INSTITUTO DE PESQUISAS TECNOLÓGICAS DO ESTADO DE SÃO PAULO

ANGELO ISAIAS BAGGIO

MÉTODO DE APLICAÇÃO DE PADRÕES (PATTERNS) NO PROCESSO UNIFICADO

São Paulo

2004

ANGELO ISAIAS BAGGIO

MÉTODO DE APLICAÇÃO De PADRÕES (PATTERNS) NO PROCESSO UNIFICADO

ANGELO ISAIAS BAGGIO

MÉTODO DE APLICAÇÃO DE PADRÕES (PATTERNS) NO PROCESSO UNIFICADO

Dissertação apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, para obtenção do título de Mestre em Engenharia de Computação.

Área de concentração: Engenharia de Software

Orientador: Dr. Jorge Luiz Risco Becerra

SÃO PAULO
2004

Baggio, Angelo Isaias

Método de aplicação de padrões (Patterns) no processo unificado. / Angelo Isaias Baggio . São Paulo, 2004.
83p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Engenharia de Software.

Orientador: Prof. Dr. Jorge Luiz Risco Becerra

1. Engenharia de software 2. Processo unificado 3. Padrões de referência
4. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Centro de Aperfeiçoamento Tecnológico II. Título

CDU 004.414(043)
B144m

AGRADECIMENTOS

Agradeço, acima de tudo, a Deus por minhas inspirações.

Ao Prof. Dr. Jorge Luiz Risco Becerra pela compreensão, apoio, incentivo e motivação que tornaram possível a elaboração deste trabalho.

À minha esposa Solange, minhas filhas Tatiane e Milena pela compreensão nos momentos em que estive ausente para me dedicar à pesquisa e à dissertação, bem como pelo permanente incentivo.

Aos meus pais Guido e Maria e aos amigos e a todos os colegas de trabalho da CRK, Banco Panamericano e BOVESPA que participaram desta fase importante de minha vida, apoiando e tendo paciência em me ouvir.

Meus agradecimentos a todas as pessoas que me apoiaram de uma forma ou de outra. Sem dúvida nenhuma, contribuíram muito para que eu alcançasse este objetivo.

RESUMO

Esta dissertação de mestrado visa apresentar um método de aplicação de padrões no contexto metodológico do Processo Unificado.

Para atingir este objetivo, o método é composto de procedimentos que guiam arquitetos, projetistas, analistas e gerentes na utilização de padrões (patterns) em projetos de *software*.

Esta dissertação por meio do método, apresenta como relacionar padrões à tecnologia Microsoft .NET, apesar do método ter a característica de independência de tecnologia de software.

O método enfatiza fortemente a utilização de padrões conhecidos, denominados de padrões de referência, são os padrões dos autores [Gamma, 1995], [Bushmann, 1996], [Larman, 1999] e [PnP, 2002]. O método deixa clara a distinção entre os padrões de Arquitetura e padrões de Projeto, para isso há procedimentos específicos para cada um.

No experimento do método, o estudo de caso tem como base o sistema de Custódia de Títulos da BOVESPA, onde o objetivo é concretizar os conceitos do método em um projeto real, visando assim facilitar a sua compreensão.

Palavras-chave: Padrões; Design Patterns; Processo Unificado; Arquitetura.

ABSTRACT

This master degree dissertation aims to present a method for applying patterns in the methodological context of the Unified Process.

In order to reach this goal, this method comprises some procedures, which guide architects, designers, analysts and managers in the use of patterns in software projects.

Although this method is independent of the software technology used, this dissertation shows how to relate patterns to Microsoft .Net Technology using the method proposed.

During the text, the recommendation of using recognized patterns, usually called reference patterns, as [Gamma, 1995], [Bushmann, 1996], [Larman, 1999] and [PnP, 2002] are strongly stressed. This method makes clear the distinction between Architecture Patterns and Design Patterns and it illustrates the different procedures for each one of them.

In the method experiment, the case study refers to the Assets Custody System of BOVESPA, with the objective of employ the method in an actual project, in order to ease its comprehension.

Key-words: Patterns; Design Patterns; Unified Process; Architecture.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BOVESPA	Bolsa de Valores do Estado de São Paulo
CBLC	Companhia Brasileira de liquidação e Custódia
GOF	Gangue Of Four
GRASP	General Responsability Assignment Software Patterns
MVC	Model View Controller
OO	Orientação a Objetos
POSA	Pattern Oriented Software Architecture
RUP	Rational Unified Process
TI	Tecnologia da Informação
UML	Unified Modeling Language
UP	Unified Process

SUMÁRIO

1 INTRODUÇÃO

Este capítulo tem como objetivo apresentar de forma sintética este trabalho. Permitirá que os assuntos abordados em todo o trabalho sejam introduzidos de forma a facilitar o entendimento do leitor

Os próximos itens descrevem as considerações iniciais, os objetivos, as justificativas e motivações desta dissertação de mestrado intitulada "Método de aplicação de padrões (patterns) no Processo Unificado"

1.1 Considerações Iniciais

O assunto principal deste trabalho foca em apresentar um método para a utilização de padrões (pattern) no desenvolvimento de software, inserindo-o no contexto do Processo Unificado. Pretende-se com este método, guiar os participantes de um projeto na elaboração de projetos de classes e na elaboração de arquiteturas de sistemas.

Os padrões aqui mencionados se apresentam de duas formas: uma como sendo padrões de projetos e outra como padrões de arquitetura.

Os padrões de projetos são as experiências de outros projetistas na elaboração do modelo de classes que podem ser reutilizados. Estas experiências começaram a ser registrada em 1995 quando, Erich Gamma *et.al.* [Gamma, 1995] lançaram um livro em que reuniram e organizaram padrões de projetos (Design Pattern), tendo como ponto de partida as experiências dos autores em projetos orientados a objetos. Desde então, muitos outros autores reuniram suas experiências em Projetos Orientados a Objetos e lançaram várias obras a respeito de padrões.

Com relação a padrões de arquitetura, a sua abordagem se fundamenta em Buschmann [Buschman, 1996] que especializa ainda mais os padrões e subdividi-os em camada de arquitetura, projeto e implementação. Larman [Larman, 1999] é outro importante autor que apresenta uma proposta de padrão de arquitetura de sistema em camadas.

Todos os autores sobre padrões mencionados anteriormente e mesmo outros que não são mencionados nesta dissertação, possuem um papel importante no método. Seus padrões são considerados no método como sendo os padrões de referência, onde todos os padrões a serem aplicados devem ter sua origem nestas referências.

O método pretende guiar gerentes de projetos, arquitetos, analistas de sistemas e projetistas na arte de modelar um projeto ou uma arquitetura, para isso o método é constituído de etapas que auxiliarão cada um dos envolvidos no projeto.

O gerente de projeto terá a visão do processo, ou seja onde o método será aplicado no processo de desenvolvimento de software. Os outros participantes terão procedimentos operacionais que os conduzirão na elaboração de suas tarefas.

O método é inserido no contexto do processo de engenharia de software denominado Processo Unificado. Este processo é o guia geral das disciplinas, atividades e fases, necessárias em um processo.

O Processo Unificado é considerado um processo de engenharia de software. Possui três conceitos fundamentais que são: dirigido a caso de uso, centrado na arquitetura e, iterativo e incremental. Está dividido em quatro fases denominadas de Concepção, Elaboração, Construção e Transição e entre estas fases são varias as disciplinas que são utilizadas, entre elas está a Modelagem de negócio, Requisitos, Análise e projeto, Implementação e Testes [Jacobson, 1999] [Kruchten, 2000].

O método é composto de duas atividades: a aplicação de padrões de arquitetura e a aplicação de padrões de projeto. A proposta do método é a aplicação destas duas atividades na disciplina de Análise e projeto do Processo Unificado, onde devem ser distribuídas entre as fases de Concepção, Elaboração e Construção conforme o esforço necessário para cada uma das atividades do método.

E por fim para fundamentar o método, esta dissertação apresenta no experimento a aplicação do mesmo no Sistema de Custódia da BOVESPA.

1.2 Objetivo

O objetivo principal deste trabalho de dissertação é apresentar um método para aplicar padrões de projeto e de arquitetura de sistema no Processo Unificado. Este método visa auxiliar nas tarefas dos participantes de um projeto de desenvolvimento de software, principalmente no tocante a elaboração do modelo de classes de projetos e do modelo de arquitetura de sistema.

Para tanto, o método é constituído de disciplinas com suas respectivas atividades que foram elaboradas para ser um guia na aplicação de padrões.

Outro objetivo é validar o método em um experimento, onde o experimento apresenta um caso real cujo contexto é o Sistema de Custódia de Títulos da BOVESPA. Por ser o sistema de Custódia muito complexo, o escopo do experimento foi reduzido a apenas algumas funcionalidades, porém o suficiente para ser possível a sua aplicação.

Por último, o objetivo é disseminar no meio acadêmico e profissional os conhecimentos adquiridos nas pesquisas para a elaboração desta dissertação, utilizando os elementos aqui apresentados para impulsionar novas pesquisas e futuros trabalhos.

1.3 Motivação

A motivação deste trabalho de dissertação de mestrado é fundamentada em três aspectos: adquirir e aprofundar os conhecimentos nos conceitos que evolvem este trabalho; criar mecanismos de produtividade no desenvolvimento de software; colaborar na disseminação do conhecimento e de práticas de padrões de projeto e arquitetura.

A aquisição de conhecimento se deu por meio da pesquisa sobre padrões de projetos e arquitetura, Processo Unificado, RUP e .NET. Estas pesquisas levaram o autor a sedimentar seus conhecimentos quanto a estes assuntos.

Com os conhecimentos adquiridos, o passo seguinte foi elaborar um mecanismo que pudesse colaborar com a produtividade de um projeto de desenvolvimento de software. Isto se deu pela criação do método de aplicação de padrões no Processo Unificado, sendo possível utilizar neste o conhecimento adquirido nas pesquisas.

A justificativa com relação a padrões é dada pelo fato de os mesmos poderem ser reutilizados. A reutilização é um mecanismo que possibilita a aplicação de conhecimentos e técnicas que já foram utilizadas em outros projetos [Alencar, 1995a, 1995b, 1996] [Gamma, 1995] [Buschmann, 1996] [Larman, 1999].

Ainda com relação aos padrões, eles são tratados neste trabalho como sendo de projeto e arquitetura, onde os padrões de projetos são utilizados para solucionar problemas de classes e subsistemas e seus relacionamentos, já os padrões de arquitetura definem a estrutura de um sistema e de seus relacionamentos [Buschmann, 1996].

Um método de aplicação de padrões deve ser parte de um processo de desenvolvimento de software, onde o ambiente de tecnologia e de negócios devem estar totalmente integrados por meio de um processo de desenvolvimento de software [Mehl, s.d.].

Com relação ao Processo Unificado, a motivação e a justificativa se dão pelo fato deste processo apresentar mecanismos capazes de solucionar as necessidades de um processo de desenvolvimento de software, tanto no aspecto gerencial, operacional e de ambiente, como também prover a integração da área de tecnologia com a área de negócio [Jacobson, 1999] [Kruchten, 2000].

Por último, é uma tendência natural do mercado desenvolvedor de software, que utilizam linguagem de software Microsoft, migrarem para os novos produtos Microsoft .NET. Por serem as linguagens .NET orientadas a objetos, é importante que desenvolvedores se preocupem com a orientação a objeto, conseqüentemente, com a aplicação de padrões de projetos em seus sistemas.

1.4 metodologia

Para a elaboração da dissertação, a metodologia aplicada segue os seguintes passos:

a) Pesquisa: A pesquisa tomou a maior parte do trabalho, em que os vários assuntos que compõem a dissertação foram identificados e estudados. Durante o período de pesquisa, as informações foram catalogadas para serem utilizadas na fase de dissertação. As pesquisas continuaram durante a elaboração da dissertação apenas quando houve a necessidade de aprofundamento de determinados assuntos.

b) Análise: Trata-se do estudo das pesquisas para melhor aproveitá-las no trabalho final.

c) Elaboração do método: Esta fase consistiu na elaboração do método objeto desta dissertação. A elaboração do método deu-se pela comparação de processos e da junção dos diversos fatores levantados na fase de análise. O resultado foi um método que atendesse à necessidade específica do contexto desta dissertação.

d) Estudo de caso: Esta fase consistiu na aplicação prática do método em um sistema que está sendo reestruturado em paralelo com esta dissertação. O sistema tem grande importância no meio financeiro e é um dos sistemas mais importantes da BOVESPA. O sistema é uma espécie de estoque de papéis, onde as ações de quase todas as empresas estão custodiadas (armazenadas).

1.5 ESTRUTURA DO TRABALHO

Este trabalho está dividido em seis capítulos, que são distribuídos quanto ao seu conteúdo da seguinte forma:

- Primeiro capítulo – apresenta aos leitores uma visão geral do que esta dissertação propõe: seus objetivos, motivação, justificativa, e a metodologia utilizada para o desenvolvimento do trabalho.

- Segundo capítulo – apresenta a fundamentação teórica dos elementos pertencentes a esta dissertação de mestrado. Nele são apresentados os conceitos de padrões, Processo Unificado, Arquitetura de sistemas e classes .NET.
- Terceiro capítulo – apresenta a proposta de trabalho representada por um método que visa a utilização de padrões e a sua aplicação no Processo Unificado.
- Quarto capítulo – apresenta o experimento do trabalho, aplicando o método desta proposta em um caso real.
- Quinto capítulo – apresenta as conclusões da pesquisa, os trabalhos que poderão ser desenvolvidos a partir da solução proposta, comentários e contribuições acadêmicas.

2 OS PADRÕES, O PROCESSO UNIFICADO E SOLUÇÕES DE PROJETO E ARQUITETURA, NA VISÃO .NET

Neste capítulo são apresentadas as fundamentações teóricas. O objetivo é passar ao leitor as teorias e conceitos referentes aos elementos que envolvem esta dissertação.

Inicia-se pelos padrões, que são identificados e explicados. O objetivo é tornar claros os padrões e seu uso, abordando os conceitos e as estruturas destes padrões.

Na seqüência, são apresentados os conceitos do Processo Unificado, suas fases e atividades, e também são apresentadas as diferenças entre o RUP e o Processo Unificado visando melhor discernimento a respeito de ambos.

Outro assunto abordado são as soluções de arquitetura em camadas, onde são apresentados cada camada e exemplos da aplicação da tecnologia Microsoft .NET nestas camadas.

Por último, será apresentada a relação entre os padrões [Gamma, 1995] e as classes .NET, demonstrando claramente qual a classificação desta relação.

2.1 O QUE SÃO PADRÕES

Este item começa por dar uma visão do que os autores das obras mais conhecidas sobre padrões dizem a respeito deste assunto. Na seqüência é descrito qual a estrutura dos padrões e por último são apresentados os padrões de [Gamma, 1995] [Buschman, 1996] e [Larman, 1999].

2.1.1 Introdução

Antes da conceituação dos padrões, é importante saber quais são as citações dos autores mais importantes de padrões. Começando por Larman, em [Larman, 1999], há a seguinte citação: “Na tecnologia de objetos, um padrão é uma descrição nomeada de um problema e sua solução. Que pode ser utilizado em novos

contextos. Em termos ideais, ele fornece aconselhamento sobre como utilizá-lo em circunstâncias variáveis".

Larman também enfatiza que: "Padrões usualmente não contêm novas idéias. A justificativa para padrões não é descobrir e expressar novos princípios de engenharia de software. Na verdade, trata-se exatamente do oposto, padrões tentam codificar o conhecimento, os idiomas e os princípios existentes; quanto mais apurado e amplamente usado, melhor".

Em [PnP, 2003], obra realizada pelos funcionários da Microsoft, há a seguinte citação referente a padrões: "Os benefícios dos padrões se tornam claros se você imaginar a quantidade de trabalho a mais que poderia ser despendido na definição de soluções sem o uso de padrões".

Também em [Buschmann, 1996], há a citação referente a padrões, que diz: "O padrão de alguém é um bloco de construção primitivo de outro, diz um provérbio da tecnologia de objetos, mostrando o caráter vago de que pode ser chamado padrão".

Outro autor [Eide, 2002], apesar de não ser criador de padrões, na sua obra sobre padrões Gamma, cita que: "Padrões são avaliados durante a fase inicial de desenvolvimento do sistema, onde poderão ajudar os Arquitetos de software no esboço e no planejamento da estrutura estática e dinâmica do software antes da implementação".

Os conceitos de padrões começaram a se popularizar a partir de um livro lançado em 1995, denominado *Design Patterns* [Gamma 1995]. Este Livro define 23 padrões, classificados como de criação, de estrutura e de comportamto. O livro se tornou um marco sobre o assunto, pelo pioneirismo e pela qualidade – os padrões nele definidos efetivamente são fundamentais. Os seus quatro autores passaram a serem conhecidos como Gangue dos Quatro, ou GOF (de Gang of Four). Esse rótulo se estendeu para o livro, e também para os padrões, que hoje são amplamente conhecidos como padrões GOF.

Vários outros padrões surgiram na seqüência, dentre eles destaca-se Larman que criou os padrões GRASP (*General Responsability Assignment Software Patterns*). São padrões que definem princípios básicos relacionados à atribuição de responsabilidades e são apresentados em [Larman, 1999].

2.1.2 Estrutura dos Padrões

O padrão é expresso por meio de um nome, da descrição de um problema e da solução para esse problema. Geralmente, a definição do padrão também caracteriza o problema em termos de forças, que são fatores correlacionados ao problema, e que o contextualizam, e descreve os benefícios e contra-indicações do uso do padrão.

Freqüentemente, a descrição textual do padrão é completada por uma visão gráfica, que representa a estrutura e/ou o comportamento dos elementos que o compõem. Quando se trata de padrões de projeto, a representação da estrutura é elaborada em diagramas de classe, e a representação do comportamento em diagramas de interação (colaboração ou seqüência).

Usa-se, no contexto do trabalho, a terminologia UML [Fowler, 2000] para a representação gráfica, apesar de esta tecnologia não existir quando os primeiros trabalhos sobre padrões foram escritos. Entretanto, esses trabalhos utilizam linguagens gráficas antecessoras da UML, e semelhantes a esta, de modo que mesmo alguém que só conheça a notação UML não terá dificuldades para entender os diagramas dos padrões originais.

2.1.3 Classificação dos padrões

Desde que o livro GoF [Gamma, 1995] que foi o primeiro e é o atual mais popular dos livros de padrões de software, foi lançado o termo "*Design Pattern*", aqui mencionado como Padrão de projeto, é freqüentemente utilizado para se referir a qualquer padrão que diretamente endereça suas características para arquitetura, projeto e implementação de software.

Muitos escolheram por fazer uma importante distinção entre estes 3 níveis conceituais caracterizando-os por Padrão de Arquitetura, Padrão de Projeto e Linguagem ou implementação, estes últimos também chamados de padrões de código.

De acordo com Buschmann [Buschmann, 1996], a comunidade separa padrões em três camadas chamadas de: Arquitetura/ Framework, Projeto e implementação. A seguir, são descritas em mais detalhes.

Arquitetura – é o mais alto nível de abstração em padrões de software, pois define a estrutura de um sistema e seu relacionamento. Contudo, há uma limitação nesta classificação uma vez que alguns padrões são difíceis de serem categorizados devido a sua abrangência, como o MVC (***Model-View-Controller)***.

Projeto – Descreve como resolver problemas providenciando uma solução de classes ou subsistemas e define o relacionamento entre eles.

Implementação – é o nível mais baixo dos padrões. Descreve como o código pode ser implementado em uma linguagem particular de programação.

2.1.4 Padrões Buschmann

Buschmann em [Buschmann, 1996], como ficou conhecido o trabalho desenvolvido por um grupo em 1996, que tem como título POSA (*Pattern Oriented Software Architecture*), apresentou uma proposta de padrões de arquitetura. A seguir são descritos alguns destes padrões.

Layers: para o sistema que deve lidar com questões em níveis diferentes de abstração. Por exemplo: questões de controle de hardware, questões ligadas a serviços comuns e questões específicas do domínio. É indesejável escrever componentes verticais que tratam de questões em todos os níveis, pois uma mesma questão teria que ser tratada múltiplas vezes em componentes distintos, podendo introduzir inconsistências.

Estruture o sistema em grupos de componentes que formam camadas umas sobre as outras. Faça com que as camadas superiores utilizem serviços apenas das camadas abaixo delas (nunca acima). Tente não utilizar serviços além daqueles da camada imediatamente inferior (não pule camadas, a menos que as camadas intermediárias apenas adicionem componentes de passagem).

Broker: Pode ser utilizado para a estrutura de sistemas distribuídos, com componentes desacoplados que interagem por meio de invocação de serviços remotos. Um componente denominado *broker* (corretor, intermediário) é responsável por coordenar a comunicação, transmitindo requisições, resultados e mensagens de exceção.

MVC: O padrão propõe a divisão de uma aplicação interativa em três componentes. O modelo (*model*) contém a funcionalidade central e dados; as visões (*view*) exibem informações aos usuários; os controladores (*controllers*) tratam as ações dos usuários. Em conjunto, *views* e *controllers* abrangem a interface com o usuário. Um mecanismo de propagação de mudanças garante a consistência entre a interface com o usuário e o modelo.

2.1.5 Padrões Gamma

Este autor traz em sua obra [Gamma, 1995] soluções de padrões para projetos. São 23 no total, como descrito anteriormente. No entanto, apenas alguns, considerados mais importantes, serão comentados abaixo:

Adapter: Converte a interface de uma classe em outra interface esperada pelos clientes. O Adapter permite que certas classes trabalhem em conjunto, pois de outra forma seria impossível por causa de suas interfaces incompatíveis.

Façade: Fornece uma interface unificada para um conjunto de interfaces em um subsitema. O Façade define uma interface de nível mais alto que torna o subsistema mais fácil de usar.

