UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

SOFTWARE DE CONTROLE DE ENTREGAS USANDO DISPOSITIVOS MÓVEIS E WEB SERVICE SOBRE A PLATAFORMA .NET

IVAN CARLOS JUNGES

BLUMENAU
2006

2006/2-16

IVAN CARLOS JUNGES

SOFTWARE DE CONTROLE DE ENTREGAS USANDO DISPOSITIVOS MOVEIS E WEB SERVICE SOBRE A PLATAFORMA .NET

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. Francisco Adell Péricas , Msc – Orientador

BLUMENAU
2006

2006/2-16

SOFTWARE DE CONTROLE DE ENTREGAS USANDO DISPOSITIVOS MOVEIS E WEB SERVICE SOBRE A PLATAFORMA .NET

Por

IVAN CARLOS JUNGES

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

Presidente:	Prof. Francisco Adell Pericas, Msc – Orientador, FURB
Membro:	Prof. Oscar Dalfovo, Dr – FURB
Membro:	Prof. Sergio Stringari, Msc – FURB

Blumenau, 01 de Dezembro de 2006

AGRADECIMENTOS

A meus pais, por toda a força e apoio nos momentos mais difíceis.

A minha namorada, pela compreensão e por estar sempre forte ao meu lado.

Aos meus amigos e colegas que me acompanharam durante esta jornada na universidade e na batalha pela conclusão deste trabalho.

A todos os professores que de alguma forma contribuíram com a formação do meu conhecimento, em especial agradeço meu orientador, Francisco Adell Péricas, por ter acreditado na conclusão deste trabalho.

Não receie a adversidade: Lembre-se que os papagaios de papel sobem contra o vento e não a favor dele.

H.Nobie

Consultor do NASA Marshall Space Flight Center (EUA)

RESUMO

Este trabalho apresenta a especificação e desenvolvimento de um aplicativo para dispositivos móveis, um Web Service e um Web Site, baseados na plataforma .NET. A aplicação visa explorar a troca de dados e a integração entre estas tecnologias sobre uma mesma plataforma de desenvolvimento. Para isso foi escolhido um caso de uso: "Entrega de Encomendas", onde a integração destas tecnologias tem como objetivo auxiliar o trabalho em campo das empresas, assim como permitir que clientes utilizem de forma transparente o Web Service para consultar informações pertinentes a si. Este trabalho demonstra a tendência tecnológica dos dispositivos móveis e Web Services, assim como o crescimento da presença destas tecnologias em nosso cotidiano.

Palavras-chave: Dispositivos móveis. Plataforma .NET. Web service.

ABSTRACT

This work presents the specification and development of an application for mobile devices, a Web Service and a Web Site, based on the platform .NET. The application aims to explore the exchange of data and the integration among these technologies on a same development platform. For this, use case was chosen: "Orders Delivery", where the integration of these technologies has as objective to assist the work in field of the companies, as well as allowing that customers to use in a transparent way the Web Service to consult related information itself. This work demonstrates the technological trend of the mobile devices and Web Services, as well as the growth of the presence of these technologies in our daily.

Key-words: Mobile devices. Platform .NET. Web service.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

ASP – *Active Server Page*

BCC – Curso de Ciências da Computação – Bacharelado

cHTML – *Compact* HTML

CLR – *Common Language Runtime*

CLS – *Common Language Specification*

HTML – *HyperText Markup Language*

HTTP – *HyperText Transfer Protocol*

IIS – *Internet Information Service*

SMTP – *Simple Mail Transfer Protocol*

SOAP – *Simple Object Access Protocol*

TCP – *Transmission Control Protocol*

UDDI – *Universal Description, Discovery and Integration*

VS.NET – *Visual Studio* .NET

W3C – *World Wide Web Consortium*

WAE – *Web Application Extension*

WAP – *Wireless Application Protocol*

Windows CE – *Windows Compact Edition*

WML – *Wireless Markup Language*

WSDL – *Web Service Definition Language*

XML - *eXtensible Markup Language*

XSD – XML *Schema Definition*

SUMÁRIO

1 INTRODUÇÃO

A necessidade da mobilidade é visível em nosso cotidiano assim como seu crescimento. Cada vez mais são observadas empresas que procuram por soluções em aplicativos móveis e encontram um mercado carente da tecnologia.

Empresas buscam não só a facilidade e a flexibilidade do uso de dispositivos móveis em campo, como procuram disponibilizar serviços a seus clientes portadores de dispositivos móveis, como por exemplo, uma determinada consulta do cliente por seu celular com acesso a internet. Com o avanço dos celulares e outros dispositivos, cada vez mais haverá procura por empresas que disponibilizem algum tipo de serviço para seus clientes adeptos a tecnologia.

Além das empresas enfrentarem o problema da falta de aplicativos para dispositivos móveis e seus clientes necessitarem de serviços que permitam consultar informações usando seus celulares, existe ainda pouca exploração, por parte dos desenvolvedores, das tecnologias que permitem o desenvolvimento para tais dispositivos, como a plataforma .NET por exemplo.

Segundo Borges Jr. (2005, p. 21), a mobilidade vem crescendo muito, e com esse crescimento, a necessidade de integrar com aplicações corporativas existentes nas empresas e no mundo. Tornando assim não só uma necessidade mas também passando a ser a meta de diversas empresas.

Visando essa necessidade de mercado, que existe e deve ser mais explorada, surgiu a idéia de utilizar dispositivos móveis (*Pocket PC* ou *Pocket PC Fone Edition*) e Web Service, para criar uma aplicação que atenda a empresa na coleta de informações e o cliente na consulta de informações pertinentes a ele. Para que esse trabalho possa se concretizar foi escolhida uma área que se beneficie com estas tecnologias. A área escolhida é a área de entrega de encomendas. Desta forma, as informações coletadas em campo com o uso de um dispositivo móvel durante as entregas, poderão ser disponibilizadas em um Web Service para consulta. No instante em que as informações forem atualizadas no Web Service será possível consultá-las pelo solicitante da entrega a qualquer momento, a partir de um *desktop* ou de um celular com acesso a internet.

Tanto o aplicativo para o dispositivo móvel como o Web Service será desenvolvido utilizando uma tecnologia emergente de desenvolvimento: a plataforma .NET. A plataforma .NET possibilita desenvolver ambas as tecnologias com uma mesma linguagem, o que agiliza e facilita o desenvolvimento das soluções.

Segundo a Microsoft Corporation (2001, p.50), “Diferentemente das tecnologias dos componentes atuais, os Web Services não são acessados por meio de protocolos específicos de modelo baseado em objetos [...]. Em vez disso, os Web Services são acessados por meio de protocolos da web e formatos de dados onipresentes”. Com essa afirmação a respeito dos atuais componentes para acesso remoto à informação, é visto que o Web Service é o mais indicado para solucionar o problema de integração das informações cliente/servidor, tornando-se invisível às pessoas que consomem e/ou enviam informações a um Web Service.

