UNIVERSIDADE DO PLANALTO CATARINENSE
CURSO DE SISTEMAS DE INFORMAÇÃO
(BACHARELADO)

BRUNO LUIS KUHNEN RAMOS

PADRÕES DE DESENVOLVIMENTO DE SOFTWARE:
UM ESTUDO DE CASO

LAGES (SC)

2013

BRUNO LUIS KUHNEN RAMOS

PADRÕES DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO

Trabalho de Conclusão de Curso submetido à Universidade do Planalto Catarinense para obtenção dos créditos de disciplina com nome equivalente no curso de Sistemas de Informação - Bacharelado.

Orientação: Prof. Ricardo Tiago de Sá, Esp.

LAGES (SC)

2013

BRUNO LUIS KUHNEN RAMOS

PADRÕES DE DESENVOLVIMENTO DE SOFTWARE: UM ESTUDO DE CASO

ESTE RELATÓRIO, DO TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS DA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO, DO 8º. SEMESTRE, OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM SISTEMAS DE INFORMAÇÃO

Lages (SC), 13 de Dezembro de 2013.

Prof. Ricardo Tiago de Sá, Esp. .
Orientador

BANCA EXAMINADORA:

Prof. Rafael Gattino Furtado, Esp.
UNIPLAC

Prof. Sergio Murilo Schutz, Msc.
UNIPLAC

Prof. Claiton Camargo de Souza
Coordenador de Curso / Professor de TCC

Dedico este trabalho a minha família e amigos pelo apoio prestado durante toda a minha caminhada.

Agradeço em primeiro lugar a Deus, que iluminou o meu caminho durante esta caminhada. Agradeço também a minha namorada Jennifer, que de forma especial e carinhosa me deu força e coragem, me apoiando nos momentos de dificuldades. E não deixando de agradecer de maneira grata e grandiosa aos meus pais e avós, que não mediram esforços para que eu chegasse até esta etapa de minha vida.

Epígrafe:

Nunca te é concedido um desejo sem que
te seja concedida também a facilidade de
torná-lo realidade. Entretanto, é possível
que tenhas que lutar por ele.
(Albert Camus)

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

MVC	- *Model, View, Control*
GRASP	- *General Responsibility and Assignment Software Patterns*
GOF	- *Gang of Four*
W3C	- *World Wide Web Consortium*
HTML	- *HyperText Markup Language*
XHTML	- *EXtensible HyperText Markup Language*
CSS	- *Cascading Style Sheets*
XML	- *EXtensible Markup Language*
UML	- *Unified Modeling Language*
API	- *Application Programming Interface*

RESUMO

Com o aumento da complexidade dos sistemas, desenvolvê-los tornou-se uma tarefa muito difícil. Um dos fatores que gera esta dificuldade é que muitas vezes o entendimento do problema não está muito claro. Além disso, há uma escassez grande na documentação dos problemas e nas soluções encontradas para a implementação de suas soluções. Para diminuir ou até mesmo sanar este problema são utilizados padrões de desenvolvimento de software, que são entendidos como estruturas descritas em termos de objetos e classes que podem ser reutilizáveis para solucionar questões de implementação em um projeto, ou até mesmo apenas boas práticas no desenvolvimento de software. Focalizar-se em tais mecanismos durante o processo de desenvolvimento de um sistema, pode levar a uma arquitetura menor, mais simples e muito mais compreensível do que aquelas produzidas quando estes padrões são ignorados. Neste cenário, este trabalho apresenta um estudo de caso sobre padrões de desenvolvimento de software, abrangendo, suas características e funcionalidades.
Dentre estes padrões de desenvolvimento serão abordado padrões de projeto, web, arquiteturais entre outros. Apresenta-se também após a conclusão do estudo de caso, um protótipo desenvolvido utilizando o padrão arquitetural MVC.

Palavras-chave:
Padrões de desenvolvimento de software; características; funcionalidades; MVC.

ABSTRACT

With the increasing complexity of systems, develop them has become a very difficult task. One factor that generates this difficulty is that often the understanding of the problem is unclear. In addition, there is a great shortage in the documentation of the problems and solutions encountered in the implementation of their solutions. To lessen or even solve this problem are used for software development standards , which are understood as structures described in terms of objects and classes that can be reusable to address implementation issues in a project , or even just good practice in developing software. Focusing on these mechanisms during the process of developing a system architecture may lead to a smaller, simpler and more comprehensive than those produced when these patterns are ignored. In this scenario, this paper presents a case study on patterns of software development, covering its features and functionalities. Among these patterns of development will be discussed design patterns, web, and other architectural. We also present after completion of the case study, a prototype developed using the MVC architectural pattern.

Keywords:
Standards for software development; characteristics; functionality; MVC.

SUMÁRIO

1 INTRODUÇÃO

1.1 Apresentação

Segundo Massoni et al. (2003), o desenvolvimento de software tem se tornado mais complexo ao longo dos anos. As exigências por parte dos clientes são cada vez maiores, principalmente em termos de produtividade, qualidade de software e prazos.

O surgimento de novas tecnologias e a necessidade da realização de mudanças nos softwares desenvolvidos para atender às exigências dos clientes também dificulta a tarefa de desenvolver software com qualidade.

Acompanhar as mudanças tecnológicas e atender às necessidades de mudança pode ser uma tarefa bastante complicada se o software não estiver preparado para suportar tais alterações.

Conforme Osterweil (1996), alguns estudos sugerem que pouco mais da metade do esforço utilizado para o desenvolvimento de software esteja voltado a atividades voltadas para assegurar a qualidade de software.

Dois recursos utilizados como forma de buscar o desenvolvimento de software com qualidade são a utilização de processos de desenvolvimento e a utilização de padrões, além de que a utilização de padrões durante o processo de desenvolvimento pode ser uma das técnicas utilizadas para assegurar qualidade nos documentos gerados.

Alexander (1977) afirma que: “cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma forma”.

Muitos dos documentos criados durante um processo de desenvolvimento de software são comuns a vários sistemas. Algumas vezes as mesmas técnicas são utilizadas para a criação desses documentos e, portanto, os mesmos problemas são

encontrados. Os padrões apresentam uma forma de descrever soluções para esses problemas comuns baseado na experiência de outras pessoas.

Este trabalho está dividido em cinco capítulos, que além deste primeiro, introdutório, apresenta o capítulo 2 como referencial teórico buscando explorar as ferramentas e recursos utilizados para o desenvolvimento de software, o capítulo 3 apresentando histórias de usuário e modelagem do protótipo a ser desenvolvido, o capítulo 4 destacando detalhes da implementação e por fim o capítulo 5 com as considerações finais.

1.2 Descrição do problema

Com o atual avanço da tecnologia relacionada ao desenvolvimento de sistemas, a forma como os desenvolvedores trabalham também passa por consideráveis mudanças, e essas mudanças muitas vezes não utilizam um padrão no desenvolvimento de sistemas, seja por falta de conhecimento ou até mesmo pela forma da empresa trabalhar, dificultando assim muitas vezes o entendimento e pondo em duvida a qualidade do código.

Desta forma, um estudo bibliográfico e um estudo de caso podem dimensionar o valor dos padrões de desenvolvimento de software?

1.3 Justificativa

Segundo Laborde (2013), a utilização de padrões possibilita várias vantagens no desenvolvimento de software, dentre elas, pode ser citada à diminuição do processo de aprendizagem de um novo engenheiro de software dentro de um projeto, a reutilização e customização em projetos de desenvolvimento.

O uso de padrões auxilia no desenvolvimento de um projeto com bom nível de coesão e reusabilidade, o que facilita o processo de manutenção do software. (FERREIRA, 2013).

Com a utilização de padrões é possível obter vários benefícios além dos citados

anteriormente. Abaixo serão demonstrados mais alguns benefícios segundo Laborde (2013):

- A reutilização ao invés de redescobrir, pois todos os padrões já foram previamente testados e implementados em projetos que deram certos;
- Aperfeiçoa a comunicação entre os desenvolvedores através da unificação da "língua" entre os envolvidos no projeto, pois todos têm ciência dos termos empregados;
- Facilita a análise, porque quando ela é iniciada, já terá base e estará bem direcionado com relação às necessidades do seu projeto;
- Proporciona alterações menos "dolorosas", uma vez que existe uma padronização, as modificações tornam-se cada vez mais simples devido às estruturas do projeto;
- Aumenta a confiabilidade e reduz a complexidade do código.

1.4 Objetivo geral

Apresentar um estudo de caso sobre padrões de desenvolvimento de software, e por fim desenvolver um protótipo utilizando o padrão arquitetural MVC.

1.5 Objetivos específicos

a) Apresentar um estudo de caso e analisar funções e características de padrões de desenvolvimento de software.
b) Desenvolver um protótipo de registros para controle de contratos de uma empresa utilizando o padrão de projeto arquitetural MVC (*Model, View, Control*).

1.6 Metodologia

Inicialmente foi apresentado um estudo sobre padrões de desenvolvimento,

através de artigos, revistas, *Internet* e livros, como o de SOMMERVILLE. (2011), OSTERWEIL (1996), CRISTOPHER (1977), FOWLER (2006), GAMMA et al.(2000), a fim de entender o tema abordado, o que são, porque utilizar, quais suas características.

Foi realizado também uma pesquisa dos trabalhos correlatos na Universidade do Planalto Catarinense, sendo que encontrou-se apenas trabalhos com diferentes focos, como o da aluna Vanessa Lima Ferreira, que apresentou em 2007 o trabalho com o título "sistema para acompanhamentos dos TCC's do curso de sistema de informação", trabalho este que fala sobre padrões web, diferenciando o objetivo principal do trabalho, pois o objetivo da do trabalho dela não era realizar um estudo de caso e sim desenvolver um sistema utilizando a linguagem de desenvolvimento web PHP. Outro trabalho encontrado na pesquisa foi o da Aluna Deiviane Faelen Ramos de Souza, que em 2012 apresentou um trabalho com o título "sistema de informação web para gerência dos equipamentos médicos de estabelecimentos assistenciais a saúde", neste trabalho foi explanado de forma simples e objetiva sobre padrões de projeto e o padrão MVC, porém o objetivo principal do trabalho tinha um foco completamente diferente do proposto neste trabalho.

Após estes estudos, descoberta suas características, funcionalidades e tendo um maior conhecimento sobre padrões de desenvolvimento, foi realizada a escolha de qual padrão será utilizado no desenvolvimento do protótipo.

Com o padrão já definido, realizou-se a modelagem e o levantamento de requisitos através de historias de usuários.

Depois do processo anterior já estar concluído, iniciou-se a parte de desenvolvimento do sistema.

Por fim, foram realizadas as considerações finais, explanando os resultados adquiridos com o estudo de caso realizado sobre padrões de desenvolvimento, juntamente com as considerações sobre o desenvolvimento do protótipo que utilizou a ferramenta Visual Studio 2012 e o padrão arquitetural MVC 4.

2 PADRÕES DE DESENVOLVIMENTO DE SOFTWARE

Neste capítulo serão apresentados conceitos, características, funcionalidades e exemplos de padrões de desenvolvimento de software. Além disso, apresentam-se conceitos e características sobre o desenvolvimento web e a utilização do padrão MVC no desenvolvimento de aplicações.

2.1 Introdução

"Um padrão descreve um problema que ocorre inúmeras/par vezes em determinado contexto, e descreve ainda a solução para esse problema, de modo que essa solução possa ser utilizada sistematicamente em distintas situações." (ALEXANDER 1977).

Inicialmente o estudo de Alexander foi realizado com o intuito de descobrir se havia alguma regra que pudesse especificar quando uma construção é de boa ou de má qualidade.

A partir dos estudos de Christopher Alexander, profissionais de desenvolvimento de software começaram a incorporar princípios designados por ele na criação das primeiras documentações de padrões.

Gamma et al. (2000) afirmam que em geral, um padrão tem quatro elementos essenciais:

- O Nome do padrão: é o ponto de partida para sua documentação. É através do nome que será possível uma maior compreensão do problema a ser resolvido;
- O Problema: deve descrever o seu contexto de forma clara. Alguns

problemas incluem ainda uma lista de condições que deve ser satisfeitas para que o padrão faça algum sentido;

- A Solução: deve ser descrita como um conjunto de classes com seus relacionamentos, responsabilidades e colaborações;
- As Consequências na aplicação dos padrões podem incluir a análise de impactos sofridos pelo sistema em relação à flexibilidade, extensibilidade ou portabilidade, ajudando a compreensão e avaliação para a sua utilização.

2.2 Padrões de Projeto

2.2.1 O que são padrões de projeto

Segundo Negrão (2013), padrões de projetos são combinações de classes e algoritmos associados que cumprem com propósitos comuns de projetos. São normalmente soluções consagradas que se baseiam nas estruturas da orientação a objeto em sua melhor forma.

Alexander (1977) afirma que: “cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma forma”.

Alexander estava referindo-se apenas as necessidades dele (padrões de projeto em construções e cidades), porém, as mesmas afirmações estão corretas em relação aos padrões de projeto orientado a objeto.

A partir das citações acima fica claro que padrões de projeto são soluções para problemas que alguém algum dia já teve e resolveu utilizando um modelo que foi documentado e que pode ser adaptado totalmente ou de acordo com a necessidade de cada aplicação.

2.2.2 Porque utilizar padrões de projeto

A utilização de padrões de projeto possibilita várias vantagens no

desenvolvimento de software, dentre elas pode-se citar: Diminuição do processo de aprendizagem de um novo engenheiro de software dentro de um projeto, a reutilização e customização em projetos de desenvolvimento (LABORDE, 2013).

Com a utilização de padrões de projeto é possível obter vários benefícios além dos citados anteriormente. Abaixo serão demonstrados mais alguns benefícios:

- A reutilização ao invés de redescobrir, pois todos os padrões já foram previamente testados e implementados em projetos que deram certos;
- Aperfeiçoa a comunicação entre os desenvolvedores através da unificação da "língua" entre os envolvidos no projeto, pois todos têm ciência dos termos empregados;
- Facilita a análise, porque quando é iniciada a parte análise, já terá base e estará bem direcionado com relação às necessidades do seu projeto;
- Proporciona alterações menos "dolorosas", uma vez que existe uma padronização, as modificações tornam-se cada vez mais simples devido às estruturas do projeto;
- Aumenta a confiabilidade e reduz a complexidade do código.

2.2.3 Padrão GRASP

O conjunto de padrões conhecido como GRASP (*General Responsibility and Assignment Software Patterns*), teve inicio com o livro "Utilizando UML e Padrões: Uma introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo" e conforme Larman (2008), estes padrões descrevem os princípios fundamentais da atribuição de responsabilidades a objetos, expressas na forma de padrões.

Fowler (2006) diz que "entender responsabilidades é essencial para o bom projeto orientado a objetos".

Seguindo esta mesma linha de pensamento e fazendo-se necessário o bom entendimento de responsabilidade no contexto citado acima Oleques (2012) diz que: as responsabilidades de um projeto podem ser divididas em "conhecer" e "fazer":

- As responsabilidades “conhecer” estão relacionadas à distribuição das características do sistema entre as classes;
- As responsabilidades “fazer” estão relacionadas com a distribuição do comportamento do sistema entre as classes.

A Figura 2 apresenta os padrões de projeto segundo GRASP.

FIGURA 1 - Padrões de Projeto GRASP

Design Patterns GRASP	
Padrões Fundamentais	Controller
	Creator
	Information Expert
	Low Coupling
	High Cohesion
Padrões Avançados	Indirection
	Protected Variations
	Polymorphism
	Pure Fabrication

Fonte: (LARMAN, 2008)

Na Figura 1 são mostrados todos os padrões de projeto GRASP e sua divisão, que ocorre entre padrões de projeto fundamentais e padrões de projeto avançados.

Está sendo visto que os padrões fundamentais são: *Controller, Creator, Information Expert, Low Coupling e High Cohesion*. Enquanto os padrões avançados são: *Indirection, Protected Variations, Polymorphism e Pure Fabrication*.

2.2.4 Padrão GoF

Após o lançamento do livro “Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos”, cujos autores foram Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides, estes quatro escritores ficaram conhecidos como “Gangue dos Quatro” (*Gang of Four* ou GOF), onde neste livro é formado um catálogo de boas decisões.

Segundo GAMMA et al. (2000), este catálogo é dividido em três tipos de padrões:

- Padrões de Criação: preocupam-se em como criar objetos;

- Padrões de Estrutura: preocupam-se em como compor objetos;
- Padrões de Comportamento: preocupam-se em como os objetos devem interagir.

Os padrões GoF refletem situações muito recorrentes em projeto Orientado a Objeto, e podem ser vistos como o mínimo que todo Projetista Orientado a Objeto deveria saber.

Neste catálogo também está descrita a estrutura de documentação de um padrão e como os padrões se relacionam, conforme apresenta a figura 1.

FIGURA 2 - Classificação dos 23 Padrões de Projeto Segundo GoF

Fonte: GAMMA et al. (2000)

A Figura 2 mostra a classificação dos 23 padrões de projeto segundo a Gang of Four.

2.2.5 Modelos de Padrões de Projeto segundo GoF

A seguir serão apresentados todos os 23 padrões segundo Gamma et al., seus objetivos, vantagens e algumas de suas características.

2.2.5.1 Padrões de Criação

Segundo Leite (2013), padrões de criação são aqueles que abstraem e ou

adiam o processo de criação dos objetos. 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 objeto delegará a instanciação para outro objeto.

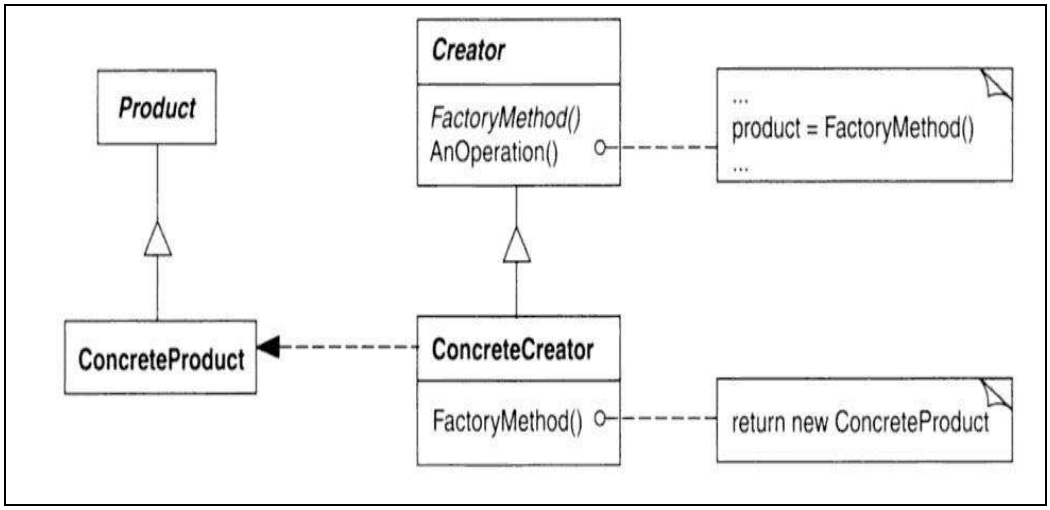
2.2.5.1.1 Padrão Factory Method

O padrão *Factory Method*, também conhecido como Virtual *Constructor* tem como principal objetivo conforme Gamma et al. (2000), definir uma interface para criar um objeto, mas deixas as subclasses decidirem que classes instanciar, permite também adiar a instanciação para subclasses.

A definição acima mostra que ao invés de criar objetos diretamente em uma classe concreta, nós definimos uma interface de criação de objetos e cada subclasse fica responsável por criar seus objetos.

Na Figura a seguir será mostrada a estrutura do padrão *Factory Method*:

FIGURA 3 - Estrutura do padrão *Factory Method*



Na Figura 3, é mostrada a estrutura do padrão *Factory Method*, e seus participantes são os seguintes: *Produtct*, que define a interface de objetos que o método fábrica cria, o *ConcreteProduct* que é o responsável pela implementação da interface de *Produtct* e por último o *Creator* que é quem define um método fábrica abstrata que as subclasses implementam para criar um produto, e pode possuir um ou

mais métodos com seus devidos comportamentos que chamarão factoryMethod.

Na sequência serão expostas algumas vantagens na utilização do padrão Factory Method segundo Furtado (2013):

- Elimina a necessidade de montar um código em função a uma classe específica;
- Dá maior flexibilidade para as classes, pois criar um objeto em uma classe que utiliza o *Factory Method* é melhor que fazê-lo em separado, funcionando assim, como uma conexão para que uma das subclasses forneça uma versão estendida de um objeto;

Conforme Ortiz (2013), algumas desvantagens da utilização do padrão *Factory Method* são:

- Especializar uma classe apenas para instanciar um objeto de uma subclasse de outra superclasse pode se revelar bastante improdutiva;
- Manutenção: Sempre que a classe mudar os critérios de seleção, o objeto de todas as aplicações que usam a hierarquia de classes deverá ser submetido a uma mudança correspondente;
- Flexibilidade: Como não pode ser parametrizada, qualquer alteração no processo de criação pode se tornar complexa de se alterar, já que ele também não é flexível.

2.2.5.1.2 Padrão Asbstract Factory

Também conhecido como *kit*, este padrão fornece uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas (NASCIMENTO, 2013a).

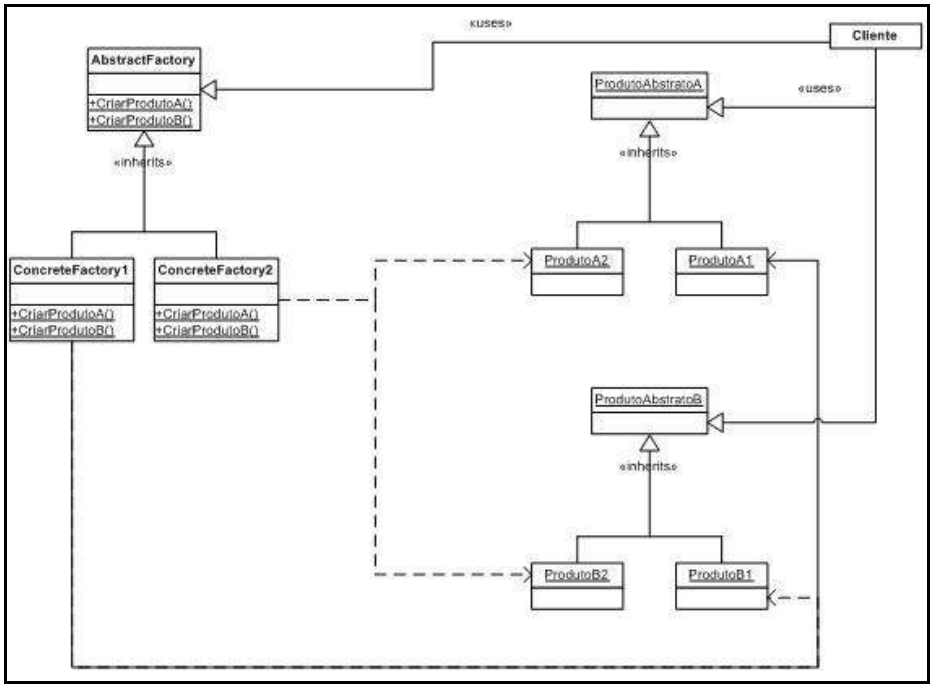
Segundo Gamma et al. (2000) deve ser utilizado o *Abstract Factory* quando:

- O sistema precisar ser, independente de como seus produtos são criados, compostos e representados;
- Uma família de objetos de produtos relacionados é projetada para ser usada de forma conjunta e você deva garantir esta restrição;
- Você quer prover uma biblioteca de classes de produtos e quer revelar

apenas suas interfaces e não suas implementações.

