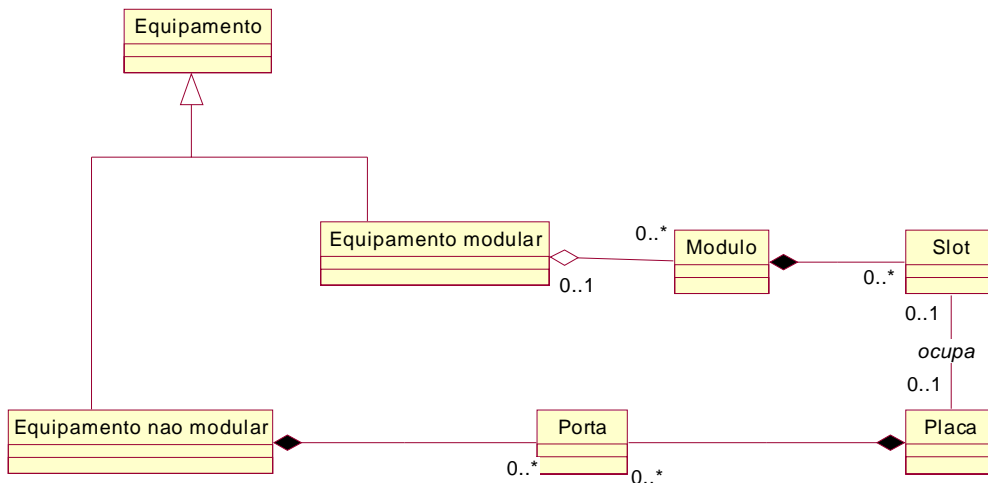


Figura 6-2 exemplo de diagrama de classes

No exemplo de Diagrama de Classes, apresentado na Figura 6-2, acima, podemos verificar os diferentes tipos de relacionamentos encontrados em Diagramas de Classe. São eles: herança, agregação, composição e associação. Na seção 6.2 abaixo, mostraremos as três etapas da Metodologia para o desenvolvimento de uma classe simples, sem considerar os diferentes tipos de relacionamento dos quais essa classe pode participar. No próximo capítulo, Heurísticas, na seção 7.1-Relacionamentos, serão discutidas formas de tratamento dos relacionamentos, de acordo com a Metodologia.

O Modelo de Análise pode ser composto por mais de um Diagrama de Classes. Outros diagramas, como, Diagramas de Objetos, Estados e Sequência [UML99] também podem fazer parte do Modelo de Análise; eles representam aspectos do sistema que devem ser considerados no desenvolvimento do mesmo.

A Metodologia é baseada na estruturação da Camada de Negócio a partir das classes apresentadas no Diagrama de Classes, que será o diagrama necessário para a utilização da Metodologia. Os outros diagramas do Modelo de Análise são úteis para a Análise e o Desenvolvimento de um sistema; entretanto, não representam uma necessidade fundamental para a aplicação da Metodologia, que é uma metodologia estrutural, de onde decorre a sua evolução sobre um modelo estático de classes apresentado, o Diagrama de Classes.

O Diagrama de Classes não será apenas utilizado na parte inicial do desenvolvimento, ele também refletirá o Projeto e a Implementação do sistema, representando uma documentação do Desenvolvimento. Alterações que venham a acontecer na Implementação devem ser refletidas no Modelo de Análise, da mesma forma que alterações no Modelo de Análise devem refletir na Implementação.

6.2 Etapas

Esta seção apresenta as três etapas para construção do Projeto, de acordo com a Metodologia.

6.2.1 Definição do Módulo de Acesso

O Módulo de Acesso (MA), mencionado no capítulo anterior, é composto por Classes de Acesso cuja geração é feita a partir de classes do Modelo de Análise. Cada classe do Modelo de Análise gera uma Classe de Acesso. As Classes de Acesso possuirão os mesmos atributos das respectivas classes do Modelo de Análise, sendo estes privados, apenas acessíveis pelos métodos de acesso, os quais serão os únicos métodos destas

classes. Para cada atributo *Atrib*, a Classe de Acesso apresentará os métodos *getAtrib* e *setAtrib*.

As figuras (Figura 6-3, Figura 6-4), abaixo, ilustram uma classe do Modelo de Análise e sua respectiva Classe de Acesso no Módulo de Acesso.



Figura 6-3 classe do Diagrama de Classes – classe Usuário

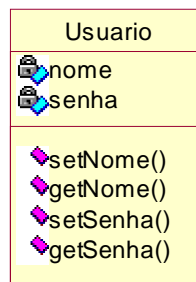


Figura 6-4 classe do Módulo de Acesso –Classe de Acesso Usuário

A Figura 6-5, abaixo, apresenta o Diagrama de Classes e o Módulo de Acesso, composto por Classes de Acesso, de acordo com a Metodologia.

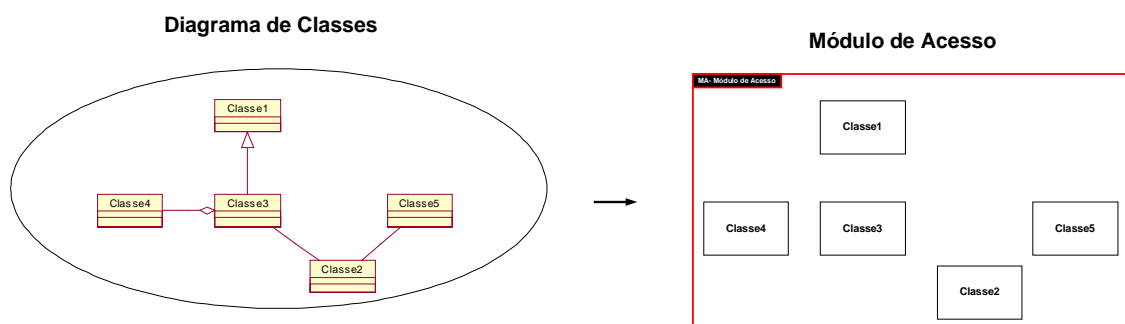


Figura 6-5 Diagrama de Classes e Módulo de Acesso

6.2.2 Organização da Camada de Negócio

Nesta Metodologia, a Camada de Negócio é formada por Unidades de Negócio (UN), que são o agrupamento de classes de Projeto que tratarão de todos os aspectos relativos às funcionalidades oferecidas pela respectiva classe do Modelo de Análise, ou seja, cada UN representa uma classe do Modelo de Análise.

A Figura 6-6 apresenta um Diagrama de Classes e a Camada de Negócio, formada pelas UNs, de acordo com a Metodologia.

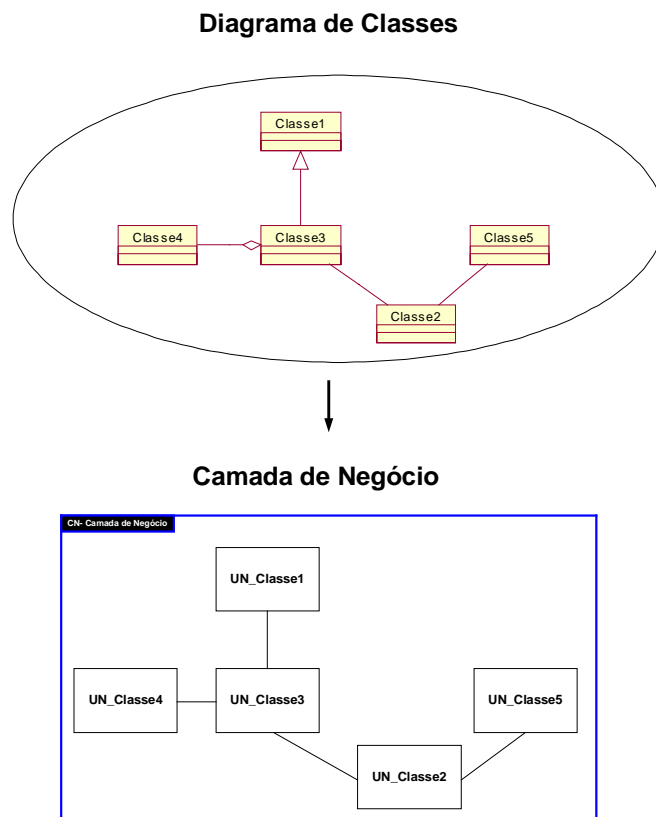


Figura 6-6 UNs da Camada de Negócio, geradas a partir do Modelo de Análise

As UNs possuem relacionamentos entre elas. Esses relacionamentos refletem os relacionamentos existentes entre as classes do Modelo de Análise. A figura apresentada também ilustra este mapeamento entre relacionamentos das UNs e os relacionamentos existentes no Diagrama de Classes apresentado. Na próxima etapa, será esclarecido o

funcionamento interno de cada UN, e com isto, esclarecer-se-á a responsabilidade de gerenciar tal relacionamento.

6.2.3 Detalhamento das Unidades de Negócio (UNs)

Esta etapa trata da implementação de cada UN definida na etapa anterior. Cada UN, como ilustrado na figura abaixo, poderá ser composta pelas seguintes classes :

- Intermediários de Interface (II);
- Gerente (G);
- Intermediários de Persistência (IP);
- Classes Auxiliares (CX);

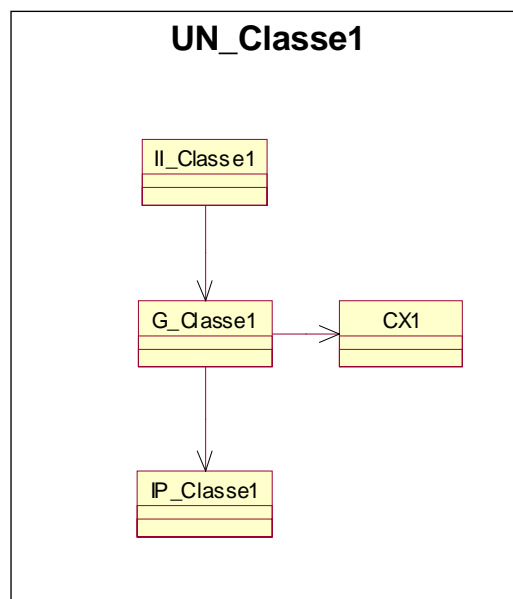


Figura 6-7 Unidade de Negócio

Cada uma destas classes possui responsabilidades bem definidas dentro de uma UN.

Classes das UNs utilizam Classes de Acesso. Entretanto, não há a necessidade de cada UN conhecer todas as Classes de Acesso do MA. As UNs devem utilizar apenas as Classes de Acesso necessárias para realizar as funcionalidades relativas à UN específica. As Classes de Acesso são utilizadas como parâmetros e como retornos de métodos que realizam as funcionalidade oferecidas por cada UN.

Intermediários de Interface (II)

Os Intermediários de Interface (IIs) são classes responsáveis pelo oferecimento das funcionalidades da UN para a CI. Essas classes definirão a interface da Camada de Negócio, ou seja a Intercamada Interface/Negócio deve acessar somente os IIs.

Caso uma UN não apresente interface, ela não possuirá nenhuma Classe II. Geralmente, cada UN apresenta uma II; entretanto, mesmo não sendo necessária a existência de mais de uma Classe II, por motivos práticos, pode-se definir mais de um II por UN, onde cada II representará uma visão da UN.

Por exemplo, seja UN_Usuário uma UN que trata de aspectos relativos às funcionalidades de usuários. Por motivos de segurança, pode ser definido um II para encapsular a funcionalidade relativa a validação de uma senha, enquanto existirá outro II para as outras funcionalidades oferecidas para a UN_Usuário. Este exemplo encontra-se ilustrado na figura abaixo.

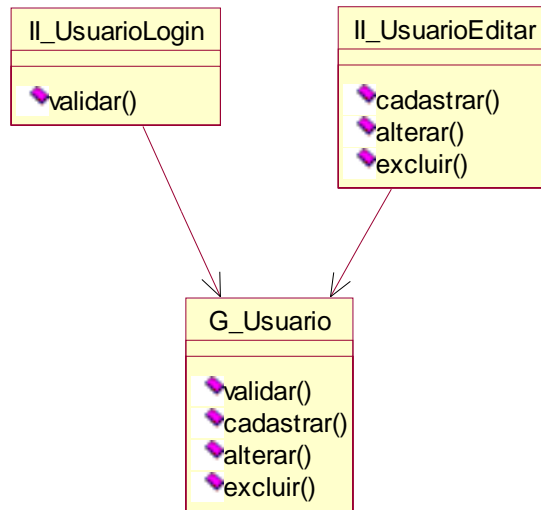


Figura 6-8 Intermediários de Interface para Usuário dentro da UN_Usuário

Os Gerentes das respectivas UNs colaboram com IIs fornecendo a esses as funcionalidades da UN. Assim sendo, cada II está associado a um único gerente, o gerente da UN.

Gerentes (G)

As classes Gerentes são responsáveis pela realização de toda a lógica do sistema relativa a sua UN. Cada UN possui uma classe Gerente, que realiza toda a implementação das regras de negócio relativas à sua UN. Juntando todas as funcionalidades oferecidas pelos Gerentes de cada UN, obtemos toda a funcionalidade oferecida pela Camada de Negócio, e conseqüentemente, pelo sistema.

Cada Gerente pode acessar quantos Intermediários de Persistência (IP) forem necessários para tratar de aspectos relativos à persistência de sua UN. O Gerente também pode acessar as Classes Auxiliares (CX) necessárias para suprir as funcionalidades de sua UN.

O IIs de suas respectivas UN e Gerentes de outras UNs são as únicas classes que acessam os Gerentes. O relacionamento entre Gerentes de diferentes UNs implementa o relacionamento entre UNs, mostrada na etapa anterior. A Figura 6-9, abaixo, é uma

extensão da Figura 6-6, apresentada anteriormente, esclarecendo, agora, as classes responsáveis pela realização dos relacionamentos entre as UNs.

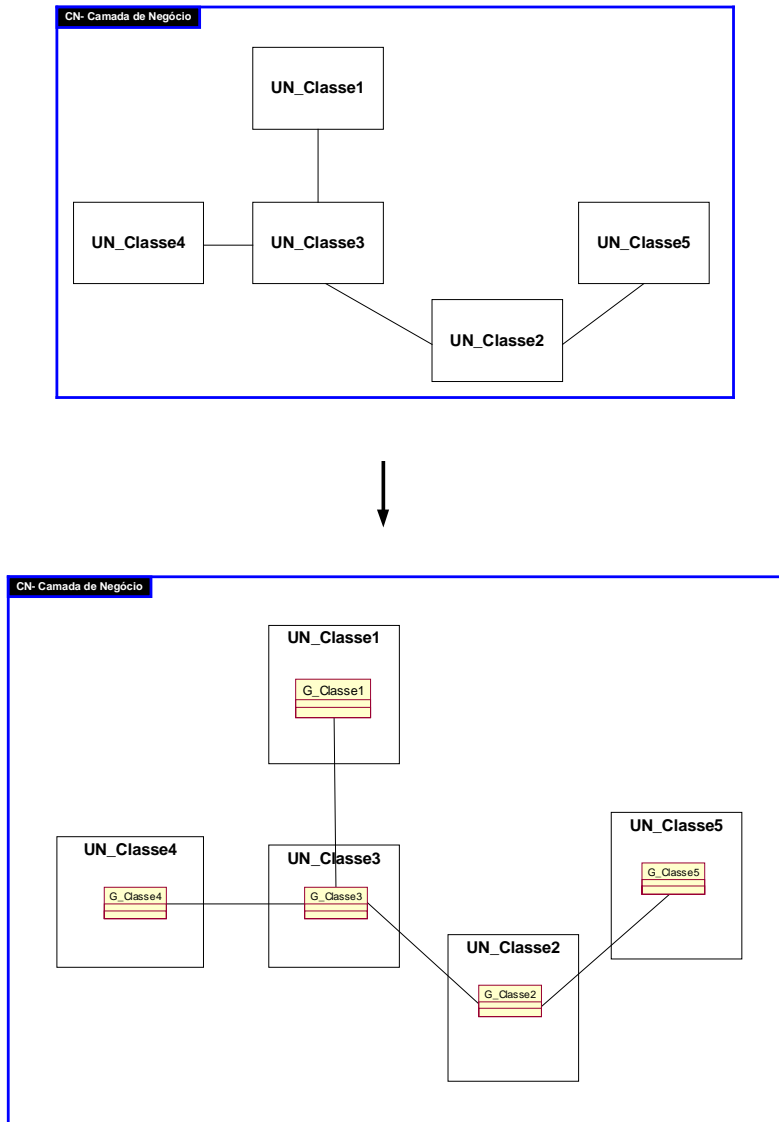


Figura 6-9 Associação entre Gerentes das UNs

Intermediários de Persistência (IP)

A Classe IP é responsável pela definição e pela realização da persistência necessária para sua UN. Toda informação que precisar ser persistida na UN será realizada pela classe IP.

Uma UN pode acessar mais de um IP, sendo que cada IP poderia tratar da persistência de uma parte da UN.

Somente os Gerentes das UNs poderão acessar os respectivos IPs. No caso de uma UN não possuir informação para ser persistida, não haverá IPs.

Os IPs conhecem a interface oferecida pela Intercamada Negócio/Persistência, e repassam seus pedidos de persistência para as classes desta intercamada responsáveis pela execução da persistência.

Classes Auxiliares (CX)

Classes auxiliares são classes que tratam de funcionalidades específicas que aparecem em uma UN e que realizam funcionalidades que são delegadas por seus Gerentes específicos.

Um exemplo de uma Classe Auxiliar é a classe CX_Criptografóloga, da UN_Usuário, responsável pela realização da criptografia em senhas de usuários. Em um primeiro momento, o Gerente desta UN, ao cadastrar a senha de um usuário, por questão de segurança, criptografa a senha antes de armazená-la. Em outro momento, quando o gerente for validar um usuário, ele novamente criptografa a senha recebida e a compara com a armazenada. Nota-se que essa funcionalidade poderia estar inserida na classe Gerente; entretanto, a criação de uma Classe Auxiliar para tratar deste aspecto apresenta uma solução mais flexível, facilitando, assim, a troca de algoritmos de criptografia.

Dependendo da complexidade da funcionalidade tratada por uma Classe Auxiliar, ela pode utilizar outras Classes Auxiliares, para as quais pedidos serão delegados.

A Figura 6-10, abaixo, ilustra a UN_Usuário com suas respectivas IIs, Gerente, IPs e Classe Auxiliar.

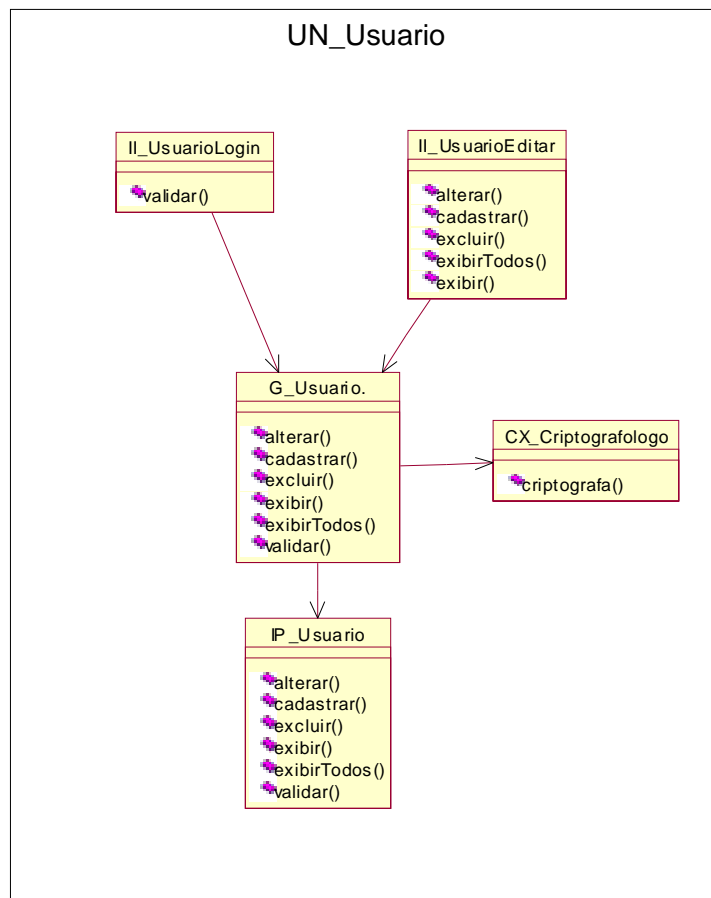


Figura 6-10 Unidade de Negócio Usuário (UN_Usuário)

6.3 Resultados

Esta seção objetiva discutir os resultados obtidos a partir da utilização da Metodologia e apresentar uma breve discussão sobre estes resultados. A próxima seção, 6.4 - Avaliação, fará uma análise mais detalhada da Metodologia.

A Metodologia apresenta duas formas de tratamento às classes do Modelo de Análise, conforme ilustrado na Figura 6-11, abaixo. No Módulo de Acesso (MA) realiza-se um tratamento relativo a dados tratados pelo Projeto, enquanto que nas camadas fornece-se um tratamento relativo às funcionalidades oferecidas a determinados níveis de abstração. Conforme explicado no capítulo anterior, cada camada apresenta um tratamento para suas funcionalidades específicas.

