PAULA VILLENA REDONDO

APLICANDO SOLUÇÕES COM PADRÕES DE PROJETO EM UM PORTAL DE VENDAS DE CONTEÚDOS PARA DISPOSITIVOS MÓVEIS

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para conclusão do Curso de MBA em Engenharia de Software

Área de concentração:
Engenharia de Software

Orientador:
Prof. Dr. Paulo Sergio Muniz Silva

São Paulo
2007

FICHA CATALOGRÁFICA

Redondo, Paula Villena
Aplicando soluções com padrões de projetos em um portal de vendas de conteúdos para dispositivos móveis / P.V. Redondo. -- São Paulo, 2008. 7
90 p.

Monografia (MBA em Tecnologia de Software) – Escola Politécnica da Universidade de São Paulo. Programa de Educação Continuada em Engenharia.

1. Padrões de software 2.Software orientado a objetos 3.Software (projeto, sistemas) Reúso I.Universidade de São Paulo. Escola Politécnica. Programa de Educação Continuada em Engenharia II.t.

AGRADECIMENTOS

Aos meus queridos pais Wagner Redondo e Roseli Martinez Villena Redondo por apoiarem em todos os momentos de minha vida. Em especial ao meu pai pelas vírgulas e acentos que colocou no texto.

Ao meu noivo Ricardo Fabio Sato por compreender e me apoiar nos momentos de dedicação a esta monografia.

Aos meus colegas de trabalho Felipe Gallucci Curilla e Luis Augusto de Sá Pessoa, por ter emprestados livros e ajudado no esclarecimento de dúvidas, e em especial ao meu colega Luis Augusto de Sá Pessoa, por ter dedicado grande parte do seu tempo explicando sobre padrões de projeto.

Ao professor Dr. Paulo Sergio Muniz Silva pelos direcionamentos e tranqüilizações durante as reuniões de orientação.

OK

RESUMO

Cada problema de software não deve ser solucionado a partir de princípios elementares ou do zero. Deve-se reutilizar soluções que já funcionaram no passado e aplicá-las em novos projetos.

É possível, através da utilização de padrões de projetos nominar, abstrair e identificar os principais aspectos comuns de uma estrutura de projeto, para tornar esta estrutura útil para a criação de um projeto orientado a objetos reutilizáveis.

Atualmente, há um conjunto significativo de padrões de projetos aplicáveis a problemas comuns em contextos bastante diversificados. Tal situação requer uma forma estruturada para descrever os problemas que facilite a busca de padrões de projetos candidatos aplicáveis e a seleção da solução apropriada. Para tanto, surgiram linguagens para descrever padrões de projeto, que procuram tornar bem compreendidas as soluções empregando estes padrões.

Este trabalho tem como objetivo apresentar sugestivas de soluções utilizando padrões de projeto, para problemas de projeto num Portal de vendas de conteúdo para dispositivos móveis, descrevendo a descoberta e a seleção do padrão aplicado pela solução. A apresentação segue um idioma de padrões definido previamente.

Palavras-chave: Padrões de Projeto. Linguagem de Padrões. Reutilização de Software.

ABSTRACT

The solution for each software problem should not occur through elementary principles or through the very beginning approach. The idea is to apply solutions that have successfully worked for projects in the past and reuse these solutions for new projects.

The utilization of design patterns allows us to abstract and identify the main common aspects of a project structure. Thus, becoming it useful for the creation of a project oriented toward reusable objects.

Nowadays, there are quantities of important design patterns applicable to common problems and also to much diversified contexts. Such condition requires a manner somewhat standard to describe the problems which will facilitate the search of ideal design patterns and the choice of the appropriated solution. In addition, languages emerged to describe the design Patterns, which seek to make the solutions through these patterns pretty well understood.

This work has the objective to present some worthy solutions by utilizing the design patterns, for project problems in a content sale Portal for mobiles devices, describing the discovery and the election of the applied pattern for the solution. This presentation follows a definitive language of patterns previously established.

Keywords: Design Patterns, Language of Patterns, Reusable Software.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACK	*Acknowledge*
API	*Application Programming Interface* ou Interface de Programação de Aplicativos
CSS	*Cascading Style Sheets* ou Folhas de Estilo em Cascata
GSM	*Global System for Mobile Communications ou* Sistema Global para Comunicações Móveis.
HTML	*HyperText Markup Language* ou Linguagem de Marcação de Hipertexto
HTTP	*HyperText Transfer Protocol* ou Protocolo de Transferência de Hipertexto
IP	*Internet Protocol* ou Protocolo de Internet
MMS	*Multimedia Messaging Service* ou Serviço de Mensagens Multimídia
MO	*Mobile Originated*
MT	*Mobile Terminated*
SAC	Sistema de Atendimento ao Cliente
SB	*SimBrowsing*
SMS	*Short Message Service* ou Serviço de Mensagens Curtas
SMSC	*Short Message Service Center*
XHTML MP	*Xtensible Hypertext Markup Language*
XML	*eXtensible Markup Language*

WAP	*Wireless Application Protocol* ou Protocolo para aplicações sem fio
WEB	*World Wide Web*
WIG WML	*Wireless Internet Gateway Wireless Markup Language*
WML	*Wireless Markup Language*

GLOSSÁRIO

ACK- Sinal eletrônico de reconhecimento usado em transmissões de dados, para informar status (Como por exemplo: "pode enviar mais dados", "não recebi", etc.).

Cache – Dispositivo de acesso rápido, interno a um sistema, que serve de intermediário entre um operador de um processo e o dispositivo de armazenamento ao qual esse operador acede. A vantagem principal na utilização de uma *cache* consiste em evitar o acesso ao dispositivo de armazenamento - que pode ser demorado - e que vale a pena armazenar as informações procuradas em meio mais rápido.

CSS – Linguagem de estilo utilizada para definir a apresentação de documentos escritos em uma linguagem de marcação, como HTML ou XML.

Framework – Estrutura de suporte definida em que um outro projeto de software pode ser organizado e desenvolvido. Um *framework* pode incluir programas de suporte, bibliotecas de código, linguagens de *script* e outros softwares para ajudar a desenvolver e juntar diferentes componentes de um projeto de software.

***Gateway ou* Porta de Ligação** – Máquina intermediária geralmente destinada a interligar redes, separar domínios de colisão, ou mesmo traduzir protocolos.

Launch browser – Funcionalidade que permite que o navegador WAP seja lançado ou executado a partir do navegador *SimBrowsing*.

Layout – Esboço mostrando a distribuição física, tamanhos e pesos de elementos como texto, gráficos ou figuras num determinado espaço.

Links – Nome de um navegador WEB em modo texto, desenvolvido em código livre.

Logs – Termo utilizado para descrever o processo de registro de eventos relevantes num sistema computacional.

Mensagem MO – Mensagem SMS originada em dispositivo móvel.

Mensagem MT – Mensagem SMS enviada a dispositivo móvel.

Middleware – Programa de computador que faz a mediação entre outros softwares.

MMS – Serviço de mensagens de texto com áudio e imagem, que permite a distribuição automática e imediata de mensagens. Entretanto, diferente do SMS, o MMS permite ao usuário do celular enriquecer suas mensagens incorporando som, imagens e outros conteúdos elaborados, transformando-as em mensagens visuais e sonoras.

Plugins - Programa de computador (geralmente pequeno e leve) que serve normalmente para adicionar funções a outros programas maiores, provendo alguma funcionalidade especial ou muito específica.

Ringtones – Músicas adaptadas para serem reproduzidas como campainhas/toques de aparelhos celulares.

SimBrowsing – Tecnologia que permite a navegação na Internet com dispositivos móveis através do *SinCard* do aparelho celular GSM.

SMSC – Serviço que normalmente as operadoras GSM mantém que sabe como rotear mensagens SMS. O SMSC conecta-se à rede da operadora de celular e a outros SMSCs. Provê um serviço de armazenagem e envio de mensagens, ou seja, as mensagens não são enviadas diretamente do remetente ao destinatário, sendo sempre enviadas via SMS-Center.

SMSLink – *Link* de internet enviado através de mensagem SMS.

User-agent – Valor único universal de identificação de cada aparelho celular.

WAP – Tecnologia que permite a navegação na Internet com dispositivos móveis. É um protocolo de transmissão de dados sobre uma rede *Wireless* de banda estreita. É implementado em dois componentes principais: o WAP *Gateway* e o micro navegador. O *gateway* conecta o dispositivo *Wireless* à Internet, enquanto o micro navegador usa um formato de documento XML, o *Wireless Mark-up Language* (WML), para exibir as páginas.

WAP Gateway – Torna mais eficientes as transmissões para dispositivos móveis. Como o dispositivo WAP só entende WML no formato tokenized/compiled/binary, a função do WAP *gateway* é converter o conteúdo para este formato. Pelo lado do servidor HTTP, o *gateway* pode fornecer informações sobre o dispositivo *Wireless*, através dos parâmetros enviados pelo cabeçalho do HTTP.

Wireless – Tecnologia sem fio que permite a conexão entre diferentes pontos sem a necessidade do uso de cabos (nem de telefonia, nem de TV a cabo, nem de fibra ótica), através da instalação de uma antena e de um rádio de transmissão.

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

Atualmente, os sistemas orientados a objetos permitem agregar qualidades importantes aos sistemas desenvolvidos, como sua reutilização, por exemplo, mas somente por estar utilizando-a, não é garantia da obtenção dessas qualidades.

Em desenvolvimento de software é possível identificar, que a procura por uma solução para um problema específico possui características similares, ou mesmo, igual a de um projeto anteriormente desenvolvido. Mas, muitas vezes, a solução e o problema não foram registrados, documentados e compreendidos, impossibilitando ou dificultando o reaproveitamento das soluções. Devido a essa falta de documentação, problemas parecidos que se repetem em outros projetos não são identificados, consumindo tempo e recursos em busca de soluções que já haviam sido encontradas.

Cada problema não deve ser resolvido a partir de princípios elementares ou do zero. Por isso, deve-se utilizar soluções que já funcionaram no passado, para sua reutilização repetidamente em novos projetos.

Os padrões de projeto ajudam a documentar e mostrar um problema importante que pode ocorrer no projeto ou na implementação de uma aplicação, discutindo a melhor solução prática para o problema, proporcionando elementos que conduzem ao reaproveitamento de soluções para projetos, e não apenas a reutilização de código.

Os padrões de projeto tornam mais fácil a reutilização de soluções bem sucedidas para construir novos projetos orientados a objetos de forma flexível. O uso de padrões de projeto em software orientado a objetos, se bem projetado, possibilita aos desenvolvedores reutilizar e empregar a solução em novos projetos.

Sabe-se que os padrões de projeto possuem um vocabulário comum de projeto, facilitando a comunicação, documentação e aprendizado dos sistemas de software.

Num Portal de vendas de conteúdos para dispositivos móveis foram encontrados alguns problemas que poderiam ser resolvidos com padrões de projetos, problemas que envolvem adaptação de *layout* para cada modelo de aparelho celular, pois, com a grande diversidade destes aparelhos no mercado, cada um possui suas características específicas. Outro problema relacionado a um Portal de vendas de conteúdos, é a grande diversidade de conteúdos disponíveis para serem oferecidos ao usuário. Isso introduz uma lentidão na recuperação de todos os conteúdos para serem exibidos no aparelho celular, além de outros problemas levantados no presente trabalho.

Ao se utilizar padrões de projetos, quando ocorrem novos acréscimos de requisitos, a manutenção do código é facilitada. Esse fato pode auxiliar a resolução de problemas encontrados no Portal pois, muitas vezes há grandes alterações de código já em uso.

Com a utilização de padrões é possível que os problemas e suas soluções sejam documentados e deixados com um vocabulário comum, para que posteriormente esta solução possa ser reaproveitada em outros Portais.

1.2 Objetivo

O objetivo deste trabalho é apresentar sugestões de soluções, com a utilização de padrões de projeto, para alguns problemas de projeto e manutenção de sistemas de software de um Portal de conteúdo para dispositivos móveis.

A descrição das soluções utiliza um formato definido em um idioma de padrões e procura apresentar, de modo compreensível, as argumentações envolvidas nas decisões da adoção de determinadas soluções. O principal objetivo de se seguir esse tipo de apresentação é o de procurar transmitir, a um projetista recém-

contratado pela empresa responsável pelo Portal ou ao leitor, as razões subjacentes à tomada de decisão sobre as escolhas de padrões de projeto como soluções para os problemas analisados.

1.3 Metodologia

Foi adotado como procedimento metodológico neste trabalho, o estudo dos padrões de projetos clássicos de (GAMMA et al., 2000), para ser obtido uma maior compreensão sobre padrões de projeto, como também o estudo do idioma de padrões de (MESZAROS; DOBLE, 1998), para que as razões da escolha do padrão para a solução de um problema seja bem estruturada de uma forma que o público-alvo compreenda.

Para identificar o melhor padrão a ser aplicado nos problemas do estudo de caso utilizou-se a sugestão de (GAMMA et al., 2000), de como selecionar e utilizar um padrão de projeto que resolvam problemas específicos. A partir desses conceitos foi identificado os padrões candidatos e qual seria o melhor que se aplicasse ao problema.

1.4 Estrutura de Trabalho

Este trabalho está estruturado em 5 capítulos, organizados na seguinte estrutura:

Capítulo 1 – Estão descritos: a introdução, a motivação, os objetivos, e a estrutura do trabalho.

Capítulo 2 – São apresentados os principais conceitos a respeito dos padrões clássicos de projeto e as principais motivações para a sua utilização.

Capítulo 3 – É apresentada uma linguagem de padrão, utilizada no estudo de caso para a escrita das soluções dos problemas estudados.

Capítulo 4 – É apresentado um estudo de uma plataforma tecnológica de suporte a um Portal para dispositivos móveis denominada Plataforma Alpha. Analisa-se como os padrões de projeto oferecem soluções para os problemas enfrentados por essa Plataforma.

Capítulo 5 – São apresentadas as conclusões do trabalho.

2 PADRÕES DE PROJETO

Para facilitar a compreensão dos capítulos que seguem, o presente capítulo reúne as principais definições dos padrões clássicos de projeto e apresenta as principais motivações para a sua utilização.

2.1 O que são padrões de projeto

Os padrões de projeto ganharam popularidade durante a última década como um modo de captar, organizar e reutilizar importantes soluções de software. (COPLIEN, 2004)

Um padrão de projeto documenta e mostra um problema importante que pode ocorrer no projeto ou na implementação de uma aplicação, discutindo a melhor solução prática para o problema.

Na literatura pesquisada, foram identificadas algumas definições para padrões de projeto:

> Cada padrão descreve um problema que ocorre repetidas vezes em nosso ambiente, e então descreve o núcleo da solução para aquele problema, de tal maneira que pode-se usar essa solução milhões de vezes sem nunca fazê-la da mesma forma duas vezes. (ALEXANDER, C., 2001), sobre padrões em Arquitetura.
>
> Os padrões de projeto são descrições de objetos e classes comunicantes que são customizados para resolver um problema geral de projeto num contexto particular. (GAMMA et al., 2000, p.20), sobre padrões em software
>
> Um padrão de projeto é uma estrutura de experiência que contribui à estrutura global de um sistema. O sistema é construído para algum propósito, e cada padrão tem o objetivo de resolver um propósito. (COPLIEN, 2004, p.5)

Padrões de projeto descrevem os núcleos das soluções para problemas recorrentes no desenvolvimento de sistemas de software orientados a objetos. Cada padrão

descreve um problema e sua solução, de tal forma que se alguém utilizar uma solução repetidas vezes poderá nunca fazê-la exatamente da mesma maneira. (GAMMA et al., 2000)

Os padrões de projeto são modelos padronizados de uma estrutura ou de um processo que possam ser aplicados em casos específicos de uma forma consistente. Os padrões de projeto permitem assim reutilizar não somente o código, mas também os resultados de uma análise de um projeto em outros contextos. (BLILIE, 2002)

Cada padrão de projeto identifica e focaliza um problema, também estabelece um nome, define o problema, a solução, quando aplicar esta solução as suas conseqüências e os custos e benefícios advindos dessa utilização. Através de um padrão de projeto é possível nominar, abstrair e identificar os aspectos chaves de uma estrutura de projeto comum para torná-la útil para a criação de um projeto orientado a objetos reutilizável. (GAMMA et al., 2000)

2.2 Necessidade da utilização de padrões de projeto

Os padrões de projeto são necessários porque proporcionam uma facilidade na reutilização de soluções de software, como também possuem um vocabulário comum de projeto, facilitando a comunicação, documentação e aprendizado dos sistemas de software.