Observer: Diferentes tipos de objetos assinantes (*subscribers*) estão interessados em mudanças de estado ou eventos de um objeto publicador (*publisher*),

e desejam reagir, cada um a seu modo, quando o publicador gera um evento. Mais ainda, o publicador deseja manter baixo acoplamento com os assinantes. Define-se, então, uma Interface denominada Assinante (em inglês, a interface é denominada *subscriber* ou *listener*). Objetos que desempenham o papel de assinantes (ou seja, que demandam informações produzidas por outros) implementam essa Interface. Cada publicador pode registrar dinamicamente assinantes interessados em um determinado evento, e notificá-los quando o evento ocorrer.

Command: Encapsula uma solicitação como um objeto, desta forma permitindo a parametrização de clientes com diferentes solicitações, enfileira ou registra (*log*) solicitações e suporta operações que podem ser desfeitas.

Composite: Compõe objetos em estrutura de árvore para representarem hierarquias parte-todo. Permitem aos clientes tratarem de maneira uniforme objetos individuais e composição de objeto.

Strategy: A função é definir uma família de algoritmos, encapsular cada um deles e fazê-los intercambiáveis. O Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam.

2.1.6 Padrões Larman

Larman em sua obra [Larman, 1999] apresentou padrões que denominou de padrões de responsabilidades. Abaixo são apresentados dois padrões GRASP.

Information Expert: Aplica-se este padrão quando é necessário e desejável definir um princípio geral para atribuição de responsabilidades a objetos. Então, atribui-se a responsabilidade ao *expert* da informação (a classe que tem a informação necessária para atender à responsabilidade).

Creator: O propósito básico do Creator é encontrar um criador que precisa estar conectado ao objeto criado em qualquer evento. O acoplamento não é aumentado porque o objeto criado provavelmente já seria visível pelo Creator, por conta das associações existentes. Porém, há outros padrões para definição de

responsabilidade de instanciação de objetos, quando o Creator não se mostra adequado.

2.2 Processo Unificado

Nos próximos subitens são conceituados o Processo Unificado e o RUP. Apesar do Processo Unificado ser parte do tema deste trabalho, menciona-se o RUP para que fique clara a diferença entre os dois.

2.2.1 Processo Unificado (UP) e *Rational Unified Process* (RUP)

O Processo Unificado foi desenvolvido no final dos anos 90 por Jacobson, Booch e Rumbaugh que se reuniram para desenvolver o *Unified Software Development Process* (UP) [Jacobson, 1999]. Posteriormente, o UP foi instanciado e especificado pela empresa *Rational,* criando o produto *Rational Unified Process* (RUP), descrito por Kruchten em [Kruchten, 2000].

Tanto o UP como o RUP são antes de qualquer coisa processos de desenvolvimento de *software*, e, como tal, seu objetivo é transformar os requisitos do usuário em sistema de *software*. Suas características fundamentais são o fato de serem baseados em componentes, ou seja, o *software* desenvolvido é formado por componentes de *software* que se comunicam por meio de interfaces bem definidas. O padrão adotado para representação dos modelos é a Unified Modeling Language (UML).

O Processo Unificado (UP) é considerado um processo de engenharia de software, possuindo uma abordagem disciplinada direcionada a tarefas atribuídas e as responsabilidades. O UP objetiva assegurar a produção de software de alta qualidade, que atenda às necessidades de usuários finais com prazos e falhas previsíveis.

O UP é caracterizado como um processo de desenvolvimento de software iterativo. A abordagem iterativa é requerida para permitir um entendimento incremental do problema por meio de sucessivos refinamentos, buscando incrementar uma solução eficaz sobre as múltiplas iterações.

A abordagem iterativa ajuda a combater riscos em progresso. Freqüentes *releases* executáveis são geradas, permitindo envolvimento do usuário final e conseqüentemente a geração de *feedback*. Estas *releases* favorecem à equipe de desenvolvimento focar na produção de resultados. Constante status é gerado para assegurar que o projeto está no prazo.

2.2.2 Conceitos fundamentais

O Processo Unificado (UP) foi concebido tomando como base três conceitos fundamentais: dirigido por casos de uso, centrado na arquitetura, iterativo e incremental, conforme descrito a seguir:

- **Dirigido por Caso de Uso:** "Um caso de uso é uma parte da funcionalidade do sistema que fornece ao usuário um resultado de valor" [Jacobson, 1999]. Os casos de uso são utilizados para a captura e definição dos requisitos funcionais do sistema de *software*, facilitando a comunicação e o entendimento entre os *stakeholders* (envolvidos no projeto). São também utilizados para projetar os casos de teste;

- **Centrado em arquitetura:** no UP, os casos de uso e a arquitetura vão sendo desenvolvidos em paralelo. O conceito de arquitetura engloba os aspectos mais relevantes, tanto estáticos como dinâmicos do sistema; e

- **Iterativo e Incremental:** o UP utiliza pequenos ciclos de projeto (mini projetos) que correspondem a uma iteração e que resultam em um incremento no *software*. Iterações referem-se a passos no *workflow* e incrementos na evolução do produto.

2.2.3 Fases do Processo Unificado

O desenvolvimento de *software* efetuado por meio do UP é incremental, sendo cada incremento desenvolvido utilizando-se quatro fases: Concepção, Elaboração, Construção e Transição, como pode ser visto na **Figura 2.1**. A isto se denomina ciclo de desenvolvimento. Após a fase de transição o produto pode voltar a

percorrer cada uma das fases, constituindo a fase de evolução, na qual se produz uma nova geração do produto [Kruchten, 2001].

Ainda na **Figura 2.1**, pode ser visto a estrutura estática onde se apresentam as disciplinas do processo: Modelagem de Negócio, Requisitos, Análise e Projeto, Implementação, Teste e Implantação. Na estrutura dinâmica estão as fases do processo; Concepção, Elaboração, Construção e Transição, abaixo destas fases estão a representação do esforço empregado em cada disciplina dentro destas fases do processo. Pode se ver em cada fase que as mesmas podem ser divididas em iterações que representam os incrementos.

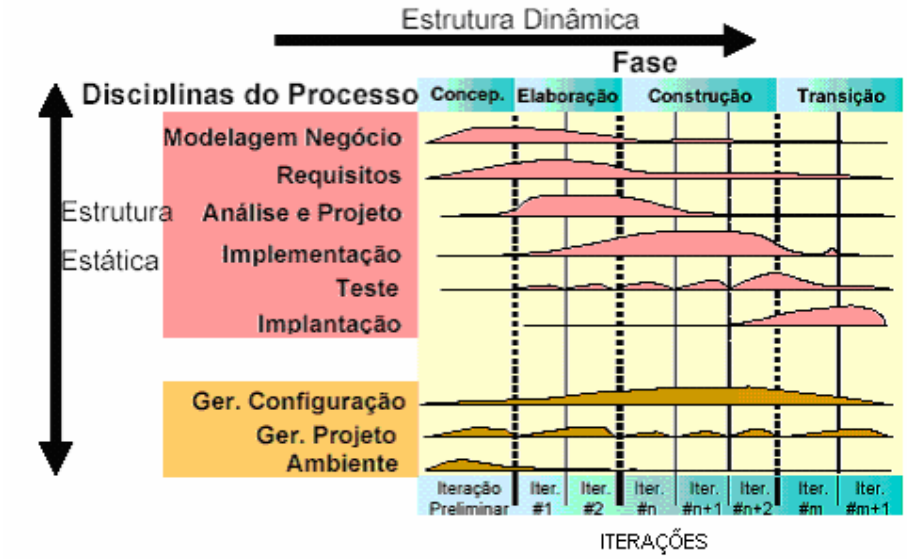

Figura 2.1 Estrutura estática e dinâmica do RUP

- **Concepção:** fase de compreensão do problema e da tecnologia por meio da definição dos casos de uso mais críticos. No final desta fase deve-se ter definido o escopo do produto, os riscos e ter-se demonstrado que o projeto é viável do ponto de vista do negócio da organização;

- **Elaboração:** fase de descrição da arquitetura do *software* na qual os requisitos que mais causam impacto na arquitetura são capturados em forma

de casos de usos. No final da fase de elaboração deve ser possível estimar custos, elaborar o cronograma e o plano de construção;

- **Construção:** fase na qual o *software* é construído e preparado para a transição para os usuários. Além do código, propriamente dito, também são produzidos os casos de teste e a documentação; e

- **Transição:** fase de treinamento dos usuários e transição do produto para utilização.

2.2.4 *Workflows* do Processo Unificado

Cada uma das quatro fases do UP é adicionalmente dividida em iterações e finalizada com um ponto de checagem que verifica se os objetivos daquela fase foram alcançados. Toda iteração é organizada em termos de *workflows* de processo, que são conjuntos de atividades realizadas por responsáveis que produzem artefatos. Os principais *workflows* de processo, também chamados de disciplinas, são descritos a seguir:

- **Modelagem de Negócio:** provê um entendimento comum entre os *stakeholders* sobre quais os processos de negócio devem ser apoiados. A modelagem dos processos de negócio é feita por meio dos casos de uso de negócio;

- **Requisitos:** objetiva capturar os requisitos que serão atendidos pelo produto de *software*. Nas fases de iniciação e elaboração, a ênfase será maior nesta disciplina de requisitos, pois o objetivo destas fases é o entendimento e a delimitação do escopo do produto de *software;*

- **Análise e Projeto:** objetiva compreender mais precisamente os casos de uso definidos na disciplina de requisitos, produzindo um modelo já voltado para a implementação que deverá estar adequado ao ambiente de implementação. Esta disciplina é bastante utilizada na fase de elaboração e durante o início da fase de construção;

- **Implementação:** objetiva a organização do código em termos de implementação de subsistemas, implementa as classes e objetos em termos de componentes, testa os componentes em termos de unidades e integra os códigos produzidos. Esta disciplina é bastante utilizada na fase de construção;

- **Teste:** objetiva analisar, por meio de testes, se os requisitos foram atendidos e que os defeitos serão removidos antes da implantação. Os modelos de testes são criados para descrever como os testes serão realizados. Sua ênfase é maior no final da fase de construção e no início da fase de transição;

- **Entrega:** objetiva produzir *releases* do produto e entregá-los aos usuários finais. Isto pode incluir atividades de beta-teste, migração de dados ou *software* existente e aceitação formal.

2.2.5 Diferenças entre o RUP e o Processo Unificado

No UP, descrito em [Jacobson, 1999] não havia a disciplina de modelagem de negócios, partindo-se diretamente para a disciplina de requisitos. Esta disciplina surgiu no RUP descrito em [Kruchten, 2001].

Também as disciplinas de análise e projeto apareciam separadamente, ressaltando a importância de se efetuar o modelo de análise. No RUP elas aparecem juntas.

Outras disciplinas que surgiram com o RUP são descritas no próximo item.

2.2.5.1 Disciplinas de Apoio do RUP

As disciplinas de apoio foram definidas como forma de suprir algumas das lacunas deixadas pelo UP [Jacobson, 1999]. Seu objetivo é auxiliar a execução das disciplinas principais. São elas:

- **Configuração e Gerenciamento de Mudanças:** controla as diversas versões dos artefatos produzidos durante o projeto, garantindo sua integridade, ou seja, assegurando que os resultados não sejam conflitantes;
- **Gerência de Projeto:** fornecem um *framework* para a gerência de projetos, orientações para planejamento, alocação de recursos e gerência de riscos; e
- **Ambiente:** fornece a descrição para a organização do ambiente de desenvolvimento em termos de processos e ferramentas.

2.3 Solução de Arquitetura em Camadas: UP e .NET

As disciplinas de Análise e projeto do UP apresentam, como um dos objetivos, desenvolver uma arquitetura robusta para o sistema, em que a finalidade é definir e validar uma Arquitetura de Software.

Para melhor entendimento sobre Arquitetura, serão descritos a seguir maiores detalhes sobre a Arquitetura de s*oftware* e os Mecanismos de arquitetura.

2.3.1 Arquitetura de *Software*

A arquitetura é um aspecto do projeto do *software* (ou uma parte do projeto) que define as questões mais amplas referentes ao projeto e diz respeito à estrutura: como o software vai ser organizado em subsistemas ou componente, que por sua vez serão decompostos em outros subsistemas ou componentes, e por meio de quais interfaces esses elementos irão se comunicar e prover determinados comportamentos.

A definição de uma arquitetura de software adequada é importante por vários fatores:

- Permite à equipe exercer um controle intelectual maior sobre o projeto, gerenciando adequadamente complexidade e garantindo maior integridade do produto;

- Define uma base para reuso do *software*, numa escala maior e potencialmente geradora de muito mais benefícios que apenas o reuso de classes ou componentes individuais. Também a própria arquitetura do *software*, ou parte dela, pode ser reutilizada em outros projetos;

- Oferece oportunidade adequada para definir como atender a requisitos de qualidade, como desempenho, disponibilidade, portabilidade e segurança;

- É uma base para a gerência de projetos, pois a estrutura do *software* será fator determinante para o planejamento das atividades, atribuições de responsabilidades em função de competências e também como organização para entrega (ou entregas parciais) do produto.

O UP representa a arquitetura de software por meio de um conjunto de visões (denominado Modelo de visão 4+1), conforme representado na **Figura 2.2**.

A visão lógica apresenta a estrutura conceitual do sistema. É uma abstração do modelo de projeto, identificando os principais pacotes de projeto, subsistemas e classes e, eventualmente, padrões (*patterns*) de arquitetura, por exemplo, a organização do sistema em camadas.

A visão de implementação descreve a organização estática dos módulos de *software* no ambiente de desenvolvimento, em termos de empacotamento, camadas, e gerência de configuração.

A visão de processos aborda os aspectos concorrentes do sistema em execução: tarefas, *threads* ou processos, e suas interações.

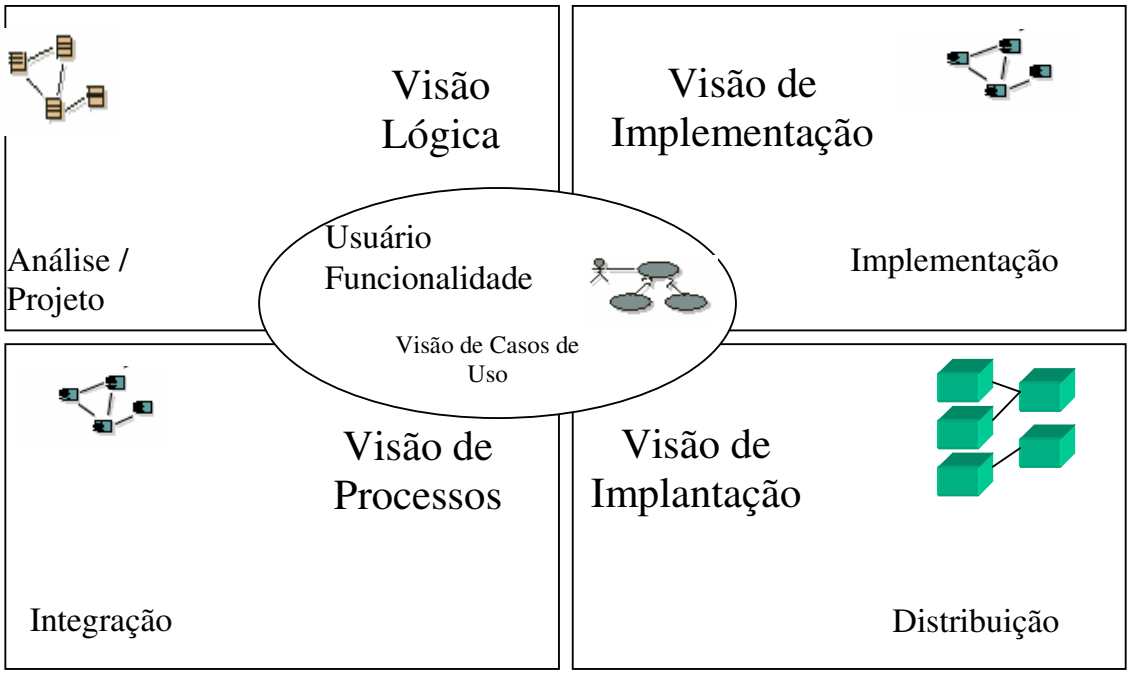

Figura 2.2 Visões da arquitetura de *software*

A visão de implantação mostra como os elementos de *software* executável estarão distribuídos nos nós computacionais.

A visão de casos de uso contém alguns cenários ou casos de uso chaves utilizados para definir e validar a arquitetura.

2.3.2 Mecanismos de Arquitetura

Mecanismo arquitetural é um termo genérico para se referir a mecanismos de análise, de projeto ou de implementação. Um mecanismo caracteriza uma solução concreta para problemas encontrados freqüentemente e pode definir um padrão de estrutura, comportamento, ou ambos. Um mecanismo pode ser visto como uma definição mais informal de um padrão, talvez para uso em um único projeto (mas não necessariamente). Um mecanismo de análise pode ser refinado por um mecanismo de projeto, e este por um mecanismo de implementação. O primeiro será uma definição mais conceitual, os demais acrescentam detalhes progressivamente mais concretos à solução.

A disciplina Análise e Projeto do Processo Unificado enfatiza a definição de mecanismos arquiteturais, de modo a tornar coerente e simplificar o projeto do projeto. Mecanismos podem compor a visão lógica da arquitetura de *software* e serem expressos por meio de colaborações (diagramas de classe e de interação).

2.3.3 Disciplinas da Análise e Projeto

O *workflow* definido pelo RUP para a disciplina Análise e Projeto é apresentado na **Figura 2.3**, através de um Diagrama de Atividades em que cada elemento é um Detalhe de *Workflow*. Cada detalhe de *workflow*, por sua vez, é descrito no RUP através de papéis, atividades e artefatos. Como o objetivo é fornecer uma idéia mais geral, apresenta-se uma descrição textual dos objetivos de cada detalhe de *workflow*.

O detalhe de *workflow* Efetuar Síntese da Arquitetura é opcionalmente realizado durante a fase de Concepção. Visa a definição e validação de uma prova de conceito da arquitetura, ou seja, um protótipo que mostre a viabilidade da concepção do sistema.

O detalhe de *workflow* Definir uma Arquitetura Candidata é executado no início da fase de Elaboração e tem, como objetivo, definir um esboço da arquitetura de software, a ser refinado em iterações posteriores. Contempla a definição inicial da organização do sistema em camadas e elementos principais (subsistemas ou componentes), bem como a seleção dos primeiros casos de uso ou cenários de casos de uso a serem realizados em termos de colaborações. Além disso, uma primeira definição dos mecanismos de análise para o projeto deve ser realizada.

O detalhe de *workflow* Analisar Comportamento transforma as descrições funcionais dos casos de uso em elementos que servirão como base para o projeto do sistema. A atividade chave é Analisar Caso de Uso e o produto desta atividade é constituído pelas colaborações (diagramas de classe e de interação, principalmente), ainda no nível de análise.

O detalhe de *workflow* Refinar a Arquitetura visa efetuar a transição da análise para o projeto, definindo os elementos de projeto, a partir de elementos de

análise (por exemplo, enriquecendo os diagramas de classe e interações com os elementos tipicamente de projeto). Para manter a consistência da arquitetura de software em elaboração, os elementos de projeto identificados devem ser adequadamente integrados com elementos existentes (normalmente definidos em iterações anteriores).

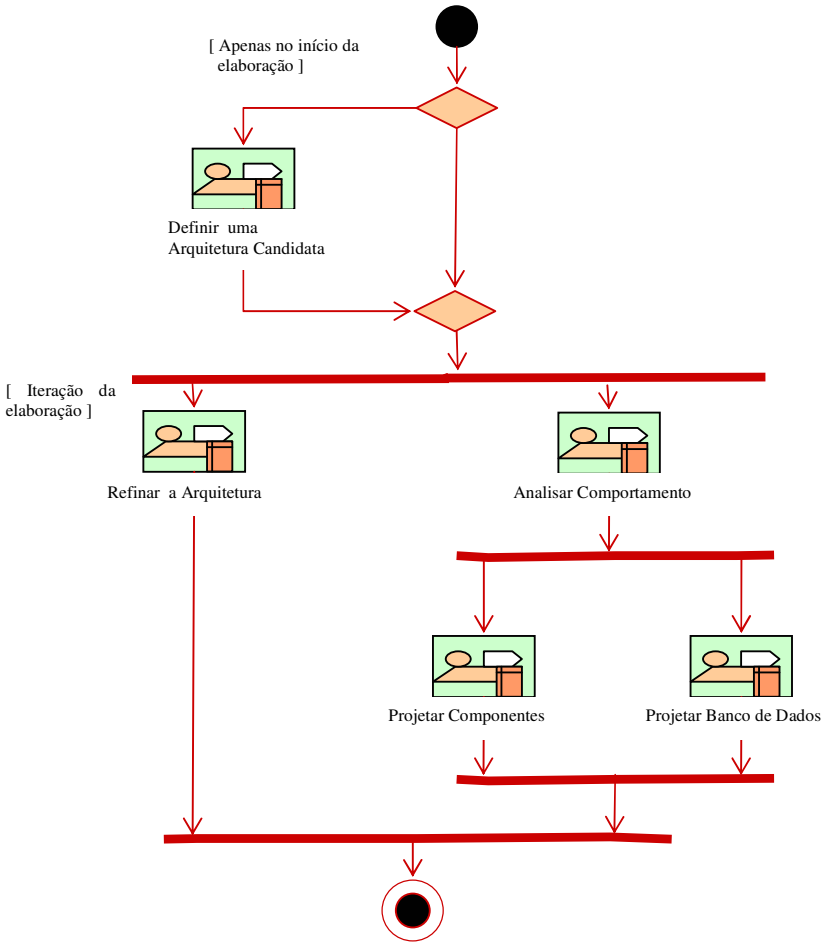

Figura 2.3 Workflow da disciplina de Analise e Projeto

O detalhe de *workflow* Projetar Componentes atua claramente no nível de projeto. Visa projetar os casos de uso e as classes individuais, bem como revisar os elementos de projeto definidos.