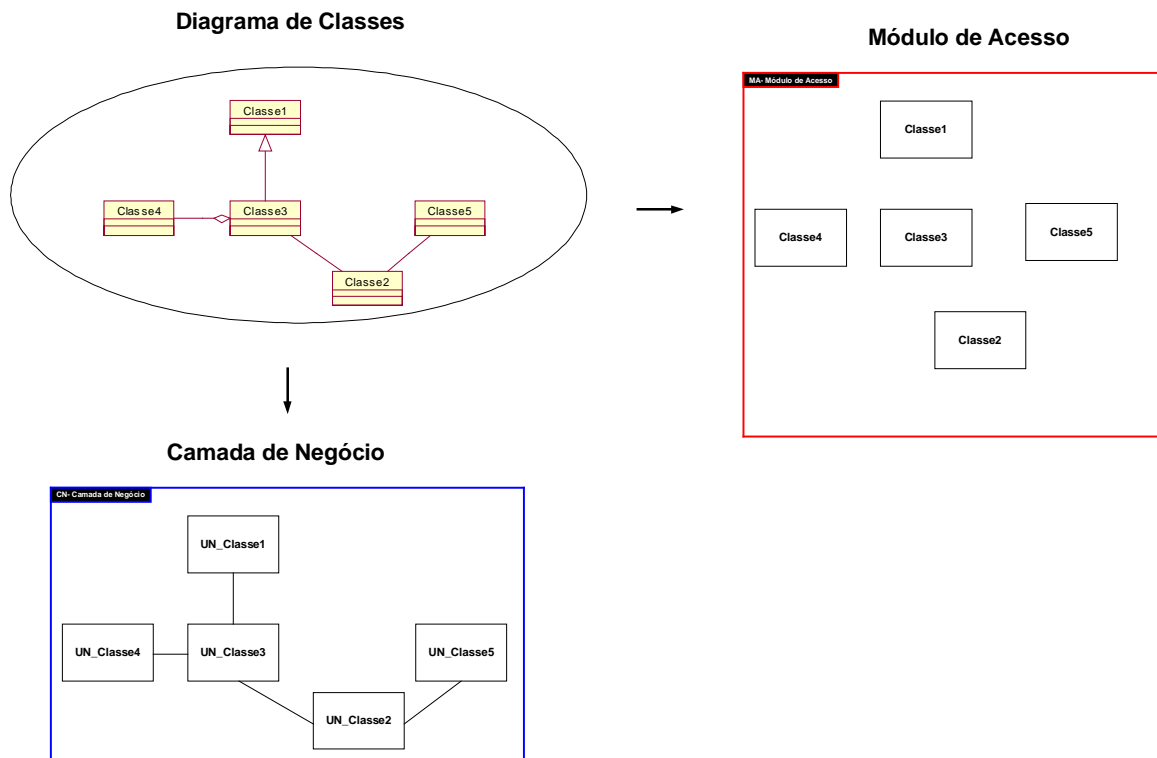


Figura 6-11 Tratamento de Dados x Tratamento de Funcionalidades

[Wiris-Brock90] sugere que operações de classes possam ser desacopladas de atributos de classes, acessando estes somente através de métodos que acessem estes atributos para seleção ou modificação. Dessa forma é que a Metodologia propõe que sejam criadas as Classes de Acesso no MA.

A vantagem de separar dados de funcionalidade está relacionada ao fato de que, na Engenharia de Software, cálculos e manipulações sobre dados sofrem alterações mais frequentemente do que os próprios dados. Com a solução adotada pela Metodologia, os dados estão separados das funcionalidades que os manipulam, e com isso, atinge-se uma solução mais flexível e suscetível a mudanças.

6.4 Avaliação

O objetivo desta seção é discutir aspectos relacionados ao Projeto obtidos após a aplicação da Metodologia. De acordo com [Scott97], não medimos o Projeto, mas sim características do Projeto. Nesta seção, analisaremos um conjunto de características de Projeto com uma breve descrição de seu conceito e de sua aplicabilidade na Metodologia. As características analisadas são:

1. Tamanho
2. Complexidade
3. Acoplamento
4. Qualidade de Abstração
 - Coesão
 - Suficiência
 - Completude
5. Simplicidade
6. Similaridade
7. Volatibilidade

6.4.1 Tamanho

Segundo [Scott97], Tamanho é um atributo interno do produto de Software que contribui para medir produtos, processos e projetos de software. Também é uma característica que contribui para estimar esforço e cronograma envolvendo projetos.

O problema analisado possui um tamanho que caracteriza entidades, relacionamentos e restrições. Este tamanho é o tamanho do domínio do problema, o qual, para a Metodologia proposta, encontra-se representado no Modelo de Análise. A Metodologia apresenta um processo para gerar um Projeto de solução de software para o problema

analisado. O tamanho da solução gerada reflete o tamanho do problema analisado, medido no Modelo de Análise, multiplicado por um fator que representa a Metodologia proposta para gerar o Projeto da solução.

Considerando um desenvolvimento Orientado a Objetos, tipicamente, teríamos, a partir do Modelo de Análise, um mapeamento para um Projeto, e em seguida, para uma Implementação. O mapeamento do Modelo de Análise para um Projeto varia de acordo com o projetista e com as características que ele deseja priorizar. Com este tipo de desenvolvimento, o Tamanho do Projeto obtido e o Fator de quanto o tamanho do Modelo de Análise influi neste tamanho (Tamanho do Projeto) não são conhecidos antes do final da fase do Projeto, o que dificulta a tarefa de estimar esforço e cronograma associado a um sistema. Esta situação é ilustrada na Figura 6-12

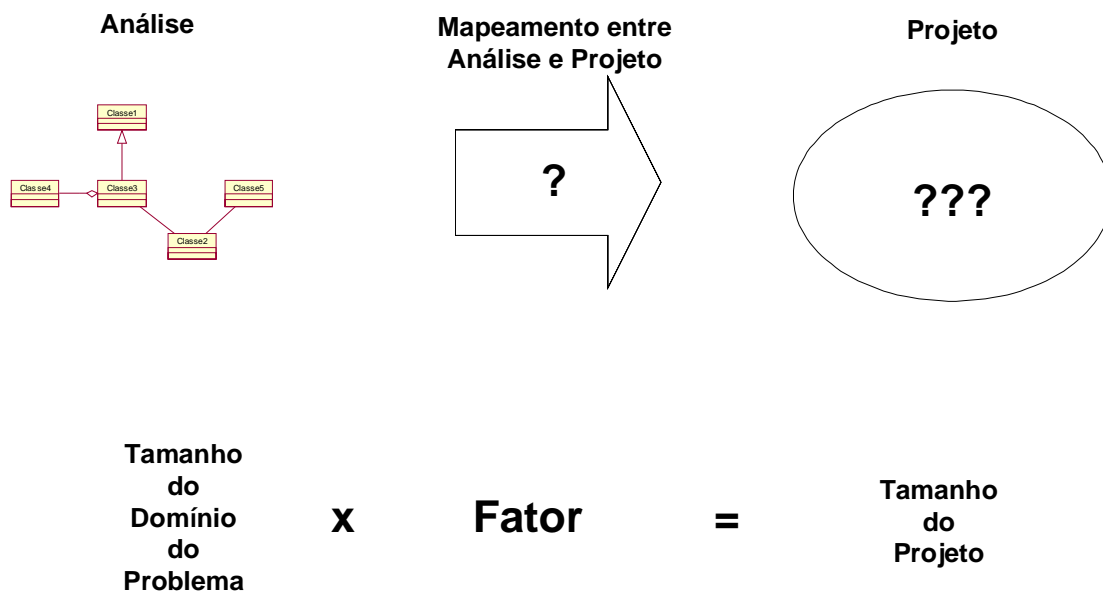


Figura 6-12 Medida do tamanho de um projeto

Ao tratarmos do tamanho do Modelo de Análise estamos considerando o número de classes envolvidas, seus atributos, métodos e relacionamentos. A Metodologia proposta separa cada classe do Modelo de Análise em uma Classe de Acesso no MA, e uma UN na CN.

De acordo com a Metodologia, atributos estão encapsulados nas Classes de Acesso, métodos estão disponibilizados nas UNs, e relacionamentos entre classes, quando refletem funcionalidades e envio de mensagens, são tratados pelos Gerentes das respectivas UNs, e quando refletem busca de dados, são tratados como atributos das Classes de Acesso, e conseqüentemente, estão disponíveis através de seus métodos de acesso.

Com o mapeamento do Diagrama de Classes para a estrutura apresentada na Metodologia, conseguimos medir o tamanho do Projeto realizado para solucionar o problema. A Metodologia apresenta uma solução para realizar o Projeto da CN e do MA. Se também conseguirmos medir o tamanho relativo à CI e à CP, obteremos uma medição bem razoável de todo o projeto do sistema, e desta forma, conseguiremos obter um fator que possa ser multiplicado pelo tamanho do domínio do problema, gerando, assim, uma aproximação do tamanho do Projeto. Este cálculo, mostrado na Figura 6-13, abaixo, auxilia a tarefa de estimar o esforço e o cronograma relacionados ao Projeto do Sistema.

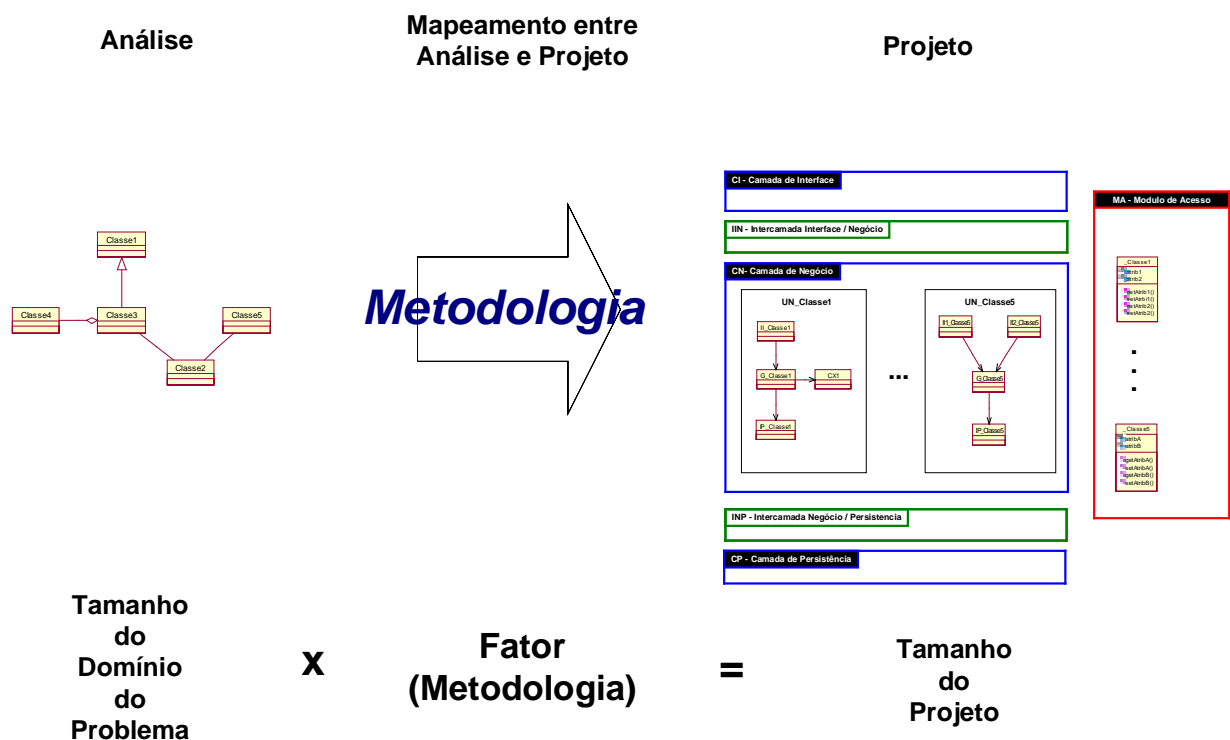


Figura 6-13 Medida do tamanho do Projeto, resultado da aplicação da Metodologia

6.4.2 Complexidade

Segundo [Myers76], “*A complexidade de um objeto é alguma medida do esforço mental necessário para compreender tal objeto*”.

Complexidade é uma importante característica do projeto de software, pois representa a habilidade de entender, de utilizar ou de modificar um pedaço de software. Complexidade está relacionada às tarefas de testar, de modificar e de manter sistemas de software. De acordo com [Henderson-Sellers96], complexidade é um indicador de usabilidade, entendabilidade, manutenibilidade, adaptabilidade e segurança do sistema.

Em Engenharia de Software, o maior gerador de complexidade é o projetista. [Booch94]. A descrição do problema apresenta a complexidade intrínseca do sistema. O Projeto é resultado desta complexidade do problema, acrescido da complexidade relacionada à solução adotada pelo projetista.

Neste trabalho, estamos considerando a complexidade como sendo o grau de dificuldade gasto para entender ou verificar um componente de um sistema.

[Henderson-Sellers96] divide a complexidade relacionada a Engenharia de Software em 3 grupos: características do programador, características do problema e complexidade estrutural. De acordo com sua divisão, características do programador incluem experiência e conhecimento nos domínios do problema e da solução; características do problema refletem as dificuldades inerentes ao problema em questão; enquanto que complexidade estrutural está relacionado à estrutura da solução adotada.

A Metodologia apresenta uma estruturação da CN; por conseguinte a complexidade que pretendemos medir nesta seção é a complexidade estrutural (segundo [Henderson-Sellers96]), que é relativamente mais simples se comparada a complexidade estrutural de um sistema monolítico. A Metodologia “quebra” em partes menores e menos complexas o modelo e o tratamento do problema apresentado. Todavia, as características do

programador e as características do problema não são tratadas pela solução discutida neste trabalho.

A Metodologia realiza uma distribuição da complexidade do sistema em partes menores. A CN é estruturada em UNs que apresentam um tratamento específico para cada entidade tratada pelo negócio. Esta organização atinge um encapsulamento das várias entidades tratadas pelo sistema. Alterações no sistema podem ser tratadas localmente nas UNs envolvidas, minimizando alterações em outras partes do sistema.

No escopo da Camada de Negócio, dentro de cada UN, Gerentes são responsáveis pelo tratamento de todas as funcionalidades oferecidas pela UN. Eles podem utilizar Classes Auxiliares para distribuir a complexidade interna a UN. Em sistemas mais complexos, Gerentes podem ser construídos como Mediadores [Buschman96], delegando pedidos a suas Classes Auxiliares. Toda complexidade interna a uma UN não é visível para outras UNs a ela relacionadas, pois a comunicação entre as UNs realiza-se através dos Gerentes, os quais não enxergam outras classes internas.

No escopo da comunicação entre camadas, a complexidade da CN não é visível para a CI. Todo acesso da CI à CN é realizado pelos IIs, que encapsulam toda a funcionalidade a ser oferecida por sua UN para a CI.

6.4.3 Acoplamento

Segundo [IEEE90], acoplamento é a medida da interdependência entre módulos de um sistema computacional. Em Orientação a Objetos, [Wirfs-Brock90] definiu acoplamento entre classes como sendo a medida de quanto estas dependem entre si.

No contexto deste trabalho, consideramos acoplamento como uma característica definida em termos das conexões entre elementos participantes do Projeto.

Acoplamento está fortemente relacionado à tarefa de controlar e de realizar alterações no sistema. Quanto mais dependentes forem as classes do Projeto, maior será a parte do

sistema que deverá ser examinada para se realizar alguma alteração. Idealmente, alterações em uma classe devem ocorrer localmente, minimizando, assim, a necessidade de examinar todo sistema.

Na Metodologia, cada classe do Modelo de Análise é mapeada para uma UN da Camada de Negócio, responsável pelo tratamento das funcionalidades relativas a esta classe, e para uma Classe de Acesso, do Módulo de Acesso, responsável pelo tratamento dos dados relativos a esta classe. Os relacionamentos entre UNs apresentam o acoplamento inerente existente entre as classes do Modelo de Análise. Estes relacionamentos são tratados, internamente, pelos Gerentes de cada UN.

De acordo com a organização de Projeto apresentada pela Metodologia, alterações que ocorrem em classes do Modelo de Análise podem ser separadas em 3 tipos:

- Alterações relativas a dados da classe, as quais afetam as respectivas Classes de Acesso no MA;
- Alterações relativas a funcionalidades da classe, as quais afetam as respectivas UNs;
- Alterações relativas aos relacionamentos entre classes; estas afetam a comunicação entre Gerentes das respectivas UNs, e podem também afetar as respectivas Classes de Acesso no MA, dependendo do tratamento dado aos relacionamentos.

As alterações que ocorrem em um sistema podem variar de tamanho e de complexidade. De acordo com a Metodologia, estas alterações serão visualizadas no Modelo de Análise, podendo ser mapeadas para a estrutura do Projeto e, conseqüentemente, para a Implementação. Os diferentes tipos de alterações, apresentadas acima, podem ocorrer em diversas combinações, e a solução gerada é beneficiada pelo desacoplamento apresentado pela Metodologia, que permite o tratamento específico para cada tipo de alteração.

Como os componentes apresentados pelo Projeto estão fracamente acoplados, e suas dependências estão claramente mapeadas, estes podem ser reutilizados separadamente.

Outros sistemas podem beneficiar-se de Classes de Acesso e de UNs já implementadas. Em Projetos que apresentam solução fortemente acopladas, a reutilização é uma tarefa mais árdua e que pode representar que, para reutilizar uma parte do código, outras partes, desnecessárias ao sistema, tenham de ser trazidas juntamente com a parte desejada.

6.4.4 Qualidade de Abstração

[Scott97] apresenta duas definições para qualidade de abstração:

- para Modelo de Análise, qualidade de abstração é definida como uma medida da característica do conjunto de abstrações no Modelo de Análise, que reflete as propriedades das entidades do domínio.
- para Projeto, qualidade de abstração é definida como a medida de quanto os componentes de Projeto implementam as propriedades das respectivas abstrações do Modelo de Análise.

[Scott97] também afirma que qualidade de abstração representa 3 componentes: suficiência, completude e coesão.

Neste trabalho, iremos nos concentrar na análise da qualidade de abstração relativa ao Projeto. Para isto, serão analisadas as características de coesão, suficiência e completude na Metodologia apresentada.

Coesão

[Booch94] apresenta o conceito de Coesão através do seguinte exemplo: *“Uma classe Cachorro é funcionalmente coeso se sua semântica representa o comportamento relativo a cachorro, todo seu comportamento e nada mais que o comportamento de cachorro. Uma classe Cachorro que contém outros comportamentos não relativos a cachorro não é coesa.”*

As Classes de Acesso do MA devem ser coesas, representando o comportamento básico da classe equivalente do Modelo de Análise, sem acrescentar funcionalidades relativas à interface, às regras do negócio ou à persistência. Como discutido, anteriormente, na seção 6.2.1, Definição do Módulo de Acesso, classes mais especializadas podem ser definidas localmente nas camadas, mas a Classe de Acesso deve ser funcionalmente coesa.

As UNs da Camada de Negócio também devem ser coesas, assim sendo, cada UN trata das funcionalidades relativas a sua entidade em questão, toda a sua funcionalidade, mas não apresentaria funcionalidades relativas a outras entidades (que devem estar nas suas respectivas UN).

Suficiência e Completude

“Suficiência é o grau no qual um conjunto de abstrações apresenta as propriedades das entidades do domínio de tal forma que este conjunto represente tais entidades para a visão necessária ao sistema em questão.” [Scott97].

“Completude é o grau no qual um conjunto de abstrações apresenta as propriedades das entidades do domínio de tal forma que este conjunto represente tais entidades para todos os sistemas que o utilizam.” [Scott97].

Das definições apresentadas acima, consideramos que um Projeto de uma entidade do Modelo de Análise é suficiente se este Projeto suprir as necessidades relacionadas à entidade para o sistema em questão. O Projeto será completo se suprir a necessidade relacionada ao atual sistema e a todos os sistemas que utilizam tal entidade.

Se para uma dada entidade do Modelo de Análise, o Projeto gerado for suficiente e completo, este poderá ser reutilizado mais facilmente.

A Metodologia, por si só, não garante que o Projeto gerado possua as características de suficiência e completude. Todavia, sua organização em UNs e em Classes de Acesso

representa uma ferramenta útil para a verificação destas características. Idealmente, o Projeto construído deve ser suficiente, completo e coeso; a Metodologia, apesar de não garantir estas características, auxilia projetistas a realizarem suas tarefas e a verificarem os resultados obtidos.

6.4.5 Simplicidade

Einstein divulgava que a melhor forma de resolver um problema é apresentar a solução mais simples possível, e não mais simples que esta.

A simplicidade da solução gerada para o problema deve ser atingida primeiramente pelo Modelo de Análise, que deve ser o mais simples possível para resolver tal problema. A simplicidade que estaremos mencionando neste trabalho está relacionada ao Projeto gerado a partir da Metodologia.

O objetivo da Metodologia é apresentar uma solução simples para realizar o Projeto. Para atingir simplicidade, a solução adotada foi a de dividir o problema (implementação de um sistema que possua interface, negócio e persistência) de forma que suas “partes” possam ser atacadas separadamente. A solução apresentada para tratar do negócio do sistema apresenta uma primeira divisão entre dados e funcionalidades, localizados, respectivamente, nas Classes de Acesso do MA e nas UNs na CN. Dentro de cada UN, ocorre uma divisão entre suas classes, separando-as em Gerentes, Intermediários de Interface, Intermediários de Persistência e Classes Auxiliares, sendo que cada uma delas possui responsabilidades bem definidas.

Segundo [Brown98], arquiteturas que apresentam um particionamento pobre geram aplicações inoperáveis e difíceis de serem reutilizadas.

A estruturação apresentada para o Projeto relativo a uma entidade no Modelo de Análise, se comparada com um “Projeto Convencional”, monolítico, no qual uma classe trataria de todos os aspectos de interface, de regras de negócio e de persistência apresenta uma

solução mais dividida e, conseqüentemente, com cada parte mais simples do que no “Projeto Convencional”.

6.4.6 Similaridade

[Scott97] define dois objetos como sendo estruturalmente similares quando estes possuem estruturas similares. A estrutura dos objetos pode ser separada em atributos e operações. Logo, a similaridade pode localizar-se nos atributos ou nas operações. A seguir, apresentaremos uma análise da Metodologia para esses tipos de similaridade.

Atributos Similares

Sejam duas classes, no Modelo de Análise, que apresentam atributos similares e operações distintas. De acordo com a Metodologia, tais classes gerarão duas UNs, sendo que cada UN trata de suas funcionalidades específicas. O MA pode apresentar uma única Classe de Acesso, representando os atributos similares das duas classes. Esta situação é ilustrada na Figura 6-14 abaixo.

