UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

PROPOSTA DE ARQUITETURA EM TECNOLOGIAS MICROSOFT

ANDRÉ LUIS VOLTOLINI SOUSA

BLUMENAU
2009

2009/1-01

ANDRÉ LUIS VOLTOLINI SOUSA

PROPOSTA DE ARQUITETURA EM TECNOLOGIAS MICROSOFT

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Everaldo Artur Grahl, Mestre – Orientador

BLUMENAU
2009

2009/1-01

PROPOSTA DE ARQUITETURA EM TECNOLOGIAS MICROSOFT

Por

ANDRÉ LUIS VOLTOLINI SOUSA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Everaldo Artur Grahl, Mestre - Orientador, FURB
Membro:	Prof. Fabiane Barreto Vavassori Benitti, Doutora – FURB
Membro:	Prof. Paulo Fernando da Silva, Mestre – FURB

Blumenau, 08 de Julho de 2009

Dedico este trabalho a todas as pessoas que contribuíram para a minha recuperação e me ajudaram a nunca desistir.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, pela compreensão e carinho.

Aos meus amigos, pelos bons momentos vividos.

Ao meu orientador, Everaldo Artur Grahl, por ter acreditado na conclusão deste trabalho.

Em especial ao professor José Roque da Silva por sua compreensão e paciência.

À todos os demais professores que contribuíram para a minha formação.

RESUMO

Este trabalho apresenta o desenvolvimento de uma arquitetura e um aplicativo desenvolvido com tecnologias Microsoft. Na especificação do aplicativo foi utilizado um processo ágil de modelagem, na arquitetura foram aplicados padrões de projetos focados na produtividade, na alta coesão e no baixo acoplamento de todos os itens componentes. Na implementação foram utilizadas convenções, boas práticas e modelos de documentação. De forma didática é apresentada os aspectos técnicos da arquitetura e fatores motivadores que resultaram na utilização de cada tecnologia. Como resultado obteve uma arquitetura desenvolvida seguindo as melhores práticas descritas pela Microsoft.

Palavras-chave: Processo ágil. Arquitetura. Microsoft.

ABSTRACT

This paper presents the development of an architecture and an application developed with Microsoft technologies. In specifying the application process was used agile modeling in architecture standards were applied to projects focused on productivity, high cohesion and low coupling components of all items. Were used in the implementation agreements, and models of best practice documentation. Is presented in a teaching point of view the technical aspects of architecture and motivating factors that led to the use of each technology. As a result achieved an architecture developed by following best practices outlined by Microsoft.

Key-words: Agile process. Architecture. Microsoft.

LISTA DE ILUSTRAÇÕES

Figura 1 – Tarefa de análise de requisitos 16
Figura 2 – Tarefa análise e projeto preliminar 17
Figura 3 – Tarefa projeto 18
Figura 4 – Tarefa implementação 19
Figura 5 – Sintaxe C# 21
Figura 6 – Arquitetura EDRA 23
Figura 7 – LINQ 25
Figura 8 – Microsoft Ado.Net Entity Framework 26
Figura 9 – *Security Token Service* 30
Figura 10 – Diagrama de objetos de domínio 33
Figura 11 – Diagrama de caso de uso do aplicativo 34
Quadro 1 – Descrição do caso de uso incluir produto a compra 35
Quadro 2 – Descrição do caso de uso finalizar compra 36
Quadro 3 – Descrição do caso de uso cadastrar cliente 36
Quadro 4 – Descrição do caso de uso consultar produtos já comprados 37
Quadro 5 – Descrição do caso de uso consultar produtos 37
Quadro 6 – Descrição do caso de uso manter consultas 38
Figura 12 – Diagrama de robustez do caso de uso incluir produto a compra 39
Figura 13 – Diagrama de robustez do caso de uso finalizar compra 39
Figura 14 – Diagrama de robustez do caso de uso cadastrar cliente 40
Figura 15 – Diagrama de robustez do caso de uso consultar produtos já comprados 40
Figura 16 – Diagrama de robustez do caso de uso consultar produto 41
Figura 17 – Diagrama de robustez do caso de uso manter consultas 41
Figura 18 – Diagrama de classes do domínio 42
Figura 19 – Diagrama de seqüência do caso de uso adicionar produto a compra 43
Figura 20 – Diagrama de seqüência do caso de uso finalizar compra 44
Figura 21 – Diagrama de seqüência do caso de uso cadastrar cliente 44
Figura 22 – Diagrama de seqüência do caso de uso consultar produtos já comprados 45
Figura 23 – Diagrama de seqüência do caso de uso consultar produto 46
Figura 24 – Diagrama de seqüência manter consultas 47
Figura 25 – Modelo entidade relacionamento 48

Figura 26 – Diagrama de componentes da arquitetura do aplicativo 49
Figura 27 – Diagrama de componentes da arquitetura dos serviços 50
Figura 28 – Diagrama de seqüência da arquitetura do aplicativo..... 50
Figura 29 – Modelo conceitual..... 52
Quadro 7 – LINQ to Entities 52
Quadro 8 – Obtenção de múltiplos objetos 53
Quadro 9 – Persistência de um objeto de domínio 54
Quadro 10 – Inclusão de um objeto de domínio..... 54
Quadro 11 – Exclusão de um objeto de domínio 55
Quadro 12 – Interface 55
Quadro 13 – O padrão *request* e *response*..... 56
Quadro 14 – BusinessEntityService 56
Quadro 15 – Global.asax 57
Quadro 16 – *Master Page*..... 57
Quadro 17 – Página Asp.Net com componente Silverlight..... 58
Quadro 18 – Xaml 58
Quadro 19 – Código fonte componente Silverlight..... 59
Quadro 20 – Número de versão único para todos os *assemblies*..... 59
Figura 30 – Página inicial..... 60
Figura 31 – Página da categoria de produtos televisores..... 61
Figura 32 – Página do produto 62
Figura 33 – Página de resumo da compra..... 62
Figura 34 – Página de cadastro e *login*..... 63
Figura 35 – Página minhas compras..... 64
Figura 36 – *Back office*..... 64

LISTA DE SIGLAS

CLR – *Common Language Runtime*

CORBA – *Common Object Request Broker Architecture*

DCOM – *Distributed Component Object Model*

DMZ – *Demilitarized zone*

EDRA – *Enterprise Development Reference Application*

HTML – *Hyperlink Markup Language*

HTTP – *Hypertext Transfer Protocol*

JSON – *Javascript Object Notation*

LINQ – *Language Integrated Query*

OO – Orientação a objeto

RIA – Aplicativo rico para internet ou *Rich internet application*

RUP – *Rational Unified Process*

SAML – *Security Assertion Markup Language*

SOA – Arquitetura orientada a serviços ou *Service Oriented Architecture*

SOAP – *Simple Object Access Protocol*

SQL – *Structured Query Language*

SSL – *Secure Sockets Layer*

SSO – *Single Sign On*

SLA – *Service Level Agreement*

STS – *Security Token Service*

UML – *Unified Modeling Language*

XML – *Extensible Markup Language*

XP – *Extreme Programming*

WCF – *Windows Communication Foundation*

WEB – *World Wide Web*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS DO TRABALHO 14
1.2 ESTRUTURA DO TRABALHO 14
2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 PROCESSO ICONIX 15
2.1.1 ANÁLISE DE REQUISITOS 15
2.1.2 ANÁLISE E PROJETO PRELIMINAR 16
2.1.3 PROJETO 17
2.1.4 IMPLEMENTAÇÃO 18
2.2 ARQUITETURA ORIENTADA A SERVIÇOS 19
2.3 TECNOLOGIAS MICROSOFT 21
2.3.1 C# 21
2.3.2 Asp.Net 21
2.3.3 Silverlight 22
2.3.4 Windows Communication Foundation 23
2.3.5 Enterprise Development Reference Architecture (EDRA) 23
2.3.6 Enterprise library 24
2.3.7 LINQ 24
2.3.8 Entity Framework 26
2.4 PADRÕES DE PROJETO 26
2.5 SECURITY TOKEN SERVICE 29
2.6 TRABALHOS CORRELATOS 30
2.6.1 CSLA.Net 31
2.6.2 Arquitetura de referência para sistemas da informação e portais de serviços de governo eletrônico 31
3 DESENVOLVIMENTO DO APLICATIVO 32
3.1 ANÁLISE DE REQUISITOS 32
3.1.1 Requisitos funcionais 32
3.1.2 Requisitos não funcionais 33
3.1.3 Modelo de domínio 33
3.1.4 Prototipagem 34

3.1.5 Diagrama de caso de uso 34
3.2 ANÁLISE E DESENHO PRELIMINAR 35
3.2.1 Caso de uso incluir produto a compra 35
3.2.2 Caso de uso finalizar compra 36
3.2.3 Caso de uso cadastrar cliente 36
3.2.4 Caso de uso consultar produtos já comprados 37
3.2.5 Caso de uso consultar produtos 37
3.2.6 Caso de uso manter consultas 38
3.2.7 Diagrama de robustez do caso de uso incluir produto a compra 38
3.2.8 Diagrama de robustez do caso de uso finalizar compra 39
3.2.9 Diagrama de robustez do caso de uso cadastrar cliente 39
3.2.10 Diagrama de robustez do caso de uso consultar produto já comprado 40
3.2.11 Diagrama de robustez do caso de uso consultar produtos 40
3.2.12 Diagrama de robustez do caso de uso manter consultas 41
3.2.13 Diagrama de classes 42
3.3 DESENHO 42
3.3.1 Diagrama de seqüência do caso de uso incluir produto a compra 43
3.3.2 Diagrama de seqüência do caso de uso finalizar compra 43
3.3.3 Diagrama de seqüência do caso de uso cadastrar cliente 44
3.3.4 Diagrama de seqüência do caso de uso consultar produtos já cadastrados 45
3.3.5 Diagrama de seqüência do caso de uso consultar produtos 45
3.3.6 Diagrama de seqüência do caso de uso manter consultas 46
3.3.7 Modelo entidade relacionamento 47
3.4 ESPECIFICAÇÃO DA ARQUITETURA 48
3.5 IMPLEMENTAÇÃO 51
3.5.1 Técnicas e ferramentas utilizadas 51
3.5.2 Processo de implementação 60
3.6 RESULTADOS E DISCUSSÃO 65
4 CONCLUSÕES 66
4.1 EXTENSÕES 66
4.2 LIMITAÇÕES 67
REFERÊNCIAS BIBLIOGRÁFICAS 68

1 INTRODUÇÃO

Existe hoje uma infinidade de arquiteturas, tecnologias e modelos de desenvolvimento nas mais diversas plataformas, devido a crescente demanda por aplicativos mais focados no negócio e não na tecnologia de implementação, tornando a tomada de decisão para arquitetos e desenvolvedores cada vez mais complexa.

Com o advento da plataforma .Net, em meados de 2001, a Microsoft posicionou-se na vanguarda do desenvolvimento de aplicativos sólidos e coesos, disponibilizando diversas soluções tecnológicas, visando sempre atender as necessidades do mercado de forma clara e objetiva.

Desde lá muitas definições e conceitos surgiram aumentando a complexidade na escolha de tecnologias para o desenvolvimento de aplicativos. Para auxiliar este processo de escolha a Microsoft disponibiliza informações sobre as melhores práticas na utilização das tecnologias e na definição de uma arquitetura compatível com os requisitos do aplicativo.

Não apenas a documentação explicativa sobre uma classe, método ou biblioteca, mas também um direcionamento para a melhor aplicabilidade de cada tecnologia, possibilitando ao arquiteto e desenvolvedor algum subsídio para um início da definição de uma arquitetura confiável e segura. Com isso o grau de abstração no desenvolvimento do aplicativo torna-se maior, possibilitando o foco no negócio e não nas tecnologias empregadas.

Magela (2007a, p. 99) afirma que a modelagem permitiu olhar o problema de cima. Em termos abstratos, ficou claro que vários problemas aparentemente diferentes possuíam uma forma específica de solução. Definitivamente, reutilizar modelos de solução passou a ser mais vantajoso do que reutilizar código.

Visando auxiliar arquitetos e desenvolvedores, surge a intenção do desenvolvimento de uma arquitetura com necessidades específicas de escalabilidade, segurança, baixo acoplamento, alta coesão, orientada a objetos e orientada a serviços.

Este trabalho resultou no desenvolvimento de uma arquitetura utilizando somente tecnologias Microsoft e no desenvolvimento de um aplicativo, este sendo uma prova de conceito da própria arquitetura demonstrando sua viabilidade. Este trabalho foi desenvolvido de forma didática para facilitar a compreensão das técnicas e tecnologias empregadas.

O aplicativo desenvolvido trata da área comercial, especificamente na comercialização de produtos pela Internet, para usufruir de todos os aspectos da arquitetura.

A modelagem do negócio utilizou o processo Iconix, que foca nos objetos de domínio

como principal característica. Na arquitetura do aplicativo foram utilizados conceitos da arquitetura *Enterprise development reference application* (EDRA) e da arquitetura orientada a serviços (*Service oriented architecture* - SOA). A arquitetura e o aplicativo foram desenvolvidos na plataforma Microsoft Framework 3.5 e descrito na linguagem C#.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é a implementação de um aplicativo de referência com foco em comércio eletrônico utilizando uma arquitetura em várias camadas, orientada a serviços, tecnologias Microsoft, padrões de projeto e o processo de modelagem Iconix.

Os objetivos específicos do trabalho são:

a) modelar o aplicativo de referência no processo Iconix;
b) modelar a arquitetura em várias camadas e orientada a serviços;
c) implementar o aplicativo de referência;
d) implementar a camada de apresentação como um aplicativo Rico para Internet.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está dividido em quatro capítulos, sendo que o primeiro compreende a introdução ao trabalho, com sua contextualização, seus objetivos e sua organização.

O segundo capítulo apresenta os conceitos, técnicas e ferramentas mais relevantes ao propósito do trabalho, bem como a descrição de trabalhos correlatos.

O terceiro capítulo engloba a metodologia de desenvolvimento do trabalho, apresentando os requisitos principais do problema, a especificação, a implementação e a discussão do resultado obtido.

No quarto capítulo são apresentadas as conclusões obtidas com o desenvolvimento do trabalho, bem como suas vantagens, limitações e as sugestões de extensão para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os principais conceitos a respeito dos seguintes conceitos e tecnologias: processo de modelagem Iconix; arquitetura orientada a serviços; tecnologias Microsof, padrões de projeto e boas práticas. Também são relatados os trabalhos correlatos.

2.1 PROCESSO ICONIX

De acordo com Scott (2006), o Iconix é um processo de desenvolvimento de software ágil, mais prático do que o modelo RUP e mais centrado do que o processo XP. O processo faz uso da UML como linguagem de modelagem e baseia-se fortemente no modelo de domínio, modelo de casos de uso e modelo de classes. Sugere o refinamento sucessivo dos modelos no decorrer da modelagem. Segundo Magela (2007b, p. 34), teoricamente é possível produzir um software perfeito, ou seja, sem erros e que atenda aos requisitos do usuário. Isto seria tangível se fosse possível estagnar o problema e aplicar métodos formais na construção de programas. Sendo essa realidade impraticável para tal a modelagem iterativa permite aumentar gradativamente o nível de formalidade em relação à linguagem natural.

Segundo Vidotti (2003), a metodologia consiste na produção de artefatos que retratam as visões dinâmica e estática de um sistema, e que vão sendo desenvolvidos incrementalmente e em paralelo. O processo é divido em tarefas principais, sendo: análise de requisitos; análise e projeto preliminar; projeto e implementação.

2.1.1 ANÁLISE DE REQUISITOS

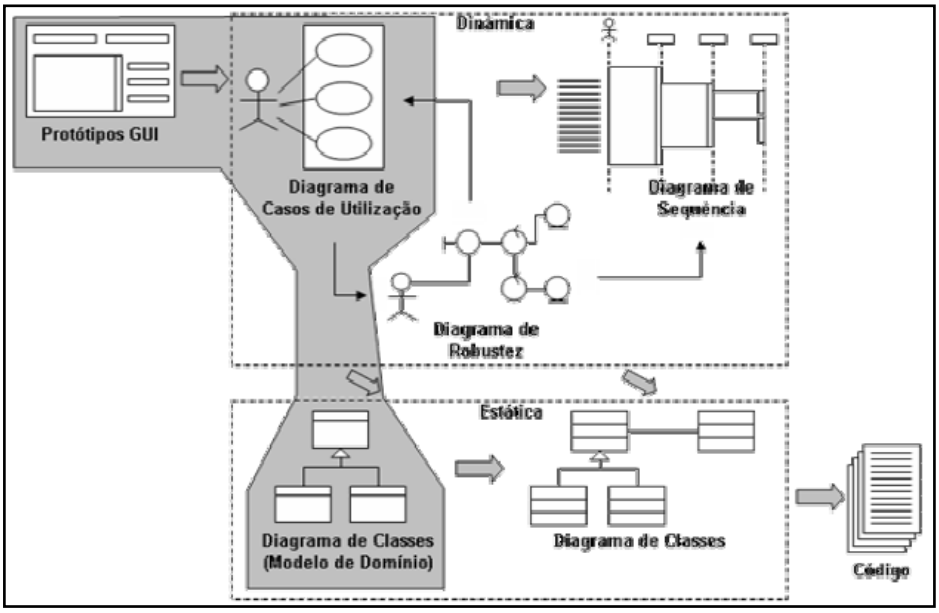
A tarefa de análise de requisitos consiste na realização das seguintes atividades:

a) identificar os objetos do mundo real e todas as relações de generalização, associação e agregação entre esses objetos. Desenhar o correspondente diagrama de classes de alto nível, designado por modelo de domínio;
b) se for razoável, desenvolver protótipos de interface homem-máquina, diagramas de

navegação, etc., de forma que os utilizadores e clientes possam entender melhor o sistema pretendido;

c) identificar os casos de utilização envolvidos no sistema. Desenhar os diagramas de casos de utilização realçando os atores envolvidos e as suas relações;
d) organizar em grupos os casos de utilização. Capturar essa organização através de diagramas de pacotes;
e) associar requisitos funcionais aos casos de utilização e aos objetos do domínio.

Na figura 1 é ilustrado a tarefa de análise de requisitos que consiste na realização de um protótipo de interface junto ao cliente, um diagrama de caso de uso e a definição do modelo de domínio.



Fonte: Vidotti (2003).

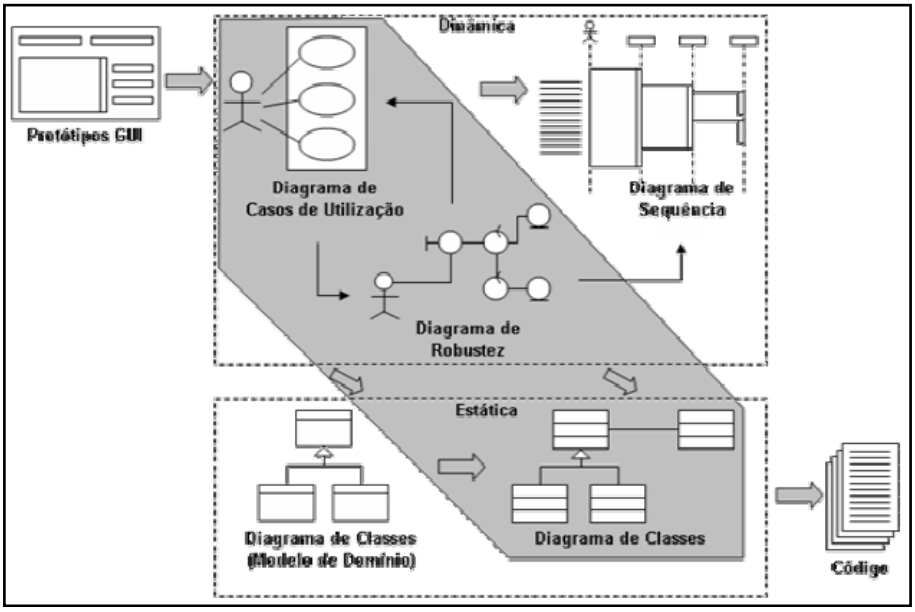
Figura 1 – Tarefa de análise de requisitos

2.1.2 ANÁLISE E PROJETO PRELIMINAR

A tarefa de análise e desenho preliminar consiste na realização das seguintes atividades:

a) fazer as descrições dos casos de utilização com os cenários principais, cenários alternativos e cenários de exceções;
b) fazer a análise de robustez;
c) terminar a atualização do diagrama de classes de modo a refletir a conclusão da fase de análise.

Já a figura 2 refere-se à tarefa de descrição dos cenários dos casos de uso, a análise de robustez para cada caso de uso e em seguida a atualização do diagrama de classes.



Fonte: Vidotti (2003).