Os padrões de projeto podem ser referências de qualidade para um sistema orientado a objetos. Assim, um sistema de software pode ser caracterizado, entre outros aspectos, pelo número de padrões de projetos utilizados como referência. Porém é muito importante que uma pessoa escolha os padrões de projeto adequados para cada situação, caso contrário, uma escolha imprópria acarretará enormes estruturas de classe, que poderão resultar na pior situação, que é a diminuição da qualidade do código. (BALANYI, FERENC, 2003)

Normalmente, os padrões de projeto apresentam soluções que requerem vários passos. Deve-se utilizar padrões de projeto em soluções que se repetem várias vezes em diferentes contextos. Se o problema só aparece em um determinado contexto, não faz sentido sua utilização. (SCHNEIDE, 1999)

Os padrões de projeto são necessários para melhorar o projeto do sistema, e também para aplicação em qualquer ponto do ciclo de vida do projeto. Eles são documentados normalmente em um nível relativamente alto de abstração, e fornecerão grandes benefícios, quando aplicados no início do projeto. Porém, se o padrão for aplicado durante a fase de implementação, muito provavelmente que o código existente deverá ser retrabalhado. (ALUR, CRUPI, MALKS, 2004)

As necessidades da utilização dos padrões de projeto estão atreladas às causas e benefícios seguintes: (ALUR, CRUPI, MALKS, 2004)

- Experimentos testados e comprovados que refletem a experiência e conhecimento dos técnicos e analistas que utilizaram estes padrões com sucesso em seu trabalho;
- Utilização de um vocabulário comum capaz de expressar grandes soluções;
- Delimita um espaço para a solução. Usar um padrão restringe e cria limites dentro de um espaço no qual um projeto ou uma implementação pode ser aplicada;
- São reutilizáveis porque fornecem uma solução que pode ser adaptada para diferentes problemas;
- Facilitam e reduzem o tempo de aprendizado de uma determinada biblioteca de classes. E isto é fundamental para o aprendizado dos analistas e técnicos novatos;
- Minimizam a necessidade de retrabalhos, porque quanto mais cedo são utilizados ocasionarão menos retrabalhos em etapas mais avançadas do projeto.

Existem algumas propostas de padrões de projeto no mercado. A proposta utilizada como base fundamental deste trabalho são os padrões clássicos de (GAMMA et al., 2000), pois eles são os mais conhecidos pelos analistas e técnicos. Os padrões clássicos possuem uma solução genérica, possibilitando a utilização com qualquer linguagem de programação.

Existem outras propostas de padrões de projetos tais como: (ALUR, CRUPI, MALKS, 2004), que oferecem soluções de padrões especialmente para desenvolvedores J2EE.

Também existem as propostas de padrões de projeto de (GRAND, 1998), que, além das soluções de seus padrões, incluem também os padrões de (GAMMA et al., 2000).

2.3 Como selecionar e usar um padrão de projeto

Uma questão central para a aplicação de padrões de projeto é a seleção do padrão adequado para aplicar em um problema específico.

Inicialmente é necessário um bom conhecimento sobre os padrões de projetos para se ter acesso ao seu código e retrabalhá-lo. Sem esse conhecimento é muito difícil escolher qual padrão deve ser utilizado em uma determinada ocasião, bem como começar a aplicá-los a novos projetos. (FREEMAN Eric, FREEMAN, Elisabeth, 2005) (COPLIEN, 2004)

Diferentes autores fazem escolhas de projetos diferentes e suas descrições para o mesmo padrão variam. Desse modo, ao se procurar um código fonte para um padrão específico provavelmente não será fácil encontrá-lo, pois a variação faz com que o código fonte para um padrão específico raramente exista. (WIRFS-BROCK, 2006)

Com muitos padrões disponíveis para se escolher, normalmente é difícil encontrar aquele que possa tratar um problema de projeto em específico. Para a seleção de

um padrão de projeto adequado para um problema (GAMMA et al., 2000) sugerem as recomedações que seguem.

1. Considere como padrões de projeto solucionam problemas de projeto

Os padrões de projeto ajudam a solucionar muitos problemas de várias maneiras diferentes. Alguns dos problemas que podem ser resolvidos pelos padrões são:

- Encontrar objetos apropriados;
- Determinar a granulação dos objetos;
- Especificar a interface dos objetos;
- Especificar as implementações dos objetos;
- Distiguir herança de classe de herança de interface;
- Programar para uma interface e não para uma implementação;
- Colocar os mecanismos de reutilização para funcionar;
- Distinguir herança de composição;
- Delegar;
- Distinguir heranças de tipos parametrizados;
- Relacionar estruturas de tempo de execução e de tempo de compilação;
- Projetar para permitir mudanças.

2. Examinar as seções de descrição dos problemas de cada padrão

Ler as definições das Intenções de todos os padrões para selecionar um ou mais padrões que possivelmente poderão solucionar o problema.

3. Estudar como os padrões se interrelacionam

O estudo desses relacionamentos pode ajudar o direcionamento para o padrão adequado.

4. Estudar padrões de finalidades semelhantes

O estudo dos padrões de criação, dos padrões estruturais e dos padrões comportamentais fornece uma visão sobre as semelhanças e diferenças de padrões com propósitos similares.

5. Examinar uma causa de reformulação de projeto

Verificar as causas para a reformulação citados no item 1 e identificar quais são os padrões que ajudam a identificar estas causas.

6. Considerar o que deveria ser variável no projeto

Esta abordagem é o oposto de se focalizar nas causas de reformulação. Considerar o que pode forçar uma mudança em um projeto e o que quer mudar sem alterá-lo.

Para utilizar padrões de projeto, não existe uma técnica oficial definida. Existem apenas sugestões que os autores geralmente oferecem para introduzir uma maneira de utilizar um padrão. Normalmente, no decorrer da utilização de padrões, cada pessoa desenvolve sua própria técnica que mais lhe agrada para trabalhar com padrões de Projeto. (GAMMA et al., 2000) sugerem que uma vez escolhido um padrão de projeto, deve-se seguir passo a passo como aplicar efetivamente um padrão de projeto. Os passos a serem seguidos são:

1° passo: Ler o padrão por inteiro uma vez, para obter sua visão geral.

2° passo: Estudar as seções que descrevem a estrutura, os participantes e as colaborações.

3° passo: Ver a seção que contenha um exemplo de código, para um exemplo concreto.

4° passo: Definir nomes para os participantes do padrão que tenham sentido no contexto da aplicação.

5° passo: Definir as classes.

6° passo: Definir nomes específicos da aplicação para as operações no padrão.

7° passo: Implementar as operações para apoiar as responsabilidades e colaborações presentes no padrão.

Além de entender como usar um padrão de projeto é preciso entender quando não utilizá-los.

Os padrões de projeto não devem ser aplicados sem necessidade. O uso inadequado pode acarretar em uma grande quantidade de classes desnecessárias, deixando o código muito difícil de compreender e de manipular, deteriorando o

desempenho do sistema. Um padrão de projeto deverá apenas ser aplicado quando a flexibilidade que lhe é oferecida é realmente necessária.

2.4 Padrões de Projeto Clássicos

Na proposta de (GAMMA et al., 2000) foram identificados 23 tipos diferentes de padrões de classe e de objeto, divididos em padrões de criação, estruturais e comportamentais. É possível também classificar os padrões de projeto em análise ou em projeto. Os padrões de projeto permitem maneiras padronizadas de executar um sistema de software. Os padrões de análise são modelos conceituais dos elementos reais dos sistemas que o software representa. (BLILIE, 2002)

Os 23 padrões da proposta de (GAMMA et al., 2000) estão selecionados e organizados conforme o Quadro 1.

		Propósito		
		1. Criação	2. Estrutura	3. Comportamento
Escopo	Classe	Método Fábrica (*Factory Method*)	Adaptador (*Adapter*)	Intérprete (*Interpreter*) Método Gabarito (*Template Method*)
	Objeto	Fábrica Abstrata (*Abstract Factory*) Construtor (*Builder*) Protótipo (*Prototype*) Filho Único (*Singleton*)	Adaptador de Objeto (*Object Adapter*) Ponte (*Bridge*) Compósito (*Composite*) Decorador (*Decorator*) Fachada (*Facade*) Peso-Pluma (*Flyweight*) Procurador (*Proxy*)	Cadeia de Responsabilidades (*Chain Of Responsibility*) Comando (*Command*) Iterador (*Iterator*) Mediador (*Mediator*) Memento (*Memento*) Observador (*Observer*) Estado (*State*) Estratégia (*Strategy*) Visitante (*Visitor*)

Quadro 1 - Organizações dos padrões de projetos. (GAMMA et al., 2000)

2.4.1 Padrões de Criação

Esses padrões estão relacionados à criação de objetos. Eles podem decidir o tipo do objeto e sua multiplicidade. Não é possível identificar claramente a estrutura do padrão porque a real funcionalidade é escondida nas implementações das funções. Embora seja difícil de reconhecer estes padrões, felizmente não é impossível porque as criações de objetos podem ser identificadas a partir do código fonte. (BALANYI, FERENC, 2003)

Os padrões de criação abstraem o processo de instanciação. Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados. Um padrão de criação de classe usa a herança para variar a classe que é instanciada, enquanto que um padrão de criação de objetos delegará a instanciação para outro objeto. (GAMMA et al., 2000)

Estes padrões se tornam ainda mais importantes à medida que os sistemas evoluem no sentido de depender mais na composição de objetos do que na herança de classes. Criar objetos com comportamentos particulares exige mais do que simplesmente instanciar uma classe.

Existem dois assuntos recorrentes nesses padrões:

1º Todos emcapsulam conhecimento sobre quais classes concretas são usadas pelo sistema.

2º Ocultam o modo como as instâncias destas classes são criadas e juntadas.

Tudo o que o sistema sabe é que suas classes são definidas por classes abstratas. Conseqüentemente, esses padrões possibilitam uma grande flexibilidade no que é criado, quem cria, como e quando é criado. Eles permitem configurar um sistema estaticamente, isto é, em tempo de compilação, ou dinamicamente, isto é, em tempo de execução. (GAMMA et al., 2000)

Serão apresentadas em seguida, as finalidade dos 5 padrões de criação segundo (GAMMA et al., 2000).

Fábrica Abstrata (*Abstract Factory*) - Fornece uma interface para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.

Construtor (*Builder*) - Separa a construção de um objeto complexo da sua representação, de modo que o mesmo processo de construção possa criar diferentes representações.

Método Fábrica (*Factory Method*) - Define uma interface para criar um objeto, e deixa as subclasses decidirem que classe instanciar. O Método Fábrica permite adiar a instanciação para as subclasses.

Protótipo (*Prototype*) - Especifica quais serão os tipos de objetos a serem criados usando uma instância de protótipo e criando novos objetos pela cópia desse protótipo.

Filho Único (*Singleton*) - Garante que uma classe tenha somente uma instância e fornece um ponto global de acesso a ela.

Segundo (GAMMA et al., 2000), existem duas maneiras de parametrizar um sistema pelas classes de objetos que ele cria:

1° Criando subclasses da classe que cria os objetos, esta maneira corresponde ao padrão Método Fábrica. A principal desvantagem dessa solução é que necessita da criação de uma nova subclasse somente para alterar a classe do produto. Essas mudanças podem causar um excesso de modificações no sistema.

2° Concentrando-se na composição dos objetos, ou seja, identificando um objeto que será responsável por conhecer a classe dos objetos-produto e transformando-a em um parâmetro do sistema. Esse é o aspecto principal dos padrões Fábrica Abstrata, Construtor e Protótipo.

No contexto de Fábrica Abstrata, o objeto-fábrica cria objetos de várias classes. Em Protótipo o objeto-fábrica constrói um objeto-produto copiando um objeto prototípico. Nesse caso, o objeto-fábrica e o protótipo são o mesmo objeto, porque o protótipo é responsável por retornar o produto. No padrão Construtor um objeto-fábrica constrói

incrementalmente um objeto complexo usando um protocolo igualmente complexo. (GAMMA et al., 2000)

Decidir qual padrão será usado em um sistema de software depende de muitos fatores. O padrão Método Fábrica faz com que o projeto seja mais adaptável. Outros padrões de projeto requerem novas classes, enquanto que o Método Fábrica somente exige uma nova operação. Normalmente as pessoas utilizam como maneira padrão de criar os objetos o Método Fábrica, porém ele não é necessário quando a classe que é instanciada nunca muda ou quando a instanciação ocorre em uma operação que subclasses podem facilmente redefinir. (GAMMA et al., 2000)

Projetos que utilizam Fábrica Abstrata, Protótipo ou Construtor são mais flexíveis do que aqueles que utilizam o Método Fábrica, porém eles são mais complexos. Normalmente os projetos começam utilizando o Método Fábrica e evoluem para outros padrões de criação à medida que o projetista descobre onde é necessária maior flexibilidade. (GAMMA et al., 2000)

2.4.2 Padrões Estruturais

Esses padrões tratam da composição de classes ou objetos. Eles definem hierarquias de classe. Nesses padrões a maioria das características é descrita através das declarações das operações e atributos, sendo assim mais fáceis de serem reconhecidos do que os padrões de criação. (BALANYI, FERENC, 2003)

Os padrões estruturais se preocupam com a forma como as classes e objetos são compostos para formar estruturas maiores. Os padrões estruturais de classe utilizam a herança para compôr interfaces ou implementações. Os padrões estruturais de objetos descrevem maneiras de compôr objetos para obter novas funcionalidades. A composição de objetos permite a flexibilidade de compôr os objetos em tempo de execução, o que é impossível com a composição estática. (GAMMA et al., 2000)

A finalidade dos 7 padrões estruturais serão apresentadas de uma forma sucinta segundo (GAMMA et al., 2000).

Adaptador (*Adapter*) - Converte a interface de uma classe em outra interface esperada pelos clientes. O Adaptador permite que classes com interfaces imcompatíveis trabalhem em conjunto.

Ponte (*Bridge*) - Desacopla uma abstração da sua implementação, de modo que as duas possam operar independentemente.

Compósito (*Composite*) - Compõe objetos em estruturas de árvores para representarem hierarquias partes para o todo. Possibilita o tratamento uniforme de objetos individuais e de composição de objetos.

Decorador (*Decorator*) - Agrega dinamicamente novas responsabilidades a um objeto. Permite o uso de subclasses para extensão da funcionalidade.

Fachada (*Façade*) - Fornece uma interface unificada para um conjunto de interfaces em um sistema.

Peso-Pluma (*Flyweight*) - Usa o compartilhamento para dar suporte eficiente a uma grande quantidade de objetos de granulação fina.

Procurador (*Proxy*) - Fornece um substituto ou marcador da localização de outro objeto para controlar o acesso a ele.

Todos os parágrafos que seguem baseiam-se em (GAMMA et al., 2000).

Ambos os padrões, Adaptador e Ponte, possibilitam a flexibilidade ao fornecer um nível de endereçamento indireto para outro objeto. Ambos repassam as solicitações para este objeto, partindo de uma interface diferente da sua própria interface.

A principal diferença entre esses padrões está nas suas intenções. O Adaptador e a Ponte são freqüentemente usados em pontos diferentes do ciclo de vida do software. Um adaptador se torna necessário quando duas classes incompatíveis precisam trabalhar em conjunto, para evitar duplicação de código e que o acoplamento não tenha sido previsto. Ao contrário, a Ponte compreende desde o início que uma abstração deve ter várias implementações e que ambas possam

evoluir independentemente. Em outras palavras, o Adaptador faz as coisas funcionarem após elas terem sido projetadas e a Ponte faz funcionarem antes que elas existam.

O padrão Compósito e o Decorador possuem diagramas de estruturas similares, partindo do princípio que ambas dependem da composição recursiva para organizar um número indefinido de objetos. Pode-se pensar que um objeto Decorador é um Compósito degenerado, mas isso ignora o significado do padrão Decorador. A similaridade termina na composição recursiva, pois as suas intenções são diferentes. O foco do padrão Compósito não está na decoração e sim na representação.

Outro padrão com a estrutura similar do Decorador é o Procurador. Ambos os padrões descrevem como fornecer um nível de endereçamento indireto para um objeto, e suas implementações. Eles mantêm uma referência para um outro objeto para o qual repassam solicitações.

O Procurador compõe objetos e fornece uma interface idêntica para os clientes, a exemplo do que faz o Decorador. Porém, o padrão Procurador diferentemente do Decorador não está preocupado em incluir ou excluir propriedades dinamicamente, e não está projetado para a composição recursiva.

O Decorador trata da situação em que a funcionalidade total de um objeto não pode ser determinada em tempo de compilação. Esse detalhe torna a composição recursiva uma parte essencial do Decorador. Não sendo o caso do Procurador, pois o Procurador focaliza um relacionamento – entre o Procurador e seu objeto – e esse relacionamento pode ser expresso estaticamente.

2.4.3 Padrões Comportamentais

Os padrões comportamentais preocupam-se com algoritmos, com atribuição de responsabilidades entre objetos e descrevem como as classes interagem entre si. O

comportamento está definido nas operações. Isto faz com que esses padrões sejam mais difíceis de reconhecer. (BALANYI, FERENC, 2003)

Tais padrões caracterizam fluxos de controle difíceis de serem seguidas em tempo de execução. Eles afastam o foco do fluxo de controle para permitir a concentração somente na maneira como os objetos são interconectados. (GAMMA et al., 2000)

Os padrões comportamentais de classe utilizam a herança para distribuir o comportamento entre classes, enquanto os padrões comportamentais de objetos utilizam a composição de objetos em vez de herança. Alguns padrões descrevem como um grupo de objetos pares cooperam para a execução de uma tarefa que nenhum objeto sozinho poderia executar por si só. (GAMMA et al., 2000)

Apresentam-se, a seguir, a finalidade dos 11 padrões comportamentais de (GAMMA et al., 2000).