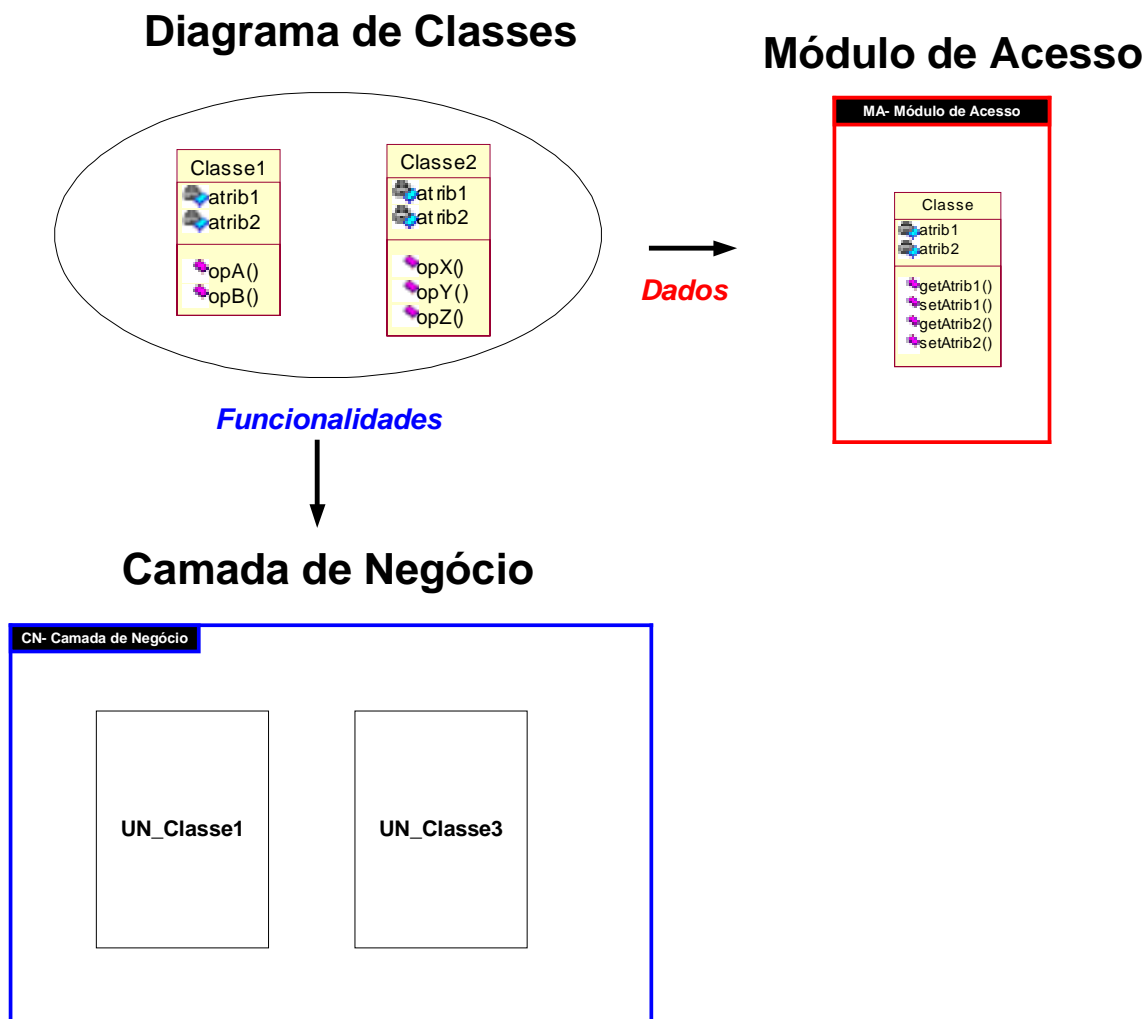


Figura 6-14 Atributos Similares

Operações similares

Sejam duas classes, no Modelo de Análise, que apresentam operações similares e atributos distintos. De acordo com a Metodologia, tais classes podem ser contempladas por uma UN, que trata das funcionalidades, e por duas Classes de Acesso, representando os atributos das duas classes do Modelo de Análise. A Figura 6-15, abaixo, ilustra tal situação:

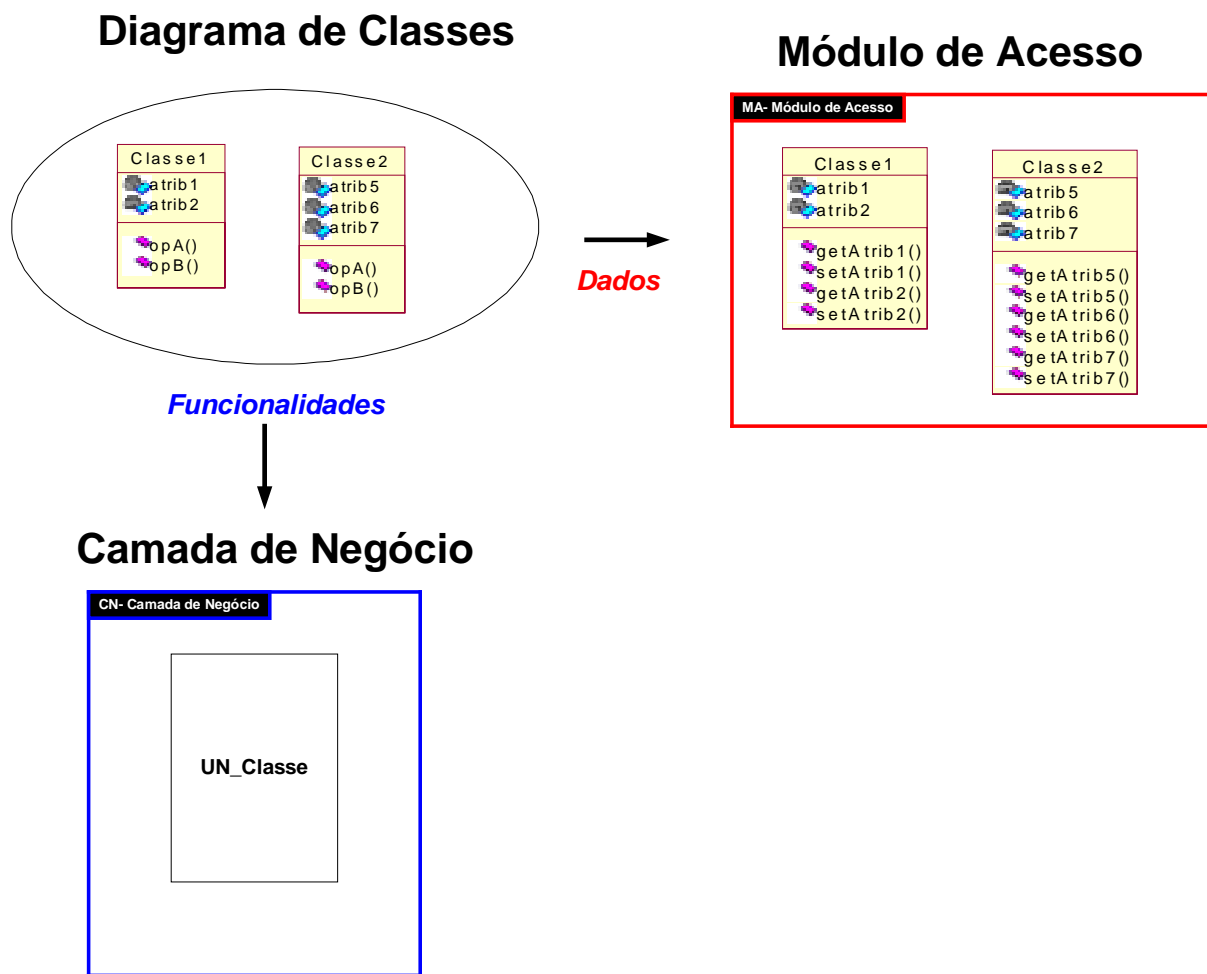


Figura 6-15 Operações Similares

6.4.7 Volatilidade

Volatilidade é a possibilidade ou a probabilidade de que mudanças venham a ocorrer. [Scott97]

Muitos trabalhos na área de Engenharia de Software relacionam o esforço de realizar manutenção ao tamanho e/ou à complexidade do sistema em questão. Ambos, tamanho e complexidade do sistema, são fatores relacionados à quantidade de esforço a ser gasto para realizar manutenção; entretanto, se o sistema nunca sofre alterações, não importa o tamanho ou a complexidade do mesmo, pois o esforço de realizar manutenção será nulo.

Tal raciocínio implica que uma boa medida para o esforço relacionado à manutenção em sistemas relacionará tamanho, complexidade e volatilidade.

Segundo [Scott97], volatilidade, em Sistemas de Software, possui duas origens:

- mudanças causadas em função de alterações no domínio do problema;
- mudanças causadas em função de mudanças realizadas em outras partes do sistema.

No primeiro caso, a Metodologia apresenta um mapeamento entre o Modelo de Análise, o Projeto e a sua implementação; desta forma, as alterações realizadas no domínio do problema devem ser mapeadas para alterações no Modelo de Análise, de onde são, especificamente, mapeadas para suas respectivas estruturas no Projeto.

No segundo caso, a Metodologia separa o tratamento de cada entidade no Modelo de Análise, de forma a tentar atingir independência entre o projeto e a implementação relativos a estas classes. Desta forma, deve-se minimizar as mudanças causadas por mudanças realizadas em outras partes do sistema.

7 Heurísticas

Este capítulo objetiva fornecer heurísticas de utilização da Metodologia apresentada no capítulo anterior. Estas heurísticas representam “dicas” de desenvolvimento de um Projeto de acordo com os conceitos introduzidos pela Arquitetura e pela Metodologia propostas nesta dissertação.

7.1 Relacionamentos

Nesta seção, serão apresentadas possíveis soluções de Projeto, de acordo com a Metodologia, para os relacionamentos de Associação e Herança. [UML99]

7.1.1 Associação

Associação é um relacionamento estrutural, que especifica que objetos de um tipo estão conectados a objetos de outro tipo. [UML99]

A Figura 7-1, abaixo, apresenta um exemplo de associação.



Figura 7-1 Exemplo de Associação : Fornecedor-Produto

O exemplo apresenta a associação *fornece* entre *Fornecedor* e *Produto*. Esta associação *fornece* especifica que objetos do tipo *Fornecedor* estão conectados a (fornecem) objetos do tipo *Produto*.

Uma associação, no Modelo de Análise, está representada, no Projeto, em duas partes: uma parte trata do comportamento, e a outra dos dados. A parte relacionada ao comportamento está sendo tratada pelos Gerentes das UNs, enquanto que a parte relacionada aos dados é tratada pelas Classes de Acesso no MA.

Para o exemplo de associação Fornecedor-Produto, apresentado na Figura 7-1, acima, teríamos, no Projeto gerado pela Metodologia, a Camada de Negócio e o Módulo de Acesso, conforme ilustrado a seguir.

Na CN, os Gerentes G_Fornecedor e G_Produto, respectivamente, das UN_Fornecedor e UN_Produto, estão associados, e tratam de toda a funcionalidade relativa às suas UNs e da comunicação necessária entre elas. A CN está representada na Figura 7-2 abaixo.

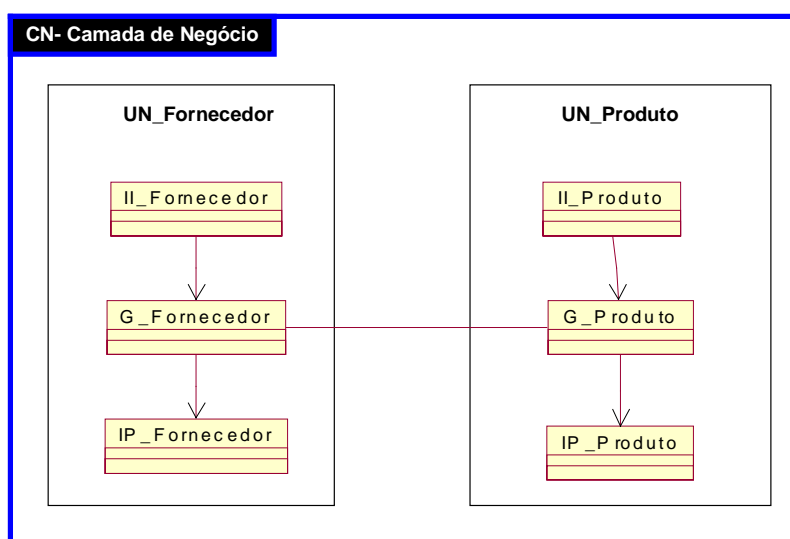


Figura 7-2 Exemplo Fornecedor-Produto (CN-Comportamento)

O MA apresenta as Classes de Acesso *Fornecedor* e *Produto*. Como neste exemplo a multiplicidade da associação é 0,* e 1,*, cada objeto *Fornecedor* possui uma lista de objetos *Produto*, enquanto que este mantém uma lista de objetos *Fornecedor*. Estas classes apresentariam, além de seus métodos de *get* e *set* para cada atributo, os métodos *addFornecedor*, *removeFornecedor*, *getFornecedores*, e *addProduto*, *removProduto*, *getProdutos*, respectivamente, para as Classes de Acesso *Produto* e *Fornecedor*, manipulando as listas de *Fornecedores* e de *Produtos*. A Figura 7-3, abaixo, representa o MA deste exemplo.

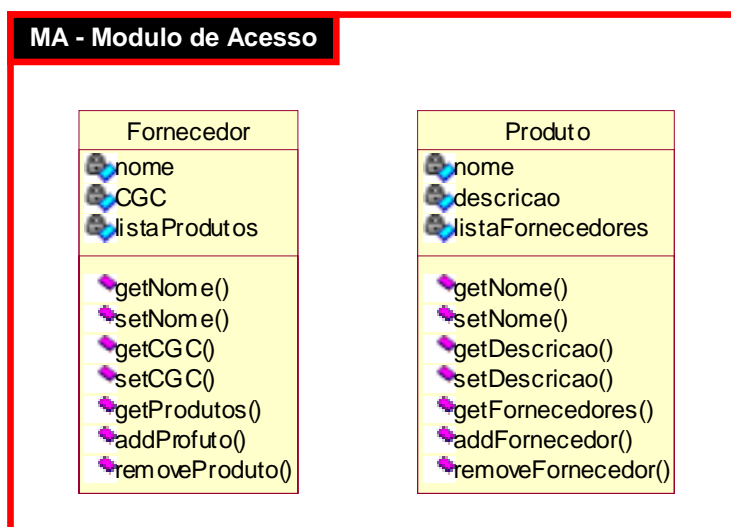


Figura 7-3 Exemplo Fornecedor-Produto (MA-Dados)

As Classes de Acesso, mostradas na Figura 7-3, possuem coleções de *Produtos* e *Fornecedores*, que devem ser privadas, sendo somente acessíveis através dos métodos de *get*, *add* e *remove*. Estas coleções tornam-se necessárias uma vez que a multiplicidade das classes respectivas, no Modelo de Análise, são de 0.* e 1.*. No caso da multiplicidade ser 0,1, não há a necessidade de utilizar uma coleção, mas somente uma referência ao objeto com o qual existe a associação. Esta referência também seria privada, sendo somente acessível através de métodos *get* e *set*.

Abaixo, mostra-se, nas figuras (Figura 7-4, Figura 7-5 e Figura 7-6), outro exemplo de associação na qual um dos lados da associação apresenta a multiplicidade 1. Novamente ilustrar-se-á a CN e o MA para este exemplo. Verifica-se que, por a multiplicidade de *Professor* ser 1 (cada *Turma* possui um e somente um *Professor*), a Classe de Acesso *Turma* apresentará um atributo privado *professor*, acessível por *get* e *set*.

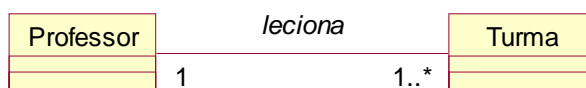


Figura 7-4 Exemplo Professor-Turma

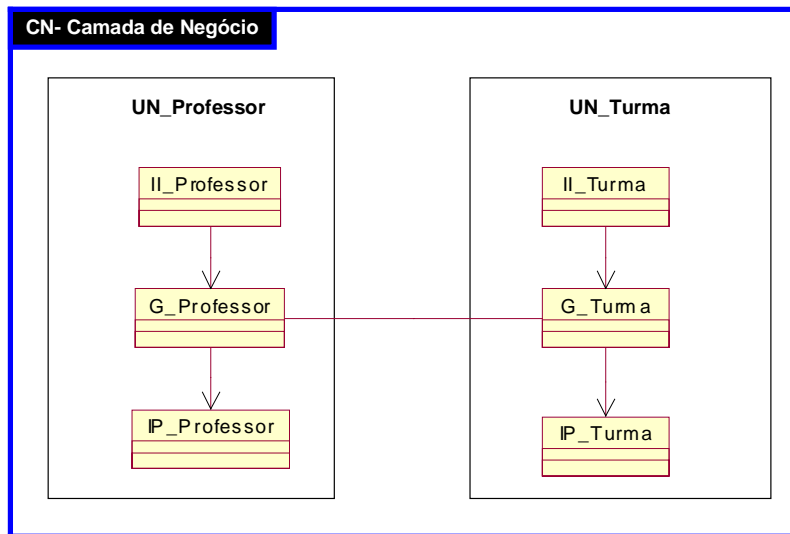


Figura 7-5 Exemplo Professor-Turma (CN-Comportamento)

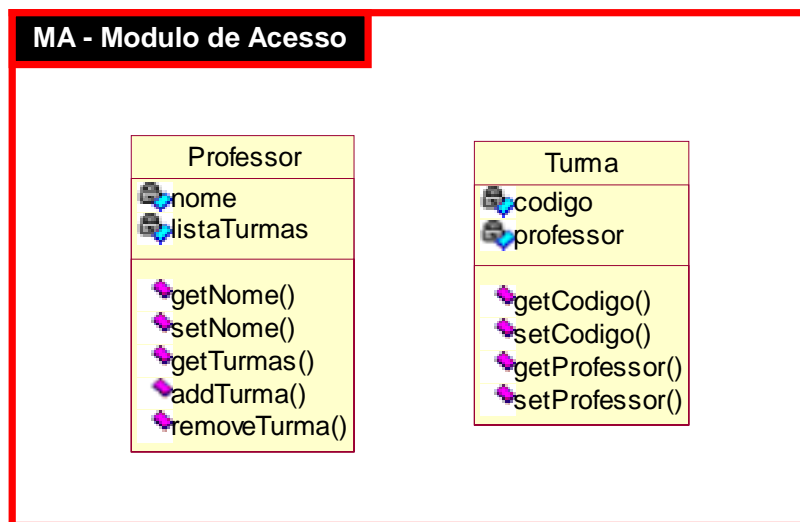


Figura 7-6 Exemplo Professor-Turma (MA-Dados)

Uma das forma mais utilizadas para implementar associações é substituindo a associação entre classes por uma classe responsável pelo tratamento da associação. O exemplo Fornecedor-Produto será rerepresentado, abaixo, modificando seu Modelo de Análise, apresentando, agora, a classe *Fornece* tratando do relacionamento entre *Fornecedor* e *Produto*.

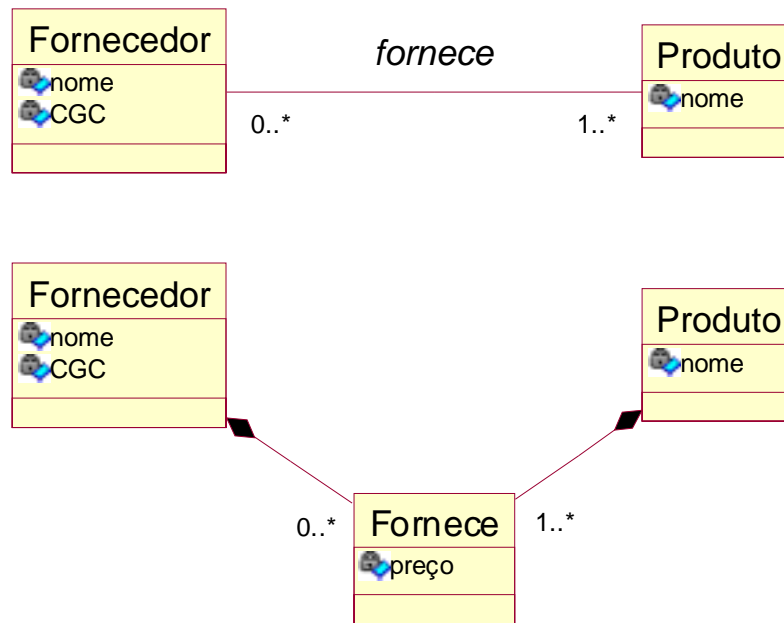


Figura 7-7 Classe substituindo relacionamento

Classes que tratam de associações realizam um melhor controle das associações. Com estas classes, alcança-se uma melhor distribuição da complexidade das classes envolvidas nas associações, gerando, assim, uma melhor divisão da inteligência do sistema. Uma associação pode apresentar atributos próprios, como, por exemplo o preço de um Produto fornecido por um Fornecedor; estes seriam atributos dessas classes.

Este exemplo apresentado seria mapeado no Projeto gerado após a utilização da Metodologia, conforme ilustrado nas figuras (Figura 7-8 e Figura 7-9), abaixo. A classe *Fornece* apresentaria uma UN_Fornece na Camada de Negócio e uma respectiva Classe de Acesso *Fornece* no MA.

As funcionalidades relativas ao fornecimento de Produtos por Fornecedores (associação *Fornece*) são tratadas no Projeto pela UN_Fornece da CN; enquanto que os dados relativos a esta associação são tratados pela Classe de Acesso *Fornece*.