O detalhe de *workflow* Projetar Banco de Dados visa identificar as classes persistentes no modelo, projetar (ou identificar em um modelo) os elementos do banco de dados, que serão utilizados para armazenar as informações das classes persistentes, e definir mecanismos e estratégias para implementar a persistência.

2.3.4 Papéis na Análise e Projeto

O RUP estabelece dois papéis principais na disciplina de análise e projeto. Os papéis são os do Arquiteto e do projetista, a seguir descritos em maiores detalhes.

O Arquiteto é caracterizado por Kruchten como sendo aquele que lidera e coordena as atividades técnicas e artefatos ao longo do projeto [Kruchten, 2000]. Ele estabelece a estrutura geral de cada visão da arquitetura: a decomposição da visão, o agrupamento de elementos e as interfaces entre os agrupamentos maiores. Em contraste com a abordagem dos outros papéis, a do arquiteto é abrangente em vez de profunda.

O Projetista (*designer*) efetua o trabalho complementar ao do Arquiteto. Ele projeta cada caso de uso (define as colaborações correspondentes aos cenários e casos de uso) e, em um grau de detalhe maior, cada classe, em termos de responsabilidade, operações, atributos e relacionamentos com outras classes.

Outros papéis que compõem a disciplina são citados a seguir e têm responsabilidades mais específicas que os citados logo acima:

- Projetista de Banco de Dados;
- Projetista de Cápsulas (para projeto de sistemas de tempo real);
- Revisor da Arquitetura;
- Revisor do Projeto.

2.3.5 Padrão de arquitetura *Three-Layered Services Aplication* aplicado no .NET

Neste item, inicia-se por descrever o Contexto, o Problema e as Forças deste padrão. Na seqüência o padrão é apresentado e detalhado e para cada camada são sugeridas algumas tecnologias Microsoft que orienta o leitor no entendimento

destas camadas. Este padrão foi elaborado pela Microsoft e pode ser encontrado em [Pnp, 2002].

Contexto

Ao elaborar um desenho de uma aplicação em camadas, percebe-se que existe a necessidade de expor algumas das funcionalidades vitais da aplicação, como serviço que outras aplicações podem utilizar e também utilizar serviços expostos por outras aplicações.

Problema

Como elaborar as camadas para aplicações orientadas a serviços e como determinar quais são os componentes em cada camada.

Forças

Algumas forças determinam a utilização deste padrão:

- Minimizar impactos ao adicionar serviços a uma aplicação;
- Serviços são freqüentemente expostos para clientes fora do *firewall,* e há diferenças na segurança e nos requisitos operacionais que fazem o componente de negócio;
- Comunicar com outros serviços envolve importantes conhecimentos sobre protocolos e formatos de dados;
- Quando houver a necessidade de separar os negócios dos componentes, e o intuito é isolar a lógica de negócio da tecnologia adquirida para acessar serviços externos.

Solução

Elaborar uma arquitetura em três camadas: apresentação, negócio e dados. Este padrão apresenta uma visão geral das responsabilidades de cada camada e os componentes que compõem cada uma.

Na seqüência serão apresentados maiores detalhes sobre o padrão de arquitetura *Three-Layered Services Aplication.* A **Figura 2.4** dá uma visão sobre a distribuição das camadas, e na seqüência são detalhados cada componente. No detalhamento, constam algumas tecnologias Microsoft .NET que podem ser usadas em cada camada. A constatação destes produtos visa facilitar o entendimento da relação entre os componentes do padrão e a aplicação prática do mesmo.

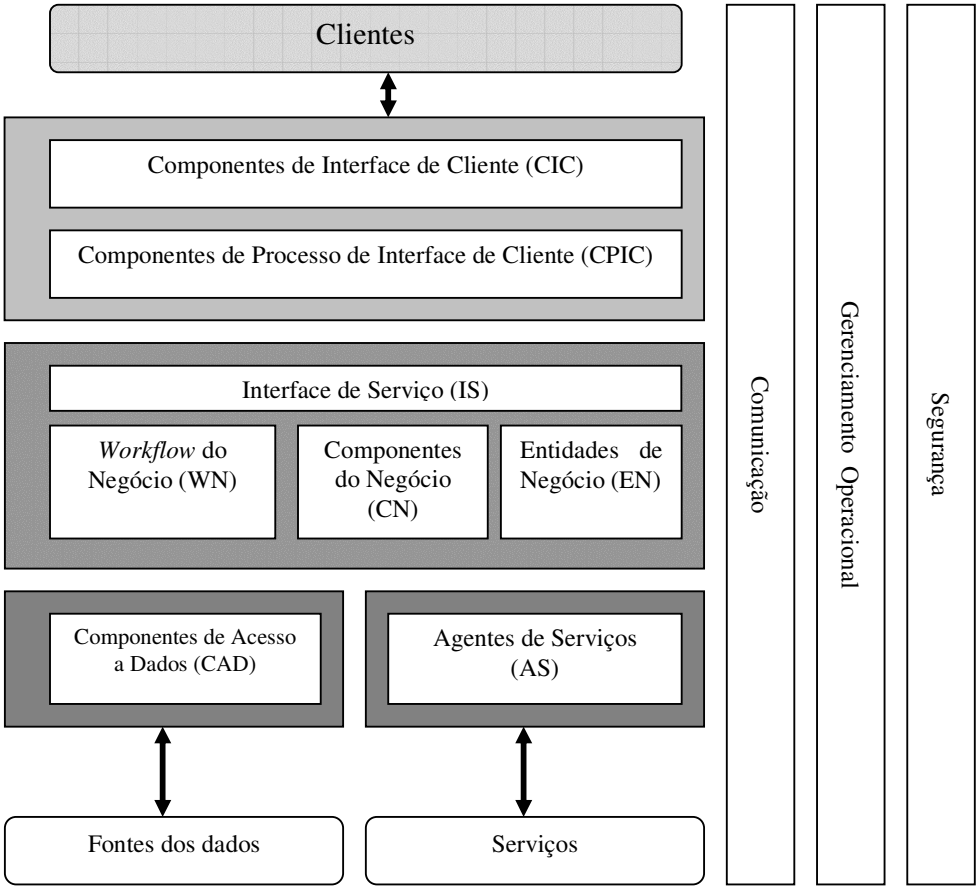

Figura 2.4 Serviços de aplicação em três camadas

O *Three-Layered Services Aplication*, como apresentado aqui, é basicamente uma arquitetura em três camadas. As três camadas são conceituadas como sendo:

- **Apresentação**. A camada de apresentação fornece as aplicações de interface de cliente. Pode ser usada a ferramenta Microsoft .NET *Windows Forms* para criar telas na linguagem ASP.NET.

- **Negócio**. A camada de negócio implementa as funcionalidades de negócio da aplicação. As camadas de domínio podem ser desenvolvidas utilizando uma ou mais linguagens de programação .NET como VB e C#. Estes componentes podem ser complementados com *Microsoft .NET Enterprise Services* para soluções escaláveis de componentes distribuídos, *Microsoft BizTalk Server* para *workflows* orquestrados ou *Web Service*.

- **Dados**. A camada de dados fornece acesso para sistemas externos, como banco de dados. A tecnologia primária .NET envolvida nesta camada é o ADO.NET. Contudo, pode ser também usado o .NET XML.

Cada camada pode ser estruturada e descrita como nos próximos itens:

2.3.5.1 Camada de Apresentação

A camada de apresentação consiste em páginas com as quais o cliente interage. Cada página contém campos vindos das camadas inferiores, bem como a coleta das entradas de dados informadas pelos clientes.

Dois tipos de componentes que implementam paginas baseadas em interfaces de cliente são:

- Componentes de Interface de Clientes e

- Componentes de Processos de Interfaces de Clientes.

Componentes de Interface de Clientes: Para aplicações de cliente, este padrão utiliza Componentes de IC da coleção de classes *.NET Framework System.Windows.Forms*. Para aplicações Web utiliza componentes ASP.NET. Se os componentes padrões não atenderem às necessidades, o .NET suporta subclasses dos componentes padrões ou até mesmo que sejam utilizados componentes da instituição no *framework.*

Componentes de Processo de Interface de Clientes: As Interfaces de clientes complexas freqüentemente requerem muitas telas altamente complexas. Para aumentar o reusabilidade, a manutebilidade e a extensibilidade, pode-se criar um

processo de interface de cliente separado, que servirá para encapsular dependências entre telas e a lógica associada com a navegação entre eles. Pode-se aplicar o mesmo conceito para as dependências, validações, e navegação entre os componentes de uma tela simples. Estes componentes podem ser aplicados baseados no padrão *Mediator* [Gamma, 1995].

A integração entre os componentes CIC e CPIC pode ser feita pela aplicação dos padrões *Model-View-Controller* ou *Presentation-Abstraction-Controller* [Buschmann, 1996].

2.3.5.2 Camada de Negócio

As aplicações normalmente são estruturadas em torno dos conceitos de processos de negócios ou componentes de negócio. Estes conceitos são expressos por meio de um número de componentes, entidades, agentes e interfaces na camada de negócio.

Componentes de Negócio: Componentes de negócio são as realizações de conceitos de negócio. Eles são a unidade primária do projeto, implementação, manutenção e gerenciamento para o ciclo de vida da aplicação do negócio. Componentes de negócio encapsulam a lógica de negócio, também chamada de regras de negócio. Estas regras contêm o comportamento dos conceitos do negócio para atender às necessidades de uma empresa.

***Workflow* de Negócio**: Os processos de negócio refletem um nível macro das atividades que são exigidas pelo negócio. Os processos são encapsulados pelos componentes do *workflow* de negócio. Estes conduzem um ou mais componentes de negócio para implementar o processo de negócio.

Entidade de Negócio: As entidades de negócio são depósitos de dados. Eles encapsulam e escondem os detalhes de formatos específicos de representação de dados. Além disso, uma entidade de negócio pode inicialmente encapsular um *recordset* obtido de um banco de dados relacional. Em seguida, a mesma entidade de negócio poderá ser modificada para envolver um documento XML, provocando mínimos impactos para o resto da aplicação.

Componentes de negócio e de *workflow* de negócio podem interagir independentemente de componentes de entidades de negócio, ou podem usá-los para alterar seus próprios estados. Entidades de negócio podem ser usadas como descrito no padrão *Data Transfer Objects* [Fowler, 2003]. Os componentes de acesso a dados freqüentemente retornam entidades de negócios ao invés de estruturas específicas de banco de dados. Com isso, ajuda a isolar detalhes de banco de dados na camada de banco de dados.

Serviços de Interface: Uma aplicação pode expor algumas de suas funcionalidades como serviço para que outras aplicações possam usá-las. O serviço de interface apresenta seus serviços para o mundo exterior da aplicação. Na essência, esconde detalhes da implementação e expõe somente as informações necessárias para atender solicitações do negócio. A implementação do Serviço de interface pode ser feita utilizando o *XML Web services*.

2.3.5.3 Camada de Dados

A maioria das aplicações de negócios precisam acessar dados que são armazenados em banco de dados corporativos, sendo que na maioria são bancos de dados relacionais. Os componentes de acesso a dados nesta camada são responsáveis por expor os dados armazenados nos bancos de dados para a camada de negócio.

Componente de Acesso a Dados: Componentes de acesso a dados isolam as camadas de negócio dos detalhes de uma solução específica de armazenamento de dados. Este isolamento fornece os seguintes benefícios:

- Minimiza os impactos das mudanças no banco de dados;
- Minimiza o impacto das mudanças nas representações de dados (Ex: mudança no esquema do banco de dados);
- Encapsula todo o código que manipula um item de dados em particular em um único lugar. Facilitando, assim, os testes e as manutenções.

Neste caso, o ADO.NET pode ser usado diretamente como um componente de acesso a dados para aplicações simples. Aplicações mais complexas podem ser beneficiadas desenvolvendo classes a partir do ADO.NET.

Agentes de Serviços: Componentes de negócio freqüentemente precisam acessar interna ou externamente serviços ou aplicações. Um agente de serviço é um componente que encapsula sua interface, protocolo, e código requisitado para usar tais serviços. Por exemplo, uma solução de negócio requisita informações do sistema de contabilidade para completar o processo de negócio. A solução é delegar toda a interação com o sistema de contabilidade ao agente de serviço. O agente de serviço torna mais fácil as mudanças externas dos serviços fornecidos. O agente de serviço pode também simular serviços externos para facilitar os testes da camada de domínio.

2.3.5.4 Serviços Fundamentais

Em adição, as três camadas padrões, *Tree-Layered Services Aplication,* definem um conjunto de serviços fundamentais que todas as camadas podem usar. Estes serviços estão contidos em três categorias básicas:

- **Segurança**. Estes serviços mantêm a segurança da aplicação.
- **Gerenciamento operacional**. Estes serviços gerenciam componentes e recursos associados, bem como requisitos operacionais como escalabilidade e tolerância a falhas.
- **Comunicação**. Estes são serviços como o *.NET remoting, SOAP,* e mensagens assíncronas, na qual fornecem a comunicação entre os componentes.

2.3.6 Proposta de arquitetura em três camadas apresentada por Larman

Para complementar e embasar os conceitos do padrão apresentado nos itens anteriores, cita-se [Larman, 2001] na abordagem de um padrão de arquitetura. Este padrão é o padrão Layers, que possui conceitos fundamentais de arquitetura de software (separação de áreas de interesse, baixo acoplamento, alta coesão). Larman

apresenta um exemplo de organização de um sistema em camadas, como pode ser visto na **Figura 2.5**.

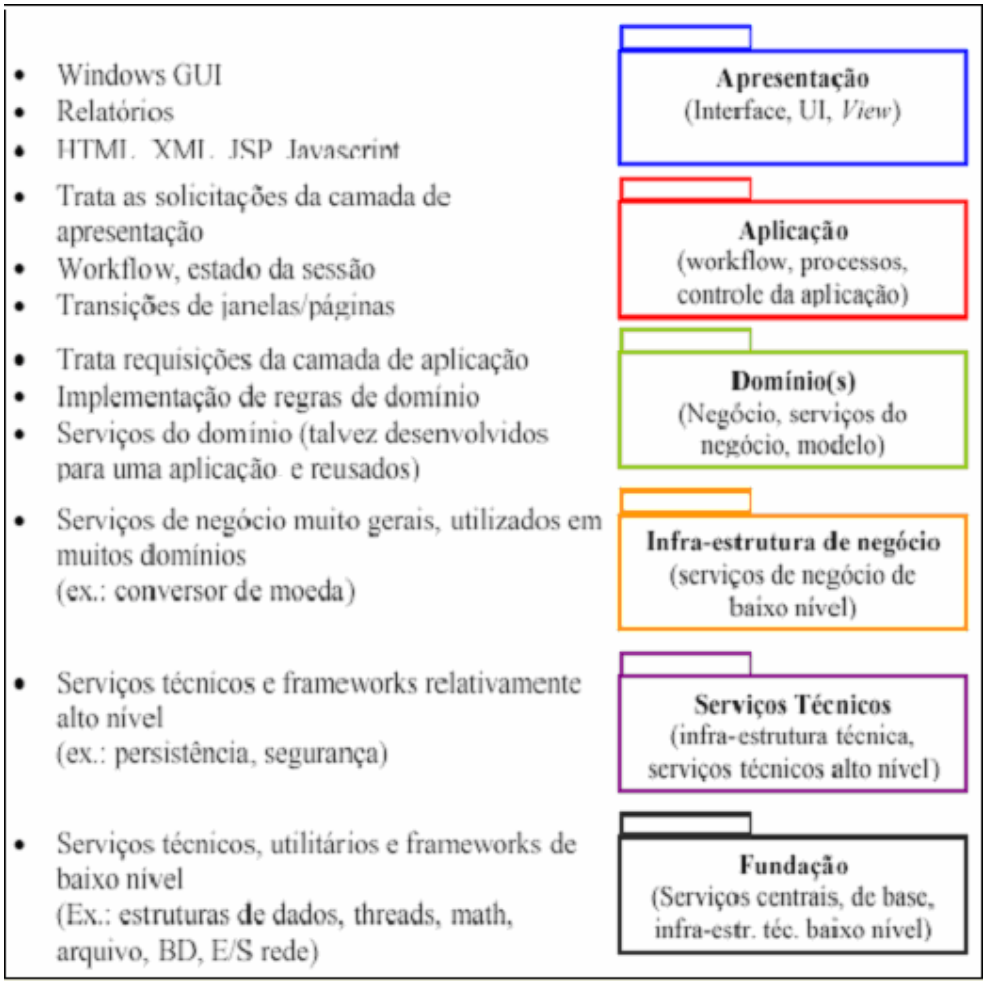

Figura 2.5 Arquitetura em camadas apresentada por Larman

Cada uma das camadas é representada por um pacote, em que um pacote é um elemento estrutural da UML, que serve para conter outros elementos de modelagem. Como se trata de uma visão lógica da arquitetura, não se entra no mérito de componentes (elementos de implementação, não de projeto); por isso escolheu-se representar uma camada como sendo um pacote. Segundo conceitos da UML, um pacote pode conter outros pacotes e também outros elementos de modelagem (como classes). Pode-se entender, portanto, que o pacote que representa cada camada normalmente vai conter outros pacotes, que representam os elementos estruturais de maior granularidade contidos em cada camada (subsistemas, por exemplo).

2.4 Padrões de projetos em aplicações .NET

O objetivo deste item é enfatizar de que forma é a relação entre padrões e a bibliotecas de classes .NET. Nos próximos itens são descritos exemplos de padrões que podem se relacionar com classes da biblioteca de classes Microsoft .NET.

Para dar um breve entendimento referente às classes Microsoft .NET, [Maioriello, 2002] cita: "A Microsoft .NET possui um framework de bibliotecas de classes que fornece acesso a um baixo nível de funcionalidades do sistema e é designado para ser o alicerce no qual todas as aplicações .NET são construídas".

Este framework de bibliotecas de classes fornece classes concretas e abstratas que podem ser usadas diretamente ou para construir tipos específicos que derivam suas definições das classes de interface .NET.

A relação entre padrões e .NET é dada pelo fato de que a plataforma .NET é orientada a objetos e padrões de projetos que são na essência OO [Maioriello, 2002].

O .NET framework possui diferentes coleções de classes [Microsoft, 2001], e um exemplo delas é ilustrado na **Figura 2.6**. Algumas possuem responsabilidades específicas, como classes *System.Reflection* e *System-Web-Security,* enquanto outras são dadas em termos mais gerais como as classes *System.Collection.* Dentro de cada coleção de classes, há classes concretas e abstratas que provêm muitos meios para alcançar uma determinada tarefa.

O grupo de classes *System.Collection* expõe um grande número de classes com propósitos gerais, mas também fornece classes especializadas que podem ser utilizadas pelas classes da aplicação. Por exemplo: pode ser criada uma coleção de classes por herança de uma das muitas coleções de classes do .NET Framework e ser adicionado códigos para implementar a customização de funcionalidades.

O problema com esta abordagem é que os tipos (variáveis/referência de objeto) das customizações são expostos e estão abaixo do sistema .NET. Qualquer

mudança feita no .NET (ex: correções de erros/novas características) propagará para as classes inferiores. Isto aumentará o tempo necessário na manutenção da aplicação. Neste momento, a abordagem de padrões pode ser usada para fornecer as seguintes opções:

a) Identificar a fonte de variação;

b) Modelar uma estrutura de classes que possa isolar esta fonte para a aplicação.

Concluindo, a situação mencionada anteriormente leva a criação de uma interface simples para o *System.Collection,* com o intuito de obter as funcionalidades necessárias. Por fim, todos os tipos de customizações deverão derivar desta interface.

A **Figura 2.7** tem como objetivo ilustrar a relação entre classes .NET e os padrões de projetos descritos em [Gamma, 1995].

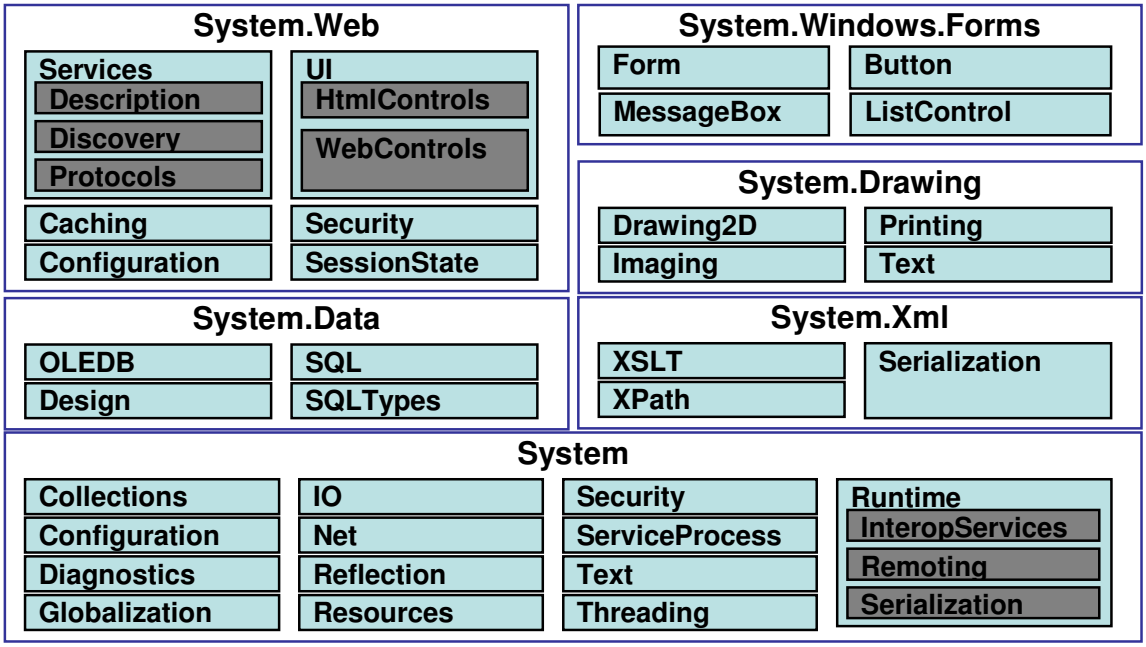

Figura 2.6 Coleções de classes .NET

O dois sub-itens seguintes descrevem em detalhes sobre estas relações.