O desenvolvimento de um software para controle de entrega de encomendas é um exemplo de aplicação para as tecnologias citadas e um exemplo de área que necessita evoluir para o uso de dispositivos móveis, abrindo possibilidades para outras áreas.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho foi desenvolver um aplicativo para dispositivos móveis e Web Service sobre a tecnologia .NET, usando como exemplo prático a entrega de encomendas.

Os objetivos específicos do trabalho são:

a) controlar as entregas de encomendas utilizando um dispositivo móvel com .NET *Compact Framework*, extinguindo assim o uso de papéis e a necessidade de re-digitar as informações;

b) desenvolver um Web Service para ser consultado pelo dispositivo móvel para receber e enviar informações sobre as encomendas a serem entregues;

c) desenvolver uma página WEB que consulte o Web Service e que possa ser acessada tanto por *desktops* como por celulares, pelos donos das entregas.

1.2 TRABALHOS CORRELATOS

Em Galvin (2004) é apresentado um software para auxiliar empresas no relacionamento com seus clientes. Desenvolvida na linguagem VB.NET sobre a plataforma .NET e utilizando banco de dados SQL Server 2000, a solução agiliza a negociação e tomada de decisões, onde representantes utilizam dispositivos móveis para coleta de informações em campo, e atualizam as informações utilizando Web Service.

Em Ramos (2004) é apresentada uma solução utilizando tecnologia .NET para envio de mensagens criptografadas para dispositivos móveis, possibilitando assim comunicação sigilosa e integração entre dispositivo móvel e Web Service. Foi utilizado para desenvolvimento a linguagem VB.NET e a ferramenta de desenvolvimento Microsoft Visual Studio.

Em Schaefer (2004) é apresentada uma solução de integração entre dispositivo móvel e um sistema de informação, coletando informações das atividades externas de uma empresa de transportes e disponibilizando para consulta e análise em uma base local. Foi desenvolvido na plataforma Java 2 *Micro Edition* (J2ME).

1.3 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está apresentada em capítulos, divididos em cinco principais temas.

O primeiro capítulo apresenta uma visão sobre dispositivos móveis e tecnologias disponíveis, abordando o posicionamento da Microsoft (fabricante da tecnologia .NET) nesta área.

O segundo capítulo aborda as particularidades da tecnologia usada para o desenvolvimento deste trabalho, a plataforma .NET, onde é especificada a tecnologia WEB utilizada para desenvolvimento sobre a plataforma .NET, o ASP.NET, suas facilidades e vantagens.

Web Service e seus conceitos são abordados no terceiro capítulo, assim como padrões e suas definições.

O quarto capítulo descreve todo o desenvolvimento do trabalho, incluindo sua

especificação, ferramentas utilizadas e metodologia.

E, finalizando, apresenta-se no capítulo cinco as conclusões que se chegou com o desenvolvimento deste trabalho e são descritas as finalizações.

2 DISPOSITIVOS MÓVEIS

"Mobilidade é o termo utilizado para identificar dispositivos que podem ser operados a distância ou sem fio. Dispositivos que podem ser desde um simples BIP, até os mais modernos *Pocket PC's*." (MOSIMANN NETTO, 2004).

Segundo Pekus (2004), os dispositivos móveis não são mais apenas assistentes pessoais ou agendas eletrônicas, mas sim computadores que podem ser levados à qualquer lugar com facilidade, criados para atender a necessidade de profissionais que necessitam de rapidez e acesso a informações corporativas.

Para Tolomelli (2005), um dos motivos dado ao crescimento da necessidade da mobilidade está ligado ao fato da internet estar se tornando algo imperceptível, pois hoje já é possível acessar internet por meio de dispositivos móveis como *Pocket PC's* e telefones celulares.

Grandes empresas como bancos, mercado financeiro, área médica, entre outras áreas, estão cada vez mais em busca de mobilidade. Mas isso ainda é privilégio de poucas empresas e usuários. Segundo Miranda (2005), para os usuários e profissionais do mercado, o acesso a conectividade ou acesso a informação apenas em casa ou no trabalho está se tornando cada vez mais insuficiente. Exatamente com o surgimento da necessidade de estar sempre on-line os celulares estão cada vez mais se tornando, além de meros telefones, verdadeiros computadores de mão.

Para aqueles que consomem grande parte do seu tempo trabalhando em campo, sem a possibilidade de estar sempre em frente a um *desktop*, surge a necessidade de ter em mãos um computador portátil, mais leve, fácil e versátil que um *notebook*. E é justamente nessa área que existe um forte crescimento de equipamentos compactos, multifuncionais e de uso genérico.

Para estes profissionais houve o surgimento de uma nova linha de dispositivos, conhecidos como *Palm Tops* ou *Pocket PC's*, sendo utilizados por usuários comuns ou empresas que necessitam disponibilizar aplicações e informações a seus funcionários que precisam automatizar seus processos ou coletar informações em tempo real (MIRANDA, 2005).

Para Mosimann Netto (2005), no surgimento destas tecnologias, as linguagens para desenvolvimento eram muito específicas. Aliado a isso, a má documentação das tecnologias até então existentes, faziam com que fosse quase que inviável e muito caro desenvolver

aplicativos móveis. Vendo o crescimento do mercado e visando a expansão que se aproximava, a Microsoft planejou e iniciou a criação de uma plataforma unificada para o desenvolvimento de aplicações.

Neste mercado de dispositivos móveis, a Microsoft optou por desenvolver um sistema operacional que é na verdade uma versão simplificada do Windows. Chamado Windows *Compact Edition* (Windows CE), ele foi criado para ser embutido em equipamentos com baixo processamento ou capacidade de memória. Por isso, em parceria com diversas empresas, a Microsoft sugeriu a padronização de diversos componentes.

Hoje é possível encontrar o sistema operacional Windows CE em uma ampla gama de dispositivos, mantendo assim a interface atraente que o usuário já conhece do seu *descktop*, assim como os desenvolvedores têm à disposição não só a tecnologia .NET mas também o Microsoft SQL Server CE como opção de gerenciador banco de dados (PEKUS, 2004)

Na figura 1 pode ser observado em um *Pocket PC* a interface do Windows CE.

Fonte: Miranda (2005).
Figura 1 – *Pocket PC* com Windows CE

Com o crescimento em paralelo dos *Pocket PC's* e celulares, está surgindo um outro dispositivo móvel agora com o poder de processamento do *Pocket PC* aliado ao celular .