Cadeia de Responsabilidades (*Chain Of Responsibility*) - Evita o acoplamento do remetente de uma solicitação ao seu receptor ao dar a mais de um objeto a oportunidade de tratar a solicitação. Também encadeia os objetos receptores, passando a solicitação ao longo da cadeia até que um objeto o trate.

Comando (*Command*) - Encapsula uma solicitação como um objeto, possibilitando parametrizar clientes com diferentes solicitações, enfileirar ou efetuar o registro de solicitações e dar suporte a operações que podem ser desfeitas.

Intérprete (Interpreter) - Define uma representação para uma gramática de uma linguagem, juntamente com um interpretador que usa a representação para interpretar sentenças da linguagem.

Iterador (*Iterator*) - Disponibiliza uma maneira de acessar, seqüencialmente, os elementos de um objeto agregado sem expor a sua representação subjacente.

Mediador (Mediator) - Define um objeto que encapsula a forma como um conjunto de objetos interage. O Mediador promove um fraco acoplamento ao evitar que os objetos se refiram uns aos outros explicitamente e permite variar suas interações independentemente.

Memento (*Memento*) - Captura e externaliza um estado interno de um objeto, sem violar o encapsulamento, de maneira que o objeto possa ser restaurado para este estado mais tarde.

Observador (*Observer*) – Define uma dependência um-para-muitos entre objetos, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente.

Estado (*State*) – Permite a um objeto alterar seu comportamento quando o seu estado interno muda.

Estratégia (*Strategy*) – Define um conjunto de algoritmos, encapsula cada um deles e torná-os intercambiáveis. Este padrão permite que o algoritmo varie independentemente dos clientes que o utilizam.

Método Gabarito (Template Method) – Define o esqueleto de um algoritmo em uma operação, postergando alguns passos para as subclasses. Este padrão permite que subclasses redefinam certos passos de um algoritmo sem mudar a estrutura dele.

Visitante (Visitor) – Representa uma operação a ser executada nos elementos de uma estrutura de objetos. Permite definir uma nova operação sem mudar as classes dos elementos sobre os quais opera.

2.5 Padrões candidatos do estudo de caso

Esta seção irá detalhar os padrões utilizados no estudo de caso.

2.5.1 Comando *(Command)*

Esse padrão permite desconectar o autor de uma solicitação do objeto que deverá executar a ação solicitada. É possível desconectá-lo introduzindo objetos de comando no projeto. Um objeto de comando está no centro dessa desconexão e encapsula uma solicitação para fazer algo em um objeto específico, Exemplo: O objeto de comando correspondente a cada botão em um controle remoto. Quando o botão for pressionado será solicitado ao objeto de comando para fazer algo. O controle remoto não terá a menor idéia de qual seja essa tarefa, ele só tem um objeto de comando que sabe como se comunicar com o objeto certo para executar a tarefa. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

Um invocador faz uma solicitação em um objeto de comando chamando um método de execução, que invoca ações no receptor. Os invocadores podem ser parametrizados por comandos, mesmo dinamicamente durante uma execução. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

Os comandos fornecem suporte à reversão das ações executadas através da implementação de um método que restaura o objeto ao seu estado anterior, antes que seu método de execução seja chamado. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

O padrão Comando pode ser aplicado para: (GAMMA et al., 2000)

- Parametrizar objetos por uma ação a ser executada;
- Especificar, enfileirar e executar solicitações em tempos diferentes;
- Desfazer operações, permitindo a armazenagem de estados para reverter seus efeitos no próprio comando;
- Registrar mudanças, de maneira que possam ser reaplicadas no caso de uma queda de sistema;

- Estruturar um sistema em torno de operações de alto nível construídas sobre operações primitivas.

O padrão Comando possui as seguintes conseqüências: (GAMMA et al., 2000)

- Desacopla o objeto que invoca a operação daquele que sabe como executá-la;
- São objetos que podem ser manipulados e estendidos como qualquer outro objeto;
- Pode-se montar comandos para formar um comando composto;
- É fácil acrescentar novos comandos, pois não é preciso modificar classes existentes.

2.5.2 Cadeia de Responsabilidades *(Chain Of Responsibility)*

Esse padrão cria uma cadeia de objetos que examinam seqüencialmente uma solicitação. Após examinar a solicitação, cada objeto pode processá-la ou passá-la adiante para o próximo objeto na cadeia. Cada objeto na cadeia funciona como um processador e tem um objeto sucessor, isto é, o objeto pode tratar a solicitação ou pode repassá-la a seu sucessor. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

O primeiro objeto da cadeia, o que recebe a solicitação, trata a solicitação ou a repassa para o próximo sucessor da cadeia, que faz a mesma coisa. O objeto que fez a solicitação não tem conhecimento explícito de quem a tratará, neste caso é dito que a solicitação tem um receptor implícito. (GAMMA et al., 2000)

Para repassar a solicitação ao longo da cadeia e para garantir que os receptores permaneçam implícitos, cada objeto compartilha uma interface comum para o tratamento de solicitações e para acessar seu sucessor na cadeia. (GAMMA et al., 2000)

O padrão Cadeia de Responsabilidades pode ser aplicado quando: (GAMMA et al., 2000)

- Mais de um objeto pode tratar uma solicitação. O objeto que tratará a solicitação deve ser escolhido automaticamente;
- Deseja-se omitir uma solicitação para um ou mais objetos, sem especificar explicitamente o receptor;
- O conjunto de objetos que pode tratar uma solicitação deveria ser especificado dinamicamente.

Esse padrão tem os benefícios e as conseqüências a seguir: (GAMMA et al., 2000) (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

- Possui um acoplamento bem baixo, isto é, desconecta o remetente da solicitação dos seus destinatários;
- Simplifica o objeto, porque ele não precisa conhecer a estrutura da cadeia e nem manter referências diretas aos seus membros;
- Permite uma flexibilidade na atribuição de responsabilidades a objetos, possibilitando acrescentar e remover dinamicamente responsabilidades.
- Não é garantido que um receptor tratará uma solicitação, isto é, uma vez que uma solicitação não tenha um receptor explícito, não há garantia de que ela será tratada (o que pode ser uma vantagem ou uma desvantagem).
- A detecção e correção de erros durante a execução tende a ser difícil.

2.5.3 Procurador *(Proxy)*

Salvo menção em comentário, esta seção está baseada em (FREEMAN Eric, FREEMAN, Elisabeth, 2005).

O Procurador é um substituto de um objeto real. Finge que é um objeto real mas, na verdade, estará apenas se comunicando através de uma rede com o objeto real.

Existem várias implementações do padrão Procurador, como o Procurador de *Cache*, Procurador de Sincronização, Procurador de *Firewall*, Procurador de Cópia-Sobre-Gravação, Procurador Remoto, Procurador Virtual, Procurador de Proteção e etc. Todos os tipos de Procurador seguem aproximadamente o mesmo projeto geral. Todas têm em comum o fato de interceptarem uma chamada de método que o cliente está invocando.

Um Procurador Remoto atua como um representante local de um objeto remoto. Um objeto remoto é um objeto que está sendo executado em um espaço de endereçamento diferente. Um representante local é um objeto que, quando seus métodos locais são chamados, pode encaminhar as solicitações para o objeto remoto.

Um Procurador Virtual controla o acesso a um objeto cuja criação é onerosa. Esse tipo de Procurador geralmente retarda a criação desse objeto até que ela seja realmente necessária. Além disso, ele age como um substituto do objeto antes e durante a sua criação. Depois disso, o Procurador delega as solicitações diretamente ao objeto real.

Um Procurador de Proteção controla o acesso aos métodos de um objeto com base na origem da chamada.

Um Procurador de *Firewall* controla o acesso a um conjunto de recursos de uma rede, protegendo o objeto de destino contra os clientes não desejados.

Um Procurador de *Cache* fornece armazenagem temporária para os resultados de operações onerosas. Também pode permitir que múltiplos clientes compartilhem os resultados, para reduzir o processamento ou a latência da rede.

Um Procurador é estruturalmente semelhante a um Decorador, mas suas finalidades são diferentes. O padrão Decorador acrescenta comportamento a um objeto, enquanto que o Procurador controla o acesso a ele.

O padrão Procurador é aplicável em diversas situações, são elas: (GAMMA et al., 2000)

- Um Procurador Remoto disponibiliza um representante local para um objeto em um espaço de endereçamento diferente;
- Um Procurador Virtual cria objetos de grandes proporções no momento em que são solicitados;
- Um Procurador de Proteção controla o acesso para o objeto original. Este tipo de Procurador é útil quando os objetos devem ter diferentes direitos de acesso;
- Uma Referência Esperta (*Smart Reference*) é um substituto para um simples apontador que executa ações adicionais quando um objeto é acessado.

Esse padrão acrescenta um nível de referência indireta no acesso a um objeto dependendo do tipo de Procurador. As conseqüências da utilização de um Procurador são: (GAMMA et al., 2000)

- Um Procurador Remoto pode ocultar o fato de que um objeto existe em um espaço de endereçamento diferente;
- Um Procurador Virtual pode executar otimizações, tais como a criação de um objeto sob demanda;
- Tanto Procurador de Proteção quanto Referência Esperta permite tarefas adicionais de organização quando um objeto é acessado.

2.5.4 Decorador *(Decorator)*

Os decoradores têm o mesmo tipo dos objetos que eles vão decorar. Quando um decorador é composto com um componente, ele estará adicionando um novo

comportamento. É adquirido um novo comportamento não o herdando de uma superclasse, mas compondo os objetos. A herança é uma forma de extensão, mas não necessariamente a melhor maneira de obter a flexibilidade nos projetos. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

Os decoradores mudam o comportamento de seus componentes adicionando novos recursos antes e/ou depois de chamadas de métodos para o componente. Os decoradores geralmente são transparentes para o cliente do componente, a menos que o cliente esteja utilizando o tipo concreto do componente. Além disso, os decoradores podem resultar numa quantidade muito grande de pequenos objetos e o uso exagerado pode ficar complexo. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

O padrão Decorador é aplicável quando: (GAMMA et al., 2000)

- Acrescenta novas responsabilidades a objetos individuais de forma dinâmica;
- Puder ser removidas responsabilidades a objetos individuais de forma dinâmica;
- A extensão através do uso de subclasses não é prática.

Este padrão tem dois benefícios e duas deficiências: (GAMMA et al., 2000)

- Maior flexibilidade do que a herança estática;
- Evita classes sobrecarregadas de características na parte superior da hierarquia;
- Um decorador e o seu componente não são idênticos;
- Grande quantidade de pequenos objetos.

2.5.5 Visitante *(Visitor)*

O padrão de projeto Visitante permite acrescentar comportamento a uma estrutura de classe composta sem mudar as definições das classes compostas existentes. Um Visitante especifica operações para executar durante a composição de objetos.

O padrão Visitante pode ser usado para reduzir o número de operações originais de uma classe. Para que cada nova operação possa ser acrescentada separadamente, e as classes sobre os quais estão sendo aplicadas as operações sejam independentes das operações que se aplicam a elas, podem ser usadas dois passos, são elas: empacota-se inicialmente as operações relacionadas de cada classe num objeto separado, chamado um Visitante; e passando para ele elementos da árvore sintática abstrata à medida que ela é percorrida. Quando um elemento permite o visitante, envia uma solicitação para o visitante que codifica a classe do elemento. Ela também inclui o elemento como argumento. O Visitante então executará a operação para aquele elemento. (GAMMA et al., 2000)

Com esse padrão é possível definir duas hierarquias de classes: uma para os elementos sobre os quais estão sendo aplicadas as operações e uma outra para os visitantes que definem as operações sobre os elementos. É criada uma nova operação acrescentando uma nova subclasse à hierarquia da classe visitante. (GAMMA et al., 2000)

O padrão Visitante é aplicável quando: (GAMMA et al., 2000)

- Uma estrutura de objetos contém muitas classes de objetos com interfaces que diferem, e deseja-se executar operações sobre estes objetos que dependem das suas classes concretas;
- Muitas operações distintas e não-relacionadas necessitam ser executadas sobre objetos de uma estrutura de objetos;
- As classes que definem a estrutura do objeto raramente mudam e se quer definir novas operações sobre a estrutura.

O padrão tem como benefícios e conseqüências: (GAMMA et al., 2000)

- Torna fácil a adição de novas operações;
- Um Visitante reúne operações relacionadas e separa as operações não-relacionadas;
- A hierarquia de classes de um Visitante pode ser difícil de ser mantida quando novas classes são acrescentadas com freqüência;
- Um Iterador pode visitar os objetos numa estrutura à medida que os percorre pela chamada das suas operações.

2.5.6 Adaptador *(Adapter)*

Uma classe projetada para ser reutilizada pode não ser reutilizável porque sua interface não corresponde à interface específica de um domínio requerido por uma aplicação. Nesse caso, o padrão Adaptador pode resolver, pois um Adaptador é responsável por funcionalidades não oferecidas pela classe adaptada. (GAMMA et al., 2000)

Um adaptador funciona como um intermediário, recebendo as solicitações do cliente e convertendo-as para um formato que faça sentido para as classes do novo fornecedor. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

Esse padrão poderá ser utilizado quando: (GAMMA et al., 2000)

- Usa-se uma classe existente, mas sua interface não corresponde à interface de que necessita;
- Cria-se uma classe reutilizável que não tenha interfaces compatíveis;
- Necessita-se utilizar várias subclasses existentes, sendo impraticável adaptar estas interfaces criando subclasses para cada uma.

Esse padrão apresenta vários pontos a serem considerados, são eles: (GAMMA et al., 2000)

- Verificar quantas adaptações o Adaptador poderá fazer;
- Adaptadores conectáveis, isto é, a adaptação de interfaces permite incorporar uma classe a sistemas existentes que podem estar esperando interfaces diferentes para a classe;
- Utilização de adaptadores de dois sentidos para fornecer transparências. Um problema relevante dos adaptadores, sucede-se do fato de que eles não são transparentes para todos os clientes. Um objeto adaptado não disponibiliza a interface do objeto original, por isso não pode ser usado onde o original o for. Adaptadores de dois sentidos (*two-way adapters*) podem fornecer essa transparência. Eles são úteis quando dois clientes diferentes necessitam ver um objeto de forma diferente.

2.5.7 Ponte *(Bridge)*

Esse padrão coloca a implementação e a abstração em hierarquias separadas de classes para permitir a variação das duas ao mesmo tempo, e permite também que ambos os lados da hierarquia possam variar independentemente. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

Quando uma abstração pode ter várias implementações possíveis, a maneira usual de implementá-las é usando a herança. Uma classe abstrata define a interface para a abstração, e subclasses concretas a implementam de formas diferentes. Mas esta solução normalmente não é muito flexível. A herança liga uma implementação à abstração permanentemente, o que torna difícil modificar, aumentar e reutilizar abstrações e implementações independentemente. Para isso, o padrão Ponte trata desses problemas colocando a abstração e sua implementação em hierarquias de classes separadas. (GAMMA et al., 2000)

Esse padrão poderá ser utilizado quando: (GAMMA et al., 2000)

- Se deseja evitar um vínculo permanente entre uma abstração e sua implementação;
- Tanto as abstrações como suas implementações tiverem de ser extensíveis por meio de subclasses;
- Mudanças na implementação de uma abstração não deveriam ter impacto sobre os clientes;
- Se deseja ocultar completamente dos clientes a implementação de sua abstração;
- Se deseja compartilhar uma implementação entre múltiplos objetos.

As conseqüências de utilizar esse padrão são apresentadas a seguir: (GAMMA et al., 2000):

- Desacopla a interface da implementação;
- Extensibilidade melhorada;
- Ocultação de detalhes de implementação dos clientes.

2.5.8 Fábrica Abstrata *(Abstract Factory)*

Esse padrão permite a criação de famílias de objetos relacionados ou dependentes, através de uma única interface e sem que a classe concreta seja especificada.

O padrão Fábrica é útil para construir objetos individuais, sem que o cliente tenha conhecimento das classes específicas que serão instanciadas. Se for necessário criar um grupo combinado de tais objetos, então o padrão Fábrica Abstrata é o mais apropriado. (GUJ, 2002-2006)

O padrão Fábrica Abstrata está em um nível de abstração maior do que o padrão Método Fábrica. Pode-se usar este padrão quando se deseja retornar uma das muitas classes de objetos relacionadas, cada qual podendo retornar diferentes objetos se requisitado. Em outras palavras, o padrão Fábrica Abstrata é uma fábrica de objetos que retorna uma das várias fábricas. (GUJ, 2002-2006)

Esse padrão poderá ser utilizado quando: (GAMMA et al., 2000)

- Um sistema deve ser independente de como seus produtos são criados, compostos ou representados;
- Um sistema deve ser configurado como um produto de uma família de múltiplos produtos;
- Uma família de objetos-produto for projetada para ser usada em conjunto, e necessita garantir esta restrição;
- Quer-se fornecer uma biblioteca de classes de produtos e se quer revelar somente suas interfaces, não suas implementações.