2.4.1 Padrões de estrutura

Esta camada indireta, citada no exemplo anterior, proporciona um meio de criar coleções customizadas menos acopladas e mais coesas. Este é um exemplo do uso do padrão chamado *Façade*, que tem como intenção isolar um subsistema do cliente colocando um subsistema atrás de uma interface simples e que expõe apenas as funcionalidades que o cliente precisa. Pode ser usado para criar uma interface simples para qualquer coleção de classes .NET.

Um outro padrão utilizado para resolver características específicas de estrutura é o *Adapter*. Este padrão tem o objetivo de prover uma classe que converte uma interface incompatível para uma que o cliente espera. Ele também pode ser utilizado para criar tipos de customizações que derivam de uma coleção *System.Xml.* Desta forma, o XML emitido de um tipo de customização pode ser customizado para as necessidades de um outro cliente.

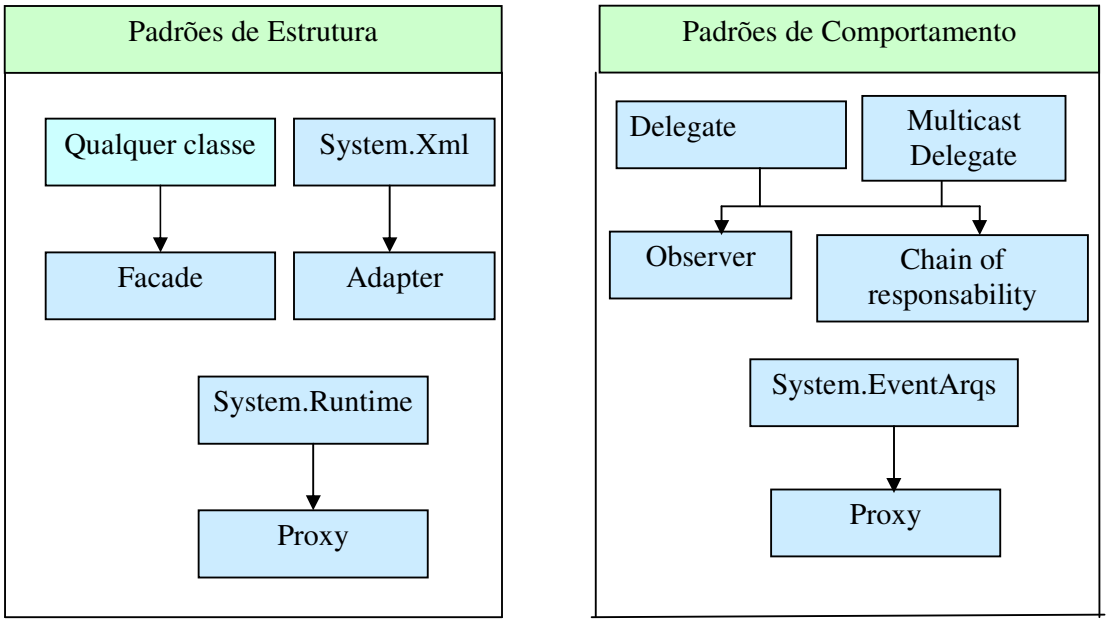

Figura 2.7 Relação dos padrões com classes .NET

No caso do padrão *Proxy,* que tem por objetivo oferecer um objeto substituto para outro objeto para controlar seu acesso, faz uso das coleções de classes remotas *System.Runtime,* que por sua vez providenciam suporte para a criação de objetos *Proxy* e serializam sua representação.

2.4.2 Padrões de comportamento

Em complemento ao Padrão Estrutural, os padrões de comportamento foram concebidos para solucionar problemas de projetos relacionados com comportamentos de aplicações. Na plataforma .NET os padrões podem ser usados para promover baixo acoplamento entre as classes. Por exemplo, a coleção de classes .NET *Delegates* utiliza o padrão *Observer* para fornecer um mecanismo de retorno, pelo qual um objeto que cria um evento pode invocar chamadas de métodos em eventos de objetos sem o conhecimento do cliente.

Neste modelo, o evento de origem declara um objeto chamado de *Delegate*. Qualquer classe, desejando ser notificada sobre este evento, simplesmente irá providenciar um método com a mesma assinatura como é do *Delegate* e este método responderá para o evento.

Esta abordagem permite facilmente adicionar eventos receptores sem a necessidade de modificar o código do evento origem. Tudo que é preciso fazer é criar um método com a mesma assinatura do *Delegate* e responder ao evento. O *Delegate* gera um nível de direcionamento que manterá o evento origem com informações sobre qual objeto receberá o evento.

Este mecanismo pode ser incrementado usando o padrão *Chain of Responsability* . Este padrão descreve uma abordagem para evitar acoplamento entre a origem e o destino da mensagem, enquanto permite que mais de um objeto possa responder à mensagem.

No .NET, este padrão pode ser implementado usando as classes *Multicast Delegates*. Esta coleção de classes difere de um *Delegate* regular por referenciar para mais de um método.

Quando o *Multicast Delegates* é invocado, os métodos são executados de forma síncrona na ordem que aparecem. Então, pode-se enviar um evento para uma corrente de objetos e dar a qualquer um deles a chance de responder. Este padrão

pode ser usado para manter *Event Handlers* para menu de itens e outras interfaces de controles.

Outro padrão que pode ser usado para referenciar classes .NET é o *Command*. Este padrão descreve como encapsular a requisição de um objeto. Permite a parametrização da requisição reconhecendo-o como objeto.

No .NET, a incorporação do padrão *Command* tem por objetivo fazer uso das classes *System.Event.Arqs,* em que tipos de customizações poderão derivar destas classes, sendo possível definir dados customizados associados a eventos destas mesmas classes.

2.4.3 Considerações

Os exemplos citados nos itens anteriores são apenas amostras do uso de classes .NET na utilização de padrões. Outros exemplos deste uso podem ser vistos em [PnP, 2003], em que padrões de outros autores, além de Gamma, são apresentados e fazem uso das classes .NET. Também em [Cooper, 2002], pode-se ver exemplos práticos de aplicação dos padrões Gamma utilizando a linguagem C# .NET.

3 MÉTODO DE APLICAÇÃO DE PADRÕES NO PROCESSO UNIFICADO

Neste capítulo será apresentado o Método de Aplicação de Padrões no Processo Unificado (MAPPU).

A **Figura 3.1** apresenta a estrutura do método, na qual são apresentados os seguintes componentes: Processo Unificado, Processo, Aplicação de padrões de Arquitetura, Aplicação de Padrões de Projeto, Padrões de referência e Biblioteca de Classes.

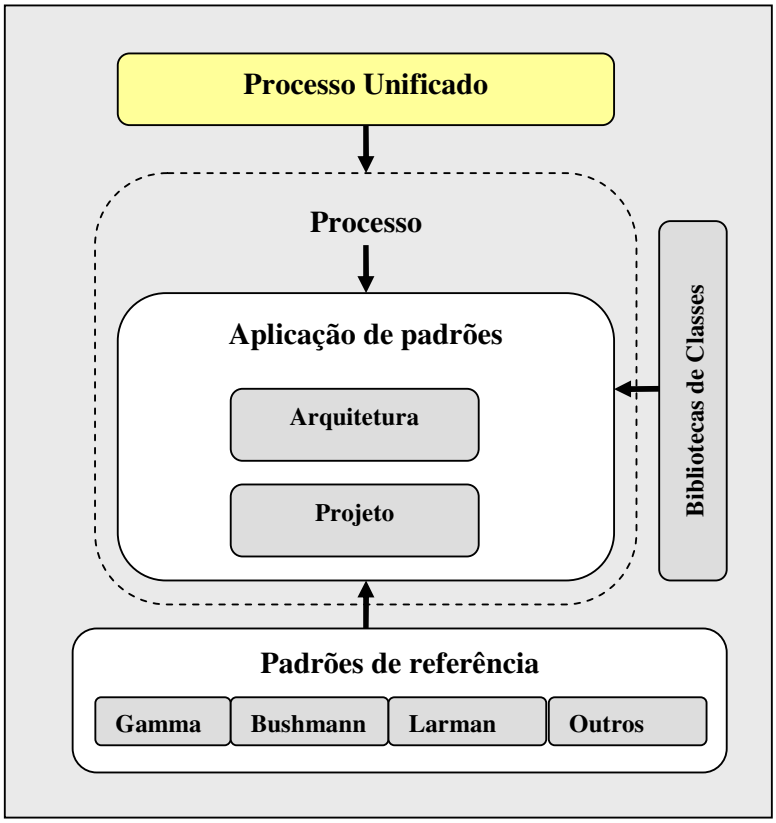

Figura 3.1 Estrutura do método

Na **Figura 3.1** a seta relacionando o componente Processo Unificado e o componente processo, identifica que o Processo Unificado tem como função guiar o método com as diretrizes do processo. No componente aplicação de padrões, a

Arquitetura e o Projeto representam as disciplinas do método. A seta entre a Aplicações de padrões e o componente Padrões de referência representa que o método referencia-se a padrões conhecidos. E por ultimo, a seta que relaciona a Biblioteca de classes e a Aplicação de padrões, indica o relacionamento entre o método e as bibliotecas de classes.

A seguir será descrita em detalhes a função que cada componente possui no contexto do método.

3.1 DETALHAMENTO DO MÉTODO

Os componentes que fazem parte do método são descritos nos próximos itens. Cada um, exceto Padrões de referência e Biblioteca de classes, é apresentado com seus participantes e suas diretrizes.

3.1.1 Processo

Este componente do método tem como participantes gerentes de projetos, líderes e coordenadores responsáveis por planejar e gerir as atividades relacionadas a este componente. O objetivo é direcionar estes participantes na organização das atividades no desenvolvimento de um projeto de software.

A organização deste tópico é definida em função da organização da disciplina de Análise e Projeto, no UP. O UP é descrito em termos de detalhes de *workflow,* que por sua vez, são detalhados em termos de atividades. Os detalhes de *workflow* agrupam atividades afins, mas o processo efetivamente é descrito pelas atividades e artefatos correlacionados. A **Figura 3.2** apresenta o *workflow* de Análise e Projeto, como um diagrama de atividades da UML, no qual cada elemento representa um detalhe de *workflow*.

Ainda na **Figura 3.2,** são apresentadas as atividades da disciplina de Análise e Projeto que se correlacionam com a aplicação de padrões, pode ser visto na aplicação de padrões de arquitetura que a sua aplicação se dá na atividade de Definir uma Arquitetura Candidata. No caso da aplicação de padrões de projetos, a figura

ilustra que esta atividade acorrerá nas atividades de Refinar a Arquitetura e Analisar Comportamento.

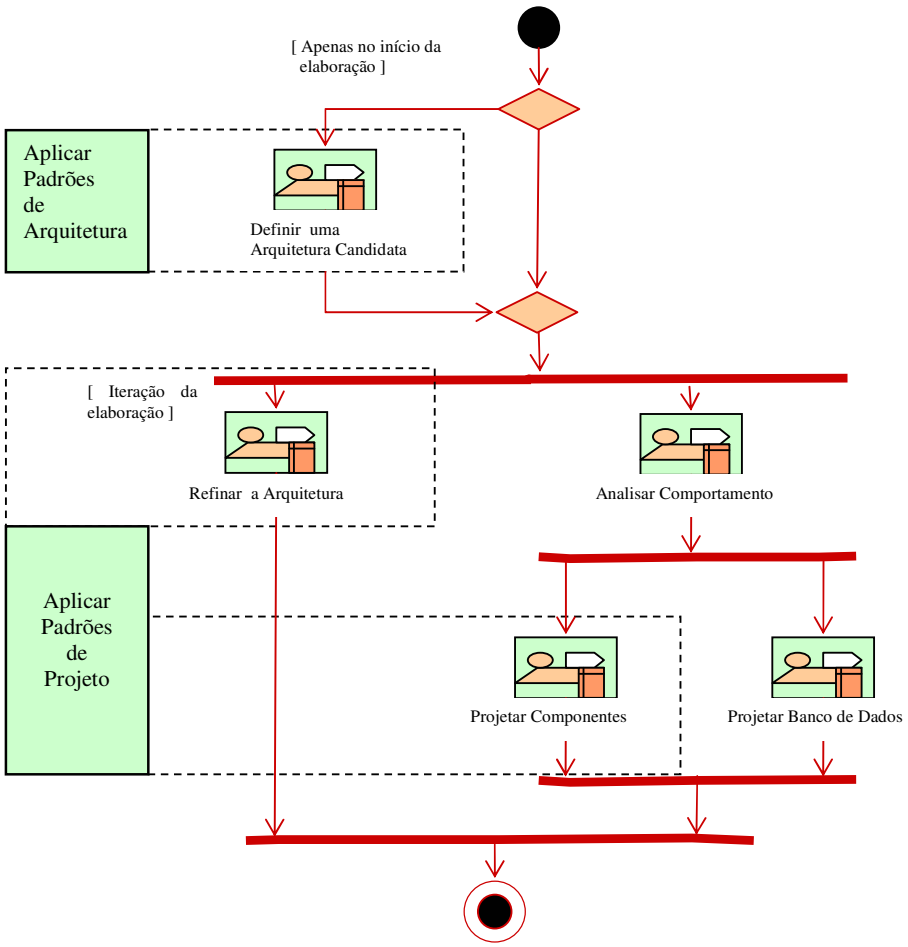

Figura 3.2 *Workflow* da disciplina de Análise e Projeto

A **Figura 3.3** ilustra o esforço necessário, exigidos dos participantes do projeto, para realizar a disciplina de análise e projeto. As barras de diferentes tamanhos que aparecem em cada fase representam o tempo utilizado para realizar a as atividades desta disciplina nestas mesmas fases. Isto significa que a disciplina tem seu início na fase de Concepção e vai tomando mais tempo dos participantes do projeto quando vai se aproximando do final da fase e início da fase de Elaboração. O esforço atinge o máximo na fase de Elaboração e sua intensidade vai diminuindo a medida que decorre a fase de Construção, onde finalmente deixa de ser utilizada.

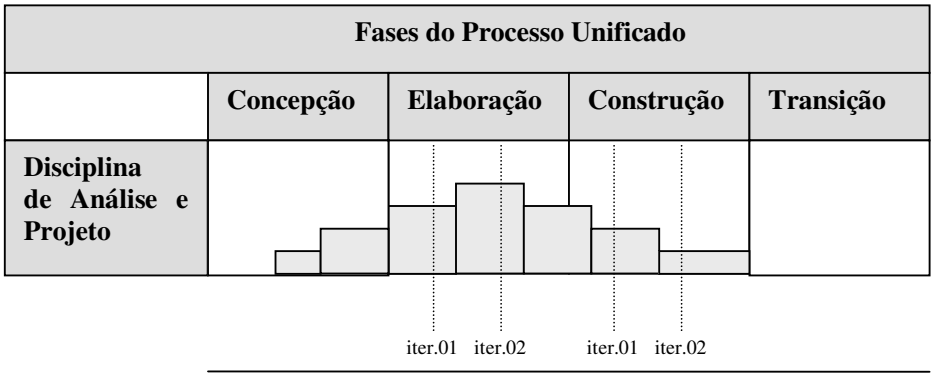

Figura 3.3 Esforço da Análise de Projeto nas fases do Processo Unificado

Por fim, as linhas pontilhadas em cada fase representam as iterações do processo. Neste caso, cada fase pode ter uma ou mais iterações, esta decisão deve ser tomada pelos participantes do projeto, que devem visar um sucessivo refinamento do projeto. A figura não apresentou iteração na primeira fase, isto acontece por ser uma fase de curta duração e não cabem iterações. No final de cada iteração os responsáveis devem definir os próximos passos e determinar se o projeto deve continuar ou ser finalizado.

Apenas para relacionar este item a trabalhos de outros autores, cita-se Alencar em [Alencar, 1995a, 1995b, 1996], que apresenta um método na qual enfatiza a importância do uso de padrões de projetos aliado a um processo de desenvolvimento de software.

Também Oliver Mehl em [Mehl, s.d.], apresenta um modelo onde ambiente de TI é totalmente integrado aos negócios do cliente, ao ambiente tecnológico e ao processo de desenvolvimento de software. Neste modelo, o uso de um método de aplicação de padrões surge como sendo parte do processo de desenvolvimento de software.

O método desta dissertação também possui o mesmo objetivo ao proposto por estes autores, ou seja aplicação de padrões em um processo. No entanto, as semelhanças estão apenas no objetivo, uma vez que nem as diretrizes do método nem o processo são similares.

3.1.2 Padrões de referência

O objetivo do método é auxiliar na elaboração de projetos a partir de projetos existentes. Para isso, o método se fundamenta no uso de padrões de projetos de autores conhecidos, aqui denominados de padrões de referência. Trabalho semelhante a este pode ser visto em [Mehl, s.d.] que propõe o uso de uma memória de padrões com a denominação de catálogo de padrões.

Entre os padrões conhecidos estão: [Gamma, 1994], [Bushmann, 1996], [Larman, 1999] e [PnP, 2002]. Estes são alguns padrões que poderão ser usados como referência na aplicação do método.

A **Tabela 3.1** apresenta duas colunas onde são ilustrados os padrões e seus autores. Na coluna da esquerda estão descritos os autores e nas superiores o tipo de padrão, no cruzamento de ambas está descrito alguns padrões de cada autor. No caso de Buschmann e Pnp, os seus padrões são considerados padrões de arquitetura e Gamma e Larman padrões de projeto. Esta tabela ilustra a distinção entre padrões de arquitetura e padrões de projeto dos padrões dos autores acima citados.

Autores	**Padrão de Projeto**
Gamma	Abstract Factory, Builder, Factory Method, Prototype, Singleton, Adapter, Bridge, Composite, Decorator, Façade, Flyweight, Proxy, Chain of Responsability, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor.
Larman	Expert, Creator, Controller, Low Coupling, High Cohesion, Polymorphism, Purê Fabrication, Indirection, Don't Talk to Strangers
	Padrão de Arquitetura
Buschmann	Layers, Blackbord, Pipers and Filters, Broker, ModeL-View-Controller, Presentation-Abstraction-Control, Micro Kernel, Reflection.
Pnp	Three-Layered Services Aplication

Tabela 3.1 Tabela dos autores e padrões

3.1.3 Bibliotecas de classes

Um padrão pode ser utilizado tanto no *framework* J2EE como no .NET. No entanto, apesar da independência, este componente apresenta possíveis relações de padrões com as classes das bibliotecas de classes do *framework* .NET.

Classes .NET	Padrões	
Qualquer classe	Façade	Estrutura
System.Xml	Adapter	
System.Runtime	Proxy	
Delegate	Observer e Chain of responsability	Comportamento
Multicast Delegate	Observer e Chain of responsability	
System.EventArqs	Proxy	

Tabela 3.2 Relação de classes .Net e Padrões

A **Tabela 3.2** apresenta a relação entre classes .NET e padrões de projetos, na coluna de Classes .NET constam algumas classes que podem ser referenciadas quando for utilizado algum dos padrões identificados na coluna de padrões. Estes padrões por sua vez estão conceituados em [Gamma, 1995] como sendo de estrutura e de comportamento.

3.1.4 Aplicação de padrões

São dois os componentes que fazem parte da aplicação de padrões: A aplicação de padrões de projeto e a aplicação de padrões de arquitetura. A seguir cada aplicação é vista com maiores detalhes.

3.1.4.1 Aplicação de padrões de arquitetura

A **Figura 3.4** apresenta o *workflow* da disciplina Análise e Projeto, e identifica a atividade Aplicar padrões de Arquitetura, ressaltados nos tracejados, como parte de detalhe do *workflow* denominado Definir uma Arquitetura Candidata.

O Processo Unificado define que o objetivo desta atividade é desenvolver:

- uma arquitetura candidata (a ser a arquitetura efetiva do sistema, e que será posteriormente detalhada e validada);
- padrões (*patterns)* de arquitetura, mecanismos chave e convenções de modelagem para o sistema.

As decisões tomadas pelo arquiteto de software neste ponto influenciarão toda a análise. É como se essa atividade funcionasse como uma configuração de uma atividade executada posteriormente na Análise dos Casos de Uso, na qual os casos de uso são realizados, em termos de objetos que interagem, para executar a funcionalidade descrita textualmente pelo caso de uso.

Caso se adote a abordagem de arquitetura em camadas pode-se dizer que esta atividade foca as camadas mais altas do sistema, definindo a macro organização do sistema em elementos (pacotes, subsistemas) que interagem através de interfaces definidas. Outras atividades focam as camadas mais baixas da arquitetura.

Em essência, o Processo Unificado identifica quatro passos referentes a essa atividade:

- Definir a organização dos subsistemas em alto nível (vista lógica da arquitetura de software);
- Identificar mecanismos de análise;
- Identificar abstrações chave;
- Criar as realizações de casos de uso.