Para Miranda (2005), com o crescimento dos usuários em todo o mundo que começaram a ter em seus bolsos um *Pocket PC* e um aparelho celular, é natural que surja uma solução completa que contemple as duas necessidades. Visando sanar esta necessidade que surgiram os primeiros cartões e expansões com este objetivo.

Com visão neste mercado de integração entre PDA e telefone celular que a Microsoft lançou uma versão especial do sistema operacional para *Pocket PC*, integrando funções como

atender ou cancelar ligações, envio de mensagens de texto, entre outras funcionalidades de um celular.

É grande o número de pessoas que são usuárias de celular hoje em dia, o que torna o desenvolvimento de soluções para este público algo muito cobiçado e atraente.

Observando o que há de disponível nas tecnologias Microsoft, não é possível criar aplicativos que sejam executados diretamente no celular. Um dos motivos para tal é a diversidade de celulares, processadores e hardwares. Para suprir a necessidade deste público em potencial, a Microsoft disponibiliza uma tecnologia de desenvolvimento WEB para dispositivos móveis, bastando então o celular ter suporte a internet.

Visto como uma evolução natural dos telefones celulares hoje já é possível encontrar aparelhos celulares com maior capacidade de memória, processamento, recursos de expansibilidade, telas maiores e mais facilidades para acesso à internet. Estes dispositivos são chamados de *SmartPhone*. Para estes celulares já há uma versão do Windows CE, trazendo algumas características e funcionalidades do sistema operacional Microsoft (MIRANDA, 2005).

A figura 2 mostra um *SmartPhone* com Windows CE.

Fonte: Miranda (2005).
Figura 2 – *SmartPhone* com Windows CE.

3 PLATAFORMA .NET

A Microsoft iniciou seus empreendimentos em termos de desenvolvimento para a internet no ano de 1995. Durante muito tempo seu foco foi apenas a migração de sistemas operacionais *desktop* e de servidores para a tecnologia de 32 bits (BARWELL et al; 2004, p. 1).

Em julho do ano 2000, em uma conferência para desenvolvedores profissionais da Microsoft, foi apresentada a plataforma .NET. Por mais de dois anos em desenvolvimento a Microsoft quis revolucionar o desenvolvimento de aplicações Windows e WEB com o .NET (ALEXANDER; HOLLIS, 2002, p. 4).

A Microsoft criou o *Framework* .NET para tornar viável a interconexão de qualquer tipo de aplicação, dispositivo ou sistema, tendo como objetivo a simplicidade, extensibilidade, práticas e padrões baseados na web e padrões de modelos de programação (TARIFA; FACUNDE; GARCIA, 2005).

D'Angelo (2003) a define como um ambiente de desenvolvimento poderoso, que permite o desenvolvimento de aplicações *desktop*, aplicações para dispositivos móveis e desenvolvimento de aplicações WEB. Com a proposta de proporcionar um ambiente avançado de desenvolvimento, a Microsoft unificou todas as soluções de desenvolvimento nessa plataforma.

Segundo Barwell et al (2004, p. 1), o Microsoft .NET é a primeira plataforma de desenvolvimento a ser desenvolvida desde a estaca zero tendo a internet em mente, embora o .NET não seja exclusivo para o desenvolvimento na internet. Ao invés disto, a plataforma fornece um modelo consistente de desenvolvimento, podendo ser adotada em diversos tipos de aplicações.

Foi com a visão em uma nova classe de aplicativos que o .NET foi estruturado, visando o desenvolvimento de aplicativos Web Service e de troca de dados formatados em *eXtensible Markup Language* (XML). O acesso universal aos dados é um dos conceitos fundamentais da extratégia .NET (DEITEL et al, 2004, p. 13).

Muitos dos objetivos que a Microsoft tinha em mente ao desenvolver o .NET refletem as limitações que haviam em ferramentas e tecnologias anteriores. Entre os principais objetivos que a Microsoft procurava ao construir uma plataforma como o .NET destacam-se a facilidade em desenvolver aplicativos altamente distribuídos e melhorar a interface com o usuário via WEB, entre outros.

O .NET oferece suporte a diversas linguagens de programação como por exemplo: VB.NET, C#, J#, Smalltalk, Fortran, Cobol, entre outras linguagens. Basta que a linguagem siga as especificações da plataforma (CEMBRANELLI; 2003, p. 11).

Esta é uma facilidade nas situações onde um projeto exige que desenvolvedores com conhecimentos em linguagens diferentes possam migrar para uma plataforma robusta sem um grande custo de treinamento.

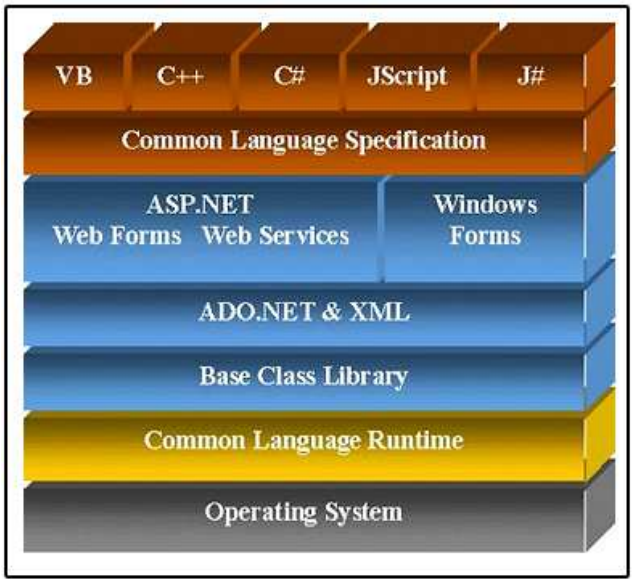
Além da independência de linguagem, Deitel et al (2004, p. 13) define a arquitetura .NET como independente de hardware, pois ela se estende a múltiplas plataformas, permitindo que programas desenvolvidos para *desktop* possam também ser executados em dispositivos móveis e até mesmo na internet. Basta que a plataforma .NET esteja instalada para qualquer aplicativo .NET executar sem qualquer problema com portabilidade.

3.1 .NET FRAMEWORK

O .NET *Framework* é a principal parte da plataforma, pois é aqui que todas as aplicações e serviços WEB são gerenciados. Todos os aplicativos desenvolvidos na plataforma .NET passam pelo .NET *Framework* antes de serem executados (DEITEL et al, 2004, p. 15).

A estrutura do .NET *Framework* cobre todas as camadas de desenvolvimento de software acima do sistema operacional, isolando assim o software desenvolvido com o .NET de particularidades do sistema operacional (BARWELL et al, 2004, p. 10).

A figura 3 mostra a estrutura do .NET *Framework*.