Este padrão tem os seguintes benefícios e desvantagens: (GAMMA et al., 2000)

- Isola as classes concretas;
- Torna fácil a troca de famílias de produtos;
- Promove a harmonia entre produtos;
- É difícil de suportar novos tipos de produtos.

2.5.9 Construtor *(Builder)*

O Construtor permite a separação da construção de um objeto complexo da sua representação, de forma que o mesmo processo de construção possa criar diferentes representações. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

Esse padrão poderá ser utilizado quando: (GAMMA et al., 2000)

- O algoritmo para a criação de um objeto complexo deve ser independente das partes que compõem o objeto e de como são montadas;
- O processo de construção deve permitir diferentes representações para o objeto que é construído.

Este padrão tem como conseqüências: (GAMMA et al., 2000)

- Permite variar a representação interna de um produto;
- Isola o código para a construção e representação;
- Oferece um controle mais rigoroso sobre o processo de construção.

2.5.10 Protótipo *(Prototype)*

O padrão Protótipo permite que sejam criadas novas instâncias simplesmente copiando instâncias existentes. Um aspecto fundamental desse padrão é que o código-cliente pode gerar novas instâncias sem saber em qual classe específica elas são baseadas.

Esse padrão poderá ser utilizado quando um sistema for independente de como seus produtos são criados, compostos e representados, e também quando: (GAMMA et al., 2000)

- As classes instanciadas são especificadas em tempo de execução;
- Para evitar uma hierarquia de classes de fábricas paralelas;
- Quando as instâncias de uma classe puderem ter uma dentre poucas combinações diferentes de estados.

Esse padrão oculta as classes concretas de produtos do cliente, reduzindo assim o número de nomes que os clientes necessitam saber. Além disso, esse padrão permite trabalhar com classes específicas de uma aplicação sem necessidade de modificá-las. Além disso, existem outros benefícios na utilização desse padrão, tais como: (GAMMA et al., 2000)

- Acrescenta e remove produtos em tempo de execução;
- Especifica novos objetos pela variação de valores;
- Especifica novos objetos pela variação de estrutura;
- Reduz o número de subclasses;
- Configura uma aplicação com classes dinamicamente.

3 A LINGUAGEM DE PADRÕES APLICADOS NO ESTUDO DE CASO

Este capítulo apresenta a meta-linguagem de escrita de padrões de (MESZAROS; DOBLE, 1998) e um idioma de padrões, a ser utilizado no estudo de caso do Capítulo 4.

3.1 Utilização de uma Linguagem de Padrões

Para que uma solução seja bem compreendida, é importante seguir uma linguagem de padrões para a escrita de padrões. No presente trabalho utiliza-se as diretivas para uma linguagem de escrita de padrões de (MESZAROS; DOBLE, 1998), que sintetizam as "melhores práticas" para a redação de soluções de software.

Os padrões de (MESZAROS; DOBLE, 1998) surgiram das experiências dos revisores do PLoP-95[1]. Depois de revisarem um grande número de padrões e de linguagem de padrões, os revisores começaram a desenvolver uma coleção de técnicas de linguagem de padrões para superar problemas que ocorrem periodicamente.

(MESZAROS; DOBLE, 1998) organizaram o conhecimento mais difundido sobre a escrita de padrões e linguagens de padrões em uma meta-linguagem para a escrita de padrões. Ou seja, a abordagem foi utilizada para definir a própria abordagem.

No presente trabalho não serão apresentados todos os detalhes dessa meta-linguagem, nem tampouco uma análise mais detalhada das decisões que nortearam a escolha do idioma de padrões utilizado no estudo de caso do Capítulo 4. Obtem-se a alguns critérios pragmáticos seguidos por (MESZAROS; DOBLE, 1998) para se ter

[1] Proceedings of PLoP-95 - "Pattern Languages of Program Design" publicado pela Addison-Wesley em 1996.

o mínimo de inteligibilidade na descrição de uma solução que aplique padrões de projeto.

De todo modo, inicialmente, uma estratégia básica é que ao se procurar uma solução para um problema particular, deve-se recorrer ao resumo do problema e da solução, para verificar se existe um problema semelhante ao que se está querendo resolver. Uma vez determinado que existe um padrão para solucionar o problema de interesse, é importante verificar a seção de forças e identificar se este padrão é aplicável à sua situação. Define-se forças como considerações que devem ser levadas em conta ao escolher uma solução para um problema.

Outra observação é que as técnicas para o uso de linguagens de padrões estão sendo melhoradas continuamente. Não há uma única forma correta para escrever uma linguagem de padrões. Os padrões descritos no Capítulo 4 devem ser tratados como sugestões a serem adotadas para descrever uma solução utilizando padrões de projeto. Tais sugestões foram utilizadas no estudo de caso em questão, seguindo um idioma de padrões descrita na Seção 3.2, para que o registro dos problemas e soluções fosse bem documentados e compreensível.

3.2 A linguagem de padrões utilizada

As linguagens de padrões devem ser utilizadas para deixar a redação das soluções utilizando padrões mais compreensíveis e estruturadas. No presente trabalho será aplicado um idioma de padrões derivado da meta-linguagem de escrita de padrões de (MESZAROS; DOBLE, 1998), para ajudar na estruturação e na documentação das soluções dos problemas utilizando padrões de projeto.

Salvo menção em comentário, todo o texto do capítulo está baseado em (MESZAROS; DOBLE, 1998).

A meta-linguagem de padrões agrupa 5 seções, conforme apresentado na Figura 1. As soluções, utilizando padrões do estudo de caso apresentado no capítulo 4, seguem as orientações descritas neste capítulo.

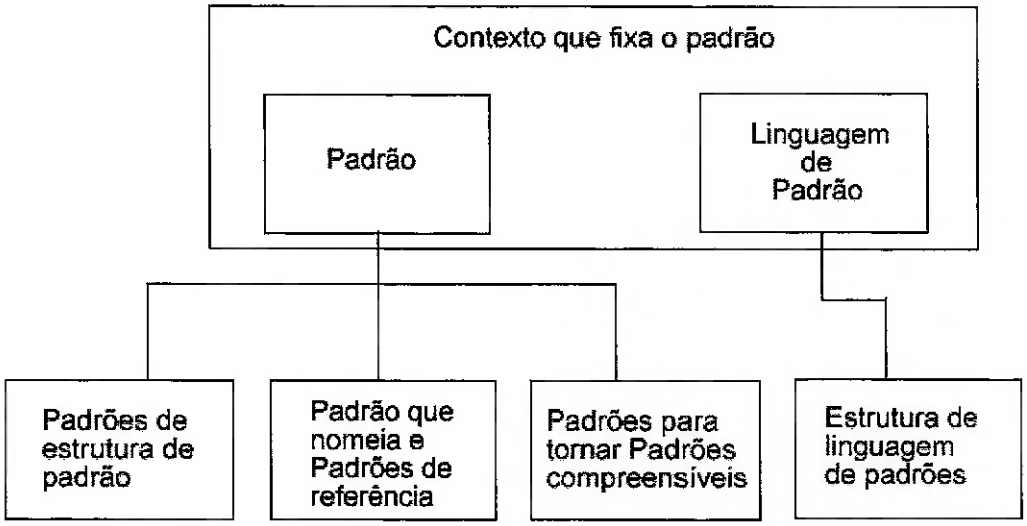

Figura 1 – Estrutura das linguagens de padrões (MESZAROS; DOBLE, 1998)

A seção **Contexto que fixa padrão**, introduz o conceito de um padrão (uma solução para um problema em um contexto) e uma linguagem de Padrão (coleções de padrões que são relacionados um ao outro em virtude de resolver os mesmos problemas ou partes de uma solução de um problema maior), de forma que eles possam ser usados ao longo desta linguagem de padrões.

A seção **Padrões de estrutura de padrão**, contêm padrões que descrevem o conteúdo desejado e a estrutura de padrões individuais.

A seção **Padrão que nomeia e padrões de referência**, contêm padrões que descrevem técnicas para nomear padrões e para referenciar outros padrões.

A seção **Padrões para tornar padrões compreensíveis**, contêm padrões que captam técnicas para fazer padrões e a linguagem de padrões mais fáceis de serem compreendidos.

A seção **Estrutura de linguagem de padrões**, contêm padrões que descrevem um conteúdo desejado e a estrutura das linguagens de padrões.

De modo pragmático, o presente trabalho enfoca apenas partes das seções, Padrões de estrutura de padrão, Estrutura de linguagem de padrões e Padrões para tornar padrões compreensíveis, para definir o idioma de padrões utilizado.

A primeira porque contêm os elementos mandatórios da estrutura de informação para descrever um padrão. A segunda, porque o objetivo principal do trabalho é a descoberta e a escolha de padrões de projeto apropriados para solucionar problemas de projetos descritos no estudo de caso do Capítulo 4. Desse modo, precisa-se de uma estrutura de descrição organizada e padronizada da argumentação da seleção de soluções candidatas e da escolha da sugestão de seleção. A terceira, porque a descrição mencionada, normalmente dissertativa e contendo vários detalhes técnicos, requer algumas balizas para torná-la compreensível.

A Tabela 1, mostra os padrões correspondentes da seção Estrutura de linguagem de padrões.

Tabela 1 – Estrutura de linguagem de padrões (MESZAROS; DOBLE, 1998)

Problema	Solução	Nome do Padrão
Como fazer para um leitor escolher facilmente padrões úteis que resolvam um problema?	Prover uma tabela que resuma todos os padrões, inclusive com uma breve descrição do problema de cada Padrão e a solução correspondente.	Sumário de Problemas e Soluções
Como oferecer ao leitor uma avaliação de um conjunto de Padrões?	Resumir o Idioma de Padrão na Introdução.	Resumo da Linguagem de Padrões
Como fazer com que os leitores escolham uma das soluções alternativas?	Quando vários padrões resolverem o mesmo problema, mostre ao leitor que há várias soluções a este problema.	Problemas Comuns Realçados
Como ajudar o leitor para que ele entenda como os padrões individuais estão englobados na estrutura	Inclua títulos de Padrão com os números da seção hierárquica, em que a seção que numera hierarquia compara a estrutura	Estrutura para Transportar Títulos

global de linguagens?	de linguagem	
Como fazer com que o leitor coloque uma linguagem de padrão em prática?	Utilizar um único exemplo em todos as linguagens de padrões.	Exemplo Único
Como deixar as terminologias pouco conhecidas mais claras em uma linguagem de Padrão sem interromper o fluxo do Padrão?	Prover um glossário de todas as condições que podem ser pouco conhecidas.	Glossário

Para possibilitar que o estudo de caso tenha uma descrição estruturada e padronizada, serão utilizados dessa seção os padrões:

- **Padrão Sumário de Problemas e Soluções:** Será apresentado um sumário de todos os problemas e soluções junto com os padrões candidatos.
- **Padrão Resumo da Linguagem de Padrões:** Será apresentado o resumo do Idioma de Padrão utilizado no estudo de caso.
- **Padrão Glossário:** Está apresentado no início do presente trabalho o glossário com todas as palavras não populares para a melhor compreensão dos problemas e soluções tratados.

A Tabela 2, mostra os padrões correspondentes à seção Padrões para tornar padrões compreensíveis.

Tabela 2 – Padrões para tornar padrões compreensíveis (MESZAROS; DOBLE, 1998)

Problema	Solução	Nome do Padrão
Como fazer um Padrão de software suficientemente claro e não ambígüo para facilitar a implementação direta?	Prover uma ou mais implementações, codificando exemplos escritos em uma linguagem de programação prevalecente e ilustrando os conceitos do Padrão.	Amostras de código
Como assegurar que a essência de um Padrão de software pode ser entendida por todo público-alvo?	Assegurar que o padrão possa ser capaz de comunicar seus conceitos essenciais mesmo se os exemplos de código fossem apagados.	Amostras de código como gratificação
Como assegurar que diagramas sejam facilmente compreendidos por todo público-alvo?	Usar notações que sejam compreendidos por membros do público-alvo. Se estiver usando uma notação padronizada, disponibilizar uma referência ao padrão utilizado.	Notações inteligíveis
Como assegurar que um padrão seja facilmente compreendido pela audiência planejada?	Identificar claramente o público-alvo para o qual se deseja comunicar uma solução. Mantenha esse público-alvo em mente enquanto estiver escrevendo o padrão. Teste o padrão com representantes desse público.	Público-alvo definido
Como maximizar a probabilidade de o leitor entender um Padrão?	Usar terminologia direcionada ao seu público-alvo. Usar somente termos que um membro típico do seu público-alvo deva entender. Testar esta terminologia com membros representativos do seu público-alvo.	Terminologia apropriada para o público-alvo

Para tornar a redação das soluções utilizando padrões compreensíveis, serão utilizados dessa seção os padrões:

- **Padrão Notações inteligíveis:** Será utilizada a UML (Unified Modeling Language), para a melhor compreensão do público-alvo.

A técnica de especificação de padrão descrita por (FRANCE et al., 2004) apóia a especificação de soluções de padrão de projeto escritas em UML. A UML é considerada o principal padrão para modelagem de projetos orientados a objetos, e há um grande crescimento de usuários que utilizam a UML para modelagem de seus projetos. (FRANCE et al., 2004)

- **Padrão Público-alvo definido:** O público-alvo do estudo de caso (Capítulo 4) são funcionários recém-contratados pela empresa que gerencia a Plataforma Alpha, com experiência em projetos na área de tecnologia de software. Fazem também parte do público algo, leitores interessados na solução de alguns problemas típicos de projetos de Portais de venda de conteúdos para dispositivos móveis.

- **Padrão Terminologia apropriada para o público-alvo:** Serão utilizados termos que um funcionário com alguma experiência na área tecnológica entenda.

Finalmente, para solucionar o problema da cobertura das informações, necessárias para descrever um padrão, a solução proposta por (MESZAROS; DOBLE, 1998) é a de incluir os seguintes elementos: Nome do Padrão, Problema, Contexto, Forças e Solução. Seguem as definições dos elementos.

- Nome: Nome pelo qual o problema pode ser referenciado;

- Problema: O problema específico que pode ser resolvido;

- Contexto: Descrito freqüentemente por uma "situação";

- Força: Considerações que devem ser levadas em conta ao escolher uma solução a um problema;

- Solução: A solução proposta para o problema. Muitos problemas podem ter mais que uma solução, e a identificação de uma solução para um problema é afetada pelo contexto em qual o problema acontece. Cada solução leva em conta certas forças.

Em resumo, as decisões anteriores definem a seguinte estrutura geral do Idioma para os padrões apresentados no estudo de caso.

Padrão Sumário de Problemas e Soluções

- Sumário dos problemas e os padrões candidatos a solução.

Padrão Resumo da Linguagem de Padrões

- Estrutura baseada nas seguintes seções:
  - Nome;
  - Problema;
  - Contexto;
  - Força;
  - Solução.

Padrão Glossário

- Vide seção Glossário do presente trabalho.

Padrão Notações Inteligíveis

- Utilização da UML nos diagramas da descrição do padrão.

Padrão Público-Alvo Definido

- Pessoas com experiência em projeto de software.

Padrão Terminologia Apropriada para o Público-Alvo

- Terminologia típica de projetos de software, em descrições que contextualizam detalhes da tecnologia de dispositivos móveis.

4 ESTUDO DE CASO - APLICANDO SOLUÇÕES COM PADRÕES DE PROJETO EM UM PORTAL PARA DISPOSITIVOS MÓVEIS

Este capítulo apresenta um estudo de caso de uma plataforma tecnológica de suporte a um Portal para dispositivos móveis denominada Plataforma Alpha. Analisa-se como os padrões de projeto oferecem soluções para os problemas enfrentados por essa Plataforma, utilizando-se o idioma de padrões apresentado na seção 3.2. Siglas e termos específicos do domínio de dispositivos móveis podem ser encontrados em lista de abreviaturas e siglas, e no glossário do presente trabalho.

No estudo, após análise dos padrões de projeto, identificou-se que os padrões: Cadeia de Responsabilidades, Procurador de Proteção, Decorador, Procurador de Cache, Ponte e Construtor se adequariam melhor na resolução dos problemas levantados por esta Plataforma.

4.1 Descrição Geral do Estudo de Caso: Plataforma Alpha

A Plataforma Alpha é uma solução de serviços para entrega de conteúdo digital em dispositivos móveis. É uma plataforma projetada para entregar aos usuários finais grandes variedades de conteúdo digital. Possui um Portal que disponibiliza diversos produtos, tais como: papel de parede, ringtone monofônico, ringtone polifônico, MP3, jogo, vídeo, entre outros.

O Acesso a esses Portais pode ser feito através das interfaces WEB, WAP e *SimBrowsing*, cujas funcionalidades básicas são a navegação entre as categorias de conteúdo, a busca e a compra de conteúdos. Para cada uma dessas interfaces existe uma linguagem própria utilizada na camada de apresentação.

A Tabela 3, apresenta a relação da interface de acesso ao Portal com a linguagem utilizada na camada de apresentação.

Tabela 3 – Relação Interface com linguagem utilizada

Interface	Linguagem
WEB	HTML
WAP1	WML
WAP2	XHTML MP
SIMBrowsing	WIG WML

Os recursos de apresentação de cada interface variam. A interface WEB em geral é mais flexível e complexa. Pode-se utilizar imagens, sons e mesmo aplicações executadas no navegador do cliente. Por outro lado, a tecnologia *SimBrowsing* é composta unicamente por texto.