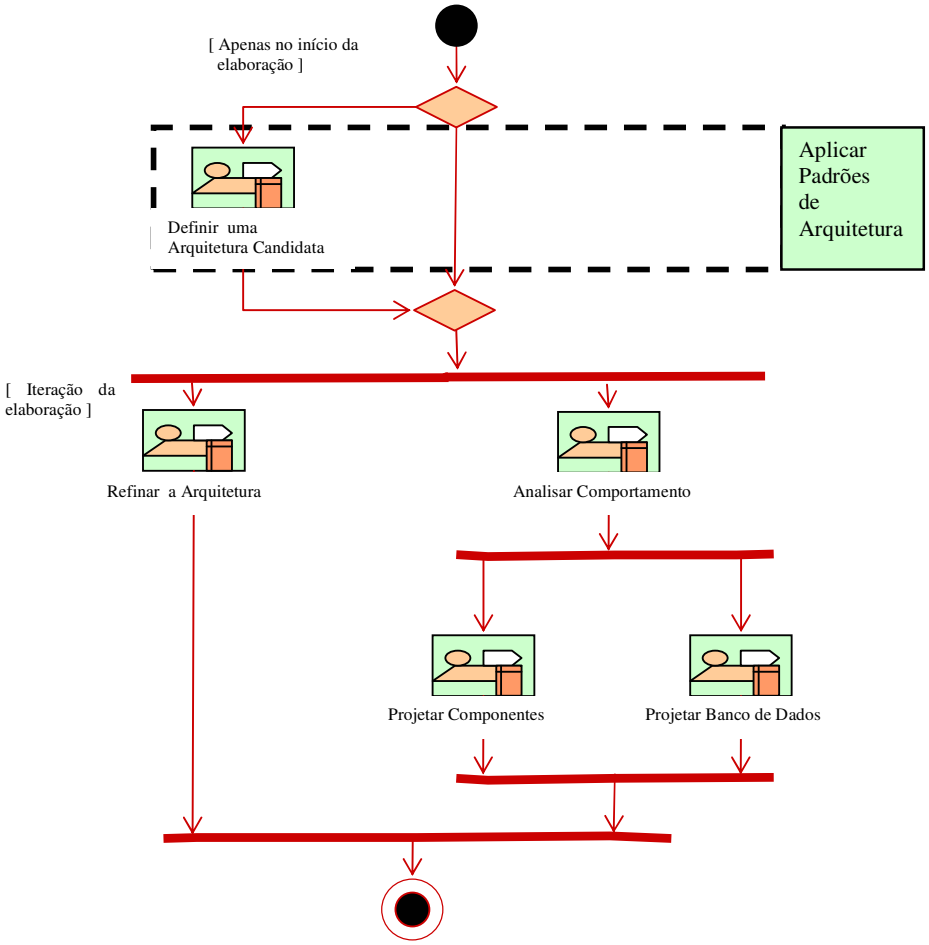

Figura 3.4 Workflow da disciplina de Análise e Projeto, destacando a atividade de Aplicar padrões de Arquitetura

Dos quatro itens citados anteriormente, apenas Definir a organização dos sistemas em alto nível é considerado neste trabalho. Nesta atividade o arquiteto deve analisar os requisitos do sistema e elaborar o modelo de arquitetura, e este modelo de arquitetura deverá ser inspirado nos modelos apresentados pelos padrões de referência.

A seguir será apresentada a atividade de aplicar padrões de arquitetura na sua forma operacional, onde são descritos os passos desta atividade e seus respectivos participantes.

3.1.4.1.1 Atividades da aplicação de padrões de arquitetura

A **Figura 3.5** apresenta o fluxo das atividades e os responsáveis por executá-las, pode se ver na parte superior os responsáveis pelas atividades e abaixo dos mesmos as respectivas atividades. Na seqüência são detalhadas estas atividades:

Atividade 1 – Analisar requisitos do sistema é a atividade onde se obtém todas as informações necessárias para dar início a elaboração da arquitetura. Nestes requisitos devem conter todas as informações necessárias que propiciem a elaboração da arquitetura.

Atividade 2 – Definir a arquitetura candidata é o esboço da arquitetura elaborada a partir dos requisitos de arquitetura, nesta atividade elabora-se as necessidades da organização em termos de camadas de sistema.

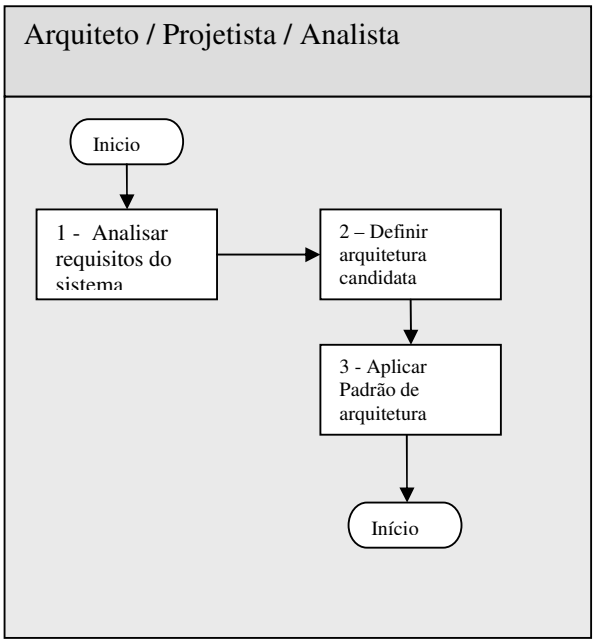

Figura 3.5 Fluxo da atividade de aplicar padrão de arquitetura

Atividade 3 – Aplicar Padrões de arquitetura é atividade de busca por padrões referência, ou seja, tendo como requisito uma arquitetura candidata, refina-se ainda mais esta arquitetura aplicando um padrão referência.

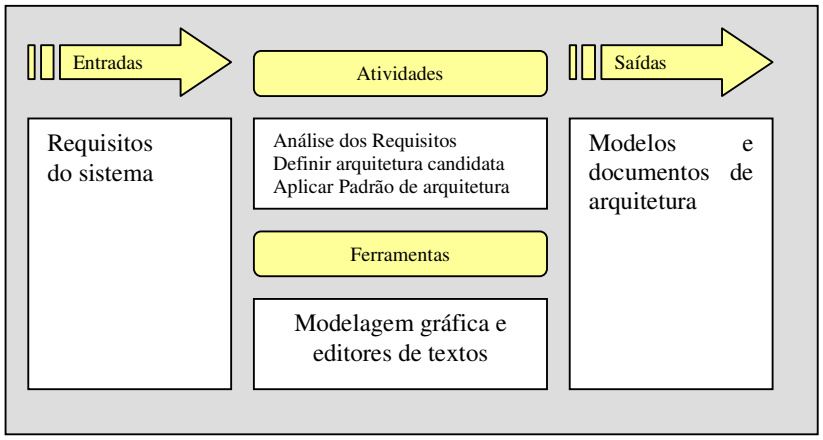

Figura 3.6 Entradas, saídas, atividades e ferramentas para aplicar padrões de arquitetura

A **Figura 3.6** apresenta as entradas, atividades, ferramentas e saídas destas atividades. Apresenta os requisitos de sistema como sendo entradas da atividade de análise dos requisitos. No caso das disciplinas definir Arquitetura Candidata e aplicar Padrão de Arquitetura, as suas entradas obedecem as saídas das atividades anteriores a elas. Apresenta como ferramenta desta atividade, ferramentas de modelagem gráficas e editores de texto e tem como saída modelos e documentos de arquitetura.

3.1.4.2 Aplicação de padrões de Projeto

Com o objetivo de tornar claro como esta atividade interage com as disciplinas do Processo Unificado, A **Figura 3.7** ilustra exatamente, através dos tracejados, quais são estas atividades. Estão envolvidas as atividades de Analisar Comportamento e Refinar Arquitetura, a seguir detalhadas.

O detalhe de *workflow* Analisar Comportamento transforma as descrições funcionais dos casos de uso em elementos que servirão como base para o projeto do sistema. A atividade chave é Analisar Caso de Uso e o produto desta atividade é constituído pelas colaborações (diagramas de classe e de interação, principalmente), ainda no nível de análise.

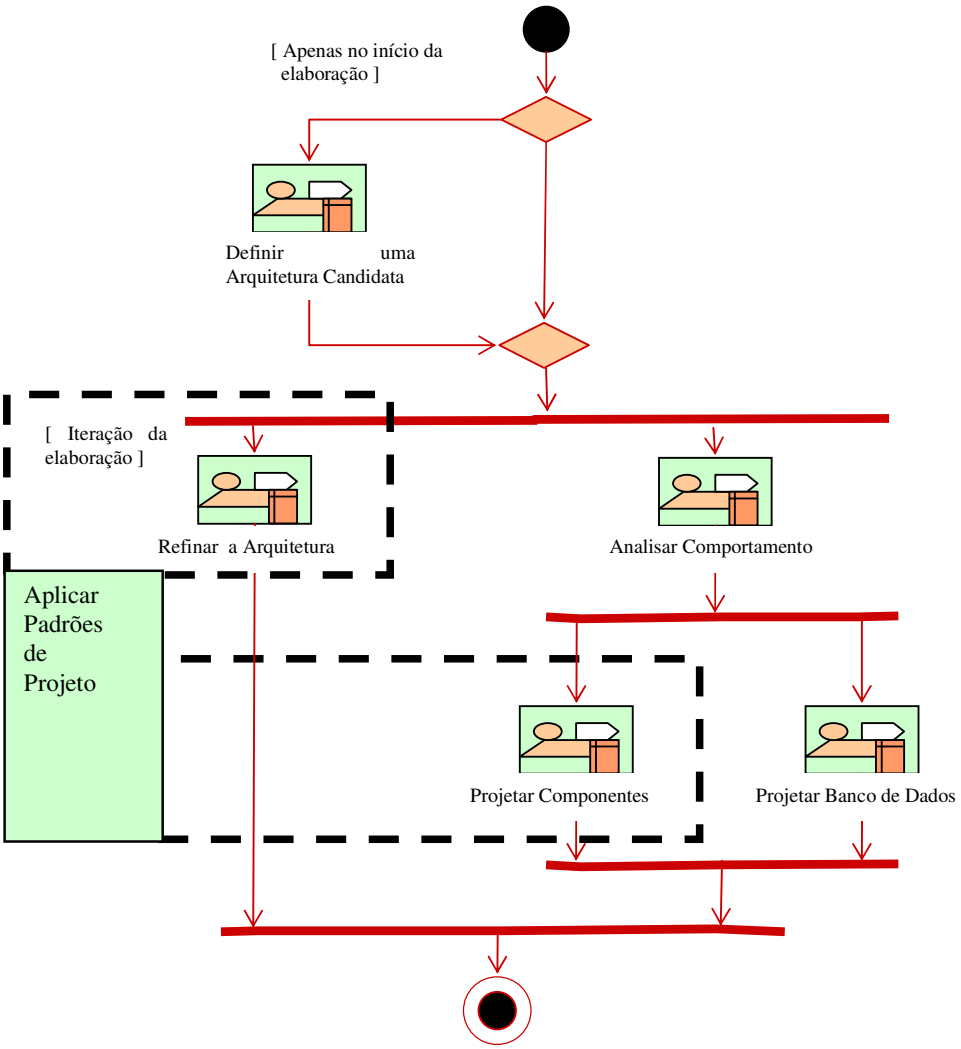

Figura 3.7 Workflow da disciplina de Análise e Projeto, destacando a atividade de Aplicar Padrões de Projeto

O detalhe de *workflow* Refinar a Arquitetura visa efetuar a transição da análise para o projeto, definindo os elementos de projeto, a partir de elementos de análise (por exemplo, enriquecendo os diagramas de classe e interações com os elementos tipicamente de projeto). Para manter a consistência da arquitetura de software em elaboração, os elementos de projeto identificados devem ser adequadamente integrados com elementos existentes (normalmente definidos em iterações anteriores).

Antes de descrever sobre o fluxo de atividades, sobre os responsáveis por estas atividades e sobre os artefatos de entrada e saída, serão conceituados; Estrutura,

Propósito e Implementação [Shull, 1996] que são utilizados nesta etapa para auxiliar na aplicação de padrões. Além das técnicas que serão descritas na seqüência, outros autores sugerem diferentes formas de avaliar padrões em um projeto, é caso de [Saeki, 2001] e [Muraki, 2002] que apresentam um método que utiliza técnicas de Meta modelo aliado a formulas matemáticas, este método apresenta um alto nível de complexidade no uso das formulas.

3.1.4.2.1 Avaliar a Estrutura, o Propósito e a Implementação

Estrutura - Refere-se a classes, objetos e aos relacionamentos com o qual o padrão é construído. Usa-se a estrutura como o primeiro passo na identificação de potenciais padrões, na comparação entre a estrutura dos potenciais padrões e dos padrões de referência podem-se identificar onde os mesmos possuem soluções equivalentes.

A **Tabela 3.3** apresenta alguns indicadores para auxiliar na identificação de potenciais padrões. A coluna de indicadores de estrutura de classes traz questionamentos sobre algumas características que indicam se a classe está propensa a se tornar um padrão. A outra coluna tem a função de validar se é ou não aplicável o questionamento para a classe que está sendo avaliada. Se uma ou mais questões forem aplicáveis, então a classe avaliada é um potencial padrão.

Propósito - Todo padrão precisa especificar para que ele deve ser usado, o que ele propõe a solucionar e quais efeitos (positivo/negativo) ele pode ter no sistema. A intenção é demonstrar onde desenvolvedores estão tentando resolver os mesmos problemas dos padrões de referência.

Implementação - Com a implementação o foco é avaliar se os padrões descobertos são específicos para uma determinada implementação, ou seja, se eles serão utilizados apenas para um projeto específico, não sendo útil para futuros projetos, ou se eles serão reutilizáveis.

Para analisar a conformidade entre potenciais padrões e padrões referência, utiliza-se a Tabela de Verificação de Conformidade, vista na **Tabela 3.4,**

para cada padrão, onde as análises deverão ser feitas na comparação com as descrições de cada padrão apresentadas por seus autores.

Indicadores de estrutura de classes	É aplicável
A classe está na linha final de comunicação de um modelo de classes ? (Se estiver, ela pode executar algumas regras na mediação de interações entre outras classes.)	[ ] Sim [ ] Não
A classe recebe mensagens de muitas classes ? (Se recebe pode agir como interface para outras classes)	[ ] Sim [ ] Não
Classe com herança paralela ? (Ou seja, cada subclasse de uma classe é relacionada com subclasse de outras classes ?) (Se forem são propensas a trabalharem muito próximas)	[ ] Sim [ ] Não
A classe possui alto grau de acoplamento ? (Se for pode indicar uma relação de comunicação entre dois sistemas)	[ ] Sim [ ] Não
A classe possui hierarquia todo-parte ?	[ ] Sim [ ] Não

Tabela 3.3 Indicadores de estrutura de classes

Implementação	Propósito: Parcial	Propósito: Completo
Completo	Apenas parte do padrão encontrado, mas esta parte tem uma implementação sofisticada — 2	Próximo ou exato — 4
Parcial	Não relevante — 1	Um padrão é encontrado e apresenta o mesmo propósito, mas sua implementação não satisfaz — 3

Tabela 3.4 Tabela de verificação de conformidade

Na mesma **Tabela 3.4,** há um cruzamento entre a Implementação e o propósito, e ainda dentro de cada um destes elementos está a indicação se atende parcialmente ou por completo as avaliações. Ao serem avaliados os potenciais padrões e os de referência, deve ser observado entre as quatro observações da tabela em qual delas se enquadra. Se alcançar a escala 4 na tabela, o padrão pode então ser aplicável no projeto.

3.1.4.2.2 Fluxo de atividades

O fluxo de atividades apresentado na **Figura 3.8** descreve as atividades e os responsáveis por estas atividades, pertencentes a este componente do método. Cada atividade deste fluxo deve ser executada da seguinte forma:

Atividade 1 - Elaborar Modelo de classe de analise é uma atividade de responsabilidade do Analista de sistemas e tem como objetivo o modelo de classes de análise, visando apresentar um modelo de classes preliminar.

Atividade 2 – Elaborar o modelo de classes de projetos é uma atividade de responsabilidade do projetista e tem como finalidade um modelo de classes de projetos, visando os aspectos físicos de implementação. Este modelo é um refinamento do modelo de classes de análise.

Atividade 3 – Identificar potenciais padrões é o refinamento do modelo de projeto, onde deve ser avaliada a estrutura do modelo de classes aplicando a tabela de Indicador de estrutura de classes.

Atividade 4 – Analisar potenciais padrões é o momento que deve ser verificado se o modelo de classe que está sendo avaliado pode ser substituído por um padrão de referência, para isso deve ser utilizada a tabela de verificação de conformidade.

Atividade 5 – Aplicar padrão é o momento que deve ser substituído o modelo de classes de projeto pelo modelo padrão identificado nos padrões de referência.

Atividade 6 – Verificar e aplicar classes .NET, são respectivamente verificar se é pertinente no novo modelo de classes herdar operações ou atributos das classes .NET e conseqüentemente aplicar estas relações.

Atividade 7 – Empacotar as classes significa agrupar as classes que possuem afinidades entre si para que possam ser distribuídas entre as camadas.

Atividade 8 – Distribuir os pacotes na arquitetura do sistema significa que deve ser avaliado em qual camada o pacote deve ser aplicado. Um exemplo disto é identificar qual pacote é de interface de cliente, qual é de negócio e qual é de acesso a dados, e assim distribuí-los nas suas respectivas camadas, ou seja interface de cliente na camada de interface de cliente, negócio na camada de negócio e dados na camada de acesso a dados.

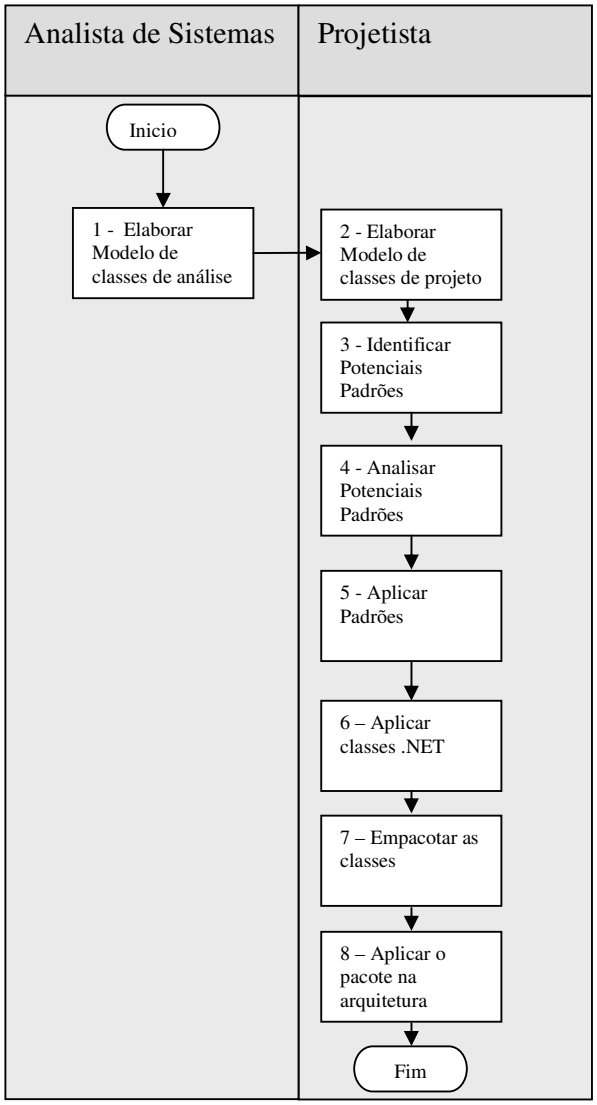

Figura 3.8 Fluxo de atividades da aplicação de padrões de projeto

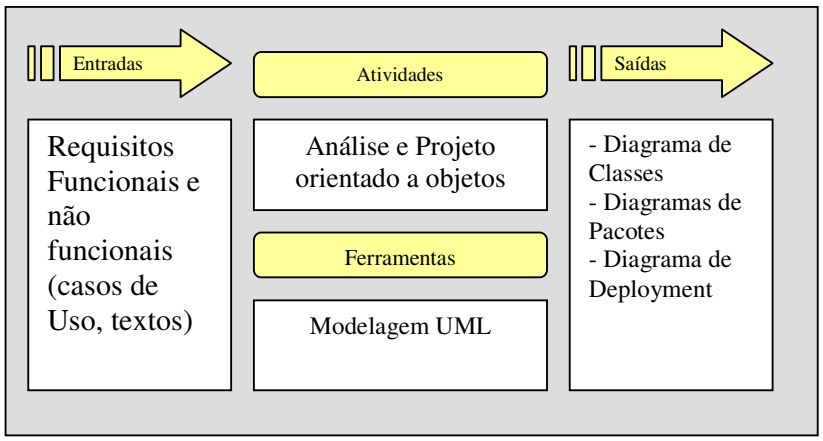

Figura 3.9 Entradas, saídas, atividades e ferramentas da aplicação de padrões de projeto

A **Figura 3.9** apresenta os requisitos funcionais e não funcionais como sendo as entradas das atividades de análise e projeto orientado a objetos utilizado na disciplina de aplicação de padrões de projeto. Apresenta como ferramenta destas atividades a modelagem UML [fowler, 2000] e tem como saída o modelo de classes de projeto.

3.2 Conclusão

Neste capítulo, foi descrito o método de aplicação de padrões no Processo Unificado, onde o mesmo foi dividido em duas etapas: aplicação de padrões de arquitetura e aplicação de padrões de projeto. Onde para cada uma apresentou-se os fluxos das atividades, a descrição de cada atividade e por fim as figuras de entradas, saídas, atividades e ferramentas.

O método foi apresentado na estrutura gerencial e operacional, objetivando dar a perspectiva gerencial e uma perspectiva operacional. Na perspectiva gerencial apresenta-se onde e quando este método pode-ser aplicado no Processo Unificado e na perspectiva operacional, apresenta-se como o mesmo pode ser aplicado.