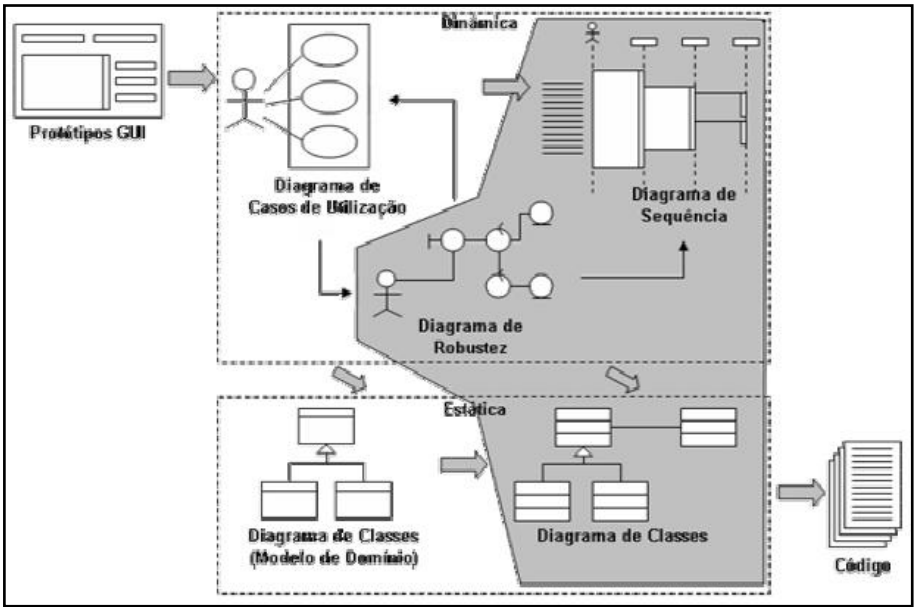
Figura 2 – Tarefa análise e projeto preliminar

2.1.3 PROJETO

A tarefa de projeto consiste na realização das seguintes atividades:

a) especificar o comportamento para cada caso de uso, identificar os objetos, as mensagens trocadas entre objetos e os métodos que são invocados. Desenhar um diagrama de seqüência com o texto do caso de utilização do lado esquerdo, e informação do desenho do lado direito. Continuar a atualizar o diagrama de classes com os objetos e atributos, entretanto descobertos;
b) terminar o modelo estático, adicionando informação detalhada sobre o projeto.

A figura 3 ilustra a tarefa de especificação do comportamento para cada caso de uso.



Fonte: Vidotti (2003).

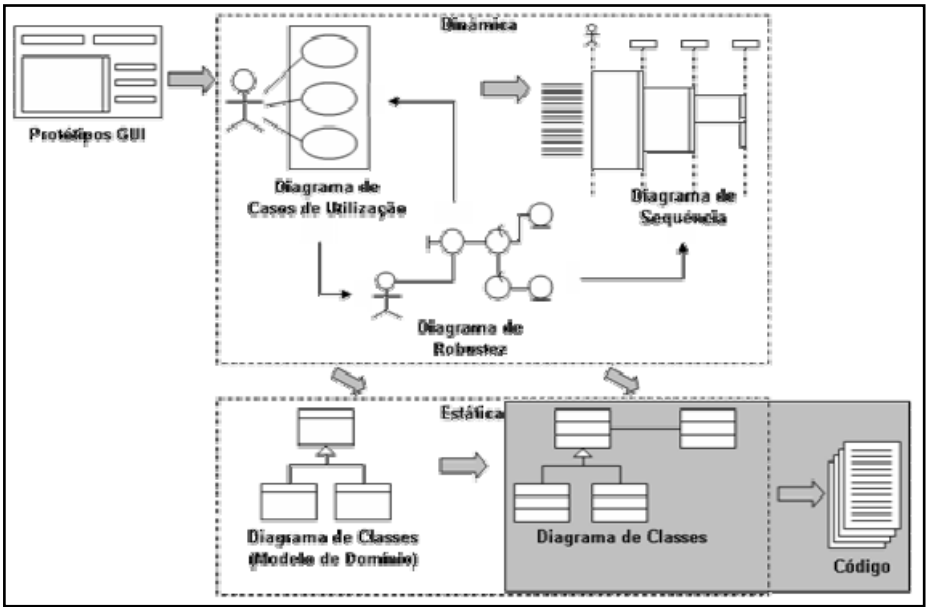
Figura 3 – Tarefa projeto

2.1.4 IMPLEMENTAÇÃO

A tarefa de implementação consiste na realização das seguintes atividades:

a) consoante as necessidades, produzir diagramas de arquitetura, tais como, diagramas de instalação e de componentes, que apóiem a tarefa de implementação;
b) escrever o código;
c) escrever testes unitário e de integração;
d) realizar testes de sistema e de aceitação.

Na figura 4 tem-se a tarefa implementação que consiste na geração do código fonte e dos testes unitários e de integração.



Fonte: Vidotti (2003).

Figura 4 – Tarefa implementação

2.2 ARQUITETURA ORIENTADA A SERVIÇOS

De acordo com Fowler (2005, p. 1) arquitetura é um termo que muitas pessoas tentam definir, com pouca concordância. Existem dois elementos comuns: um no mais alto nível é a separação de um sistema em partes, e o outro, decisões que são difíceis de alterar. Arquitetura orientada a serviços ou comumente conhecida como SOA expressa um conceito onde aplicativos ou rotinas são disponibilizados como serviços em uma rede de computadores. Orientação a serviços é expresso pelos serviços provenientes de processos de cada organização, portanto normalmente estes diferem entre si.

O conceito arquitetural do SOA não é novo e já era feito com CORBA e DCOM. SOA é uma abordagem arquitetural para criar sistemas formados de serviços autônomos. Os princípios distintos que podem ajudar a garantir uma arquitetura de baixo acoplamento são:

e) os limites são explícitos, a orientação a serviços baseia-se em um modelo de transmissão de mensagens explícitas e cruzar um limite é considerado um ato explícito;
f) os serviços são autônomos, as alterações feitas em um serviço não devem causar nenhum impacto em outro serviço, o serviço deve poder ser compilado sem causar impacto nos clientes;
g) os serviços compartilham esquemas e contratos, não classes. Esse princípio está

relacionado em grande parte a serviços que não expõem suas partes internas.

A compatibilidade do serviço é determinada com base em política. No nível do modelo de negócios, esse princípio refere-se essencialmente a controle e SLA e suas definições. A política definida descreve o recurso de semântica de um serviço com base em um conjunto de instruções explícitas das suas funcionalidades.

Segundo Sprott (2004), uma arquitetura de *software* é um conceito abstrato que dá margem a uma série de definições. Uma definição comumente utilizada afirma que uma arquitetura de *software* trata basicamente de como os componentes fundamentais de um sistema se relacionam intrinsecamente e extrinsecamente. Uma arquitetura orienta a serviços tem como seu componente fundamental o conceito de serviços. Apesar das distintas definições um serviço é usado para referenciar um componente de *software* binário baseado em um contrato. Serviço é um conceito importante, mas não são apenas WEB *services*, WEB *services* são um conjunto de protocolos com que os serviços são publicados, descobertos e utilizados em uma tecnologia neutra. SOA não é apenas uma arquitetura de serviços vista da perspectiva de uma tecnologia, mas sim políticas, padrões de projeto e *frameworks* que garantem os direitos e os limites em como um serviço é fornecido e consumido.

De acordo com Pijanowski (2007), a arquitetura orientada a serviço é uma idéia que, hoje, ocupa o pensamento de muitos desenvolvedores, arquitetos e executivos. Muitas vezes, no desenvolvimento e na comunicação de novas idéias, cria-se confusão porque definições não são estabelecidas previamente. Portanto, é importante criar um glossário de termos, bem definidos, como primeira etapa no desenvolvimento e na comunicação de novas idéias. No desenvolvimento de software, a implementação de uma idéia é a capacidade. A confusão sobre SOA começa com o próprio termo SOA. Hoje em dia, nas publicações e referências, as definições do termo SOA variam muitíssimo. Algumas alinham o termo SOA mais a uma técnica de computação distribuída que enfatiza a conectividade e a interoperabilidade.

Outras definições consideram uma visão de serviços que abrange toda a empresa e os respectivos ciclos de vida. Essas definições concentram-se em reuso, gerenciamento, governança, agilidade e alinhamento com o negócio. Essas definições repercutem entre os arquitetos corporativos e os executivos de alto escalão, responsáveis pelas mudanças organizacionais que afetam um grande portfólio de aplicativos. Mas o conceito adequado é que uma arquitetura orientada a serviço permite o baixo acoplamento, interoperabilidade, capacidade de descoberta, gerenciamento da mudança e operação dos serviços do negócio em um ambiente bem controlado.

2.3 TECNOLOGIAS MICROSOFT

A seção a seguir descreve algumas tecnologias desenvolvidas pela Microsoft que foram utilizadas neste trabalho.

2.3.1 C#

Segundo Microsoft (2005), C# é uma linguagem simples, moderna, orientada a objetos e *type-safe* derivada do C e C++. Combina a alta produtividade do Visual Basic e o poder do C++. O C# é liberado com parte integrante do Visual Studio 7.0. A plataforma Microsoft .Net define uma especificação comum para todas as linguagens para garantir a interoperabilidade entre as mesmas. Com isto qualquer linguagem compatível com a especificação comum pode fazer uso de todos as bibliotecas do conjunto de classes .Net. A figura 5 ilustra um exemplo da sintaxe da linguagem.

```
using Oxy.Common;

namespace BusinessEntityService.BusinessLogic
{
    public class OxyEntityServiceImplementation : IOxyEntityService
    {
        public ProdutoCategoria GetCategoryByName(string name)
        {
            if (string.IsNullOrEmpty(name))
                throw new ArgumentNullException(name);

            using (oxyEntities entities = new oxyEntities())
            {
                return entities.ProdutoCategoria.First(c => c.NOME == name);
            }
        }
    }
}
```

Figura 5 – Sintaxe C#

2.3.2 Asp.Net

Kalani (2002, p. 32), afirma que Asp.Net é uma infraestrutura construída dentro do .Net framework para a execução de aplicações WEB. A infraestrutura consiste em duas grandes partes, um conjunto de classes e interfaces para realizar a comunicação entre o

navegador e o servidor WEB. Uma plataforma de execução que trata todas as requisições WEB, que por sua vez carrega o CLR dentro de um processo, cria o domínio da aplicação, e executa a requisição dentro do domínio. Em um nível mais alto um aplicativo Asp.Net é executado através de uma série de requisições e repostas HTTP. Conforme Kalani (2002, p. 32) as vantagens do Asp.Net sobre o Asp 3.0 são:

a) utilização integrada do Visual Studio para o desenvolvimento e depuração;
b) trabalhar com elementos HTML como objetos;
c) mantém o estado da página automaticamente;
d) disponibilidade de uso de todas as classes do *framework* .Net;
e) performance;
f) escalabilidade;
g) segurança;
h) extensibilidade.

2.3.3 Silverlight

Segundo MacDonald (2008, p. 23), Silverlight é um *framework* para construir aplicações hospedadas em um navegador que podem ser executados em uma variedade de sistemas operacionais. O Silverlight faz a sua mágica através de um *plugin* para o navegador.

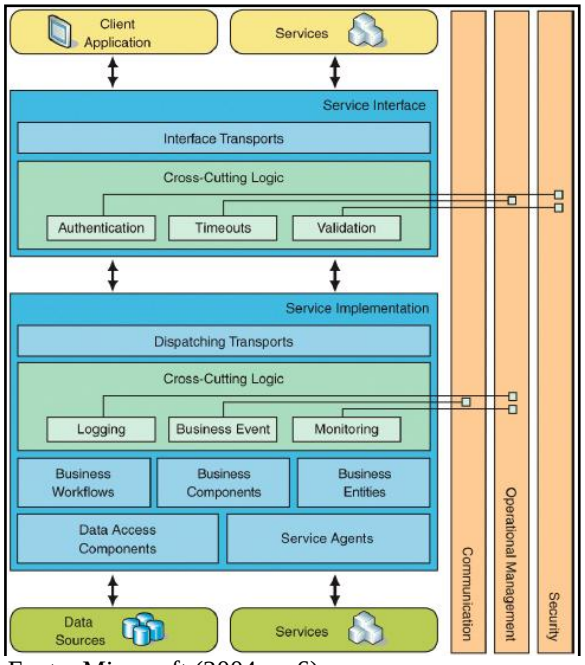
Quando o usuário está navegando em uma página na WEB que possui conteúdo Silverlight o *plugin* no navegador é ativado, executando código e renderizando o seu conteúdo. Isto parece muito familiar devido que, outras tecnologias também são um *plugin* para um navegador de internet como Java, ActiveX, ShockWave e Adobe Flash. Embora todas estas tecnologias estarem em uso nenhuma se tornou dominante no desenvolvimento de aplicativos ricos para a internet. Muitas destas tecnologias possuem vários problemas com instalação, ferramentas de desenvolvimento pobres, insuficiência na compatibilidade entre navegadores e sistemas operacionais. A única tecnologia que conseguiu desviar o desenvolvedor destes problemas foi o Adobe Flash mas somente a pouco tempo foi convertido de uma tecnologia multimídia para a internet para um conjuntos de ferramentas para a programação. No entanto oferecendo poucos recursos se comparada a plataforma Microsoft .Net. Silverlight combina uma plataforma de execução variada e todos os fundamentos de programação .Net, possibilitando que programadores de aplicativos ricos para a internet utilizem C#.

2.3.4 Windows Communication Foundation

Segundo Resnick, Crane e Bowen (2007, p. 27), WCF é um modelo de programação para desenvolver aplicativos distribuídos na plataforma Microsoft. Tornando obsoletas as tecnologias anteriores como ASMX, .NET Remoting, DCOM e MSMQ. XML *web services* é a técnica mais comum para computação distribuída nas aplicações atuais, é utilizada para expor funções técnicas e de negócio em redes privadas e públicas. Algumas vezes fazem uso de especificações SOAP, transmissão de informações como documentos de texto, protocolo HTTP para o transporte. WCF é um *framework* para desenvolver XML *web services*.

2.3.5 Enterprise Development Reference Architecture (EDRA)

EDRA é uma documentação sobre a arquitetura que uma organização poderá utilizar para padronizar o desenvolvimento de um aplicativo distribuído. O EDRA segue alguns objetivos para fazer a transição de uma arquitetura de alto acoplamento para uma arquitetura de baixo acoplamento, sendo separação da implementação de um serviço de sua interface, separação da lógica de negócio de outros requisitos periféricos como instrumentação, tratamento de exceções e rastreamento (MICROSOFT, 2007). A figura 6 demonstra as camadas lógicas da arquitetura EDRA.



Fonte: Microsoft (2004, p. 6).

Figura 6 – Arquitetura EDRA

As camadas estão dispostas entre o aplicativo cliente e a execução das regras de negócio, sendo basicamente duas camadas: a camada de interface de serviço e a camada de implementação do serviço. A camada de interface do serviço serve como fronteira, rejeitando requisições não autorizadas e inválidas. A camada de implementação do serviço invoca regras de negócio. Conforme Jezierski (2002), o desenvolvimento de aplicações distribuídas não é uma tarefa simples. Muitas decisões são tomadas em nível de arquitetura, modelagem e implementação, estas decisões terão impacto nas habilidades da segurança do aplicativo, na escalabilidade, na disponibilidade e na manutenção.

2.3.6 Enterprise library

Segundo Microsoft (2008), o Microsoft Enterprise Library é uma coleção de componentes reutilizáveis criados para ajudar o desenvolvimento de aplicativos, com preocupações que não pertencem ao negócio do aplicativo (*cross-cutting concerns*), como log, validação, acesso a dados, tratamento de exceções e outros. O Enterprise Library é distribuído como *software* livre, tendo código fonte, documentação e a possibilidade de alterações de acordo com a necessidade do desenvolvedor. Segundo Microsoft (2008) os benefícios em utilizar o Enterprise Library são:

a) foi desenvolvido pela Microsoft utilizando as melhores práticas de desenvolvimento .Net;
b) permite facilmente a customização;
c) fácil de usar, documentação de código fonte, grande preocupação com a usabilidade, ferramentas gráficas de configuração e processo de instalação simples;
d) todos os *applications blocks* foram desenvolvimentos para trabalhar em conjunto ou isoladamente.

2.3.7 LINQ

Segundo Hejlsberg (2005), após duas décadas, a indústria alcançou um ponto estável na evolução das tecnologias de programação OO. Os programadores agora aceitam naturalmente recursos como classes, objetos e métodos. Observando-se a geração atual e futura de tecnologias, torna-se aparente que o próximo grande desafio na tecnologia de

programação é reduzir a complexidade em acessar e integrar informações que não são nativamente definidas utilizando a tecnologia OO. As duas fontes mais comuns de informações não-OO são bancos de dados relacionais e XML.

Em vez de adicionar recursos relacionais ou específicos do XML as linguagens e tempo de execução de programação, com o projeto LINQ, tem-se uma abordagem mais geral adicionando facilidades de consulta de propósito geral ao .NET *framework* que se aplicam a todas as fontes de informações, não apenas a dados relacionais ou XML. Essa facilidade é chamada LINQ (.NET Language Integrated Query).

O LINQ permite expressões de consulta que se beneficiam de metadados valiosos, verificação de sintaxe em tempo de compilação, tipagem estática e IntelliSense que estavam previamente disponíveis apenas ao código imperativo. O LINQ também permite que uma facilidade de consulta declarativa de propósito geral seja declarada a todas as informações em memória, não apenas as informações derivadas de fontes externas.

O LINQ define um conjunto de operadores padrão de consulta de propósito geral que permite que operações de travessia, de filtragem e de projeção sejam expressas do modo direto, porém declarativo, em qualquer linguagem de programação baseada em .NET. Os operadores padrão de consulta permitem que consultas sejam aplicadas a qualquer fonte de informações baseada em `IEnumerable<T>`. Ao aderir às convenções do padrão LINQ, tais implementações se aproveitam da mesma integração de linguagem e suporte a ferramentas dos operadores padrão de consulta. A figura 7 demonstra a utilização do LINQ.

```
using System;
using System.Query;
using System.Collections.Generic;

class app {
  static void Main() {
    string[] names = { "Burke", "Connor", "Frank",
                       "Everett", "Albert", "George",
                       "Harris", "David" };

    IEnumerable<string> expr = from s in names
                               where s.Length == 5
                               orderby s
                               select s.ToUpper();

    foreach (string item in expr)
      Console.WriteLine(item);
  }
}
```

Fonte: Hejlsberg (2005).

Figura 7 – LINQ

2.3.8 Entity Framework

Segundo Microsoft (2006), os desenvolvedores escrevem uma grande quantidade de código para cobrir a impedância entre a representação de um dado (objeto, classe) e suas as entidades em uma base de dados. O Entity Framework cria um nível de abstração no desenvolvimento de aplicativos voltados a dados, ajudando a eliminar a impedância entre o modelo de dados e a linguagem de desenvolvimento. Para isto a Microsoft criou duas novas tecnologias o Ado.Net Entity Framework e o LINQ. A figura 8 demonstra uma atualização de uma entidade de dados sem a utilização de SQL, apenas a manipulação de objetos.

```
using(OrderTracking orderTracking = new OrderTracking())
{
    var orders = from order in orderTracking.SalesOrders
                 where order.Status == "Pending Stock Verification" &&
                       order.SalesPerson.State == "WA"
                 select order;

    foreach(SalesOrder order in orders)
    {
        List<StockAppProduct> products = new List<StockAppProduct>(
            from orderLine in order.Lines
            select new StockAppProduct {
                ProductID = orderLine.Product.ID,
                LocatorCode = ComputeLocatorCode(orderLine.Product)
            }
        );

        if(StockApp.CheckAvailability(products))

            order.Status = "Shippable";
    }
    orderTracking.SaveChanges();
}
```

Fonte: Microsoft (2006).

Figura 8 – Microsoft Ado.Net Entity Framework

2.4 PADRÕES DE PROJETO

Segundo Gamma, Helm, Johnson e Vlissides (2004, p. 324), estudos de programadores especialistas em linguagens convencionais mostram que o conhecimento e a experiência não são organizados simplesmente em torno da sintaxe, mas sim em torno de estruturas conceituais maiores, tais como algoritmos e estrutura de dados. Os cientistas da computação nomeiam e catalogam algoritmos e estruturas de dados, porém, raramente são nomeados

outros tipos de padrões. Os padrões de projetos fornecem um vocabulário comum para comunicar, documentar e explorar alternativas de projeto. Os padrões de projeto tornam um sistema menos complexo ao permitir falar sobre ele em um nível de abstração mais alto do que aquele de uma notação de projeto ou uma linguagem de programação. Os padrões de projeto elevam o nível no qual o projetista discute o projeto. Como exemplo o padrão *command* que encapsula uma solicitação como um objeto, desta forma permitindo que seja parametrizado clientes com diferentes solicitações ou o padrão *factory* que define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe a ser instanciada. Este padrão permite a uma classe postergar a instanciação de subclasses.

De acordo com Hofstader (2007), em todas as disciplinas de engenharia maduras existem padrões. Na engenharia de software também é assim. Aprender como aproveitar os padrões para definir uma solução de software requer tempo e esforço. Não basta conhecer o vocabulário dos padrões para ser proficiente em projeto. A capacidade de aplicar padrões para criar estruturas específicas de domínio é o mais importante. A melhor forma de obter proficiência na aplicação de padrões é projetar e desenvolver soluções de software, na prática.

Segundo Hofstader (2007) com experiência e reforços constantes, o projeto de uma solução de software baseada em padrões passará a ser um hábito. Os padrões permitem aos arquitetos e projetistas de software ter a capacidade de projetar rapidamente uma solução robusta, utilizando técnicas comprovadas. Os padrões também dão aos profissionais de software um vocabulário comum, com base no qual podem transmitir idéias sem ter de descrever cada detalhe do propósito do projeto.