Fonte: Cembranelli (2003).

Figura 3 – Estrutura do .NET *Framework*

No primeiro bloco da estrutura estão as linguagens .NET e o *Common Language Specification* (CLS). O CLS é um conjunto de regras e normas as quais qualquer empresa deve seguir para criar uma linguagem .NET. No segundo bloco encontrasse o ASP.NET, tecnologia de programação para Web, e ao lado estão os *Windows Forms*, responsável pela interface de formulários. A *Base Class Library* contém todas as classes bases do .NET *Framework*. E o CLR é o administrador e responsável pela execução no .NET.

Segundo Cembranelli (2003, p. 11) de toda a estrutura do .NET *Framework* os dois componentes principais são: o CLR e a biblioteca de classes. Responsáveis por boa parte dos objetivos da plataforma.

3.1.1 Common Language Runtime

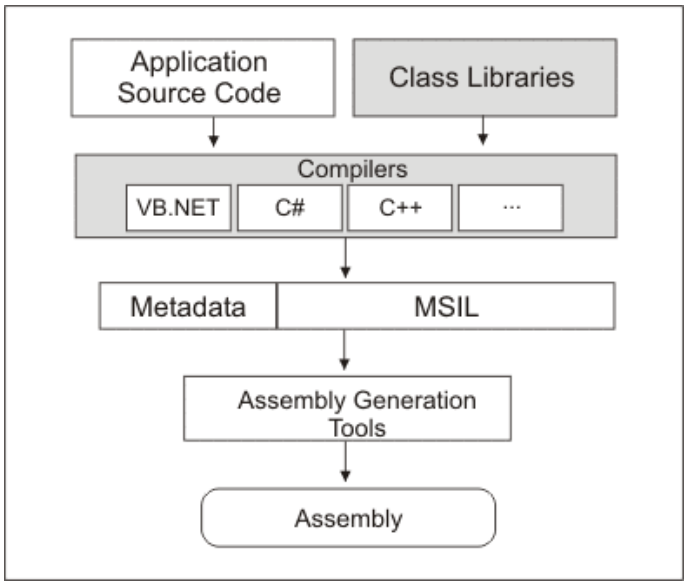
O *Common Language Runtime* (CLR) é um ambiente de tempo de execução, que realiza tarefas como: gerenciamento de memória, tratamento de erros, segurança e controle de verções. O código executado dentro do CLR é chamado de código gerenciado, e este deve seguir as especificações da plataforma, ou seja, a CLS (CEMBRANELLI, 2003, p. 11).

As funcionalidades que o CLR oferece estão disponíveis para todas as linguagens .NET. Desde o gerenciador de memória, o *Garbage Collector* , responsável pela liberação de espaço em memória, até heranças e tratamento de exceções (BARWEL et al, 2004, p. 46).

O código escrito em qualquer uma das linguagens aceitas pelo .NET são compiladas em um formato intermediário denominado *Microsoft Intermediate Language* (MSIL). Este código não é interpretado, mas sim compilado em tempo de execução. Gerando então um conjunto de arquivos denominados "*Assembly*" (CHERRY; DEMICHILLIE, 2002, p. 5).

Para Barwell et al (2004, p. 11) existem dois níveis de compiladores no .NET. O compilador da linguagem que pega o código-fonte e cria o MSIL. Este código em MSIL é portável para qualquer plataforma .NET, seja *desktop*, dispositivos móveis ou WEB. E no momento da execução ele será então novamente compilado, agora para código especifico da máquina em que irá rodar.

A figura 4 exemplifica os processos do código gerenciado, desde o código desenvolvido pelo programador até o *assembly*.



Fonte: Cherry e Demichillie (2002).

Figura 4 – Código gerenciado .NET

Este código denominado MSIL é onde são criados os metadados com informações descritivas do código, como os tipos declarados e métodos implementados. Este código em MSIL só será transformado em código binário de máquina quando passar pelo CLR, o que é feito em tempo de execução (CEMBRANELLI, 2003, 13).

Desta maneira o código compilado por diferentes linguagens, pode ser executado em um mesmo ambiente. Este suporte a diversas linguagens torna a escolha da linguagem uma questão de gosto, pois as linguagens habilitadas para o .NET serão compiladas para código MSIL e terão as mesmas funcionalidades, tipos de dados e características de desempenho.

3.1.2 Biblioteca de Classes

O segundo principal componente do .NET *Framework* é a biblioteca de classes, que oferece aos desenvolvedores os componentes de software que necessitam para desenvolverem programas que rodem sob o gerenciamento do CLR.

A biblioteca de classes fornece os serviços e modelos de objetos para dados, entrada/saída, segurança, entre outras funcionalidades. Esta biblioteca contém milhares de classes e interfaces. Entre várias funcionalidades encontradas é possível destacar: classes de acesso a bancos de dados, controles de processamento, interfaces com usuários de *desktop*, dispositivos móveis e WEB, entre outras classes (BARWELL et al, 2004, p. 11).

Todas as classes desta biblioteca são agrupadas e descritas de forma lógica, o que facilita na busca de funções necessárias para o desenvolvimento de aplicativos. Estas classes podem ser usadas por qualquer linguagem .NET e em qualquer plataforma, seja WEB, *desktop* ou dispositivos móveis (CEMBRANELLI, 2003, p. 14).

Além desta organização lógica, os tipos de dados padrão, como por exemplo os tipos de dados mais usados (números inteiros, reais, *strings*) eliminam a necessidade de cada linguagem implementar seu tipo de dado, extinguindo assim a possibilidade de aplicações desenvolvidas sobre o .NET em linguagens diferentes terem dados incompatíveis ou exclusivos (CHERRY; DEMICHILLIE, 2002, p. 8).

Algumas das classes fornecem funcionalidades iguais às anteriores disponíveis através das *Application Programming Interface* (API) do Windows, ou de funções específicas de linguagens, mas sua grande maioria são totalmente novas.

3.2 .NET COMPACT FRAMEWORK

Segundo Mosimann Netto (2005), o .NET *Compact Framework* é parte do .NET *Framework*, criado para possibilitar o desenvolvimento de aplicativos para dispositivos inteligentes (*Smart Device)* da mesma forma que são desenvolvidos aplicativos Windows *desktop*.

“Muito já se tem ouvido falar sobre computação móvel, mas nada se compara à posição do mercado depois do lançamento do .NET *Compact Framework*.” (MOSIMANN

NETTO, 2005).

O .NET *Compact Framework* permite desenvolver softwares para *Pocket PC*, celulares, relógios e *Smartfone*, com recursos encontrados no .NET *Framework*, aproveitando assim todo o conhecimento e código de aplicações desenvolvidas para *desktop* (HADDAD, 2003).