A Figura 3 mostra a estrutura do Padrão de Projeto *Abstract Factory.*

FIGURA 4 - Estrutura do Padrão de Projeto *Abstrct Factory*



Fonte: (NASCIMENTO, 2013a)

A Figura 4 mostra a estrutura do padrão de projeto *Abstract Factory*, em que a Classe *AbstractFactory* declara uma interface para a criação de objetos abstratos, a classe *ConcreteFactory* implementa as operações que criam objetos concretos, a classe *AbstractProduct* declara uma interface para um tipo de objeto, enquanto a classe *ConcreteProduct* define um objeto a ser criado pela fábrica concreta correspondente e implementa a interface de *AbstractProduct* e por fim a classe *Client* utiliza somente as interfaces declaradas pelas classes *AbstractFactory* e *AbstractProduct.*

2.2.5.1.3 Padrão Builder

Seu objetivo é separar a construção de objetos complexos da sua representação de forma que o mesmo processo de construção possa criar diferentes representações (NASCIMENTO, 2013b).

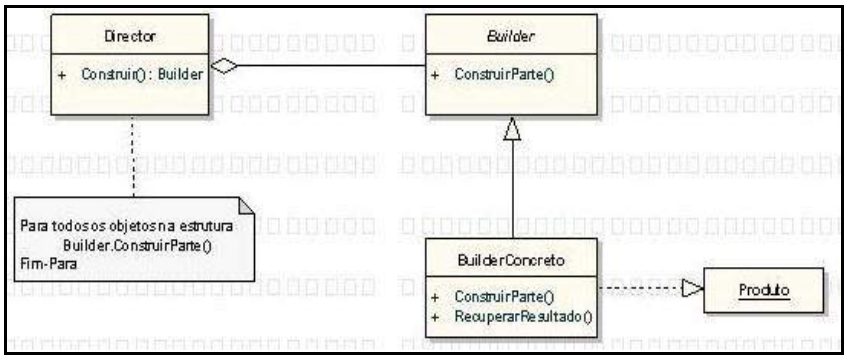
Segundo Gamma et al. (2000) contém os seguintes elementos:

- *Director:* Constrói um objeto utilizando a interface do *builder*;

- *Builder:* Especifica uma interface para um construtor de partes do objeto-produto;
- *Concrete builder:* Define uma implementação da interface *builder*, mantém a representação que cria fornece interface para recuperação do produto;
- *Product:* O objeto complexo acabado de construir. Inclui classes que definem as partes constituintes.

A Figura 4 mostra a estrutura do Padrão de Projeto *Builder*.

FIGURA 5 - Estrutura do Padrão de Projeto *Builder*



Fonte: (NASCIMENTO, 2013b)

A Figura 5 mostra a estrutura do padrão de projeto *Builder*, em que a classe *Builder* especifica uma interface abstrata para criação de partes de um objeto, a classe *Builder*Concreto constrói e monta partes do objeto pela implementação da interface Builder; define e mantém a representação que cria e fornece uma interface para recuperação do produto. Mostra também que a classe *Director* constrói um objeto usando a interface de *Builder* e por fim a classe Produto representa o objeto complexo em construção; incluem classes que definem as partes constituintes, inclusive as interfaces para a montagem das partes no resultado final.

2.2.5.1.4 Padrão Prototype

Segundo Gamma et al. (2000), o padrão *Prototype* tem como função especificar os tipos de objetos a serem criados usando uma instância-protótipo e criar novos objetos pela cópia desse protótipo.

Desta forma, este padrão permite clonar/copiar um objeto a partir de uma

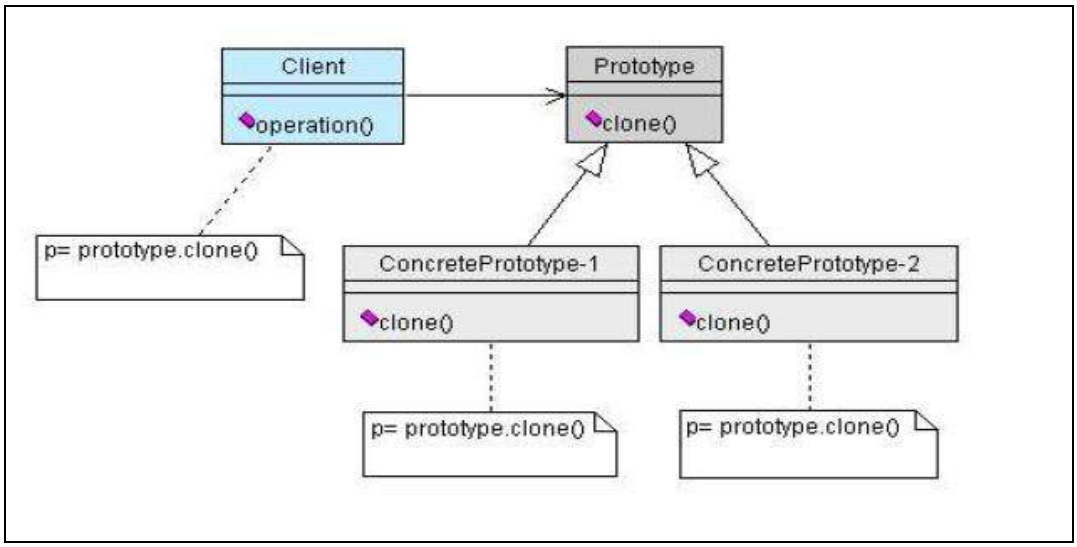
classe, ou seja, o desenvolvedor pode criar uma instância clonada e definir o objetivo em tempo de execução.

Ainda conforme Gamma et al. (2000), algumas das motivações de se utilizar o padrão Prototype são:

- Um detalhe que torna o *Prototype* único em relação aos outros padrões de criação é que ele utiliza objeto para criar os produtos, enquanto os outros utilizam classe.
- O padrão *Prototype* leva grande vantagem quando o processo de criação de seus produtos é muito caro, ou mais caro que uma clonagem.
- Os produtos do *Prototype* podem ser alterados livremente apenas mudando os atributos.

Abaixo será mostrada a estrutura genérica do padrão *Prototype:*

FIGURA 6 - Estrutura genérica Padrão Prototype



FONTE: (VANINI, 2013)

Na Figura 6 é mostrado o cliente, que solicita a um protótipo que crie uma cópia de si mesmo, gerando outro objeto, o *Prototype*, que especifica uma interface para clonar a si próprio e por último o *Concrete Prototype*, que implementa uma operação para clonar a si próprio.

Abaixo serão explanadas algumas consequências da utilização do padrão *Prototype* conforme Vanini (2013):

- Permite acrescentar e remover produtos em tempo de execução;
- Especifica novos objetos pela variação de valores ou pela variação de estrutura;
- Reduz o número de subclasses;
- Configura dinamicamente as classes ou objetos da aplicação;
- Cada subclasse deve implementar a operação clone(), o que pode ser difícil em alguns casos.

Segundo Hartwig (2013), algumas das desvantagens de utilizar o padrão *Prototype* são:

- Cada subclasse de *Prototype* deve implementar a operação clone, o que pode ser difícil.
- A implementação de clone pode ser complicada quando uma estrutura interna da classe inclui objetos que não suportam operação de cópia ou têm referencias circulares.

2.2.5.1.5 Padrão Singleton

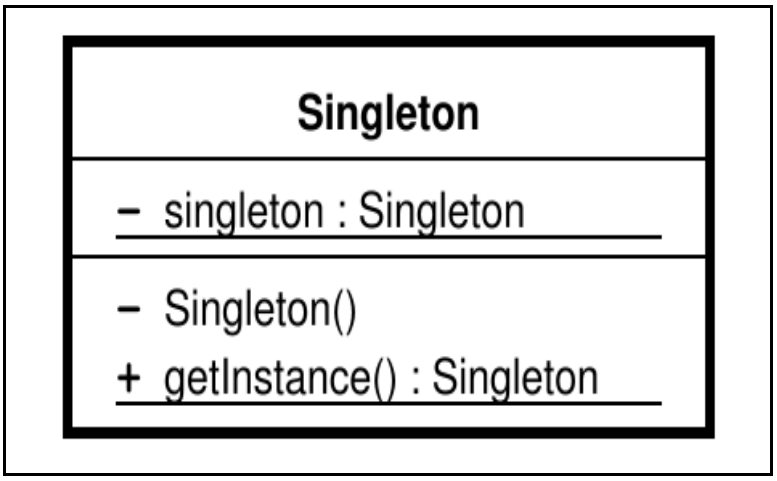
"O padrão *Singleton* permite criar objetos únicos para os quais há apenas uma instância. Este padrão oferece um ponto de acesso global, assim como uma variável global, porém sem as desvantagens das variáveis globais" (MEDEIROS, 2013b).

Como exemplo pode utilizar que, em um sistema operacional, existe a necessidade de representar a fila de impressão de uma impressora. Outra utilização do padrão de projeto *singleton* pode ser em uma aplicação que precisa de uma infraestrutura de *log* (registro) de dados, pode-se implementar uma classe no padrão *singleton.* Desta forma existe apenas um objeto responsável pelo *log* em toda a aplicação que é acessível unicamente através da classe *singleton.*

O padrão de projeto *singleton* tem como principais benefícios o controle sobre como e quando os clientes acessam a instância e é mais flexível que métodos estáticos por permitir o polimorfismo.

A Figura 7 mostra o diagrama de classe do padrão de projeto *singleton.*

FIGURA 7 - Diagrama de Classe Padrão de Projeto *Singleton*



Fonte: (MEDEIROS, 2013b)

A Figura 7 mostra o atributo *singleton* que é do tipo da sua própria classe e é estático, nessa variável tem-se a única instância da classe. Nos métodos pode-se observar a presença do construtor da classe *Singleton*() que é privado. Ou seja, um construtor privado não permite que a classe seja instanciada a não ser que seja feito por ela mesmo na qual será instanciada pelo método *GetInstance*() que é estático e assim pode ser acessado de qualquer outra classe sem precisar instanciar *Singleton* (MEDEIROS, 2013b).

2.2.5.2 Padrões Estruturais

Conforme Leite (2013), os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores. Os padrões estruturais de classes utilizam a herança para compor interfaces ou implementações, e os de objeto ao invés de compor interfaces ou implementações, eles descrevem maneiras de compor objetos para obter novas funcionalidades. A flexibilidade obtida pela composição de objetos provém da capacidade de mudar a composição em tempo de execução, o que não é possível com a composição estática.

2.2.5.2.1 Padrão Class Adapter e *Object Adapter*

Segundo Gamma et al. (2000), o padrão *Adapter*, também conhecido como *Wrapper* tem como intenção converter a interface de uma classe em outra interface,

esperada pelos clientes. O *Adapter* permite que classes com interfaces incompatíveis trabalhem em conjunto, o que, de outra forma seria impossível.

Conforme Soares (2013), sua principal vantagem é:

- Adapta o adaptador para o Alvo através de uma classe concreta. Como consequência, uma classe adaptada não funcionará para adaptar uma classe e suas subclasses;

2.2.5.2.2 Padrão Bridge

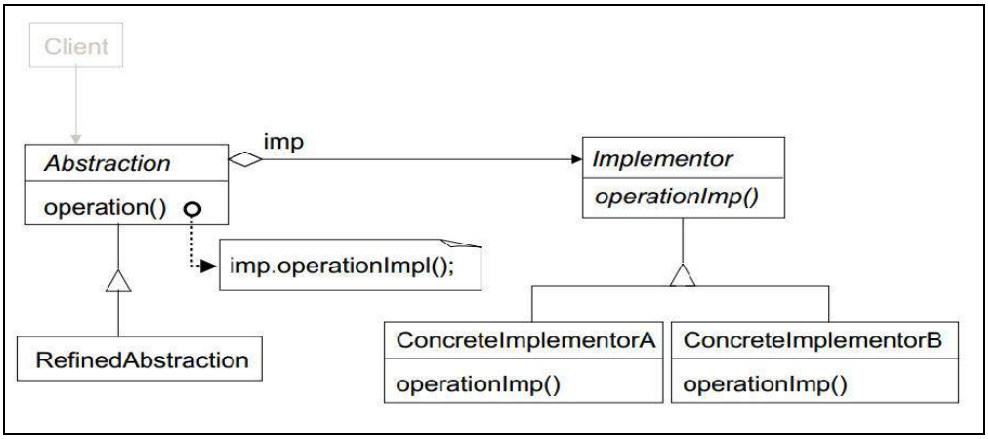
Conforme Gamma et al. (2000), a principal função do padrão *Bridge* é desacoplar uma abstração da sua implementação, de modo que as duas possam variar independentemente.

Segundo Fernandes (2013), as principais motivações para a utilização do padrão *Bridge* são as seguintes:

- Quando uma abstração pode ter várias implementações a solução usual é acomodar todas as implementações através de herança;
- No entanto, herança liga de forma permanente uma abstração a uma implementação;
- O padrão *Bridge* permite colocar as abstrações e suas implementações em diferentes hierarquias de classes, que permite que variem de forma independente.

Na sequencia será demostrado o padrão genérico do padrão *Bridge*:

FIGURA 8 - Estrutura genérica do padrão *Bridge*



FONTE: (FERNANDES, 2013)

Na Figura 8 é mostrada a estrutura genérica do padrão *Bridge*, sendo que seus participantes são os seguintes: o *Abstraction* que define a interface de abstração e mantém uma referência para um objeto do tipo *Implementor*, o *RefinedAbstraction*, que estende a interface definida por *Abstraction*, em seguida o *Implementor*, que é o responsável por definir a interface para as classes de abstração, esta classe não precisa corresponder exatamente à interface de *Abstraction*; de fato, as duas interfaces podem ser bem diferentes. A interface do *Implementor* fornece somente operações primitivas e *Abstraction* define operações de nível mais alto baseadas nessas primitivas, e por último a classe *ConcreteImplementor*, que é quem implementa a interface *Implementor* e define sua implementação concreta.

A seguir será mostrado algumas das consequências da utilização do padrão *Bridge* conforme Ribeiro (2013a):

- Desacopla a Interface, pois uma implementação não fica permanentemente presa a uma interface. A implementação de uma abstração pode ser configurada em tempo de execução. É até mesmo possível para um objeto mudar sua implementação em tempo de execução;
- Esse desacoplamento encoraja o uso de camadas que podem melhorar a estruturação de um sistema. A parte de alto nível de um sistema somente tem que ter conhecimento de *Abstraction* e *Implementor;*
- Extensebilidade melhorada, pois é possível estender as hierarquias de *Abstraction* e *Implementor* independentemente;
- Ocultação de detalhes de implementação do cliente: protege e isola os clientes de detalhes de implementação, tais como compartilhamento de objetos *Implementor* e o mecanismo de contagem de referências que os acompanham.

2.2.5.2.3 Padrão Composite

Segundo Sierra (2009), o padrão *Composite* permite que você componha objetos em estruturas de árvore para representar hierarquias parte-todo. Com esse

padrão, os clientes podem tratar objetos individuais ou composições de objetos de maneira uniforme.

Conforme Fernandes (2013), algumas das consequências de se utilizar o padrão Composite são as seguintes:

- Definição de hierarquias de classes consistindo de objetos primitivos e compostos. Sempre que um cliente espera um objeto primitivo pode receber um objeto composto;
- Torna o cliente mais simples;
- Facilita a criação de novas classes de componentes (e compostos);
- Pode tornar o objeto excessivamente genérico.

2.2.5.3 Padrões de Comportamento

Segundo Leite (2013), os padrões de comportamento se concentram nos algoritmos e atribuições de responsabilidades entre os objetos. Eles não descrevem apenas padrões de objetos ou de classes, mas também os padrões de comunicação entre os objetos. Os padrões comportamentais de classes utilizam a herança para distribuir o comportamento entre classes, e os padrões de comportamento de objeto utilizam a composição de objetos em contrapartida a herança. Alguns descrevem como grupos de objetos cooperam para a execução de uma tarefa que não poderia ser executada por um objeto sozinho.

2.2.5.3.1 Padrão Facade

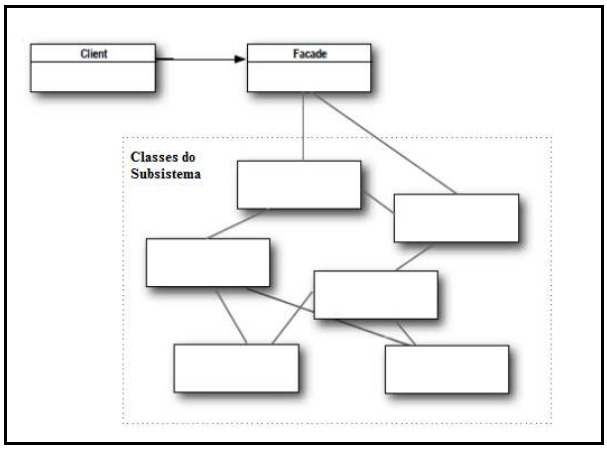
O padrão de projeto *Facade* oculta toda a complexidade de uma ou mais classes através de uma *Facade* (fachada). A intenção deste padrão de projeto é simplificar uma interface (MEDEIROS, 2013a).

Conforme Gamma et al. (2000), o padrão de projeto *Facade* pode:

- Tornar uma biblioteca de *software* mais fácil de entender e usar;
- Tornar o código que utiliza esta biblioteca mais fácil de entender;
- Reduzir as dependências em relação às características internas de uma biblioteca, trazendo flexibilidade no desenvolvimento do sistema;

A Figura 9 mostra o diagrama de Classe do Padrão de Projeto *Facade.*

FIGURA 9 - Diagrama de Classe do Padrão de Projeto *Facade*



Fonte: (MEDEIROS, 2013a)

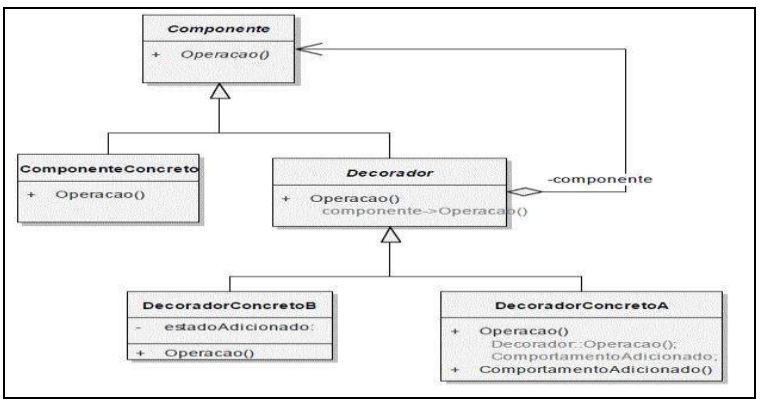
Na Figura 9 se tem o *Client* que é quem acessa a *Facade* que, por sua vez, é uma interface simplificada do subsistema, sendo esta unificada e fácil de ser utilizada pelo cliente. Abaixo da *Facade* têm-se as classes do sistema que podem ser chamados diretamente, mas estão sendo agrupados da *Facade* (MEDEIROS, 2013a).

2.2.5.3.2 Padrão Decorator

Segundo Sierra (2009), o padrão *Decorator* anexa responsabilidades adicionais a um objeto dinamicamente. Os decoradores fornecem uma alternativa flexível de subclasse para estender a funcionalidade.

De forma mais simples podemos dizer que o padrão *decorator* permite estender dinamicamente as características de uma classe usando a composição.

FIGURA 10 - Diagrama de classe padrão *Decorator*



FONTE: (MARCORATTI, 2013a)

Na Figura 10 são mostrado as classes/objetos do padrão *Decorator*, que são o componente, que define a interface para objetos que podem ter responsabilidades adicionadas a eles dinamicamente, o componente concreto, que define um objeto para o qual responsabilidades adicionais podem ser anexadas, o decorador, que é responsável por manter uma referência para um objeto componente e definir uma interface compatível com interface de componente, e por ultimo o decorador concreto, que é responsável por adicionar responsabilidades ao componente.

Segundo Marcoratti (2013a), algumas das vantagens de se utilizar o padrão *Decorator* são as seguintes:

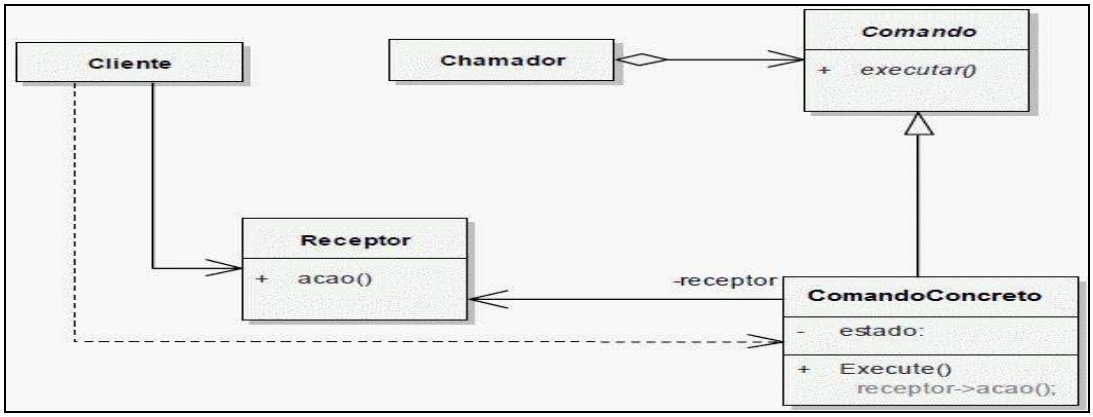
- É uma alternativa a utilização da herança que é definida em tempo de compilação enquanto uma classe decorada é definida em tempo de execução;
- Adere ao principio Open-Closed, onde as classes devem estar abertas para extensão, mas fechadas para modificação;

2.2.5.3.3 Padrão Command

Segundo Sierra (2009), o padrão *Command* encapsula uma solicitação como um objeto, o que lhe permite parametrizar outros objetos com diferentes solicitações, enfileirar ou registrar solicitações e implementar recursos de cancelamento de operações.