O uso de padrões para definir uma solução de software é uma tarefa analítica que exige pensamento abstrato. Existem vários padrões documentados que permitem a definição da estrutura e do comportamento da solução, em diferentes níveis de abstração, para a arquitetura da solução e para as estruturas contidas na solução afima Gamma, Helm, Johnson e Vlissides (2004, p. 324).

Conhecer os limites é a primeira etapa para definir uma solução de software. Definir os limites do sistema acrescenta enfoque à solução, permitindo que seja projetada e desenvolvida. Sem limites claros, a solução será um alvo móvel e poderá mergulhar na ambigüidade. Em geral, existem alguns requisitos de sistema, formais ou informais, provenientes de diferentes participantes, que servirão de base para derivar os limites da solução.

Além de ter um conhecimento perfeito do propósito global da solução, os seguintes requisitos do sistema são críticos na definição dos limites: eventos externos que acionam a

funcionalidade da solução e os processos externos dos quais depende a solução. Considerado de modo holístico, esses requisitos permitem ao arquiteto definir o contexto da solução. Ter consciência dos eventos que desencadeiam o comportamento da solução e conhecimento dos dados que residem nas dependências da solução permitem ao arquiteto "normalizá-la", separando os objetos internos dos externos à solução. Entender os processos dos quais depende a solução permite ao arquiteto limitar o comportamento da solução, aproveitando aquele dos processos externos.

Depois de os limites da solução estarem definidos, o arquiteto estará pronto para decidir como estruturar a solução. A estrutura da solução proporciona um entendimento conceitual do espaço da solução e deve ser definida com bastante antecipação no processo, pois esta é uma etapa crítica para a definição do framework arquitetural da solução que ajudará a garantir a consistência de toda a solução, com o objetivo de fazê-la mais extensível e de fácil manutenção. Similarmente à definição dos limites de uma solução, existem vários requisitos de sistema que permitem aos arquitetos definir a estrutura da solução. Às vezes, esses requisitos poderão, aparentemente, se contradizer, e várias compensações terão de ser ponderadas. Os padrões estruturais escolhidos para uma solução não devem acarretar a implementação de quaisquer tecnologias específicas. As tecnologias usadas para implementar a solução devem se basear na capacidade da tecnologia de atender aos requisitos do sistema, funcionais ou não, dentro da estrutura da solução definida.

Com os requisitos do sistema usados para definir os limites da solução, existem alguns outros requisitos que podem pesar sobre a estrutura da solução: requisitos de segurança, desempenho e transacionais

Os padrões são ferramentas importantes para a análise e o projeto de uma solução de software. Oferecem ao arquiteto a possibilidade de conceitualizar uma solução em diferentes níveis e são, ainda, uma forma valiosa de comunicar conceitos entre os profissionais de software.

Arquiteturas e projetos de solução devem mapear os requisitos funcionais. Ao implementar padrões no projeto de uma solução, o arquiteto pode criar estruturas específicas de domínio para fazer com que a solução de software seja totalmente consistente. Considerando que a arquitetura e o projeto de sistemas são altamente subjetivos, a melhor forma de avaliar uma estrutura é saber se cumpre os requisitos funcionais definidos e se é flexível, extensível e permite manutenção.

Não é preciso ter um processo formal para definir uma solução baseada em padrões, uma abordagem ágil para codificar os requisitos funcionais a estruturas baseadas em padrões

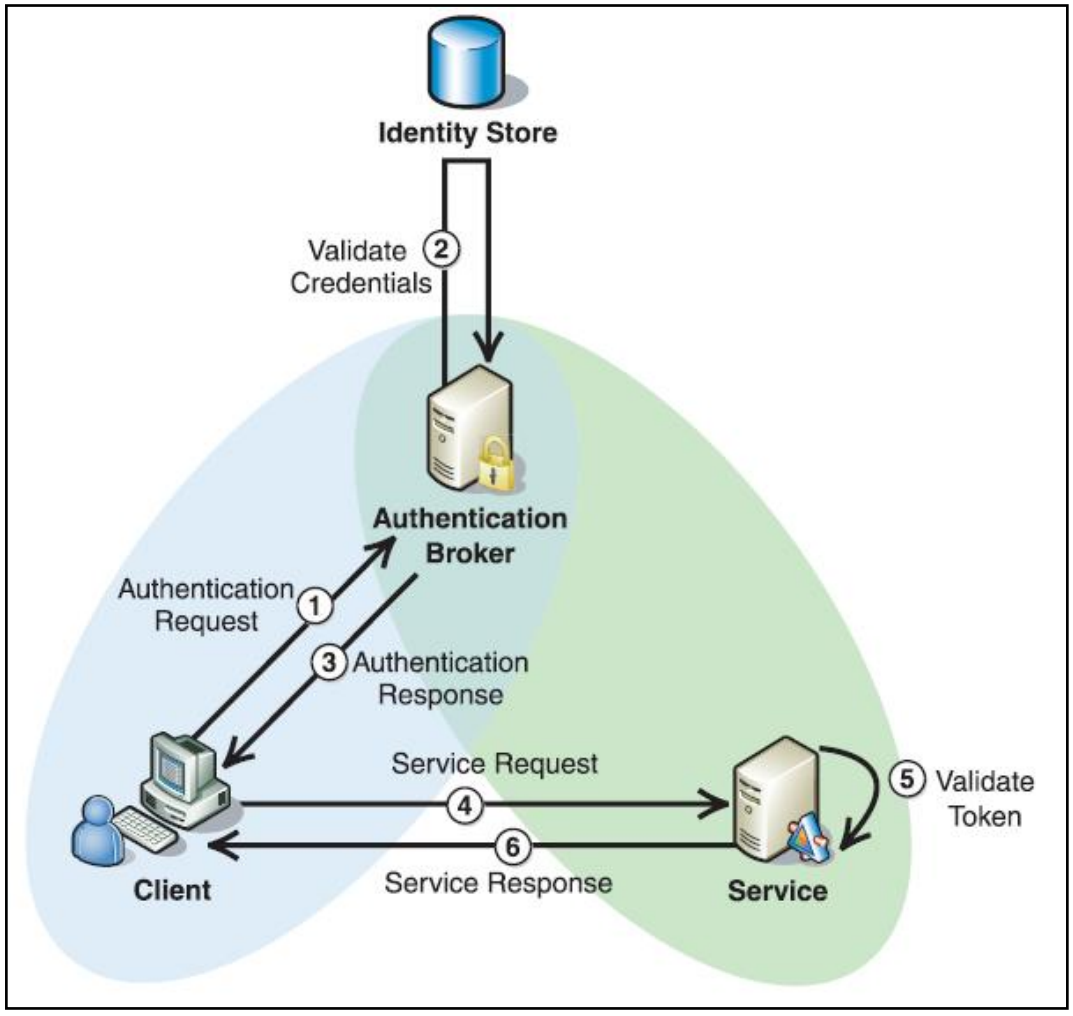
poderá ser ideal se o domínio não estiver inicialmente bem definido. Ao contrário, se o domínio estiver bem definido, o projeto da solução poderá ser iniciado com padrões e modificado para cumprir outros requisitos, como desempenho.

2.5 SECURITY TOKEN SERVICE

De acordo com Microsoft (2003), a segurança das mensagens trafegadas entre a camada de apresentação e a camada de negócio se dá através de um serviço federado denominado STS, que prove a integridade, a confidencialidade e o não-repúdio das mensagens trafegadas entre a camada de apresentação e a camada de serviço. A figura 9 ilustra as etapas da comunicação entre as partes cliente, serviço, STS e base de identidades. Ao iniciar uma requisição ao serviço de entidades o mecanismo verifica se já existe um *token* de segurança válido para o usuário, caso não exista o mesmo é solicitado ao STS. Onde é verificada a autenticidade das credenciais fornecidas pelo cliente e gerado um token criptografado e assinado digitalmente. Ao receber o *token* o cliente o guarda em um cachê para futuras requisições e anexa o *token* a cada mensagem enviada ao serviço de entidades. Que por sua vez valida o *token* utilizando chaves assimétricas, não requerendo que o serviço de entidade valide novamente as credenciais do cliente, este processo se chama SSO.

O serviço de entidade não necessita revalidar as credenciais do usuário pois foi previamente estabelecido uma relação de confiança entre a máquina que hospeda o STS e a máquina que hospeda o serviço de entidades. A relação de confiança é estabelecida através de chaves assimétricas públicas e privadas que somente ambas máquinas possuem, sendo assim um *token* criptografado com a chave pública do serviço de entidades poderá ser descriptografado com a chave privada do serviço de entidades. Além da criptográfica o *token* é assinado pelo STS no momento da geração possibilitando ao serviço de entidade pode confirmar que o *token* foi emitido pelo STS e que o *token* não foi adulterado durante o processo.

Pelo uso desta relação de confiança as máquinas do STS e do serviço de entidade podem ficar isoladas por uma DMZ. Os *tokens* trafegados utilizam um padrão de mercado chamado SAML que utiliza XML provendo interoperabilidade entre qualquer elemento. A versão do protocolo utilizado neste trabalho é a especificação 1.1.



Fonte: Microsoft (2003).

Figura 9 – *Security Token Service*

2.6 TRABALHOS CORRELATOS

Foram encontrados alguns trabalhos como o CSLA.Net pela coerência com os paradigmas atuais, pela clareza nos temas abordados. Podendo citar ainda como trabalho correlato o trabalho concluído por Marcelo Domingos (DOMINGOS, 2004) que constitui em uma arquitetura que dá base ao desenvolvimento de sistemas da informação e portais de serviços de governo eletrônico.

2.6.1 CSLA.Net

Rockford (2007) afirma que o CSLA.Net é um *framework* para o desenvolvimento de aplicativo que reduz o custo de desenvolvimento e manutenção. É uma base tecnológica para o desenvolvimento de software que fornece uma maneira padronizada de utilizar objetos de negócio para desenvolver aplicativos orientados a objetos. Objetos de negócio são objetos que abstraem entidades de negócio em um programa orientado a objetos. Estas entidades podem ser pedidos, funcionários, faturas, etc. O CSLA surgiu em 1996, para Visual Basic 6 e COM+, e na sua evolução atual de 2007 utiliza os recursos do Framework .NET 2.0, da versão 2005 das linguagens Visual Basic e C#. Um fator limitador importante deste framework são as tecnologias desenvolvidas pelo próprio autor que desconsidera as tecnologias desenvolvidas pela Microsoft, criando assim soluções que poderão ficar incompatíveis com componentes os desenvolvidos pela Microsoft.

2.6.2 Arquitetura de referência para sistemas da informação e portais de serviços de governo eletrônico

Segundo Domingos (2004, p. 13) a arquitetura de referência para sistemas da informação e portais de serviços de governo eletrônico é voltada aos requisitos das aplicações do governo, cujos processos de desenvolvimento estejam baseados em uma estrutura que permita acelerar a produção de aplicações para um custo mais baixo e com ganhos na qualidade final.

Domingos (2004, p. 21) afirma especificar uma arquitetura bem definida para sistemas de informação e portais de serviços governamentais que respeitem a arquitetura integrada de governo eletrônico e que sejam voltados à captura de informações junto a cidadãos, visando aumentar a qualidade das aplicações e reduzir os custos no desenvolvimento. Uma arquitetura bem definida estabelece modelos abstratos e precisos que permitem projetar, implementar e manter um aplicativo, avaliando e garantindo suas qualidades. As limitações quanto ao trabalho proposto é a dificuldade de escalabilidade, Domingos especifica uma arquitetura fortemente amarrada entre os componentes da camada de negócio e a camada de apresentação inviabilizando uma separação das camadas físicas e lógicas de arquitetura.

3 DESENVOLVIMENTO DO APLICATIVO

Neste capítulo serão apresentados os requisitos do trabalho, bem como a sua especificação e ferramentas utilizadas para a implementação e a discussão do resultado obtido através das etapas do processo Iconix.

3.1 ANÁLISE DE REQUISITOS

A tarefa de análise de requisitos consiste na identificação dos requisitos funcionais, no desenvolvimento do diagrama de domínio, a prototipagem da interface junto ao usuário e identificar os casos de usos.

3.1.1 Requisitos funcionais

O aplicativo proposto neste documento tem como requisitos funcionais:

a) permitir o cadastro de fornecedores de produtos ofertados na loja;
b) permitir o cadastro de categorias de produtos;
c) permitir o cadastro de produtos;
d) permitir o cadastro de usuários de administração do aplicativo;
e) permitir o acesso ao aplicativo de clientes não cadastrados;
f) permitir que os clientes possam se cadastrar;
g) disponibilizar um conjunto de consultas a produtos, por preço, categoria e fornecedor;
h) permitir que o cliente possa selecionar os produtos que deseja comprar;
i) permitir que o cliente possa finalizar a compra informando a forma de pagamento e o endereço para entrega;
j) permitir que o cliente possa salvar as consultas efetuadas;
k) permitir que o cliente possa consultar as suas compras efetuadas;
l) permitir a manipulação dos cadastros somente por usuários administrativos.

3.1.2 Requisitos não funcionais

O aplicativo proposto neste documento tem como requisitos não funcionais:

a) ter a camada de apresentação implementada em RIA;
b) implementar um mecanismo de transporte de mensagens segura entre camada de apresentação e camada de fachada de serviços;
c) implementar conceitos da arquitetura EDRA;
d) implementar a arquitetura orientada a serviços;
e) ser implementado utilizando o Microsoft .Net Framework 3.5;
f) ser implementado na liguagem C#;
g) utilizar banco de dados Microsoft Sql Server 2008;
h) ser compatível com os sistemas operacionais Microsoft Windows XP, Vista ou superior;
i) ser compatível com os navegadores de Internet Mozilla FireFox, Microsoft Internet Explorer, Apple Safari e Google Chrome.

3.1.3 Modelo de domínio

A figura 10 apresenta o diagrama de objetos de domínio do aplicativo, sendo estes: `fornecedor`, `produto`, `produtoCategoria`, `compra`, `usuário`, `cliente` e `consulta`. Os objetos de domínio são as principais entidades do aplicativo.

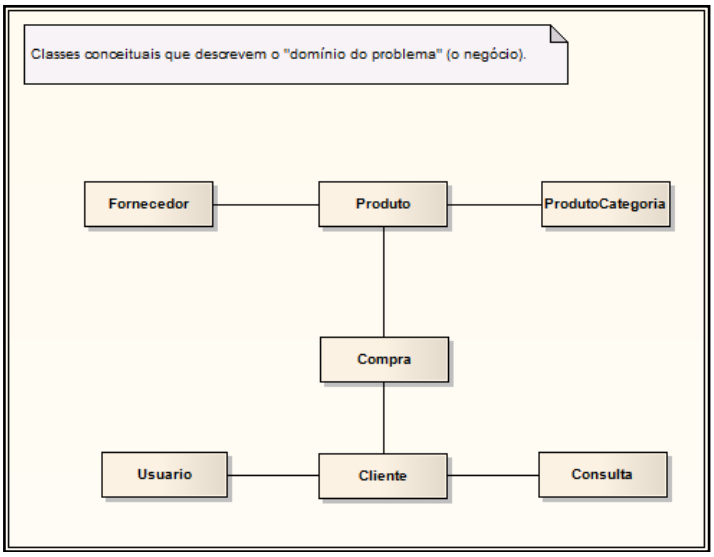


Figura 10 – Diagrama de objetos de domínio

3.1.4 Prototipagem

Nesta etapa do processo a atividade de desenvolvimento de protótipos de interface com o usuário é opcional, e tendo em vista a baixa complexidade do aplicativo proposto não se deferiu a necessidade de prototipagem.

3.1.5 Diagrama de caso de uso

Esta atividade constitui na identificação da funcionalidade do aplicativo a partir da atuação dos atores envolvidos. A figura 11 apresenta os casos de uso pertinente ao usuário (cliente) do aplicativo, que são: incluir produto a compra, finalizar compra, cadastrar cliente, consultar produtos já comprados, consultar produtos e manter consultas. Os casos de uso do usuário administrador do site não são apresentados.

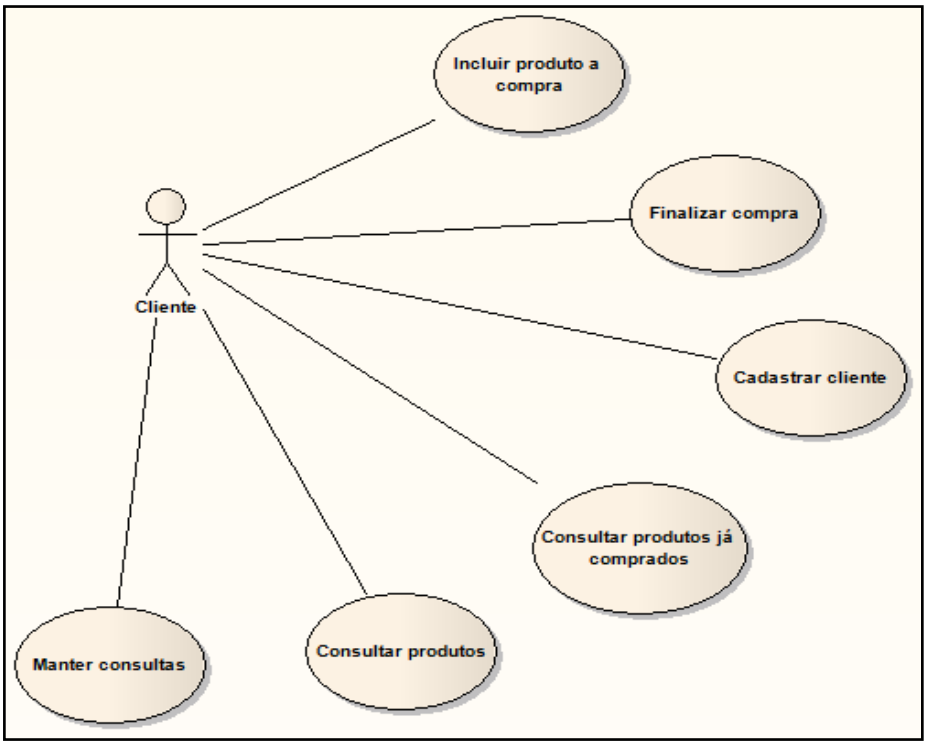


Figura 11 – Diagrama de caso de uso do aplicativo

O caso de uso “incluir produto a compra” refere-se à funcionalidade que permite o usuário adicionar um produto a compra ainda em andamento. Enquanto que o caso de uso “finalizar compra” refere-se a encerrar o processo de compra, a totalização do valor dos produtos adicionados e possibilita o usuário informar o endereço de envio e a forma de pagamento. Já o caso de uso “cadastrar cliente” diz respeito a incluir o usuário ao cadastro de

cliente possibilitando a finalização de uma compra.

Há o caso de uso “consultar produtos já comprados”, que exibe ao usuário uma listagem com todas as compras efetuadas e o acompanhamento do despacho de compras ainda não recebidas. Ainda há o caso de uso “consultar produtos”, que permite ao usuário verificar todos os produtos disponíveis. O caso de uso “manter consultas” refere-se à funcionalidade que possibilita o usuário a desenvolver suas próprias pesquisas sobre os produtos ofertados e ainda persistindo no cadastro do usuário para que em futuras visitas o mesmo posso usufruir das consultas previamente criadas.

3.2 ANÁLISE E DESENHO PRELIMINAR

Esta atividade descreve todos os cenários dos casos de uso do aplicativo e o diagrama de classes.

3.2.1 Caso de uso incluir produto a compra

No quadro 1 é descrito o caso de uso incluir produto que refere-se à funcionalidade que permite o usuário adicionar um produto a compra ainda em andamento.

Descrição	Incluir produto a compra
Ator	Cliente
Pré-condições	Deve existir produtos cadastrados
Fluxo principal	1. O sistema apresenta a página com um produto e a opção de comprar. 2. O usuário pressiona o botão comprar. 3. O sistema adiciona o produto a compra do usuário. 4. O sistema redireciona o usuário para a página de resumo da compra.
Fluxos alternativos e exceções	
Pós-condições	Produto adicionado a lista de compra

Quadro 1 – Descrição do caso de uso incluir produto a compra

3.2.2 Caso de uso finalizar compra

No quadro 2 é descrito o caso de uso finalizar compra que refere-se a encerrar o processo de compra, a totalização dos produtos adicionados e possibilita ao usuário informar o endereço para envio e a forma de pagamento.

Descrição	Finalizar compra
Ator	Cliente
Pré-condições	Possuir no mínimo um produto na lista de compra
Fluxo principal	1. O sistema apresenta a página de resumo da compra. 2. O usuário informa o CEP para o cálculo do frete. 3. O usuário pressiona o botão finalizar compra. 4. O sistema valida as informações 5. O sistema apresenta a tela de finalização da compra. 6. O usuário informa o endereço de envio e a forma de pagamento e pressiona o botão concluir. 7. O sistema valida as informações 8. O sistema finaliza a compra 9. O sistema redireciona o usuário a página inicial do aplicativo.
Fluxos alternativos e exceções	No item 3, ao pressionar o botão finalizar compra e o usuário não estiver logado no aplicativo: 3.1. O sistema apresenta a página de cadastro de usuário ou login. 3.2. O usuário informa as suas credencias e pressiona acessar e vai para o passo 4. No item 4, caso o CEP informado não seja válido uma mensagem é exibida ao usuário No item 7, caso as informações de pagamento não esteja corretas uma mensagem é exibida ao usuário
Pós-condições	Compra finalizada

Quadro 2 – Descrição do caso de uso finalizar compra

3.2.3 Caso de uso cadastrar cliente

O quadro 3 descreve o caso de uso cadastrar cliente que refere-se a inclusão do usuário ao cadastro de clientes possibilitando a finalização de uma compra.