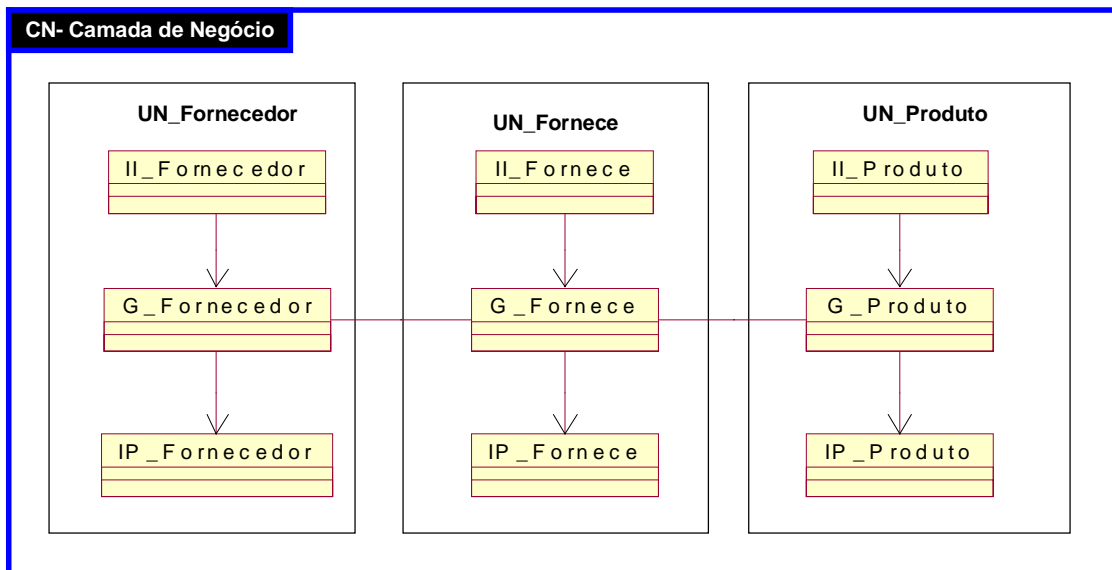


Figura 7-8 Exemplo Fornecedor-Produto (CN-Comportamento)

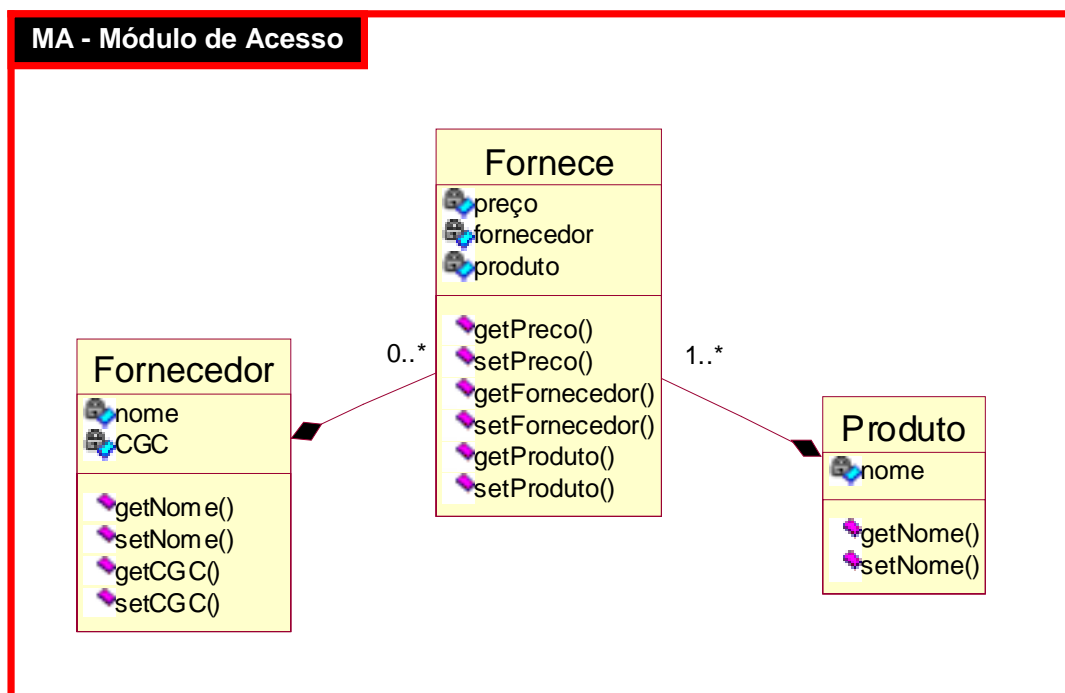


Figura 7-9 Exemplo Fornecedor-Produto (MA-Dados)

7.1.2 Agregação e Composição

De acordo com [UML99]:

“Agregação é uma denominação conceitual usada para distinguir o relacionamento de todo-parte em uma associação. Uma agregação não relaciona os ciclos de vida das classes Todo e Parte.”

“Composição é uma variação de Agregação que adiciona a esta importantes semânticas. Composição é uma forma de Agregação, com um forte conceito de posse e dependência entre os ciclos de vida das classes Todo e Parte. A classe Todo possui objetos da classe Parte e o ciclo de vida destes está limitado pelo ciclo de vida de sua classe Todo.”

À Agregação deve-se oferecer um tratamento similar ao realizado com as associações apresentadas previamente nesta seção. As figuras (Figura 7-10, Figura 7-11 e Figura 7-12), a seguir, exemplificam, para uma Agregação, seu Modelo de Análise e os respectivos CN e MA do Projeto realizado pela Metodologia.



Figura 7-10 Exemplo de Agregação : Computador-Placa

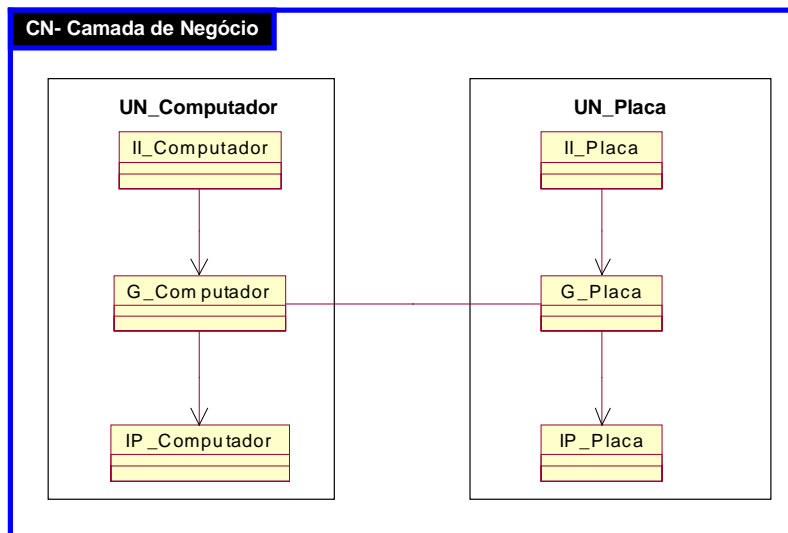


Figura 7-11 Exemplo Computador-Placa (CN-Comportamento)

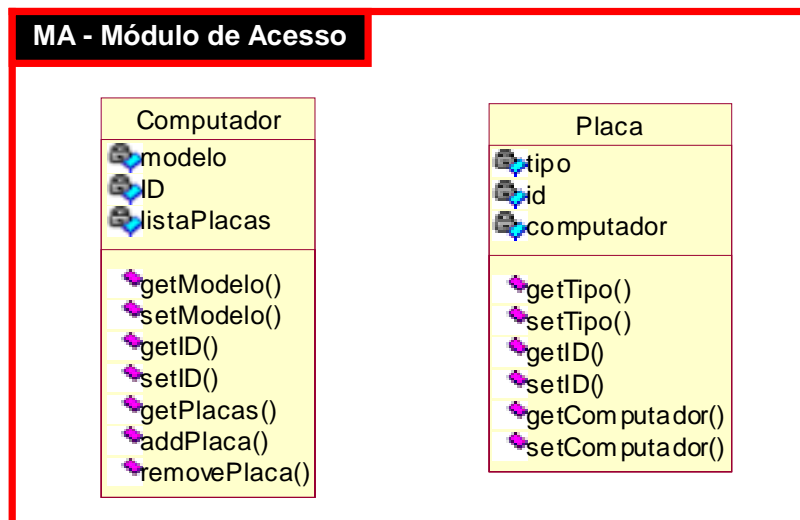


Figura 7-12 Exemplo Computador-Placa (MA-Dados)

A Composição deve receber um tratamento mais específico, o que garante a dependência nos ciclos de vida dos objetos das classes *Parte* e *Todo*. As figuras (Figura 7-13, Figura 7-14 e Figura 7-15), abaixo, apresentam um exemplo de Composição.



Figura 7-13 Exemplo de Composição : Placa-Porta

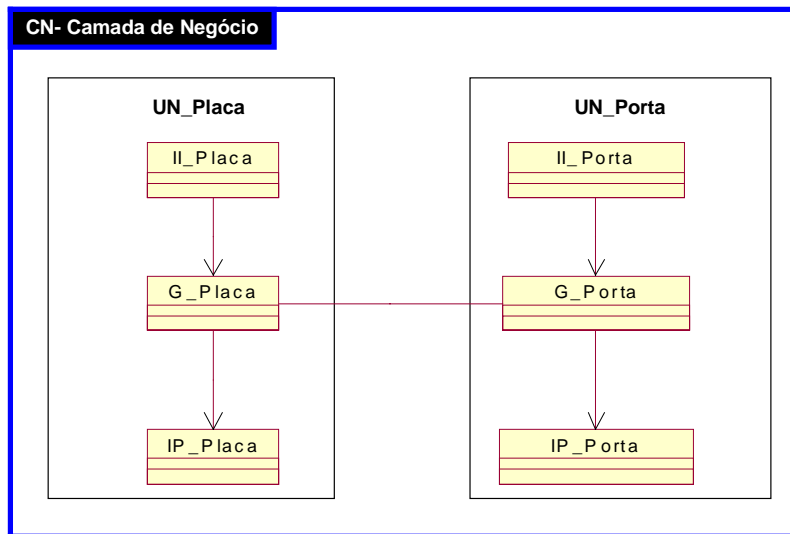


Figura 7-14 Exemplo Placa-Porta (CN-Comportamento)

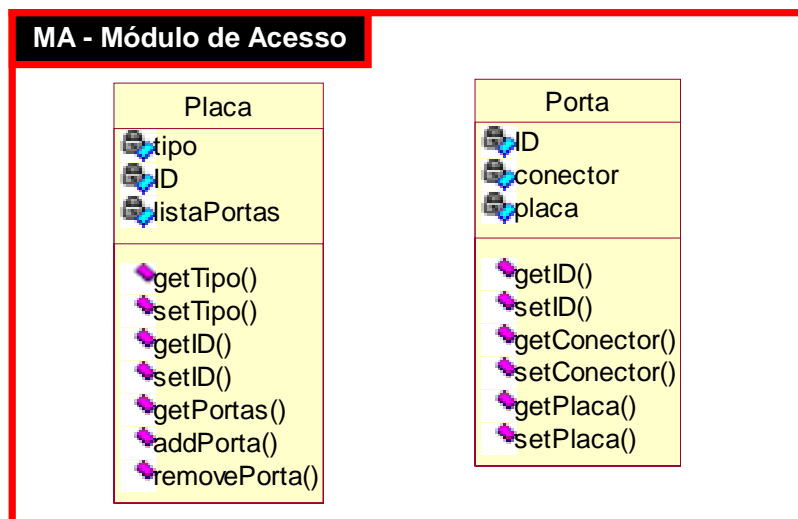


Figura 7-15 Exemplo Placa-Porta (MA-Dados)

A diferença básica entre as associações Todo-Parte entre *Computador* e *Placa* (Agregação) e *Placa* e *Porta* (Composição) é que ao se inutilizar um *Computador*, suas *placas* podem ser aproveitadas em outro *computador* (ciclo de vida independente); enquanto que ao se inutilizar uma *placa*, suas *portas* também inutilizar-se-ão (ciclo de vida dependente).

Alguns mecanismos de implementação podem ser utilizados para garantir a semântica da Composição. Admita-se o exemplo Placa-Porta :

Um objeto da classe *Porta* deve ser instanciado somente para um objeto da classe *Placa*, não podendo mudar de *placa*. Na Classe de Acesso *Porta*, não há a necessidade do método *setPlaca*. A referência para o objeto *placa* deve ser criada assim que o objeto *porta* for instanciado, logo o construtor de *Porta* deve receber *placa* como parâmetro.

Ao eliminar um objeto da classe *Placa*, este deve ser responsável pela eliminação de suas *portas*. O destrutor de *Placa* deve percorrer a lista de *portas*, eliminando-as.

7.1.3 Herança

Segundo [UML99], ao modelar um sistema, muitas vezes, encontramos classes com estrutura e/ou comportamento similares. Essas classes podem ser modeladas como abstrações distintas e não relacionadas, ou de uma forma melhor, na qual a estrutura e o comportamento comum são extraídos e colocados em uma classe mais genérica, da qual as classes especializadas herdam.

A Figura 7-16, abaixo, apresenta uma Herança, na qual as classes B e C herdam da classe A.

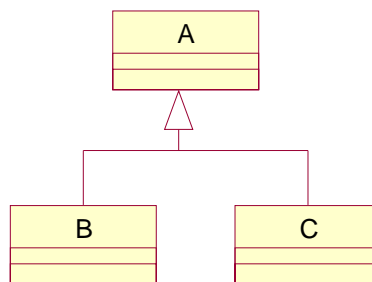


Figura 7-16 Herança

Uma Herança no Modelo de Análise está refletida no Projeto em duas partes: uma parte trata da Herança de comportamento, e a outra da Herança dos dados. A parte relacionada

ao comportamento está sendo tratada pelas UNs, enquanto que a parte relacionada aos dados é tratada pelas Classes de Acesso no MA.

Na CN, temos que as UN_B e UN_C devem realizar, pelo menos, o mesmo comportamento apresentado pela UN_A. Caso as classes B e C do Modelo de Análise acrescentem algum comportamento àquele herdado da classe A, este comportamento deve ser acrescido às suas respectivas UNs. A Figura 7-17, abaixo, ilustra a Herança de comportamento para este exemplo genérico entre as classes B, C e A.

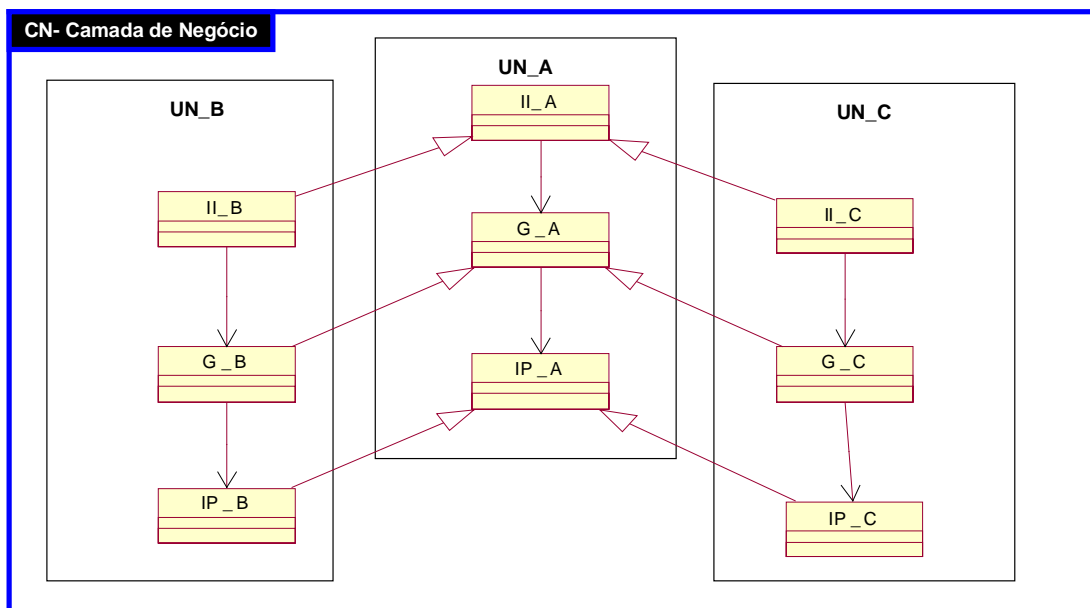


Figura 7-17 CN - Herança

O MA apresenta as Classes de Acesso *B* e *C* que herdam da Classe de Acesso *A*. As Classes de Acesso *B* e *C* apresentarão os atributos da Classe de Acesso *A* e seus métodos de acesso, acrescidos dos seus atributos próprios e dos respectivos métodos de acesso.

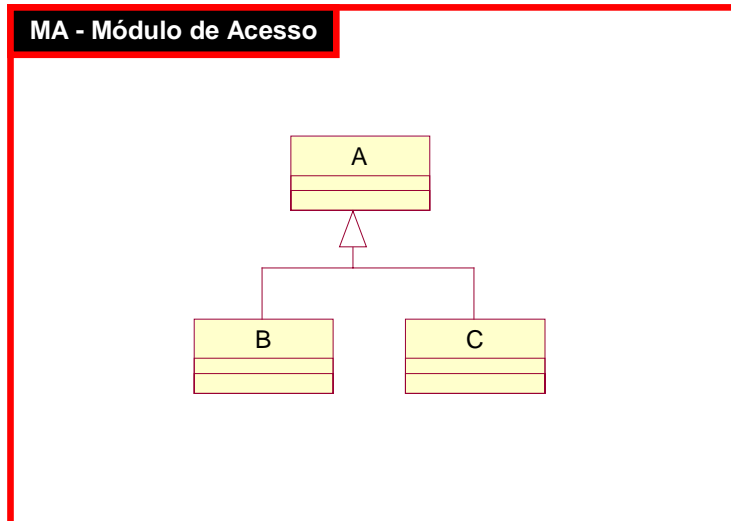


Figura 7-18 MA - Herança

No caso da classe *A*, do Modelo de Análise ser uma classe abstrata, sua *UN_A* e sua respectiva Classe de Acesso *A* também serão abstratas.

No caso de haver Herança somente de comportamento ou de dados, não haverá a necessidade da Herança ocorrer na CN e no MA. A Herança na CN ocorre especificamente para Herança de comportamento, enquanto que a Herança no MA ocorre especificamente para a Herança de dados.

A seguir, nas figuras (Figura 7-19, Figura 7-20 e Figura 7-21), apresenta-se um Modelo de Análise e seu respectivo Projeto com CN e MA para um exemplo de Herança.

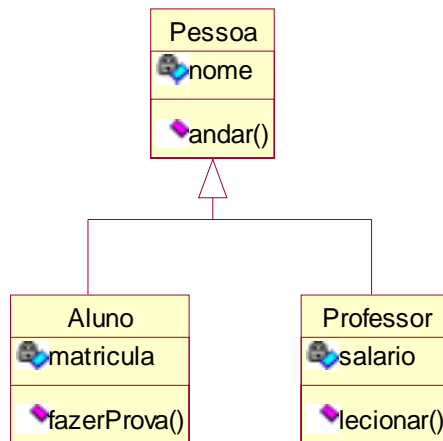


Figura 7-19 Exemplo de Herança : Aluno,Professor,Pessoa

Neste exemplo, a classe *Pessoa* apresenta o comportamento *andar* e o dado *nome*, comuns às classes *Aluno* e *Professor*. Logo, a classe *Aluno*, no Modelo de Análise, possui os dados *nome* e *matrícula* e apresenta os comportamentos de *andar* e *fazerProva*.

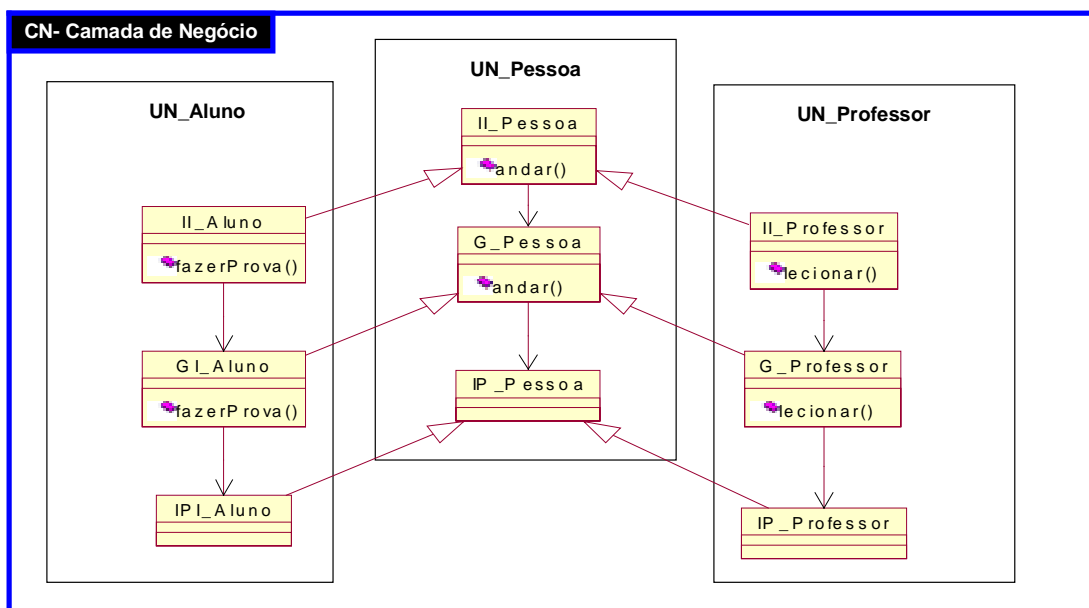


Figura 7-20 Exemplo Aluno, Professor ->Pessoa (CN-Comportamento)

A UN_Aluno oferece a funcionalidade *andar*, herdada das respectivas classes II_Pessoa e G_Pessoa da UN_Pessoa, e acrescenta a funcionalidade *fazerProva*, específico da sua UN. A UN_Professor também oferece o comportamento *andar*, herdado da UN_Pessoa, e acrescenta o comportamento *lecionar*, específico da sua UN. Neste exemplo, não foram mostrados, propositadamente, os métodos das classes IP, a fim de evitar a falsa impressão de que as classes II, G e IP possuem necessariamente os mesmos métodos.

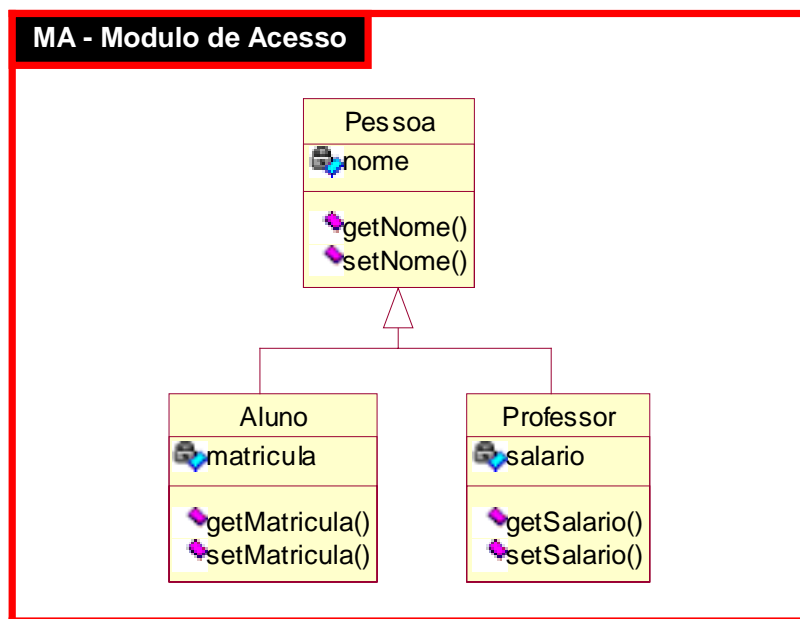


Figura 7-21 Exemplo Aluno, Professor, Pessoa (MA-Dados)

A Classe de Acesso *Aluno* apresenta os atributos *nome* e *matricula* e os métodos de acesso *getNome*, *setNome*, *getMatricula* e *setMatricula*. O atributo *nome* e os respectivos métodos de acesso foram herdados da Classe de Acesso *Pessoa*. A Classe de Acesso *Professor* apresenta, além do atributo *nome* e de seus métodos de acesso, herdados da Classe de Acesso *Pessoa*, o atributo *salario*, assim como, seus métodos de acesso.