Na sequencia será mostrado o diagrama de classes para o padrão *Command*:

FIGURA 11 - Diagrama de classes do padrão Command



FONTE: (MARCORATTI, 2013b)

A Figura 11 mostra o digrama de classes do padrão *Command*, tendo os seguintes participantes:

- Comando (*Command):* Declara uma interface para executar uma operação; Esta classe abstrata é a classe base para todos os objetos *Command*. A classe também define um método abstrato que é usado pelo Chamador para executar comandos;

2.2.5.3.4 Padrão Iterator

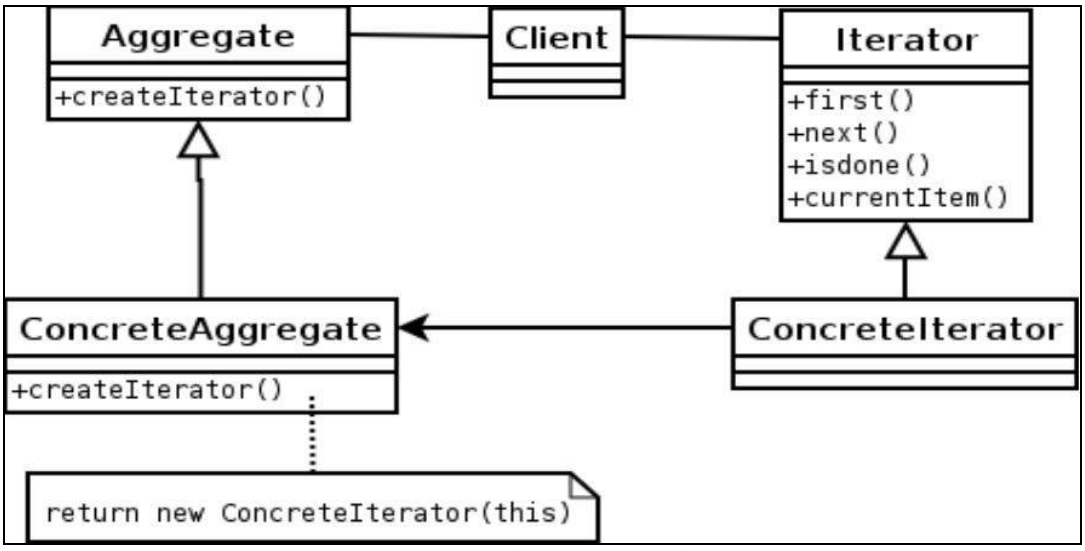
Sierra (2009) afirma que, o padrão *Iterator* fornece uma maneira de acessar sequencialmente os elementos de um objeto agregado sem expor a sua representação subjacente.

Segundo Vidal (2013), o padrão *Iterator* tem como motivação:

- Não expor a estrutura interna de elementos agregados;
- Percorrer um objeto agregado a uma lista de diferentes maneiras e de acordo com um filtro, sem sobrecarregar a interface da lista com essas funcionalidades;
- Percorrer a mesma lista de diferentes maneiras e simultaneamente;
- Permitir iteração poli fórmica.

Logo abaixo será mostrada a estrutura do padrão *Iterator*:

FIGURA 12 - Estrutura do Padrão *Iterator*



FONTE: (VIDAL, 2013)

A Figura 12 mostra a estrutura do padrão *Iterator*, que tem os seguintes participantes:

- *Agregatte:* define a interface para criar um objeto *iterator*;
- *ConcreteAgreggate:* implementa a interface de criação *iterator* para retornar a instância apropriada de *ConcreteIterator*;
- *Iterator*: Define a interface para acessar e percorrer elementos agregados;
- *ConcreteIterator:* Implementa a interface *Iterator* e mantém a posição corrente ao percorrer o agregado.

Como consequências Vidal (2013), afirma que:

- O padrão Iterator permite variações no percurso de um agregado;
- Iterators simplificam a interface do agregado;
- Pode existir mais de um percurso acontecendo simultaneamente.

2.2.5.3.5 Padrão State

Conforme Sierra (2009), o padrão *State* permite que um objeto altere o seu comportamento quando o seu estado interno muda. O objeto parecerá ter mudado de classe.

Segundo Gamma et. Al (2000), a principal consequência na utilização do padrão *State* é tornar explícito os estados de um objeto e as transições de estados.

2.2.5.3.6 Padrão Proxy

Segundo Sierra (2009), o padrão *Proxy* fornece um substituto ou representante de outro objeto para controlar o acesso a ele.

Na sequencia será explanado um pouco sobre as duas formas de utilização do padrão *Proxy:*

Conforme Mariotti (2013a), utilizando o *Simple Proxy* um objeto geralmente possui uma interface que é praticamente idêntica à interface do *Proxy* que irá substituí-lo. O *Proxy* vai realizar seu trabalho criteriosamente por encaminhamento de solicitações para o objeto subjacente.

Enquanto utilizando o *Remote Proxie* provê um representante local para um

objeto em um espaço de endereçamento diferente.

Conforme Gamma et. Al (2000), algumas das consequências da utilização do padrão *Proxy* são:

- Adiciona 1 nível de indireção. O uso da mesma depende do tipo:
- *Remote Proxy* esconde o fato de o objeto real residir em um espaço de endereçamento diferente;
- *Virtual Proxy* provê otimizações, tais como criar um objeto sob demanda.

2.2.5.3.7 Padrão Memento

Segundo Gamma et al. (2000), o padrão Memento armazena o estado interno do objeto originador (cria um memento contendo um instantâneo do seu estado interno corrente, utiliza o memento para restaurar seu estado interno). Ele pode armazenar pouco ou muito do estado interno do originador, conforme necessário e segundo critérios do seu originador. Mementos têm duas interfaces, o Caretaker (é responsável pela custódia do memento, porém nunca opera ou examina seus conteúdos), que vê uma interface mínima do memento, e o originador, que diferentemente do Caretaker, vê uma ampla interface, o que lhe permite acessar todos os dados necessários para restaurar ao seu estado prévio. Idealmente, somente o originador que produziu o memento teria acesso permitido ao seu estado interno.

Abaixo serão explanados alguns pontos positivos e negativos do padrão de software memento conforme Gamma et al. (2000):

Pontos positivos

- Preservação das fronteiras. Com o memento existe a exposição de informação que somente um originador deveria administrar, mas que, contudo, deve ser armazenada fora do originador. O padrão protege outros objetos de aspectos internos potencialmente complexos do Originador, desta maneira preservando as fronteiras de encapsulamento.
- Simplifica o Originador. Em outros projetos de encapsulamento, o

Originador mantém as versões do estado interno solicitado pelos clientes. Isso coloca toda a carga de administração do armazenamento sobre o Originador. Deixar o clientes administrarem o estado que solicitam simplifica o Originador e evita que eles tenham que notificar os originadores quando terminarem a utilização.

Pontos Negativos

- O uso de mementos pode ser computacionalmente caro. Mementos podem produzir custos adicionais consideráveis se o Originador tiver de copiar grandes quantidades de informação para armazenar no memento, ou se os clientes criam e desenvolvem mementos para o Originador com muita frequência. A menos que seja barato encapsular e restaurar o estado do Originador, o padrão pode não ser apropriado.
- Custos ocultos na custódia de mementos. Um caretaker é responsável por deletar o memento do qual ele tem a custódia. Contudo, o caretaker não tem a ideia do volume ocupado pelo estado do memento. Daí um caretaker leve pode incorrer grandes custos de armazenamento quando armazena mementos.

2.2.5.3.8 Padrão Interpreter

Segundo Gamma et al. (2000), o propósito do padrão Interpreter é: dada uma linguagem, cria uma representação para a gramatica da linguagem, juntamente com um interpretador que usa esta representação para interpretar sentenças na linguagem.

Conforme Steve (2002), as consequências de sua utilização são as seguintes:

- É simples modificar e estender a gramatica;
- Adicionar novas formas de interpretar expressões é simples.

2.2.5.3.9 Padrão Strategy

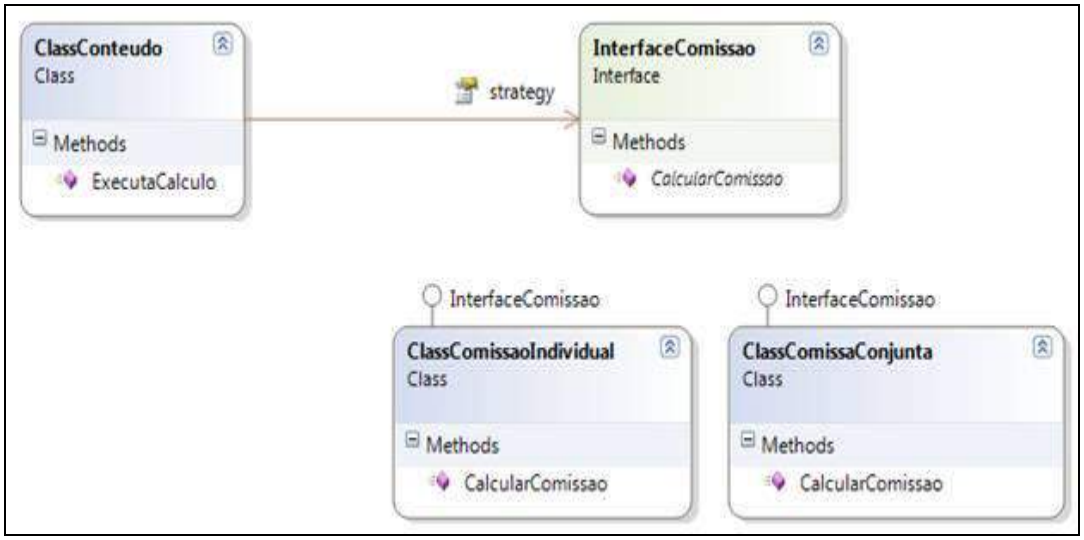
Segundo Mariotti (2013b), o padrão *Strategy* é um conjunto de algoritmos encapsulados e intercambiáveis, sendo que a estratégia permite que o algoritmo se diversifique independentemente dos clientes que os utilizam.

Em outras palavras, o padrão *Strategy* funciona como um plano de ação para alcançar um determinado objetivo após receber as condições certas de entrada. Portanto, este padrão é semelhante a um algoritmo ou um método que processa e reduz resultados a partir de um conjunto de entradas. Sua principal intenção é encapsular abordagens alternativas em classes distintas mas com operação comum.

Conforme Mariotti (2013b), o padrão *Strategy* funciona da seguinte forma: a estratégia define as entradas e saídas da operação, mas deixa a implementação para as classes individualmente, com isso as classes implementam diversas abordagens para a mesma ação, sendo portanto, intercambiáveis (são chamados de intercambiáveis as operações que contem diferentes comportamentos, mas com a mesma interface para o cliente).

Abaixo será mostrado um exemplo de utilização do padrão de software *Strategy:*

FIGURA 13 - Exemplo utilização padrão *Strategy*



FONTE: (MARIOTTI,2013b)

A Figura 13 mostra que a classe Conteúdo representa o objeto que faz referência para a interface *Strategy*, ou seja, é um conjunto de instruções do sistema que faz a chamada e passa as diretrizes de entrada para o método de estratégia, enquanto isso, Interface Comissão define a interface que vai dar suporte as classes de estratégia. A classe conteúdo usa essa interface para chamada dos algoritmos definidos pela interface.

Por ultimo, a *ConcreteStrategy (ClassComissaoIndividual, ClassComissaoConjunta*) cuidam da implementação de algoritmos que fazem uso da interface *Strategy.*

Pode-se concluir que este padrão permite que o conteúdo faça chamada a estratégia usando polimorfismo para executar a estratégia certa, o que permite o desenvolvimento de um código simples e limpo.

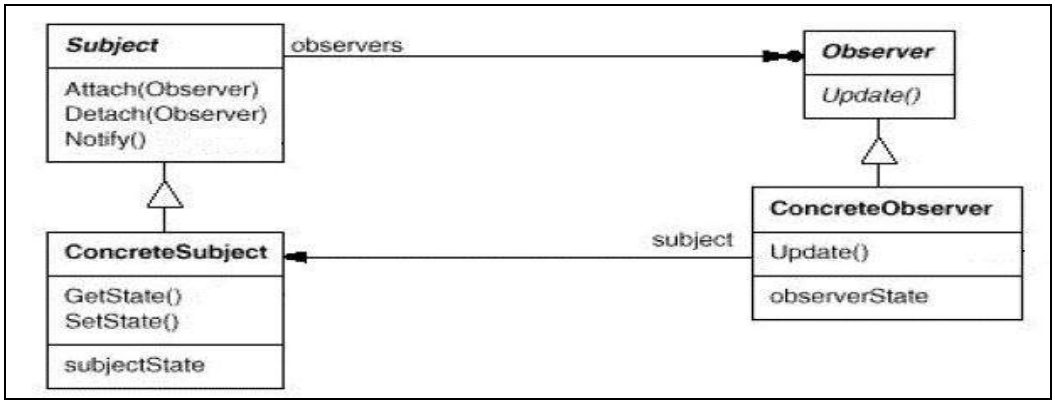
2.2.5.3.10 Padrão Observer

Segundo Macoratti (2013c), o padrão de projeto *Observer* é um padrão comportamental que representa uma relação 1-N (de um para muitos) entre objetos. Assim, quando um objeto altera o seu estado, os objetos dependentes serão notificados/informados/avisados e atualizados de forma automática. Este padrão possibilita que objetos sejam avisados da mudança de estado de outros eventos em outro objeto.

Conforme Gamma et al. (2000), o padrão *Observer* permite definir uma dependência um-para-muitos entre objetos para que quando um objeto mudar de estado, todos os seus dependentes sejam notificados e atualizados automaticamente.

Abaixo será mostrado o diagrama de classes para o padrão *Observer:*

FIGURA 14 - Diagrama de classes padrão *Observer*



FONTE: (MARCORATTI, 2013c)

Na Figura 14, o sujeito (*subject),* conhece os seus obersvadores, um número qualquer de observadores é permitido. Provê uma interface para adicionar/remover

observadores. O observador (*observer)* define uma interface de atualização para os objetos que devem ser notificados sobre as mudanças no sujeito. O sujeito concreto (*ConcreteSubject*) armazena o estado de interesse para os objetos de *ConcreteObserver;* envia uma notificação aos observadores concretos quando seu estado mudar. E por último o observador concreto (*ConcreteObserver*) implementa a interface definida pelo observador para tratar as notificações.

2.2.5.3.11 Padrão Visitor

Segundo Gamma et al. (2000), o padrão Visitor tem como intenção representar uma operação a ser executada nos elementos de uma estrutura de objetos. Visitor permite definir uma nova operação sem mudar as classes dos elementos sobre os quais opera.

A definição acima mostra que sua ideia é separar as operações que serão executadas em determinada estrutura de sua representação. Assim, incluir ou remover operações não terá nenhum efeito sobre a interface da estrutura, permitindo que o resto do sistema funcione sem depender de operações específicas.

Conforme Brizeno (2013), o padrão Visitor oferece uma excelente alternativa quando é necessário realizar uma série de operações sobre um conjunto de dados, dado que estas operações são pouco estáveis, ou seja, sofre alterações constantemente. Desta forma, para decidir pela utilização deste padrão, é necessário ter certeza de que a estrutura dos elementos seja bem estável (não sofra alterações ao longo do projeto) e que a interface desta estrutura permita acesso suficiente para os objetos visitantes. Elementos podem ter interfaces diferentes, contando que estas interfaces sejam estáveis e provejam acesso às classes visitantes.

Abaixo serão mostrados alguns pontos positivos e negativos da utilização do padrão Visitor segundo Pietro (2013):

Pontos positivos:

- A adição de novas operações torna-se fácil usando o padrão Visitor;
- Acomoda numerosos algoritmos para a mesma estrutura de objetos heterogêneos;

- Visita classes na mesma ou em diferentes hierarquias;
- Chama métodos específicos de um tipo em classes heterogêneas sem realizar conversões de tipos;
- Um visitante reúne operações relacionadas e separa as operações não-relacionadas.

Pontos negativos:

- Pode quebrar o encapsulamento de classes visitadas;
- Uma nova classe a ser visitada requer um novo método *accept*, juntamente com um método *visit* em cada Visitor;
- Adiciona complexidade quando uma interface comum poderia tornar homogêneas classes heterogêneas.

2.2.5.3.12 Padrão Mediator

Segundo Gamma et al. (2000), o objetivo do padrão *Mediator* é definir um objeto que encapsula a forma como um conjunto de objetos *interage*. O *Mediator* promove o acoplamento fraco ao evitar que os objetos se refiram uns aos outros explicitamente e permite variar suas interações independentemente.

Conforme Ribeiro (2013b), o padrão *Mediator* é utilizado quando é necessário que um conjunto de objetos se comunique de maneiras bem-definidas, porém complexas. As interdependências resultantes são desestruturadas e difíceis de entender.

Abaixo serão explanados alguns pontos positivos e negativos da utilização do padrão *Mediator* segundo Ribeiro (2013b):

Pontos posivitos:

- Ele limita o uso de subclasses: Um mediador localiza o comportamento que, de outra forma, estaria distribuído entre vários objetos. Mudar este comportamento exige a introdução de subclasses somente para o Mediador.
- Ele desacopla colegas: Um mediador promove um acoplamento fraco entre colegas. As classes *Colleague* e *Mediator* podem variar e ser

reutilizadas independentemente.

- Ela abastrai a maneira como os objetos cooperam. Tornando a medição um conceito independente e encapsulando-a em um objeto, permite-lhe focalizar na maneira como os objetos interagem independentemente do seu comportamento individual.

Pontos negativos:

- Ele centraliza o controle: o padrão *Mediator* troca a complexidade de interação pela complexidade no mediador.

2.2.5.3.13 Padrão Flyweight

Conforme Gamma et al. (2000), a intenção do padrão *Flywieght* é usar o compartilhamento para suportar eficientemente grandes quantidades de objetos de granularidade fina.

Ainda conforme Gamma et. Al (2000), as principais consequências de se utilizar o padrão *Flyweight* são:

- Redução no número total de instâncias;
- Todos os estados extrínsecos são computados ou armazenados.

2.2.5.3.14 Padrão Template Method

Segundo a Rodrigues (2013), o padrão *Template Method* tem como proposta facilitar a implementação de cenários onde se deseja encapsular algoritmos, fornecendo as classes derivadas o poder de definir as pré-condições do algoritmo e deixando a classe Base responsável pela estrutura do algoritmo, pré-condições globais que fazem sentido a todas as classes derivadas, bem como, as pós-condições.

2.2.5.3.15 Padrão Chain of Responsability

Segundo Gamma et al. (2000), este padrão tem como função representar um encadeamento de objetos para realizar o processamento de uma série de requisitos diferentes.

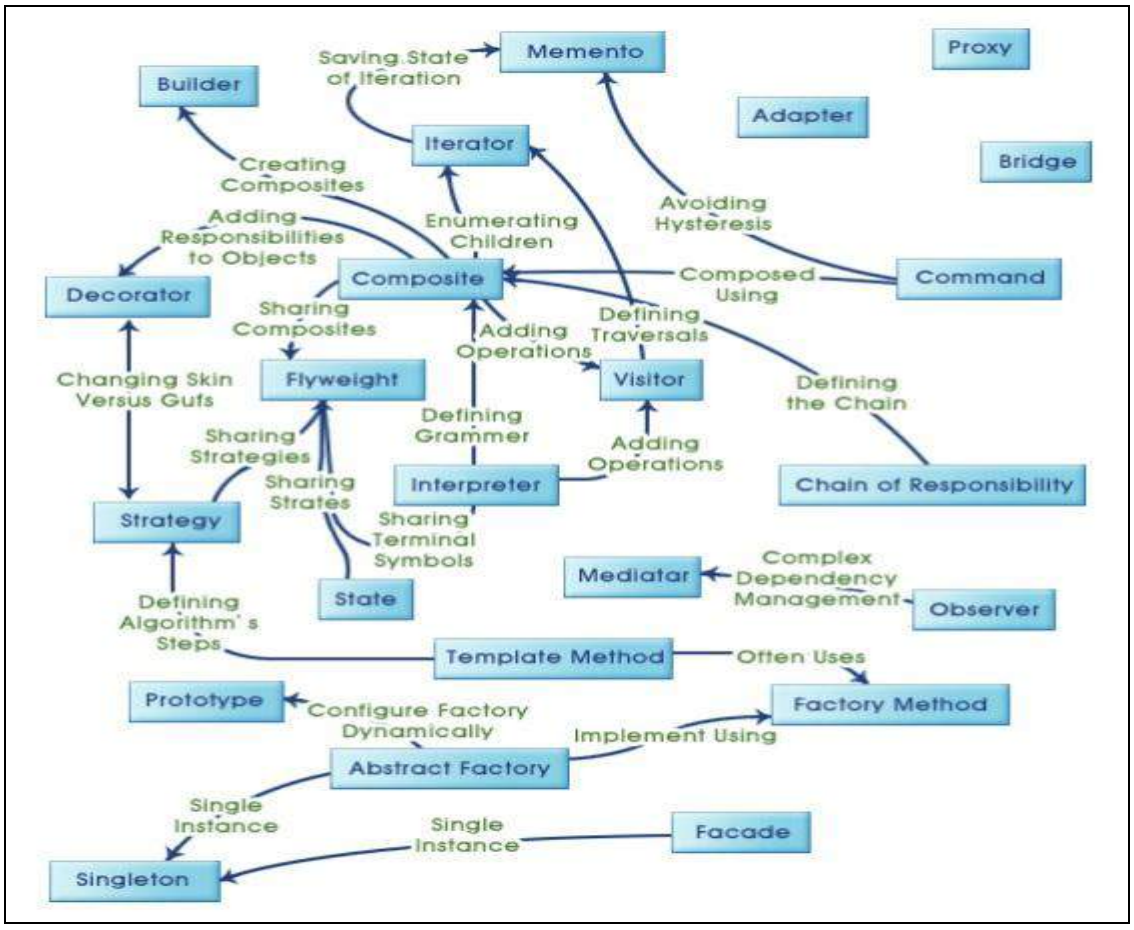
A seguir serão mostradas algumas vantagens deste padrão segundo Furtado

(2013):

- Evita acoplamento do transmissor de uma requisição com seus receptores, fazendo com que mais de um projeto tenha a chance de manipular a requisição;
- Encadeia os objetos receptores e passa a requisição ao longo dessa cadeia até que um objeto possa manipulá-lo;
- Adiciona flexibilidade;
- Permite que um conjunto de classes atue como uma classe; eventos produzidos em uma classe podem ser enviados para outras classes dentro da composição.

2.2.6 Relação Entre os Padrões de Projeto segundo GOF

FIGURA 15 - Relação entre padrões de projeto segundo GOF



FONTE: (GAMMA ET AL., 2000)

A Figura 15 mostra a relação entre os padrões de projeto segundo GOF, e segundo Gamma et al. (2000), existem muitas maneiras de organizar os padrões de projetos.

2.3 Padrões *Web*

2.3.1 O que são os padrões Web

Quando se fala de normas para a web, trata-se, na prática, de três componentes independentes: estrutura, apresentação e comportamento, ou ainda de linguagens estruturais (XHTML, XML e HTML que significam *EXtensible HyperText Markup Language*, *EXtensible Markup Language* e *HyperText Markup Language* respectivamente), linguagens de apresentação (CSS, *Cascading Style Sheets*), dentre outras.

Segundo Wyke (2005), a utilização de padrões para *Web* é extremamente vantajosa, pois proporciona um maior controle sobre a página. Quando é dito que uma página é compatível com os padrões, significa que o documento consiste de HTML ou XHTML válido, utiliza CSS para leiaute, é bem estruturado e semanticamente correto.

Esses fatores podem garantir que o site seja acessado por qualquer dispositivo, seja ele móvel, tátil, *desktop* etc.

Assim, segundo Zeldman (2003), sites construídos de acordo com estes padrões, custam menos, funcionam melhor e são acessíveis a mais pessoas e dispositivos.