Descrição	Cadastrar cliente
Ator	Cliente
Pré-condições	
Fluxo principal	1. O sistema apresenta a página de cadastro de cliente 2. O usuário informa o nome, senha, confirmação da senha e pressiona o botão cadastrar. 3. O sistema valida as informações 4. O sistema cadastra o cliente. 5. O sistema redireciona o usuário para a página principal.
Fluxos alternativos e exceções	No item 2, ao pressionar o botão cadastrar mas as informações não foram validadas: 2.1. O sistema apresenta uma mensagem informando que os dados informados pelo usuário não são válidos. No item 3, caso as informações fornecidas pelo cliente não forem válidas o sistema apresenta uma mensagem para cada informação não válida
Pós-condições	O cliente é autenticado no site

Quadro 3 – Descrição do caso de uso cadastrar cliente

3.2.4 Caso de uso consultar produtos já comprados

No quadro 4 é descrito o caso de uso consultar produtos já comprados que exibe ao usuário uma listagem com todas as compras já efetuadas.

Descrição	Consultar produtos já comprados
Ator	Cliente
Pré-condições	O cliente deverá estar autenticado no site
Fluxo principal	1. O sistema apresenta na página principal o link minhas compras na parte superior direita da página. 2. O cliente clica no link minhas compras 3. O sistema redireciona o usuário para a página minhas compras. 4. O sistema apresenta uma lista com as compras do usuário.
Fluxos alternativos e exceções	
Pós-condições	

Quadro 4 – Descrição do caso de uso consultar produtos já comprados

3.2.5 Caso de uso consultar produtos

No quadro 5 é descrito o caso de uso consultar produtos que permite ao usuário visualizar todos os produtos disponíveis a venda.

Descrição	Consultar produtos
Ator	Cliente
Pré-condições	Deve existir produtos cadastrados, o cliente está na página principal
Fluxo principal	1. O sistema apresenta a página principal, na barra superior são exibidos as categorias dos produtos e a caixa de pesquisa, ao centro da página são exibidos os produtos em oferta e na barra inferior os produtos mais vendidos. 2. O cliente pressiona o link de uma categoria de produto. 3. O sistema redireciona o usuário para a página de produtos da categoria selecionada.
Fluxos alternativos e exceções	No item 1, o cliente informa um texto na caixa de pesquisa e pressiona o botão pesquisar: 1.1. O sistema apresenta uma lista com os produtos que atendem ao critério de seleção informado na pesquisa. No item 1, o cliente pode selecionar um produto no centro da página: 1.1. O sistema redireciona o usuário a página do produto. No item 1, o cliente seleciona um produto na barra inferior da página: 1.1. O sistema redireciona o usuário para a página do produto.
Pós-condições	Um produto será exibido ao cliente

Quadro 5 – Descrição do caso de uso consultar produtos

3.2.6 Caso de uso manter consultas

O quadro 6 descreve o caso de uso manter consultas que refere-se a funcionalidade que permite ao usuário desenvolver suas próprias pesquisas sobre os produtos ofertados e ainda persistir no cadastro de cliente para que em futuras visitas o mesmo possa usufruir das suas consultas previamente criadas.

Descrição	Manter consultas
Ator	Cliente
Pré-condições	O cliente deverá estar autenticado no site
Fluxo principal	1. O sistema apresenta na página principal na parte superior direita o link minhas consultas. 2. O cliente pressiona o botão minha consultas. 3. O sistema redireciona o cliente a página de consultas. 3. O sistema apresenta a página com as consultas já criadas pelo usuário e as opções incluir, editar e remover. 4. O cliente pressiona o botão incluir. 5. O sistema apresenta a página de inclusão de consulta. 6. O cliente informa o nome da consulta, o tipo do critério de seleção, o valor e pressiona salvar. 7. O sistema valida as informações 8. O sistema redireciona o cliente para a página de consultas do cliente.
Fluxos alternativos e exceções	No item 4, o cliente obta pela edição No item 4, o cliente obta pela exclusão A qualquer momento o cliente pode cancelar a operação e retornar ao passo 1.
Pós-condições	Consulta incluída, alterada ou excluída

Quadro 6 – Descrição do caso de uso manter consultas

3.2.7 Diagrama de robustez do caso de uso incluir produto a compra

A figura 12 ilustra o diagrama de robustez do caso de uso incluir produto a compra, sendo que o usuário interage com a interface Silverlight que por sua vez aciona o controlador adicionar a compra e faz uso da entidade produto.

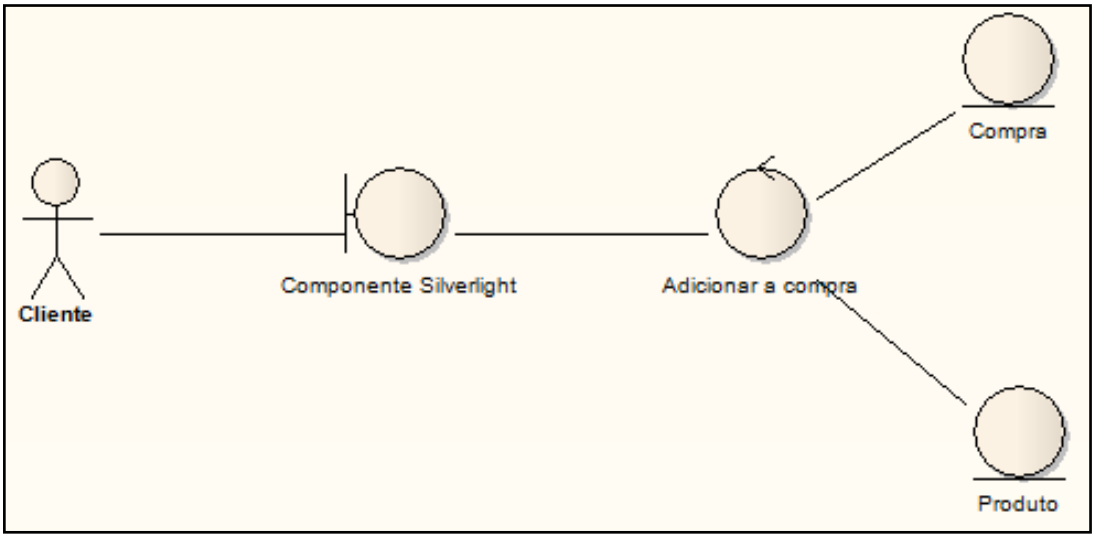


Figura 12 – Diagrama de robustez do caso de uso incluir produto a compra

3.2.8 Diagrama de robustez do caso de uso finalizar compra

A figura 13 demonstra o diagrama de robustez do caso de uso finalizar compra, sendo que o usuário interage com a página de resumo de compra que aciona o controlador finalizar compra no componente Silverlight através do objeto compra.

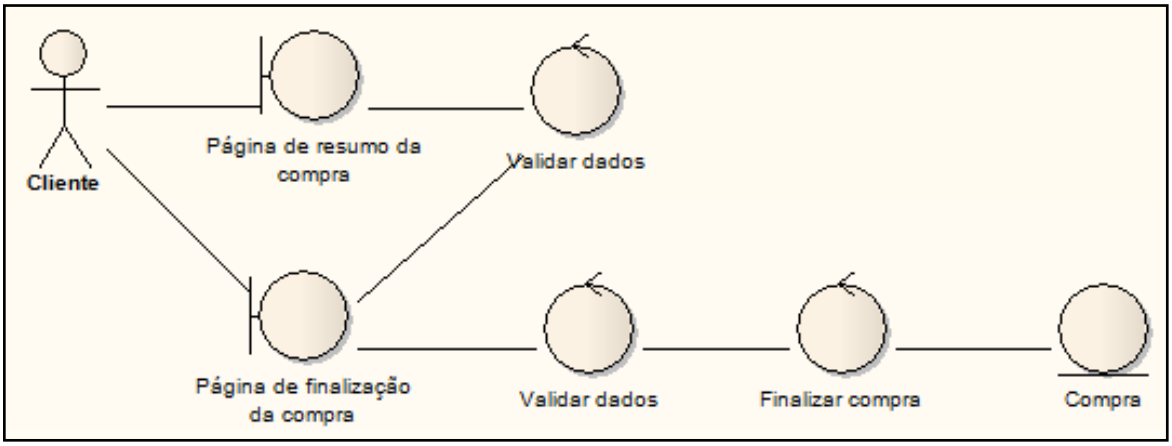


Figura 13 – Diagrama de robustez do caso de uso finalizar compra

3.2.9 Diagrama de robustez do caso de uso cadastrar cliente

Na figura 14 é representado o diagrama de robustez do caso de uso cadastrar cliente, onde o usuário interage com a página de cadastro invocando o controlador validar dados através do botão salvar. Que por sua vez aciona o controlador incluir dados e manipula a entidade cliente.

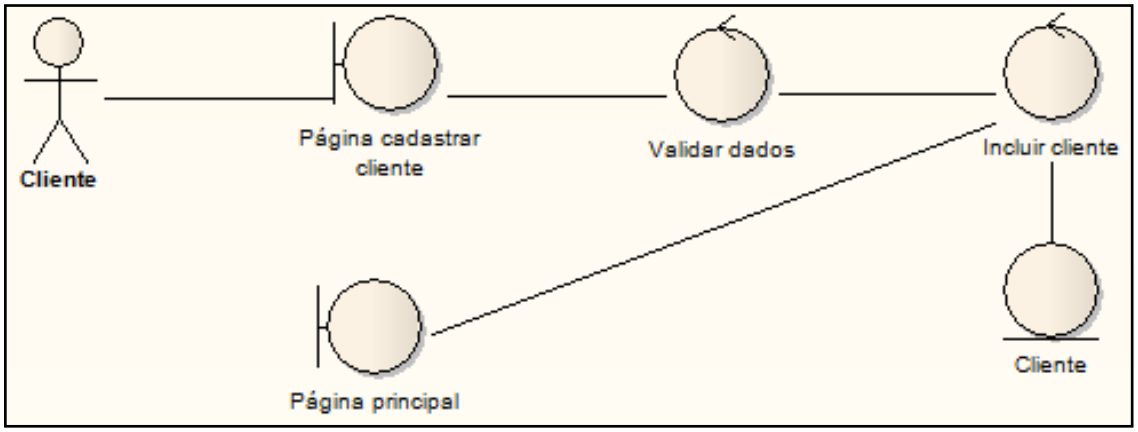


Figura 14 – Diagrama de robustez do caso de uso cadastrar cliente

3.2.10 Diagrama de robustez do caso de uso consultar produto já comprado

Na figura 15 é representado o diagrama de robustez do caso de uso consulta produtos já comprados. Onde o usuário através da página de *login* aciona o controlador obter últimas compras no componente Silverlight que obtém uma coleção de entidade do tipo compra.

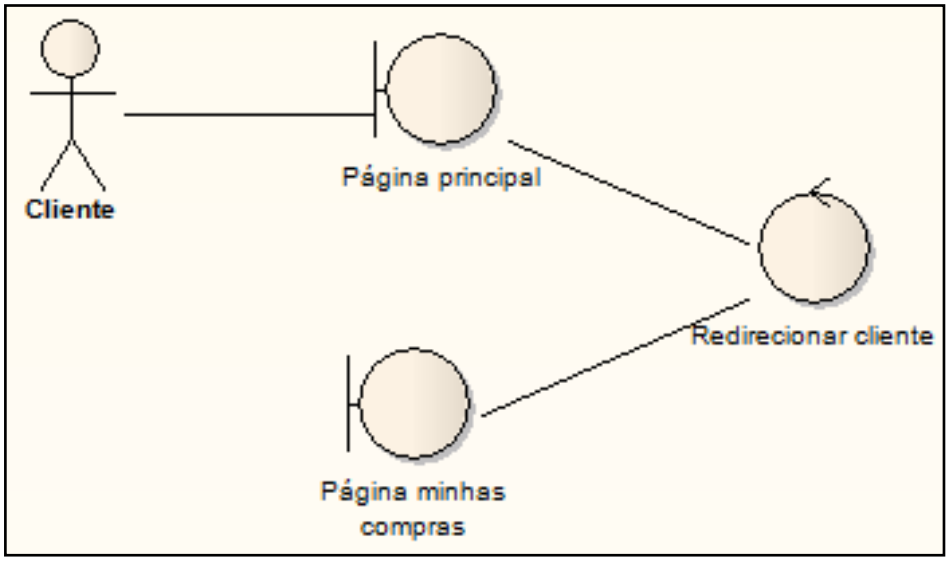


Figura 15 – Diagrama de robustez do caso de uso consultar produtos já comprados

3.2.11 Diagrama de robustez do caso de uso consultar produtos

A figura 16 demonstra o diagrama de robustez do caso de uso consultar produto, onde o usuário através da página principal interage com o menu ou o painel principal para selecionar a categoria desejada, na seqüência o controlador obter produtos no componente Silverlight obtém uma coleção de entidade do tipo produto.

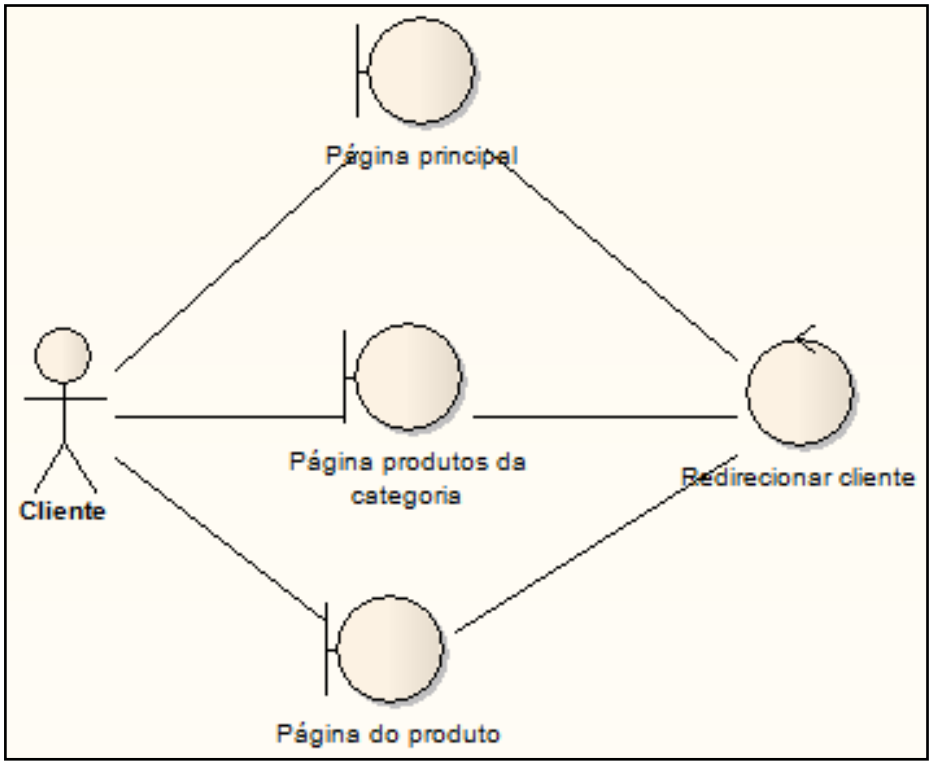


Figura 16 – Diagrama de robustez do caso de uso consultar produto

3.2.12 Diagrama de robustez do caso de uso manter consultas

A figura 17 ilustra o diagrama de robustez do caso de uso manter consultas, o usuário interage com a interface de consultas e através de controladores realiza as operações de criação, leitura, gravação e exclusão de objetos consulta.

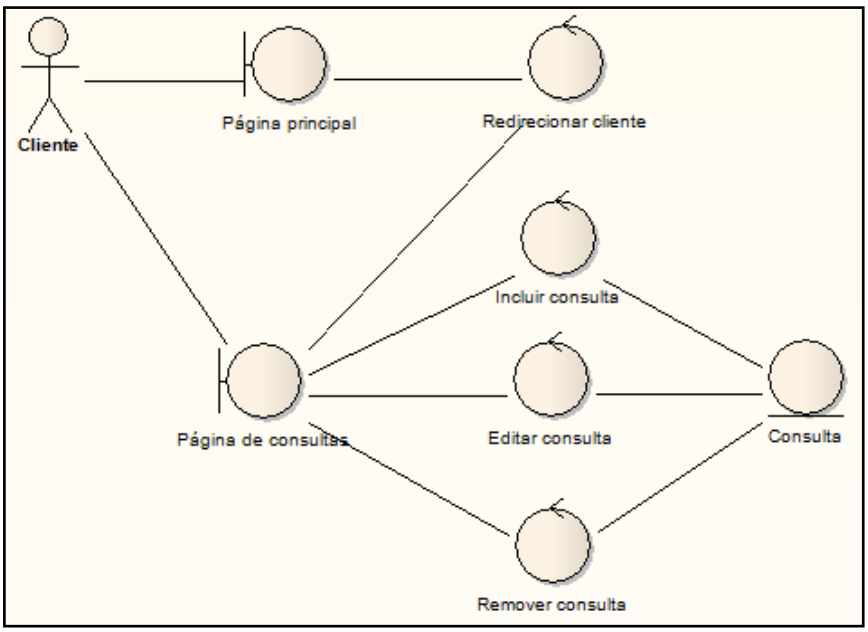


Figura 17 – Diagrama de robustez do caso de uso manter consultas

3.2.13 Diagrama de classes

Na figura 18 é ilustrado o diagrama de classes do domínio com as seguintes classes: `produto`, `produtoCategoria`, `fornecedor`, `compra`, `cliente` e `usuário`.

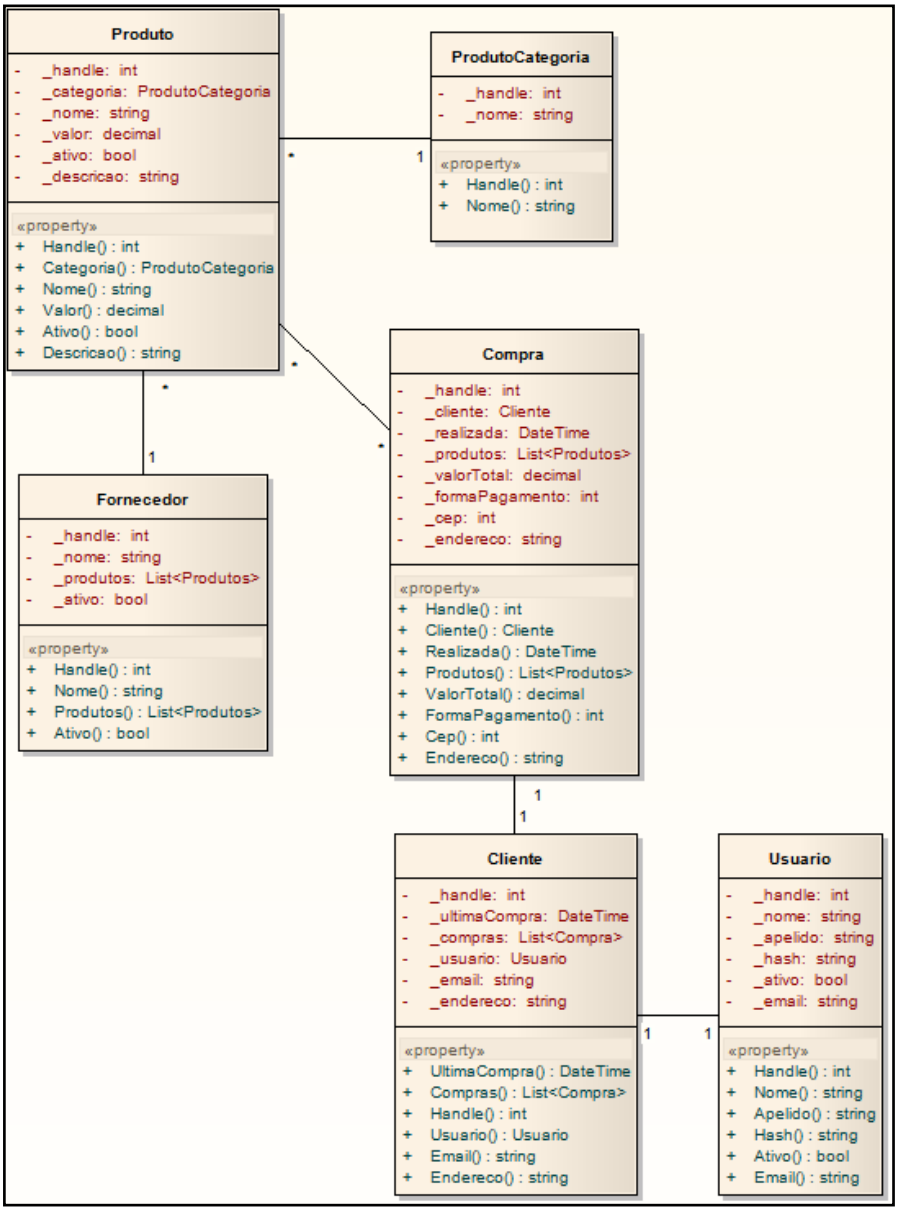


Figura 18 – Diagrama de classes do domínio

3.3 DESENHO

Esta atividade especifica o comportamento do aplicativo para cada caso de uso.