7.1.4 Visão Global

Um Modelo de Análise apresenta classes e seus relacionamentos. O tratamento proposto na Metodologia para estes relacionamentos foi apresentado separadamente nas seções

acima. Ao realizar um Projeto completo, estes tratamentos são sobrepostos, tanto na CN, quanto no MA. Logo, o comportamento apresentado em uma UN é relativo à sua respectiva classe do Modelo de Análise, acrescido do comportamento gerado por seus relacionamentos com outras classes. Já os dados tratados por uma Classes de Acesso no MA são específicos da classe correspondente no Modelo de Análise, acrescido dos dados provenientes dos relacionamentos desta classe com outras classes do Modelo de Análise.

7.2 Concorrência

Nesta seção, estaremos considerando, de acordo com a Metodologia, aspectos de concorrência na Camada de Negócio para o sistema projetado. Como a Metodologia realiza o tratamento diferenciado entre dados e funcionalidades, os aspectos de concorrência devem ser tratados de acordo com esta divisão.

7.2.1 Concorrência baseada em Dados

A concorrência baseada em dados trata dos aspectos relativos à forma pela qual os dados do sistema serão “acessados” pelos usuários do mesmo. Todos os dados persistentes do sistema encontram-se na Camada de Persistência. Para “acessar” tais dados, estes são encapsulados em objetos e disponibilizados no Módulo de Acesso. Estes objetos são instâncias das Classes de Acesso, criadas pela Metodologia na etapa de Definição do Módulo de Acesso.

O Módulo de Acesso, por ser o local onde, em tempo de Projeto, definem-se as Classes de Acesso, e em tempo de execução, localizam-se os objetos que encapsulam os dados utilizados pelo sistema, representa, naturalmente, o lugar onde as políticas de acesso aos dados do sistema devem ser definidas.

A Figura 7-22 apresenta o Módulo de Acesso em tempo de execução. O tratamento da concorrência baseada em dados deve ser realizado pelo Módulo de Acesso, já que ele é o componente responsável pelos dados do sistema.

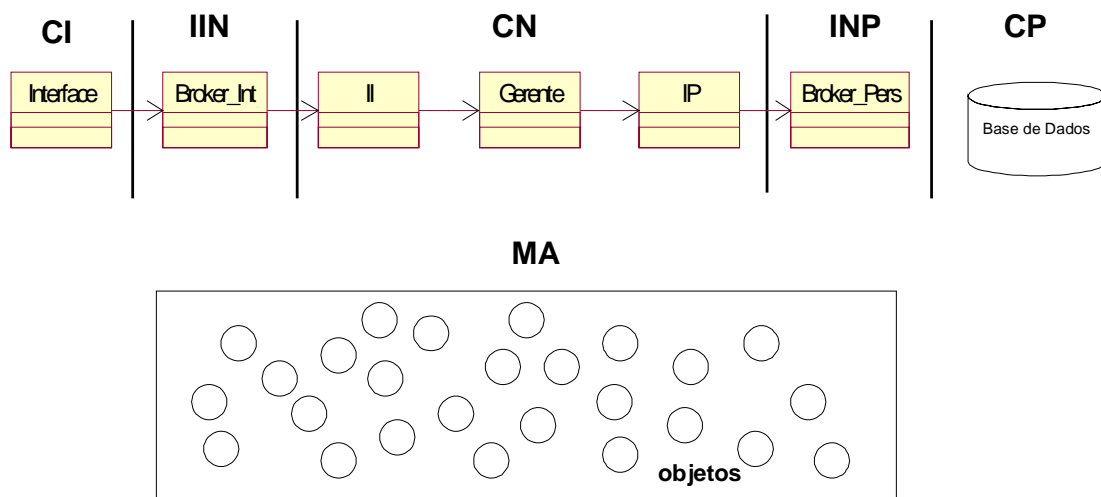


Figura 7-22 Módulo de Acesso, como Repositório de Classes de Acesso

Não faz parte do escopo deste trabalho definir as possíveis formas de se realizar este controle baseado em concorrência sobre os dados do sistema, entretanto, no Apêndice B, apresenta-se um projeto real em que a Solução proposta por este trabalho está sendo utilizada. Neste apêndice, discute-se, brevemente, a solução adotada.

7.2.2 Concorrência baseada em Funcionalidades

A concorrência baseada nas funcionalidades trata da forma pela qual as funcionalidades do sistema são disponibilizadas para os “usuários”. Na solução proposta, toda funcionalidade do sistema encontra-se tratada pela Camada de Negócio, distribuída entre suas Unidades de Negócio.

Toda a funcionalidade disponibilizada para a Camada de Interface encontra-se encapsulada nas classes Intermediárias de Interface (II) das Unidades de Negócio. Na etapa 6.2.3-Detalhamento das Unidades de Negócio (UNs), deve-se analisar a quantidade de acessos que cada UN deve suprir e deve-se tomar as decisões com relação a classes II, que permitirão os acessos, e a suas classes Gerentes. No Apêndice B, uma solução para a Camada de Negócio é apresentada. Nesta solução, as classes IIs e Gerentes são classes Singleton [Gamma95], que realizam toda disponibilização das funcionalidades das UNs. Estas classes estão ilustradas na figura abaixo.

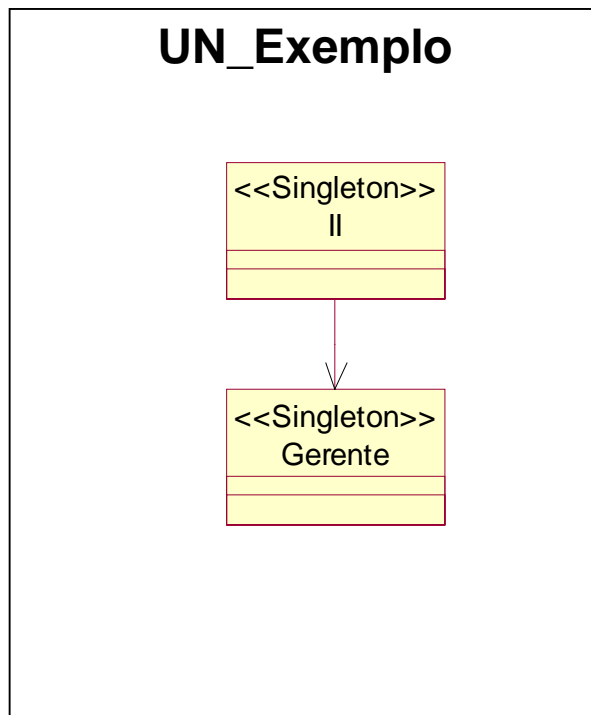
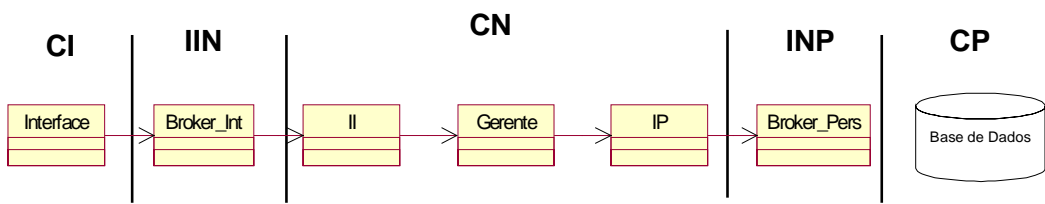
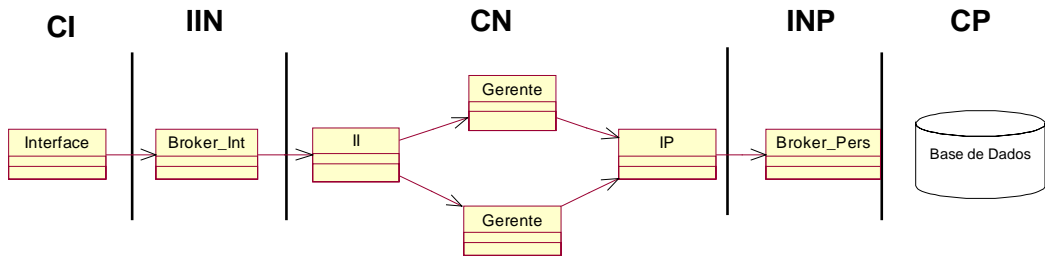


Figura 7-23 Classes II e Gerente como Singleton

Nesta solução baseada em classes Singleton, uma possível forma de tratar concorrência baseada nas funcionalidades, oferecidas pela UN, seria através da sincronização dos métodos das classes II e Gerente. Estas classes ainda podem apresentar a “inteligência” de controlar os números de acesso que elas respondem. No caso deste número crescer demasiadamente, eles podem instanciar novos objetos destas classes, de forma que cada objeto sempre comporte uma determinada quantidade de acessos. A Figura 7-24 ilustra a situação em que o Gerente “percebe” que não está dando suporte à quantidade de acessos e instancia um novo Gerente para dividir o trabalho com ele.



Situação 1: número de acessos Normais



Situação 2: número de acessos excessivos

Figura 7-24 Divisão de trabalho (instanciação de um novo Gerente)

8 Trabalhos Futuros

A seguir, são listados alguns trabalhos que podem ser realizados como continuidade dos temas abordados nesta dissertação.

8.1 Ferramenta

Como continuidade deste trabalho, pode-se implementar uma ferramenta para auxiliar no desenvolvimento e na organização do Projeto, de acordo com a Arquitetura e a Metodologia propostas.

Esta ferramenta receberia como entrada um Diagrama de Classes, em UML, e poderia gerar toda a estrutura de Projeto em uma linguagem de Implementação, como por exemplo, *Java*.

A Figura 8-1, abaixo, mostra um Diagrama de Classes construído com a ferramenta *Rational Rose* [ROSE]. Este diagrama seria a entrada da ferramenta aqui proposta.

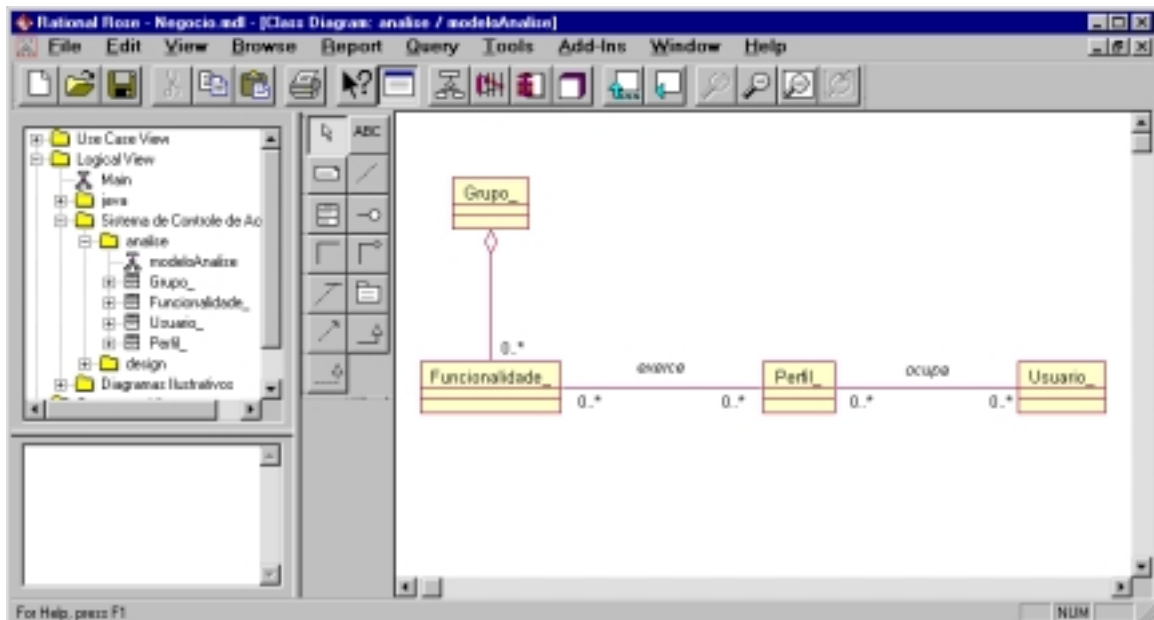


Figura 8-1 Exemplo de Diagrama de Classes na ferramenta Rational Rose

O código gerado pela ferramenta está ilustrado na Figura 8-2 abaixo. Este é o ambiente de programação *Visual Age for Java* [VA-JAVA]. Nota-se que, de acordo com a Arquitetura, o sistema apresenta-se organizado nos pacotes *interface*, *interface_Negócio*, *negócio*, *negócio_Persistência* e *móduloDeAcesso*. Estes pacotes representam as camadas e o Módulo de Acesso, apresentados no capítulo 5 (A Arquitetura).

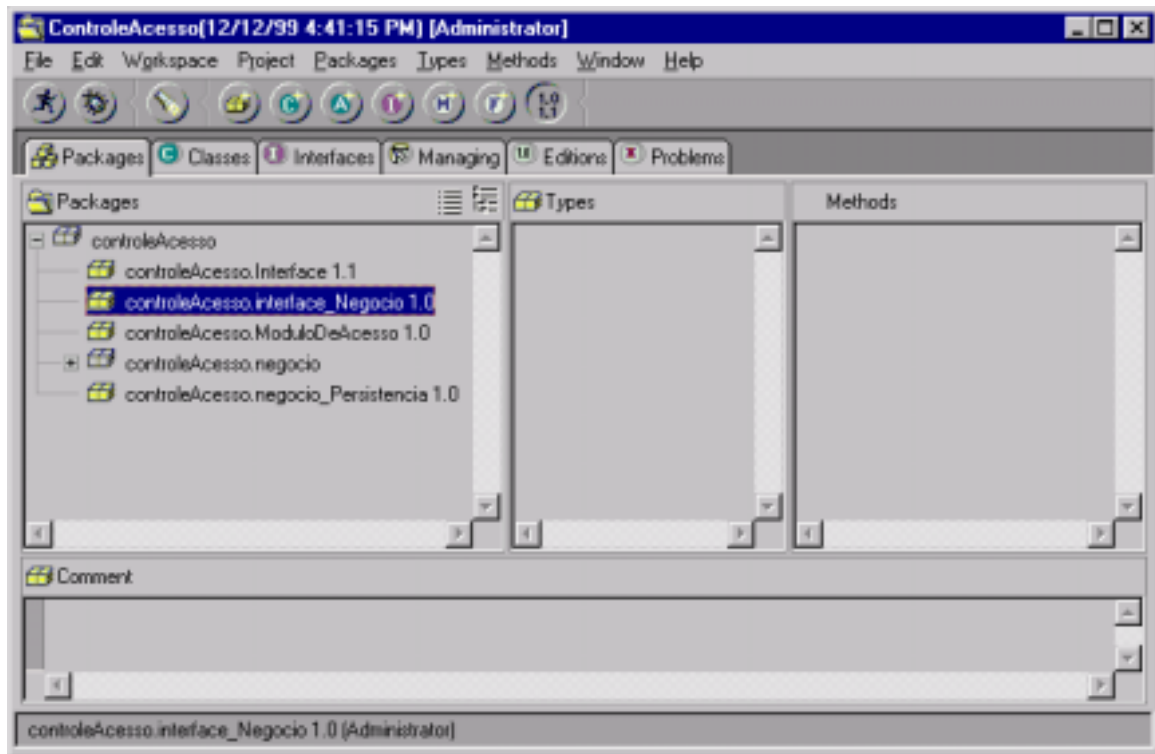


Figura 8-2 Exemplo na ferramenta Visual Age for Java

A Figura 8-3, abaixo, mostra, agora, o pacote *negócio* e seus subpacotes (as Unidades de Negócio).

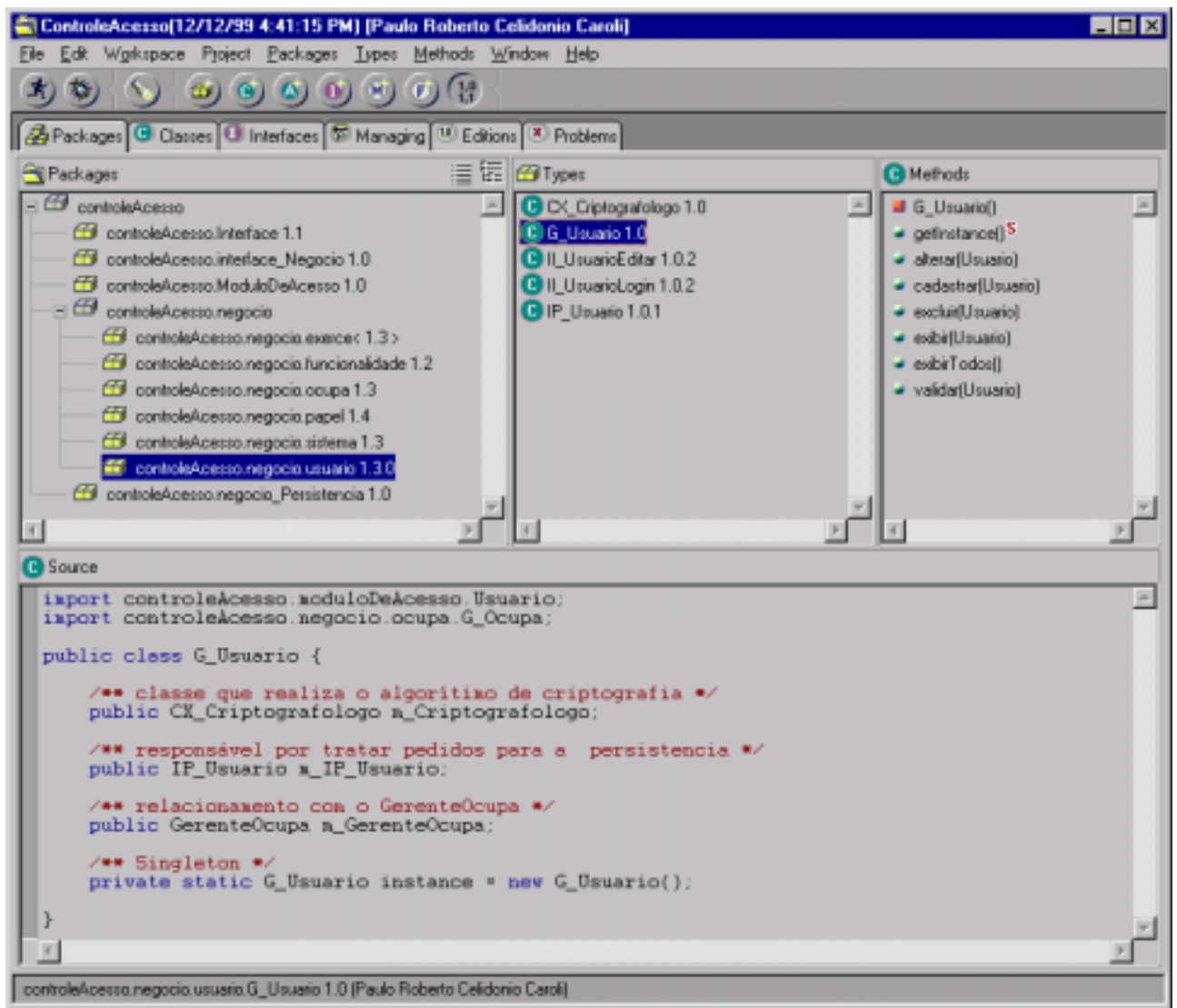


Figura 8-3 Camada de Negócio e suas UNs no Visual Age for Java

De acordo com a Metodologia, o pacote negócio está organizado nos pacotes *sistema*, *usuario*, *papel*, *funcionalidade*, *exerce* e *ocupa*, representando as UNs da Camada de Negócio. Cada um desses subpacotes apresenta as classes II, Gerente, IP e possíveis classes auxiliares (CX). A classe Gerente possui todas as funcionalidades apresentadas por sua respectiva classe no Diagrama de Classes. Também cabe ressaltar que esses pacotes importam as suas respectivas Classes de Acesso, apresentadas na Figura 8-4 a seguir.

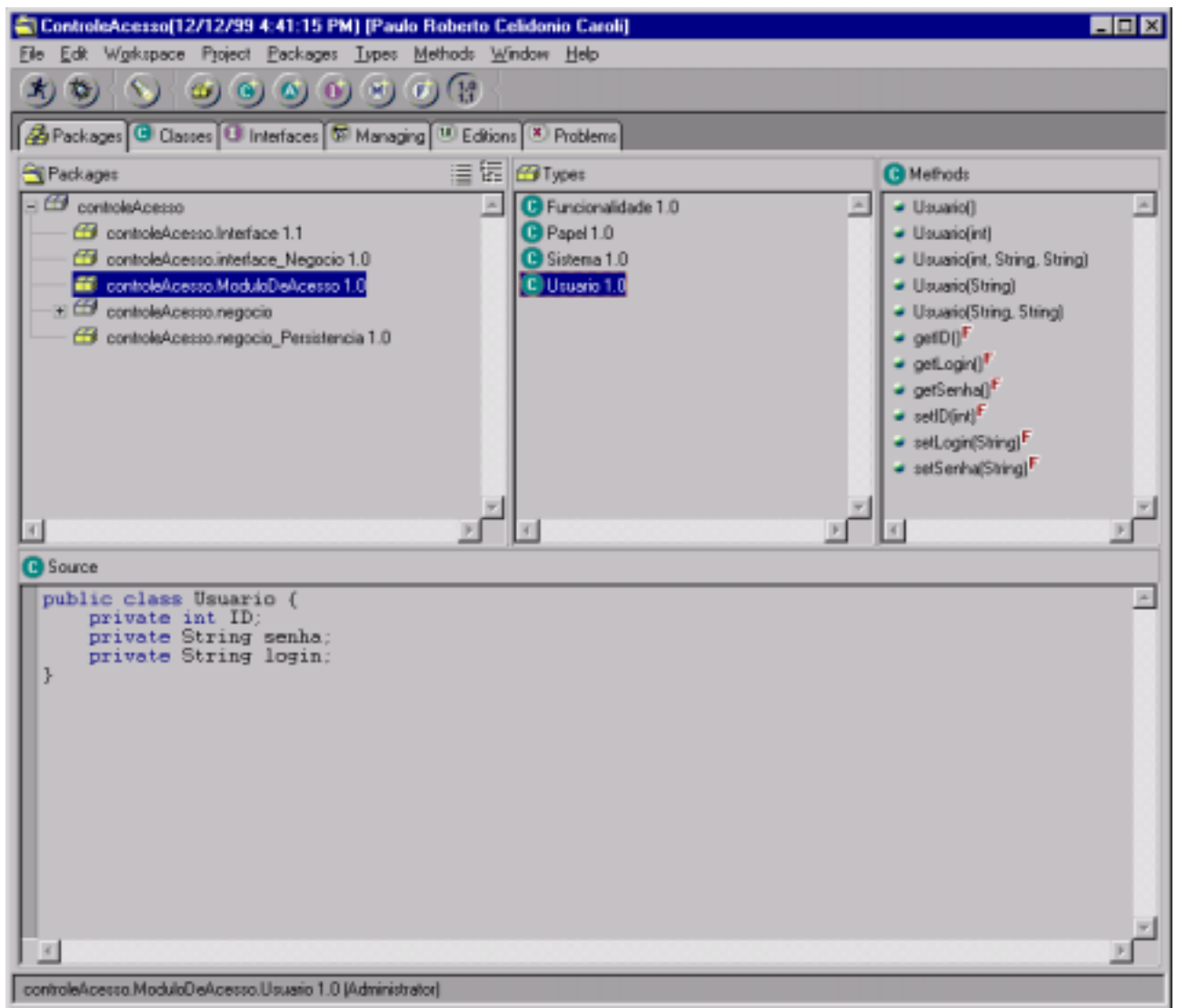


Figura 8-4 o Módulo de Acesso (MA) no Visual Age for Java

A Figura 8-4, apresentada, ilustra o pacote móduloDeAcesso e suas Classes de Acesso. Nota-se que, para cada classe do Diagrama de Classes, temos uma classe de Acesso, e estas Classes de Acesso possuem os métodos de acesso relativos aos atributos e aos relacionamentos existentes nas respectivas classes no Diagrama de Classes.