O conteúdo vendido deve ser compatível com o dispositivo que o receberá, pois vários modelos de aparelhos celulares não têm suporte para determinados produtos.

O conteúdo é entregue de duas maneiras: através de mensagens de texto SMS (*ringtones* monofônicos, por exemplo) ou por meio do *download* direto do item via WAP (papel de parede, por exemplo).

A Figura 2 apresenta o sistema e suas fronteiras.

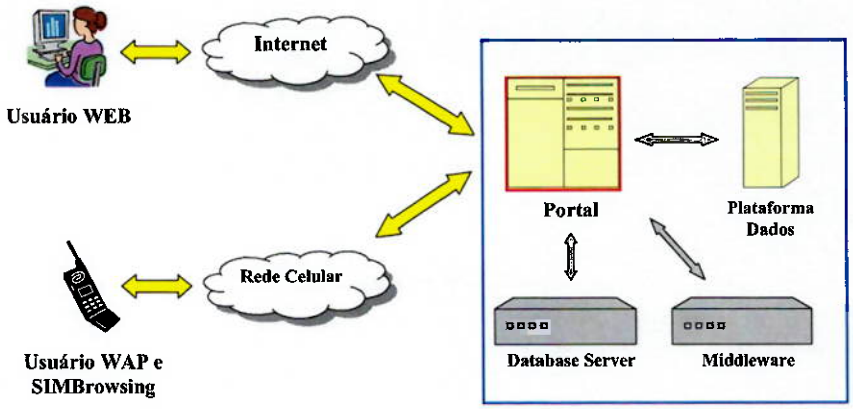

Figura 2 - Plataforma Alpha e suas Fronteiras

O Portal Alpha utiliza os serviços da Plataforma de Dados para o gerenciamento da estrutura de categorias na qual o conteúdo está distribuído, para o gerenciamento das regras de compatibilidade de conteúdo por aparelho celular. O Portal interage com a Plataforma de Dados, utilizando sua API para prover os serviços oferecidos aos usuários.

A estrutura de categorias organiza categorias que serão oferecidas para o usuário navegar até a escolha de um item, que deseja adquirir. A Figura 3 apresenta um exemplo da estrutura de categorias do Portal Alpha.

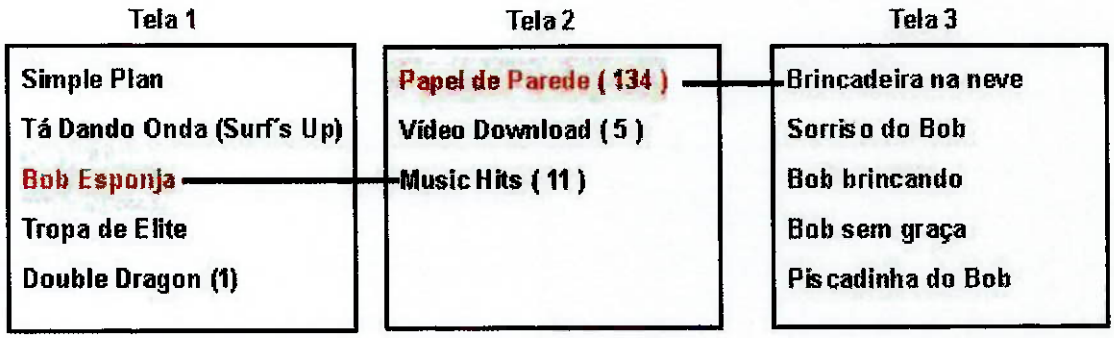

Figura 3 – Exemplo da Estrutura de Categorias apresentada no Portal Alpha

A Tela 1 apresenta a lista de *links* de categorias que serão oferecidas ao usuário. Ao ser acionado o *link* Bob Esponja, será exibida a Tela 2 com a lista de *links* de

categorias finais (categorias finais são categorias que apresentam abaixo dela apenas itens). Ao ser acionado o *link* Papel de Parede, será exibida a Tela 3 com a lista de itens.

O Portal Alpha utiliza os serviços do *Middleware* como interface única para a comunicação com as operadoras de telefonia.

O Portal Alpha utiliza um servidor de banco de dados para armazenar informações que a Plataforma de dados não possui. O Portal utilizará os dados cadastrados no banco de dados para permitir o acesso do usuário ao conteúdo, apresentando-os conforme a sua região (regionalização de conteúdo), tipo de dispositivo usado no acesso. Há também a adaptação da apresentação das páginas conforme a linguagem de cada interface. Permite também adaptações da apresentação conforme o modelo de aparelho usado no acesso ao Portal, possibilitando que todos os usuários tenham a mesma qualidade visual no acesso ao Portal.

O Portal tem como suas principais características:

- A aplicação modular e distribuída, com forte utilização de APIs para integração de componentes.
- Acesso a partir de interfaces SMS, *SimBrowsing*, WAP e WEB utilizando a mesma aplicação. Inclusive a interface de SAC utiliza a mesma aplicação. A interface de SAC, possui informações completas das transações efetuadas pelos clientes e tem como objetivo disponibilizar as funcionalidades de visualização de extrato de compras, autorização de retentativa e estorno, caso tenha ocorrido algum problema ao efetuar a compra do conteúdo.
- Reaproveitamento da aplicação principal (núcleo) na construção de outros Portais.
- Utilização de *cache* para aumento da velocidade de acesso.
- Camada de apresentação configurável conforme o dispositivo.
- As utilizações de APIs asseguram que a conectividade de um parceiro à rede da operadora possa ser estabelecida com agilidade.

Para que as características principais deste estudo de caso possam ser resolvidas utilizando padrões de projeto, será seguido a estrutura sugestiva que será

apresentada na próxima seção, para descrever uma solução mais compreensível e estruturada.

4.2 Soluções dos Problemas da Plataforma Alpha empregando um Idioma de Padrões

Esta seção apresenta as soluções dos problemas empregando o Idioma de Padrões definido na seção 3.2.

4.2.1 Sumário dos Problemas da Plataforma Alpha

A Tabela 4, sumariza todos os problemas enfrentados pela Plataforma Alpha, e seus respectivos padrões candidatos para resolver esses problemas.

Tabela 4 – Sumário dos problemas da Plataforma Alpha

Problema	Padrões Candidatos	Seção do Capítulo
Adaptação de *layout* para cada modelo de aparelho celular	• Comando • Cadeia de Responsabilidades	4.2
Controle de Acesso	• Procurador • Decorador	4.3
Recuperação e Integração de Conteúdo	• Decorador • Visitante	4.4
Histórico de Compra	• Decorador	4.5
Cache de Conteúdo	• Procurador	4.6

Processo de Compra	• Adaptador • Ponte	4.7
Envio de Comandos para o *Middleware*	• Fábrica Abstrata • Construtor • Protótipo	4.8

4.2.2 Adaptação de layout para cada modelo de aparelho celular

Problema:

Atualmente existem diversos modelos de celulares disponíveis no mercado. Cada modelo possui suas características, tais como tamanho de tela, configurações, navegador entre outras. São necessárias adaptações de apresentação do *layout* conforme o modelo de aparelho celular usado no acesso ao Portal WAP. Essas adaptações têm o objetivo de apresentar aos usuários finais, independentemente do modelo de celular utilizado, a qualidade de visualização do *layout* no Portal WAP.

Contexto:

A grande quantidade de modelos de aparelho celular disponíveis no mercado faz com que haja a preocupação de como o *layout* do Portal será visualizado em cada aparelho, pois cada um tem a sua limitação. Um dos maiores problemas é que a maioria dos aparelhos tem a dimensão de tela diferente e com isso seria importante a utilização das medidas de altura e largura da tela em porcentagem, para maior flexibilidade de adaptação do *layout* na tela. Entretanto, alguns aparelhos não permitem medidas em porcentagem, apenas em *pixel*. Com isso, se for colocado uma medida fixa em *pixels,* os aparelhos que possuem telas menores ou maiores do que o tamanho definido não ficariam com a mesma qualidade visual no Portal. Outro problema é que alguns aparelhos aparecem com a fonte muito pequena, outros com a fonte muito grande também distorcendo o *layout.* Sendo assim, é necessário adaptar o *layout*, tratando as requisições ao Portal para que os *layouts* sejam

adaptados de acordo com a regra do celular requisitante, de modo que cada aparelho tenha sua qualidade visual garantida.

Além desses fatores, um outro problema muito relevante para a necessidade da adaptação do *layout* é que existem duas versões de interface WAP: a versão 1.0, desenvolvida na linguagem WML, e a versão 2.0, desenvolvida na linguagem XHTML. Ambas são linguagens de programação baseadas no padrão XML sendo parecidas com a linguagem HTML. A versão 2.0 apresenta melhorias na apresentação de conteúdos através de suporte às Folhas de Estilo em Cascata ou *Cascading Styles Sheets* (CSS).

Devido aos problemas identificados, é necessária a adaptação do *layout* ao modelo do aparelho celular utilizado, como também a modificação com flexibilidade da versão exibida na interface acessada pelo usuário, sem precisar alterar a aplicação.

Forças:

- Os micro-navegadores utilizam linguagens diferentes;
- Aparelhos celulares possuem telas de tamanhos diferentes, fontes de tamanhos diferentes, entre outras características;

Solução:

Uma solução possível seria identificar o modelo do aparelho celular e criar um conjunto de páginas para serem exibidas naquele aparelho, isto é, criar as páginas com as configurações adequadas conforme as limitações do modelo do aparelho. Porém, essa solução se torna inviável perto das centenas de modelos existentes no mercado, e também porque muitas vezes o que fica ruim no *layout* de um aparelho é apenas um pequeno pedaço de página, não sendo necessário portanto a criação de todas as páginas novamente.

Para a interface WAP2 existe a flexibilidade de utilizar a configuração do *layout* nas CSS. Com isso, fica mais fácil utilizar um interceptador para apenas modificar a CSS conforme a necessidade. No entanto, existem aparelhos que não conseguem

interpretar a CSS, e a configuração de estilo tem que ser desenvolvida no próprio código.

Uma solução é adaptar dinamicamente a resposta de uma solicitação. Nesse caso seria a adaptação da página propriamente dita, para retorná-la adaptada de acordo com o modelo do aparelho celular que o usuário estiver acessando, mas de forma independente do atendimento da requisição em si, como mostrar itens, categorias, o resultado da busca, etc. Isso possibilitaria que o Portal tornasse flexível o acréscimo de aparelhos conforme a demanda do mercado, permitindo assim sua escalabilidade.

Foram identificados dois padrões, através de suas intenções, que poderiam ajudar a resolver esse problema. São eles: o padrão Comando e o padrão Cadeia de Responsabilidades.

Como visto no capítulo 2, o padrão Comando tem como intenção encapsular uma solicitação como um objeto, enfileirar ou fazer o registro de solicitações e dar suporte a operações que possam ser desfeitas. O padrão permite a objetos fazer solicitações de objetos não especificados, transformando a própria solicitação num objeto. O objeto pode ser armazenado e passado para outros objetos. A chave desse padrão é uma classe de Comando abstrata, a qual declara uma interface para execução de operações. As subclasses concretas de Comando especificam um par receptor-ação através do armazenamento do receptor como uma variável de instância e pela implementação de um método de execução para invocar a solicitação, isto é, o receptor tem o conhecimento necessário para poder executar a solicitação. (GAMMA et al., 2000)

Como também visto no capítulo 2, o padrão Cadeia de Responsabilidades intenciona evitar o acoplamento do remetente de uma solicitação ao seu receptor, ao dar a mais de um objeto a oportunidade de tratar a solicitação. Encadeia os objetos receptores, passando a solicitação ao longo da cadeia até que um objeto a trate. O objeto que trata a solicitação deve ser escolhido automaticamente. O conjunto de objetos que pode tratar uma solicitação deve ser especificado dinamicamente. (GAMMA et al., 2000)

Para o problema em questão a melhor solução é utilizar o padrão Cadeia de Responsabilidades, pois este faz exatamente o que é necessário para resolver este problema. Sua intenção é dar a mais de um objeto a possibilidade de processar uma solicitação, como se fosse um filtro.

Para conseguir adaptar todos os modelos das operadoras deverá ser criada uma cadeia de objetos que recebe a solicitação examinando-a seqüencialmente. Após examinar a solicitação, cada objeto pode processá-la ou passá-la adiante para o próximo objeto na cadeia. A solicitação será uma página padrão (WML), na qual a página é modificada dinamicamente para cada modelo de aparelho realizado por um dos objetos da cadeia. Para reconhecer o modelo do aparelho, capturam-se os parâmetros que o celular envia através do cabeçalho da requisição da solicitação. Nesses parâmetros existe um parâmetro chamado *user-agent*, contendo o valor único universal de identificação de cada aparelho celular. Através de um mapeamento cadastrado é possível definir a regra definida para cada aparelho. A Figura 4, apresenta uma solução para o problema de acordo com o padrão Cadeia de Responsabilidades. A regra será cadastrada utilizando a linguagem XSLT (transformação com XSL). Primeiramente, a requisição entra na classe *IdentifyModelHandler* da cadeia que, por sua vez, recupera os parâmetros enviados pelo cabeçalho da requisição do aparelho celular. Pelo parâmetro *user-agent* é possível identificar qual é o modelo que está acessando o Portal. Em seguida, a requisição é encaminhada para o próximo sucessor, a classe *AdapterLayoutHandler*, que verifica se o aparelho é WAP1 ou WAP2, e se necessário aplica a regra definida para o aparelho. Todos os aparelhos que não estão cadastrados como WAP2 são obrigatoriamente WAP1. Assim que é efetuada a adaptação, a requisição é encaminhada para o próximo sucessor, o *RequestRunHandler*, o qual retorna a página já adaptada para o usuário. Nesse caso, a ordem das classes na cadeia é importante, pois primeiramente tem que ser identificado o aparelho para assim adaptar o *layout*.

Para repassar a solicitação ao longo da cadeia, e para garantir que os receptores permaneçam implícitos, cada objeto da cadeia compartilha uma interface comum para o tratamento de solicitações para ter acesso ao seu sucessor na cadeia. O sistema define uma interface *PortalViewHandler* com uma operação chamada *processRequest*. Todos os objetos da cadeia realizam essa interface.

As classes *IdentifyModelHandler*, *AdapterLayoutHandler* e a *RequestRunHandler* sobrescrevem a operação *processRequest* para tratar a página conforme suas circustâncias.

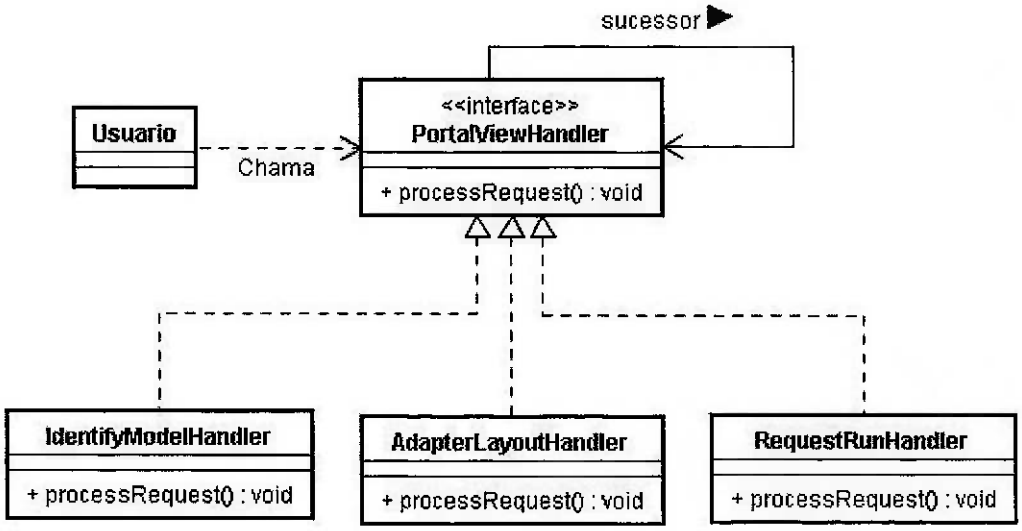

Figura 4 – Diagrama de Classe da solução utilizando o padrão Cadeia de Responsabilidades

4.2.3 Controle de Acesso

Problema:

O acesso à interface WAP deve ser protegida, pois somente servidores com determinados IPs podem ter acesso a ela.

Contexto:

Nos dias de hoje devido ao crescimento da tecnologia, existem diversas maneiras de se ter acesso a um *site* WAP. Ele pode ser acessado através de simuladores, como da *OpenWave*, por exemplo, que permite simular a transferência das mensagens MMS usando um *gateway*.

Além desses simuladores, é possível montar um simulador no navegador *Firefox* com *plugins*. Mas isso é um elemento de preocupação, pois nos navegadores *WEB*, é possível copiar as imagens e também baixar os *ringtones*.