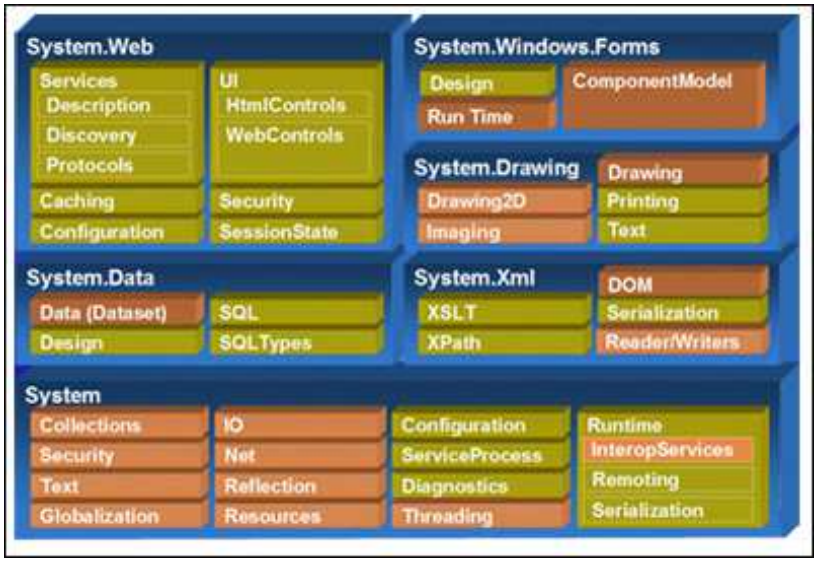
Apesar de ser uma versão reduzida do .NET *Framework*, grande parte das funcionalidades encontradas no *Framework* original são preservadas. O que permite o desenvolvimento de aplicações para dispositivos móveis da mesma maneira que é feito para criar aplicações para *desktop* (MIRANDA, 2005).

Da mesma forma que a versão original, o código produzido em qualquer das linguagens ligadas a plataforma .NET é compilado para uma linguagens intermediária, que por sua vez será executado pelo CLR.

Para Galvin (2004), uma das importâncias em ter o código gerenciado pelo .NET *Compact Framework* em dispositivos móveis é a robustez e a segurança na execução dos códigos desenvolvidos, pois o código não está atrelado ao sistema operacional, mas sim à camada da plataforma.

Algumas das funcionalidades perdidas no .NET *Compact Framework* em relação a versão original estão ligadas a controles de formulário. Alguns por não fazerem sentido existir em um dispositivo móvel, outros por serem complexos de serem portados.

A figura 5 mostra os componentes encontrados no .NET *Compact Framework*.



Fonte: Romelli (2002).

Figura 5 - Componentes do .NET Compact *Framework*

3.3 ASP.NET

ASP.NET é a plataforma da Microsoft voltada à criação de conteúdo WEB dinâmico e também aplicativos WEB mais elaborados e confiáveis como os Web Services. Tem como objetivo reduzir as dificuldades no desenvolvimento WEB, tornando-se assim parte essencial da plataforma no que diz respeito a aplicativos WEB (CEMBRANELLI, 2003, p. 17).

Fazendo parte do .NET *Framework*, todas as características e vantagens da plataforma .NET podem ser atribuídas ao ASP.NET, como o uso da biblioteca de classes .NET, compilação do código em MSIL, orientação a objeto entre outras.

Para Cembranelli (2003, p. 42), uma aplicação ASP.NET é definida como um conjunto de arquivos, páginas, módulos e códigos executáveis que podem ser acessados por meio de um diretório virtual ou subdiretório em um servidor WEB. Aplicações ASP.NET são executadas totalmente separadas, onde cada aplicação é tratada dentro de um domínio de aplicação, podendo assim cada aplicativo WEB manter suas próprias características de processamento e configuração.

Por ser uma tecnologia de servidor, o ASP.NET necessita que, além da plataforma .NET instalada no servidor, haja também um aplicativo de servidor WEB para que possa ser processado. Isso o torna dependente do *Internet Information Service* (IIS), servidor WEB Microssoft (CHERRY; DEMICHILLIE, 2002, p. 17).

Sucessor do *Active Server Page* (ASP), o ASP.NET traz aprimoramentos significativos em relação ao ASP em vários critérios. Destacam-se as mudanças no modelo de programação, gerenciamento de estado e os benefícios herdados da plataforma .NET propriamente dita (CHERRY; DEMICHILLIE, 2002, p. 18).

Segundo Cembranelli (2003, p. 18), entre diversos motivos que levaram a Microsoft a desenvolver uma nova tecnologia WEB destacam-se os seguintes problemas com a tecnologia ASP:

a) código com estrutura confusa: nos códigos desenvolvidos em ASP é possível misturar na página código *HyperText Markup Language* (HTML), *Scripts*, entre outros elementos. Com essa mistura de códigos e tecnologias a manutenção e reutilização do código se tornam muito complexas;
b) páginas interpretadas: no ASP as páginas são interpretadas, ou seja, cada linha do arquivo é lida, interpretada e então transformado em HTML para que seja enviado para o cliente (*browser*). Todo esse processo é repetido a cada solicitação ao

servidor;

c) diversidade de *browsers*: com o aumento de dispositivos com acesso a internet, como celulares, TV's, *Pocket PC*'s, entre outros, cresce também a necessidade de considerar diferentes saídas para uma mesma aplicação.

Estes problemas encontram-se resolvidos no ASP.NET. Além da solução destes problemas houve uma evolução em diversos segmentos.

Em relação ao seu modelo de programação, o ASP.NET tem como diferencial a programação orientada a eventos, ou seja, é possível criar ações que serão executadas quando um específico evento ocorrer. Esse modelo é evolucionário no desenvolvimento WEB, pois chega o mais próximo do que conhecemos do desenvolvimento de aplicativos para *desktop*.

É possível desenvolver métodos que estejam relacionados a eventos da própria página e não só a controles, como por exemplo, ao iniciar ou ao finalizar o carregamento de uma página .