2.3.2 Porque utilizar Padrões Web

Segundo Reis (2007), o desenvolvimento tradicional de *websites* tem sido empregado com o objetivo de fazê-los parecerem perfeitos em alguns navegadores principais. Desenvolver nos padrões significa utilizar a web como uma ampla ferramenta acessível por um grande número de usuários e uma variedade de dispositivos, além de diversos outros benefícios, como:

- Separação de conteúdo e apresentação, para tornar o código limpo e correto;
- Manutenção e desenvolvimento simplificados: usar HTML semântico e bem-estruturado torna mais fácil à compreensão do código e reduz custos e trabalho desnecessário;
- Compatibilidade com as leis e diretrizes de acessibilidade sem comprometer a beleza, o desempenho e sofisticação;
- Adaptação simplificada: tornam-se funcionais em vários navegadores e plataformas, sem a dificuldade e a despesa de criar versões separadas, suportando dispositivos não tradicionais – desde acessórios sem fio e telefones celulares, até leitores de *Braille* e de tela usados por usuários com deficiências físicas – apenas por vincular a um arquivo *CSS* diferente;
- Compatibilidade com versões futuras: sites projetados utilizando padrões definidos e códigos válidos reduzem o risco de novos navegadores serem incapazes de renderizar a codificação utilizada;
- Maior velocidade no carregamento da página: menos HTML resulta em arquivos de tamanho menor e *download* mais rápido, fazendo com que os navegadores modernos renderizem as páginas mais rapidamente;
- Melhor posicionamento em mecanismos de busca.

2.3.3 Definição de Padrões Web

Padrões Web segundo a *W3C (World Wide* Web *Consortium*) – consórcio de empresas de tecnologia que desenvolvem padrões para a criação e a interpretação dos conteúdos para a web - são recomendações, as quais são destinadas a orientar os desenvolvedores para o uso de boas práticas que tornam a web acessível para todos.

Com esses padrões, o site desenvolvido pode ser acessado por qualquer pessoa ou tecnologia.

2.3.4 Por que utilizar o W3C?

Utilizando os padrões estabelecidos pelo *W3C*, ao desenvolver um site, está sendo garantido que qualquer pessoa poderá acessar o site, através de qualquer aparelho ou software.

Os sites que não seguem estes padrões, talvez não sejam interpretados corretamente pelos navegadores e programas para pessoas com necessidades especiais, ou seja, são sites limitados e podem não oferecer uma experiência agradável a todos os visitantes.

Porém, segundo Gérman (2000), os padrões web classificam-se em domínios de aplicações e interesses específicos:

- Padrões de Construção de Componentes: estes padrões resolvem problemas relacionados ao modo como são combinados componentes básicos mais complexos;
- Padrões de Navegação: são padrões que tratam problemas relacionados ao modo como uma aplicação é interligada e o modo como o leitor é orientado;
- Padrões de Apresentação: são padrões relacionados ao modo como o conteúdo é apresentado ao usuário do sistema quando em execução;

2.3.5 Dificuldade na utilização dos padrões web

Mesmo com as vantagens explanadas anteriormente, ainda são encontradas muitas dificuldades na utilização dos padrões web segundo Reis (2007), e algumas delas serão mostradas abaixo.

Diferença na implementação dos padrões por parte dos navegadores: apesar das recomendações do *W3C* terem sido criadas com o propósito de padronizar a codificação das páginas web, os navegadores atuais não apresentam completo suporte a estes padrões e não conseguem implementar corretamente todas as recomendações do *W3C (REIS, 2007).*

Segundo Reis (2007), a popularidade dos editores visuais WYSIWYG também

é uma das dificuldades na utilização dos padrões web, pois, o termo WYSIWYG que é o acrônico de "*What You See Is What You Get" ("*O que você vê é o que você tem") e refere-se a ferramentas de desenvolvimento que permitem desenvolver *websites* sem precisar ter conhecimento de *tags* HTML, básicas, ao mesmo tempo em que permite visualizar o site com a mesma aparência que ele terá no navegador depois de pronto.

Apesar de alguns destes editores visuais terem evoluído nestes últimos anos e oferecerem um desenvolvimento mais próximo aos padrões, o grande problema está na mentalidade criada com a utilização dos editores WYSIWYG. Estes editores geram comodidade nos desenvolvedores que criam o hábito de confiar apenas na ferramenta, contribuindo com códigos semanticamente incorretos em diversas páginas da internet.

Dificuldades de aprendizado: A adoção dos padrões não costuma ser algo simples para desenvolvedores acostumados a leiautes em tabelas, pois envolve o domínio de uma nova linguagem (CSS) e uma mudança de mentalidade na forma de desenvolvimento que passa a ser baseado no conteúdo e não mais no visual (REIS, 2007).

2.4 Padrões de Arquitetura

Padrões arquiteturais expressam um esquema de organização estrutural fundamental para sistemas de software (BUSCHMANN et al., 1996).

Segundo Sommerville (2011), padrões arquiteturais permitem a construção de uma arquitetura aderente a certas propriedades.

> "Uma arquitetura de software envolve a descrição de elementos arquiteturais dos quais os sistemas são construídos, interações entre esses elementos, padrões que guiam suas composições e restrições sobre estes padrões" (PFLEEGER, 2003).

2.4.1 Estilos arquiteturais

Um estilo arquitetural define uma família de sistemas em termos de um padrão de organização estrutural (SHAW e GARLAN, 1996).

2.4.2 Padrões Estruturais

Cliente/Servidor: é uma arquitetura organizada como um conjunto de serviços requer uma estrutura de rede para clientes acessarem os serviços e servidores disponíveis, porém, os servidores não sabem quem são os clientes;

2.4.3 Padrões de Sistemas Distribuídos

2.4.3.1 Broker

É um padrão que permite estruturar aplicações distribuídas através de componentes desacoplados, que interagem via invocações remotas. Coordena toda a comunicação através de encaminhamento de requisições, resultados e exceções. *Broker* provê a integração de duas tecnologias: a de distribuição e a de objetos (BUSCHMANN et al., 1996);

2.4.3.2 Filters and pipes (*filtros e dutos)*

É um padrão de organização da dinâmica de um sistema, onde os dados de entrada se movem pelos dutos, são transformados pelos filtros até serem convertidos em dados de saída (BUSCHMANN et al., 1996);

2.4.3.3 Camadas

Auxilia na estruturação de aplicações que possam ser decompostas em grupos de sub-tarefas, nas quais cada grupo está em um nível de abstração particular (BUSCHMANN et al., 1996);

2.4.4 Padrões Interativos

2.4.4.1 MVC

Padrão que divide uma aplicação interativa em três componentes: Modelo (model), Visão (view) e Controller (controle). (BUSCHMANN et al., 1996).

Em outras palavras o MVC tem como principal objetivo separar dados ou lógicos de negócio (model) da interface (view), e o fluxo da aplicação (controller), a ideia é permitir que uma mensagem da lógica de negócios pudesse ser acessada e visualizada através de várias interfaces. Na arquitetura MVC, a lógica de negócios não sabe quantas e nem quais as interfaces com o usuário está exibindo seu estado, a camada view não se importa de onde está recebendo os dados, mas ela tem que garantir que sua aparência reflita o estado do modelo, ou seja, sempre que os estados de modelo mudam, o modelo notifica as view para que as mesmas atualizem-se.

2.4.4.1.1 Caracteristicas da arquitetura MVC

Segundo Pressman (1995), algumas das características do padrão arquitetural MVC são:

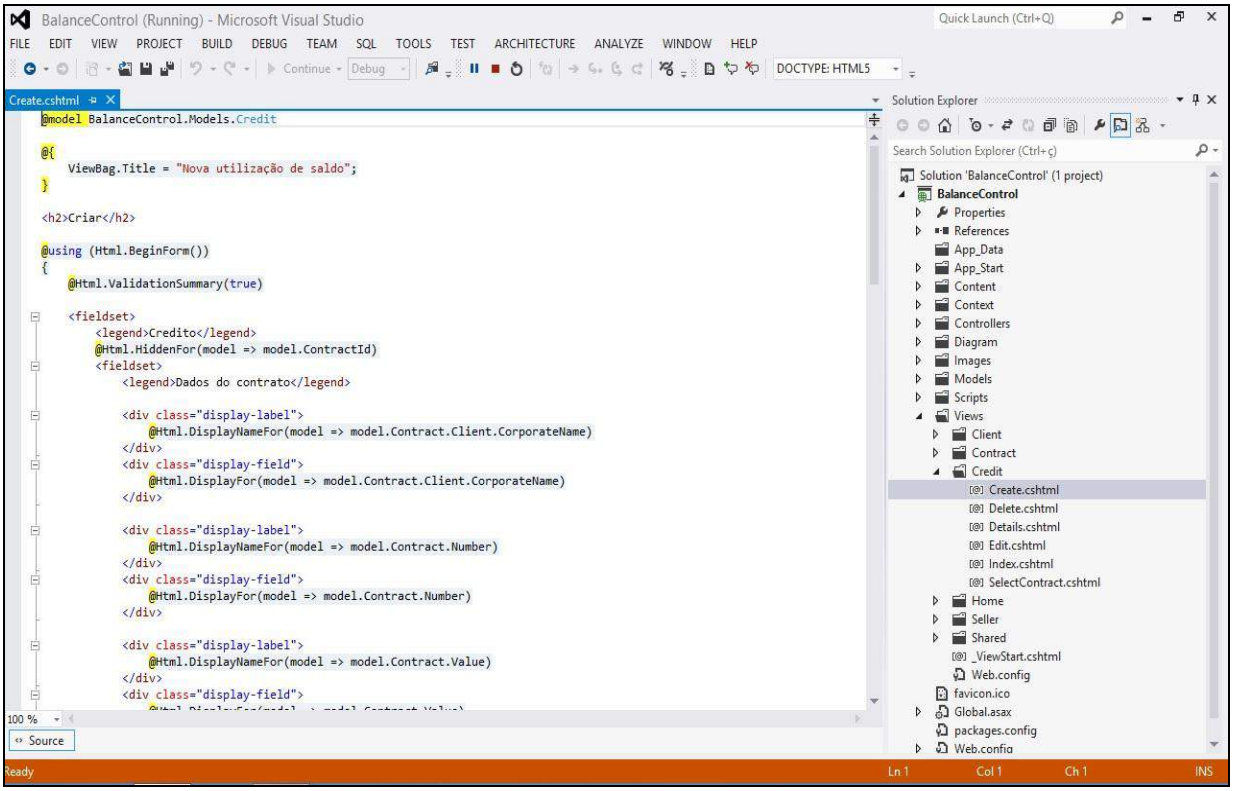
- Separação entre os códigos, View e Controller que gerencias as relações entre Model e a View;
- Divide as responsabilidades, ou seja, programadores na programação e web designers na construção visual do software;

2.4.4.1.2 Camada View

Segundo Pressman (1995), a camada View, é a camada de apresentação com o usuário, é a interface que proporcionará à entrada de dados e a visualização de respostas geradas, nas aplicações web é representado pelo HTML que é mostrado pelo browser, geralmente a visão contém formulários, tabelas, menus e botões para entrada e saída de dados. A visão deve garantir que sua apresentação reflita o estado do modelo, quando os dados do modelo mudam, o modelo notifica as visões que dependem dele, cada visão tem a chance de atualizar-se.

A Figura 16 demonstra um exemplo de utilização da camada *View*, onde é definido o que será apresentado ao usuário.

FIGURA 16 - Exemplo de representação da Camada *View*



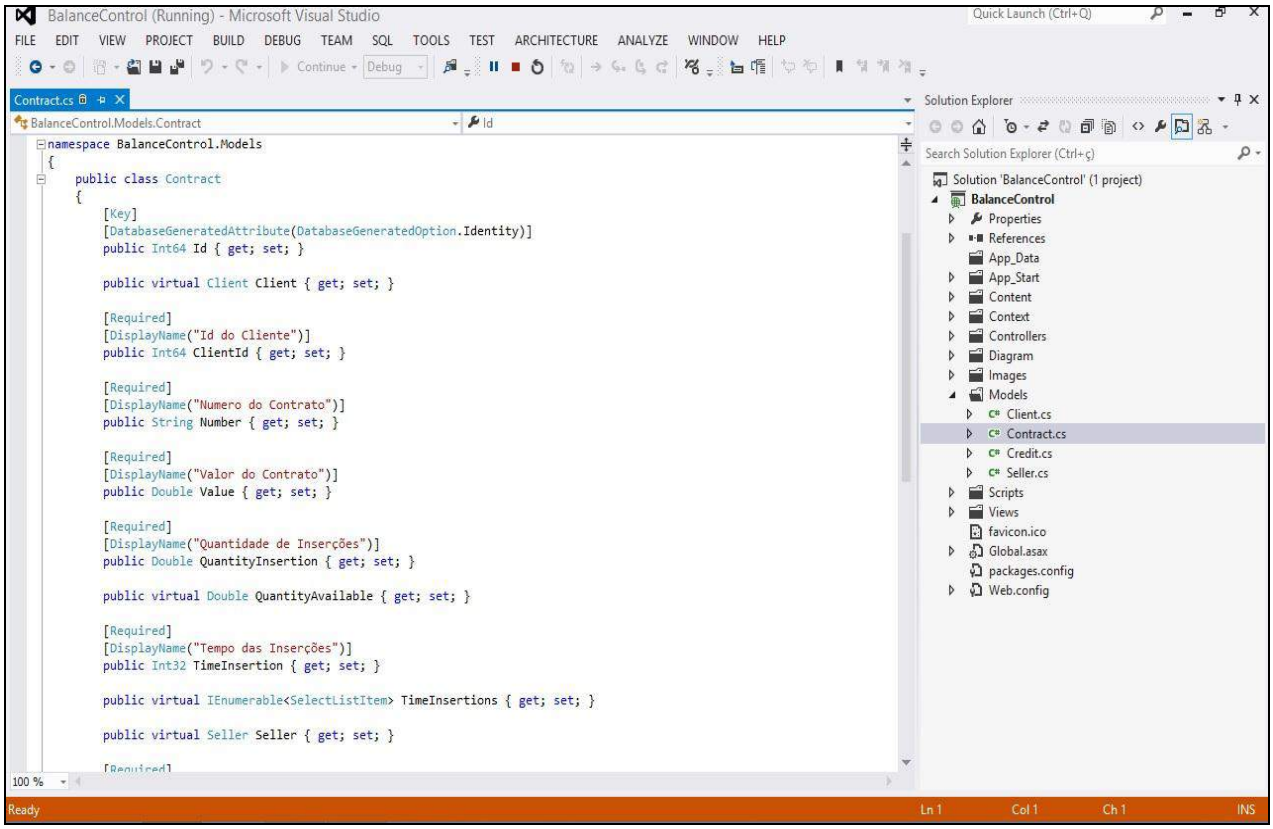
FONTE: (BRUNO RAMOS, 2013)

2.4.4.1.3 Camada Model

Conforme Pressman (1995), a camada *Model* é a que contém a lógica da aplicação, é responsável pelas regras de negócio, para sistemas persistentes, o modelo representa a informação (dados) dos formulários e as regras SQL para manipular dados do banco, o modelo mantém o estado persistente do negócio e fornece ao controlador a capacidade de acessar as funcionalidades da aplicação, o modelo é o principal responsável por toda a aplicação e deve representar o modelo, atua isoladamente e não tem conhecimento de quais serão a ou as interfaces que terá de atualizar, o modelo somente acessa a base de dados e deixa os dados prontos para o controlador, este por sua vez encaminha para a *View* certa.

A Figura 17 mostra a camada *Model*, como é sua implantação, deixando clara a sua função e utilização, que é responsável pelas regras de negócios do sistema.

FIGURA 17 - Exemplo de utilização da camada *Model*



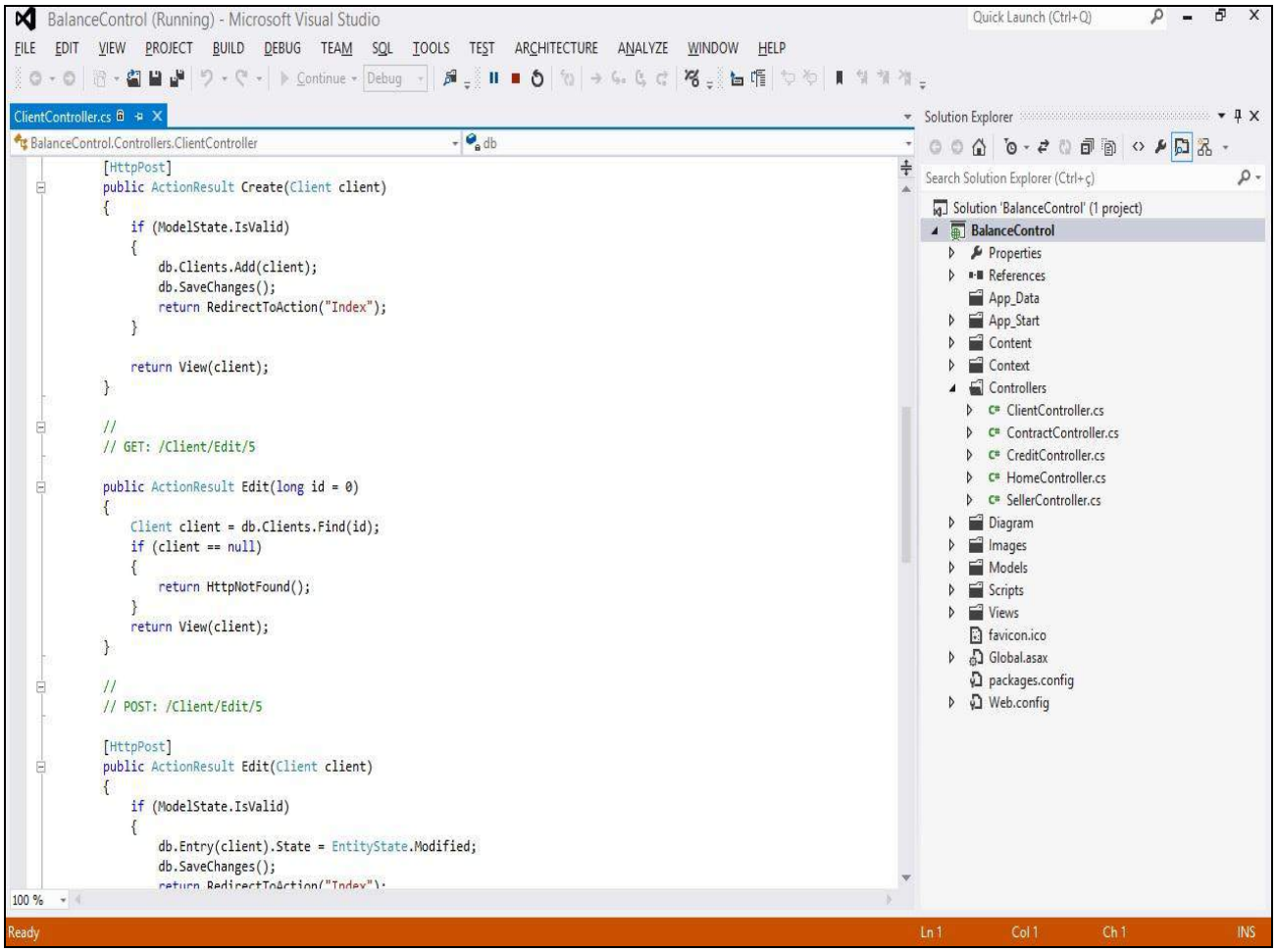
FONTE: (BRUNO RAMOS, 2013)

2.4.4.1.4 Camada Controller

Conforme Pressman (1995), a *Controller* ou controlador funciona de intermediário entre a camada de apresentação e a camada de negócios, sua função como já diz é controlar e coordenar o envio de requisições feitas entre a visão e o modelo. O *Controller* define o comportamento da aplicação, o *Controller* é quem interpreta as solicitações (cliques, seleções de menus) feitas por usuários com bases nestes requerimentos, o *Controller* comunica-se com o modelo que seleciona a *View* e atualiza-se para o usuário, ou seja, o *Controller* controla e mapeia as ações.

A Figura 18 mostra a utilização e implementação da camada *Controller*, camada esta que é responsável pelo intermédio entre a camada *View* e a camada *Model*.

FIGURA 18 - Representação da camada *Controller*



FONTE: (BRUNO RAMOS, 2013)

2.4.4.1.5 Camada de mecanismo de eventos

Segundo Pressman (1995), embora o MVC só contenha três camadas, há outra camada fundamental para o bom andamento da arquitetura, esta é um mecanismo de eventos necessário a comunicação entre outros três elementos, este elemento permite uma comunicação assíncrona que é invocada quando algum evento interessante acontece, esta quarta camada contém os *beans* de identidade onde se localizam os métodos *get* e *set* das classes.

2.4.4.1.6 Padrões de projeto aplicados na arquitetura MVC

Segundo Gamma et. Al (2000), o MVC usa outros padrões de projeto, tais como *Factory Method*, para especificar por falta (by default) a classe controladora

para uma visão e *Decorator,* para acrescentar capacidade de rolagem (*scrolling)* a uma visão. Mais os principais relacionamentos do MVC são fornecidos pelos padrões *Observer, Composite* e *Strategy.*

2.4.4.1.7 Padrão Composite na camada View

O padrão *Composite* é um padrão estrutural usado na camada de visão onde os componentes das telas são compostos. "[...] Quando o controlador determina a visualização que atualize a tela, esta, tem de transmitir a ordem ao componente mais alto nível, porque o padrão *Composite* cuida do restante." (FREEMAN & FREEMAN, 2007).

2.4.4.1.8 Padrão Observer na camada Model

O *Model* pode fazer uso do padrão *Observer* que separa a visão do estado de um objeto do próprio objeto, permitindo que sejam fornecidas visões alternativas mantendo os objetos interessados constantemente informados sobre suas mudanças de estado.

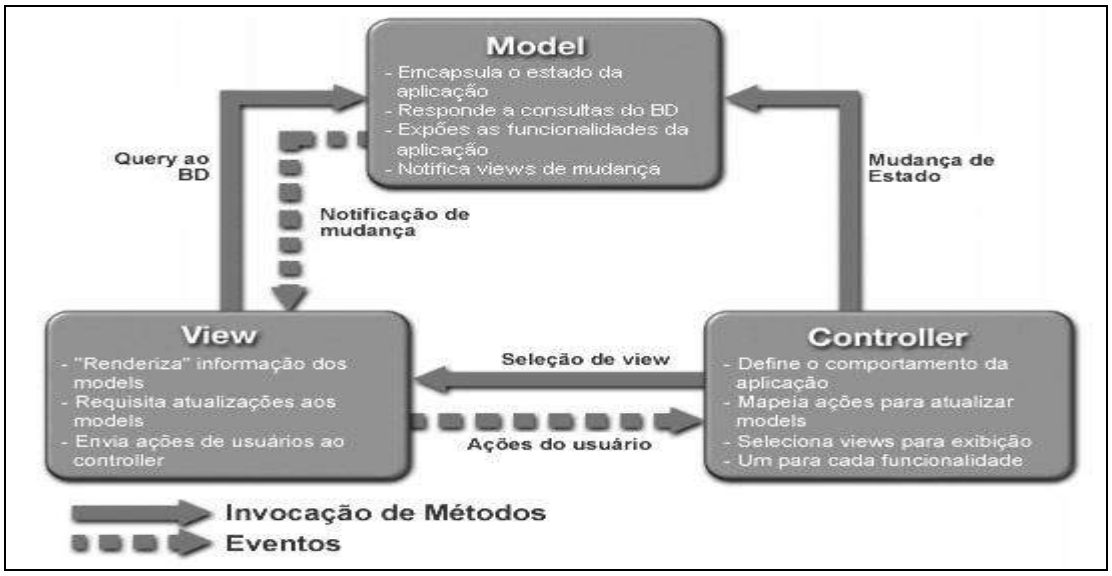
Conforme Gamma et. Al (2000), o padrão *Observer* mantém o modelo totalmente independente das visualizações e controladores, o que nos permite utilizar múltiplas visualizações ao mesmo tempo, como, por exemplo, graficamente ou como texto partindo do mesmo modelo.