A partir da organização gerada pela ferramenta, o desenvolvedor precisaria realizar as seguintes tarefas para completar o desenvolvimento da Camada de Negócio:

1. Para cada uma destas UNs, deve-se analisar que funcionalidades serão disponibilizadas para a Interface (encapsuladas nos IIs), que funcionalidades acarretam ou necessitam de informações persistentes (deve estar nos IPs) e se existe a necessidade do Gerente delegar funcionalidades para outras classes (Classes Auxiliares (CX)).
2. Verificar para cada UN a necessidade de importar Classes de Acesso. Não deve haver a necessidade de importar Classes de Acesso para as quais suas respectivas classes no Diagrama de Classes não se relacionem. Caso isto ocorra, alguma inconsistência deve estar ocorrendo no Diagrama.
3. Implementar o código das classes nas UN. Deve ser verificado se a classe II pode acessar, somente, seu respectivo Gerente, e se este somente pode ter acesso a seus respectivos IPs; além de ter de verificar com quais Gerentes ele pode se relacionar (aqueles com os quais suas respectivas classes no Diagrama de Classes se relacionavam). Para implementar os IPs, é necessário que estes conheçam a interface da Intercamada Negócio/Persistência, para que eles utilizem as classes devidas.

Os passos apresentados podem ser obtidos de forma assistida, fazendo com que a ferramenta auxilie o desenvolvimento de acordo com a solução proposta por este trabalho.

O trabalho apresentado nesta dissertação centraliza-se no desenvolvimento da Camada de Negócio, mas a ferramenta pode auxiliar no desenvolvimento de todo o sistema, podendo apresentar soluções também para as camadas de Interface e Persistência.

Idealizando-se um nível mais elevado de abstração, esta ferramenta poderia ser um *framework* [Mattsson96] para a implementação de Sistemas em 3 Camadas. O Projeto resultante da utilização da Metodologia, para a construção da Camada de Negócio, seria um *hot spot*, para a Camada de Negócio. Outros *hot spots* deste *Framework* seriam geradores de projetos para as camadas de Interface e Persistência. Um usuário deste *Framework* poderia utilizá-lo, por exemplo, tomando as seguintes decisões:

- para a Interface: Utilizar o *hot spot Servlets* (o *Framework* geraria classes auxiliares para o desenvolvimento de classes *Java Servlets* na camada de Interface e *Brokers* de comunicação para a Intercamada Interface/Negócio);
- para a Camada de Negócio: Utilizar a comunicação entre IIs, Gerentes e IPs (O *Framework* realizaria a solução proposta por esta dissertação);
- para a Camada de Persistência: Utilizar uma Base de Dados Relacional (o *Framework* utilizaria o *framework Persistence Builder* [PERSISTENCE BUILDER] para gerar a Camada de Persistência e a Intercamada Negócio/Persistência, e ainda auxiliaria no tratamento de concorrência no MA).

8.2 Controle de Transação

Muito ainda pode ser pesquisado em relação a como realizar o controle de transação das operações realizadas pelo sistema. O tratamento em relação a transações não é um assunto tratado por este trabalho; entretanto, este aspecto deve ser analisado para a implementação de sistemas.

Para o sistema apresentado no Apêndice B, o aspecto relativo a transação foi analisado e um tratamento específico foi realizado. Todavia, este tratamento não ficou encapsulado em uma única camada. Trabalhos futuros em relação a este aspecto podem discutir este problema, analisando a viabilidade de encapsular o tratamento de transação em um única camada, na tentativa de que as camadas sejam realizadas independentes deste aspecto.

Devem também ser analisado onde terminam transações de Banco de Dados e onde começam as do sistema. Este trabalho pode verificar a relação “custo/benefício” para estes aspectos. O problema atualmente relacionado a este tipo de controle é que ferramentas de Banco de Dados (SGBDs) oferecem mecanismos de tratamento acoplados a suas Bases de Dados. Ao optar-se por utilizar tais tratamentos na Camada de Persistência, pode-se estar colocando regras de negócio em outra camada; em contrapartida, ao optar-se por realizar este tratamento na Camada de Negócio, pode-se

aumentar em muito o trabalho de implementação desta camada. Estudos deste tipo podem ser realizados, indagando-se até que ponto a utilização de *features* disponíveis em ferramentas é benéfica para o desenvolvimento de sistemas.

8.3 Separação Física

Todo trabalho aqui apresentado considera a separação lógica em camadas. Deve ser realizada uma análise em relação à separação física entre as camadas, considerando-se aspectos de eficiência e possíveis tecnologias para a implementação de cada camada.

Pode ser estudada a viabilidade de construir uma ferramenta auxiliar para a verificação da quantidade de comunicação entre componentes dentro de um sistema. Em um Sistema em 3 Camadas, esta ferramenta pode ser utilizada para determinar a separação física entre os componentes logicamente separados em 3 camadas, permitindo, assim, obter o melhor desempenho do sistema.

9 Conclusões

Este capítulo busca verificar se os objetivos propostos por esta dissertação foram atingidos. Resumidamente, estes objetivos eram:

- Auxiliar o acompanhamento de desenvolvimento de sistemas;
- Garantir a continuidade de sistemas grandes e suscetíveis a alterações e evoluções;
- Auxiliar na comunicação entre as diferentes pessoas envolvidas no projeto de sistemas;
- Dar suporte ao desenvolvimento Orientado a Objetos, coexistindo com diferentes paradigmas;
- Apresentar uma documentação efetiva do projeto.

A base de validação desses objetivos baseia-se na experiência da aplicação dos conceitos propostos nesta dissertação em projetos reais de sistemas de informação para a Web. O Apêndice A apresenta uma breve descrição desses projetos.

A solução apresentada ajudou na etapa inicial de desenvolvimento, apresentando o ponto de partida para os projetos e, durante todo o andamento dos projetos, ajudou a realizar as decisões, permitindo discussões do tipo “esta funcionalidade deveria estar nesta camada?” ou “se não existe este relacionamento no Diagrama de Classes, por que estou importando esta Classe de Acesso nesta UN?”

De acordo com as constatações em resultados práticos, verificamos que a utilização da solução auxiliou na construção de Sistemas em 3 Camadas, fornecendo conceitos e vocabulário comuns para o tratamento deste tipo de sistema, apresentando as vantagens esperadas, mas não garantiu, por si só, o sucesso do desenvolvimento do projeto, ficando

este diretamente relacionado à existência e à qualidade do Modelo de Análise e à execução do Projeto.

A utilização da divisão em camadas auxiliou o desenvolvimento, a manutenção e o gerenciamento do Projeto realizado. A utilização das idéias (Arquitetura, Metodologia e Heurísticas) apresentadas mostrou-se útil durante o desenvolvimento do Projeto, facilitando a comunicação entre as equipes de desenvolvimento e entre desenvolvedores e o gerente do projeto.

Os diferentes projetos realizados utilizando-se a solução proposta apresentaram situações reais que puderam validar algumas das características priorizadas. Alguns casos relevantes vivenciados foram:

- Alterações do Modelo de Análise após o sistema já estar parcialmente implementado;
- Necessidade de um novo desenvolvedor compreender o código de um sistema já gerado;
- Erro de Projeto no controle de transação.

Ao realizar alterações em um sistema em funcionamento, causadas por mudanças no Diagrama de Classes, podê-se constatar que a existência de um mapeamento entre este diagrama e o Projeto, e de passos bem definidos pela Metodologia facilitaram a realização das alterações, que eram analisadas e, facilmente, refletidas para as estruturas internas (camadas, UNs, Módulo de Acesso).

O rápido entendimento de um novo desenvolvedor sobre o projeto mostrou que a Arquitetura auxilia a integração da equipe de desenvolvimento. A maior parte do tempo gasto pelo novo desenvolvedor foi com as tecnologias envolvidas no projeto, e não na compreensão do Projeto do sistema.

Após perceber que havia ocorrido um erro conceitual relativo ao tratamento de transações, foi necessário repensar e alterar implementações em diferentes partes do projeto para atingir o tratamento correto das transações. A clareza na estruturação do projeto decorrente da Arquitetura e da Metodologia, em muito, auxiliou na realização deste trabalho.

A Arquitetura mostrou-se ser útil não só para os desenvolvedores, como também ser uma importante ferramenta para o gerenciamento do projeto, possibilitando um melhor planejamento dos recursos necessários em relação a cada “peça” formadora de todo o sistema.

A Arquitetura também permitiu uma forma de comunicação entre os desenvolvedores envolvidos no sistema, definindo cargos e tarefas dentro do desenvolvimento do mesmo. Por exemplo, no Projeto apresentado no apêndice B, em determinado momento, após o início da implementação do sistema, decidiu-se contratar um *Designer* Gráfico para tratar dos aspectos relativos ao *Design* da Interface. Esta nova contratação não afetou as outras camadas, e a pessoa que ocupou este novo cargo somente precisou interagir com as pessoas envolvidas com a Camada de Interface.

O projeto apresentado no Apêndice B apresentou vários aspectos “complicadores”, que poderiam impactar o andamento do projeto:

- Necessidade de comunicação com sistemas legados;
- Constantes alterações no diagrama de classes e requisitos do sistema;
- Necessidade de evolução dos requisitos;

A utilização da solução tornou-se um importante alicerce para a execução do projeto, dando suporte à limitação do impacto no projeto destes “complicadores”. A solução mostrou-se também escalável, pois tratou de forma linear projetos pequenos (pequeno

número de classes), projetos relativamente grandes (grande número de classes) e projetos que cresceram de tamanho durante sua implementação.

O Projeto resultante da solução proposta apresenta um bom nível de estruturação, e os principais aspectos almejados foram atingidos com êxito. São eles: manutenibilidade, organização, documentação e auxílio para desenvolvedores e gerentes.

Em resumo, a Arquitetura apresentou uma organização para o desenvolvimento de Sistemas em 3 Camadas, explicitando as entidades envolvidas na Arquitetura, suas responsabilidades, e os aspectos de comunicação entre elas.

A Metodologia apresentou uma solução de Projeto para o desenvolvimento da Camada de Negócio em Sistemas em 3 Camadas. Esta solução mostrou cumprir os objetivos desejados, e isto foi verificado pelos projetos reais para os quais esta Metodologia foi aplicada.

Apêndice A

Este apêndice apresenta uma breve descrição dos sistemas projetados utilizando-se as idéias propostas neste trabalho.

A.1 Projeto Wandirtel

Este é um Projeto, em andamento, que está sendo realizado pelo LES para a Petrobrás. Este projeto consiste em 7 sistemas a serem realizados e disponibilizados em 2 anos. Abaixo são apresentados os sistemas que estão sendo desenvolvidos atualmente neste primeiro ano do projeto.

Atualmente, a equipe de desenvolvimento consiste em um gerente, três programadores *Java*, dois estagiários e uma programadora visual. As tecnologias utilizadas são: *Java*, *Java Servlets*, *Java Script*, *Oracle*, *Rational Rose*, *Visual Age for Java* e o *framework Persistence Builder*.

A.1.1 Gerência de Materiais

Este sistema objetiva controlar os materiais formadores da rede de telecomunicação da Petrobrás.

O Diagrama de Classes deste sistema é formado por 28 classes e 40 relacionamentos entre elas.

A.1.2 Gerência de Configuração

Este sistema objetiva controlar a configuração dos materiais formadores da rede de telecomunicação da Petrobrás. Ele apresenta uma interseção com a Gerência de Materiais apresentada acima e precisa comunicar-se com sistemas legados, já existentes em uso na empresa.

O Diagrama de Classes deste sistema é formado por 40 classes e 51 relacionamentos entre elas.

A.1.3 Gerência de Clientes

Este sistema objetiva controlar os clientes e os serviços disponibilizados pela rede de telecomunicação da Petrobrás, além de apresentar uma interseção com a Gerência de Materiais e com a Gerência de Configuração, apresentadas acima. Ele precisa comunicar-se com sistemas legados, já existentes em uso na empresa.

O Diagrama de Classes deste sistema é formado por 26 classes e 28 relacionamentos entre elas.

A.2 Sistema - Controle de Acesso

O objetivo deste sistema é controlar o acesso que diferentes usuários apresentam em relação a um sistema, através de perfis que estes ocupam. A partir deste controle, é disponibilizada uma interface *customizada* a cada usuário, contendo apenas aquelas funcionalidades que o usuário em questão pode acessar.

O Diagrama de Classes deste sistema é formado por 6 classes e 5 relacionamentos entre elas.

Este sistema foi desenvolvido no LES por uma equipe formada por um gerente e 2 desenvolvedores. As tecnologias utilizadas foram: *Java, Java Servlets, Java Script, Java JDBC, SQL Server, Rational Rose e Visual Age for Java.*

A.3 Um Processo de Construção de Geradores

Consiste em um processo definido por Luiz Paulo Alves Franca, para seu trabalho de doutorado, em andamento, na PUC-Rio. Abaixo, segue a descrição realizada pelo próprio sobre seu trabalho:

“A utilização de Geradores de Aplicação no processo de desenvolvimento é uma forma de aumentar a produtividade da equipe e a qualidade do produto final. Os geradores, normalmente a partir do preenchimento de uma especificação, transformam esta especificação num artefato de software (documentação, programa, "script" de banco de dados, toda aplicação, etc). Apesar das inúmeras vantagens de um gerador, a sua utilização não é muita difundida, principalmente, porque os geradores são muitos específicos e o seu processo de construção tem um custo elevado. O objetivo do nosso trabalho é viabilizar um maior uso da técnica de Gerador de Aplicação, pelas equipes de desenvolvimento. Na nossa abordagem, o problema da especificidade de um Gerador foi superado por um processo de customização do gerador, já a questão do custo foi minimizada pela simplificação do processo de construção, eliminando a necessidade de desenvolvimento de tradutores sofisticados para transformar a especificação no produto final. Foram realizados alguns experimentos com sucesso utilizando o processo proposto, dentre eles, destacamos a construção de um gerador para uma aplicação Web em 3 camadas composta por fontes em *Java*, HTML, e *JavaScript*. Neste experimento, a modularização imposta pela arquitetura adotada facilitou o processo de geração, pois ela garantia um mapeamento bastante claro das classes conceituais nas classes de implementação e uma forma organizada para o código-fonte. “

A arquitetura proposta, mencionada na sua descrição, refere-se à Arquitetura apresentada neste trabalho.

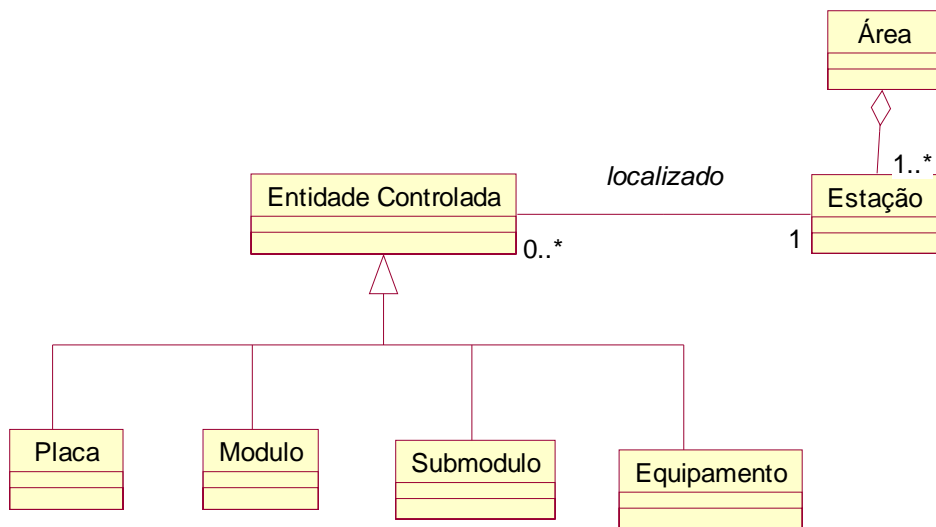


Figura B. 2 Diagrama de Classes 2

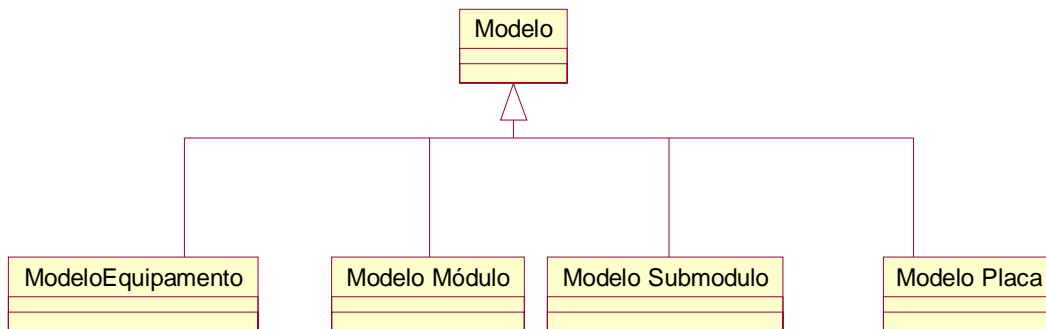


Figura B. 3 Diagrama de Classes 3

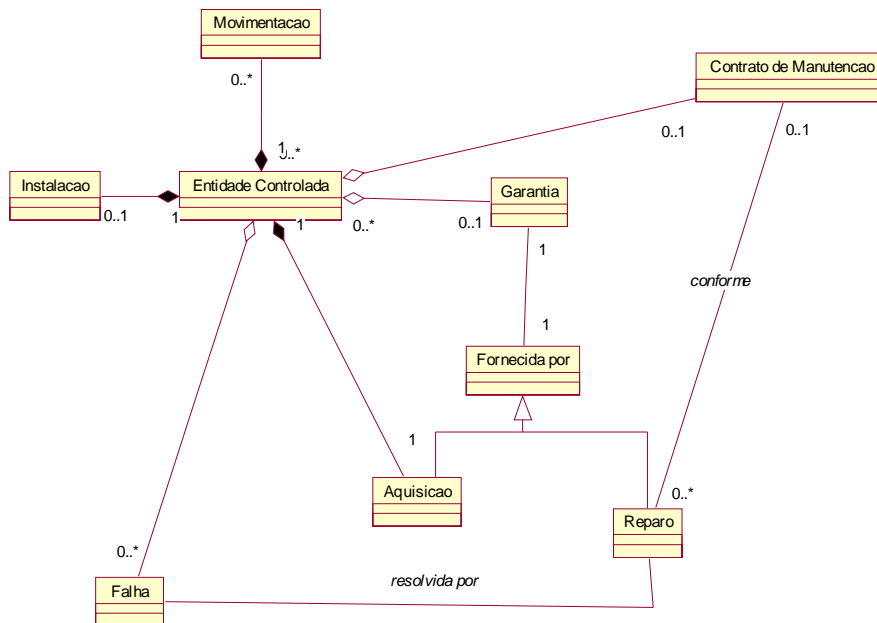


Figura B. 4 Diagrama de Classes 4

Nestes diagramas são apresentados somente as classes e seus relacionamentos. Os atributos e os métodos não foram mostrados para manter os diagramas mais “limpos” e por questões de direitos autorais do Projeto.

B.2 Código Gerado

O código deste sistema foi gerado em *Java* na ferramenta Visual Age for Java [VA-JAVA]

A Figura B. 5, abaixo, mostra a estrutura de pacotes do sistema.

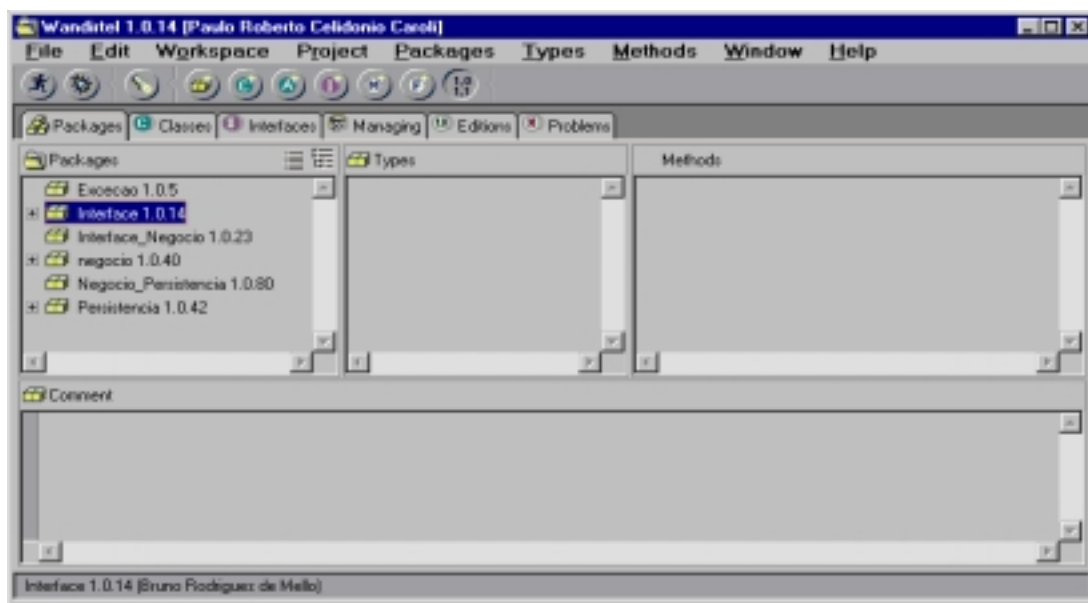


Figura B. 5 Estrutura de Pacotes

A estrutura de pacotes, ilustrada na Figura B. 5, segue a organização proposta neste trabalho, apresentando os pacotes *interface*, *interface_Negócio*, *negócio*, *negócio_Persistência* e *persistência*. O pacote *móduloDeAcesso* não aparece devido a limitações apresentadas pela ferramenta *Persistence Builder* [PERSISTENCE BUILDER], a qual foi utilizada para gerar o mapeamento entre as Classes de Acesso e as tabelas na Base de Dados. Na atual versão desta ferramenta, não é possível gerar as Classes de Acesso em um pacote “nosso”. Elas são geradas automaticamente pelo *Persistence Builder*, exatamente com os métodos de *get* e *set* para seus atributos, e encontram-se em uma estrutura de pacotes por ele definida. Esta estrutura armazena também informações relativas à persistência, por este motivo decidimos localizá-la no pacote *persistência* (Figura B. 12). O pacote *exceção* foi inserido nesta estrutura para realizar ou encapsular todo o tratamento de exceção do sistema.