3.3.1 Diagrama de seqüência do caso de uso incluir produto a compra

A figura 19 apresenta o caso de uso incluir produto a compra. O usuário seleciona o produto desejado e pressiona o botão comprar, por sua vez o componente Silverlight invoca a chamada do método remoto que adiciona o produto a sessão do usuário no servidor de camada de apresentação.

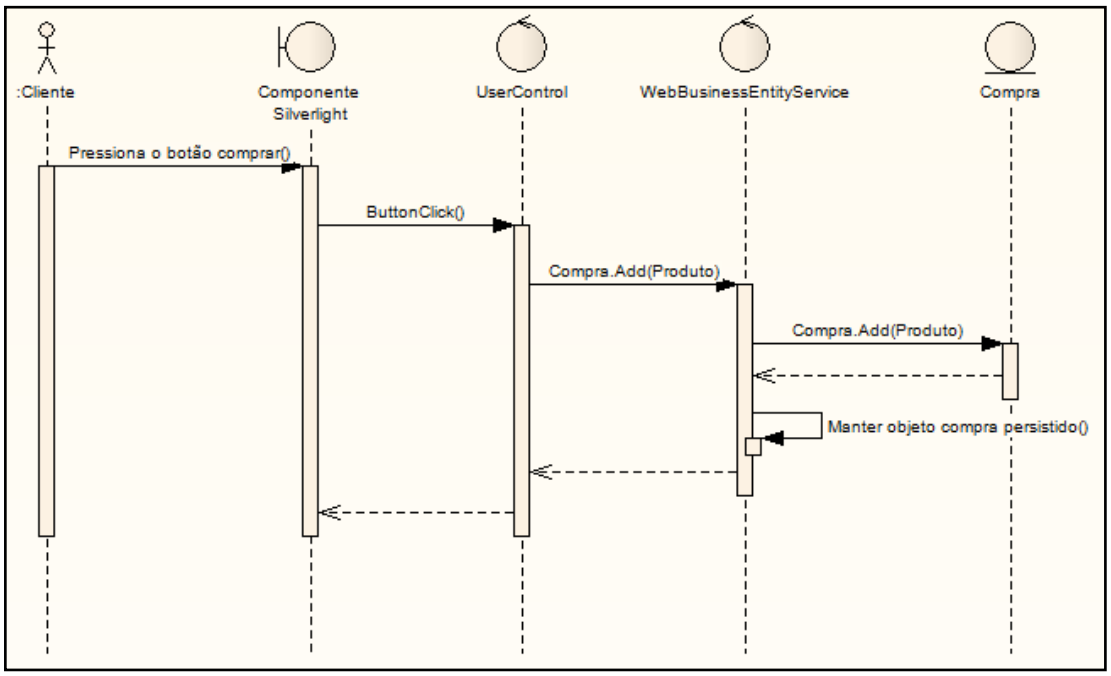


Figura 19 – Diagrama de seqüência do caso de uso adicionar produto a compra

3.3.2 Diagrama de seqüência do caso de uso finalizar compra

Na figura 20 é representado o diagrama de seqüência do caso de uso finalizar compra, sendo que o usuário está na página de resumo da compra e pressiona o botão finalizar compra, o qual é responsável pela chamada remota do método totalizar do objeto compra. Resultado na totalização dos valores a descriminação dos produtos da compra, requerendo ao usuário as informações de endereço para envio e forma de pagamento. Após o informe dos dados necessários o usuário pressiona o botão comprar que aciona o método finalizar do componente Silverlight que através de uma chamada remota faz com que uma instancia do objeto compra seja persistido na base de dados através da camada de negócio.

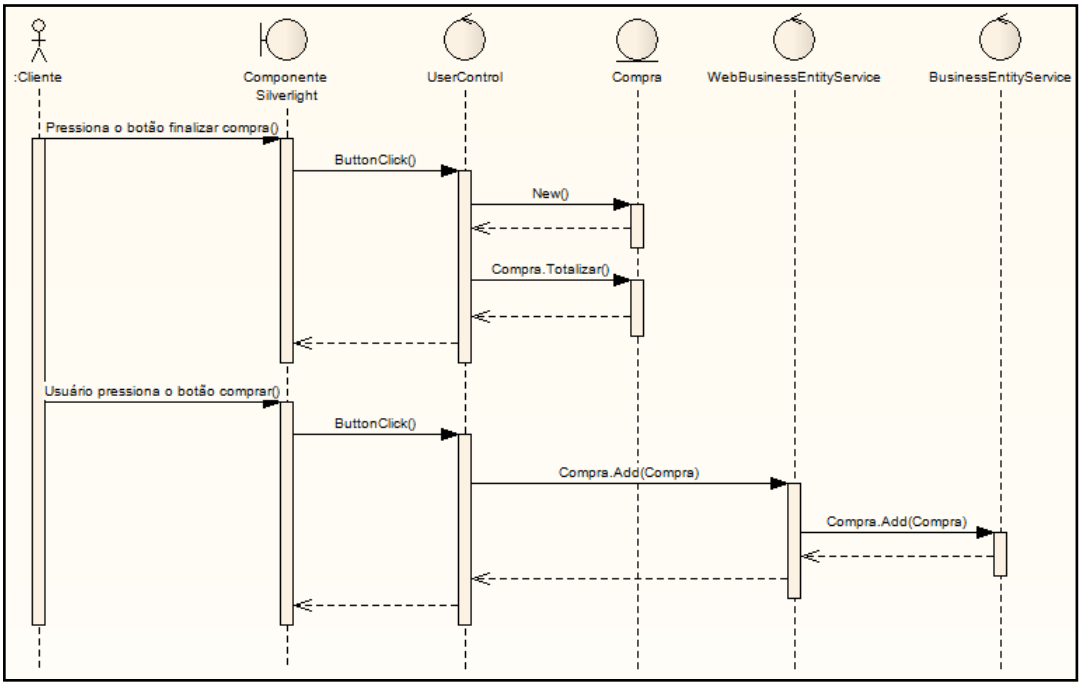


Figura 20 – Diagrama de seqüência do caso de uso finalizar compra

3.3.3 Diagrama de seqüência do caso de uso cadastrar cliente

A figura 21 representa o diagrama de seqüência do caso de uso, nomeado cadastrar cliente, onde o usuário pressiona o botão meu cadastro sendo redirecionado a página de clientes, onde o mesmo preenche as informações necessárias e pressiona o botão salvar. O próximo passo é a validação das informações e o disparo do método adicionar cliente na camada de apresentação que por sua vez repassa a chamada à camada de negócio que persiste o objeto cliente.

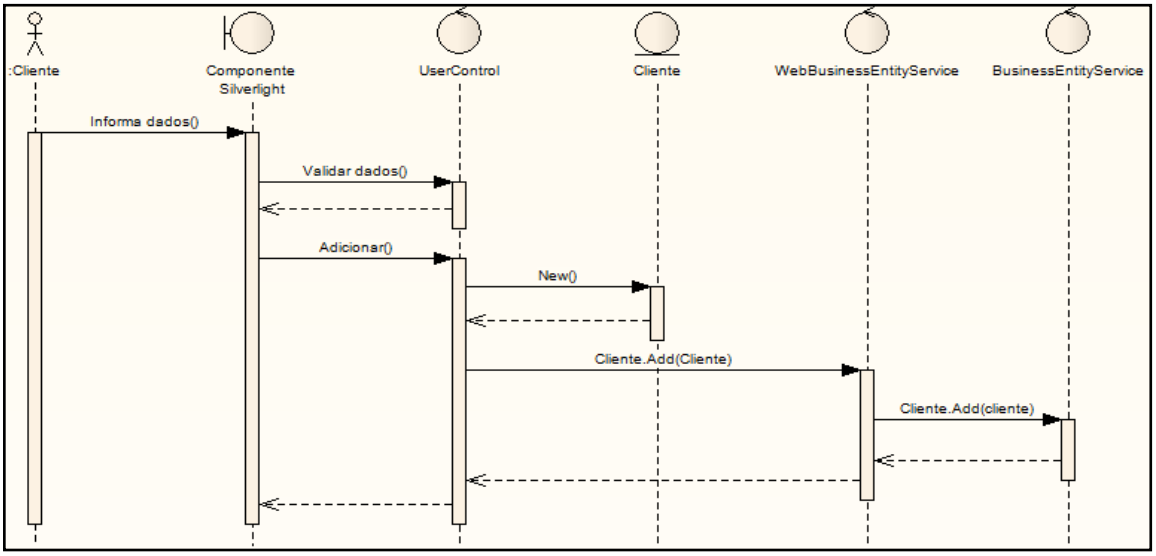


Figura 21 – Diagrama de seqüência do caso de uso cadastrar cliente

3.3.4 Diagrama de seqüência do caso de uso consultar produtos já cadastrados

A figura 22 ilustra o diagrama de seqüência do caso de uso consultar produtos já comprados, onde o usuário informa as suas credenciais e é encaminhado a uma página contendo uma listagem de todas as compras efetuadas pelo usuário. Através da obtenção da coleção de objetos compra via uma requisição ao serviço na camada de apresentação e despacha a chamada ao serviço na camada de negócio onde lê os dados em um modelo relacionais e retorna objetos fortemente tipados. O usuário por usa vez pode selecionar uma compra e visualizar todas as informações pertinentes a compra.

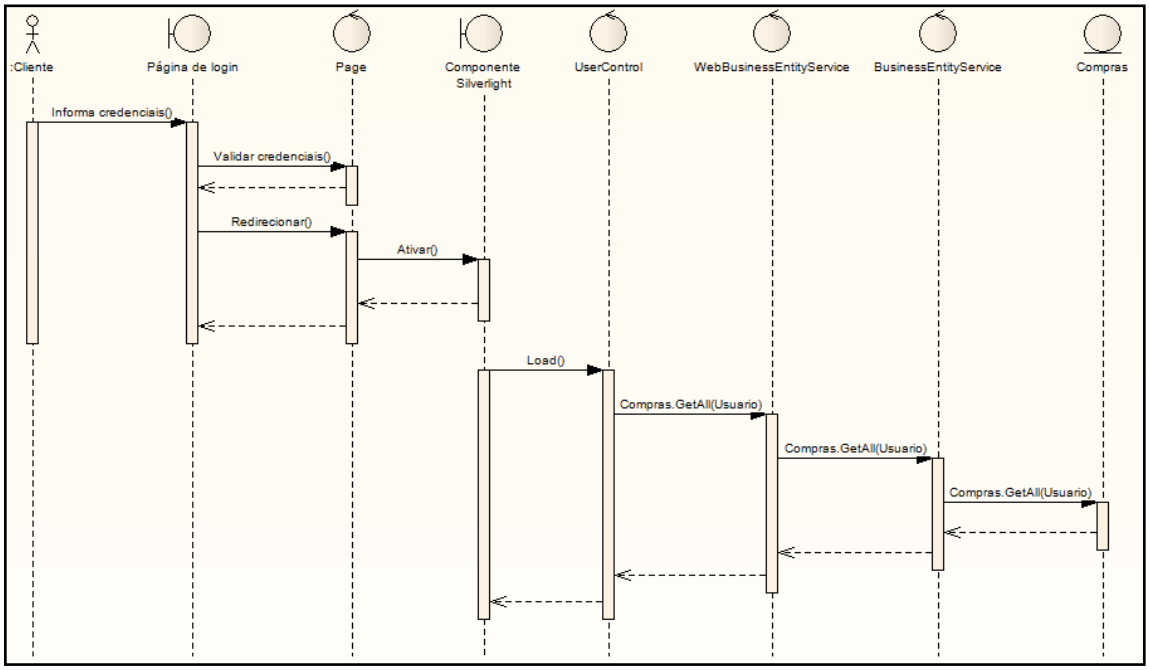


Figura 22 – Diagrama de seqüência do caso de uso consultar produtos já comprados

3.3.5 Diagrama de seqüência do caso de uso consultar produtos

A figura 23 ilustra o diagrama de seqüência do caso de uso, nomeado consultar produtos. Sendo que o usuário seleciona uma categoria através do menu superior ou na página principal, após selecionar o usuário é redirecionado a página da categoria onde o componente Silverlight irá ser instanciado. O componente ativado obterá os produtos da categoria via uma chamada remota ao `WebBusinessEntityService` que reside na camada de apresentação, chamando o *agent* do `BusinessEntityService` e este por sua vez irá obter todos os produtos da categoria selecionada via Entity Framework e LINQ.

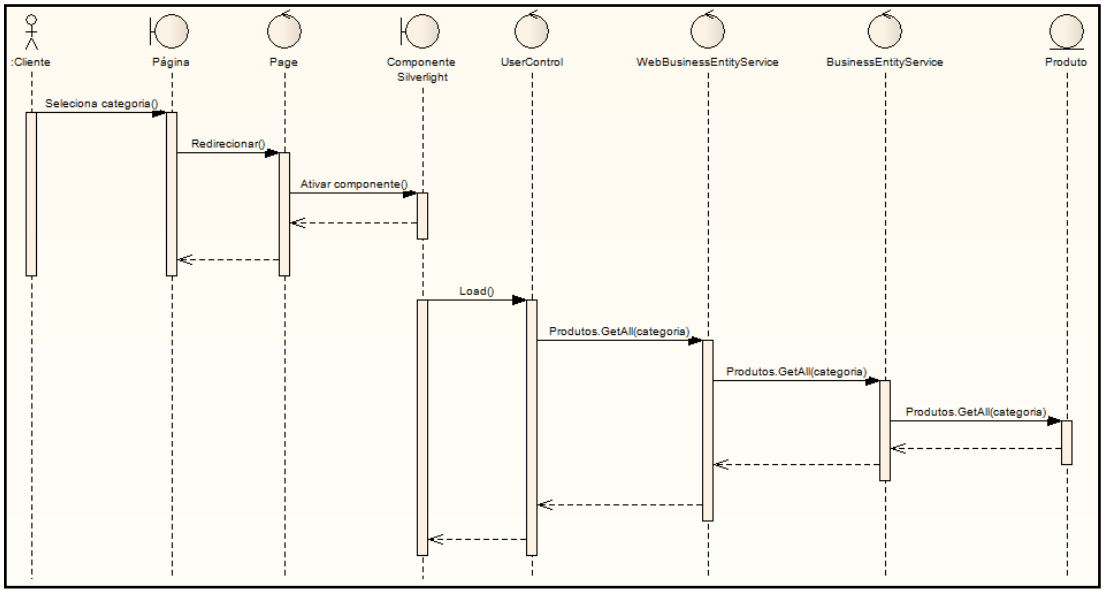


Figura 23 – Diagrama de seqüência do caso de uso consultar produto

3.3.6 Diagrama de seqüência do caso de uso manter consultas

Na figura 24 é representado o diagrama de seqüência do caso de uso manter consultas, sendo que o usuário acessa a página de consultas o componente Silverlight obtém todas as consultas do usuário e exibe na página em um componente de *grid*. Em seguida o usuário pressiona o botão incluir disparando a criação de um objeto `Consulta`, após a criação o objeto é exibido para que o usuário preencha as informações da consulta. O usuário por sua vez pode salvar a consulta encadeando a chamada do método `Salvar` do objeto `Consulta` que é propagado até a camada de negócio onde o Entity Framework faz a impedância entre o objeto e a base de dados relacional.

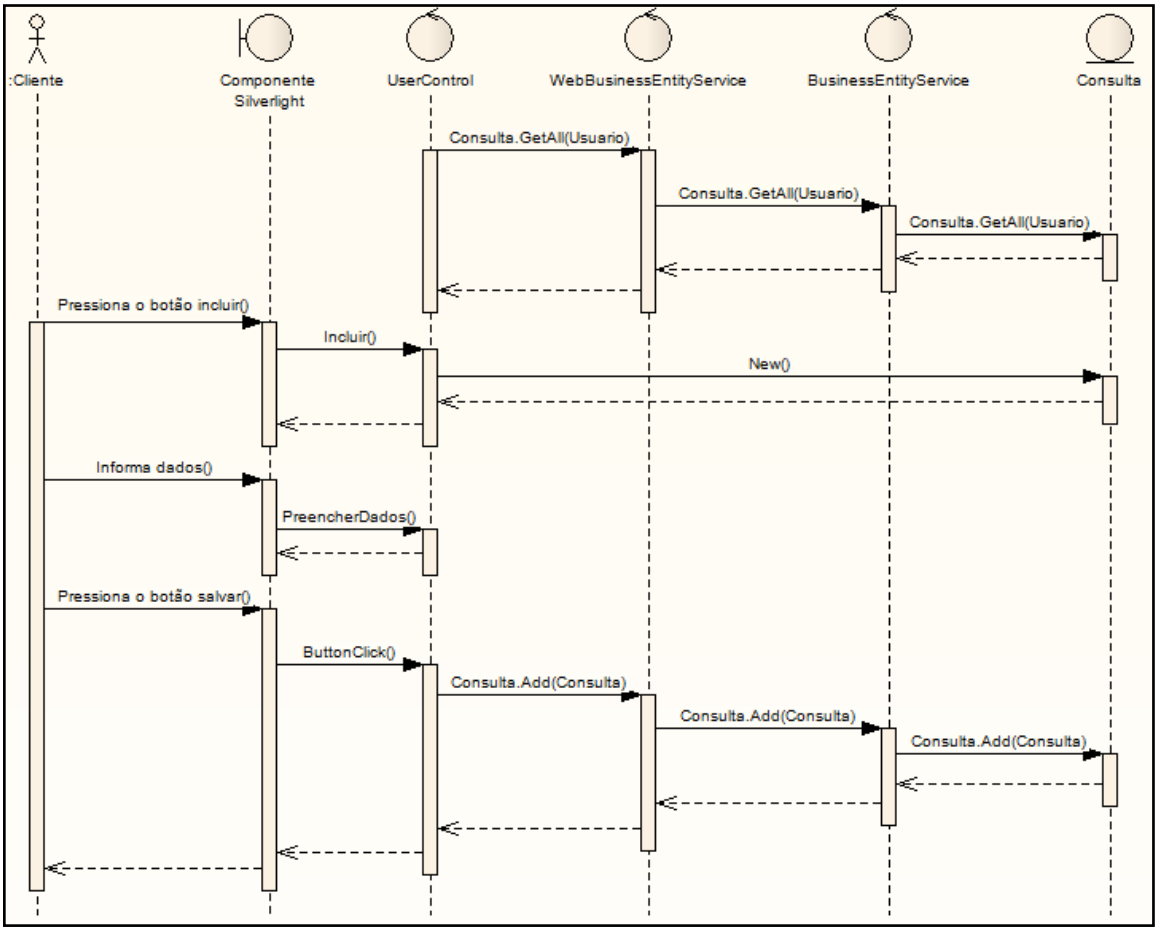


Figura 24 – Diagrama de seqüência manter consultas

3.3.7 Modelo entidade relacionamento

Quanto ao diagrama de modelo entidade relacionamento, a figura 25 apresenta as tabelas: produto, produtoCategoria, fornecedor, compra, compraProdutos, cliente, usuário e consulta.

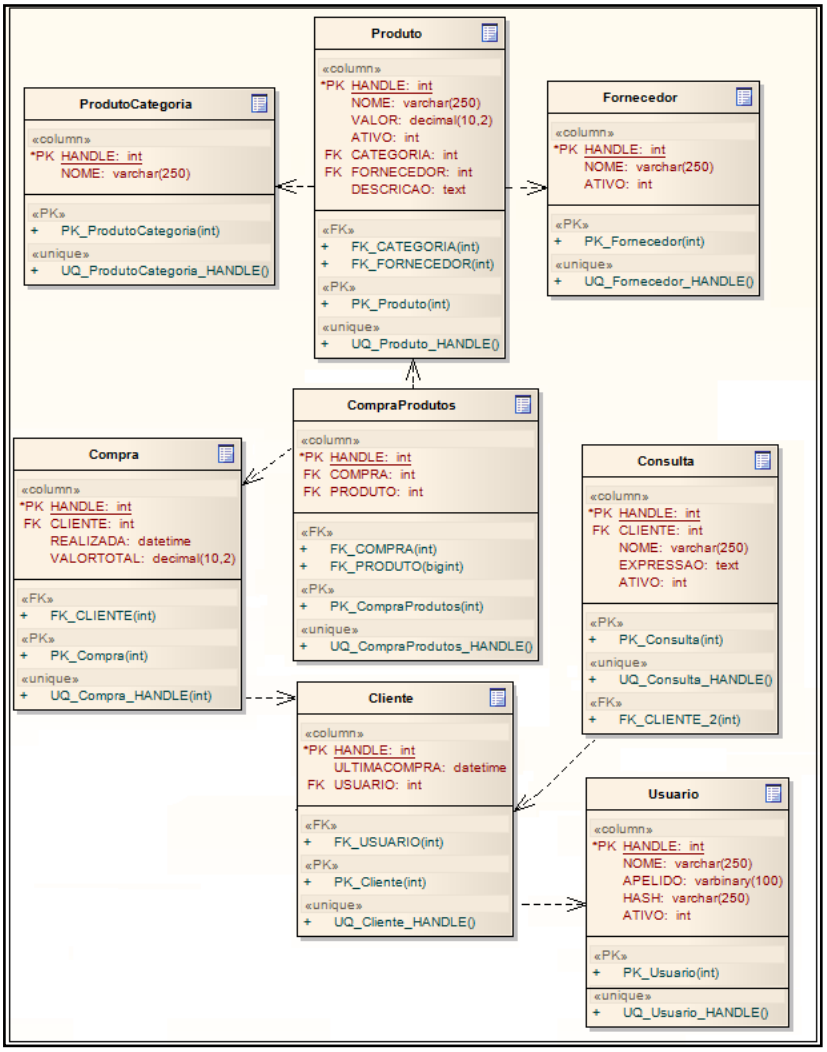


Figura 25 – Modelo entidade relacionamento

3.4 ESPECIFICAÇÃO DA ARQUITETURA

A especificação da arquitetura do aplicativo é apresentada inicialmente pelo diagrama de componentes da figura 26, que ilustra o emprego de uma arquitetura em camadas de alta coesão e baixo acoplamento. O navegador hospeda um componente Silverlight que por sua vez se comunica a um serviço WCF na camada de apresentação através de mensagens SOAP e XML. A segurança da transmissão se dá pelo uso de um certificado digital no servidor de aplicação WEB (SSL). A camada de apresentação é responsável pelo fornecimento das páginas ASP.NET, dos pacotes de aplicativos Silverlight ao navegador e também um serviço WCF que retransmite as chamadas do aplicativo Silverlight a um serviço na camada de negócio.