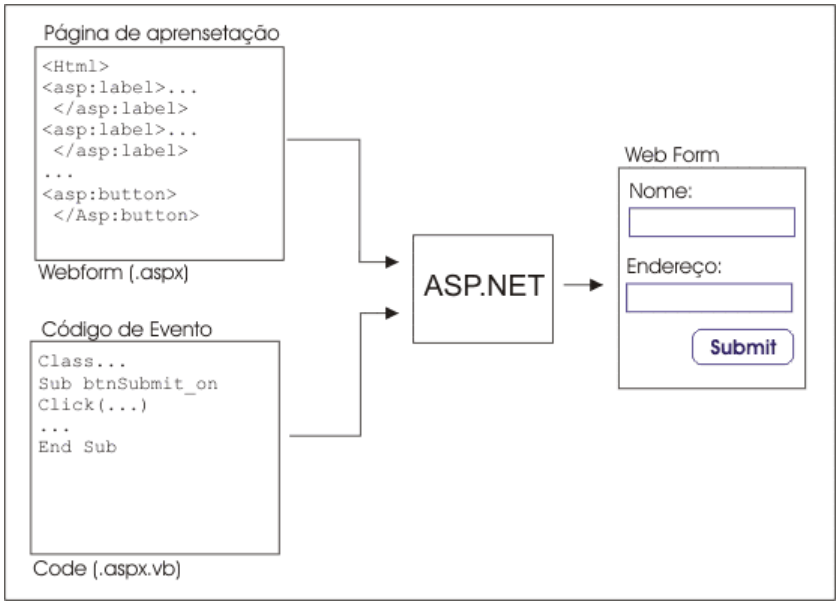
Os controles são uma outra evolução da tecnologia. Seus controles de interface, sofisticados e executados no servidor, permitem sofisticar ainda mais a interface com o usuário. Controles como calendário, *grid*, *treeview*, entre outros, podem ser utilizados sem que se perca a sua compatibilidade com diversos tipos de clientes (*browsers*) existentes.

Vários dos controles disponíveis pelo ASP.NET já eram criados com a utilização de outras tecnologias auxiliares, mas agora isso é possível apenas com o uso do ASP.NET, sem que o desenvolvedor precise conhecer outras linguagens.

Com o uso de controles mais sofisticados seria difícil fazer a depuração de um código a procura de erros, se não fosse a existência de *Debug* e *Trace* no ASP.NET. O que antes era impossível com o ASP, agora se tornou fácil. Recursos de tratamento de erros, depuração e rastreamento foram totalmente melhorados, permitindo desenvolvimento de códigos mais arrojados e complexos.

Outra grande vantagem no uso do ASP.NET, é a possibilidade de separação do código que manipula dados do código de interface com o usuário. Fazer esta separação permite que o desenvolvimento em camadas seja mais fácil de implementar. Este modo de desenvolvimento é chamado de *code behind*, pois todo o código antes misturado ao código de interface "fica por traz da página".

A figura 6 demonstra o desenvolvimento em *code behind.*



Fonte: Cherry e Demichillie (2002).

Figura 6 – Código por trás da página (*code behind*)

O ASP.NET localiza o arquivo de código ligado à página de interface e gera a página que será entregue ao cliente (*browser*). Seguindo essa forma de programação *code behind*, é possível que múltiplas páginas usem o mesmo código, o que facilita a manutenção.

Os arquivos com o código de acesso a dados tem a extensão .aspx.vb, onde .vb é a linguagem usada. Já os Web Services criados em ASP.NET terão a extensão .asmx e o arquivo com código voltado para interface com usuário terá a extensão .aspx.

Toda página criada em ASP.NET é compilada como um objeto de classe chamada Page. Esta classe funciona como um *container* para os controles que fazem parte da página solicitada, assim como seus métodos, eventos e propriedades (CEMBRANELLI, 2003, p. 24).

O código criado no ASP.NET, diferente do que ocorria com o ASP, é compilado. Quando o cliente (*browser*) faz a requisição da página ASP.NET, o *Framework* verifica se a página requisitada já foi compilada, caso não tenha sido, compila apenas a primeira vez. Assim nas próximas requisições a página já estará pronta para execução (GUIMARÃES, 2003) .

Segundo Cembranelli (2003, p. 19), existem dois tipos de aplicativos possíveis de desenvolver em ASP.NET:

a) *WebForms*: utilizado para construir páginas WEB baseado em formulários. Através de controles do servidor é possível criar elementos da interface, programá-los e reutilizados dentro da aplicação. Permite que se desenvolva páginas WEB de maneira muito parecida com o desenvolvimento de aplicações *desktop*;
b) Web Services: utilizado para acessar componentes remotamente, integrar sistemas

> e na comunicação entre aplicações diferentes. Neste modelo de desenvolvimento, é possível que outras aplicações acessem dados e regras de negócio, apenas seguindo a interface definida no Web Service.

Ambos os tipos de aplicativos WEB têm em comum uma vantagem principal em relação aos aplicativos tradicionais desenvolvidos para *desktop*: usam protocolos baseados na internet para permitir que os dados trafeguem de forma fácil pelos *firewalls* (CHERRY; DEMICHILLIE, p. 17).

3.3.1 ASP.NET Mobile Application

ASP.NET *Mobile Application* é a solução ASP.NET para desenvolvimento WEB voltados aos dispositivos baseados em um cliente (*browser*) leve. Para desenvolver páginas para estes dispositivos não é necessário a instalação de nenhum aplicativo no dispositivo cliente, mas é preciso apenas desenvolver utilizando controles específicos para dispositivos móveis. Estes controles são chamados: ASP.NET *Mobile Controls* (MIRANDA, 2005).

Uma das vantagens do desenvolvimento de páginas WEB para dispositivos móveis ao invés de aplicativos diretamente instalados no cliente, é o fato de poder desenvolver aplicações com regras de negócios complexas de acesso a dados e a manter em um servidor, devolvendo assim ao cliente (*browser*) apenas o resultado da sua solicitação.

Qualquer dispositivo que suporte *Wireless Application Protocol* (WAP), *Wireless Markup Language* (WML), *Compact* HTML (cHTML) ou HTML pode ler páginas desenvolvidas com o ASP.NET, incluindo aí grande número de celulares, *Pocket PC`s* entre outros dispositivos (MIRANDA, 2005).