4 EXPERIMENTO

Neste capítulo, é apresentado o experimento desta dissertação, onde o objetivo é utilizar o método descrito no capítulo anterior.

O experimento foi elaborado tendo como base um caso real, este caso é o Sistema de Custódia de Títulos da BOVESPA onde apenas algumas das funcionalidades deste sistema foram selecionadas única e exclusivamente por ser o objetivo deste capítulo experimentar o Método de Aplicação de Padrões no Processo Unificado.

Para que os requisitos do experimento sejam conhecidos, este capítulo descreve primeiramente o cenário do negócio de Custódia de Títulos da Bovespa, apresenta o glossário de termos e alguns casos de uso. Após estas informações é então descrito o experimento desta dissertação.

No caso dos padrões a serem experimentados, foram três os padrões escolhidos para serem aplicados. Para o experimento da aplicação de padrões de arquitetura foi utilizado o padrão *Three-Layered Services Aplication* [PnP, 2003] e para o experimento de padrões de projeto foram escolhidos os padrões *Façade* e *Composite,* ambos padrões Gamma [1995].

4.1 Cenário do Negócio

A **Figura 4.1** apresenta o fluxo de negociação do mercado de ações, em que podem ser vistos os principais participantes e suas atribuições dentro do processo. Tudo começa quando uma Empresa abre seu capital e faz uma negociação por meio de leilão de seus papéis. Esta empresa deposita suas ações (papéis/títulos) na CBLC para que estes sejam custodiados, neste caso representa uma negociação primária.

Ainda na **Figura 4.1**, após o primeiro ciclo, as ações entram para negociação no mercado secundário, onde as ações são disponibilizadas para o mercado. O investidor interessado em negociar ações, que pode ser comprar ou

vender, procura um Agente de Custódia (Corretora) para que este negocie em sua ordem.

O Agente de Custódia, por sua vez, repassa para o Participante da negociação efetivar o negócio na Bolsa. O Agente de Custódia, para fazer esta negociação, deve ser um associado da CBLC com autorização de negociar por ordem do Investidor.

Todas as ações são mantidas em Custódia na CBLC, porém isto não a torna proprietária das ações custodiadas. Na Custódia, a CBLC mantém um saldo de controle das ações, em que pode ser identificado a qual Investidor elas pertencem.

Para movimentar este saldo, o Agente de Custódia e a CBLC podem Depositar, Retirar e transferir este saldo, sempre por ordem do Investidor.

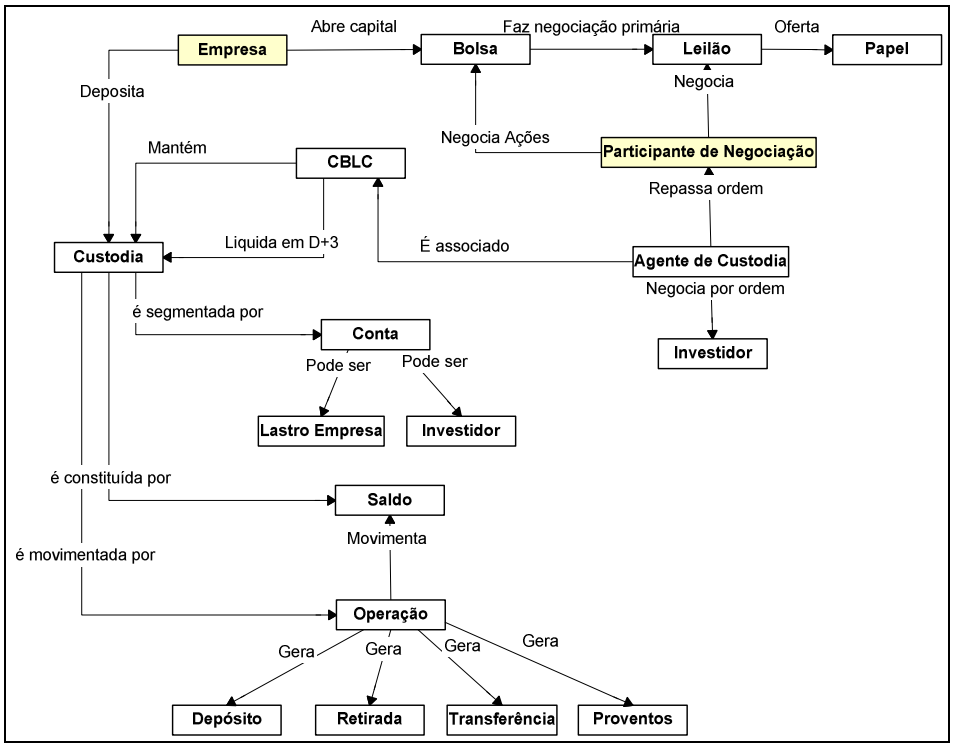

Figura 4.1 Negociação de ações na BOVESPA

4.1.1 Sistema de Custódia

A Companhia Brasileira de Liquidação e Custódia (CBLC) herdou os serviços de custódia que a Bolsa de Valores de São Paulo vinha prestando desde 1974. O Serviço de Custódia Fungível CBLC é um produto dinâmico e em constante expansão, desenvolvimento e aperfeiçoamento. Diversas medidas visando o aprimoramento dos serviços de custódia e a manutenção do alto nível de qualidade e confiabilidade sempre foram o objetivo da BOVESPA.

Além de atuar como depositária de ações de companhias abertas, a estrutura dos sistemas utilizados pelo Serviço de Custódia CBLC foi desenvolvida para prover o mesmo tipo de serviço para outros instrumentos financeiros como, por exemplo, certificados de privatização, debêntures, certificados de investimento, quotas de fundos imobiliários e títulos de renda fixa [BOVESPA, 2004].

4.1.2 Requisitos de Arquitetura

O requisito básico da área de tecnologia da empresa é o desenvolvimento do novo sistema utilizando a tecnologia Microsoft .NET e estes sistemas deverão estar distribuídos em camadas.

O sistema deverá atender tanto usuários CBLC como os usuários Agentes de Custódia externos a CBLC. Para atender funcionalidades específicas dos usuários externos, deverão ser disponibilizadas funcionalidades via Web Services, além de manter a estrutura de rede de serviços que é utilizado atualmente.

4.1.3 Requisitos Funcionais

Para auxiliar no entendimento dos requisitos, no próximo item são descritos alguns termos do Sistema de Custódia de Títulos. Outro requisito a ser descrito na seqüência são os casos de uso importantes para o entendimento do experimento, como o objetivo é apenas o entendimento, eles não serão detalhados.

4.1.3.1 Glossário

O artefato Glossário é parte da disciplina Gerência de Requisitos, descrito no Processo Unificado. Termos importantes do domínio do problema devem ser definidos de forma concisa e clara no Glossário. Isso evita ambigüidades e é extremamente útil quando a equipe de desenvolvimento não é fluente no domínio do problema. A seguir, apresenta-se o Glossário elaborado referente a alguns termos do sistema de custódia de títulos.

Ação - Título negociável que representa a menor parcela em que se divide o capital de uma sociedade anônima.

Acionista - Aquele que possui ações de uma sociedade anônima.

Bolsa de Valores - Associação civil sem fins lucrativos, cujos objetivos básicos são, entre outros, manter local ou sistema de negociação eletrônico, adequados à realização, entre seus membros, de transações de compra e venda de títulos e valores mobiliários; preservar elevados padrões éticos de negociação; e divulgar as operações executadas com rapidez, amplitude e detalhes.

Capital aberto (companhia de) - Empresa que tem suas ações registradas na Comissão de Valores Mobiliários – CVM e distribuídas entre um determinado número de acionistas, que podem ser negociados em bolsas de valores ou no mercado de balcão.

CBLC – Companhia Brasileira de Liquidação e Custódia - Sociedade anônima com capital fechado, com sede na capital do estado de São Paulo, que provê serviços de compensação, liquidação e controle de risco de operações. A CBLC também presta o Serviço de Custódia Fungível de ativos e administra o Banco de Títulos CBLC – BTC. É uma organização auto-reguladora, supervisionada pela Comissão de Valores Mobiliários – CVM.

Custódia infungível - Serviço de custódia no qual os valores mobiliários depositados são mantidos discriminadamente pelo depositante.

Sociedade Corretora - Instituição auxiliar do sistema financeiro que opera no mercado de capitais com títulos e valores mobiliários, em especial no mercado de ações. É a intermediária entre os investidores nas transações em bolsas de valores. Administra carteiras de ações, fundos mútuos e clubes de investimentos, entre outras atribuições.

4.1.4 Casos de Uso

A **Figura 4.2** tem por objetivo apresentar quais são os casos de uso utilizados no experimento. Neste caso, somente as funcionalidades de Depositar, Retirar e Transferir serão apresentadas.

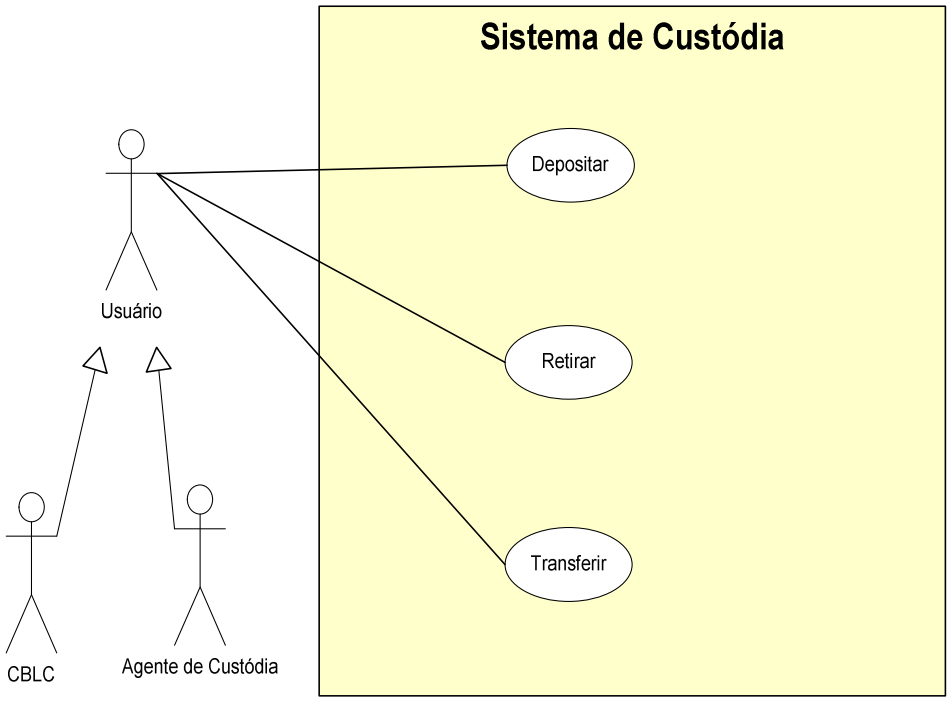

Figura 4.2 Modelo de caso de uso do experimento

4.1.5 Caso de Uso Depositar Ações

Este é um serviço disponibilizado pela BOVESPA para que os usuários possam depositar suas ações.

O processo de depósito compreende as seguintes etapas:

- Entrega das ações acompanhada da documentação que autoriza a transferência dos títulos para o nome da BOVESPA como proprietária fiduciária junto às Companhias Abertas.

- A BOVESPA recebe e confere a documentação entregue. Se os documentos estiverem corretos, a BOVESPA gera o crédito na conta de custódia do Usuário na quantidade de ações depositadas, e encaminha a Companhia Aberta o pedido de transferência das ações para o seu nome.

4.1.6 Caso de Uso Retirar Ações

A retirada de títulos da Custódia Bovespa constitui um serviço de natureza inversa ao depósito das ações. É a solicitação que o usuário faz de devolução de suas ações.

O processo de retirada das ações da Custódia Bovespa ocorre da seguinte forma:

- A Custódia Bovespa recebe o pedido de retirada e, então, encaminha às companhias emissoras solicitação para transferência das ações do seu nome para o nome do proprietário.

- Ao mesmo tempo, a Custódia Bovespa efetua o débito da quantidade de ações devolvidas na conta do Usuário.

- Efetuada a transferência, cessa a representação fiduciária da BOVESPA e as ações são remetidas ao Usuário que solicitou a retirada.

4.1.7 Caso de uso Transferir ações

A transferência de ações usualmente é motivada por dois fatores: a) realização de operações de compra e venda; b) utilização das ações como garantia de operações realizadas nos Mercados a Termo e de Opções.

As transferências são realizadas da seguinte forma:

- A BOVESPA recebe a solicitação de transferência e processa o pedido, efetuando o débito e respectivo crédito nas contas de custódia dos usuários envolvidos.

- É importante destacar que:

  a) A movimentação das ações é realizada somente por meio de registro contábil, não há necessidade de transferência de titularidade junto à companhia aberta;

  b) Todo o processo é realizado de forma automatizada por meio de terminais de computador instalados nos escritórios dos usuários.

4.2 Aplicação do Método

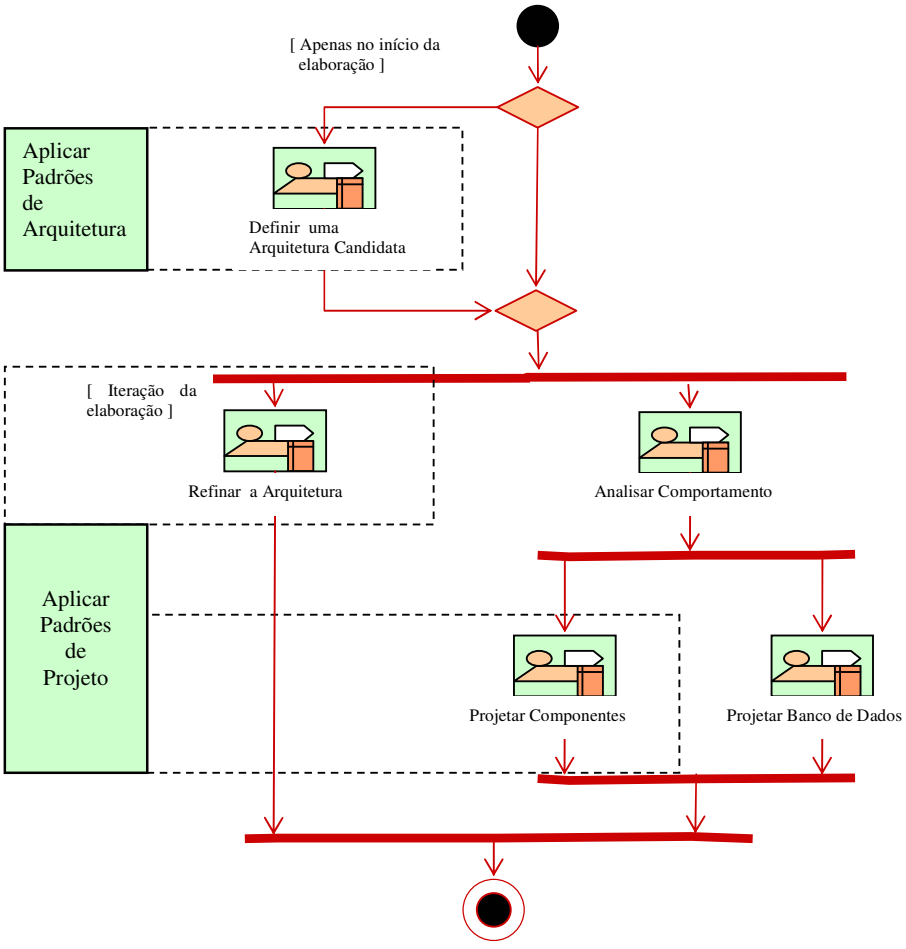

Figura 4.3 *Workflow* da disciplina de Análise e Projeto

Com o objetivo de ilustrar a visão operacional do método, a **Figura 4.3** é novamente apresentada neste ponto. As linhas tracejadas indicam exatamente em quais atividades do Processo Unificadas as atividades de aplicar padrões se enquadram

Nos itens a seguir será utilizado o Método de Aplicação de Padrões no Processo Unificado, onde o objetivo é o experimento do método visando sua comprovação prática.

4.2.1 Aplicação de padrões de Arquitetura

Nesta etapa são aplicados os passos da atividade de aplicações de padrões de arquitetura, que são: Analisar requisitos do sistema, Definir arquitetura candidata e Aplicar padrão de arquitetura. A entrada da disciplina é representada pelos requisitos do sistemas, onde após a análise dos requisitos é gerado como saída o modelo de arquitetura.

Requisitos do sistema (arquitetura) - O projeto de Custódia de Títulos, apresenta características que proporcionam a exposição de funcionalidades e serviços a clientes externos a instituição. Estas funcionalidades e serviços são expostos parte pela rede de serviços corporativa e parte via *Web Service*.

Análise dos requisitos – Na análise dos requisistos, optou-se por uma solução de arquitetura que é a Aplicação de Serviços em três camadas, podendo neste caso ser aplicado o padrão *Three-Layered Services Aplication* [PnP, 2003].

Definição da arquitetura – Após a aplicação do padrão tem-se uma opção de arquitetura em três camadas que atende aos requisitos.

Aplicação do Padrão de arquitetura – Como pode ser visto na **Figura 4.4,** a camada de interface de usuário consiste de *Web forms* e controles *Web* de cliente (*Pagelets*). A camada de negócio consiste num *Web Service*, que também é um componente disponibilizado, e dos componentes de negócio. A camada de dados consiste no ADO.NET e banco de dados SQL Server [Gordan, 2002] [O'Kelly, 2002].

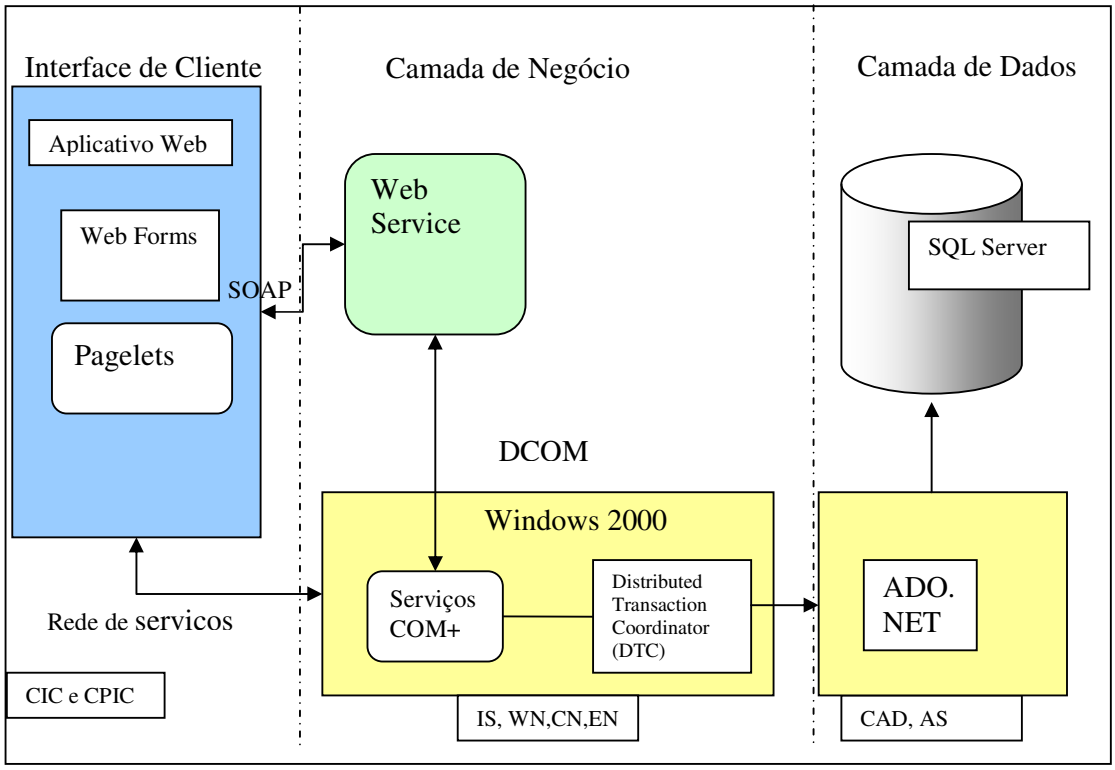

Figura 4.4 Arquitetura três camadas proposta para o sistema de Custódia

Ainda na **Figura 4.4**, o padrão *Three-Layered Services Aplication* é aplicado distribuído da seguinte forma: na camada de Interface, são aplicados Componentes de Interface de Cliente (CIC) e Componentes de Processo de Interface de Cliente (CPIC). Na camada de negócio, são aplicados as Interfaces de Serviço (IS) e Entidades de Negócio (EN). Por último, na camada de dados são aplicadas Componentes de Acesso e Dados (CAD) e Agentes de Serviços (AS).

Com isso, o sistema de custódia disponibilizará os serviços essenciais como Depósito, Retirada e transferência na rede de serviços corporativa, uma vez que os participantes de negociação são membros da Bovespa. Outros serviços são disponibilizados via Web Service, como informações sobre as ações das empresas que abriram seu capital e negociaram na Bolsa de Valores. Estas empresas necessitam acompanhar como estão suas ações, e saber em mãos de quem estão estas ações. Naturalmente, são informações estratégicas e desta forma fica disponível apenas para empresas participantes ter acesso a suas informações usando esta tecnologia.

4.2.2 Aplicação de padrões de projeto

Neste item será aplicada a disciplina de aplicação de padrões de projeto, as atividades desta disciplina são apresentadas da seguinte forma: Elaborar o modelo de Classes de análise, Elaborar modelo de classes de projetos, Identificar potenciais padrões, Analisar potenciais padrões e Aplicar Padrão. A atividade de aplicar classes .NET não será experimentada, uma vez que o foco deste experimento é a aplicação dos padrões.