Quanto à camada de negócio, tem por responsabilidade prover um serviço WCF com acesso a base de dados através do modelo de persistência orientado a objetos denominado Entity Framework. A segurança na transmissão das informações envolvendo a camada de apresentação e a camada de negócio é fornecida por um conjunto de chaves assimétricas geradas a partir de certificados digitais, garantindo o não repúdio, a integridade e a confidencialidade das mensagens trafegadas sobre a rede. A separação física entre as camadas da arquitetura consiste no estabelecimento de fronteiras de confiabilidade onde o servidor de camada de apresentação não é confiável ao servidor de camada de negócio devido a estar exposto a internet.

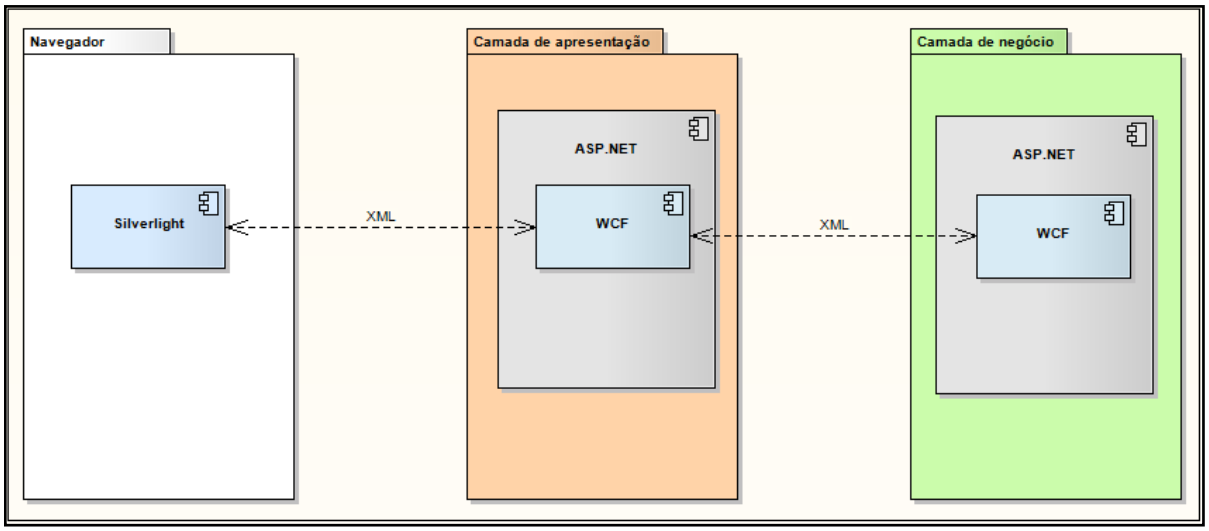


Figura 26 – Diagrama de componentes da arquitetura do aplicativo

A figura 27 ilustra o diagrama de componentes da arquitetura dos serviços, o processo de comunicação entre as camadas ocorre através de classes clientes e de serviços que implementam a mesma interface, essa sendo um contrato que é estabelecido entre os pares cliente e serviço. O componente Silverlight através da classe `WebBusinessEntityServiceAgent` se comunica com o serviço `WebBusinessEntityService` que está na camada de apresentação, o serviço `WebBusinessEntityService` utiliza a classe `BusinessEntityServiceAgent` para disparar métodos no serviço `BusinessEntityService` na camada de negócio. O `BusinessEntityService` por sua vez instancia a classe `OxyEntityServiceImplementation` onde reside a implementação dos métodos.

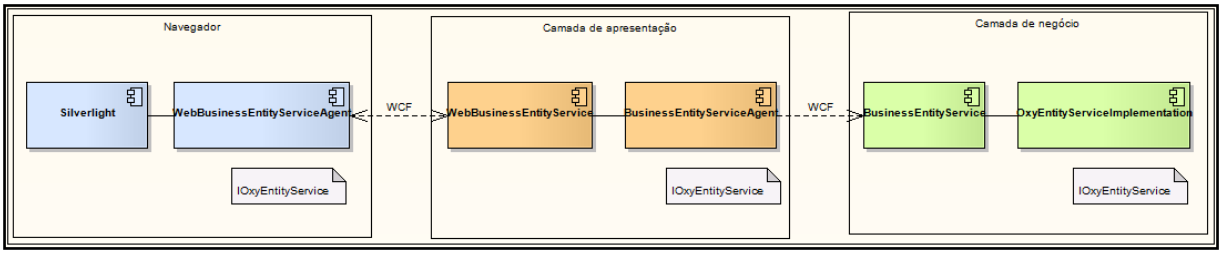


Figura 27 – Diagrama de componentes da arquitetura dos serviços

Na figura 28 está representado o diagrama de seqüência da arquitetura do aplicativo. O usuário requisita uma página WEB, em seguida o navegador solicita a camada de apresentação uma página. Ao receber o HTML o navegador instancia o componente Silverlight que requisita a camada de apresentação o pacote de aplicativo Silverlight. Após o recebimento do pacote de aplicativo é iniciada a execução do componente Silverlight que solicita ao serviço de dados na camada de apresentação uma coleção ou um objeto de negócio. A camada de negócio recebe a solicitação do serviço da camada de apresentação, trata os parâmetros da requisição e através de LINQ e Entity Framework obtém os objetos solicitados.

O Entity Framework por sua vez se comunica diretamente com a base de dados relacional e transforma *result sets* em objetos tipados. O serviço da camada de negócio retorna os objetos ao serviço da camada de apresentação que por sua vez retorna ao componente Silverlight. Onde os objetos são utilizados com fontes de dados para componentes visuais como formulários e grids, sendo estes de iteração com o usuário.

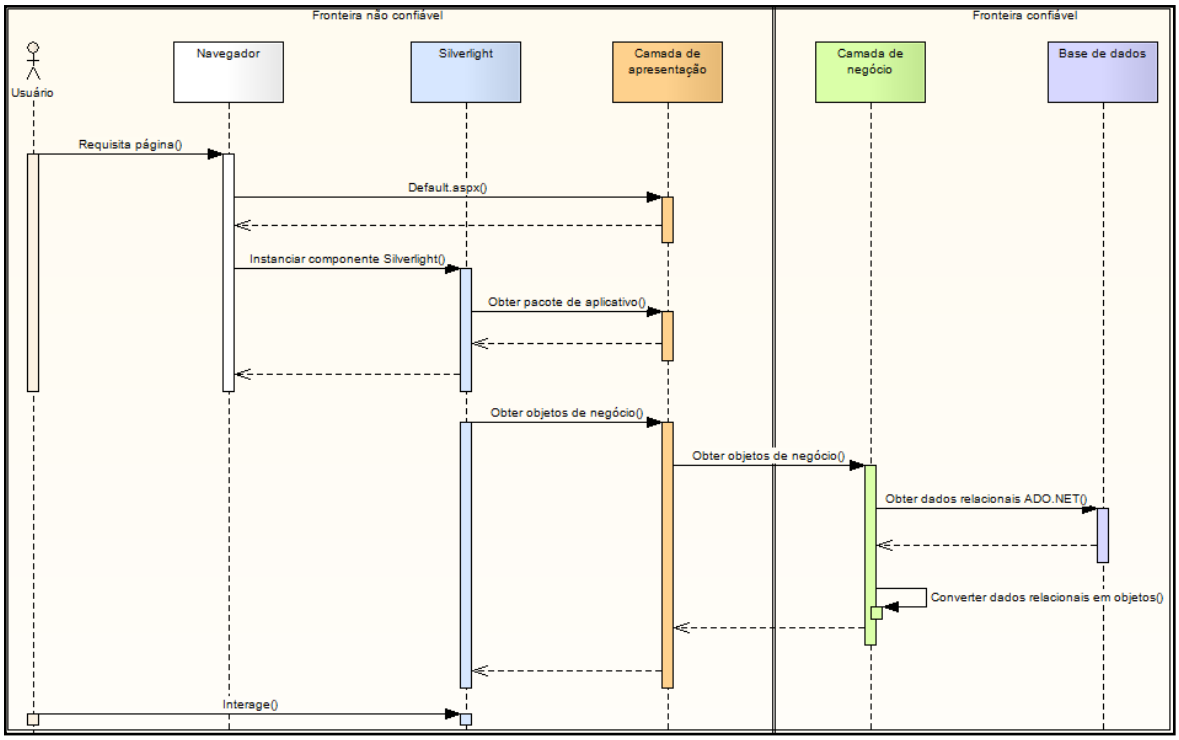


Figura 28 – Diagrama de seqüência da arquitetura do aplicativo

3.5 IMPLEMENTAÇÃO

Esta seção apresenta as técnicas e ferramentas utilizadas no desenvolvimento trabalho como: método de desenvolvimento e implementação, ambiente e linguagem de programação.

São apresentadas também a explanação, com partes do código fonte, das funcionalidades mais relevantes do sistema, como:

a) a instância de um objeto de negócio;
b) como um objeto de negócio é persistido;
c) como são trafegados os objetos de negócio entre as camadas do aplicativo;
d) a comunicação entre as camadas físicas da arquitetura;
e) o mecanismo de fonte de dados automática para os componentes visuais;
f) o mecanismo de autenticação.

3.5.1 Técnicas e ferramentas utilizadas

O processo de implementação deu-se ao término da especificação e a criação dos testes unitários de cada nova parte do aplicativo. A linguagem de programação escolhida para o desenvolvimento do trabalho foi a linguagem C#, em conjunto com o *.Net Framework 3.5*.

Quanto ao ambiente de desenvolvimento, foi escolhido o *Visual Studio 2008*, por ser uma ferramenta robusta e altamente produtiva e também pela ferramenta ter sido utilizada em trabalhos passados, criando assim um certo domínio sobre a ferramenta.

Para a persistência dos dados foi utilizado o *Entity Framework* que a partir de uma base de dados relacionais cria um conjunto de classes, gerando um modelo conceitual sobre o modelo relacional existente na base de dados. A figura 29 ilustra o modelo conceitual gerado pelo Entity Framework tendo como base o modelo relacional definido na base de dados.

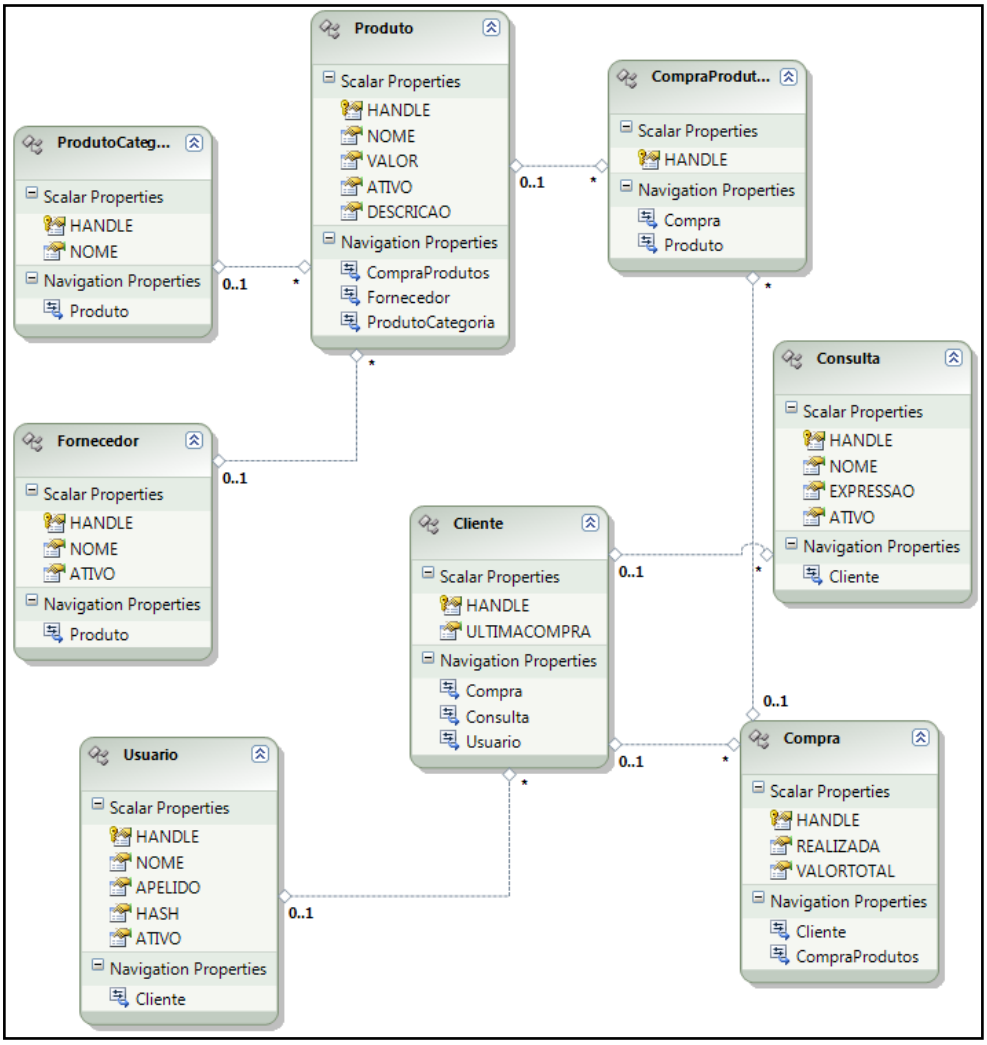


Figura 29 – Modelo conceitual

A camada de negócio manipula os objetos de domínio através do LINQ to *Entities*, que elimina a impedância entre a base de dados e o código fonte. Isto se resume a manipulação de objetos sem a necessidade da escrita de sentenças SQL.

```
public Produto GetProductByHandle(int handle)
{
    var stopwatch = TraceHelper.StartOperation("GetProductByHandle");
    try
    {
        if (handle == 0)
            throw new ArgumentNullException("handle");

        using (oxyEntities entities = new oxyEntities())
        {
            return entities.Produto.First(p => p.HANDLE == handle);
        }
    }
    catch (Exception exception)
    {
        TraceHelper.TraceError(exception);
        throw;
    }
    finally
    {
        TraceHelper.StopOperation("GetProductByHandle", stopwatch);
    }
}
```

Quadro 7 – LINQ to Entities

Ainda sobre o quadro 7, as seguintes considerações:

a) variável do tipo `oxyEntities` que contém todos os objetos de domínio e a lógica de persistência;
b) utilização de expressões Lambda no método `First` do objeto `ProdutoCategoria` para recuperar um objeto pelo nome;
c) `c` é um objeto do tipo `ProdutoCategoria` onde o nome deverá ser igual a variável `name` fornecida na assinatura do método `GetCategoryByName`.

No quadro 8 pode se notar uma pequena semelhança do LINQ para com a sintaxe do SQL, onde a palavra *from* indica a origem e a palavra *select* o que será retornado. A variável `result` irá receber todos os objetos de uma categoria.

```
public Produto[] GetProductsByCategory(string categoryName)
{
    var stopwatch = TraceHelper.StartOperation("GetProductsByCategory");
    try
    {
        if (string.IsNullOrEmpty(categoryName))
            throw new ArgumentNullException("categoryName");

        using (oxyEntities entities = new oxyEntities())
        {
            ProdutoCategoria categoria = entities.ProdutoCategoria.First
            (c => c.NOME == categoryName);

            var result = from products in entities.Produto
                         where products.ProdutoCategoria.HANDLE == categoria.HANDLE
                         select products;

            List<Produto> list = new List<Produto>();
            foreach (var produto in result)
            {
                produto.FornecedorReference.Load();
                produto.ProdutoCategoriaReference.Load();
                list.Add(produto);
            }

            return list.ToArray();
        }
    }
    catch (Exception exception)
    {
        TraceHelper.TraceError(exception);
        throw;
    }
    finally
    {
        TraceHelper.StopOperation("GetProductsByCategory", stopwatch);
    }
}
```

Quadro 8 – Obtenção de múltiplos objetos

O quadro 9 ilustra a persistência de um objeto de domínio, onde o objeto a ser persistido é passado como parâmetro no método que utiliza a chave única do objeto para recuperar o objeto da base de dados e sobrescrever a propriedade nome e em seguida é chamado o método `entities.SaveChanges()`. O método `SaveChanges` salva toda e

qualquer alteração em qualquer objeto de domínio dentro do escopo da declaração `using`.

```
public Produto SaveProduct(Produto product)
{
    var stopwatch = TraceHelper.StartOperation("SaveProduct");
    try
    {
        if (product == null)
            throw new ArgumentNullException("product");

        using (oxyEntities entities = new oxyEntities())
        {
            Produto produto = entities.Produto.First
            (p => p.HANDLE == product.HANDLE);
            if (produto != null)
            {
                produto.NOME = product.NOME;
                produto.ProdutoCategoria = product.ProdutoCategoria;
                produto.DESCRICAO = product.DESCRICAO;
                produto.Fornecedor = product.Fornecedor;
                produto.VALOR = product.VALOR;
                entities.SaveChanges();
                return produto;
            }
            return null;
        }
    }
    catch (Exception exception)
    {
        TraceHelper.TraceError(exception);
        throw;
    }
    finally
    {
        TraceHelper.StopOperation("SaveProduct", stopwatch);
    }
}
```

Quadro 9 – Persistência de um objeto de domínio

No quadro 10 é representado, a inclusão de um objeto de domínio a base de dados através do método `entities.AddToProduto` e após o método `SaveChanges`. O objeto passado como parâmetro no método `AddProduct` não possui chave única pois a mesma é auto gerada na base de dados no momento da persistência.

```
public void AddProduct(Produto product)
{
    var stopwatch = TraceHelper.StartOperation("AddProduct");
    try
    {
        if (product == null)
            throw new ArgumentNullException("product");

        using (oxyEntities entities = new oxyEntities())
        {
            entities.AddToProduto(product);
            entities.SaveChanges();
        }
    }
    catch (Exception exception)
    {
        TraceHelper.TraceError(exception);
        throw;
    }
    finally
    {
        TraceHelper.StopOperation("AddProduct", stopwatch);
    }
}
```

Quadro 10 – Inclusão de um objeto de domínio

Pode-se no quadro 11, verificar a exclusão de um objeto obtendo o objeto pela chave única, chamando o método `entities.DeleteObject(categoria)` e após o método `SaveChanges`.

```
public void DeleteProduct(Produto product)
{
    var stopwatch = TraceHelper.StartOperation("DeleteProduct");
    try
    {
        if (product == null)
            throw new ArgumentNullException("product");

        using (oxyEntities entities = new oxyEntities())
        {
            Produto produto = entities.Produto.First
            (p => p.HANDLE == product.HANDLE);
            entities.DeleteObject(product);
            entities.SaveChanges();
        }
    }
    catch (Exception exception)
    {
        TraceHelper.TraceError(exception);
        throw;
    }
    finally
    {
        TraceHelper.StopOperation("DeleteProduct", stopwatch);
    }
}
```

Quadro 11 – Exclusão de um objeto de domínio

No quadro 12 é ilustrada uma parte da interface que os clientes dos serviços e os serviços utilizam, isto para o WCF é chamado de `ServiceContract`, este por sua vez é responsável pelo normalização das operações entre cliente e serviço. A operação `AddCategory` utiliza o padrão de projeto *request response* onde os objetos trafegados estão dentros das respectivas classes de *request* e *response*.

```
[ServiceContract(Namespace = "http://Oxy.ApplicationServer.ServiceContracts/2009/05",
                 Name = "IOxyEntityService")]
public interface IProviderOxyEntityService
{
    [OperationContract(IsTerminating = false,
                       IsInitiating = true,
                       IsOneWay = false,
                       AsyncPattern = false,
                       Action = "AddCategory")]
    AddCategoryResponse AddCategory(AddCategoryRequest request);
}
```