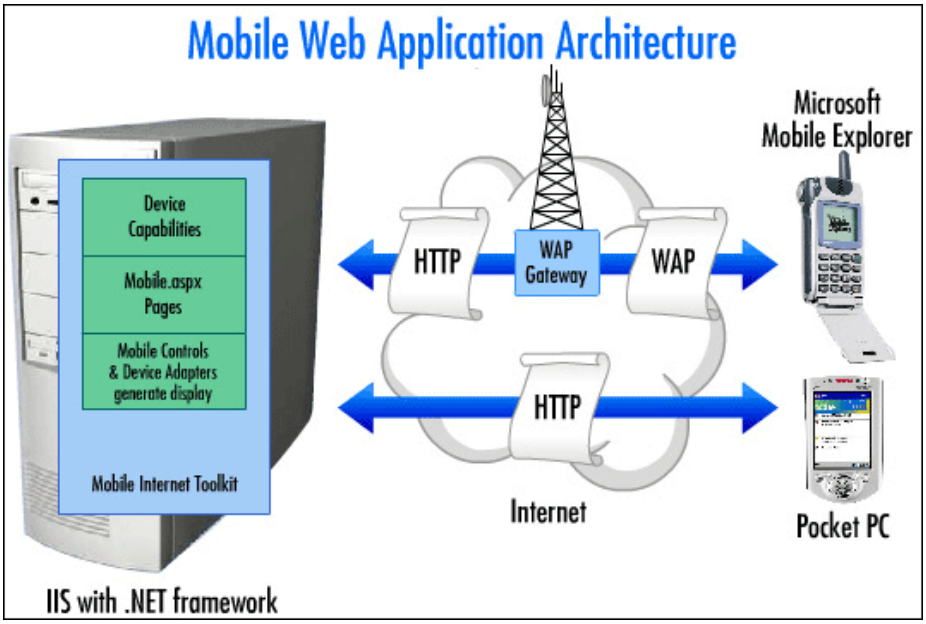
Com esta afirmação é possível observar que uma quantidade maior de dispositivos será capaz de carregar uma página WEB desenvolvida com o ASP.NET *Mobile Controls* do que se fosse desenvolvida apenas em HTML.

No caso específico de telefone celular, o aparelho deve estar ligado a uma operadora, ou seja, toda requisição passa por ela, pois é desta forma que a mesma cobra pela transmissão de dados.

Quando uma página é solicitada através de um celular, é provocada uma requisição HTTP no servidor WEB. No meio do caminho a solicitação passa pela operadora que desvia para o endereço solicitado. Quando a página ASP.NET é processada no servidor, é montada uma resposta HTTP que retorna para a operadora. A operadora com o papel de *Gateway* envia

a resposta ao celular via WAP. Como o celular suporta WML, o .NET *Framework* retorna a resposta em WML (HADDAD, 2003).

Este processo é exemplificado na figura 7.



Fonte: Microsoft Corporation (2006).

Figura 7 - Arquitetura ASP.NET *Móbile Web Application*

Para o desenvolvedor, desenvolver páginas WEB para aplicativos móveis é muito parecido com desenvolver páginas para clientes *descktop*, a não ser pelo uso de controles específicos. Apesar do desenvolvimento ser muito parecido, nem todos os controles disponíveis no ASP.NET estão disponíveis como ASP.NET *Mobile Controls*. Os controles mais sofisticados e que envolvem interfaces mais elaboradas não estão disponíveis (MIRANDA, 2005).

4 WEB SERVICES

O Web Service surgiu com o aumento da popularidade da internet, valorização das redes de computadores e necessidade de sistemas distribuídos. Como uma solução para melhorar a comunicação entre sistemas, independente de hardware, tecnologia ou local geográfico.

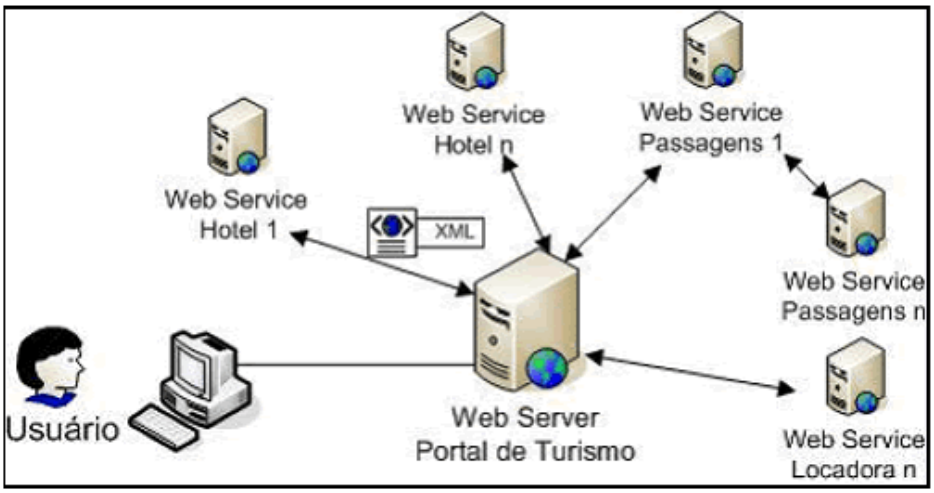
Web Services são aplicativos que usam transportes, codificações e protocolos para troca de informações. Permitem que sistemas em qualquer plataforma ou tecnologia se comuniquem com segurança, com serviços de mensagens confiáveis e transações distribuídas. (SHODJAI, 2006)

Segundo Deitel et al (2004, p. 836), um Web Service é uma classe que possibilita máquinas chamarem métodos de outras máquinas, distribuindo assim a computação entre elas, por meio de formatos de dados e protocolos comuns. Os Web Services têm suas interfaces definidas como mensagens que ele mesmo gera ou aceita. Para solicitar informações ou executar determinado método em um Web Service, o aplicativo solicitante pode ser desenvolvido em qualquer tecnologia ou linguagem, desde que crie e aceite mensagens definidas para a interface do Web Service.

É apenas conhecendo o formato dos métodos a serem chamados e quais os parâmetros a serem passados que é possível consumir os serviços de um Web Service, sem que se saiba detalhes de como o serviço foi implementado. Diferente de diversas tecnologias atuais, os Web Services não são acessados por meio de protocolos específicos. Em vez disso são acessados por meio de protocolos da WEB e formatos de dados padrões. Este conjunto de padrões são mantidos pelo *World Wide Web Consortium* (W3C), consórcio destinado a definir e desenvolver tecnologias de domínio público para a WEB.

Com o uso destes protocolos, os Web Services ultrapassam as barreiras de *firewalls* impostos para a segurança de empresas, pois o trafego é apenas constituído de dados baseados no XML.

Na figura 8 é ilustrado um sistema solicitando um Web Service e por sua vez um Web Service solicitando outros serviços.



Fonte: Reckziegel (2006).

Figura 8 – Web Service

A máquina onde um Web Service reside, normalmente é referenciado como máquina remota. O aplicativo que deseja acesso ao Web Service envia uma chamada de método e seus argumentos à máquina remota, que processa esta chamada e envia uma resposta ao aplicativo que solicitou (DEITEL et al, 2004, p. 837).

É possível então definir que, se um aplicativo pode ser acessado sobre uma rede usando uma combinação de protocolos padrões da WEB, então este é um Web Service.

4.1 ARQUITERURA DE UM WEB SERVICE

O Web Service se baseia na integração de três entidades: provedor de serviços (s*ervice provider*), servidor de registro (*service registry*) e cliente do serviço (*service consumer*). Esta iteração serve para que haja publicação, busca e execução das operações.