4.2.2.1 Elaborar modelo de classes

Esta primeira atividade tem como função analisar os casos de uso descritos como Requisitos funcionais. Como resultado desta análise é elaborado o Modelo de Classes de Análise, visto na **Figura 4.5**.

Na seqüência pode ser efetuado um refinamento do modelo de classes de análise, este refinamento faz parte da atividade de elaborar o modelo de classe de projeto e traz como resultado um modelo de classes de projeto, visto na **Figura 4.6.**

Por ser o foco deste experimento validar a praticidade do método, muitos detalhes da análise são omitidos. São apresentados apenas os modelos de classes de análise e projeto, pois é a partir deste ponto que se inicia a aplicação de padrões de projeto.

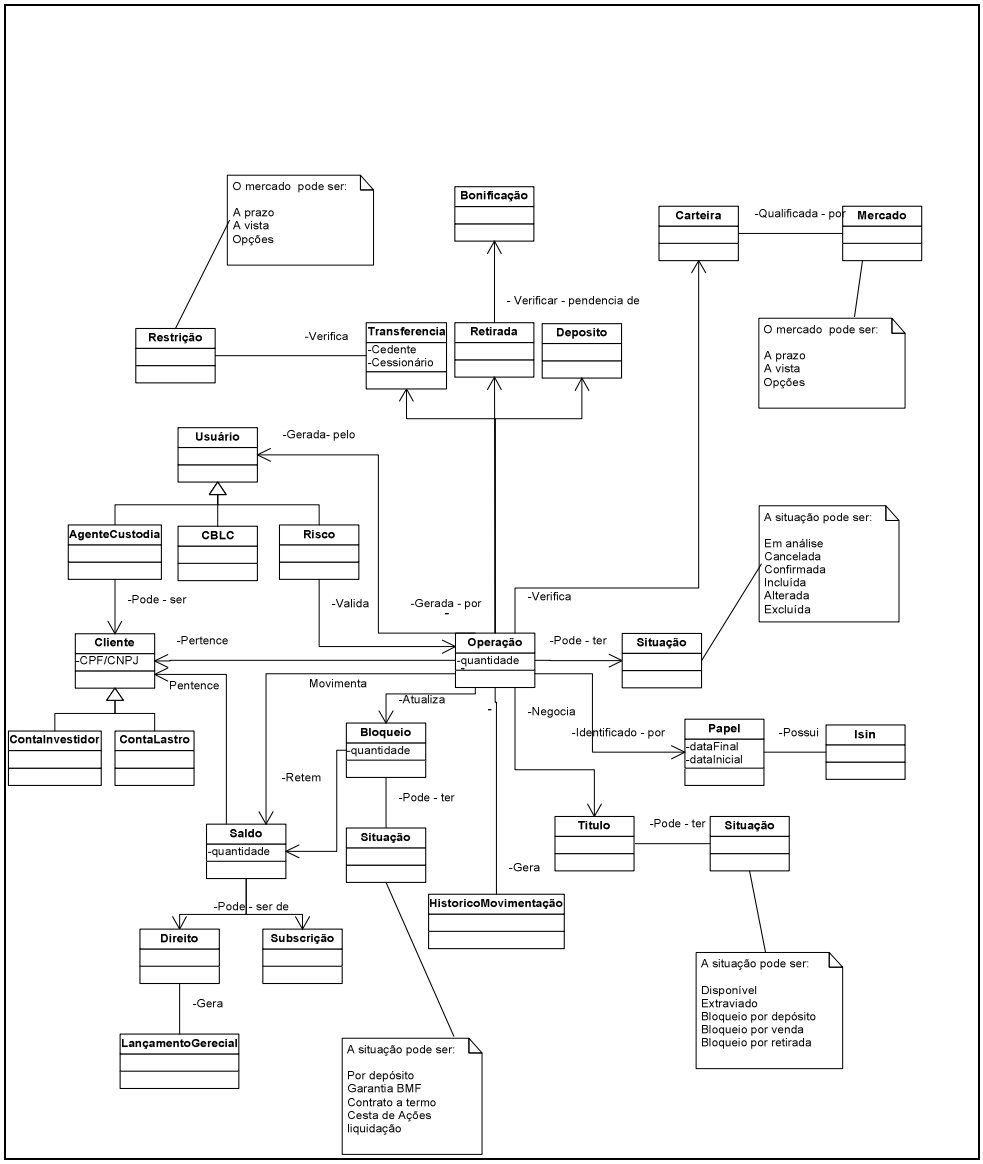

Figura 4.5 Modelo de classes de análise

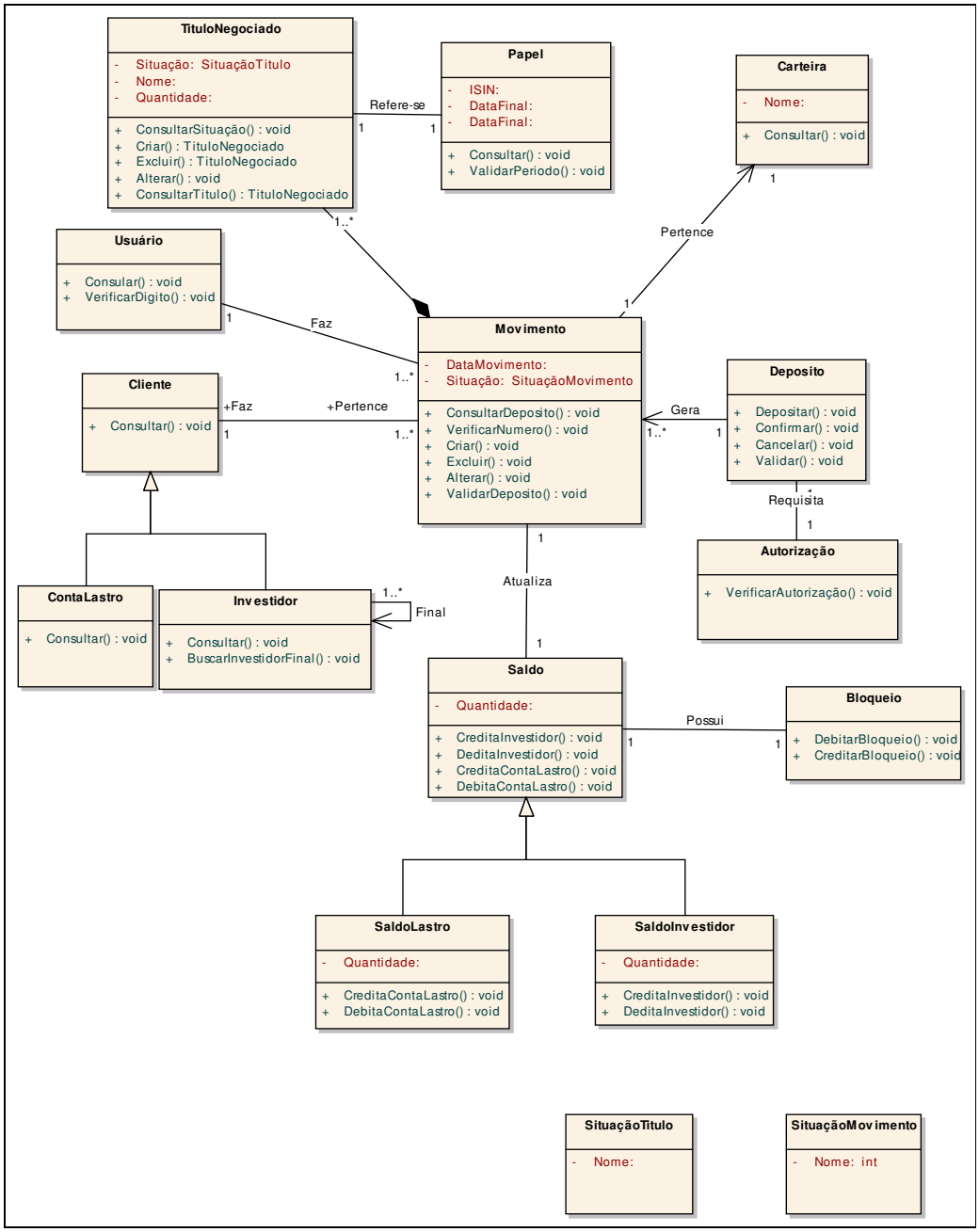

Figura 4.6 Modelo de classes de projeto

4.2.2.2 Identificar potenciais padrões

Avalia-se primeiramente um candidato a padrão, pela análise da estrutura das classes. No modelo de classes de projeto, visto na **Figura 4.6**, são identificados dois candidatos a padrão. Foi identificado um *Composite* na relação entre as classes

Movimento, TituloNegociado e um *Façade* na relação entre as classes Depósito e Autorização. Estas identificações são apresentadas nos próximos itens.

4.2.2.2.1 Identificando um *Façade* como candidato a padrão

Em uma primeira análise no modelo de classes de projeto na busca por padrões, a associação entre as classes de Depósito e Autorização indica um possível padrão. Esta possibilidade é indicada pela observação das estruturas das classes, sendo utilizado a **Tabela 4.1** para auxiliar na decisão, esta avaliação aponta estas classes como de alto grau de acoplamento.

Indicadores de estrutura de classes	É aplicável
A classe está na linha final de comunicação de um modelo de classes ? (Se estiver, ela pode executar algumas regras na mediação de interações entre outras classes.)	[ ] Sim [ ] Não
A classe recebe mensagens de muitas classes ? (Se recebe pode agir como interface para outras classes)	[ ] Sim [ ] Não
Classe com herança paralela ? (Ou seja, cada subclasse de uma classe é relacionada com subclasse de outras classes ?) (Se forem são propensas a trabalharem muito próximas)	[ ] Sim [ ] Não
A classe possui alto grau de acoplamento ? (Se for pode indicar uma relação de comunicação entre dois sistemas)	[x] Sim [ ] Não
A classe possui hierarquia todo-parte ?	[ ] Sim [ ] Não

Tabela 4.1 Indicador de estrutura de classes - *Façade*

A **Figura 4.7** ilustra que a classe Depósito tem responsabilidade no contexto do sistema de custódia, e a classe Autorização tem responsabilidades que se estendem além do sistema de custódia.

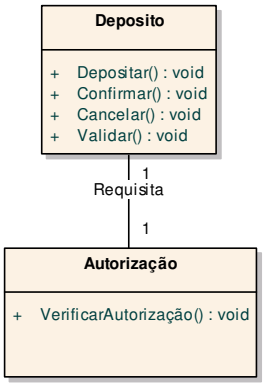

Figura 4.7 Padrão *Façade* identificado

Neste caso, a classe Autorização toma a forma de um subsistema que pode ser requisitado não apenas pela funcionalidade de depósito, mas também por outras funcionalidades do sistema de custódia e também por outros sistemas. Com isso, o padrão *façade* torna-se um forte candidato a padrão.

4.2.2.2.2 Identificando um *Composite* como candidato a padrão

Outro candidato a padrão identificado no projeto é o padrão *Composite*. A identificação se dá na avaliação das estruturas deste padrão, segundo [Gamma, 1995], se aplica a hierarquias de parte-todo de objetos.

No modelo de classes de projeto, as classes TituloNegociado e Movimento possuem uma relação de composição, em que o Movimento é o todo e TituloNegociado é parte. A **Figura 4.8** apresenta apenas as duas classes que relacionadas podem ser identificadas como possível padrão.

A **Tabela 4.2** é aplicada neste ponto para a avaliação da estrutura destas classes. Como resultado desta avaliação é verificado que a classe possui hierarquia todo-parte, sendo assim estas classes são identificadas como sendo um candidato a padrão.

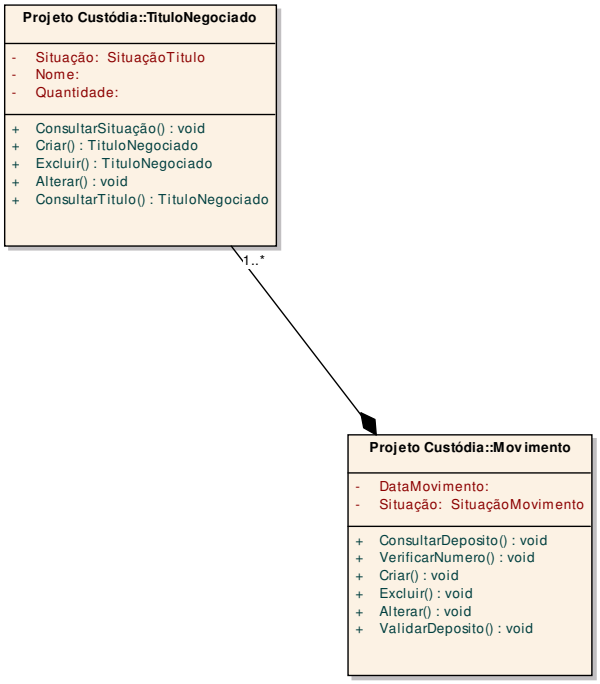

Figura 4.8 Possível *Composite* identificado

Indicadores de estrutura das classes	É aplicável
A classe está na linha final de comunicação de um modelo de classes ? (Se estiver, ela pode executar algumas regras na mediação de interações entre outras classes.)	[ ] Sim [ ] Não
A classe recebe mensagens de muitas classes ? (Se recebe pode agir como interface para outras classes)	[ ] Sim [ ] Não
Classe com herança paralela ? (Ou seja, cada subclasse de uma classe é relacionada com subclasse de outras classes ?) (Se forem são propensas a trabalharem muito próximas)	[ ] Sim [ ] Não
A classe possui alto grau de acoplamento ? (Se for pode indicar uma relação de comunicação entre dois sistemas)	[ ] Sim [ ] Não
A classe possui hierarquia todo-parte ?	[x] Sim [ ] Não

Tabela 4.2 Indicador de estrutura de classes – Composite

4.2.2.3 Analisar potenciais padrões

Nos itens seguintes são analisados os dois padrões, *Façade* e o *Composite,* identificados como potenciais padrões. Esta análise, conforme descrito no método é realizada pela avaliação do propósito e da implementação dos padrões.

4.2.2.3.1 Analisando um *Façade* candidato a padrão

Deve-se começar pela avaliação da implementação, em que deve ser verificado se este padrão poderá ser usado em outros projetos.

Em segundo lugar, avalia-se o propósito, em que deve ser avaliado a que o padrão se propõe a solucionar e quais efeitos ele pode ter no sistema.

Implementação	Propósito
Poderá ser utilizado por outros sistemas ou sub-sistemas, uma vez que, autorização pode ser considerado um serviço global na Instituição.	Fornecer uma interface unificada para um conjunto de interfaces em um sub-sistema. *Façade* define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado.

Tabela 4.3 Análise do padrão *Façade*

A **Tabela 4.3** apresenta o resultado da analise, em que foi avaliada a possível implementação do padrão em futuros projetos, de onde se conclui que é um padrão que pode ser implementado.

Com relação ao propósito, verificou-se que o padrão analisado corresponde ao padrão referência *Façade* encontrado em [Gamma, 1995].

Além da intenção do padrão, descrito no propósito encontrado na **Tabela 4.3**, este padrão tem como motivação estruturar um sistema em subsistemas com o intuito de reduzir complexidade, minimizar a comunicação e as dependências entre subsistemas.

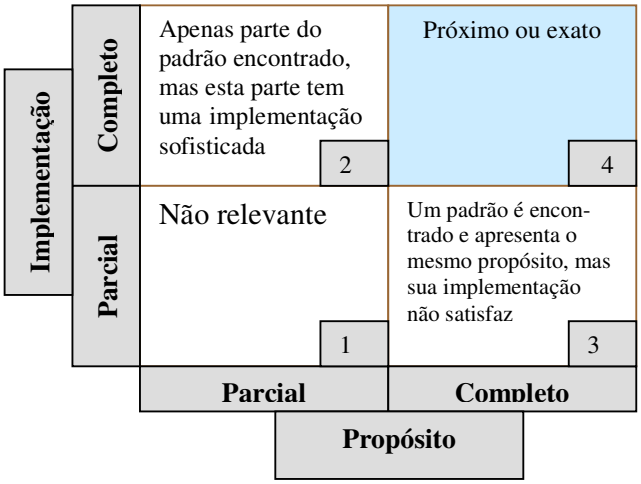

Tabela 4.4 Tabela de verificação de conformidade na análise do *Façade*

A **Tabela 4.4** tem como objetivo demonstrar que o *Façade* correspondeu com a comparação do padrão proposto e o padrão referência. Na análise e nas comparações, o padrão alcançou a escala de 4 (próximo ou exato) na aplicação da tabela. Sendo assim, o padrão poderá ser aplicado na classe Autorização.

4.2.2.3.2 Aplicando o padrão *Façade*

A **Figura 4.9** apresenta de forma simplificada como poderia ser aplicado o padrão *Façade*. O objetivo não é dar uma solução de como autorizar, mas sim dar um exemplo de quando pode ser utilizado o *Façade*. A solução é apresentada na criação de uma classe FaçadeAutorização que será a interface para outras classes.

O próximo passo é empacotamento das classes, como demonstra a **Figura 4.10**, tendo visibilidade apenas a classe *Façade*. Com isso, a classe de Depósito poderá requisitar a autorização chamando a classe FaçadeAutorização, utilizando as operações com visibilidade, e o FaçadeAutorização se encarregara de chamar a classe responsável pela operação.

Nesta mesma **Figura 4.10**, a classe depósito dividiu-se em três camadas, onde DepósitoInterface tem como função ser a classe que será a primeira a ser chamada servindo de interface a chamadas externas ao sistema (Um exemplo seria telas web), na seqüência a classe DepósitoInterface passa a responsabilidade para a classe de DepósitoNegócio que é a classe detentora das regras do negócio, executa

suas funções e por fim chama a classe de DepositoDados que é a interface com os serviços de banco de dados e outros.

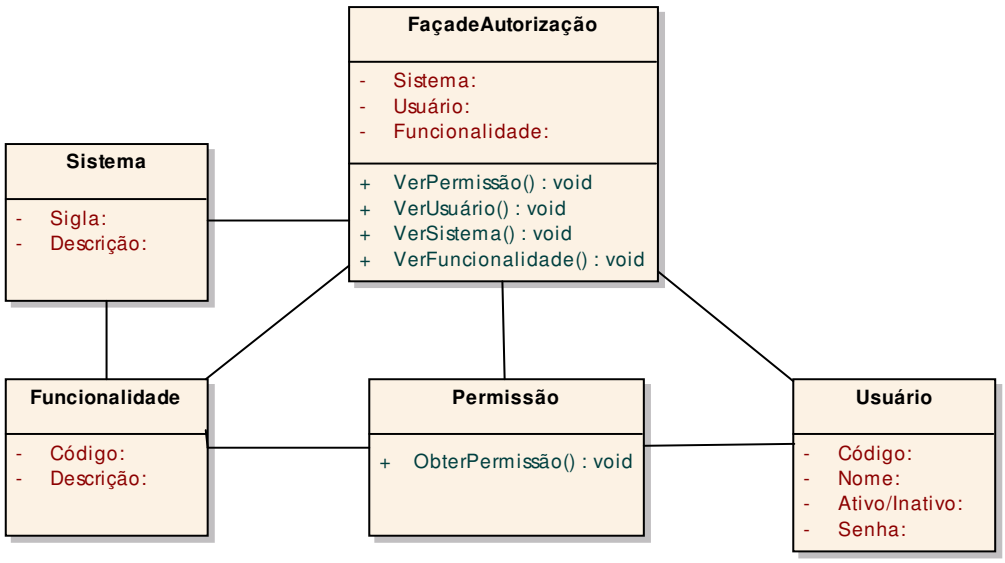

Figura 4.9 Modelo de projeto do *Façade*

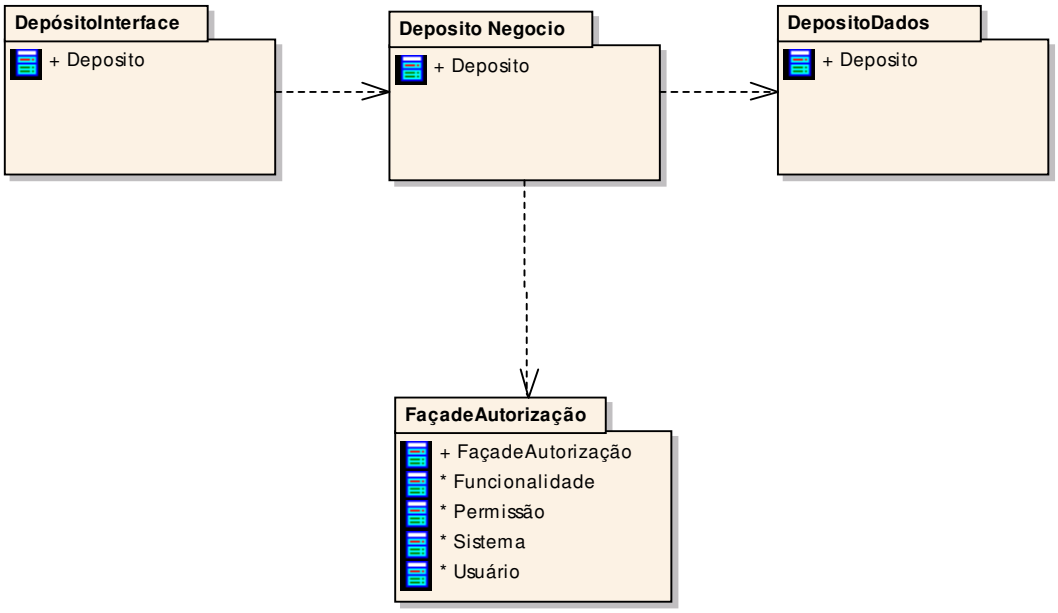

Figura 4.10 Empacotamento do FaçadeAutorização

A representação do método termina com este modelo, mas no projeto os próximos passos podem ser o detalhamento da especificação dos métodos, onde pode ser utilizado o português estruturado ou podem ser utilizadas ferramentas case, já codificando em linguagens de programação próxima da implementação.