B.2.1 Camada de Interface

Assim como acontece na Camada de Negócio, foi criado um pacote para cada classe que tínhamos no Diagrama de Classes; isto pode ser notado na Figura B. 6, abaixo. Estes

pacotes organizam todas as classes de interface relativas à respectiva classe do Diagrama de Classes.

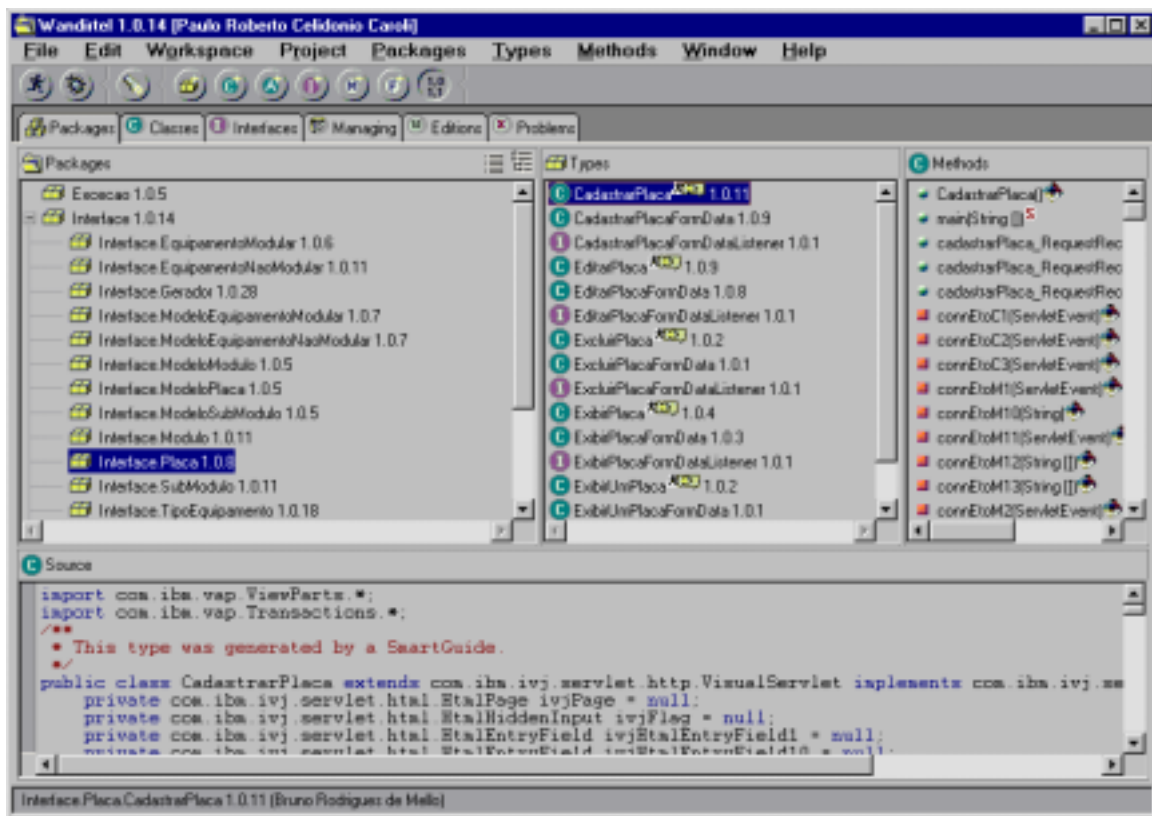


Figura B. 6 interface e seus pacotes

A Interface dos sistemas é implementada utilizando-se classes *Java Servlets*, que geram código HTML para ser interpretado pelos *Browsers*. Utilizamos a ferramenta *Servlet Builder* do *Visual Age for Java* [SERVLET BUILDER] para construir visualmente os *Servlets*. Primeiramente, o projeto visual da interface foi realizado por uma programadora visual, e em uma etapa seguinte, a programação das funcionalidades, realizadas pela interface, foi codificada.

B.2.2 Intercamada Interface/Negócio

Esta Intercamada é composta por *Brokers*, aqui chamados de Conversores, que foram responsáveis pela comunicação com as Classes IIs, necessárias para realizar toda a conversão dos tipos de dados dos objetos para os tipos necessários que aparecerem na interface (*Strings* e lista de *Strings*).

Cada classe do Diagrama de Classes gerou um Conversor.

A Figura B. 7, abaixo, mostra o pacote *interface_Negócio* e suas classes. Nela, podemos notar a inclusão de classes IIs, necessárias para o Conversor (este caso específico foi ilustrado para a classe *EquipamentoModularConversor*).

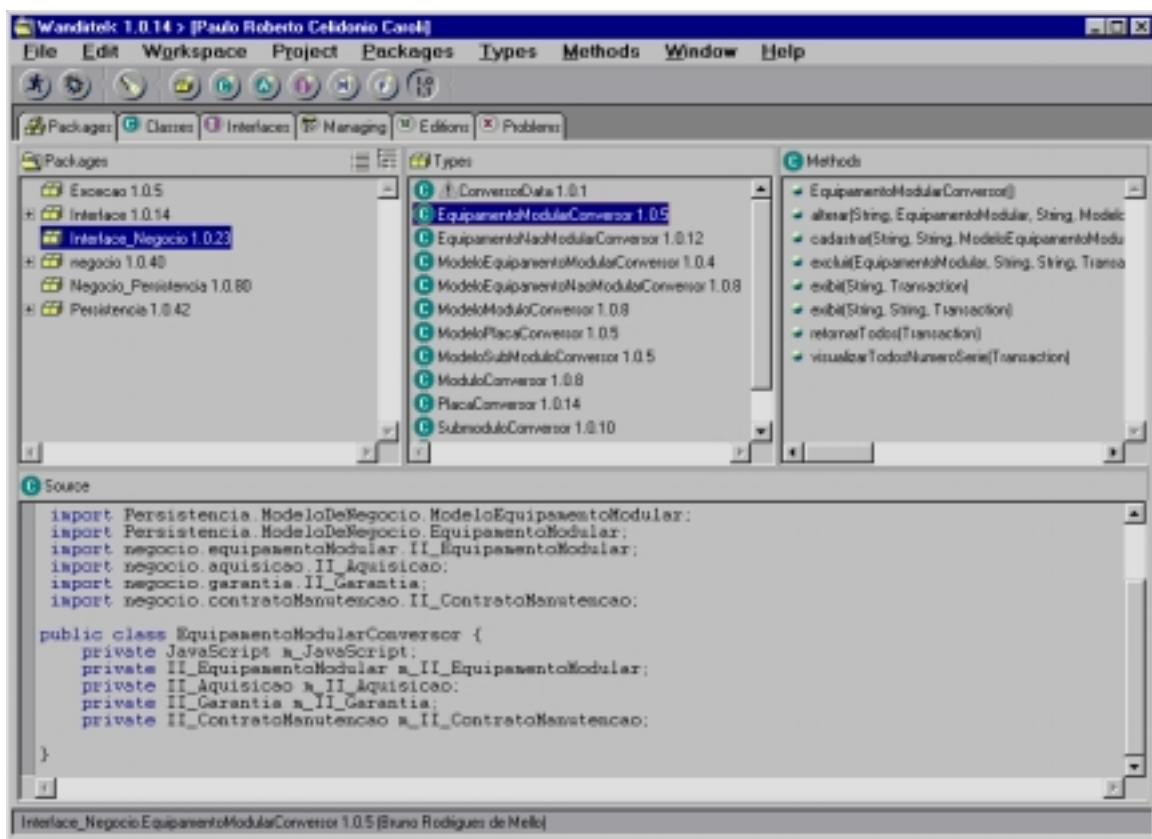


Figura B. 7 pacote *interface_Negocio*

B.2.3 Camada de Negócio

Esta camada foi realizada de acordo com a Metodologia. Para cada classe do Diagrama de Classes, foi gerado uma Unidade de Negócio com as Classes II, Gerente e classes auxiliares quando necessárias. Cada Unidade de Negócio corresponde a um pacote *Java*.

As classes II e as classes Gerentes são Singletons [Gamma95]. Durante o projeto destas classes, foi verificada a existência de herança de funcionalidades, ou seja, funcionalidades comuns que eram apresentadas pelas classes; e com isto, puderam ser criadas classes genéricas II e Gerentes, que trabalhavam com tipos Genéricos (genericidade). Estes tipos genéricos tornavam-se, novamente, específicos pela classe II, para passar o objeto para o *Broker* da Interface/Negócio ou para o *Broker* da Negócio/Persistência.

A classe IP foi suprimida deste projeto por decisão dos projetistas, fazendo com que a classe Gerente tivesse acesso diretamente a Intercamada Negócio/Persistência. Esta decisão foi realizada uma vez que acreditamos que não haverá alterações quanto à Base de Dados. De acordo com o cliente do sistema, o Banco de Dados é Relacional, o qual já está definido e não pode mudar. De qualquer forma, caso ocorra alguma alteração na Camada de Persistência, o Gerente da Unidade de Negócio terá de ser revisto; se uma classe IP estivesse sendo utilizada, o Gerente permaneceria inalterado.

A Figura B. 8 ilustra a Camada de Negócio, representada pelo pacote *negócio*, e as Unidades de Negócio (UNs). Nesta figura, temos destacado o pacote (UN) *negócio.equipamentoModular*, e dentro dele, a classe *G_EquipamentoModular*. Para esta classe, podemos verificar as seguintes características a partir dos “*imports*” realizados:

- *import Persistência.ModeloDeNegócio.EquipamentoModular;*
- *import Persistência.ModeloDeNegócio.ModeloEquipamentoModular;*
- *import Persistência.ModeloDeNegócio.Módulo;*

Para realizar as funcionalidades desta UN, foi necessário utilizar as Classes de Acesso *EquipamentoModular*, *ModeloEquipamentoModular* e *Módulo*. Podemos verificar, no Diagrama de Classes apresentado na Figura B. 1, acima, que estas classes apresentavam relacionamento entre si.

- *import negócio.módulo.G_Módulo;*

Esta UN_EquipamentoModular precisa comunicar-se com outra UN (a UN_Módulo), o que acontece através do *G_Módulo*.

- *import Negócio_Persistência.EquipamentoModularBD;*

A classe Gerente *G_EquipamentoModular* realiza acesso a Intercamada Negócio/Persistência, e mais especificamente, delega funcionalidades relativas à persistência para o *Broker EquipamentoModularBD*.

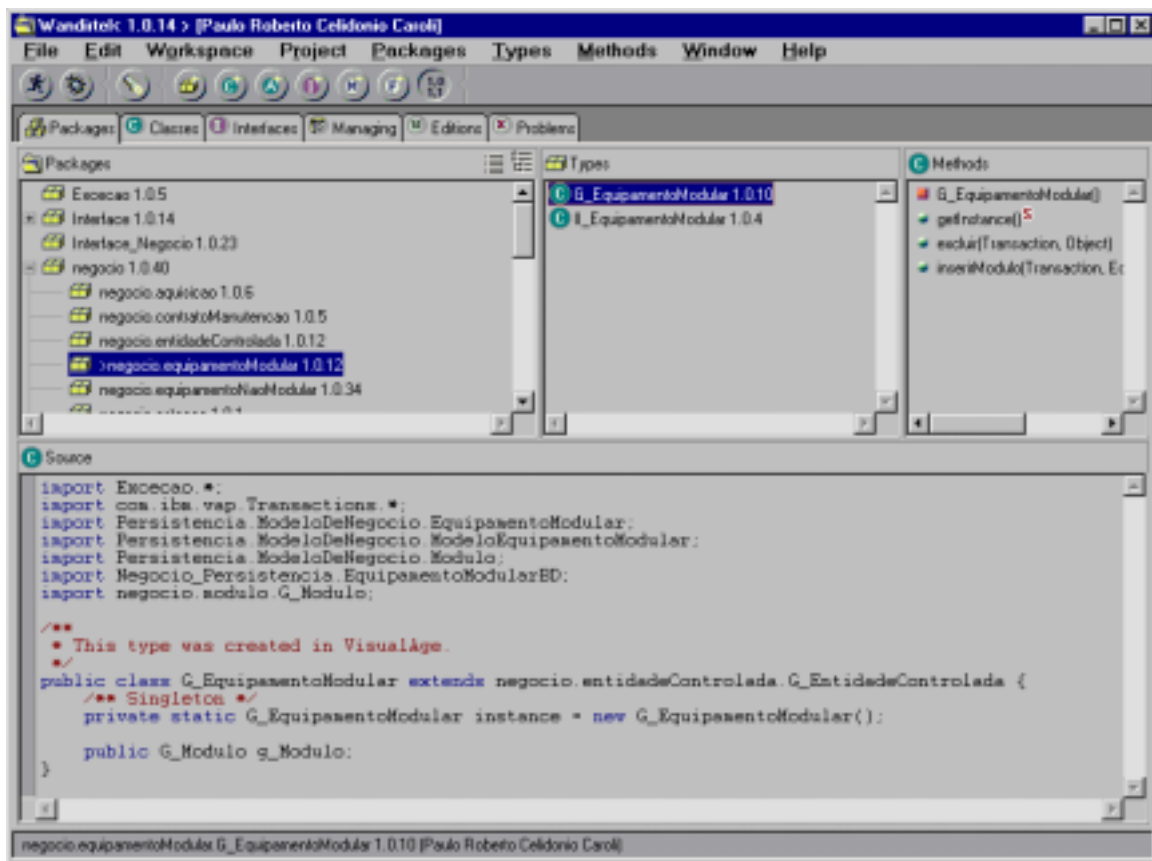


Figura B. 8 pacote negocio e seus subpacotes (UNs)

As figuras (Figura B. 9 e Figura B. 10) mostram a hierarquia de Herança existente dentro desta UN. Nota-se que foi criada a classe abstrata *G*, que apresenta funcionalidades necessárias e comuns aos Gerentes. Também pode ser verificada que, analogamente ao relacionamento de Herança no Diagrama de Classes apresentado na Figura B. 2, a classe *G_EquipamentoModular* herda as funcionalidades da classe *G_EntidadeControlada*.

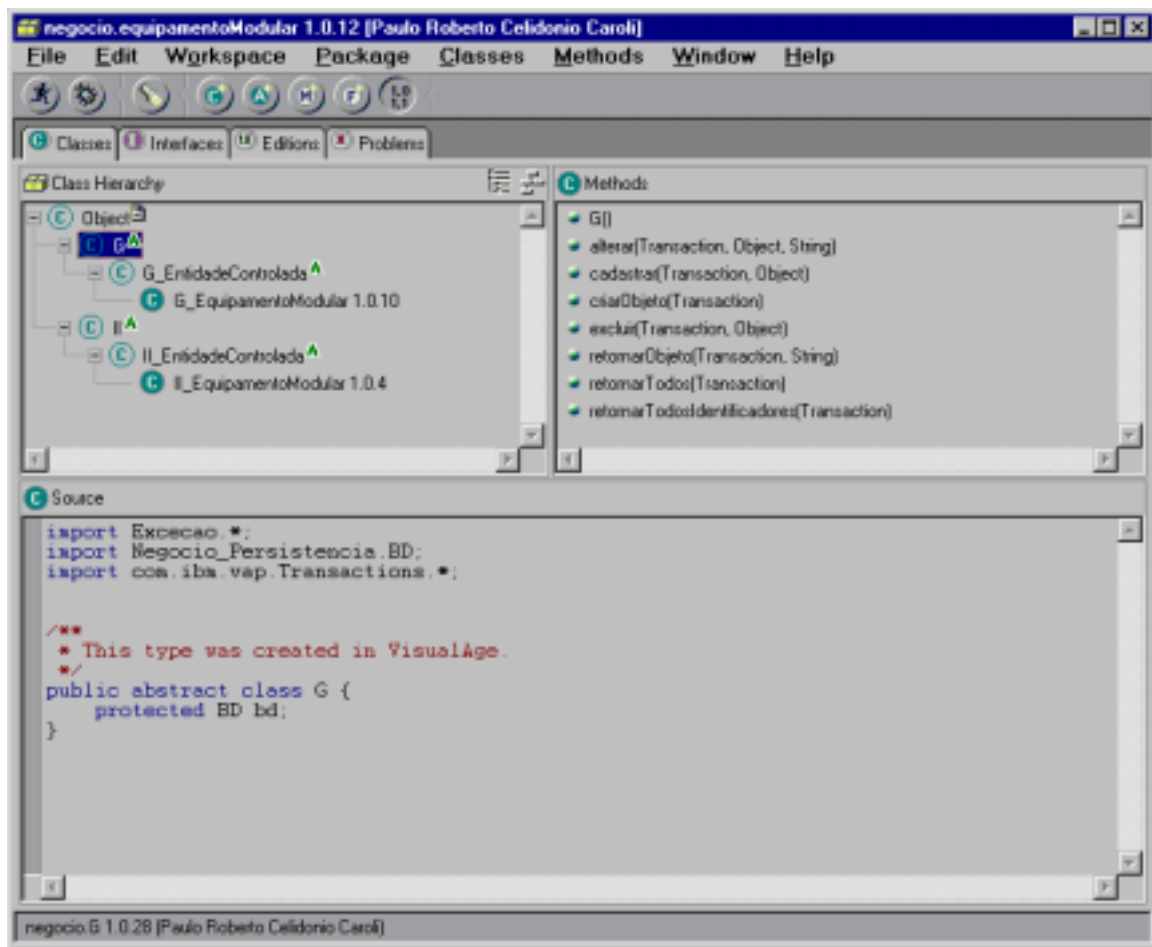


Figura B. 9 Hierarquia de Herança da UN_EquipamentoModular (1-2)

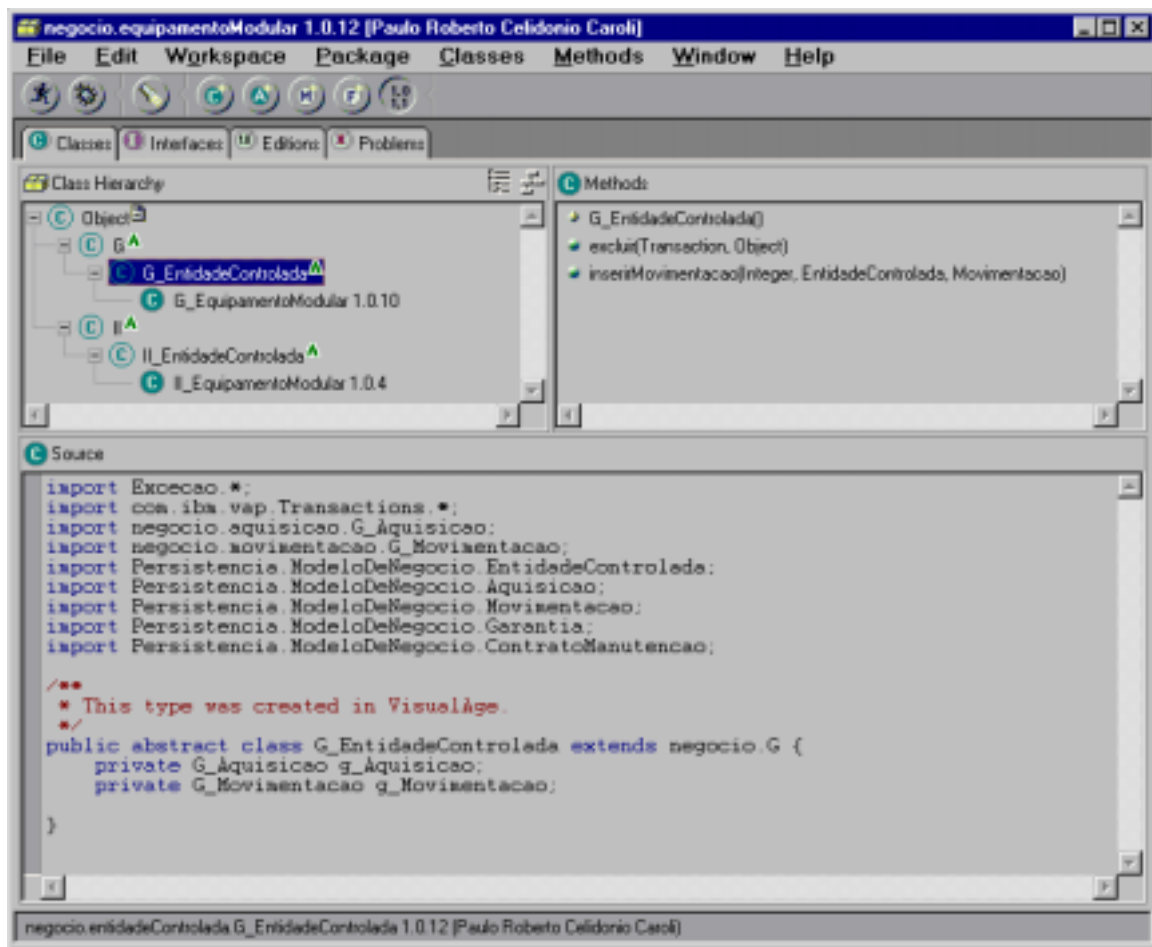


Figura B. 10 Hierarquia de Herança da UN_EquipamentoModular (2-2)

Framework Persistence Builder

Por decisão de projeto, utilizamos o *Framework Persistence Buildere* seus serviços para tratar de alguns aspectos relevantes. A utilização deste framework forneceu-nos :

- um mapeamento entre objetos e tabelas da Base de Dados Relacional,
- uma memória *cash* na qual os objetos trazidos da Base de Dados são manipulados,
- um controle dos mecanismos de transação
- controle dos aspectos de concorrência em relação aos objetos que manipulavam dados

TRANSAÇÃO

O tratamento de transação para estes sistemas não ficou encapsulado em uma única camada. As transações são “abertas” nos *Brokers* da Intercamada Interface/Negócio e são “commitadas” na Intercamada Negócio/Persistência.