2.4.4.1.9 Padrão Strategy na cama Controller

Segundo Freeman e Fremann (2007), a visualização e o controlador utilizam uma estratégia que é fornecida pelo controlador. A visualização só precisa se preocupar com os aspectos visuais do aplicativo, porque todas as decisões sobre o comportamento da interface são delegadas ao controlador, o uso deste padrão mantém a visualização desconectada do modelo, porque a responsabilidade pela iteração com o modelo por executar as solicitações do usuário cabe apenas ao controlador, a visualização não tem a mínima ideia de como isto é feito.

Em seguida será mostrada a estrutura do padrão MVC:

FIGURA 19 - Estrutura de funcionamento do padrão MVC



FONTE: (FREEMAN E FREEMAN, 2007)

Com relação ao protótipo proposto, a camada *View* é composta pelas *Views Client, Contract, Credit, Home* e *Seller.* A camada *Model* é composta pelas classes *Client, Contract, Credid* e *Seller.* Por último, a camada *Controller*, é composta pelos *Controllers, ClienteController, ContractController, CreditController, HomeController* e *SellerController.*

2.4.5 Padrões Adaptáveis

2.4.5.1 Micro Kernel

Aplica-se a sistemas de software que devem ser aptos a se adaptarem a alterações nos requisitos do sistema. Separa a funcionalidade central do sistema das partes específicas do usuário e da funcionalidade estendida e serve também como um conector para ligar estas extensões e coordenar suas colaborações (BUSCHMANN et al., 1996).

2.5 Quadro Comparativo com as funções dos padrões de projeto segundo GOF

No quadro 1 é mostrado o nome de cada padrão e sua função de forma resumida e objetiva.

QUADRO 1 - Nome e função dos padrões de projeto segundo GOF de forma simplificada

Nome do Padrão	Função do Padrão
Factory Method	Definir uma interface para criar um objeto, deixando as subclasses decidirem que classes instanciar.
Abstract Factory	Fornecer uma interface para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.
Builder	Separar a construção de objetos complexos da sua representação de forma que o mesmo processo de construção possa criar diferentes representações.
Prototype	Especificar os tipos de objetos a serem criados usando uma instância-protótipo e criar novos objetos pela cópia deste protótipo.
Singleton	Permitir criar objetos únicos para os quais há apenas uma instância.
Class e Objetct Adapter	Converter a interface de uma classe em outra interface.
Bridge	Desacoplar uma abstração da sua implementação, de modo que as duas possam variar independentemente.
Composite	Permitir que seja composto objetos em estrutura de árvore para representar hierarquias parte-todo.
Facade	Ocultar toda a complexidade de uma ou mais classes através de uma *Facade* (fachada).
Decorator	Anexar responsabilidades adicionais a um objeto dinamicamente
Command	Encapsular uma solicitação como objeto.

Iterator	Fornecer um forma de acessar sequencialmente os elementos de um objeto agregado sem expor a sua representação subjacente.
State	Permitir que um objeto altere seu comportamento quando o seu estado interno mudar.
Proxy	Fornecer um substituto ou representante de outro objeto para controlar o acesso a ele.
Memento	Armazenar o estado interno do objeto originador.
Iterpreter	Dada uma linguagem, criar uma representação para a gramática da linguagem, juntamente com um interpretador.
Strategy	É um conjunto de algoritmos encapsulados e intercambiáveis, sendo que a estratégia permite que o algoritmo se diversifique independentemente dos clientes que os utilizam.
Observer	Representar uma relação 1-N (de um para muitos) entre objetos.
Visitor	Representar uma operação a ser executada nos elementos de uma estrutura de objetos.
Mediator	Definir um objeto que encapsula a forma como um conjunto de objetos interage.
Flyweight	Usa o compartilhamento para suportar eficientemente grande quantidades de objetos de granularidade fina.
Template Method	Facilitar a implementação de cenários onde se deseja encapsular algoritmos.
Chain Of Responsability	Representar um encadeamento de objetos para realizar o processamento de uma série de requisitos diferentes.

FONTE: (BRUNO RAMOS, 2013)

2.6 Conclusão

O presente capítulo abordou conceitos básicos e exemplos de padrões de desenvolvimento, bem como detalhes de alguns destes padrões, e mostrou-se de grande importância para as próximas etapas do trabalho.

Foram explanados conceitos, características, exemplos e funcionalidades de padrões de desenvolvimento, mostrando a importância destes padrões no desenvolvimento de um sistema.

No que diz respeito a padrões web foi mostrado conceitos, diferença na classificação entre os padrões e também a dificuldade na utilização dos padrões web.

Por último foi visto conceitos e exemplos de padrões arquiteturais, sendo esta a parte que será a de maior valia para o resto do projeto, pois no próximo capítulo se dará inicio a parte de modelagem do protótipo, e este será desenvolvido utilizando o padrão arquitetural MVC devido a necessidade da empresa em implantar um sistema sem a necessidade de instalação em cada cliente e com custo menor em relação as demais formas de software.

3 MODELAGEM DO SISTEMA

Neste capítulo apresenta-se o software utilizado e a modelagem do protótipo, com o principal objetivo de esclarecer suas funcionalidades e estrutura do sistema a ser desenvolvido.

3.1 Sumário executivo

O sistema proposto é responsável por controlar os registros de entrada e saída de contratos de uma empresa, além de controlar as alterações e utilizações destes contratos e manter as informações no banco de dados.

O objetivo maior do sistema é aumentar a segurança e a confiabilidade das informações no que diz respeito ao controle de contratos, proporcionando também maior facilidade e rapidez no processo.

O protótipo proposto é desenvolvido utilizando o padrão arquitetural interativo MVC, pois além da necessidade da empresa em ter um sistema web, este padrão utiliza como *default* outros três padrões de projeto, sendo eles: o padrão *composite*, que é usado na camada *View*; o padrão *Observer* na camada *Model*; e o padrão *Strategy* na camda *Controller*.

3.2 Enterprise Architect

É uma ferramenta de análise e design UML, que possibilita o desenvolvimento de software a partir de um conjunto de requisitos, analise de estágios, modelos de design, testes e manutenção. Uma ferramenta multi usuário, com base no

ambiente Windows e projetada para construir softwares robustos e eficazes. Os recursos flexíveis e de alta qualidade permitem o auxílio do desenvolvimento de suas aplicações. Suporta geração e engenharia reversa de códigos fonte para diversas linguagens, incluindo C++, C#, Java, Delphi, VB.Net, Visual Basic e PHP. Suporte para todos os 13 diagramas UML 2.0, entre outros:

Diagramas estruturais:

- Classe;
- Objeto;
- Composição;
- Pacote;
- Componente;
- Distribuição.

Diagrama de comportamento:

- Caso;
- Comunicação;
- Sequencia;
- Vista geral da interação;
- Atividade;
- Estado e Sincronismo.

Extensão:

- Análise (atividades simples);
- Personalizada (para requerimentos, alterações, UI).

3.3 Histórias de Usuário

Nos quadros 2 a 18 serão apresentadas as histórias de usuário.

QUADRO 2 - História de Usuário 01: Cadastro de Clientes

1. **Título**: Cadastro de Clientes	**Obrigatoriedade:** Sim

Como um usuário, eu quero que o sistema cadastre novos clientes.

O quadro 2 diz que o sistema deve poder cadastrar novos clientes, salvando todos em um banco de dados, gerando assim um ID para cada cliente cadastrado, para que estes registros possam ser utilizados quando necessários.

QUADRO 3 - História de Usuário 02: Campos para Cadastro de Clientes

2. **Título:** Campos para Cadastro de Clientes	**Obrigatoriedade:** Sim
Como um usuário, preciso que os campos disponíveis para cadastro de clientes sejam: Razão social, nome fantasia, Endereço, cidade, região e vendedor.	

Como vimos no quadro 3, os campos que devem estar presente no sistema para o cadastro de novos clientes são: razão social, nome fantasia, endereço, cidade, região, vendedor, sendo que região e vendedor já deverão estar devidamente cadastrados em sistema.

QUADRO 4 - História de Usuário 03: Cadastro de Saldos

3. **Título:** Cadastro de Contratos e Utilizações de Saldos	**Obrigatoriedade:** Sim
Como um usuário, preciso que o sistema permita cadastrar novos saldos para os clientes já cadastrados.	

No quadro 4 diz que o sistema deve permitir cadastrar novos saldos, porém, como pré-requisito só poderá ser cadastrado um novo contrato ou utilização se o cliente que está fazendo esta utilização já estiver devidamente cadastrado no banco de dados.

QUADRO 5 - História de Usuário 04: Dados dos Saldos

4. **Título:** Campos de Cadastro de Novos Contratos	**Obrigatoriedade:** Sim
Como um usuário, preciso que os campos disponíveis para cadastro contratos e utilização de saldos sejam: Nome do cliente, Número de contrato, quantidade de inserções ou valor do contrato, tempo dos comerciais.	

Como vimos no quadro 5, os campos que devem estar presente no sistema para o cadastro de novos contratos e utilizações de saldos são:

- Nome do cliente: Este dado deverá ser buscado no banco de dados, na tabela cliente, pois é necessário que o cliente já esteja cadastrado para

lançar um novo contrato para ele.

- O número de contrato: Este número deve ser único, porém, não necessariamente sequencial;
- Quantidade de inserções ou valor do contrato: Alguns contratos são debitados em quantidade, mas alguns clientes preferem ter seus contratos debitados em valores, sendo que desta última forma os mapas de utilização devem vir com os devidos valores;

QUADRO 6 - História de Usuário 05: Utilização de Saldos

5. **Título:** Campos para Cadastro de Utilização de Saldos	**Obrigatoriedade:** Sim
Como um usuário, preciso cadastrar as utilizações de saldo de acordo com as planilhas enviadas, subtraindo o total do contrato, com o total de cada planilha de utilização, cadastrando o mês da utilização, a quantidade e o tempo das inserções.	

O quadro 6 mostra a forma como será feito o processo de utilização de saldo, que será efetuado buscando o contrato do cliente já cadastrado e através da subtração do total do contrato, com o total de cada planilha de utilização, cadastrando o mês da utilização.

QUADRO 7 - História de Usuário 06: Tempo de Inserções

6. **Título:** Tempo de Inserções	**Obrigatoriedade:** Sim
Como um usuário, desejo que sejam pré-definido os tempos de 7, 15, 30 e 60 segundos como tempo de inserções.	

No quadro 7, está sendo apresentado os tempos que o sistema devem ser pré definidos através de um ComboBox, sendo possível ter utilizações de 7, 15, 30 e 60 segundos.

QUADRO 8 - História de Usuário 07: Alteração de Clientes

7. **Título:** Alteração Clientes	**Obrigatoriedade:** Sim
Como um usuário, preciso que os dados dos clientes possam ser alterados quando necessário.	

O quadro 8 mostra que se necessário, é preciso buscar no banco de dados os dados dos clientes já cadastrados e altera-los conforme necessário, não alterando ID ou qualquer outra utilização ou contrato cadastrado neste cliente.

QUADRO 9 - História de Usuário 08: Alteração de Saldos

8. **Título:** Alteração Saldos	**Obrigatoriedade:** Sim
Como um usuário, preciso que os dados e utilizações de saldos possam ser alterados conforme necessidade.	

O quadro 9 pede que o sistema, quando necessário, possa alterar os saldos já cadastrados.

QUADRO 10 - História de Usuário 09: Exclusão de Saldos

9. **Título:** Exclusão de Saldos	**Obrigatoriedade:** Sim
Como um usuário, preciso poder excluir contratos ou/e utilizações quando necessário.	

No quadro 10 está sendo pedido que o sistema possa fazer a exclusão de saldos quando necessário.

QUADRO 11 - História de Usuário 10: Exclusão de Clientes

10. **Título:** Exclusão de clientes	**Obrigatoriedade:** Sim
Como um usuário, preciso poder excluir clientes, desde que este não tenha saldos cadastrados neste cliente.	

No quadro 11 diz que deverá ser permitida a exclusão de clientes, desde que o este não tenha nenhum tipo de cadastro amarrado a ele.

QUADRO 12 - História de Usuário 11: Relatório por Cidade

11. **Título:** Relatórios por Cidade	**Obrigatoriedade:** Sim
Como um usuário, desejo gerar relatórios a partir das cidades cadastradas.	

O quadro 12 mostra que será necessário que o sistema gere relatórios a partir das cidades cadastradas, sendo que neste relatório será exibido a cidade em questão, todos os clientes cadastrados com esta cidade e seus respectivos saldos.

QUADRO 13 - História de Usuário 12: Relatório por Região

12. **Título:** Relatórios por Região	**Obrigatoriedade:** Sim
Como um usuário, desejo poder gerar relatórios a partir das regiões cadastradas.	

O quadro 13 mostra que será necessário que o sistema gere relatórios a partir das regiões cadastradas, sendo que neste relatório será exibido a região solicitada, todos os clientes que sejam desta região e seus respectivos saldos.

QUADRO 14 - História de Usuário 14: Relatório por Vendedor

13. **Título:** Relatórios por Vendedor	**Obrigatoriedade:** Sim
Como um usuário, desejo gerar relatórios a partir de cada vendedor cadastrado no sistema.	

O quadro 14 mostra que será necessário que o sistema gere relatórios a partir dos vendedores cadastrados, sendo que neste relatório será exibido o vendedor em questão, todos os clientes que sejam deste vendedor e seus respectivos saldos.

QUADRO 15 - História de Usuário 15: Impressão de Saldos

14. **Título:** Impressão de Saldos	**Obrigatoriedade:** Sim
Como um usuário, desejo poder imprimir os saldos de acordo com a necessidade, mostrando nessa impressão os seguintes dados: nome, contratos e utilizações do cliente.	

O quadro 15 pede que o sistema possa imprimir quando necessário os saldos, mostrando nessa impressão todos os dados do cliente.

QUADRO 16 - História de Usuário 16: Consultas

15. **Título:** Consultas	**Obrigatoriedade:** Sim
Como um usuário, desejo consultar os saldos conforme necessidade, podendo ver todos os contratos e utilizações de cada cliente.	

O quadro 16 diz que o sistema deverá disponibilizar uma área para efetuar consultas. Podendo ver todos os contratos e utilizações de cada cliente.

QUADRO 17 - História de Usuário 17: Usuário Administrador

16. **Título:** Usuário Administrador	**Obrigatoriedade:** Sim
O usuário administrador terá acesso total ao sistema, tendo todas as permissões liberadas.	

No quadro 17 mostra que o usuário administrador deverá ter todas as permissões liberadas para ele.

QUADRO 18 - História de Usuário 18: Usuário Comum

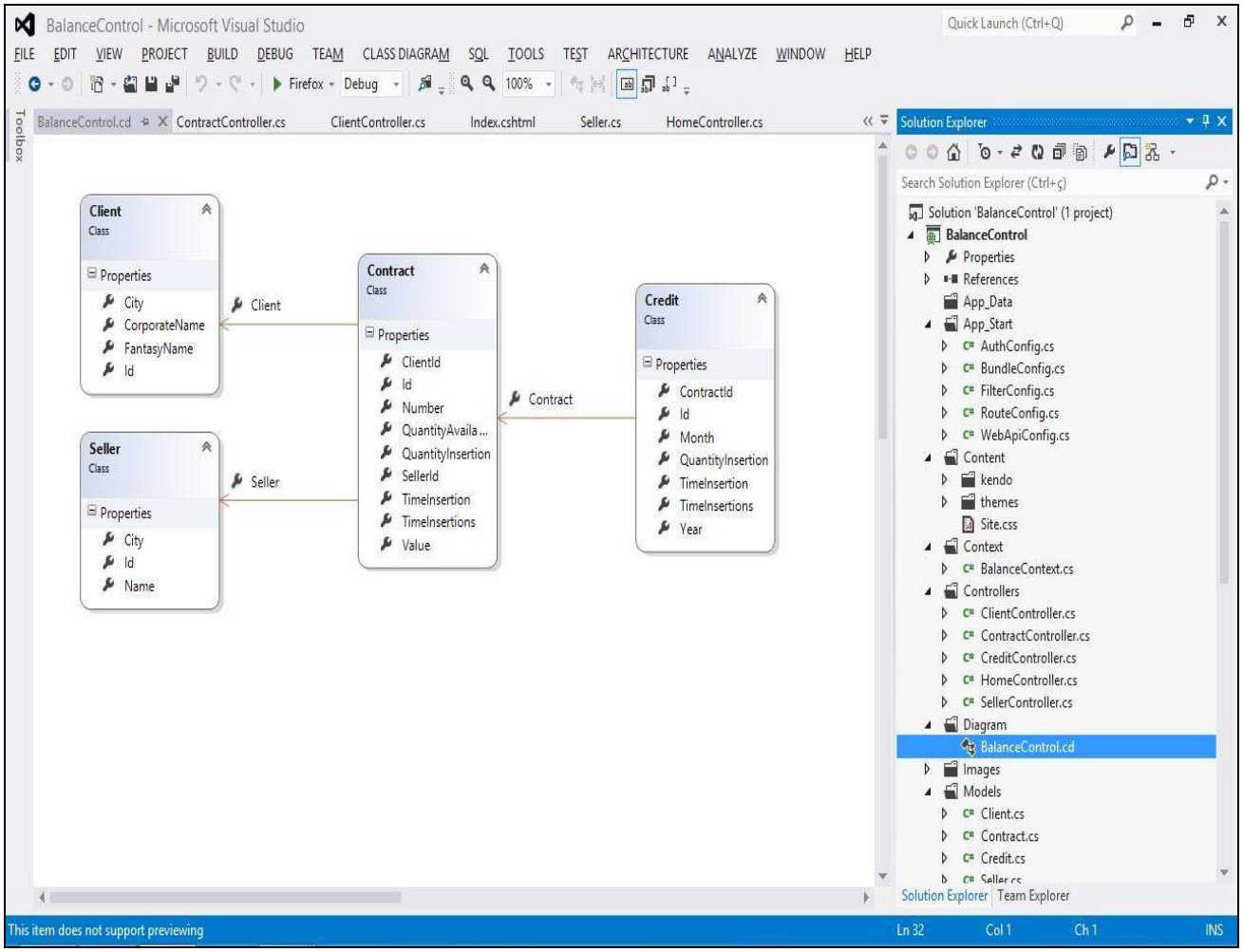
17. **Título:** Usuário Comum	**Obrigatoriedade:** Sim
O usuário comum terá acesso apenas à área de consulta e relatórios referente à sua cidade.	

O quadro 18 diz que o usuário comum deverá ter acesso apenas à área de consulta e relatórios referente à sua cidade.

3.4 Diagrama de Classe

Na Figura 20 serão mostradas as classes do sistema, e a seguir a função de cada uma destas classes.

FIGURA 20 - Diagrama de Classes



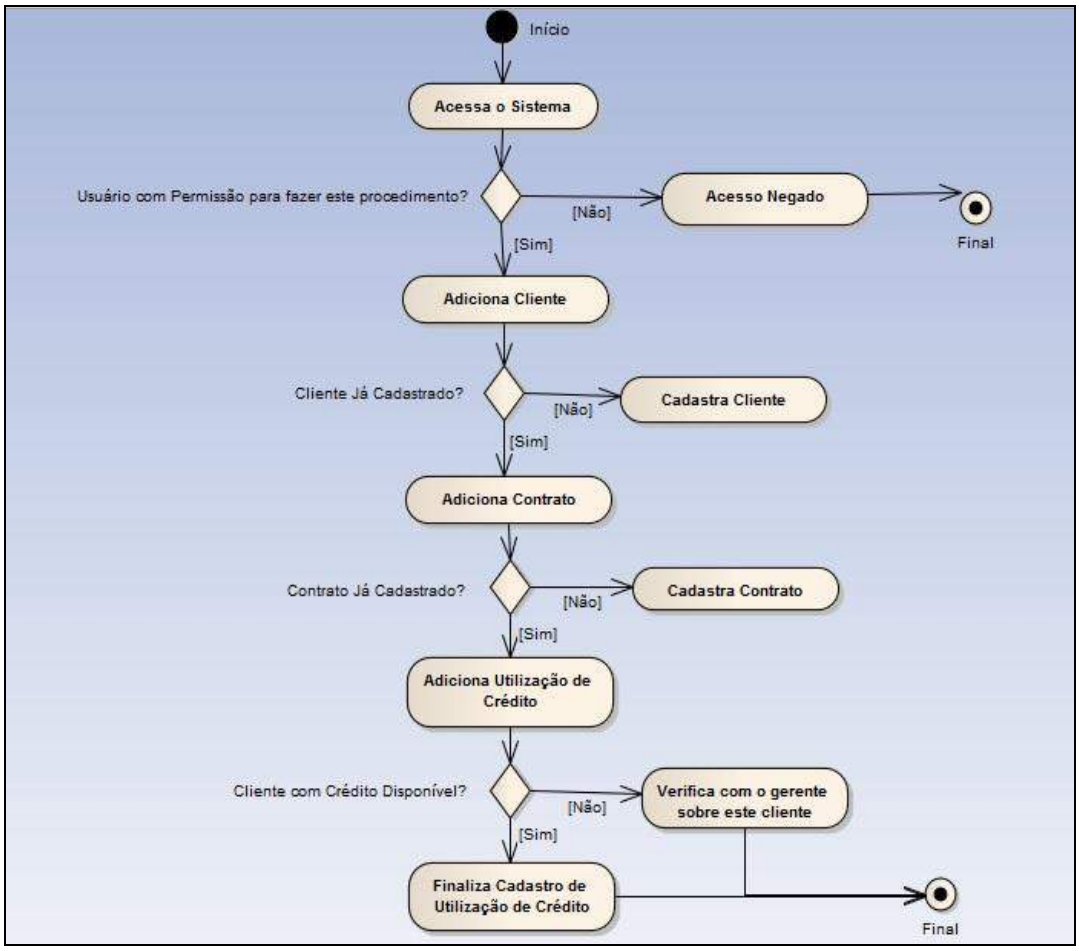
FONTE: (BRUNO RAMOS, 2013)

- Vendedor: Responsável por manter informações sobre todos os vendedores cadastrados pelo usuário no sistema.
- Cliente: Classe responsável por manter os dados dos clientes cadastrados em sistema.
- Contratos: Classe responsável por manter todos os dados e contratos no sistema.

- Creditos: Classe responsável por realizar o processo de utilizações de saldos.

3.5 Diagrama de Atividades

FIGURA 21 - Diagrama de Atividade



FONTE: (BRUNO RAMOS, 2013)

Como mostra a Figura 21, o usuário primeiro efetua o *login* no sistema, caso não tenha permissão para ter acesso a esta parte do sistema o mesmo já recebe um aviso informando que o usuário não possui acesso. Se o usuário tiver acesso a está parte ele deverá cadastrar o cliente, isso se o cliente já não esteja cadastrado em sistema, se esta segunda opção for afirmativa ele poderá cadastrar o contrato, o mesmo

serve para o contrato, só poderá cadastrar se não existir outro contrato de mesmo número no sistema, após este procedimento será necessário constatar se o cliente dispõe de crédito com a empresa, por fim, se tudo estiver certo é só adicionar a utilização de crédito em sistema.