4.2.2.3.3 Analisando um *Composite* candidato a padrão

A analise deste padrão, não é tão simples como a análise do *Façade.* É necessário então, tendo o método como guia, verificar se as classes Movimento e sua composição TituloNegociado podem ser configuradas como *Composite*.

Usando o mesmo critério adotado no item anterior, quando da avaliação do *Façade*, será feita uma avaliação quanto a implementação e o propósito do padrão.

Na avaliação dos aspectos da implementação [Gamma, 1995] estes são descritos como segue:

a) Referências explícitas aos pais: manter referências dos componentes filhos para seus pais pode simplificar o percurso e a administração de uma estrutura composta.

b) Compartilhamento de componente: é útil compartilhar componentes, mesmo que seja apenas para atender aos requisitos de espaço de armazenamento.

c) Maximização da interface de componente: um dos objetivos do padrão *Composite* é tornar os clientes desconhecidos das classes específicas *Leaf* ou *Composite* que estão usando. Para atingir este objetivo, a classe *Component* deve definir tantas operações comuns quanto possíveis para as classes *Composite e Leaf.*

O resultado da avaliação, apresentado na **Tabela 4.5**, constata que a solução apresentada para as classes endereça os mesmos problemas dos encontrados no padrão *Composite.*

Implementação	Propósito
Referência explícita aos pais. Compartilhamento de componentes. Maximização da interface de componente.	Representa uma hierarquia todo-parte. Todos os objetos na estrutura composta são tratados de maneira uniforme.

Tabela 4.5 Avaliação do *Composite*

Após as avaliações dos propósitos e das implementações, a **Tabela 4.6** dá a classificação do padrão de acordo com as medidas de proximidade. Como a avaliação dá uma classificação de 4 (próximo ou exato) na proximidade do padrão *Composite*, este então será aplicado no projeto devido a este resultado.

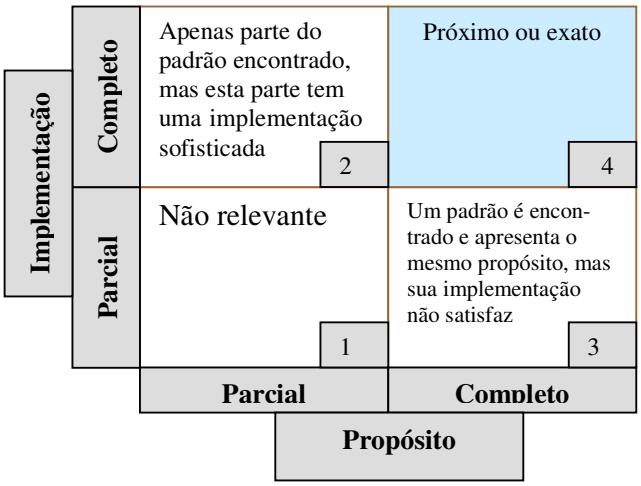

Tabela 4.6 Tabela de verificação de conformidade do padrão C*omposite*

4.2.2.3.4 Aplicando o padrão *Composite*

O próximo passo é aplicar o padrão nas classes avaliadas. Tendo como meta maximizar as associações entre as classes por meio de uma melhor avaliação das operações identificadas inicialmente. Após a incorporação do padrão nas classes avaliadas, temos o novo modelo de classes de projeto ilustrado pela **Figura 4.11**.

As classes do modelo possuem as seguintes responsabilidades no contexto do *Composite*:

Imovimento (*Component*)

- Declara a interface para os objetos na composição;
- Implementa comportamento para a interface, comum a todas as classes, conforme apropriado;

Declara uma interface para acessar e gerenciar os seus componentes-filhos.

TituloNegociado (Leaf)

- Representa objeto título de uma operação. Um título não tem filhos;
- Define comportamento para objetos primitivos (sem filhos) na composição.

Movimento (C*omposite*)

- Define comportamento para componentes que têm filhos;
- Armazena os componentes-filho;
- Implementa as operações relacionadas com os filhos presentes na interface de Imovimento.

Depósito (*Client*)

- Manipula objetos na composição por meio da interface de Imovimento.

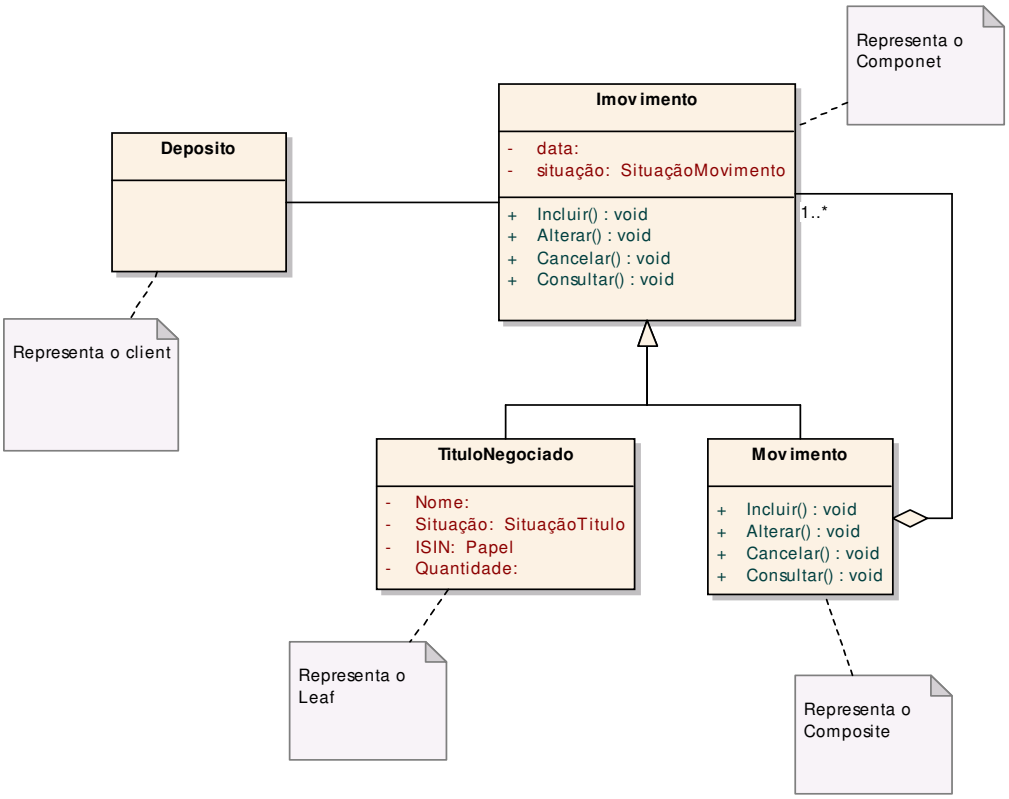

Figura 4.11 Modelo de classes do projeto no padrão *Composite*

Concluindo, o depósito usa a classe Imovimento como interface para interagir com os objetos da estrutura composta. Se o receptor é um título, então a solicitação é tratada diretamente. Se o receptor é um Movimento, então ele normalmente repassa as solicitações para os seus componentes-filhos, executando as operações adicionais antes e/ou depois do repasse.

4.2.2.3.5 Empacotando e aplicando C*omposite* na arquitetura

Este é o passo que o modelo de classes é empacotado e como pode ser visto na **Figura 4.12,** são distribuídas entre as camadas, no caso do depósito por representar o cliente, fará parte da camada de interface, já no centro do modelo o pacote de CompositeMovimento representa o negócio. Faz parte do pacote de acesso a dados classes que possuem a função de atender a solicitações da camada de negócio mediando o acesso ao banco de dados.

O exemplo, aqui citado foi apenas de parte do modelo de classes do projeto, no entanto na prática todo o modelo deve ser avaliado e juntamente com componentes de serviços serem distribuídos nas camadas da arquitetura do sistema.

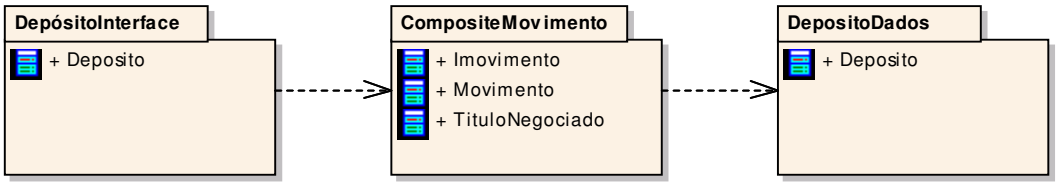

Figura 4.12 Empacotamento do Composite Movimento

4.3 Conclusão

Este capítulo teve por finalidade a comprovação do método em um experimento na qual foi utilizado o sistema de custódia de títulos da BOVESPA. Nele pode-se ver a aplicação de um padrão de arquitetura e dois padrões de projetos, na aplicação de ambos seguiram-se os procedimentos pertinentes ao método, tendo com isso comprovado a sua viabilidade.

5 CONSIDERAÇÕES FINAIS

Neste capítulo, no item 5.1, apresentam-se as principais conclusões desta dissertação, em que são analisados os pontos importantes deste trabalho, contribuindo, assim, para um entendimento melhor dos assuntos apresentados nos capítulos anteriores.

E no item 5.2, são expostas algumas sugestões para futuros trabalhos, tendo como ponto de partida esta dissertação, para que outros pesquisadores possam dar novas soluções utilizando o que aqui foi proposto.

5.1 Conclusões

As considerações desta dissertação são definidas da seguinte forma: aplicabilidade dos padrões, a adequação do método no Processo Unificado, a independência de tecnologia, a aplicabilidade dos padrões no âmbito das tecnologias e os papéis mais importantes na aplicação do método.

- Aplicabilidade dos padrões: A aplicação dos padrões em projetos definitivamente não é uma tarefa fácil, e o método de aplicação de padrões mostrou-se adequado no auxílio desta atividade. As suas diretrizes facilitam o projetista na dura tarefa de aplicar um padrão. Também os padrões de arquitetura não são fáceis de serem aplicados, porém o método mostrou-se adequado no auxílio a esta atividade, pois conduz o arquiteto na direção da utilização de padrões de arquitetura utilizados e catalogados por autores conhecidos.

- Adequação do método no Processo Unificado: A adequação do método no Processo Unificado transcorreu de forma natural. O fato de o método ter sido constituído por disciplinas, atividades e responsáveis tornou fácil sua adequação ao Processo Unificado, uma vez que o Processo Unificado também apresenta as mesmas características apresentadas pelo método. Tanto que, Kruchten em [Kruchten, 2000] recomenda que sejam utilizados padrões

de projetos e arquitetura em projetos no desenvolvimento de software em um ambiente que o processo é o Processo Unificado.

- A independência de tecnologia: No tocante à aplicação do método no âmbito tecnológico, a conclusão a que se chega é que existe uma independência do método nesta questão. Verificou-se que o mesmo é aplicável tanto à tecnologia .NET como à tecnologia J2EE.

- Aplicabilidade dos padrões no âmbito das tecnologias: A aplicabilidade dos padrões na relação com as tecnologias é dada pela relação de hierarquia que as classes dos projetos podem exercer com as bibliotecas de classes das plataformas tecnológicas, ou seja, um padrão identificado em um modelo de projeto pode fazer uso das classes bases das plataformas tecnológicas.

- Os papéis mais importantes na aplicação do método: Além das diretrizes apresentadas no método que auxiliam a análise e a identificação dos padrões, este trabalho enfatizou alguns papéis importantes em um projeto: O projetista / analista: são deles a responsabilidade de aplicar padrões em um modelo de projeto. Estes utilizam a disciplina de aplicação de padrões de projeto para melhor aplicar os padrões de referência. O gerente de projetos, que tem como função coordenar as atividades de escopo e iterações do projeto, além, é claro, de administrar os recursos humanos envolvidos no projeto, este utiliza a perspectiva gerencial do método para ter a visão da colocação da perspectiva operacional do método no Processo Unificado.

5.2 Futuros trabalhos

Nesta dissertação, o autor procurou apresentar soluções que auxiliassem arquitetos, projetistas e analistas a elaborarem modelos de arquitetura de sistemas e modelos de projetos baseados em modelos já utilizados conhecidos como padrões. Para isso, uniram-se alguns conceitos, aliados a própria experiência do autor em projetos para criar o método de aplicação de padrões no Processo Unificado.

Uma sugestão para novos trabalhos que utilize o método e a experiência apresentada nesta dissertação, é a utilização do método de aplicação de padrões em um outro processo que não seja o Processo Unificado.

Mas, certamente, uma aplicação mais aprofundada deste método poderia contribuir com a consagração do mesmo e resultar na sua melhoria. Isto pode ser feito tanto no meio acadêmico como no empresarial.

REFERÊNCIAS

[Alencar, 1995a] ALENCAR, P.,S.,C. *et al*. **A formal approach to architectural design patterns.** Dept. of Computer Science, University of Wateloo, Waterloo, Ont, Canadá. CS-95-38. IEEE, aug.,1995.

[Alencar, 1995b] ALENCAR, P.,S.,C. *et al*. **A formal approach to design patterns definition & application.** Dept. of Computer Science, University of Wateloo, Waterloo, Ont, Canadá. CS-95-34. IEEE, aug.,1995.

[Alencar, 1996] ALENCAR, P.,S.,C. *et al*. **A formal architectural design patterns-based approach to software understanding.** Dept. of Computer Science, University of Wateloo, Waterloo, Ont, Canadá N2L 3G1. IEEE, aug.,1996.

[Buschmann, 1996] BUSCHMANN, F. *et al*. **Pattern - Oriented Software Architecture**, John Wiley & Sons ISBN 0-471-95869-7, 1996.

[Copper, 2002] Cooper, J., W. **Introduction to design patterns in C#**. IBM T J Watson Research Center. February, 2002.

[Fowler, 2000] Fowler, M.; **UML Distilled Second Edition – A Brief Guide to the Standard Object Modeling Language**. Addison-Wesley - 2000.

[Fowler, 2003] Fowler, M.; **Design patterns.** IEEE *Software*. *Published by the* IEEE *Computer Society*. 0740-7469/03 - 2003.

[Gamma, 1995] GAMMA, E. *et al*. **Padrões de projetos**: soluções reutilizáveis de software orientado a objetos. trad. Luis A. Meirelles Salgado. Porto Alegre: Bookman, 2000.

[Gordan, 2002] Gordan, Alan. **Integre as tecnologias .Net.** .NET Magazine, A.1, E. 1, P. 50-58. fevereiro 2002.

[Kruchten, 2000] Kruchten, P.; **The Rational Unified Process An Introdution.** Addison-Wesley, 2000.

[Kruchten, 2001] Kruchten, P.; **What is the Rational Unified Process ?.** Rational Software Corporation White paper. www.rational.com.

[Larman, 1999] Larman, G. **Utilizando UML e Padrões - Uma introdução à análise e ao projeto orientados a objetos** - trad. Luis A. Meirelles Salgado. Porto Alegre: Bookman, 2000.

[Larman, 2001] Larman, G. **Applying UML and Patterns - An introduction to object-oriented analysis an design and the Unified Process.** 2001.

[Maioriello, 2002] Maioriello, J. **Why Use Design Padrões for Bulding .NET Applications ?.** White paper, [online] disponível em: http://www.developer.com/design/article.php/1474561 [Arquivo capturado em 11/11/2003].

[Mehl, *s.d*] MEHL, O. *et al***. A management-aware software development process using design patterns.** Cooperation & Manegement Institute of Telematics, University of Karlsruhe Zirkel 2, D-76128 Karlsruhe, Germany. IEEE.

[Microsoft, 2001] MICROSOFT .NET Framework SDK. Microsoft Corporation [cd-rom] – 2001.

[Muraki, 2002] MURAKI, T.; SAEKI, M.**. Metrics for applying GOF design patterns in refactoring processes.** Dept. of Computer Science, Tókio Institute of Technology. Ookayama 2-12-1, Meguro-Ku, Tókio 152-85552, Japan. ACM, 2002.

[O'Kelly, 2002] O'Kelly, Peter. **Comparação entre o J2EE e o .NET**. .NET Magazine, A.1, E. 6, P. 24-29. julho 2002.

[PnP, 2002] *patterns & practices*, Microsoft Corporation. **Application Architecture for .NET: Designing Applications and Services.** *MSDN Library*. Available at: *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/distapp.asp*.

[PnP, 2003] *patterns & practices*, Microsoft Corporation. **Enterprise Solution Patterns Using Microsof .Net.** *MSDN Library*. Available at: *http://msdn.microsoft.com/library/en-us/dnpatterns/html/esp.asp*.

[Saeki, 2001] SAEKI, M. **Attributed Methods: Embedding quantification techniques to development methods.** Dept. of Computer Science, Tókio Institute of

Technology. Ookayama 2-12-1, Meguro-Ku, Tókio 152-85552, Japan. ACM, 2001.

[Shull, 1996] FORREST, S.; MELO, W.; BASILI, V. – **An inductive method for discovering design patterns from object-orinted software systems.** Computer Science Department, Institute for Advanced Computer Studies. University of Maryland, College Park, MD, 20742. USA 1996.

REFERÊNCIAS CONSULTADAS

[Biscoe, 2002] Biscoe, S. T.; **Business Modelling for Componente systems with UML.** Proceedings of the Sixth International Enterprise distributed object computing conference (EDOC'02). 0-7695-1858-4/02 – 2002 - IEEE.

[Cinneide, 1998] Cinneide, O, M.; Nixon, P,; **A methodology for the automated introduction of design patterns.** Department of Computer Science, University College Dublin, Ireland. IEEE 1998.

[Cinneide, 2002] Cinneide, O, M.; Nixon, P,; **Automated software evolution towards design patterns.** Department of Computer Science, University College Dublin, Ireland. 2003 ACM.

[Coad, 1992] Coad, P. ; **Object-Oriented Design Patterns**. Communications of the ACM, 35(2): 152-159. Sep. 1992.

[Fernandes, 2002] Fernandes, M., S.; **Especialização da disciplina análise e projeto do Rational Unified Process (RUP) para a arquitetura Java 2 platform, enterprise edition (j2ee).** Escola Politécnica da USP. Dissertação de mestrado em engenharia de software. 2002.

[Herzum, 2000] Herzum, Peter e Sims, Oliver. **Business Component Factory**. John Wiley and Sons, Inc., 2002.

[Kroll, 2001] Kroll, P.; **The spirit of the RUP;** Rational Software Corporation - 2001.

[Kuntzmann, *s.d.*a] Kuntzmann, A.; *at. al.*; **Rational Unified Process – an enabler for high process maturity.** Rational unified partner program. Q-Labs. www.rational.com.

[Machado, 2001] Machado, C., A., F., *at. al.*; **Aderência do RUP a norma NBR ISO/IEC 12207.** Pontifícia Universidade Católica do Paraná. Curitiba – Paraná – Brasil. 2001.

[Manzoni, 2003] Manzoni, V., M.; Price, T., R,; **Identifying extensions required by RUP to comply with CMM levels 2 and 3.** IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL 29, NO.2, February 2003. 0098-5589/03 - 2003 IEEE.

[Noble, *s.d.*] Noble, J.; **Patterns for finding objects within designs.** Microsoft Research Institute, Macquarie University, Sydney, Australia. IEEE.

[Oberg, 2000] Oberg R., *et. al.*; **Applying requirements manegement with use cases.** Rational Software Corporation. Technical paper TP505 (version 1.4). www.rational.com – 2000.

[Pollice, *s.d.*] Pollice, G.; **Using the Rational Unified Process for small projects: Expanding Upon eXtreme programming.** Rational Software Corporation White paper. www.rational.com.

[Pressman, 1995] Pressman, R., S. **Engenharia de Software** - trad. José Carlos Barbosa dos Santos. São Paulo: Makron Books, 1995.

[Priestley, 2000] Priestley, M.; Hunter, M,; **A Unified Process for software and documentation development.** IBM Toronto Lab, Toronto. 0-7803-6431-7/00 - 2000 IEEE.

[Probasco, *s.d.*] Probasco, L.; **The ten Essentials of RUP the essence of an effective development process.** Rational Software Corporation White paper. www.rational.com.

[Rational, 2002] Rational. **Rational Unified Process builder;.** Rational Software Corporation. Process manager's guide; version: 2002.05.00; part number: 800-025091-000 . www.rational.com – 2002.

[Rational, *s.d.* a] Rational. **Reaching CMM levels 2 and 3 with the Rational Unified Process;.** Rational Software Corporation. Rational Software White paper. www.rational.com.

[Rational, *s.d.* b] Rational. **Assessing the Rational Unified Process against ISO/IEC 15504-5: Information technology – software process assessment part 5: An assessment model and indicator guidance.** Rational Software Corporation. www.rational.com.

[Rational, *s.d.* c] Rational Unified Process. **Best practices for software develepment teams.** Rational Software Corporation White paper. www.rational.com.

[Rumbaugh, 1991] RUMBAUGH, J. *et al***. Object-Oriented Modeling an Design.** Prentice Hall, Englewood Cliffs, NJ, 1991.

[Smith, *s.d.*] Smith, J.; **The estimation of effort based on use cases;.** Rational Software Corporation. Rational software White paper. www.rational.com.

[Spence, 1998] Spence, I.; **Traceability strategies for managing requirements with use cases.** Rational Software Corporation White paper. www.rational.com.

ANEXO

ANEXO A - Produtos Microsoft distribuídos em camadas

Não é objetivo desta dissertação detalhar a arquitetura ou a infra-estrutura necessária para desenvolver software utilizando os produtos Microsoft .Net, mas sim demonstrar uma relação entre os produtos e a arquitetura em camadas. Por isso, este anexo dá uma visão simplificada dos produtos distribuídos em três camadas [O'Kelly, 2002] e [PnP, 2002].