O Portal Alpha, vende justamente conteúdos de imagens, jogos, vídeos e *ringtones*. Assim, com esse tipo de simulador é possível baixar os itens sem pagar, pois é possível colocar um número qualquer de celular da operadora que o Portal permite a venda de conteúdo e assim baixá-lo desonestamente e, posteriormente, copiá-lo através de cabo de dados para o celular, fazendo com que, este numero de celular que foi informado no simulador pague pelos itens baixados.

Portanto, por causa dos simulares existentes no mercado, é necessário controlar os acessos através dos IPs dos *Gateways* das operadoras de aparelhos celulares, para que apenas esses IPs consigam acessar o Portal Alpha, possibilitando assim proteção ao Portal.

Forças:

- Apenas IPs dos *Gateways* das operadoras podem acessar o Portal.
- IPs podem ser incluídos ou removidos.

Solução:

Uma das soluções seria a configuração dos IPs no *firewall* para que este fizesse o bloqueio dos IPs. Porém, dessa forma, toda vez que fosse configurado um novo IP para acessar o Portal teria que ser solicitado ao administrador de rede para fazê-lo.

Além dos IPs das operadoras, também são configurados os IPs de saídas dos ambientes de desenvolvimento e teste, pois, quando os desenvolvedores precisam fazer os testes das funcionalidades do Portal Alpha, é muito mais simples e rápido testar usando simuladores, em vez dos próprios celulares. Os testes de *layout* só podem ser realizados nos próprios aparelhos, pois nos simuladores sempre aparece corretamente. Os testes pelos aparelhos ocorrem quando o desenvolvimento libera para a área de testes a funcionalidade pronta. Nesse momento, os testadores utilizam os aparelhos celulares para testarem.

Para facilitar a inclusão e exclusão de IPs para acessar o Portal, pode ser feita uma configuração, na qual todos os IPs configurados conseguem acessar o Portal.

Baseado na leitura das intenções dos padrões e nos padrões de finalidades semelhantes, foi possível identificar dois possíveis padrões que poderiam solucionar este problema, são eles: o padrão Procurador e o padrão Decorador.

Ambos os padrões descrevem como fornecer um nível de endereçamento indireto para um objeto, tanto de um objeto Procurador como de um objeto Decorador. Ambos mantêm uma referência para outro objeto para o qual repassam solicitações, porém eles têm finalidades diferentes. O Decorador acrescenta e remove um comportamento a uma classe, enquanto o Procurador controla o acesso a ela. Um Procurador de Proteção pode ser implementado exatamente como um Decorador. (GAMMA et al., 2000)

Como também visto no capítulo 2, no padrão Procurador, o objeto fornece as funcionalidades chaves e o Procurador fornece ou recusa o acesso ao objeto. O Procurador focaliza um relacionamento estaticamente entre o Procurador e seu objeto. Esse não é o caso do Decorador, que trata a situação em que a funcionalidade total de um objeto não pode ser determinada em tempo de compilação. (GAMMA et al., 2000)

Como visto no capítulo 2, podem-se ter vários tipos diferentes do padrão Procurador. No caso, será utilizado o Procurador de Proteção, pois têm como função controlar o acesso a um recurso com base em direitos de acesso.

A Figura 5, apresenta uma solução para o problema de acordo com o padrão Procurador de Proteção.

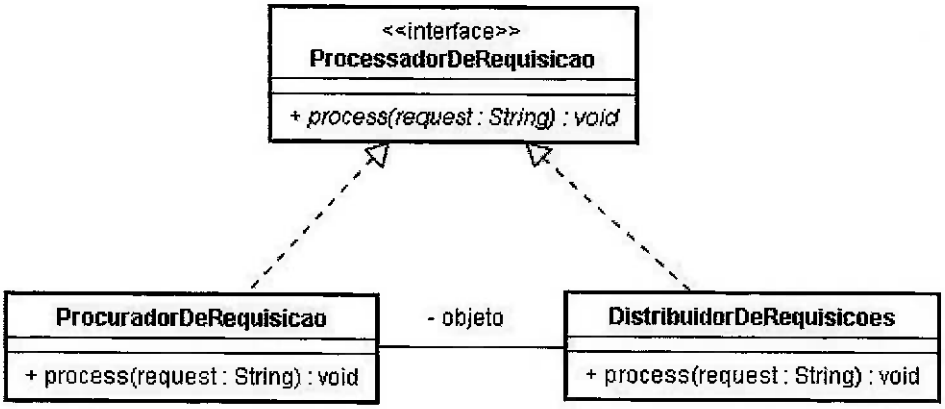

Figura 5 – Diagrama de Classes da solução utilizando o padrão Procurador de Proteção

Tanto a classe *DistribuidorDeRequisicoes* quanto a classe *ProcuradorDeRequisicao* implementam a classe *ProcessadorDeRequisicao*. Isso permite que qualquer requisição trate o Procurador como se fosse o objeto real. O *ProcuradorDeRequisicao* mantém uma referência para a classe *ProcessadorDeRequisicao*, que neste exemplo é o objeto real, para poder liberar as requisições sempre que o acesso a ele for permitido.

4.2.4 Recuperação e Integração de Conteúdo

Problema:

A exibição de conteúdos no Portal depende da região onde o usuário está localizado, do modelo do aparelho celular que o usuário está utilizando, da ordem como será exibido e de inclusão de conteúdos de *links* diretos para outros sites WAP. Tais dependências mudam conforme a interface do Portal que o usuário está acessando. Portanto, é importante a flexibilidade para acrescentar e remover essas dependências facilmente, sem afetar o resto da aplicação.

Contexto:

Ao disponibilizar a lista de conteúdos para ser exibido pelo Portal, são recuperados apenas os conteúdos que o modelo do aparelho celular utilizado aceita. São também recuperados apenas os conteúdos da região do número do celular do usuário, são incluídos os conteúdos de *links* diretos para outros sites WAP e é feita a ordenação dos itens em ordem alfabética ou aleatória.

Para recuperar os conteúdos de *links* diretos, deve-se ser conectado a outra base de dados e juntar-se os itens aos já recuperados pela base de dados da plataforma de dados. A implementação é feita na mesma classe, que recupera a lista de conteúdo retornado pela Plataforma de Dados. Também deve ser recuperada a lista de *links* diretos retornado pelo banco de dados do Portal. Isso causa um problema, pois nas interfaces WEB, *SimbBrowsing* e MO, não se pode incluir *links* diretos. Numa implementação existente, deve haver distinção das interfaces através de lógica de programação na própria classe, para que os *links* diretos não apareçam nestas interfaces.

Nem sempre a ordenação dos itens do Portal poderá ser feita antes da inclusão dos conteúdos de *links* diretos. Pode ser que sejam incluídos os conteúdos de *links* diretos e depois feita a ordenação. Para fazer tal mudança, deveriam ser feitas modificações no código já existente, podendo comprometer a aplicação que já funcionava.

Além disso, foi incluída uma nova plataforma de busca de conteúdos que substituirá o atual mecanismo de busca de conteúdos do Portal. Na nova plataforma não será mais necessário a ordenação dos itens, pois esta apresenta os itens na ordem de maior relevância. A nova plataforma possibilita que seja informado quais os campos dos itens e seus valores que queira dar maior relevância. A plataforma calcula e retorna os itens na ordem de maior relevância. É necessário portanto alterar a forma de implementação, para que a recuperação dos conteúdos ficasse mais robusta e flexível adequada à interface que o usuário está acessando.

A implementação efetuada atualmente antes da nova plataforma de busca é representada no diagrama de classes da Figura 6.

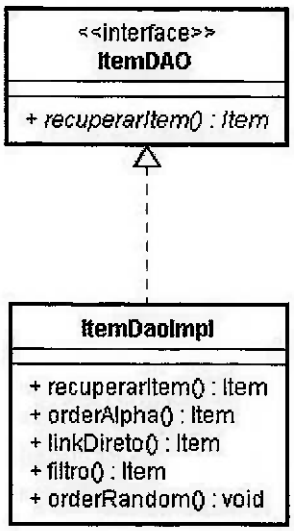

Figura 6 – Diagrama de Classe da solução sem padrão aplicado

A classe *ItemDAO* é uma interface implementada por *ItemDaoImpl*. A classe *ItemDaoImpl* possui o método *recuperarItem*(), que recupera os conteúdos para serem apresentados ao usuário utilizando os demais métodos, conforme a interface utilizada. O método *orderAlpha*() ordena os conteúdos do Portal em ordem alfabética. O método *linkDireto*() conecta-se a uma base de dados externa e adiciona os conteúdos de *link* direto aos conteúdos já recuperados pelo banco de dados do Portal. O método *filtro*() exclui alguns itens que não podem ser exibidos no Portal. O método *orderRandom*() ordena os conteúdos do Portal em ordem randômica. Toda a lógica de inclusão e remoção desses métodos, dependendo da interface utilizada, são efetuados nesta própria classe.

A manutenção do código a ser incluído ou removido é muito custoso, pois pode ter impacto em partes da aplicação que já utilizam esta classe.

Forças:

- Mudanças nas funcionalidades ou inclusão de novas funcionalidade forçará a alteração do código existente.
- Operações podem ser incluídas ou removidas.

- Interfaces de acesso aos itens são diferentes, nem sempre utilizam as mesmas funcionalidades.

Solução:

Uma possível solução seria o uso de heranças. Porém essa solução impede a combinação das funcionalidades em tempo de execução e um problema ainda maior é o grande número de classes que seriam criadas.

Pode-se acrescentar ordenação em ordem alfabética na classe *ItemDAO*, criando subclasses para obter uma classe *ItemDAOOrderAlpha*. Ou pode-se acrescentar os *links* diretos da mesma forma, para obter uma subclasse *ItemDaoLinkDireto*. Para se ter tanto a ordenação quanto a inclusão dos *links* diretos nesta ordem, pode-se criar uma subclasse *ItemDAOOrderAlphaLinkDireto.* Também pode-se primeiro incluir os *links* diretos para depois ordenar e, neste caso, teria que ser criado uma subclasse *ItemDAOLinkDiretoOrderAlpha*, e assim por diante. Essa solução levada ao extremo fará com que se acabe tendo uma classe para cada combinação possível de funcionalidades, uma solução que se tornará rapidamente intratável conforme a variedade de funcionalidades.

Para o problema é preciso acrescentar e remover funcionalidades dinamicamente sem ser necessário alterar o código já existente, prevenindo assim o comprometimento da aplicação. Os padrões Decorador e o Visitante poderiam resolver este problema, pois atendem a este propósito.

Como visto no capítulo 2, o padrão Decorador é projetado para possibilitar o acréscimo de responsabilidades a objetos sem usar subclasses. Ele evita a explosão de subclasses na tentativa de combinar estaticamente todas as combinações das funcionalidades, como na solução que utiliza a herança. O padrão Visitante representa uma operação a ser executada nos elementos de uma estrutura de objetos. Permite definir uma nova operação sem mudar as classes dos elementos sobre os quais opera. (GAMMA et al., 2000)

Para o problema, não é necessário utilizar o padrão Visitante, pois este vai além de acrescentar e remover funcionalidades a objetos. O padrão Visitante pode ser usado

quando uma estrutura de objetos contém muitas classes de objetos com interfaces diferentes e quando se deseja executar operações sobre estes objetos que dependem das suas classes concretas. (GAMMA et al., 2000) No problema não existem várias interfaces diferentes. Portanto, a melhor maneira de resolvê-lo é utilizar o padrão Decorador, pois este padrão capta o relacionamento de classe e objeto que dá o suporte ao acréscimo e remoção de responsabilidades a um objeto dinamicamente, em tempo de execução e, além disso, é mais simples de se usar. O Decorador envolve um conjunto de classes de Decoradores usadas para englobar componentes concretos. As classes de Decoradores espelham os tipos de componentes que eles decoram. Os Decoradores mudam o comportamento de seus componentes adicionando novos recursos antes e/ou depois de chamadas de métodos para o componente. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

Segue a solução utilizando o padrão Decorador. Uma vez que se saiba quem está sendo composto, pode-se tornar a funcionalidade em si um objeto. Isso possibilita alguns candidatos para a composição: a inclusão dos conteúdos do Portal, a ordenação alfabética, a ordenação aleatória, a inclusão dos conteúdos de *links* diretos e o filtro de conteúdo. A próxima etapa é quem compõe quem. Primeiramente seriam criadas as classes *ItemDAOPortal* e a *ItemDAOFast*, em que ambas implementam a classe *ItemDAO*. Depois, a classe *ItemDAOWithLinkAway,* que implementa *ItemDAO* e compõe uma classe *ItemDAO*. Em seguida, a classe *ItemDAOFiltered*, que implementa *ItemDAO* e compõe uma classe *ItemDAO.* Finalmente, a classe *ItemDAOSorted*, que implementa *ItemDAO* e compõe uma classe *ItemDAO*. O diagrama de classes da Figura 7 ilustra a aplicação do padrão Decorador.

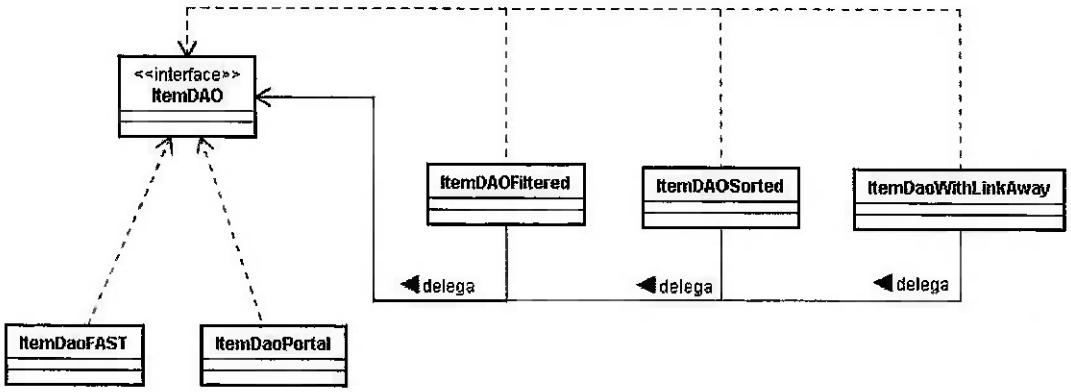

Figura 7 – Diagrama de Classe da solução utilizando o padrão Decorador

Uma vez definidas todas as classes, deve-se apenas informar a estrutura de composição no arquivo de configuração. Por exemplo: para a interface WAP, a classe *ItemDAOSorted* comporia com a classe *ItemDAOWithLinkAway* e esta, por sua vez, comporia com a classe *ItemDAOFiltered,* a qual comporia com a classe *ItemDAOFast.* Para a interface *Simbrowsing,* a classe *ItemDAOSorted* comporia com a classe *ItemDAOFiltered* e esta, por sua vez, comporia com a classe *ItemDAOPortal*, e assim por diante de acordo com a interface acessada. Portanto, implementando dessa maneira, é possível associar a interface *Simbrowsing* ao *ItemDaoPortal* configurando apenas a ordenação aleatória e o filtro de conteúdo, pois, como foi dito anteriormente, na interface *Simbrowsing* não podem aparecer os *links* diretos. Outro exemplo é: na interface WAP é possível associá-lo ao *ItemDAOFast* e configurando apenas a ordenação, o filtro de conteúdo e os *links* diretos, mantendo a flexibilidade.

4.2.5 Histórico de Compra

Problema:

O histórico de compras dos itens que foram comprados pelo Portal possui várias informações que são recuperadas de diferentes fontes, porém nas interfaces WAP e

WEB não são necessários exibir todas as informações existentes no histórico, apenas a interface de SAC deve possuir todas as informações do histórico. Portanto, é importante a flexibilidade de acrescentar e remover essas informações facilmente de acordo com a interface acessada, sem afetar o resto da aplicação.

Contexto:

Normalmente após a compra de um conteúdo, consulta-se o histórico de compras para verificar se o item comprado foi tarifado corretamente ou se o item deu erro, entre outras opções de consulta. O histórico de compra permite a visualização da data/hora que o item foi comprado, nome do item, nome do produto, valor do item, modelo do celular, canal que foi comprado o item como WAP, WEB, SB e SMS e o status da compra. O histórico de compra pode ser visualizado através da interface WAP, WEB e no SAC, sendo que as informações do histórico são recuperadas da plataforma de dados. No entanto, foi necessário incluir mais duas informações que não são armazenadas na plataforma de dados, por sua própria limitação. Para tanto, foi criado um banco de dados paralelo para armazenar essas duas informações, a saber: a quantidade de novas tentativas efetuadas para aquele item e se foi feito o estorno do item ao cliente.

Essas duas informações só são visíveis na interface de SAC, não é necessário que nas demais interfaces sejam exibidas essas funcionalidades, ou seja, não é necessário que as interface WAP e WEB se conectem ao banco de dados paralelo para obter essas informações, pois este acesso pode ser muito oneroso comprometendo a eficácia do Portal para o acesso do histórico de compra.

A implementação efetuada atualmente quando ainda não era utilizado um padrão de projeto é representada no diagrama de classes da Figura 8.