Quadro 12 – Interface

O quadro 13 demonstra as classes de *request* e *response* para a operação do serviço `AddCategory`. Para que uma classe seja trafegada pelo WCF a mesma deve estar decorada com o atributo *DataContract* e suas propriedades decoradas com o atributo *DataMember*.

```
[DataContract(Namespace = "http://Oxy.ApplicationServer.DataContracts/2009/05", Name =
              "AddProductRequest")]
public class AddProductRequest : Request
{
    [DataMember]
    public Produto Produto { get; set; }
}

[DataContract(Namespace = "http://Oxy.ApplicationServer.DataContracts/2009/05", Name =
              "AddProductResponse")]
public class AddProductResponse : Response
{
}
```

Quadro 13 – O padrão *request* e *response*

No quadro 14 é ilustrado um parte do serviço `BusinessEntityService` que implementa a interface `IProviderOxyEntityService`.

```
[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single,
                 ConcurrencyMode = ConcurrencyMode.Multiple,
                 Name = "OxyEntityService",
                 Namespace = "http://Oxy.ApplicationServer.ServiceContracts/2009/05")]
public class OxyEntityService : IProviderOxyEntityService
{
    public AddProductResponse AddProduct(AddProductRequest request)
    {
        return CallOperation(request, InternalAddProduct) as AddProductResponse;
    }

    private Response InternalAddProduct(Request requestBase,
                OxyEntityServiceImplementation localService)
    {
        var request = requestBase as AddProductRequest;
        localService.AddProduct(request.Produto);
        var response = new AddProductResponse();
        return response;
    }
    protected Response CallOperation(Request request, Func<Request,
                  OxyEntityServiceImplementation, Response> action)
    {
        try
        {
            var localService = new OxyEntityServiceImplementation();
            return action(request, localService);

        }
        catch (FaultException)
        {
            throw;
        }
    }
}
```

Quadro 14 – BusinessEntityService

Os objetos de `Principal` e `Identity` na arquitetura .NET são responsáveis por armazenar as credenciais do usuário como o nome do usuário e o tipo de autenticação utilizado. Em um ambiente WEB estes objetos necessitam serem recriado a cada requisição a uma página ou serviço para que fiquem disponíveis aos mesmos. No quadro 15 se ilustra a modificação no arquivo `global.asax` para reconstituição das classes `Principal` e `Identity`.

```
protected void Application_AcquireRequestState(object sender, EventArgs e)
{
    System.Security.Principal.IPrincipal principal;
    try
    {
        principal = (System.Security.Principal.IPrincipal)HttpContext.Current.Session["OxyPrincipal"];
    }
    catch
    {
        principal = null;
    }

    if (principal != null)
    {
        System.Threading.Thread.CurrentPrincipal = principal;
        HttpContext.Current.User = principal;
    }
}
```

Quadro 15 – Global.asax

A camada de apresentação é composta de uma `master page`, uma página de conteúdo e um componente Silverlight para cada categoria de produto do aplicativo. No quadro 16, tem-se o corpo da `master page`, por sua vez todas as páginas do aplicativos herdam a `master page`. Os objetos `ContentPlaceHolder` serão utilizados pelas páginas filhas para adicionar conteúdo a página principal.

```
<table cellpadding="0" cellspacing="0" style="width: 100%; height:100%">
    <tr>
        <td style="background-color: #EEEEEE">

        <table style="width: 100%;">
            <tr>
                <td>Oxy</td><td style="text-align: right">
                <asp:LoginStatus ID="LoginStatus1" runat="server" /></td>
            </tr>
        </table>

        </td>
    </tr>
    <tr>
        <td style="border: 1px solid #A3A3A3;
            background-color: #D1D1D1; height:50px">
                <asp:ContentPlaceHolder id="head" runat="server">
                </asp:ContentPlaceHolder>
        </td>
    </tr>
    <tr>
        <td style="height:100%; width:100%">
            <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
            </asp:ContentPlaceHolder>
        </td>
    </tr>
    <tr>
        <td style="border: 1px solid #A3A3A3;
            background-color: #D1D1D1; height:50px"">
             
        </td>
    </tr>
    <tr>
        <td style="background-color: #EEEEEE; text-align: right;">
             
            Oxy todos os direitos reservados</td>
    </tr>
</table>
```

Quadro 16 – *Master Page*

O quadro 17 demonstra uma página descendente da `master page` que instancia um componente Silverlight no navegador de internet. A *tag* `asp:Silverlight` faz referência ao pacote de aplicativo Silverlight que será instanciado no navegador.

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Oxy.master"
AutoEventWireup="true"
CodeFile="Categoria.aspx.cs"
Inherits="BackOffice_Categoria" %>
<%@ Register Assembly="System.Web.Silverlight"
Namespace="System.Web.UI.SilverlightControls" TagPrefix="asp" %>

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
Categorias
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server">
        <asp:ScriptManager ID="ScriptManager1"
runat="server"></asp:ScriptManager>
        <div  style="height:100%;">
            <asp:Silverlight ID="Xaml1" runat="server"
            SplashScreenSource="~/SilverlightClientApp/Loader.xaml"

Source="~/SilverlightClientApp/Oxy.Presentation.BackOffice.Category.xap"
            MinimumVersion="2.0.31005.0" Width="100%" Height="600px" />
        </div>
</asp:Content>
```

Quadro 17 – Página Asp.Net com componente Silverlight

No quadro 18, vê-se a representação gráfica em formato XML do componente Silverlight, esta representação será renderizada em tempo de execução. O Silverlight aplica o mesmo padrão do Asp.Net no Silverlight.

```
<UserControl
xmlns:data="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
x:Class="Oxy.Presentation.BackOffice.Category.Page"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Loaded="UserControl_Loaded">
    <Grid x:Name="LayoutRoot" Background="White">
        <Grid.RowDefinitions>
            <RowDefinition Height="210" />
            <RowDefinition Height="200" />
        </Grid.RowDefinitions>
        <data:DataGrid x:Name="Grid1" Margin="20" Grid.Row="0" IsReadOnly="True"
AutoGenerateColumns="True"
         HorizontalAlignment="Left" VerticalAlignment="Top" SelectionMode="Single"
         SelectionChanged="Grid1_SelectionChanged">
        </data:DataGrid>
            <TextBlock  Margin="2" Width="300" Height="20">Handle:</TextBlock>
            <TextBox x:Name="txHandle" IsEnabled="False" Margin="2" Width="300"
             Height="28" BorderBrush="LightGray" Text="{Binding HANDLE, Mode=TwoWay}"></TextBox>
            <TextBlock Margin="2" Width="300" Height="20">Nome:</TextBlock>
            <TextBox x:Name="txName" Margin="2" Width="300" Height="28"
             BorderBrush="LightGray" Text="{Binding NOME, Mode=TwoWay}"></TextBox>
        </StackPanel>
    </Grid>
</UserControl>
```

Quadro 18 – Xaml

No quadro 19, tem-se o código fonte do componente Silverlight que irá exibir todas as categorias dos produtos. Visualiza-se também a utilização da classe `ThreadPool`, pois a obtenção de dados não pode ocorrer na `Thread` principal da interface para não ocorrer

congelamentos de interface, prejudicando a experiência do usuário. Por sua vez a execução da obtenção de dados será agendada até que a próxima `Thread` da *pool* esteja disponível. A obtenção de objetos se dá através de um provedor de LINQ especificamente desenvolvido para este aplicativo onde é possível usufruir dos benefícios do LINQ dentro do componente Silverlight.

Após o término da obtenção dos objetos os mesmos devem ser atribuídos ao componente de `grid`, mas isto só pode ocorre na `Thread` principal, pois somente ela pode manipular os objetos de interface. Sendo assim é utilizado uma expressão lambda para executar o código na `Thread` principal.

```
private void DataLoad()
{
    ThreadPool.QueueUserWorkItem(delegate
    {
        QueryableOxyData<ProdutoCategoria> entities = new QueryableOxyData<ProdutoCategoria>();

        var query = (from categoria in entities
        where categoria.NOME != string.Empty
        select categoria).Take(10);

        var categorias = query.ToList();

        UIThread.CurrentThread.Run(() =>
        {
            Grid1.ItemsSource = categorias;
         });
    });
}
```

Quadro 19 – Código fonte componente Silverlight

Devido a questões de segurança todos os `assemblies` do aplicativo foram assinados digitalmente pela mesma chave, prevenindo adulterações nos binários e garantindo que foram desenvolvidos pelo autor. Os `assemblies` pertencentes à solução também receberem informações de versão única através de um único arquivo, sendo que todos `assembly` adiciona como referencia a si próprio o arquivo de assinatura digital e o arquivo de versionamento. Pelo quadro 20 pode-se verificar o versionamento único dos assemblies gerados pela solução.