3.6 Conclusão

Neste capítulo foi apresentada modelagem do protótipo do estudo de caso escolhido para demonstrar o uso do padrão arquitetural interativo MVC.

Ao finalizar este capítulo, fica clara a importância da modelagem de um sistema, pois é nesta etapa que é exposto as funcionalidades que o protótipo deve apresentar, facilitando assim o seu desenvolvimento.

Através desta mesma modelagem, é possível visualizar o que é necessário desenvolver, estabelecendo os critérios e objetivos necessários.

4 DESENVOLVIMENTO DO SISTEMA

Neste capitulo é apresentado a implementação do sistema proposto anteriormente, o qual foi desenvolvido utilizando o padrão arquitetural MVC, as tecnologias utilizadas para o desenvolvimento deste protótipo, suas telas e principais códigos.

4.1 Ferramentas utilizadas no desenvolvimento do sistema

Para o desenvolvimento do sistema, foram utilizadas as seguintes ferramentas:

- *Visual Studio 2012*
- *SQL Management Studio Express 2012*

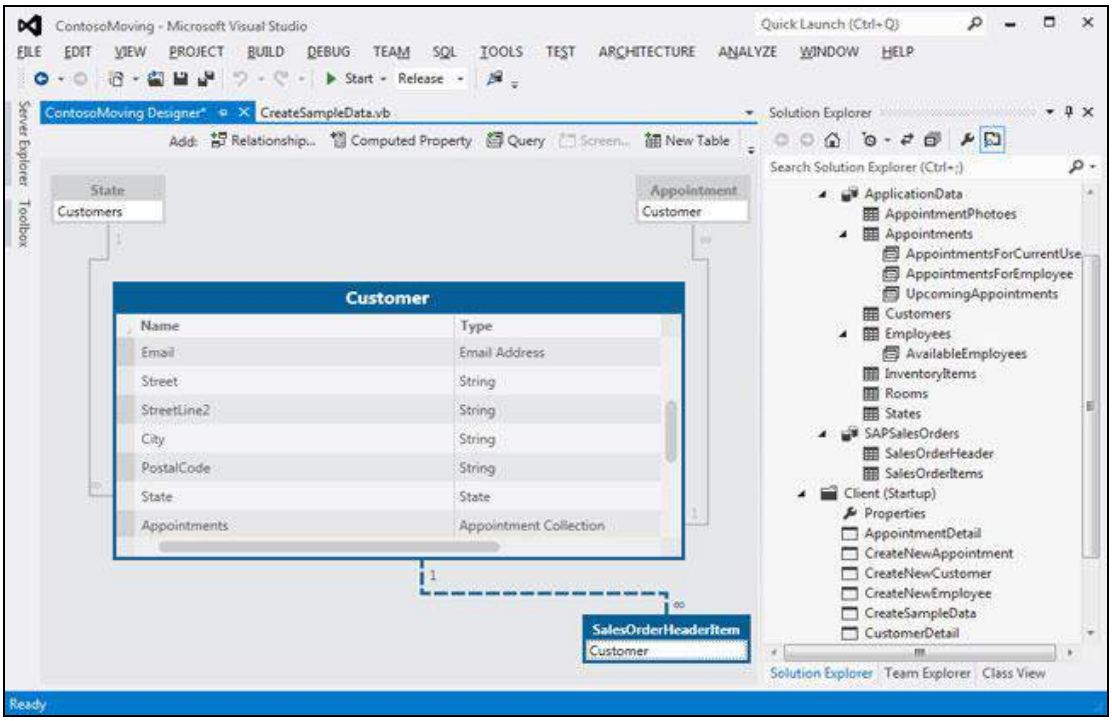
4.1.1 Visual Studio

Segundo o site da Microsoft.com (2013a), o Visual Studio é um conjunto completo de ferramentas de desenvolvimento para construção de aplicações Web ASP.NET, serviços Web *XML*, aplicações desktop e aplicativos móveis. *Visual Basic*, *Visual C# e Visual C++* usam todos o mesmo ambiente de desenvolvimento integrado (IDE), que permite o compartilhamento de ferramentas e facilita a criação de soluções com mistura de linguagens. Além disso, essas linguagens usam a funcionalidade do *.NET framework*, que fornece acesso às tecnologias chaves que simplificam o desenvolvimento de aplicativos Web em ASP e serviços *Web XML*.

Abaixo serão mostradas algumas características do *Visual Studio 2012* (versão utilizada para o desenvolvimento do sistema):

- Logo que abrir o IDE, é notório que toda a interface foi desenhada para simplificar os fluxos de trabalho e fornecer fácil acesso às ferramentas utilizadas no dia a dia. As barras de ferramentas são simples, e os caminhos rápidos para encontrar um código. Tudo isso torna fácil navegar pelo aplicativo e trabalhar de maneira simples e objetiva.

FIGURA 22 - Interface Visual Studio 2012



Fonte: (Microsoft.com, 2013a)

- O Visual Studio 2012 oferece *templates*, *designers* e ferramentas de teste e depuração, ao mesmo tempo o *Blend* para o Visual Studio oferece ferramentas visuais para que você se aproveite de todas as vantagens da nova interface do Windows 8.
- Se tratando de desenvolvimento Web, o *Visual Studio 2012* conta com *templates*, ferramentas de publicação e total suporte para padrões, como *HTML5* e *CSS3*, assim como os avanços mais recentes em ASP.NET, além disso a depuração também está mais fácil com o *Page Inspector*, através da interação com a página que está sendo

codificada.

- O Visual Studio oferece *templates* e opções de publicação, além de ótimas ferramentas para levar aplicativos para o *Windows Azure*.

4.1.2 Microsoft SQL Server Management Studio Express

Segundo o site Microsoft.com (2013b), o *Microsoft SQL Server Management Studio Express* é um ambiente de desenvolvimento integrado gratuito para acessar, configurar, gerenciar, administrar e desenvolver todos os componentes do *SQL Server*. Combina um grupo de ferramentas gráficas e editores de script sofisticados para fornecer acesso ao *SQL Server* a desenvolvedores e administradores de todos os níveis de conhecimento.

4.2 Tecnologias utilizadas no desenvolvimento do sistema

Para o desenvolvimento do sistema, foram utilizados as seguintes tecnologias:

- MVC 4
- Microsoft Entity Framework
- Framework BackBone

4.2.1 MVC 4

Segundo o site *ASP.NET* (2013), o MVC 4 é um *framework* para construção de aplicações *WEB* escaláveis, baseadas em padrões, usando padrões de projetos bem estabelecidos.

Como principais características do MVC 4 temos:

- ASP.NET Web API;
- Modelos de projeto padrão renovado e modernizado;
- Novo modelo de projeto móvel;
- Muitos novos recursos para suportar aplicativos móveis;

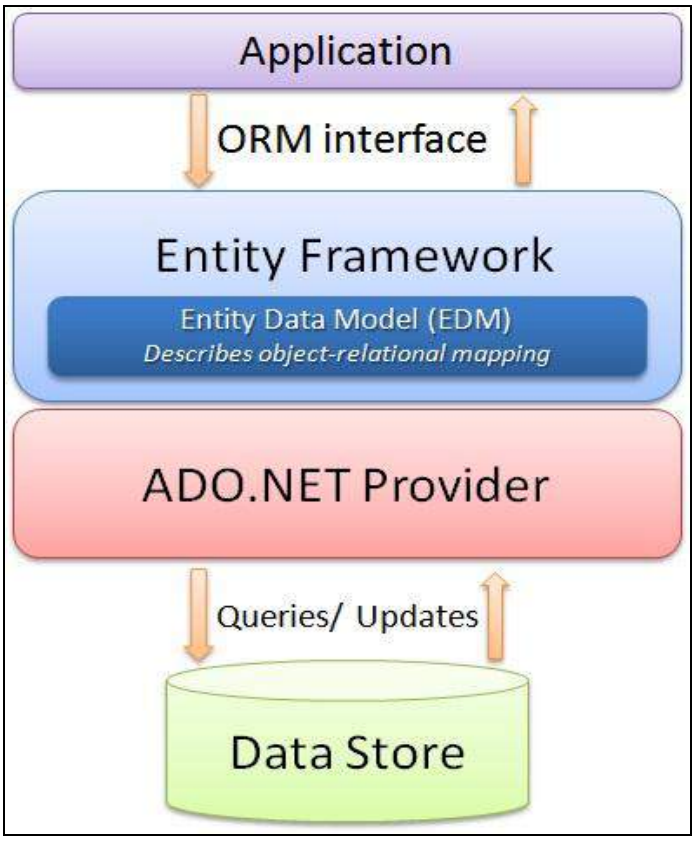
4.2.2 Entity Framework

Segundo Santos (2013), o *Microsoft Entity Framework* é uma ferramenta de mapeamento objeto relacional *(ORM - Object Relational Management)*, que permite aos desenvolvedores trabalhar com classes que correspondem a tabelas em um banco de dados, tornando transparente o acesso a estes dados e principalmente, eliminando a necessidade de escrever código de banco de dados na aplicação.

A comunicação do *Entity Framework* com o banco de dados é feita através do ADO.NET *Provider*, que funciona como um "*driver*" do banco de dados, normalmente desenvolvido pelo próprio fabricante do banco. Desta forma todos os comandos submetidos pelo *Entity Framework* são "traduzidos" para a linguagem do banco de dados através do seu *provider*, gerando os comandos SQL mais adequados a cada operação e principalmente, comandos que tenham o máximo de desempenho.

Na sequencia será mostrado um diagrama do funcionamento do *Entity Framework*:

FIGURA 23 - Diagrama do funcionamento do Entity Framework



FONTE: (SANTOS, 2013)

4.2.3 Framework BackBone

Segundo o site *backbonejs.org*, o *backbones.js* dá estrutura para aplicações web, oferecendo modelos com chave-valor, eventos de ligação e personalização, coleções com uma rica *API (Application Programming Interface)* de funções enumeráveis, visões com manipulações de eventos declarativo, e conecta tudo à sua *API* existente através de uma interface ***RESTful JSON.***

4.3 Cadastros

Neste item é demostrada a implementação de todos os cadastros do protótipo, especificando detalhadamente cada particularidade da implementação utilizando o padrão MVC.

4.3.1 Vendedores

QUADRO 19 - Cadastro de Vendedor

```
1   public ActionResult Create(Seller seller)
2      {
3         if (ModelState.IsValid)
4            {
5               db.Sellers.Add(seller);
6               db.SaveChanges();
7               return RedirectToAction("Index");
8            }
9         return View(seller);
10     }
```

Na linha 3 está sendo mostrado que se os dados digitados forem validos, o processo de criação do novo vendedor poderá seguir, caso contrário enviará uma mensagem de erro. Na linha 5 está sendo adicionado um novo cadastro de vendedor, enquanto na linha 6 os dados deste novo cadastro está sendo gravado no banco, após isso, a página redireciona para a página anterior. Este código está presente na camada *Controller*.

QUADRO 20 - Edição de Vendedor

```
1   public ActionResult Edit(Seller seller)
2      {
3         if (ModelState.IsValid)
4            {
5               db.Entry(seller).State = EntityState.Modified;
6               db.SaveChanges();
7               return RedirectToAction("Index");
8            }
9         return View(seller);
10     }
```

Assim como no quadro 19, na linha 3, está sendo mostrado que se os dados digitados forem válidos, o processo de edição poderá prosseguir. Na linha 5, será procurado no banco de dados o vendedor a que se deseja alterar, e assim ocorre a alteração, sendo que na linha 6 é salvo os novos dados do vendedor, sobreescrevendo os dados antigos. Este código está presente na camada *Controller*.

QUADRO 21 - Exclusão de Vendedor

```
1    public ActionResult DeleteConfirmed(long id)
2       {
3          Seller seller = db.Sellers.Find(id);
4          db.Sellers.Remove(seller);
5          db.SaveChanges();
6          return RedirectToAction("Index");
7       }
```

Na linha 3, é feita a busca no banco do vendedor selecionado para exclusão, enquanto na linha 4 o vendedor é excluído do banco e na linha 5 é salvo as alterações referente a esta exclusão. Este código está presente na camada *Controller*.

4.3.2 Clientes

QUADRO 22 - Cadastro de Clientes

```
1   public ActionResult Create()
2   {
3      ViewBag.SellerId = new SelectList(db.Sellers, "Id", "City");
4      return View();
5   }
6
7   [HttpPost]
8   public ActionResult Create(Client client)
9   {
10     if (ModelState.IsValid)
11        {
12          db.Clients.Add(client);
```

```
13          db.SaveChanges();
14          return RedirectToAction("Index");
15        }
16
17          return View(client);
18  }
```

Este código de cadastro se assemelha muito com o cadastro do quadro 19, tendo apenas como particularidade que para cadastrar um cliente, é necessário ter o vendedor que representa este cliente já cadastrado em sistema, o que está acontecendo na linha 3, em que é mostrada uma lista com todos os vendedores cadastrados no banco de dados.

Em relação aos códigos de edição e exclusão, são iguais aos códigos do vendedor, tendo como única mudança, que o cadastro é de clientes, e não vendedores. Este código está presente na camada *Controller*.

4.3.3 Contratos

QUADRO 23 - Exibição dos Contratos Cadastrados

```
1   public ActionResult Index()
2     {
3      var contracts = db.Contracts.Include(c =>
4   c.Client).Include(c => c.Seller);
5
6     List<Contract> resultList = new List<Contract>();
7
8     foreach (var contract in contracts)
9       {
10       var credits = db.Credits.Where(y => y.ContractId ==
11  contract.Id);
12
13       var quantity = credits.Any() ? credits.Sum(y =>
14  y.QuantityInsertion) : 0;
15
16       contract.QuantityAvailable = contract.QuantityInsertion -
17  quantity;
18
19       resultList.Add(contract);
20      }
21      return View(resultList.ToList());
22    }
```

O quadro 23 mostra como foi desenvolvida a área de exibição dos contratos cadastrados, as linhas 3 e 4 buscam todos os contratos cadastrados através dos dados do cliente, enquanto na linha 6, é criado uma lista com todos os contratos para serem

exibidos na tela. Na linha 8 inicia uma foreach para verificar se o contrato ainda tem saldo disponível, a linha 10 busca todos créditos de um contrato, a linha 13 e 14 verifica se tem algum credito, caso tenha ele faz a soma das quantidades de inserções, caso contrario ele coloca 0. Se o cliente ainda possuir saldo após a utilização, continua aparecendo na tela, caso contrário, não irá mais aparecer. Este código está presente na camada *Controller*.

QUADRO 24 - Cadastro de novo contrato

```
1   public ActionResult Create()
2     {
3        Contract contract = new Contract();
4
5       ViewBag.ClientId = new SelectList(db.Clients, "Id",
6   "CorporateName");
7       ViewBag.SellerId = new SelectList(db.Sellers, "Id", "Name");
8
9       List<SelectListItem> select = new List<SelectListItem>();
10
11    select.Add(new SelectListItem { Text = "7", Value = "7" });
12    select.Add(new SelectListItem { Text = "15", Value = "15" });
13    select.Add(new SelectListItem { Text = "30", Value = "30" });
14    select.Add(new SelectListItem { Text = "60", Value = "60" });
15
16     contract.TimeInsertions = select;
17
18     return View(contract);
19   }
20
```

O quadro 24 mostra o cadastro de novos contratos, a linha 5 faz uma seleção, podendo ser adiciona apenas clientes já cadastrados, o mesmo acontece na linha 7 com vendedor, já na linha 9, é onde pré-determina o tempo dos comerciais. Este código está presente na camada *Controller*.

4.4 Utilização de Saldos

Neste item será demonstrado à implementação de como foi desenvolvido a área de realização das utilizações de saldos do protótipo.

QUADRO 25 - Cadastro de nova utilização de saldo

```
1   public ActionResult Create(Credit credit)
2     {
3       if (ModelState.IsValid)
4         {
5           var contract = db.Contracts.Find(credit.ContractId);
6
7         if (credit.QuantityInsertion > contract.QuantityInsertion)
8         ViewBag.QuantityInsertionError = "Quantidade de  inserções
9   não pode ser maior que o saldo em contrato.";
10        else
11           {
12             var credits = db.Credits.Where(x => x.ContractId ==
13  credit.ContractId);
14
15             var quantityUsed = credits.Any() ? credits.Sum(y =>
16  y.QuantityInsertion) : 0;
17
18             var quantityAvailable = contract.QuantityInsertion -
19  quantityUsed;
20
21            if (credit.QuantityInsertion > quantityAvailable)
22            ViewBag.QuantityInsertionError = "Quantidade de
23  inserção deve ser no maximo: " + quantityAvailable;
24         else
25             {
26               db.Credits.Add(credit);
27               db.SaveChanges();
28               return RedirectToAction("Index");
29                     }
30                }
31              }
32
33         List<SelectListItem> select = new List<SelectListItem>();
34
35     select.Add(new SelectListItem { Text = "7", Value = "7" });
36     select.Add(new SelectListItem { Text = "15", Value = "15" });
37     select.Add(new SelectListItem { Text = "30", Value = "30" });
38     select.Add(new SelectListItem { Text = "60", Value = "60" });
39
40     credit.TimeInsertions = select;
41
42     return View(credit);
43   }
```

No quadro 25, a linha 7 está dizendo que caso a quantidade de inserções para

ser feita a utilização for maior que o total do contrato irá aparecer a mensagem de erro, a linha 12 ocorre caso a quantidade de inserções da utilização não seja maior que o total do contrato, e nela ocorre a busca no banco de todos os créditos de um contrato, enquanto a linha 15 verifica se tem algum credito, caso tenha ele faz a soma das quantidades de inserções, caso contrário, ele coloca 0. A linha 18, mostra que a variável quantidade disponível irá receber o valor de quantidade total do contrato – quantidade utilizada, após isso, na linha 21, diz que se a quantidade de utilização for maior que a quantidade disponível, aparecerá uma mensagem mostrando qual a quantidade disponível e que não poderá ultrapassar essa quantidade.

4.5 Visões do Sistema

Neste item são demostrada as telas do sistema, com um breve relato de quais suas funcionalidades.

4.5.1 Visão da Tela principal

FIGURA 24 - Visão da Tela principal

4.5.2 Visão do Cadastro de clientes

FIGURA 25 - Visão de Cadastro de Clientes

A Figura 22 mostra a tela de cadastro do sistema, onde deverá ser digitado a razão social e nome fantasia do cliente, a cidade deste cliente e também qual foi o vendedor que efetuou esta venda, após serem digitados todos os dados necessários clica-se em salvar para salvar os dados no banco de dados e completar processo de cadastro do cliente.

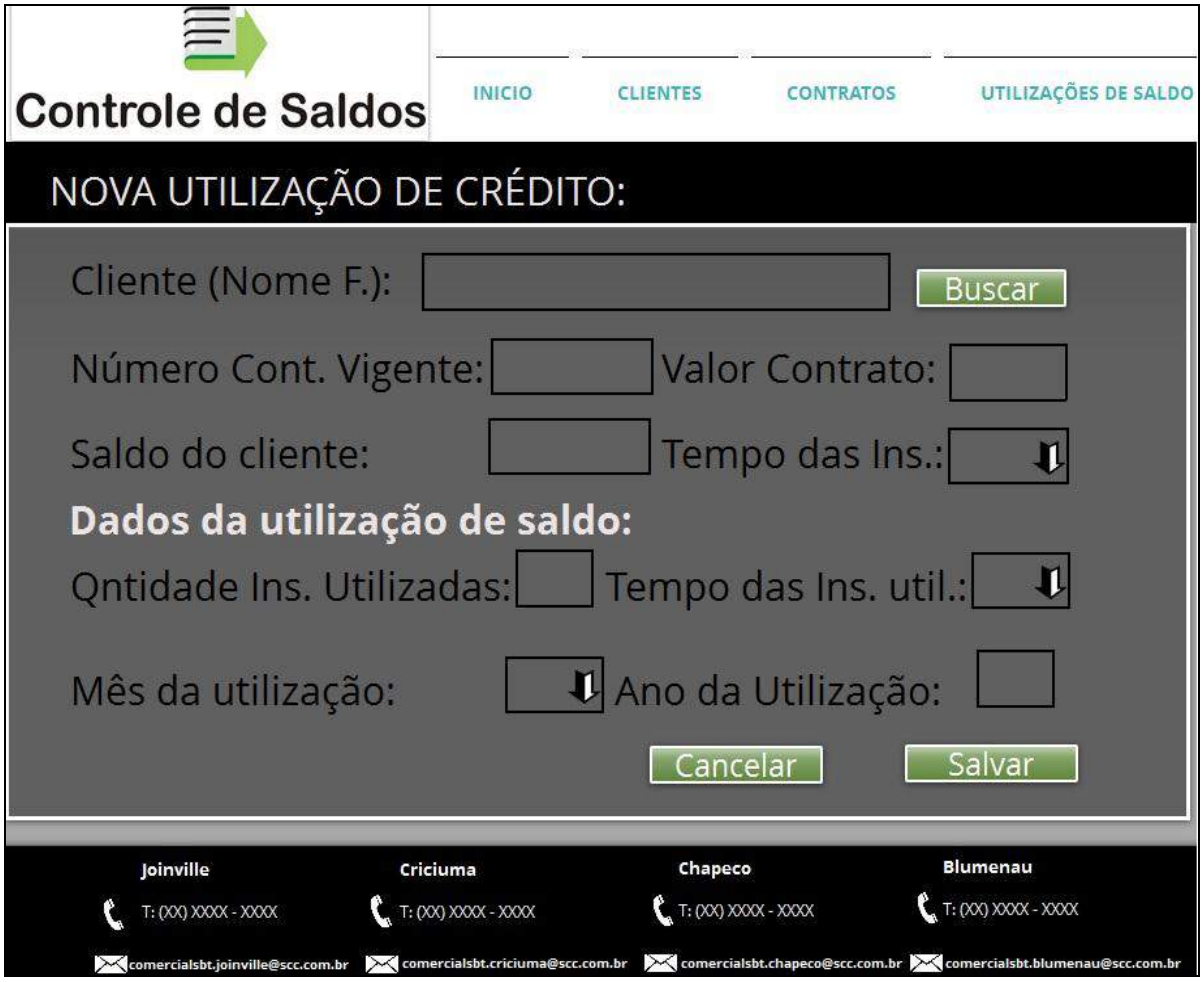
4.5.3 Visão do Cadastro de Contratos

FIGURA 26 - Visão de Cadastro de Contratos

A Figura 23 mostra a tela de cadastro do contrato, onde o usuário deverá digitar o nome do cliente (já cadastrado em sistema), o número e valor do contrato, a quantidade de inserções a que o cliente tem direito, tempo das inserções (pré-definidas) e o nome do vendedor que efetuou esta venda, após serem digitados todos os dados necessários clica-se em salvar para salvar os dados no banco de dados e completar processo de cadastro do contrato.

4.5.4 Visão da tela de Utilizações de Saldos

FIGURA 27 - Visão da tela de Utilizações de Saldos



A Figura 24 mostra a Tela para cadastrar uma nova utilização de crédito, onde o usuário irá digitar o nome do cliente e clicar em buscar, após isso, o sistema irá preencher os dados do cliente referente ao número de contrato vigente, valor do contrato, saldo atual do cliente e o tempo das inserções a que o cliente tem direito.