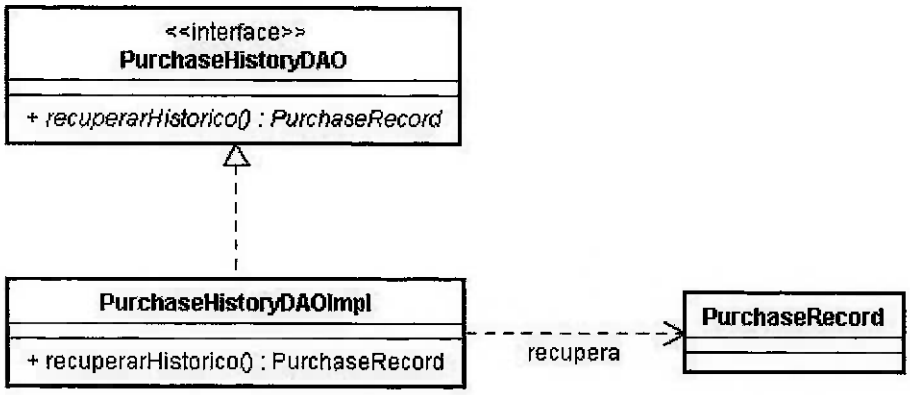

Figura 8 – Diagrama de Classe da solução sem padrão aplicado

A classe *PurchaseHistoryDAO* é uma interface implementada pela classe *PurchaseHistoryDAOImpl*. A classe *PurchaseHistoryDAOImpl* possui o método *recuperarHistorico()*, o qual recupera as informações de compra de um determinado número de celular. Conforme a interface utilizada, ela se conecta a uma base de dados paralela e adiciona as novas informações às informações já recuperados pela plataforma de dados. Toda a lógica de inclusão e remoção dessas informações dependentes da interface utilizada é efetuada no método *recuperarHistorico*() da classe *PurchaseHistoryDAOImpl*.

Para efetuar a manutenção do código ao ser incluído ou removido, novas informações devem ser alteradas no código já existente, Tais alterações podem ser onerosas, podendo impactar em partes da aplicação que já utilizam a classe *PurchaseHistoryDAOImpl*.

Forças:

- Inclusão de informações referentes ao histórico de compra que não estão na plataforma de dados forçará a alterar o código existente.
- Diversas interfaces acessam o histórico de compra, porém nem sempre elas compartilham as mesmas informações.

Solução:

O problema pode ser resolvido com o padrão Decorador, já utilizado na solução do problema do item 4.4 (Recuperação e Integração de Conteúdo), pois os problemas são semelhantes.

Segue a solução utilizando o padrão Decorador. Uma vez que se saiba quem está sendo composto, pode-se tornar a funcionalidade em si um objeto. Isso possibilita alguns candidatos para a composição: a inclusão das informações do histórico de compra retirados do banco de dados da plataforma de dados e as informações complementares retirados do banco de dados paralelo. A próxima etapa é resolver a composiçao. Primeiramente seriam criadas as classes *PurchaseHistoryDaoWithAction* e a *PurchaseHistoryDaoPortal*, em que ambas implementam a classe *PurchaseHistoryDao* e compõem uma classe *PurchaseHistoryDao.* O diagrama de classes da Figura 9 ilustra a aplicação do padrão Decorador.

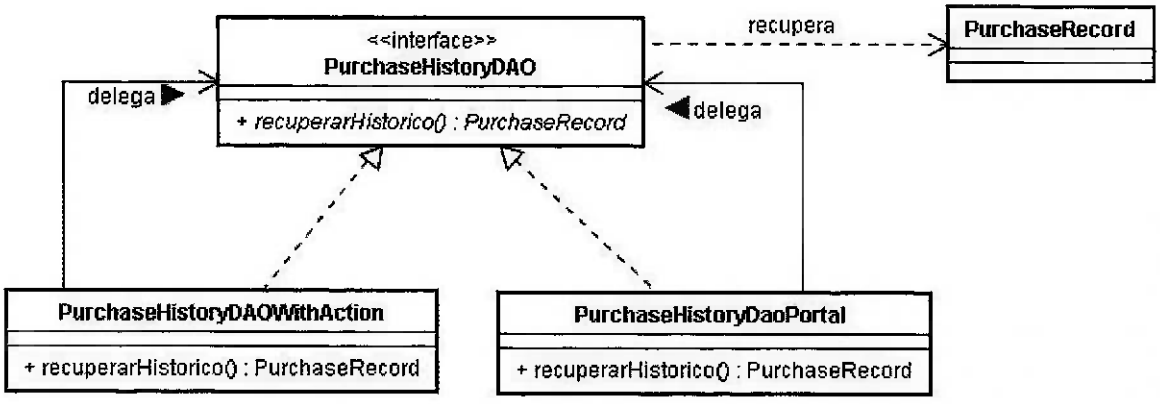

Figura 9 – Diagrama de Classes da solução utilizando o padrão Decorador

Uma vez definidas todas as classes, apenas tem-se informar a estrutura de composição no arquivo de configuração. Por exemplo: para a interface WAP e WEB só existiria a configuração da classe *PurchaseHistoryDaoPortal*, porém para a interface de SAC a classe *PurchaseHistoryDaoWithAction* comporia com a classe *PurchaseHistoryDaoPortal*. Portanto, implementando dessa maneira, é possível

associar a interface de SAC à inclusão das informações complementares que na interface WAP e WEB não devem aparecer.

4.2.6 Cache de Conteúdo

Problema:

Existem muitos conteúdos disponíveis no Portal Alpha para serem oferecidos ao usuário. Isso introduz uma lentidão na recuperação de todos os conteúdos para serem exibidos no aparelho celular.

Contexto:

O Portal oferece ao celular a venda de conteúdos de imagens, sons, vídeos, jogos, entre outros. Os conteúdos são cadastrados diariamente na plataforma de dados e, com o passar do tempo, aumenta a quantidade de conteúdos no banco de dados, aumentando assim o tempo de resposta na solicitação de recuperação dos conteúdos para serem apresentados no celular.

Os conteúdos são atualizados durante o dia, e apenas na madrugada são inseridos no servidor de conteúdos. Com isso não é necessário o acesso constante a esses servidores, utilizando assim um *cache* para minimizar o acesso. Os constantes acessos podem ser muito prejudiciais ao Portal, pois ocorrendo muitos acessos simultâneos, devido ao grande número de quantidade de celulares que possam estar acessando o Portal ao mesmo tempo, haveria sobrecarga do servidor.

A utilização do *cache* é essencial para o aumento da velocidade de acesso aos conteúdos da plataforma de dados.

No aparelho celular, a resposta a uma solicitação não pode ser demorada, pois o tempo de resposta do WAP *Gateway* de uma operadora celular é pequeno, ocorrendo muitas vezes o *timeout* da resposta a uma solicitação. Para que seja

melhorado o tempo do acesso aos conteúdos do Portal, é importante que sejam colocados em *cache*.

Forças:

- Muitos acessos simultâneos ao Portal.
- Há um volume muito grande de conteúdos oferecidos ao usuário.
- Tempo de resposta do *gateway* da operadora muito pequena.

Solução:

Para encontrar um padrão que controlasse o acesso ao banco de dados, isto é, que impedisse a recarga de um objeto já carregado anteriormente, foi possível identificar, baseado na leitura das intenções dos padrões, apenas o padrão Procurador como um padrão que poderia solucionar este problema.

Como visto no capítulo 2, no padrão Procurador o objeto fornece as funcionalidades chaves e o Procurador fornece ou recusa o acesso ao objeto. Esse padrão focaliza um relacionamento estático entre o Procurador e seu objeto. (GAMMA et al., 2000)

Para este problema pode-se usar o Procurador, pois este tem como uma de suas aplicabilidades carregar um objeto persistente para a memória quando ele for referenciado pela primeira vez. Dos tipos existentes de Procuradores, o mais adequado para este problema é o Procurador de *Cache*, pois este permite que múltiplos clientes compartilhem os resultados onerosos, possibilitando a redução do processamento ou da latência da rede. (FREEMAN Eric, FREEMAN, Elisabeth, 2005)

A Figura 10 ilustra o diagrama de classes utilizando o Procurador de *Cache*.

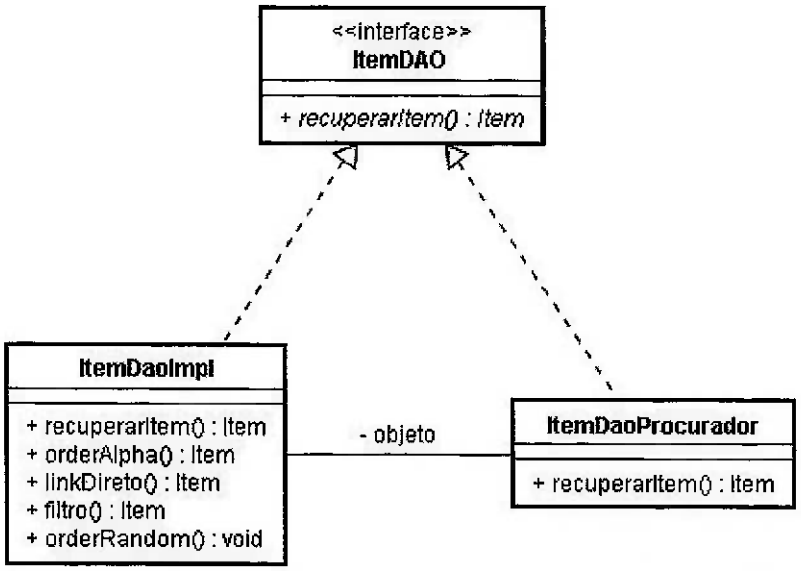

Figura 10 – Diagrama de Classes da solução utilizando o padrão Procurador Cache

Tanto a classe *ItemDaoImpl* quanto a classe *ItemDaoProcurador* implementam a classe *ItemDAO*. Isso permite que qualquer solicitação trate o Procurador como se fosse o objeto real. O *ItemDaoProcurador* mantém uma referência para a classe *ItemDaoImpl*, que, neste exemplo é o objeto real, para poder encaminhar as solicitações a ele sempre que o objeto real ainda não estiver sido carregado pelo *ItemDaoProcurador*. O objeto real geralmente é o objeto que executa a maior parte do trabalho e o Procurador controla o acesso a ele.

4.2.7 Processo de Compra

Problema:

No Portal é possível comprar conteúdos por quatro interfaces diferentes: WAP, WEB, *SimBrowsing* e MO, e para cada interface o processo de compra é diferente. Por outro lado, esses diferentes processos são sempre compostos pelas mesmas operações básicas sobre a plataforma de dados e o sistema de bilhetagem. Para que não seja necessário alterar todo o código a cada alteração do processo de

compra, ou quando quisermos mudar a plataforma de dados e bilhetagem, é necessário desacoplar uma abstração da sua implementação, de modo que as duas possam variar independentemente.

Contexto:

No Portal existem algumas maneiras de tratar a compra de um item dependendo da interface de compra.

Na interface WEB quando é solicitada a compra de um item monofônico, primeiramente é obtido o item, verificado se o número de telefone possui créditos, a seguir o usuário é cobrado pelo item e assim enviado o conteúdo ao celular. Na compra de um item de *download* é apenas obtido o item e enviado um *SMSLink* do item para o celular, para posteriormente ser feito o download. Nesse momento o item não é tarifado, só havendo tarifação no momento do *download*.

Na interface WEB, ao ser solicitada a compra de um item tanto monofônico quando de *download*, envia-se via SMS ao celular uma senha para que esta seja informada na página WEB e assim prosseguido com o processo de compra. Esta é uma segurança para que um usuário não compre um item informando um outro número de celular.

Na inteface WAP quando é solicitada a compra de um item monofônico o tratamento é idêntico à da interface WEB. Para conteúdo de *download*, o tratamento é diferente da interface WEB. Obtém-se o item, verifica-se se o número do celular possui créditos, cobra-se o usuário pelo item e só depois é feito o *download* do item.

Na interface de *SimBrowsing* e na interface MO o tratamento da compra de um item tanto monofônico quanto de *download* é idêntico ao da interface WEB. No entanto, a interface de *SimBrowsing* pode disparar o *download* automaticamente acionando o navegador WAP via *Launch browser*, mas isto apenas nos aparelhos que dão suporte a esta funcionalidade.

Forças:

- Dependendo da interface de compra a implementação do processo de compra é diferente.

Solução:

Quando uma abstração pode ter várias implementações possíveis, uma forma usual de acomodá-las é utilizando a herança. Uma classe abstrata define a interface para a abstração, e subclasses concretas a implementam de maneiras distintas. A herança liga uma implementação à abstração permanentemente, tornando difícil de modificar, aumentar e reutilizar abstrações e implementações independentemente. (GAMMA et al., 2000)

Baseado na leitura das intenções dos padrões e dos padrões com finalidades semelhantes, foi possível identificar dois possíveis padrões que poderiam solucionar este problema, a saber: o padrão Adaptador e o padrão Ponte.

A diferença entre esses padrões está nas suas intenções. Como visto no capítulo 2, o Adaptador focaliza na solução de incompatibilidade entre duas interfaces existentes. Ele não considera como as interfaces são implementadas, tampouco considera como podem evoluir independentemente. A Ponte estabelece uma ponte entre uma abstração e suas várias implementações. Ela fornece uma interface estável aos clientes ainda que permita variar as classes que a implementam. Ele também acomoda novas implementações à medida que o sistema evolui. (GAMMA et al., 2000)

Como visto no capítulo 2, o Adaptador e a Ponte são utilizados em pontos diferentes do ciclo de vida do software. Um Adaptador se torna necessário quando descobre-se que duas classes incompatíveis deveriam trabalhar em conjunto e o acoplamento não foi previsto. A Ponte compreende desde o início que uma abstração deve ter várias implementações e ambas podem evoluir independentemente. O padrão Adaptador faz as coisas funcionarem após elas terem sido projetadas. As Pontes as faz funcionarem antes que existam. (GAMMA et al., 2000)

Como já se sabe, no início do projeto, que deve-se ter várias implementações para uma abstração, portanto será utilizado o padrão Ponte.

A Figura 11, ilustra o diagrama de classe da solução utilizando o padrão Ponte.

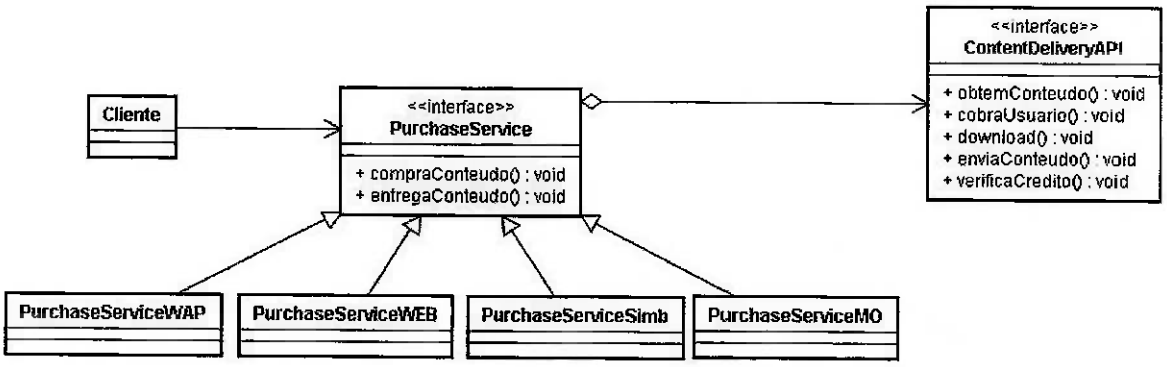

Figura 11 – Diagrama de classes da solução utilizando o padrão Ponte

A classe *PurchaseService* define a interface da abstração, mantendo uma referência para um objeto que a implementa. As classes *PurchaseServiceWAP*, *PurchaseServiceWEB*, *PurchaseServiceSimb* e *PurchaseServiceMO* implementam a interface definida pela classe *PurchaseService*. A classe *ContentDeliveryAPI* define a interface para as classes de implementação. Essa interface não precisa corresponder exatamente à interface de abstração. A interface de implementação fornece somente implementações primitivas e a interface de abstração define operações de nível mais alto baseadas nestas primitivas.

4.2.8 Envio de Comandos para o Middleware

Problema:

O *Middleware* é um sistema de software com interface única para a comunicação de operadoras de telefonia, através do qual o Portal se comunica para enviar mensagens SMS enviada a dispositivos móveis (mensagem MT), e mensagens SMS

originadas em dispositivos móveis (mensagem MO). Para isso é importante que o processo de envio seja independente da formação da consulta, pois como os comandos podem evoluir com o passar do tempo e são enviados de vários pontos do sistema, seria importante que nestes pontos só se saiba quais são os parâmetros e os comandos, e não como eles se formam, possibilitando maior escalabilidade na integração com o *Middleware*.

Contexto:

As operadoras de telefonia móvel têm constantemente a necessidade de integrar sua infra-estrutura e sistemas com os serviços oferecidos por seus parceiros comerciais, fornecedores de conteúdo SMS, mantendo controle sobre a segurança, aprovisionamento e bilhetagem.

A infra-estrutura SMSC é crítica para a operadora, o que faz do controle e da segurança requisitos vitais. O dinamismo das parcerias com empresas fornecedoras de conteúdos e serviços gera uma variedade de regras de negócio a serem consideradas na integração dos aplicativos, afetando principalmente o aprovisionamento e a bilhetagem.