Transações aninhadas podem ser abertas na Camada de Negócio, sendo sempre “comitadas” na Intercamada Negócio/Persistência.

Exceções são levantadas nas diferentes camadas. Sempre que ocorre uma Exceção, esta recebe um tratamento local, e se necessário for, é transmitida para a camada superior. Sendo necessário realizar um *rollback*, este realizava-se na Intercamada Interface/Negócio.

O *Persistence Builder* realiza a conversão dos objetos para a Base de Dados. Esta conversão é realizada automaticamente quando uma transação recebe “*commit*”. Cada classe do Diagrama de Classes apresenta uma classe *classeBD*, que é responsável pela realização das funcionalidades necessárias para a persistência (pedidos provenientes das classes “IP”). Estas *classesBD* realizam os “*commit*” e “*remove*” relativos aos dados persistentes.

B.2.4 Intercamada Negócio/Persistência

Esta Intercamada, composta por *Brokers*, responsáveis pela realização da comunicação entre as camadas de Negócio e de Persistência, realiza os comandos para armazenar e recuperar dados da Base de Dados; tais comandos são fornecidos pelo PERSISTENCE BUILDER.

Cada classe do Diagrama de Classes gerou uma classe nesta Intercamada. Os nomes dados a estas classes eram *classeBD*..

A Figura B. 11, abaixo, mostra o pacote *negócio_Persistência* e suas classes.

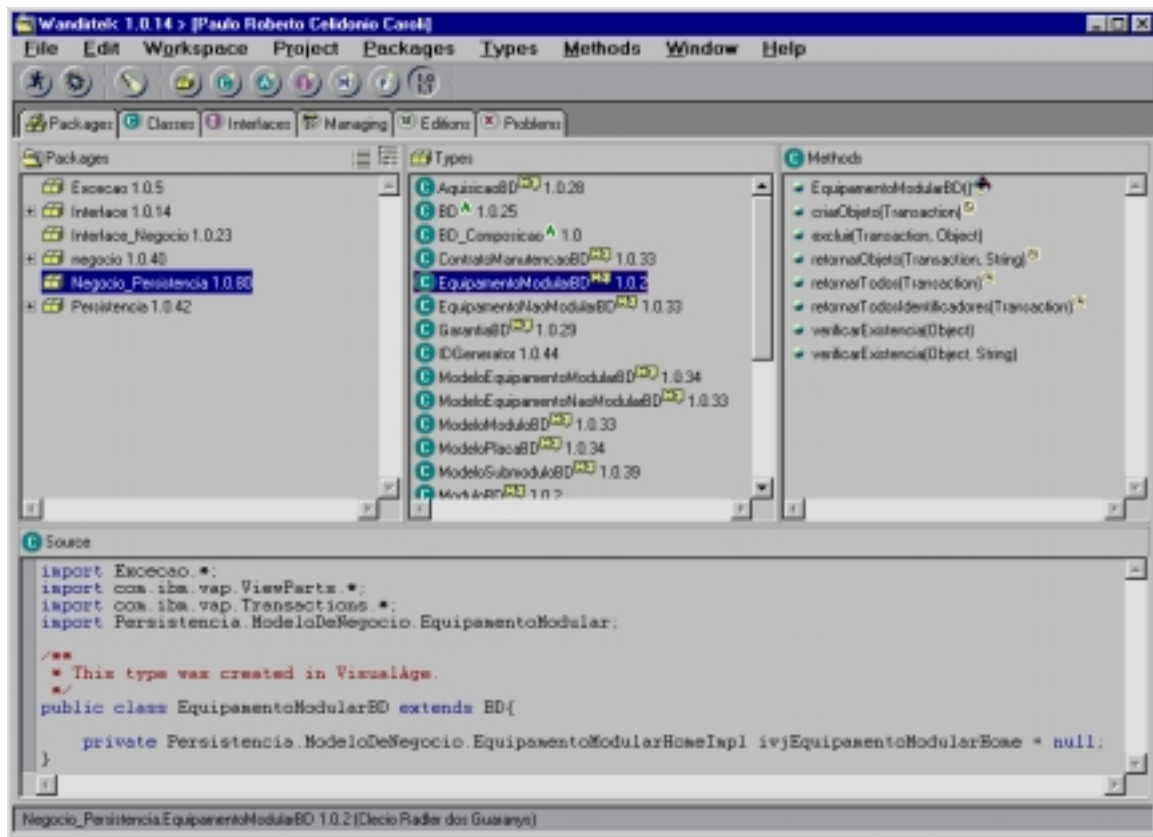


Figura B. 11 pacote *negócio_Persistência*

B.2.5 Camada de Persistência

Para persistência, estamos utilizando a Base de Dados *Oracle* [ORACLE]. O *Persistence Builder* realiza a conversão de objetos para tabelas na Base de Dados. Ele apresenta uma ferramenta de geração da Base de Dados; para utilizá-la, é necessária a entrada de informações sobre as classes que instanciam estes objetos. Estas são as classes dos Diagramas de Classes apresentados nas figuras (Figura B. 1, Figura B. 2, Figura B. 3 e Figura B. 4), mas somente com seus atributos e relacionamentos. O *Persistence Builder* gera as tabelas de forma assistida, permitindo que decisões quanto à organização das tabelas sejam realizadas.

B.2.6 Módulo de Acesso

O *Persistence Builder* gera as classes que estão mapeadas na Base de Dados. Estas classes são utilizadas através de uma *interface* (*interface Java*), também gerada pelo PERSISTENCE BUILDER, que apresenta exatamente os métodos de *get* e *set* para os atributos das classes do Diagrama de Classes. Estas interfaces correspondem às Classes de Acesso.

Em tempo de Execução, objetos destas *interfaces* (Classes de Acesso) ficam disponíveis em uma memória *cash*. Esta memória representa o conjunto de objetos que são manipulados pelas camadas da Aplicação. O *Persistence Builder* possibilita um tratamento em relação ao acesso a estes objetos; políticas de concorrência são disponibilizadas para controlar os acessos a estes objetos. Este controle é realizado através de um mecanismo que controla a versão do objeto que se encontra na memória *cash*.

A Figura B. 12, abaixo, ilustra o pacote que corresponde ao Módulo de Acesso. Podemos, agora, verificar que este é o pacote a partir do qual os “*imports*” das Classes de Acesso (rever Figura B. 8) eram realizados. Nesta figura, ilustramos a Classe de Acesso *EquipamentoModular* e seus métodos de *set*, *get*, *add* e *remove*, que controlam o acesso a seus atributos e relacionamentos.

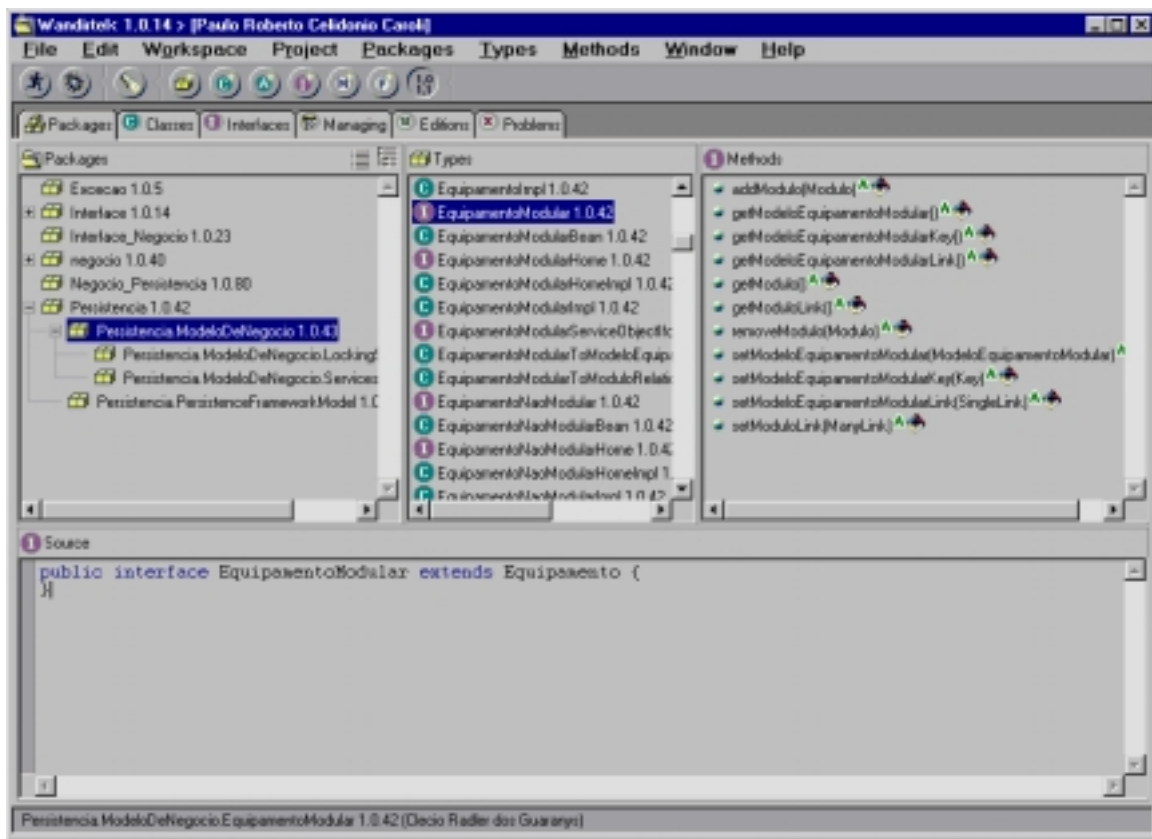


Figura B. 12 "Módulo de Acesso"

B.3 “Dumps” de Tela

A seguir, são apresentados alguns “dumps” de tela deste sistema (Figura B. 13, Figura B. 14 e Figura B. 15).

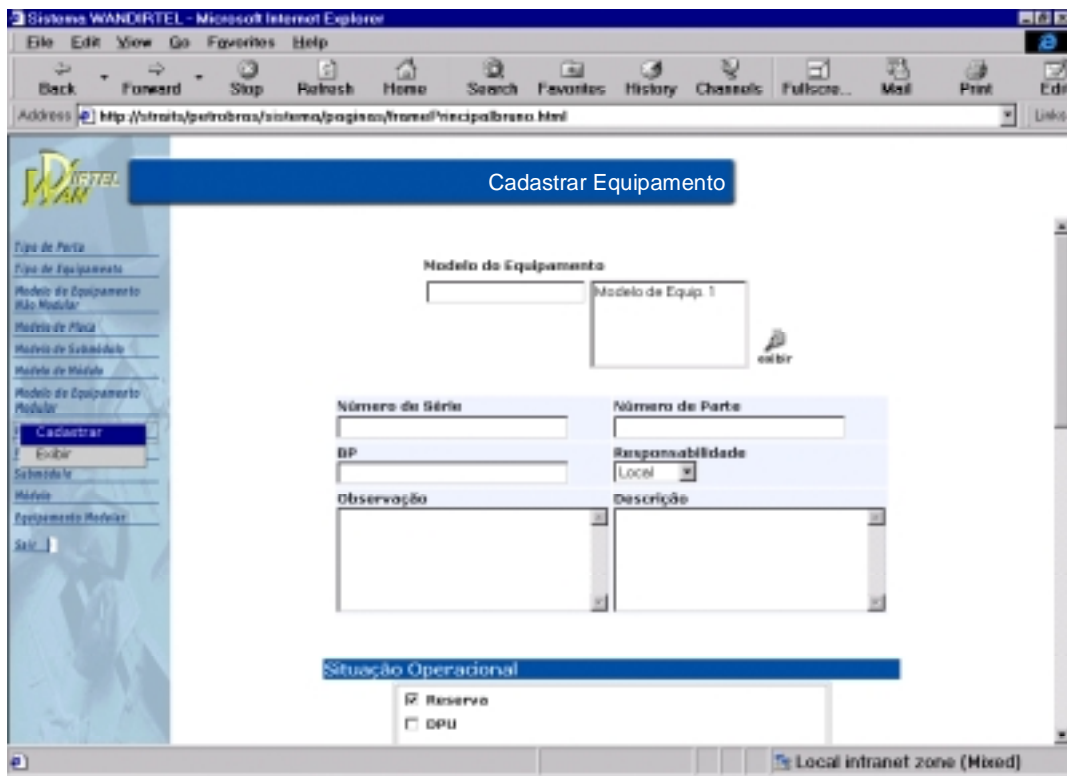


Figura B. 13 Gerencia de Materiasi - Cadastrar Equipamento

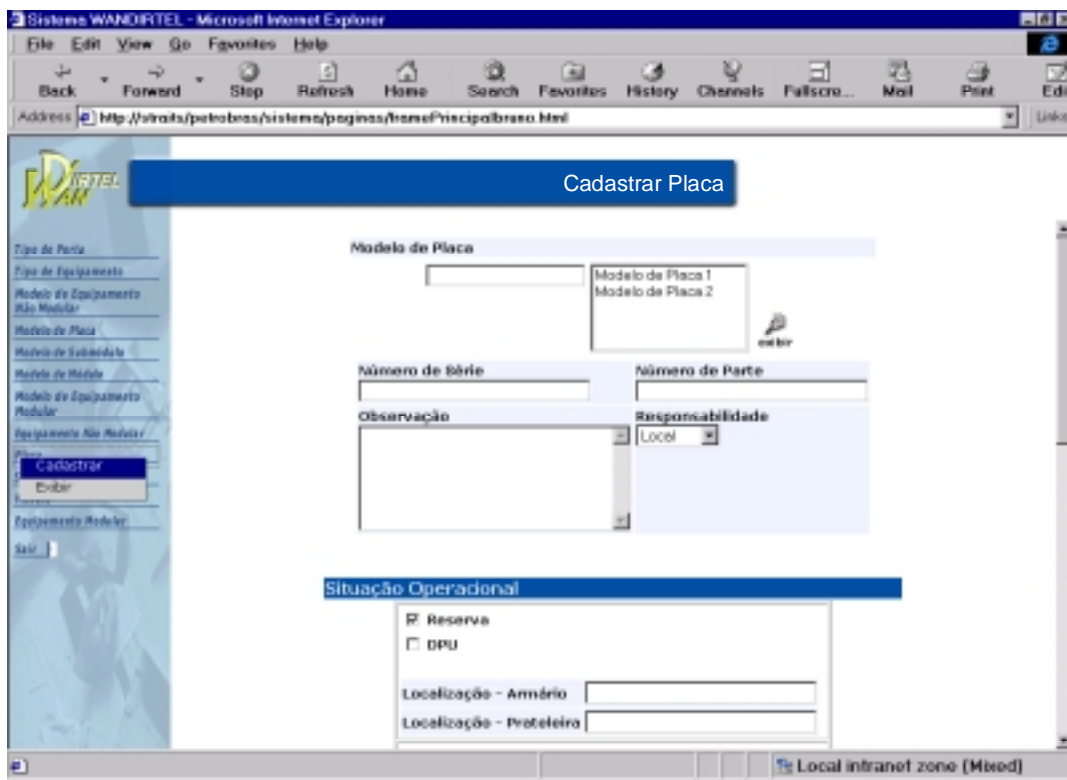


Figura B. 14 Gerencia de Materiasi - Cadastrar Placa

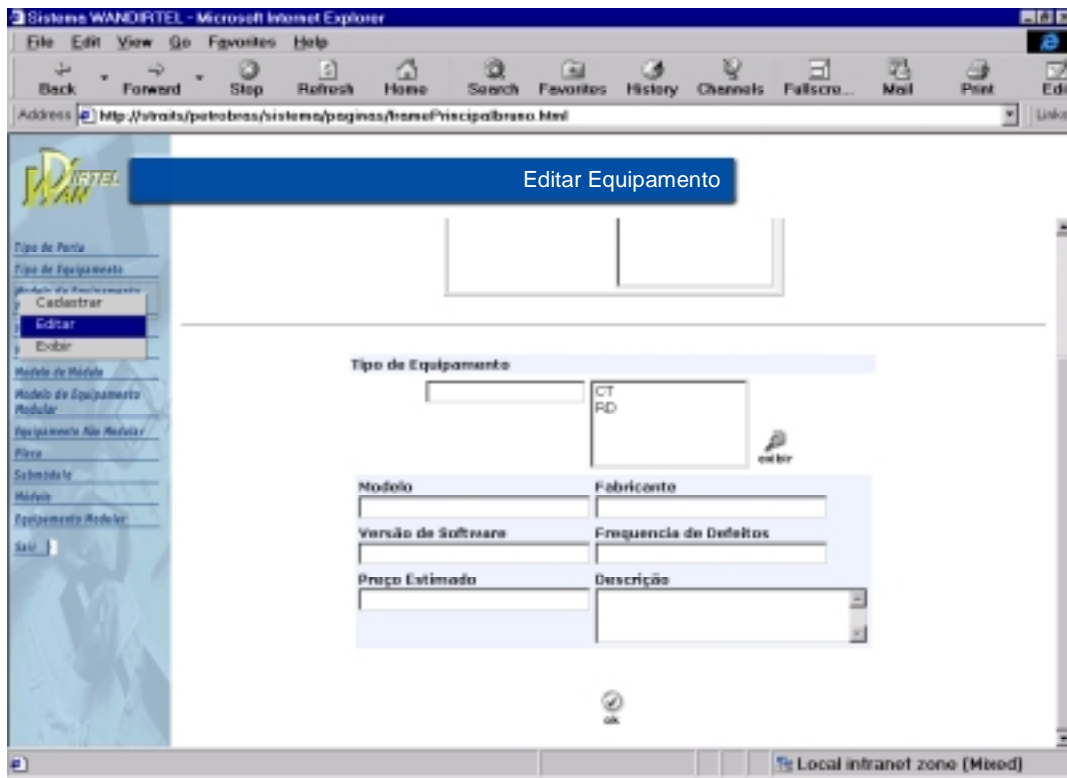


Figura B. 15 Gerencia de Materiasi – Editar Equipment

Referências

- [Booch94] Booch. G.; *Object-Oriented Analysis and Design with Applications*; Redwood City, CA; Benjamin/Cummings, 1994
- [Brown98] Brown, W.; Malvean, R.; McCornick, H.; Mowbray, T.; *Anti-Patterns : Refactoring Software, Architectures and Projects in Crisis*; John Wiley & Sons, 1998
- [Buschman96] F. Buschman, R. Meunier, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, A system of Patterns*, John Wiley & Sons, 1996
- [Cormen90] Thomas H. Cormen, Charles E. and Ronald L. Rivest, *“Introduction to Algorithms”*, Mc Graw Hill, 1990
- [Edwards97] Edwards, Jerry and DeVoe, Deborah, *3-Tier Client/Server At Work*, Wiley Computer Publishing; 1997
- [Gamma95] E. Gamma, R. Helm, R. Johnson, and j. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented software*, Addison-Wesley, 1995
- [Henderson-Sellers96] Henderson-Sellers, B.; *Object-Oriented Metrics : Measure of Complexity*; Englewood Cliffs, NJ, Prentice-Hall, 1996
- [IEEE90] Institute of Electrical and Electronic Engineers; *Standard Glossary of Software Engineering Terminology*; IEEE Std 610.12, 1990
- [ISO92] International Organization for Standardization / International Electrotechnical Committee. *“Information technology - Open Systems Interconnection - Basic Reference Model - Part 1: Basic Reference Model*, (revision of First Edition - ISO 7498:1984)”. Draft International, Standard ISO/IEC DIS 7498-1, 1992

- [Keller98] W. Keller, *Object/Relational Access Layer – A roadmap, Missing Links and More Patterns*, Europlop98, Bad Irsee, Germany, 1998
- [Mattsson96] M. Mattsson, "Object-Oriented Frameworks: "A Survey of Methodological Issues", M.Sc. Dissertation, Department of Computer Science and Business Administration, University College of Karlskrona/Ronneby, LU-CS-96-197, 1996.
- [Myers76] Myers, G.; *Software Reliability – Principles and Practices* ; New York; John Wiley & Sons, 1976
- [Petroski85] Petroski, H, 1985, *To Engeneer is Humam* , New York; St. Martin's Press.
- [Riel96] Riel, Atrthur J. , *Object-Oriented Design Heuristics*, Addison Wesley; Reading, Massachusetts; 1996
- [Rumbaugh91] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W.; *Object-Oriented Modeling and Design* ; Prentice Hall; 1991
- [Scott97] Scott A., J.; *Object-Oriented Design Measurement* ; John Wiley & Sons; 1997
- [Sharble93] Sharble, R. C. , and S. S. Cohen, 1993, The Object-Oriented Brewery; *A Comparison of Two Object-Oriented Development Methods*, ACM SIGSOFT Software Engineering Notes 18/ no. 2(April) 60-73.
- [Shaw96] M. Shaw, D. Garlan, *Software Architecture, Perspectives on an Emerging Discipline*, 1996
- [UML99] Booch, G.; Rumbaugh, J.; Jacobson, I.; *The Unified Modeling Language User Guide* ; Addison Wesley; Reading, Massachusetts; 1999

- [Uchôa99-1] Uchôa, E.M.A., "*Framework para Integração de Sistemas de Bancos de Dados Heterogêneos*", Tese de Doutorado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, 1999.
- [Uchôa99-2] Uchôa, E.M.A. & Melo, R.N., “HEROS : A Framework for Heterogeneous Database Systems Integration”, in Proceedings of the Tenth International Conference on Database and Expert Systems Applications (DEXA’99), Lecture Notes in Computer Science, n. 1677, pp. 656-667, Springer-Verlag, Florenca, Italia, Ago. 1999.
- [Vianna97] M. J. Vianna; S. Carvalho; J. Kapson; "*Patterns for Layered Object-Oriented Applications*"; EuroPLOP '97
- [Wirfs-Brock90] Wirfs-Brock, R.; Wilkerson, B.; Wiener, L.; *Designing Object-Oriented Software*; Englewood Cliffs, NJ, Prentice-Hall, 1990

Referências na Web

[JDBC] <http://java.sun.com/products/jdbc/>

[ORACLE] <http://www.oracle.com/>

[PERSISTENCE BUILDER]

<http://www7.software.ibm.com/vad.nsf>
(search *Persistence Builder* in VADD)

[ROSE] <http://www.rational.com/products/rose>

[SERVLET BUILDER]

<http://www7.software.ibm.com/vad.nsf>
(search *Servlet Builder* in VADD)

[VA-JAVA] <http://www-4.ibm.com/software/ad/vajava/>