Sendo necessário que o cliente digite a quantidade de inserções utilizadas conforme planilhas, o tempo destas inserções, o mês e o ano desta utilização.

4.5.5 Visão da tela para alteração de clientes

FIGURA 28 - Tela para alteração de clientes

4.5.6 Relatórios

Devido à necessidade de utilização da ferramenta *Reporter Viewer*, o qual foi instalado, porém, como é uma ferramenta nova para mim, não obtive o resultado esperado, ficando a parte de relatórios para uma próxima versão do sistema.

4.6 Conclusão

Neste capítulo foi apresentado, de forma detalhada, a implementação do protótipo desenvolvido após o estudo de caso sobre padrões de desenvolvimento de software, protótipo este, que utilizou o padrão MVC e as ferramentas Visual Studio 2012 e *SQL Server Management Studio 2012*.

Foi descrito as implementações através de figuras e quadros e ficou claro que a ferramenta *Visual Studio 2012* em conjunto com o padrão MVC pode facilitar o desenvolvimento de aplicações WEB.

No entanto, ressalta-se que esta foi à etapa com maior dificuldade para ser concluída, pois, apresenta escassez de materiais, além do fato que nosso curso pouco foi passado em relação a este padrão.

5 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo realizar um estudo sobre os padrões de desenvolvimento de software através da implementação de um estudo de caso, depois de concluída a etapa de estudo, foi optado por um destes padrões e desenvolvido um protótipo para automatizar um processo na empresa SBT Santa Catarina. Devido às necessidades da empresa, o protótipo desenvolvido foi um sistema Web, portanto o padrão escolhido foi o padrão arquitetural MVC.

Quanto aos objetivos específicos, o primeiro foi alcançado no capítulo 2, onde foi realizado o referencial teórico, todo ele referente aos padrões de desenvolvimento de software, onde foi apresentado, analisado funções e características de cada um dos padrões citados no capitulo em questão.

O segundo objetivo específico foi atingido através dos capítulos 3 e 4, os quais abordam, respectivamente, a modelagem e implementação do sistema.

A modelagem do sistema proposto foi definida no capítulo 3, o qual abordou, dentre outros assuntos, o software utilizado para realizar a modelagem, as historias de usuário, o diagrama de classe bem como o diagrama de atividades.

O capítulo 4, o qual abordou a implementação do sistema, descreve com detalhamento considerável o desenvolvimento do protótipo proposto, este que, utilizou a ferramenta *Visual Studio* 2012, *SQL Server Mannagement Studio* e a tecnologia MVC 4.

Como dificuldades encontradas, ressalta-se às poucas referências em relação aos padrões de projetos, pois, como estes padrões foram descritos por 4 escritores em um único livro, todas as outras referencias estão baseadas neste mesmo livro, assim dificultando boas citações de diferentes autores.

Outra dificuldade encontrada, refere-se quanto ao capítulo 4, correspondente à implementação de relatórios, esta que, não foi desenvolvida devido à necessidade de utilização da ferramenta *Reporter Viewer*, o qual foi instalado, porém, como é uma ferramenta nova para mim, não obtive o resultado esperado, ficando esta parte para uma próxima versão do sistema.

O MVC 4 e o *Visual Studio 2012* trabalham em conjunto, o que facilita o trabalho do desenvolvedor, uma vez que possibilitam utilizar a linguagem .NET.

O presente estudo foi bastante desafiador por se tratar de um assunto pouco visto no curso, e com boa parte das informações sendo publicadas em idiomas estrangeiros e em sites da internet.

Como pesquisas futuras, pretende-se estudar melhor a ferramenta *Reporter Viewer*, a fim de concluir o sistema proposto inicialmente, e aplicar os padrões de desenvolvimento conforme a necessidade do software.

REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, CHRISTOPHER, **A Pattern Language**. Oxford USA Trade, 1977.

ASP.NET. **ASP.NET MVC 4**. Disponível em: <http://www.asp.net/mvc/mvc4>. Acesso em: 15 nov. 2013.

BACKBONEJS.ORG. BackBone.Js. Disponível em: <http://backbonejs.org/>. Acesso em: 15 nov. 2013.

BRIZENO, Marcos. **Visitor.** Disponível em: <http://brizeno.wordpress.com/category/padroes-de-projeto/visitor/>. Acesso em: 15 nov. 2013.

BUSCHMANN et al. **Pattern-Oriented Software Architeture: A System of Patterns.** New Jersey: John Wiley Professio, 1996.

FERNANDES, Jorge H. C.. **Padrões Estruturais.** Disponível em: <http://www.cic.unb.br/~jhcf/MyBooks/iess/Patterns/StructuralPatterns-48slides.pdf>. Acesso em: 15 nov. 2013.

FERREIRA, Alex. **Padrões de Projeto:** O que são e por que utiliza-los?. Disponível em: <http://www.iotecnologia.com.br/padroes-de-projeto-o-que-sao-por-que-usar>. Acesso em: 18 jun. 13.

FOWLER, Martin. **Padrões de Arquitetura de Aplicações Corporativas**. Porto Alegre: Bookman, 2006. 493 p.

FREEMAN & FREEMAN, Eric & Elizabeth. **Padrões de Projetos**: Seu cérebro em padrões de projetos. Rio de Janeiro: ALTABOOKS, 2007.

FURTADO, Maicon. **Padrões de Projeto:** Factory Method. Disponível em: <http://padroesdeprojetodesoftware.blogspot.com.br/2012/05/factory-method.html>. Acesso em: 15 nov. 2013.

GAMMA, Erich et al**.** Padrões de Projeto: **Soluções Reutilizáveis de Software Orientado a Objetos**. Porto Alegre: Bookman, 2000. 363 p.

GERMAN, D. COWAN, D. **Toward a Unified Catalog Of Hypermedia Design Patterns**, 2000.

HARTWIG, Renan. **Padrões de Projeto** - Prototype. Disponível em:

<http://tadspring.blogspot.com.br/2013/06/padroes-de-projeto-prototype.html>. Acesso em: 15 nov.

LABORDE, Gregory. **Design Pattern, o que é e como implantar**. Disponível em: <http://www.oficinadanet.com.br/artigo/desenvolvimento/design-patterns-o-que-e-e-como-implantar>. Acesso em: 14 maio 2013.

LARMAN, Craig. **Utilizando UML e Padrões**: Uma introdução à análise e ao projeto orientado a objetos e ao desenvolvimento iterativo. Porto Alegre: Bookman, 2008. 695 p.

LEITE, Alessandro Ferreira. **Conheça os Padrões de Projeto.** Disponível em: <http://www.devmedia.com.br/conheca-os-padroes-de-projeto/957>. Acesso em: 15 nov. 2013.

MACORATTI, José Carlos. **.NET - O padrão de projeto Observer.** Disponível em: <http://imasters.com.br/desenvolvimento/visual-basic/net-o-padrao-de-projeto-observer/>. Acesso em: 15 nov. 2013c.

MACORATTI, José Carlos. **O Padrão de Projeto Command.** Disponível em: <http://www.macoratti.net/13/03/net_cmd.htm>. Acesso em: 15 nov. 2013b.

MACORATTI, José Carlos. **O Padrão de Projeto Decorator.** Disponível em: <http://www.macoratti.net/13/02/net_decor1.htm>. Acesso em: 15 nov. 2013a.

MARIOTTI, Flavio Secchieri. **Strategy - Padrão de Projeto.** Disponível em: <www.linhadecodigo.com.br/artigo/3268/strategy-padrao-de-projeto-com-microsoft-net-csharp.aspx>. Acesso em: 15 nov. 2013b.

MARIOTTI, Flavio Secchieri. **Proxy:** Padrão de Projet. Disponível em: <http://www.devmedia.com.br/proxy-padrao-de-projeto-com-microsoft-net-c/22183>. Acesso em: 15 nov. 2013a.

MASSONI, Tiago; SAMPAIO, Augusto; BORBA, Paulo. **A RUP-Based Software Process Supporting Progressive Implementation**. Idea Group Publishing, 2003.

MEDEIROS, Higor. **Padrão de Projeto Facade.** Disponível em: <http://www.devmedia.com.br/padrao-de-projeto-facade-em-java/26476>. Acesso em: 07 maio 2013a.

MEDEIROS, Higor. **Padrão de Projeto Singleton.** Disponível em: <http://www.devmedia.com.br/padrao-de-projeto-singleton-em-java/26392>. Acesso em: 07 maio 2013b.

MICROSOFT.COM. **Introdução ao Visual Studio**. Disponível em: <http://msdn.microsoft.com/pt-br/library/fx6bk1f4(v=vs.90).aspx>. Acesso em: 15 nov. 2013a.

MICROSOFT.COM. **Microsoft® SQL Server® 2008 Management Studio Express**. Disponível em: <http://www.microsoft.com/pt-br/download/details.aspx?id=7593>. Acesso em: 15 nov. 2013b.

NASCIMENTO, Rafael. **Padrões de Projeto: Abstract factory.** Disponível em: <http://www.devmedia.com.br/padroes-de-projeto-em-net-abstract-factory/3646>. Acesso em: 05 maio 2013a.

NASCIMENTO, Rafael. **Padrões de Projeto: Builder.** Disponível em: <http://www.devmedia.com.br/padroes-de-projeto-em-net-builder/3960>. Acesso em: 05 maio 2013b.

NEGRÃO, Eduardo. **Padrões de Projetos (Designer patterns) -** O que são? Disponível em: <http://portalengenhariadesoftware.blogspot.com.br/2010/04/padroes-de-projeto-design-patterns-o.html>. Acesso em: 14 maio 2013.

OLEQUES, Jorge. **Padrões GRASP**. Disponível em: <http://joleques.blogspot.com.br/2012/05/padroes-grasp-parte-1.html>. Acesso em: 15 maio 2013.

ORTIZ, Luciano. **Factory Method.** Disponível em: <http://luciano-ti.blogspot.com.br/2010/09/factory-method-gof-padrao-de-criacao.html>. Acesso em: 15 nov. 2013.

OSTERWEIL, Leon, et. al.**Strategic Direction in Software Quality**. ACM Press, 1996.

PFLEEGER, Shari Lawrence. **Engenharia de Software:** Teoria e Prática. 2ªEd. São Paulo: Pearson, 2003.

PIETRO, Cristian. **Padrão Visitor**. Disponível em: <http://cristianpietro.blogspot.com.br/2013/07/padrao-visitor.html>. Acesso em: 15 nov. 2013.

PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: Pearson Education do Brasil, 1995.

REIS, Renata Tibiriçá Dos. **Desenvolvimento Web com uso de Padrões.** 2007. 74 f. Monografia (Bacharelado) - Universidade Federal de Juiz de Fora, Juiz de Fora, 2007.

RIBEIRO, Angélica Aparecida de Almeida. **Padrão de Projeto - Bridge.** Disponível em: <http://www.dpi.ufv.br/projetos/apri/?page_id=681#>. Acesso em: 15 nov. 2013a.

RIBEIRO, Angélica Aparecida de Almeida. **Padrão de Projeto Mediator**. Disponível em: <http://www.dpi.ufv.br/projetos/apri/?page_id=736>. Acesso em: 15 nov. 2013b.

RODRIGUES, Leandro. **Padrões de Projeto:** Template Method.

Http://social.technet.microsoft.com/wiki/contents/articles/7985.padroes-de-projeto-template-method-pt-br.aspx. Disponível em: <http://social.technet.microsoft.com/wiki/contents/articles/7985.padroes-de-projeto-template-method-pt-br.aspx>. Acesso em: 15 nov. 2013.

SANTOS, Carlos dos. **Fundamentos do Entity Framework**. Disponível em: <http://msdn.microsoft.com/pt-br/library/jj128157.aspx>. Acesso em: 15 nov. 2013.

SHAW, Mary; GARLAN, David. **Software Architecture:** Perspectives on an Emerging Discipline. New Jersey: Prentice-hall, 1996. 242 p.

SIERRA, Kathy; BATES, Bert. **Padrões de Projetos.** 2ª Edição Rio de Janeiro: Alta Books, 20009. 478 p. (Use a Cabeça).

SOARES, Fabrízzio Alphonsus A. M. N.. **Padrões de Projeto Estruturais.** Disponível em: <http://www.inf.ufg.br/~fabrizzio/web/java/aula8.pdf>. Acesso em: 15 nov. 2013.

SOMMERVILLE, Ian. **Engenharia de Software.** 9ª Ed. Rio de Janeiro: Pearson Education, 2011.

STEVE John. **Design Patterns:** Java WorkBook, 1st edition, Addison-Wesley Professional Computing Series, 2002.

VANINI, Fernando. **Programação Orientada a Objetos**: Padrões de Projeto. Disponível em: <http://www.ic.unicamp.br/~vanini/mc857/PadroesDeProjeto.pdf>. Acesso em: 15 nov. 2013.

VIDAL, Alexandre. **Padrões de Projeto.** Disponível em: <http://www.deinf.ufma.br/~vidal/Topicos2006_2/aula04.pdf>. Acesso em: 15 nov. 2013.

WYKE, C. **Stylin' with CSS: A Designer's Guide**. 2.ed. Chicago. New Riders Press. 2005.

ZELDMAN, J. **Projetando Web Sites Compatíveis**. 1.ed. São Paulo, SP. Campus. 2003.

APÊNDICES

APÊNDICE A – ARTIGO

Padrões de Desenvolvimento de Software: Um Estudo de Caso

Bruno Luis Kuhnen Ramos

Universidade do Planalto Catarinense (UNIPLAC) – Lages – SC – Brasil

Bruno.lramos@outlook.com

Abstract. *The objective with this paper a study on patterns of software development. To this end, we created a case study to demonstrate the main features and functionality of the patterns discussed, among them are: design patterns of creation, as Singleton and Builder, structural design patterns, such as Bridge and Composite patterns behavioral design, as Decorator and Command Web standards and architectural pattern MVC (Model-View-control), the latter being implemented standard cited with a prototype developed to make the control of contracts and sales of a company. This prototype that used the tools Visual Studio 2012 and SQL Server Mannagement Studio 2008 to develop.*

Resumo. *O objetiva-se com este artigo realizar um estudo sobre padrões de desenvolvimento de software. Para tal, criou-se um estudo de caso para demonstrar as principais características e funcionalidades dos padrões abordados, dentre eles estão: padrões de projeto de criação, como o Singleton e Builder, padrões de projeto estruturais, como o Bridge e o Composite, padrões de projeto comportamentais, como Decorator e Command, padrões Web e o padrão arquitetural MVC (Model, View, Control), sendo implementado este ultimo padrão citado, em um protótipo desenvolvido para realizar o controle de contratos e saldos de uma empresa. Protótipo este que utilizou as ferramentas Visual Studio 2012 e SQL Server Mannagement Studio 2008 para ser desenvolvido.*

1. Introdução

Segundo Massoni et al. (2003), o desenvolvimento de software tem se tornado mais complexo ao longo dos anos. As exigências por parte dos clientes são cada vez maiores em termos de produtividade, qualidade de software e prazos.

O surgimento de novas tecnologias e a necessidade de realização de mudanças nos softwares desenvolvidos para atender às exigências dos clientes também dificulta a tarefa de desenvolver software com qualidade.

Acompanhar as mudanças tecnológicas e atender às necessidades de mudança pode ser uma tarefa bastante complicada se o software não estiver preparado para suportar tais mudanças.

Conforme Osterweil (1996), alguns estudos sugerem que pouco mais da metade do esforço utilizado para o desenvolvimento de software esteja voltado para atividades voltadas para assegurar a qualidade de software.

Dois recursos utilizados como forma de buscar o desenvolvimento de software com qualidade são a utilização de processos de desenvolvimento e a utilização de padrões, além de que a utilização de padrões durante o processo de desenvolvimento pode ser uma das técnicas utilizadas para assegurar qualidade nos documentos gerados.

Alexander (1977) afirma que: “cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma forma”.

Muitos documentos que são criados durante um processo de desenvolvimento de software são comuns a vários sistemas. Algumas vezes as mesmas técnicas são utilizadas para a criação desses documentos e, portanto, os mesmos problemas são encontrados. Os padrões apresentam uma forma de descrever soluções para esses problemas comuns baseado na experiência de outras pessoas.

Diante do exposto, o presente trabalho tem por objetivo a realização de um estudo de caso sobre padrões de desenvolvimento de software, bem como o desenvolvimento de um protótipo utilizando o padrão arquitetural MVC.

2. Conceitos Iniciais

2.1 Padrão de Desenvolvimento de Software

“Um padrão descreve um problema que ocorre inúmeras/par vezes em determinado contexto, e descreve ainda a solução para esse problema, de modo que essa solução possa ser utilizada sistematicamente em distintas situações.” (ALEXANDER 1977).

Inicialmente o estudo de Alexander foi realizado com o intuito de descobrir se havia alguma regra que pudesse especificar quando uma construção é de boa ou de má qualidade.

A partir dos estudos de Christopher Alexander, profissionais de desenvolvimento de software começaram a incorporar princípios designados por ele na criação das primeiras documentações de padrões.

2.2 Padrão de Projeto

Segundo Negrão (2010), padrões de projetos são combinações de classes e algoritmos associados que cumprem com propósitos comuns de projetos. São normalmente soluções consagradas que se baseiam nas estruturas da orientação a objeto em sua melhor forma.

Alexander (1977) afirma que: “cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma forma”.

Alexander estava referindo-se apenas as necessidades dele (padrões de projeto em construções e cidades), porém, as mesmas afirmações estão corretas em relação aos padrões de projeto orientado a objeto.

A partir das citações acima fica claro que padrões de projeto são soluções para problemas que alguém algum dia já teve e resolveu utilizando um modelo que foi documentado e que pode ser adaptado totalmente ou de acordo com a necessidade de cada aplicação.

2.3 Padrão Web

Padrões Web segundo a *W3C (World Wide* Web *Consortium*) – consórcio de empresas de tecnologia que desenvolvem padrões para a criação e a interpretação dos conteúdos para a web - são recomendações, as quais são destinadas a orientar os desenvolvedores para o uso de boas práticas que tornam a web acessível para todos.

Quando se fala de normas para a web, trata-se, na prática, de três componentes independentes: estrutura, apresentação e comportamento, ou ainda de linguagens estruturais (XHTML, XML e HTML que significam *EXtensible HyperText Markup Language*, *EXtensible Markup Language* e *HyperText Markup Language* respectivamente), linguagens de apresentação (CSS, *Cascading Style Sheets*), dentre outras.

Segundo Wyke (2005), a utilização de padrões para *Web* é extremamente vantajosa, pois proporciona um maior controle sobre a página. Quando é dito que uma página é compatível com os padrões, significa que o documento consiste de HTML ou XHTML válido, utiliza CSS para leiaute, é bem estruturado e semanticamente correto.

Esses fatores podem garantir que o site seja acessado por qualquer dispositivo, seja ele móvel, tátil, *desktop* etc.

Assim, segundo Zeldman (2003), sites construídos de acordo com estes padrões, custam menos, funcionam melhor e são acessíveis a mais pessoas e dispositivos.

2.4 Padrão de Arquitetura

Segundo Pfleeger (2003), uma arquitetura de software envolve a descrição de elementos arquiteturais dos quais os sistemas são construídos, interações entre esses elementos, padrões que guiam suas composições e restrições sobre estes padrões.

Um estilo arquitetural define uma família de sistemas em termos de um padrão de organização estrutural (SHAW e GARLAN, 1996).

Ainda segundo Shaw e Garlan (1996), um estilo arquitetural define: um vocabulário de componentes; tipos de conectores; conjunto de restrições que indica como os elementos são combinados.

2.5 MVC (Model, View, Controller)

Padrão que divide uma aplicação interativa em três componentes: Modelo (model), Visão (view) e Controller (controle). (BUSCHMANN et al., 1996).

Em outras palavras o MVC tem como principal objetivo separar dados ou lógicos de negócio (model) da interface (view), e o fluxo da aplicação (controller), a ideia é permitir que uma mensagem da lógica de negócios pudesse ser acessada e visualizada através de várias interfaces. Na arquitetura MVC, a lógica de negócios não sabe quantas e nem quais as interfaces com o usuário está exibindo seu estado, a camada view não se importa de onde está recebendo os dados, mas ela tem que garantir que sua aparência reflita o estado do modelo, ou seja, sempre que os estados de modelo mudam, o modelo notifica as view para que as mesmas atualizem-se.

Segundo Pressman (2005), algumas das características do padrão arquitetural MVC são:

Separação entre os códigos, View e Controller que gerencias as relações entre Model e a View; Separa a lógica de negócios da apresentação; Divide as responsabilidades, ou seja, programadores na programação e web designers na construção visual do software.

2.6 Quadro Comparativo com as funções dos padrões de projeto segundo GOF

No Quadro 1 é mostrado o nome de cada padrão e sua função de forma resumida e objetiva.

Quadro 1 - Nome e função dos padrões de projeto segundo GOF de forma simplificada

Nome do Padrão	Função do Padrão
Factory Method	Definir uma interface para criar um objeto, deixando as subclasses decidirem que classes instanciar.
Abstract Factory	Fornecer uma interface para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas.
Builder	Separar a construção de objetos complexos da sua representação de forma que o mesmo processo de construção possa criar diferentes representações.
Prototype	Especificar os tipos de objetos a serem criados usando uma instância-protótipo e criar novos objetos pela cópia deste protótipo.
Singleton	Permitir criar objetos únicos para os quais há apenas uma instância.
Class e Objetct Adapter	Converter a interface de uma classe em outra interface.
Bridge	Desacoplar uma abstração da sua implementação, de modo que as duas possam variar independentemente.
Composite	Permitir que seja composto objetos em estrutura de árvore para representar hierarquias parte-todo.
Facade	Ocultar toda a complexidade de uma ou mais classes através de uma *Facade* (fachada).
Decorator	Anexar responsabilidades adicionais a um objeto dinamicamente
Command	Encapsular uma solicitação como objeto.
Iterator	Fornecer um forma de acessar sequencialmente os elementos de um objeto agregado sem expor a sua

	representação subjacente.
State	Permitir que um objeto altere seu comportamento quando o seu estado interno mudar.
Proxy	Fornecer um substituto ou representante de outro objeto para controlar o acesso a ele.
Memento	Armazenar o estado interno do objeto originador.
Iterpreter	Dada uma linguagem, criar uma representação para a gramática da linguagem, juntamente com um interpretador.
Strategy	É um conjunto de algoritmos encapsulados e intercambiáveis, sendo que a estratégia permite que o algoritmo se diversifique independentemente dos clientes que os utilizam.
Observer	Representar uma relação 1-N (de um para muitos) entre objetos.
Visitor	Representar uma operação a ser executada nos elementos de uma estrutura de objetos.
Mediator	Definir um objeto que encapsula a forma como um conjunto de objetos interage.
Flyweight	Usa o compartilhamento para suportar eficientemente grande quantidades de objetos de granularidade fina.
Template Method	Facilitar a implementação de cenários onde se deseja encapsular algoritmos.
Chain Of Responsability	Representar um encadeamento de objetos para realizar o processamento de uma série de requisitos diferentes.

FONTE: (BRUNO RAMOS, 2013)

3. Estudo de Caso

O sistema proposto é responsável por controlar os registros de entrada e saída de contratos de uma empresa, além de controlar as alterações e utilizações destes contratos e manter as informações no banco de dados.