O IbisWare é um *gateway* SMS que atua entre o SMSC da operadora e os provedores de conteúdo, roteando as mensagens destinadas aos aplicativos dos provedores de conteúdo segundo as regras de negócios da operadora, agregando características de bilhetagem, controle e rastreamento (*logs*).

Devido à sua arquitetura baseada em *frameworks* e em componentes com baixo acoplamento, novos componentes podem ser adicionados ao IbisWare para acrescentar novas funcionalidades ou adaptá-lo a novas necessidades. Seus principais componentes são:

- InterfaceMT. Responsável por receber as mensagens MT. Responsável também pela conversão das mensagens para XML. Faz a bilhetagem da mensagem e a envia para o usuário.

- InterfaceMO. Responsável por receber as mensagens MO e convertê-las para XML. Faz o envio das mensagens para o responsável pelo processamento delas. Cria sessões para as mensagens e inicia o trabalho de bilhetagem.

- InterfaceACK. Responsável pelo recebimento de mensagens de sinal eletrônico de reconhecimento usado em transmissões de dados (ACK) enviadas via HTTP.

- InterfaceBILL. Responsável por receber as requisições de bilhetagem enviadas via HTTP, e pela gravação na fila de bilhetagem.

O Portal Alpha faz integração com o IbisWare para o envio de vários tipos de mensagens MT e SMS *Link, como* também de mensagens MO. Para isso é importante que o processo de envio seja independente da formação da consulta, pois como os comandos podem evoluir, e essas mensagens são enviados de vários pontos da aplicação, seria importante que nesses pontos se saiba apenas quais são os parâmetros e os comandos que devem ser informados, e não como eles se formam. Isso possibilitaria maior escalabilidade e centralização da construção dos XMLs para o envio de mensagens MT e MO.

Forças:

- Processo de envio independente da formação da consulta.

- Comandos são enviados a partir de vários pontos do sistema.

- Nos pontos do sistema a única coisa que se sabe são os parâmetros e o comando, mas não como as consultas se formam.

- Pode haver mais de um comando.

- Os comandos podem evoluir e mudar conforme a evolução do *Middleware*.

Solução:

Existem duas maneiras comuns de parametrizar um sistema pelos tipos de objetos que ele cria. Uma é criar subclasses da classe que cria os objetos utilizando o Método Fábrica. A principal desvantagem dessa solução é que requer a criação de uma nova subclasse somente para mudar a classe do produto. Tais mudanças podem gerar uma cascata de modificações encadeadas. (GAMMA et al., 2000)

Outra forma de parametrizar um sistema baseia-se mais na composição de objetos: define-se um objeto que seja responsável por conhecer a classe dos objetos-produto e torne-os um parâmetro do sistema. Esse é o aspecto-chave dos padrões Fábrica Abstrata, Construtor e Protótipo. Como visto no capítulo 2, todos os três padrões envolvem a criação de um novo objeto fábrica cuja responsabilidade é criar objetos-produtos. No padrão Fábrica Abstrata, o objeto-fábrica poduz objetos de várias classes. Sua intenção é fornecer uma interface para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. No padrão Construtor o objeto-fábrica constrói incrementalmente um objeto complexo usando um protocolo igualmente complexo. Sua intenção é separar a construção de um objeto complexo da sua representação, de modo que o mesmo processo de construção possa criar diferentes representações. O padrão Protótipo faz o objeto-fábrica construir um objeto-produto copiando um objeto prototípico. Sua intenção é especificar os tipos de objetos a serem criados usando uma instância protótipo, e criar novos objetos pela cópia deste protótipo. Nesse caso, o objeto-fábrica e o protótipo são o mesmo objeto, porque protótipo é responsável por retornar o produto. (GAMMA et al., 2000)

Como na aplicação do Portal existem vários pontos dos quais podem ser enviadas mensagens MT e MO para o IbisWare, propõe-se que a construção das mensagens XML que serão enviadas ao IbisWare sejam centralizadas em uma só classe. Aqui, a única coisa que se conhece é o comando e os parâmetros que serão passados à classe que constrói a consulta e a envia ao IbisWare.

Como o problema requer que a construção das consultas de envio das mensagens MT e MO sejam independentes de onde elas são enviadas, o padrão mais

apropriado para ser usado é o Construtor, pois sua intenção é separar a construção de um objeto complexo da sua representação, de modo que o mesmo processo de construção possa criar diferentes representações.

A solução do problema com o padrão Construtor, é apresentada na Figura 12.

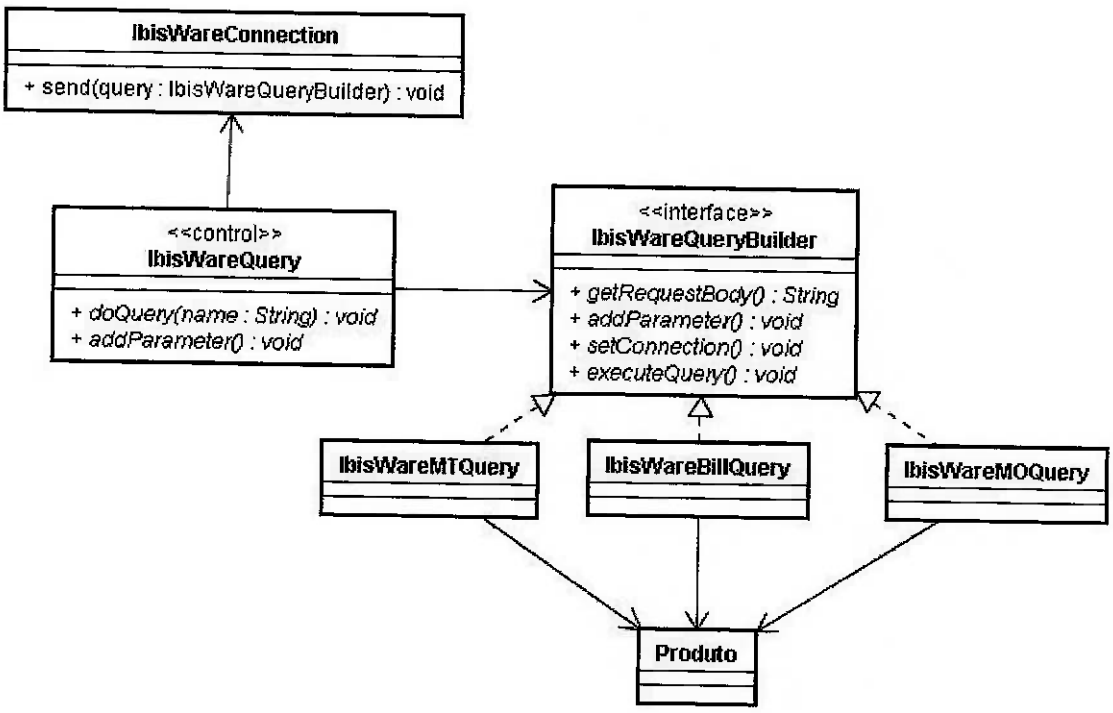

Figura 12 – Diagrama de classes da solução utilizando o padrão Construtor

A classe *IbisWareQuery* controla que tipo de construtor será criado e acerta a conexão com o IbisWare. A classe *IbisWareQueryBuilder* especifica uma interface abstrata para criação de partes de um objeto. As classes *IbisWareMTQuery*, *IbisWareBillQuery* e *IbisWareMOQuery*, constróem e montam partes do produto pela implementação da interface *IbisWareQueryBuilder*.

A classe *Produto*, representa o objeto complexo em construção. As classes que implementam a interface *IbisWareQueryBuilder* constróem a representação interna do produto e definem o processo pelo qual ele é montado.

O diagrama de seqüência da Figura 13 ilustra como o *IbisWareQuery* e o *IbisWareQueryBuilder* cooperam com um cliente.

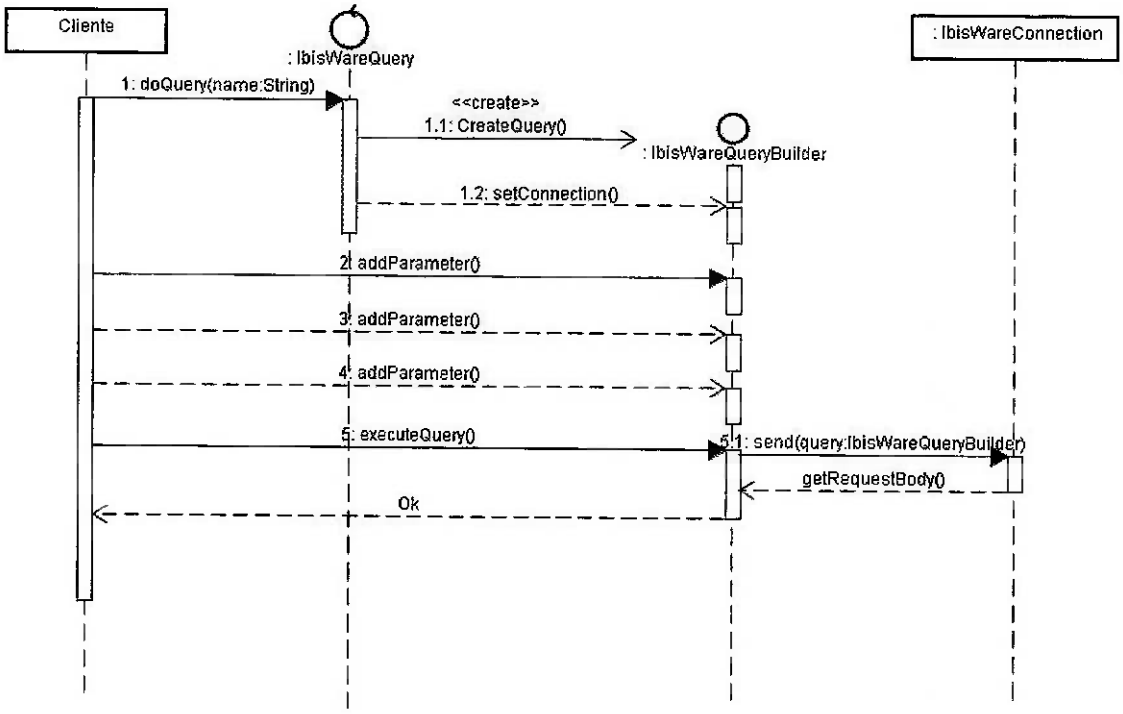

Figura 13 – Relação do comando com a consulta

O cliente solicita à classe *IbisWareQuery* a criação de um Construtor MT, por exemplo. A classe *IbisWareQuery* cria e configura o objeto Construtor desejado. O cliente adiciona os parâmetros ao Construtor criado. Ao término da adição de parâmetros, executa-se a consulta, e envia-se a mensagem MT desejada.

5 CONCLUSÕES

Este trabalho apresentou as principais definições dos padrões clássicos de projeto e apresentou as principais motivações para a sua utilização. Apresentou também a meta-linguagem de padrões de (MESZAROS; DOBLE, 1998) e o idioma de padrões utilizado no estudo de caso para que as soluções sejam bem compreendidas e documentadas.

Apresentou um estudo de caso de uma plataforma tecnológica de suporte a um Portal para dispositivos móveis, denominada Plataforma Alpha, no qual foram analisados como os padrões de projeto Cadeia de Responsabilidades, Procurador de Proteção, Decorador, Procurador de Cache, Ponte e Construtor, ofereceram soluções para os problemas enfrentados por esta plataforma.

Para atingir o objetivo da transmissão das razões de escolha e seleção das soluções para os problemas apresentados no estudo de caso, empregando padrões de projeto, a um projetista recém contratado pela empresa responsável pelo Portal ou ao leitor, utilizou-se como guia a meta-linguagem de padrões de (MESZAROS; DOBLE, 1998). Definiu-se um idioma de padrões derivado dessa meta-linguagem, para tornar a transmissão dos critérios de seleção e escolha das soluções empregando padrões compreensível para o leitor.

Com a utilização do idioma definido na descrição das soluções para os problemas da Plataforma Alpha, foi possível responder adequadamente às seguintes perguntas colocadas por (MESZAROS; DOBLE, 1998) quanto à dificuldade de redação de soluções empregando padrões:

- Como se ter certeza se todas as informações necessárias para a descrição de um padrao estão presentes?
- Como fazer para um leitor escolher facilmente padrões úteis que resolvam um problema?
- Como oferecer ao leitor uma avaliação de um conjunto de Padrões?

- Como deixar as terminologias pouco conhecidas mais claras em uma linguagem de Padrão sem interromper o fluxo do Padrão?
- Como assegurar que diagramas sejam facilmente compreendidos por todo público-alvo?
- Como assegurar que um padrão seja facilmente compreendido pela audiência planejada?
- Como maximizar a probabilidade de o leitor entender um Padrão?

No entanto, um registro mais acurado das decisões para a aplicação de padrões aos problemas necessita resolver dentre outros, os seguintes problemas também colocado por (MESZAROS; DOBLE, 1998):

- Como fazer um Padrão de projeto suficientemente claro e não ambiguo para facilitar a implementação direta?
- Como assegurar que a essência de um Padrão de software pode ser entendida por todo público-alvo?

5.1 Trabalhos Futuros

Como trabalhos futuros, pretende-se responder as perguntas acima colocadas. Uma linha de pesquisa a ser seguida é a de encontrar uma descrição estruturada para prover exemplos de implementação suficientemente abrangentes, escritos em pelo menos uma linguagem de programação prevalescente. Isso deve assegurar que o padrão possa ser capaz de comunicar seus conceitos essenciais mesmo se os exemplos de código forem apagados.

Outra linha complementar é a de cobrir os demais problemas apresentados nas tabelas 1 e 2 do capítulo 3, não tratados no presente trabalho.

REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, C. –
http://www.patternlanguage.com/leveltwo/ca.htm. Acessado em: 11 de novembro de 2007

ALUR, D.; CRUPI, J.; MALKS, D. **Core j2ee patterns:** as melhores práticas e estratégias de design. Tradução Altair Dias Caldas de Moraes. 2º Edição. Rio de Janeiro: Editora Elsevier, 2004. 587 p.

BALANYI, Z; FERENC, R. **Mining Design Patterns from C++ Source Code**, Research Group on Artificial Intelligence, University of Szeged, Hungary, IEEE - Proceedings of the International Conference on Software Maintenance, 2003.

BLILIE, C. **Patterns in Scientific software:** An introduction, IEEE-Computing in science & Engineering, maio/junho 2002. 6p.

COPLIEN J. O., **Patterns of engineering**, IEEE Potentials, abril/Maio 2004. 5 p.

FRANCE R. B., KIM DAE-KYOO, GHOSH S., SONG E. **A UML-Based Pattern Specification Technique**, IEEE – Transactions on software engineering, vol. 30, no. 3, março 2004. 14 p.

FREEMAN, ERIC; FREEMAN, ELISABETH. **Use a Cabeça! Padrões de Projetos**. *Authorized translation from English language Edition*. [S.l.] Tradução Andreza Gonçalves, Marcelo Soares e Pedro César de Conti. Editora Alta Books, 2005. 496 p.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Padrões de Projeto:** Soluções Reutilizáveis de Software Orientado a Objetos. Tradução Luiz A. Meirelles Salgado. 2º Edição. Porto Alegre: Editora Bookman, 2000. 364 p.

Grand, M. **Pattern in Java:** A Catalog of Reusable Design Patterns Illustrated with UML. Volume 1 e 2. United State of America. Editora Theresa Hudson, 1998. Volume 1 467 p. Volume 2 354 p.

GUJ, http://www.guj.com.br/java.tutorial.artigo.137.1.guj, Notícias, tutoriais, e o maior forum brasileiro sobre java, 2002-2006. Acessado em: 10 de outubro de 2007

MESZAROS, G.; DOBLE, J. **MetaPatterns:** A Pattern Language for Pattern Writing, Object System Group, Allen Telecom. Editor Addison-Wesley, 1998. 39 p.

SCHNEIDE, R. L. Design patterns. Rio de Janeiro, maio 1999. Disponível em: http://www.dcc.ufrj.br/~schneide/PSI_981/gp_6/design_patterns.html. Rio de Janeiro, nov. 1999. Acesso em: 08 out 2007.

Wirfs-Brock R. J. **Refreshing Patterns**. IEEE SOFTWARE, May/June 2006. 3 p.

BIBLIOGRAFIA CONSULTADA

Steven John Metsker, **Design Patterns Java Workbook**. Addison-Wesley, 2002

NOKIA Connecting People, http://www.nokia.com.br/nokia/0,8764,43728,00.html acessado em: 26 de agosto de 2007

Wikipédia A enciclopédia livre, http://pt.wikipedia.org/wiki, acessado em: 05 de julho de 2007

COPLIEN J. O., **Idioms and Patterns as Architectural Literature**, IEEE Potentials, Bell Laboratories, Janeiro 1997. 7 p.

PESCIO C., **Principles Versus Patterns**, IEEE Potentials, Eptacom Consulting, Setembro 1997. 2 p.

EDEN, A. H; YEHUDAI, A.; GIL, J. **Precise Specification and Automatic Application Of Design Patterns**, IEEE, 1997. 10p.