```
using System.Reflection;
[assembly : AssemblyVersion("2009.0.0.0")]
[assembly : AssemblyFileVersion("2009.0.0.0")]
[assembly : AssemblyCompany("Oxy Tecnology")]
[assembly : AssemblyCopyright("Copyright © 2008, André Luis Voltolini Sousa.
                              Todos os direitos reservados.")]
```

Quadro 20 – Número de versão único para todos os *assemblies*

3.5.2 Processo de implementação

Este tópico aborda o teste de operacionalidade da implementação, através de um estudo de caso que compreende a utilização do aplicativo do ponto de vista do usuário. Para tanto montou-se um ambiente sem separação física das camadas da arquitetura ou seja todas as camadas físicas estão sendo hospedadas pela mesma máquina. Tanto o aplicativo Asp.Net, a arquitetura e os componentes Silverlight foram compilados em versão *release* e sem símbolos de depuração, otimizando a performance. A camada de apresentação Asp.Net foi instalado sob o diretório \inetpub\wwwroot\Oxy, a camada de negócio em \inetpub\wwwroot\ApplicationServer e \inetpub\wwwroot\STS para o serviço de geração dos *tokens* de segurança. Dentro da pasta da camada de apresentação se encontra a pasta SilverlightClientApp onde residem os pacotes de aplicativos Silverlight que serão baixados pelo navegador do usuário e também o pasta ServiceHost onde está o `WebBusinessEntityService`. Os certificados digitais foram gerados localmente tendo como base um certificado de autoridade confiável da máquina como por exemplo um certificado do domínio da máquina, também foi adicionado permissões de segurança de leitura as chaves privadas dos certificados para o usuário que executa a pool do Asp.Net. A figura 30 ilustra a página inicial do aplicativo que será o ponto de entrada para o usuário.

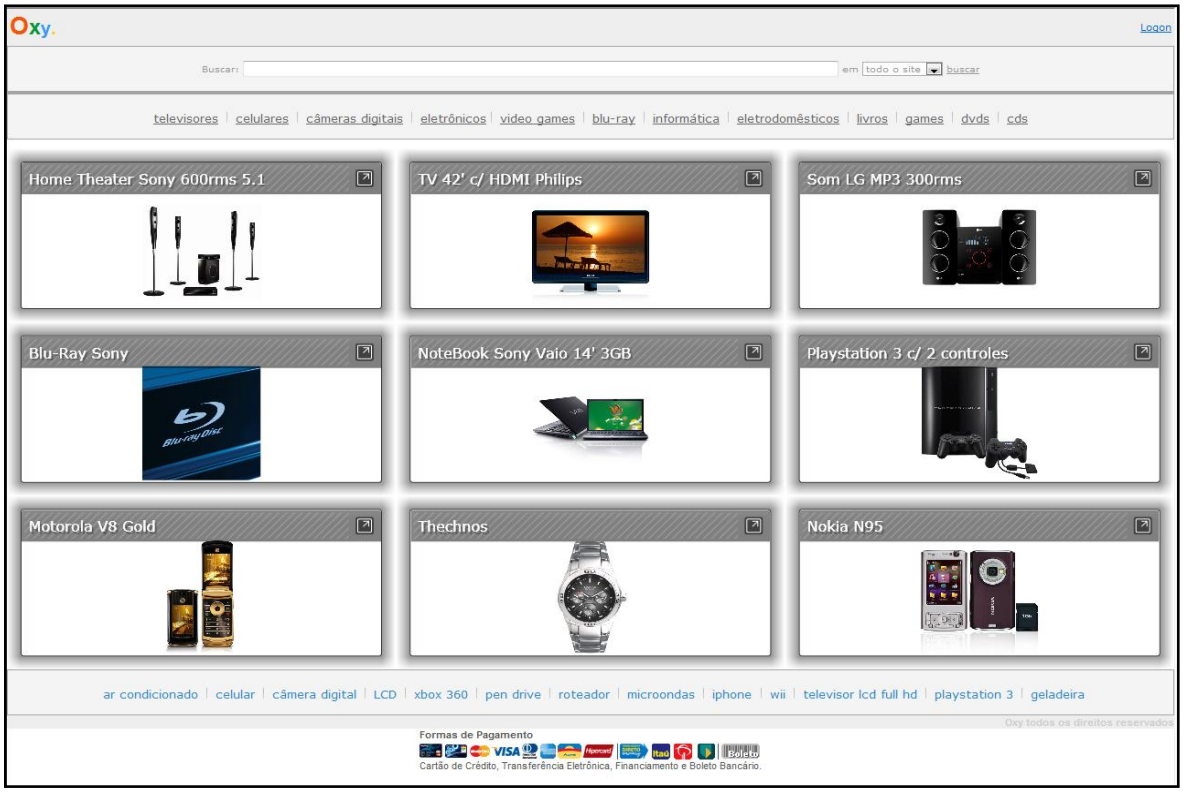


Figura 30 – Página inicial

A página inicial é acessível pelo endereço http://localhost/oxy/default.aspx, pode-se a partir deste momento navegar no site. Sendo a caixa de busca na parte superior da página onde o usuário poderá consultar qualquer informação do site, ou selecionar na caixa de opção ao lado uma categoria de produto para uma pesquisa mais focada. Logo abaixo a busca está localizado o menu principal onde estão as categorias de produtos ofertados pelo site. Ao clicar sobre o link o usuário é redirecionado a página principal da categoria. No centro da página estão os produtos em destaque, sendo um componente Silverlight permitindo a reordenação dos objetos, bem como a maximização para exibir maiores detalhes sobre o produto. A parte inferior da página possui links de rápido acesso aos produtos mais comprados. Na figura 31 tem-se a página da categoria de produtos televisores onde é exibido todos os produtos da categoria televisores paginados sendo um produto por linha.

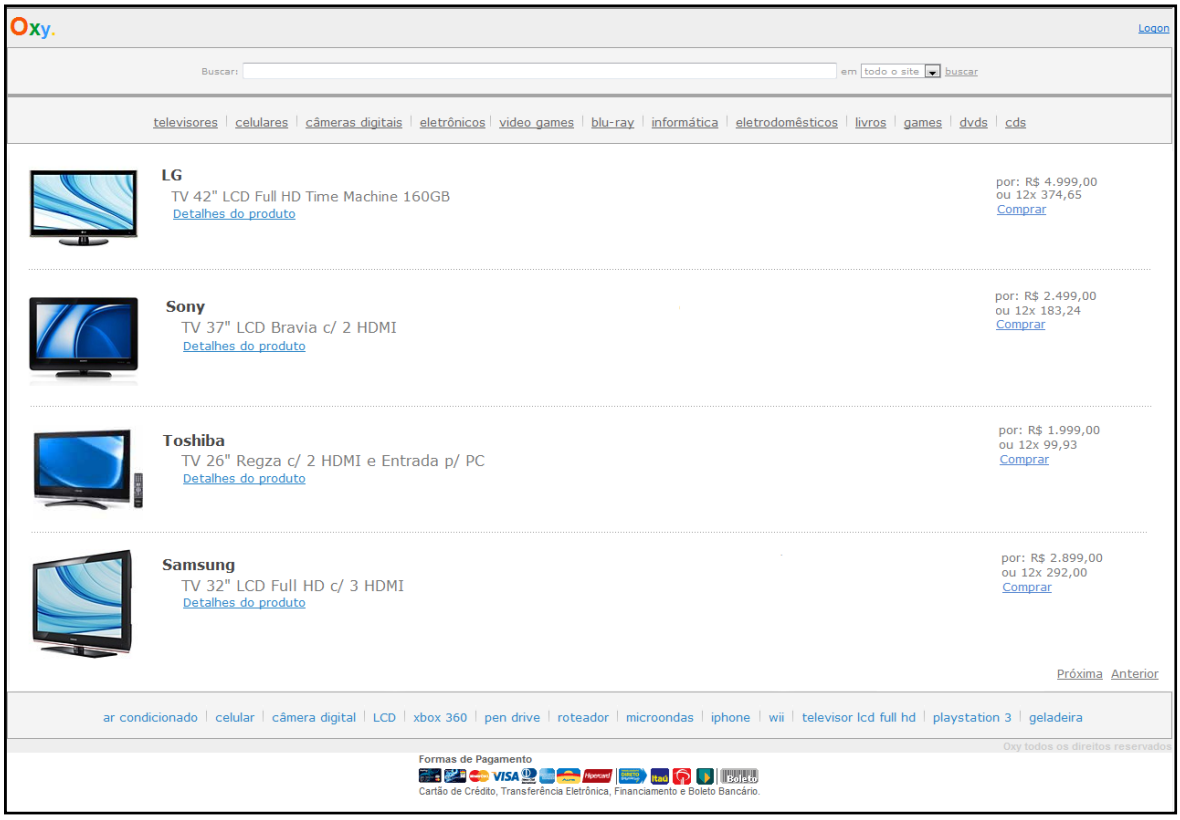


Figura 31 – Página da categoria de produtos televisores

O usuário ao clicar sobre uma produto irá ser redirecionado a página do produto conforme mostra a figura 32, podendo então pressionar o botão comprar que irá adicionar o produto a compra do usuário. A parte externa da página é HTML e o centro onde as informações são exibidas é um componente Silverlight sendo assim ao pressionar o botão comprar o componente adiciona o objeto `produto` a coleção `produtos` do objeto `compra`. O objeto `compra` está persistido na sessão Asp.Net do usuário no servidor de camada de apresentação.

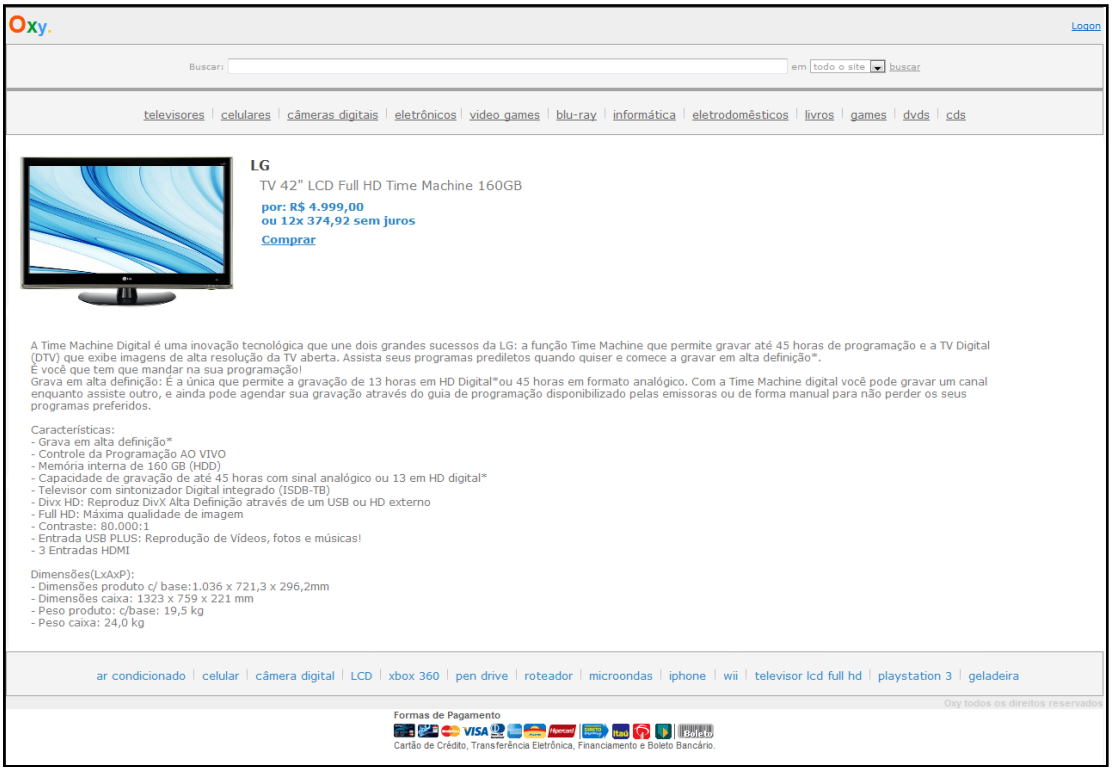


Figura 32 – Página do produto

Desta forma o usuário pode navegar no site e adicionar produtos a sua compra, após um produto ser adicionar a compra o link finalizar compra é exibido na parte superior direita da página principal. Ao clicar sobre o link finalizar compra o usuário é redirecionada para página de resumo da compra conforme figura 33.

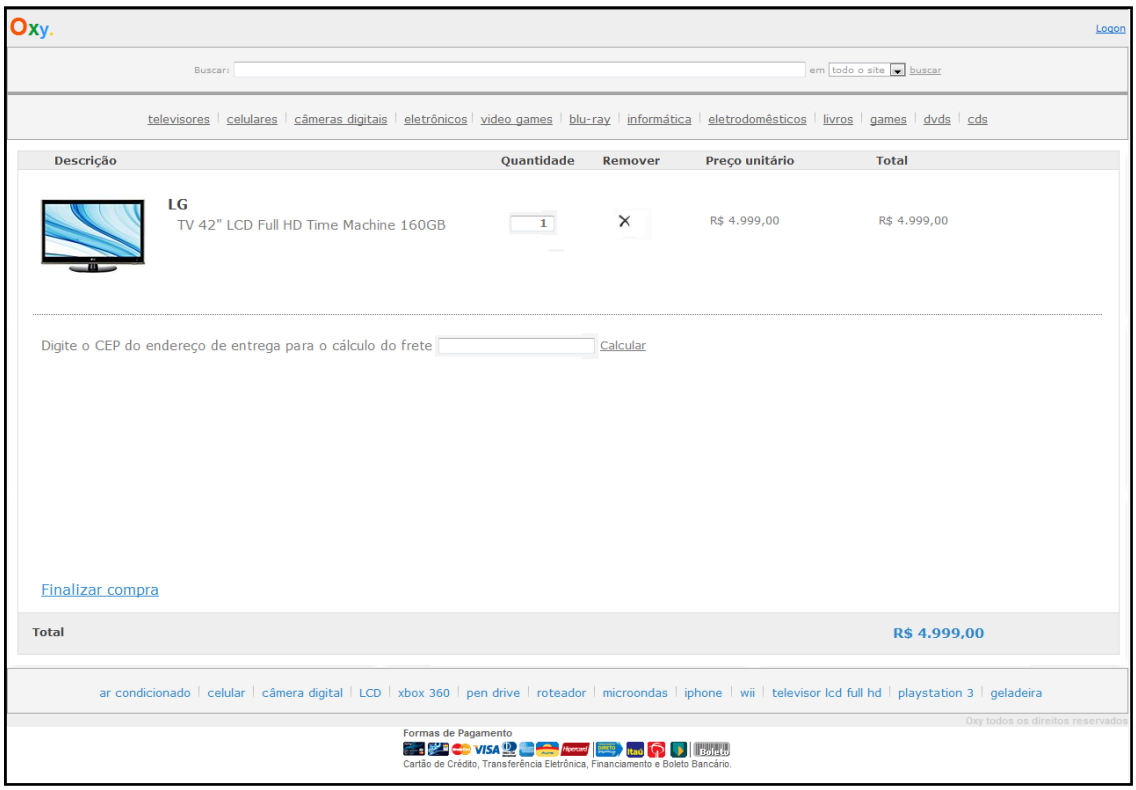


Figura 33 – Página de resumo da compra

Caso o usuário não tenha efetuado o *login* no site o mesmo será redirecionado a página de cadastro de usuários e *login* conforme mostra a figura 34.

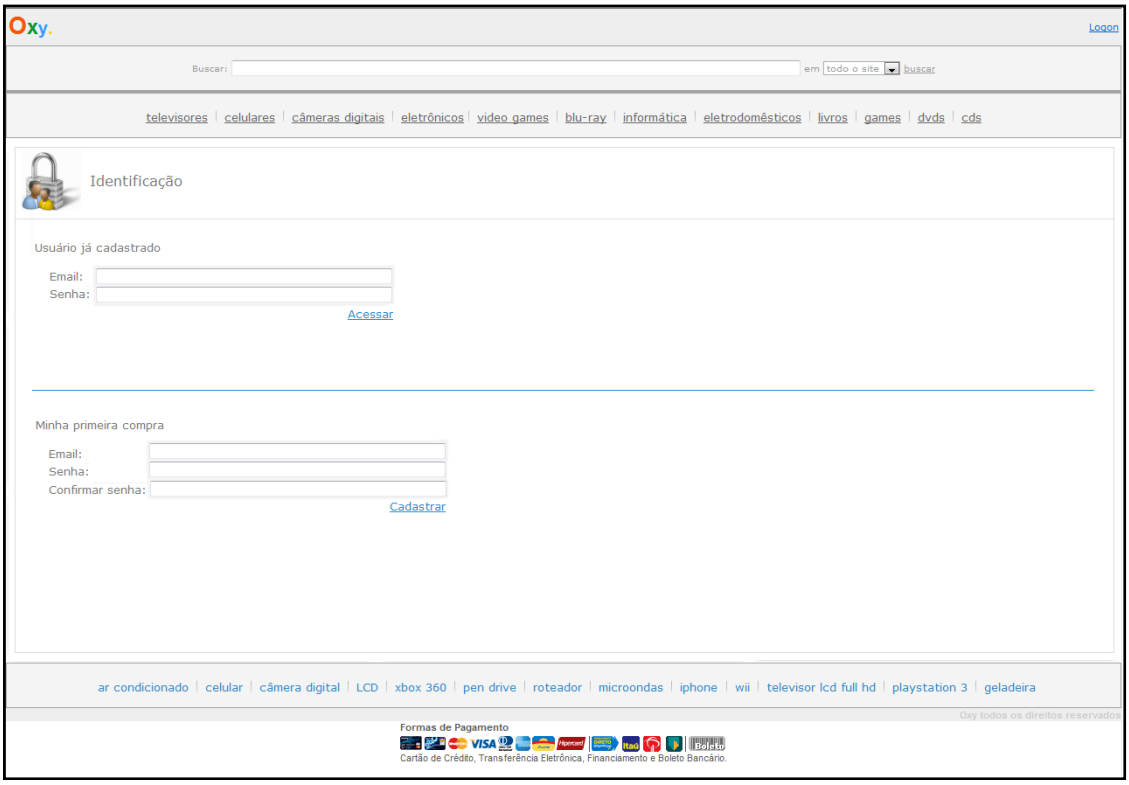


Figura 34 – Página de cadastro e *login*

Sendo usuário um cliente já cadastrado se faz necessário apenas informar a identificação do usuário (*login*) e sua respectiva senha, não possuindo um cadastro o usuário deverá preencher as informações solicitadas na página. Por sua vez após o *login* ou cadastro o usuário é redirecionado a página de resumo de compra.

Na página de resumo de compra pode-se visualizar os produtos adicionados a compra, o valor total da compra e o valor do frete. O componente permite a remoção de um produto da lista de compras, apenas marcando o produto desejado e pressionando o botão remover. Caso o usuário queira finalizar a compra é necessário informar o endereço de envio e a forma de pagamento.

Na figura 35 tem-se a relação de compras efetuadas pelo cliente, após o usuário efetuar o *login* no site o link minhas compras é exibido na parte superior direita da página principal. A página exibe todas as compras efetuadas pelo cliente, bem como o valor e os produtos de cada compra. Ao selecionar uma compra o usuário é redirecionado para a página de detalhes da compra onde são exibidos os produtos, o valor da compra, o valor do frete e o endereço de envio.

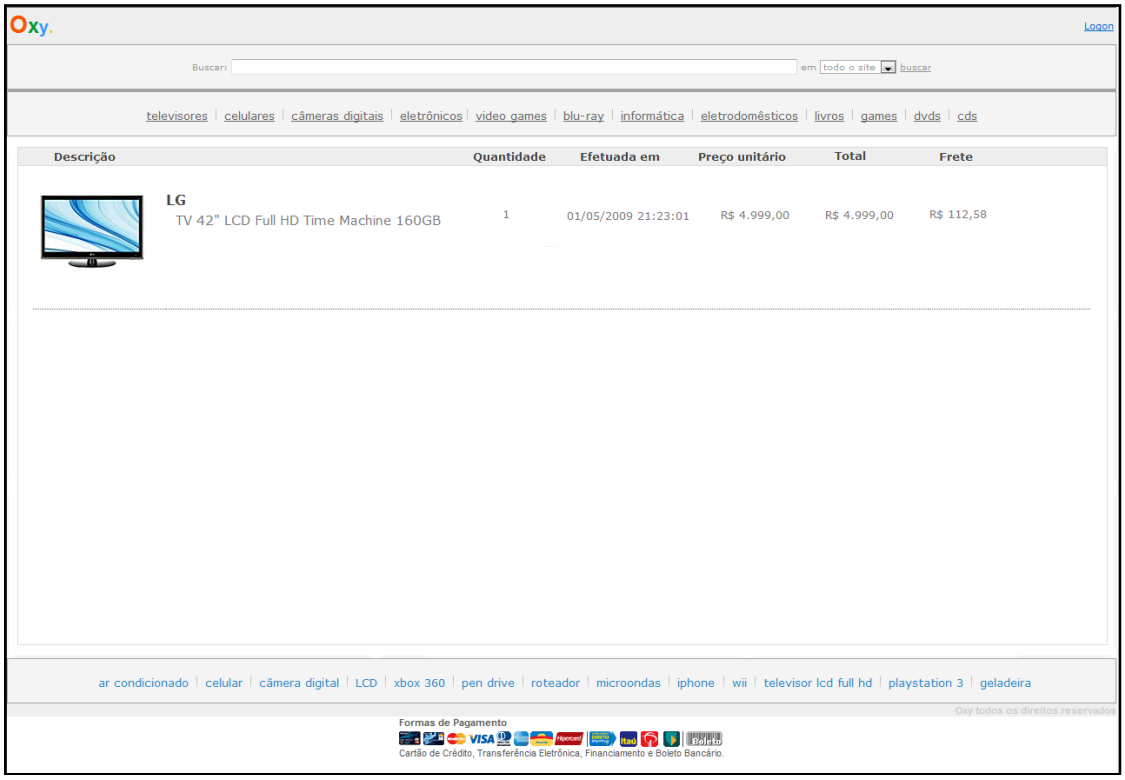


Figura 35 – Página minhas compras

A figura 36 refere-se aos cadastros de *back office* como por exemplo o de categorias de produtos. O acesso ao back office é somente permitido a usuário administrativos e autenticados.

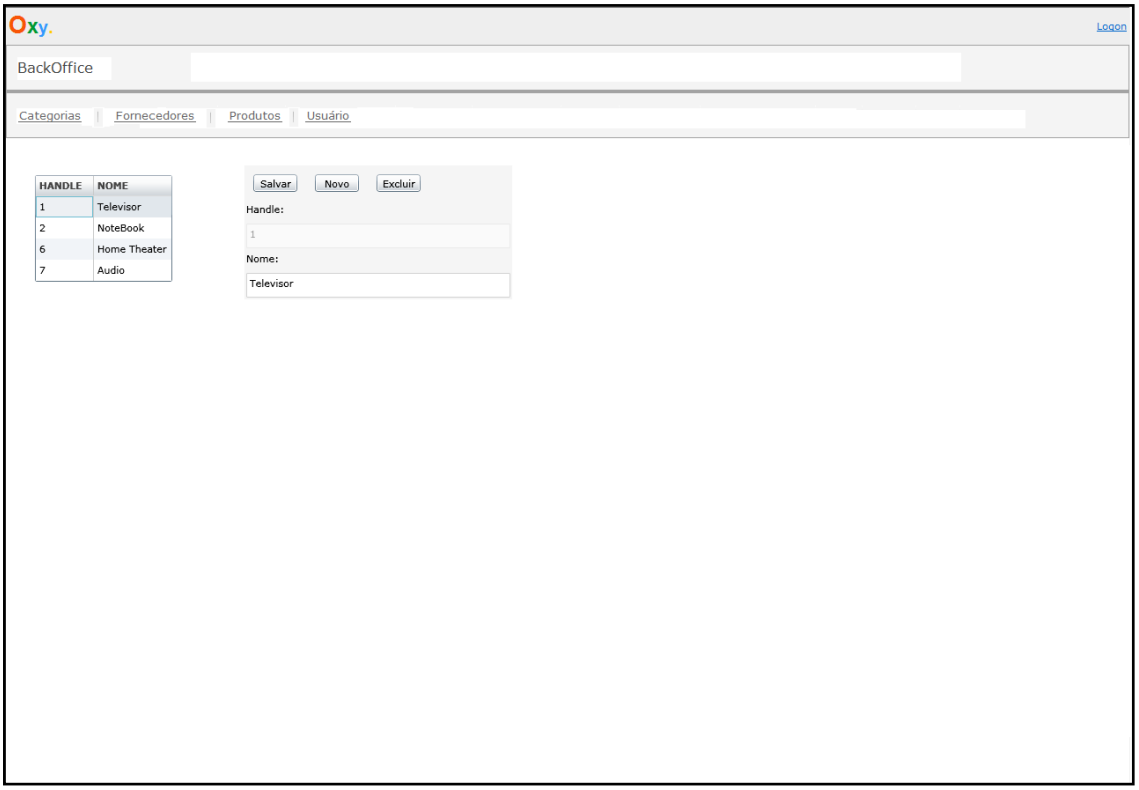


Figura 36 – *Back office*

3.6 RESULTADOS E DISCUSSÃO

A arquitetura desenvolvida neste trabalho atendeu as expectativas de escalabilidade devido à separação das camadas físicas da arquitetura em serviços possibilitando a distribuição em múltiplos servidores. Com isto possibilitando um crescimento conforme a demanda, sendo assim permitindo que o aplicativo possa alocar mais recursos em momentos de maior necessidade e menos recursos em períodos de ociosidade, isto se reflete na preservação de investimento quanto a equipamentos de infra-estrutura e servidores.

O padrão STS é comprovado por instituições renomadas com Microsoft e o consórcio OASIS que garante a segurança das informações trafegadas entre camada de apresentação e camada de negócio. A arquitetura possui um baixo acoplamento pelo fato da comunicação entre as camadas lógicas se dá através de um único ponto, sendo este configurável no arquivo de configuração de cada camada possibilitando a troca em tempo de execução. A alta coesão dos componentes da arquitetura se dá pelo uso de padrões de projeto que isolam o comportamento, pelo emprego de interfaces para a comunicação entre objetos de diferentes camadas lógicas e pelo uso de testes unitários. A comunicação entre as camadas de físicas da arquitetura ocorrem através de serviços e mensagens comprovando o emprego de uma arquitetura orientada a serviços.

O aplicativo de referência desenvolvido neste trabalho atendeu as expectativas propostas e obteve bons resultados em todos os aspectos da arquitetura, como no caso da utilização do WCF para a comunicação entre as camadas físicas de arquitetura do aplicativo. A utilização do Silverlight para a interface com o usuário melhorando a experiência e tornando o uso do aplicativo mais agradável. Outro aspecto importante foi o emprego do Entity Framework pra solucionar a persistência dos dados e bem como o uso do LINQ na manipulação dos objetos de domínio. Verificou-se a agilidade na modelagem do aplicativo através do Iconix e a boa produtividade no desenvolvimento do aplicativo, constatando se a maturidade e confiabilidade das ferramentas e técnicas utilizadas.

4 CONCLUSÕES

Ao término do desenvolvimento do trabalho pode-se concluir que os resultados foram alcançados em relação aos objetivos previamente formulados. Foi implementado um aplicativo modelado no processo Iconix e uma arquitetura em várias camadas utilizando tecnologias Microsoft, padrões de projeto e uma arquitetura orientada a serviços.

Também como especificado nos objetivos do trabalho, foi implementado a camada de apresentação como um aplicativo rico para a internet utilizando o Silverlight 2.0.

Pode-se perceber durante o desenvolvimento do trabalho a preocupação com a legibilidade do código fonte bem como a documentação de classes e métodos, provendo uma fácil compreensão das técnicas e tecnologias empregadas. Verificou-se ainda a importância de uma arquitetura em várias camadas e de baixo acoplamento possibilitando o aumento de usuários atendidos sem a necessidade de modificações no código fonte, apenas agregando máquinas as camadas físicas da arquitetura.

Através dos resultados obtidos pode-se perceber a produtividade das tecnologias Microsoft devido ao pouco tempo despendido ao desenvolvimento, boas práticas e padrões de projeto fazendo um aplicativo coeso, de baixo acoplamento, com boa performance, de fácil escalabilidade conforme a separação das camadas físicas que podem serem distribuídas em múltiplos computadores e seguro através de chaves assimétricas, assinaturas digitais e criptografia.

Este trabalho poderá ser utilizado em disciplinas acadêmicas como material de estudo das tecnologias, técnicas e ferramentas empregadas. Provendo um vasto conteúdo teórico e prático de como um aplicativo é desenvolvido utilizando tecnologias Microsoft.

4.1 EXTENSÕES

De acordo com os conhecimentos adquiridos durante o desenvolvimento do trabalho tornou-se possível à análise de diversas possibilidades que podem ser sugeridas como trabalhos futuros, sendo elas:

a) adaptar a interface para utilizar apenas *Silverlight*, ou seja não ter uma página

HTML como hospedeiro de um componente *Silverlight*;

b) a comunicação entre os componente Silverlight e a camada de apresentação através do formato de mensagens JSON pela diminuição das quantidade de informações trafegadas;
c) testar o aplicativo em um ambiente de FARM em todas as camadas físicas do aplicativo.

4.2 LIMITAÇÕES

Ao decorrer do desenvolvimento do trabalho tornou-se possível à análise limitações, sendo elas:

a) a tecnologia utilizada para remover a impedância entre objetos e base de dados o Entity Framework somente opera em banco de dados Microsoft SQL Server e Oracle;
b) a arquitetura desenvolvida não é aconselhável para projetos de pequeno porte devido a separação em camadas físicas requerendo vários servidores e pela quantidade de configurações.

REFERÊNCIAS BIBLIOGRÁFICAS

DOMINGOS, Marcelo. **Uma arquitetura de referência para sistemas de informação e portais de serviços de governo eletrônico**. 2004. 129 f. Dissertação(Mestrado em engenharia de produção) – Universidade Federal de Santa Catarina, Florianópolis.

FOWLER, Martin. **Patterns of enterprise application architecture**. Boston: Addison Wesley, 2005.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Padrões de projeto**. Porto Alegre: Bookman, 2004.

HEJLSBERG, Anders. **.Net language integrated query.** [S.l.], 2005. Disponível em: <http://msdn.microsoft.com/pt-br/library/bb308959.aspx>. Acesso em: 20 maio. 2009.

HOFSTADER, Joseph. **Usando padrões para definir uma solução de software.** [S.l.], 2007. Disponível em: < http://msdn.microsoft.com/pt-br/library/bb190165.aspx>. Acesso em: 09 jul. 2009.

JEZIERSKI, Edward. **Application architecture for .Net**: designing application and services. [S.l.], 2002. Disponível em:<http://msdn2.microsoft.com/en-us/library/ms954595.aspx>. Acesso em: 5 set. 2007.

KALANI, Amit. **Development and implementation web applications with visual c# and visual studio .net**. Indianapolis: Que Certification, 2002.

MACDONALD, Matthew. **Pro silverlight 2 in c# 2008**. New York: Apress, 2008.

MAGELA, Rogério. **Engenharia de software aplicada**: princípios. Rio de Janeiro: Alta Books, 2007a.

_____. **Engenharia de software aplicada**: fundamentos. Rio de Janeiro: Alta Books, 2007b.

MICROSOFT. **Csharp 3.0 specification.** [S.l.], 2005. Disponível em: <http://msdn.microsoft.com/pt-br/library/ms364047(en-us,vs.80).aspx>. Acesso em: 15 maio. 2009.

_____. **EDRA**: Enterprise Development Reference Architecture. [S.l.], 2004. Disponível em: **<**http://support.microsoft.com/?scid=kb%3Ben-us%3B872836&x=11&y=13>. Acesso em: 10 ago. 2007.

_____. **Enterprise library.** [S.l.], 2008. Disponível em: <http://msdn.microsoft.com/en-us/library/cc467894.aspx>. Acesso em: 3 mar. 2009.

_____. **Entity Framework.** [S.l.], 2006. Disponível em: <http://msdn.microsoft.com/en-us/library/aa697427(VS.80).aspx>. Acesso em: 17 maio. 2009.

_____. **Web service security 3.0.** [S.l.], 2003. Disponível em: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wse3.0/html/d0ed7f06-504b-40f8-939c-b884ffce77c0.asp>. Acesso em: 11 maio. 2009.

PIJANOWSKI, Keith. **Visibilidade e controle em arquitetura orientada a serviços**. [S.l.], 2007. Disponível em: < http://msdn.microsoft.com/pt-br/library/bb507204.aspx>. Acesso em: 10 jul. 2009.

RESNICK, Steve; CRANE, Richard; BOWEN, Chris. **Essential windows communication foundation**. San Francisco: Addison Wesley, 2007.

ROCKFORD, Lhotka. **CSLA.NET.** [S.l.], 2007. Disponível em: <http://www.lhotka.net/cslanet/Default.aspx>. Acesso em: 30 jul. 2007.

SCOTT, Rosen. **Introduction to the iconix process of software modeling.** [S.l.], 2006. Disponível em: < http://www.informit.com/articles/article.asp?p=167902&rl=1>. Acesso em: 09 jul. 2009.

SPROTT, David . **Service oriented architecture.** [S.l.], 2004. Disponível em: <http://msdn2.microsoft.com/en-us/architecture/aa480021.aspx>. Acesso em: 12 ago. 2007.

VIDOTTI, Cleber Aparecido. **Metodologia simplificada de desenvolvimento de software para empresas de pequeno e médio porte: uma aplicação prática na wopm informática**. 2003. 106 f. Dissertação(Mestrado em engenharia de produção) – Universidade Federal de Santa Catarina, Florianópolis.