O objetivo maior do sistema é aumentar a segurança e a confiabilidade das informações no que diz respeito ao controle de contratos, proporcionando também maior facilidade e rapidez no processo.

O sistema em questão será desenvolvido utilizando o padrão interativo arquitetural MVC, esta escolha se dar por alguns motivos relevantes, na sequência será explanados o porque desta escolha:

A empresa necessita que o sistema seja web, não necessitando sua instalação, facilitando assim sua utilização nos diferentes locais necessário; o MVC por *default* utiliza outros 3 padrões de projeto em suas camadas, o padrão *Composite* na camada *View,* o *Observer* na camada *Model* e o *Strategy* na camada *Controller.*

4. Desenvolvimento do Sistema

Para o desenvolvimento do sistema, foram utilizadas as ferramentas *Visual Studio 2012 e SQL Server Mannagement Studio 2008,* juntamente com as tecnologias MVC 4, *Entity Framework* e *Framework BackBone*.

4.1 *Visual Studio 2012*

Segundo o site da Microsoft.com (2013a), o Visual Studio é um conjunto completo de ferramentas de desenvolvimento para construção de aplicações Web ASP.NET, serviços Web *XML*, aplicações desktop e aplicativos móveis. *Visual Basic*, *Visual C# e Visual C++* usam todos o mesmo ambiente de desenvolvimento integrado (IDE), que permite o compartilhamento de ferramentas e facilita a criação de soluções com mistura de linguagens. Além disso, essas linguagens usam a funcionalidade do *.NET framework*, que fornece acesso às tecnologias chaves que simplificam o desenvolvimento de aplicativos Web em ASP e serviços *Web XML*.

4.2 *SQL Server Mannagement Studio 2008*

Segundo o site Microsoft.com (2013b), o *Microsoft SQL Server Management Studio Express* é um ambiente de desenvolvimento integrado gratuito para acessar, configurar, gerenciar, administrar e desenvolver todos os componentes do *SQL Server*. Combina um grupo de ferramentas gráficas e editores de script sofisticados para fornecer acesso ao *SQL Server* a desenvolvedores e administradores de todos os níveis de conhecimento.

4.3 MVC 4

Segundo o site *ASP.NET* (2013), o MVC 4 é um *framework* para construção de aplicações *WEB* escaláveis, baseadas em padrões, usando padrões de projetos bem estabelecidos.

Como principais características do MVC 4 temos: ASP.NET Web API; Modelos de projeto padrão renovado e modernizado; Novo modelo de projeto móvel; Muitos novos recursos para suportar aplicativos móveis; Suporte aprimorado para métodos assíncronos.

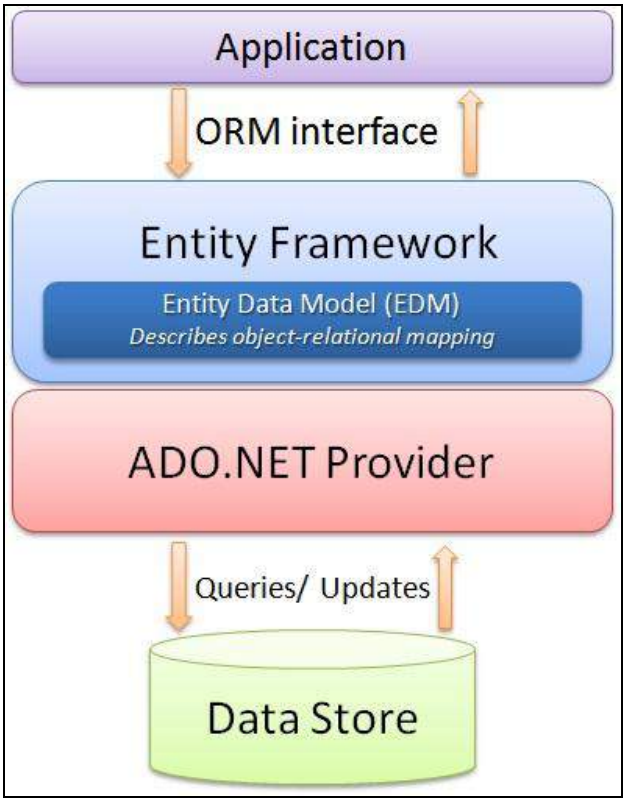
4.4 Entity Framework

Segundo Santos (2013), o *Microsoft Entity Framework* é uma ferramenta de mapeamento objeto relacional *(ORM - Object Relational Management)*, que permite aos desenvolvedores trabalhar com classes que correspondem a tabelas em um banco de dados, tornando transparente o acesso a estes dados e principalmente, eliminando a necessidade de escrever código de banco de dados na aplicação.

A comunicação do *Entity Framework* com o banco de dados é feita através do ADO.NET *Provider*, que funciona como um "*driver*" do banco de dados, normalmente desenvolvido pelo próprio fabricante do banco. Desta forma todos os comandos submetidos pelo *Entity Framework* são "traduzidos" para a linguagem do banco de dados através do seu *provider*, gerando os comandos SQL mais adequados a cada operação e principalmente, comandos que tenham o máximo de desempenho.

Na sequencia será mostrado um diagrama do funcionamento do *Entity Framework*:

Figura 1. Diagrama do funcionamento do Entity Framework



FONTE: (SANTOS, 2013)

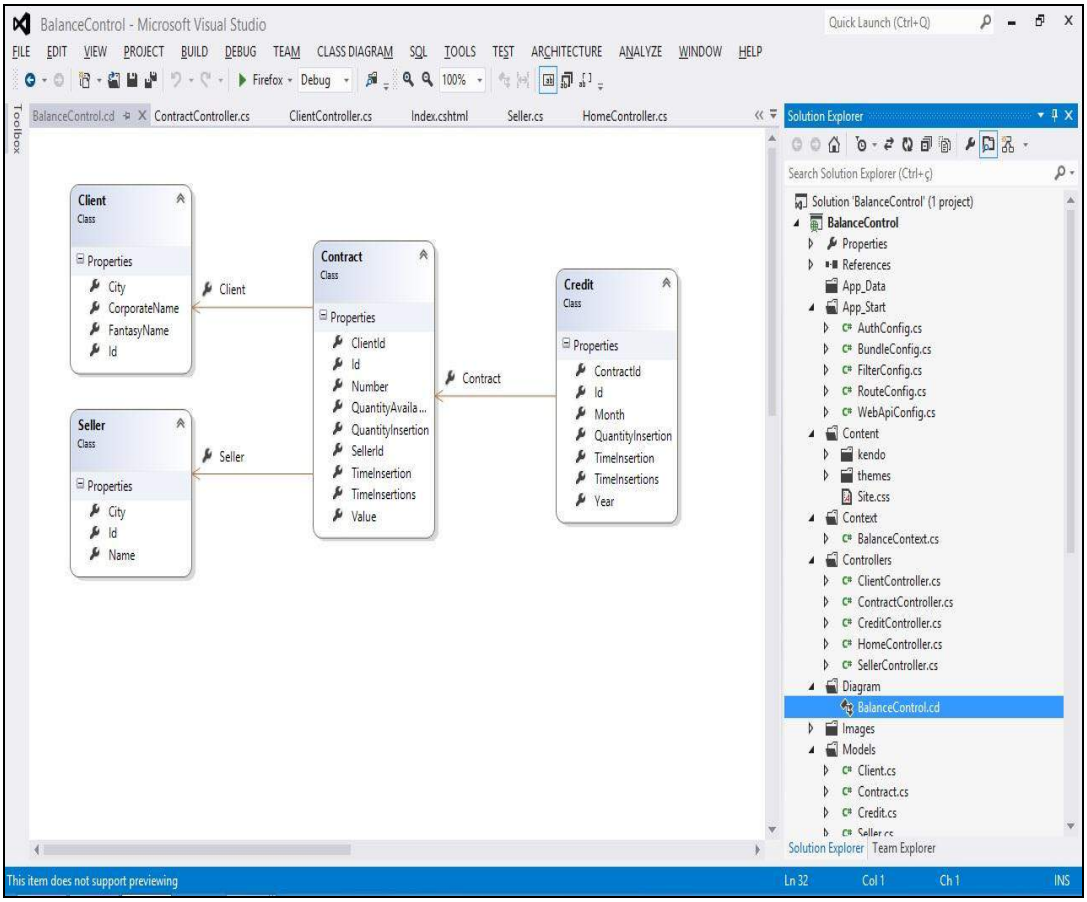
4.5 Framework BackBone

Segundo o site *backbonejs.org*, o *backbones.js* dá estrutura para aplicações web, oferecendo modelos com chave-valor, eventos de ligação e personalização, coleções com uma rica *API (Application Programming Interface)* de funções enumeráveis, visões com manipulações de eventos declarativo, e conecta tudo à sua *API* existente através de uma interface *RESTful JSON***.**

5. Implementação do Sistema

A Figura 2 demonstra o diagrama de classe da base de dados que sera utilizado no desenvolvimento do protótipo proposto.

Figura 2. Diagrama de Classes do protótipo



5.1 Cadastros

Neste item é demostrada a implementação de todos os cadastros do protótipo, especificando detalhadamente cada particularidade da implementação utilizando o padrão MVC.

Quadro 2 - Cadastro de Vendedores

```
1   public ActionResult Create(Seller seller)
2      {
3         if (ModelState.IsValid)
4            {
5               db.Sellers.Add(seller);
6               db.SaveChanges();
7               return RedirectToAction("Index");
8            }
9         return View(seller);
10     }
```

No quadro 2, na linha 3 está sendo mostrado que se os dados digitados forem validos, o processo de criação do novo vendedor poderá seguir, caso contrário enviará uma mensagem de erro. Na linha 5 está sendo adicionado um novo cadastro de vendedor, enquanto na linha 6

os dados deste novo cadastro está sendo gravado no banco, após isso, a página redireciona para a página anterior.

Quadro 3 - Edição de Vendedor

```
1   Public ActionResult Edit(Seller seller)
2      {
3         if (ModelState.IsValid)
4            {
5               db.Entry(seller).State = EntityState.Modified;
6               db.SaveChanges();
7               return RedirectToAction("Index");
8            }
9         return View(seller);
10     }
```

Assim como no quadro 2, na linha 3, está sendo mostrado que se os dados digitados forem válidos, o processo de edição poderá prosseguir. Na linha 5, será procurado no banco de dados o vendedor a que se deseja alterar, e assim ocorre a alteração, sendo que na linha 6 é salvo os novos dados do vendedor, sobreescrevendo os dados antigos.

Quadro 4 - Exclusão de Vendedor

```
1     public ActionResult DeleteConfirmed(long id)
2        {
3           Seller seller = db.Sellers.Find(id);
4           db.Sellers.Remove(seller);
5           db.SaveChanges();
6           return RedirectToAction("Index");
7        }
```

No quadro 4, na linha 3, é feita a busca no banco do vendedor selecionado para exclusão, enquanto na linha 4 o vendedor é excluído do banco e na linha 5 é salvo as alterações referente a esta exclusão.

Quadro 5 - Cadastro de Clientes

```
1   public ActionResult Create()
2   {
3      ViewBag.SellerId = new SelectList(db.Sellers, "Id", "City");
4      return View();
5   }
6
7   [HttpPost]
8   public ActionResult Create(Client client)
9   {
10     if (ModelState.IsValid)
11        {
12          db.Clients.Add(client);
13          db.SaveChanges();
14          return RedirectToAction("Index");
15        }
16
17          return View(client);
18  }
```

O código do quadro 5, referente ao cadastro de clients se assemelha muito com o cadastro do quadro 1, tendo apenas como particularidade que para cadastrar um cliente, é necessário ter o vendedor que representa este cliente já cadastrado em sistema, o que está acontecendo na linha 3, em que é mostrada uma lista com todos os vendedores cadastrados no banco de dados.

Quadro 6 - Exibição dos Contratos Cadastrados

```
1   public ActionResult Index()
2     {
3      var contracts = db.Contracts.Include(c =>
4   c.Client).Include(c => c.Seller);
5
6     List<Contract> resultList = new List<Contract>();
7
8     foreach (var contract in contracts)
9       {
10       var credits = db.Credits.Where(y => y.ContractId ==
11  contract.Id);
12
13       var quantity = credits.Any() ? credits.Sum(y =>
14  y.QuantityInsertion) : 0;
15
16       contract.QuantityAvailable = contract.QuantityInsertion -
17  quantity;
18
19       resultList.Add(contract);
20      }
21      return View(resultList.ToList());
22    }
```

O quadro 6 mostra como foi desenvolvida a área de exibição dos contratos cadastrados, as linhas 3 e 4 buscam todos os contratos cadastrados através dos dados do cliente, enquanto na linha 6, é criado uma lista com todos os contratos para serem exibidos na tela. Na linha 8 inicia uma foreach para verificar se o contrato ainda tem saldo disponível, a linha 10 busca todos créditos de um contrato, a linha 13 e 14 verifica se tem algum credito, caso tenha ele faz a soma das quantidades de inserções, caso contrario ele coloca 0.

Quadro 7 - Cadastro de Contratos

```
1   public ActionResult Create()
2     {
3        Contract contract = new Contract();
4
5      ViewBag.ClientId = new SelectList(db.Clients, "Id",
6   "CorporateName");
7      ViewBag.SellerId = new SelectList(db.Sellers, "Id", "Name");
8     List<SelectListItem> select = new List<SelectListItem>();
9     select.Add(new SelectListItem { Text = "7", Value = "7" });
10    select.Add(new SelectListItem { Text = "15", Value = "15" });
11    select.Add(new SelectListItem { Text = "30", Value = "30" });
12    select.Add(new SelectListItem { Text = "60", Value = "60" });
13     contract.TimeInsertions = select;
14
15     return View(contract);
16   }
```

O quadro 7 mostra o cadastro de novos contratos, a linha 5 faz uma seleção, podendo ser adiciona apenas clientes já cadastrados, o mesmo acontece na linha 7 com vendedor, já na linha 9, é onde pré-determina o tempo dos comerciais.

Quadro 8 - Cadastro de utilizações de saldo

```
1  public ActionResult Create(Credit credit)
2    {
3      if (ModelState.IsValid)
4        {
5          var contract = db.Contracts.Find(credit.ContractId);
6
7        if (credit.QuantityInsertion > contract.QuantityInsertion)
8        ViewBag.QuantityInsertionError = "Quantidade de  inserções
9  não pode ser maior que o saldo em contrato.";
10       else
11          {
12            var credits = db.Credits.Where(x => x.ContractId ==
13 credit.ContractId);
14
15            var quantityUsed = credits.Any() ? credits.Sum(y =>
16 y.QuantityInsertion) : 0;
17
18            var quantityAvailable = contract.QuantityInsertion -
19 quantityUsed;
20
21           if (credit.QuantityInsertion > quantityAvailable)
22           ViewBag.QuantityInsertionError = "Quantidade de
23 inserção deve ser no maximo: " + quantityAvailable;
24        else
25            {
26              db.Credits.Add(credit);
27              db.SaveChanges();
28              return RedirectToAction("Index");
29                     }
30                }
31             }
32
33        List<SelectListItem> select = new List<SelectListItem>();
34
35    select.Add(new SelectListItem { Text = "7", Value = "7" });
36    select.Add(new SelectListItem { Text = "15", Value = "15" });
37    select.Add(new SelectListItem { Text = "30", Value = "30" });
38    select.Add(new SelectListItem { Text = "60", Value = "60" });
39
40    credit.TimeInsertions = select;
41
42    return View(credit);
43   }
```

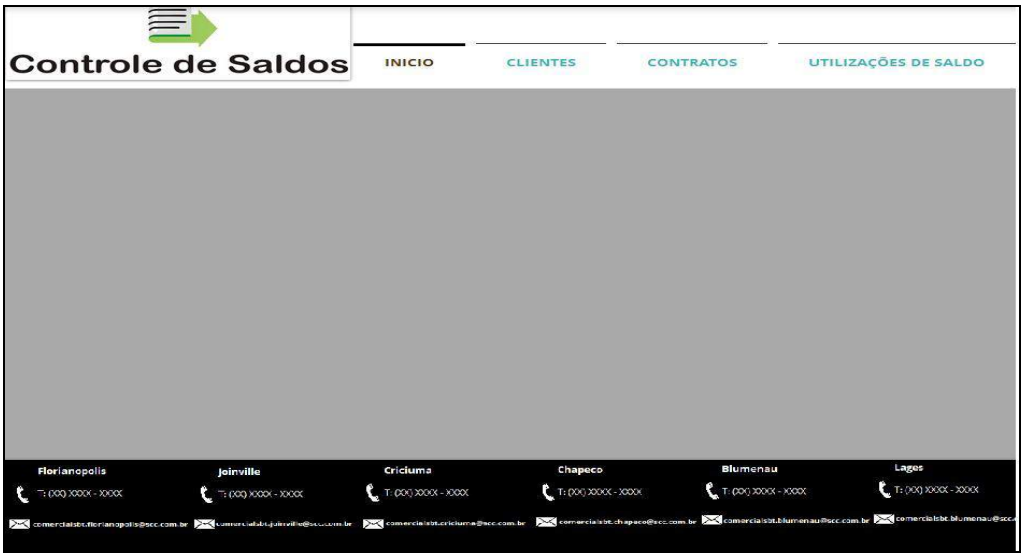
No quadro 8, a linha 7 está dizendo que caso a quantidade de inserções para ser feita a utilização for maior que o total do contrato irá aparecer a mensagem de erro, a linha 12 ocorre caso a quantidade de inserções da utilização não seja maior que o total do contrato, e nela ocorre a busca no banco de todos os créditos de um contrato, enquanto a linha 15 verifica se

tem algum credito, caso tenha ele faz a soma das quantidades de inserções, caso contrário, ele coloca 0. A linha 18, mostra que a variável quantidade disponível irá receber o valor de quantidade total do contrato – quantidade utilizada, após isso, na linha 21, diz que se a quantidade de utilização for maior que a quantidade disponível, aparecerá uma mensagem mostrando qual a quantidade disponível e que não poderá ultrapassar essa quantidade.

5.2 Visões do Sistema

Na Figura 3 é mostrada a tela principal do sistema o menu principal que fica localizado na área superior da página e possibilita ao usuário interagir com as principais páginas do site.

Figura 3. Tela principal do Sistema



A Figura 4 mostra a tela de cadastro de clientes, onde deverá ser digitado a razão social e nome fantasia do cliente, a cidade deste cliente e também qual foi o vendedor que efetuou esta venda, após serem digitados todos os dados necessários clica-se em salvar para salvar os dados no banco de dados e completar processo de cadastro do cliente.

Figura 4. Tela de Cadastro de clientes

A Figura 5 mostra a tela de cadastro de contratos, onde o usuário deverá digitar o nome do cliente (já cadastrado em sistema), o número e valor do contrato, a quantidade de inserções a que o cliente tem direito, tempo das inserções (pré-definidas) e o nome do vendedor que efetuou esta venda, após serem digitados todos os dados necessários clica-se em salvar para salvar os dados no banco de dados e completar processo de cadastro do contrato.

Figura 5. Tela de Cadastro de Contratos

Na Figura 6 é mostrada a Tela para cadastrar uma nova utilização de crédito, onde o usuário irá digitar o nome do cliente e clicar em buscar, após isso, o sistema irá preencher os dados do cliente referente ao número de contrato vigente, valor do contrato, saldo atual do cliente e o tempo das inserções a que o cliente tem direito. È necessário que o cliente digite a quantidade de inserções utilizadas conforme planilhas, o tempo destas inserções, o mês e o ano desta utilização.

Figura 6. Tela para cadastro de Utilização de Saldo

6. Considerações Finais

Este trabalho teve como objetivo realizar um estudo sobre os padrões de desenvolvimento de software através da implementação de um estudo de caso. Após concluída a etapa de estudo, foi optado por um destes padrões e desenvolvido um protótipo para automatizar um processo na empresa SBT Santa Catarina. Devido às necessidades da empresa, o protótipo desenvolvido foi um sistema Web, portanto o padrão escolhido foi o padrão arquitetural MVC.

A implementação do sistema, descreve com detalhamento considerável o desenvolvimento do protótipo proposto, este que, utilizou a ferramenta *Visual Studio* 2012, *SQL Server Mannagement Studio* 2012, e as tecnologias MVC 4, *Entity Framework* e *Framework BackBone*. O MVC 4 e o *Visual Studio 2012* trabalham em conjunto, o que facilita o trabalho do desenvolvedor, uma vez que possibilitam utilizar a linguagem .NET.

Como dificuldades encontradas, ressalta-se às poucas referências em relação aos padrões de projetos, pois, como estes padrões foram descritos por 4 escritores em um único livro, todas as outras referencias estão baseadas neste mesmo livro, assim dificultando boas citações de diferentes autores.

Como pesquisas futuras, pretende-se estudar melhor a ferramenta *Reporter Viewer*, a fim de concluir o sistema proposto inicialmente, e aplicar os padrões de desenvolvimento conforme a necessidade do software.

7. Referências

ALEXANDER, CHRISTOPHER, **A Pattern Language**. Oxford USA Trade, 1977.

ASP.NET. **ASP.NET MVC 4**. Disponível em: <http://www.asp.net/mvc/mvc4>. Acesso em: 15 nov. 2013.

BACKBONEJS.ORG. **BackBone.Js.** Disponível em: <http://backbonejs.org/>. Acesso em: 15 nov. 2013.

BUSCHMANN et al. **Pattern-Oriented Software Architeture: A System of Patterns.** New Jersey: John Wiley Professio, 1996.

MASSONI, Tiago; SAMPAIO, Augusto; BORBA, Paulo. **A RUP-Based Software Process Supporting Progressive Implementation**. Idea Group Publishing, 2003.

MICROSOFT.COM. **Introdução ao Visual Studio.** Disponível em: <http://msdn.microsoft.com/pt-br/library/fx6bk1f4(v=vs.90).aspx>. Acesso em: 15 nov. 2013a.

MICROSOFT.COM. **Microsoft® SQL Server® 2008 Management Studio Express.** Disponível em: <http://www.microsoft.com/pt-br/download/details.aspx?id=7593>. Acesso em: 15 nov. 2013b.

NEGRÃO, Eduardo. **Padrões de Projetos (Designer patterns) -** O que são? Disponível em: <http://portalengenhariadesoftware.blogspot.com.br/2010/04/padroes-de-projeto-design-patterns-o.html>. Acesso em: 14 maio 2013.

OSTERWEIL, Leon, et. al.**Strategic Direction in Software Quality**. ACM Press, 1996.

PFLEEGER, Shari Lawrence. **Engenharia de Software:** Teoria e Prática. 2ªEd. São Paulo: Pearson, 2003.

PRESSMAN, Roger S. **Engenharia de Software**. São Paulo: Pearson Education do Brasil, 1995.

SANTOS, Carlos dos. **Fundamentos do Entity Framework**. Disponível em: <http://msdn.microsoft.com/pt-br/library/jj128157.aspx>. Acesso em: 15 nov. 2013.

SHAW, Mary; GARLAN, David. **Software Architecture:** Perspectives on an Emerging Discipline. New Jersey: Prentice-hall, 1996. 242 p.

WYKE, C. **Stylin' with CSS: A Designer's Guide**. 2.ed. Chicago. New Riders Press. 2005.

ZELDMAN, J. **Projetando Web Sites Compatíveis**. 1.ed. São Paulo, SP. Campus. 2003.