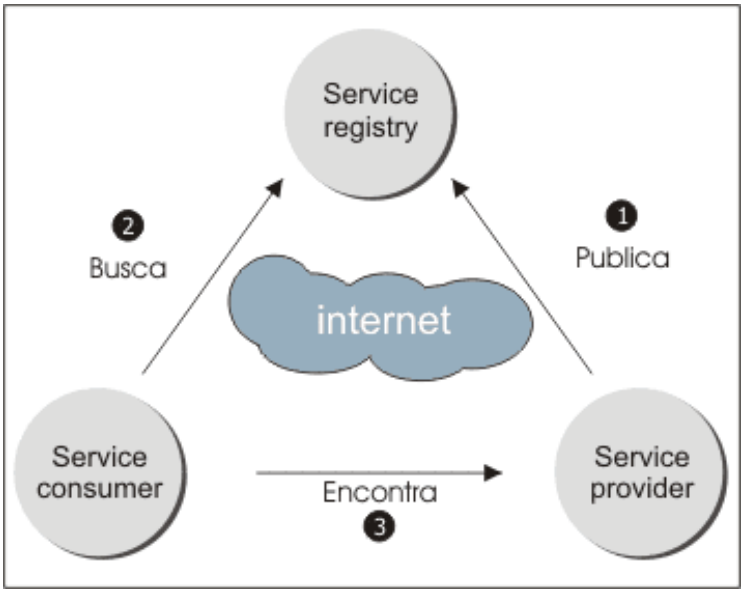
O provedor de serviços representa a entidade que hospeda o Web Service. É esta a entidade responsável por disponibilizar o serviço para que alguém possa utilizá-lo. Para que isto seja possível, seu formato deve estar descrito em um formato padrão, que seja compreensível para qualquer um que precise usar esse serviço. O provedor de serviços deve também publicar os detalhes sobre seu Web Service em um servidor de registros que esteja disponível.

O cliente do serviço pode ser um aplicativo em busca de um método ou uma pessoa acessando através de um *browser,* ou até mesmo um outro Web Service. O cliente do serviço deve conhecer a interface de comunicação do Web Service para que possa consultá-lo. A

descrição da interface é disponibilizada pelo provedor de serviços e recuperada a partir de uma pesquisa no servidor de registro. Através desta pesquisa, o consumidor de serviços pode obter o mecanismo para ligação com este Web Service.

Servidor de registro representa o registro e busca de Web Services baseados em arquivos de descrição de serviços que foram publicados pelos provedores de serviços. Quando um cliente busca por um serviço no servidor de registro ele recupera informações referentes a interface de comunicação.

A figura 9 exemplifica a integração destas três entidades.



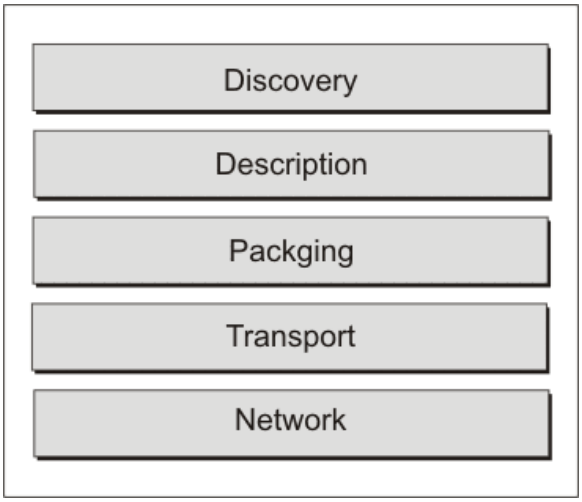
Fonte: adaptado de Snell,Tidwell e Kulchenko (2002).

Figura 9 – Arquitetura do Web Service

Todo esse processo faz uso da porta e protocolo *HyperText Transfer Protocol* (HTTP) para se comunicar. Como normalmente os *firewalls* não bloqueiam esta porta, não existem restrições para o seu funcionamento em redes internas ou internet.

A comunicação e troca de mensagens é possível por causa do pacote de camadas de um Web Service. Ela é implementada através de cinco tipos de tecnologias, organizadas em camadas definidas uma sobre a outra.

A figura 9 exemplifica esta estrutura em camadas.



Fonte: Snell,Tidwell e Kulchenko (2002).
Figura 9 – Pilha de tecnologia do Web Service

As camadas p*ackging*, *description* e *discovery* da pilha de tecnologias do Web Service são essências para o funcionamento do modelo de três entidades (s*ervice provider*, *service registry* e *service consumer*).

Como cada camada da pilha do Web Service se dirige a um problema em separado, a implementação destas camadas também é independente. Ou seja, quando uma nova camada tiver que ser criada ou alterada, não haverá necessidade de alteração nas outras camadas.

A primeira camada da pilha (*discovery*), fornece o mecanismo que um consumidor de Web Service necessita para buscar a interface e descrições do provedor de serviço. A especificação usada e reconhecida para a criação desta camada é a *Universal Description, Discovery, and Integration* (UDDI). UDDI é uma especificação técnica que tem como objetivo descrever, integrar e descobrir Web Services (SNELL; TIDWELL; KULCHENKO, 2002, p. 7, tradução nossa).

Assim como as demais tecnologias do Web Service, o UDDI é baseado em XML, a qual fornece uma plataforma de dados neutra e permite descrever relações hierárquicas (RECKZIEGEL, 2006).

A segunda camada da pilha de tecnologias Web Service (d*escription*), é a camada responsável por descrever o que o Web Service pode oferecer ao cliente. De tal maneira que o cliente do serviço possa fazer uso do serviço (SNELL; TIDWELL; KULCHENKO, 2002, p. 7, tradução nossa). É nesta camada que se descreve os serviços externos, ou interfaces que são oferecidos por um determinado Web Service. Para prover estas informações é usada uma linguagem especificada em XML, a *Web Service Definition Language* (WSDL). Por ser um documento XML sua leitura se torna fácil e acessível. Neste XML, entre vários elementos, estão definidos os tipos de dados, parâmetros de entrada e saída de um serviço e protocolo

usado para comunicação (RECKZIEGEL, 2006).

Para que os dados trafeguem pela rede, eles devem ser “empacotados” em um formato que todos entendam, ou seja, um formato padrão. A terceira camada (*packaging*) é a camada responsável por este empacotamento de dados, onde o *Simple Object Access Protocol* (SOAP) é o padrão mais comum de empacotamento, baseado em XML (SNELL; TIDWELL; KULCHENKO, 2002, p. 8, tradução nossa).

A camada de transporte (*transport*) tem como objetivo mover dados entre dois ou mais locais em uma rede. Nela incluem várias tecnologias que permitem a comunicação entre aplicações. Entre estas tecnologias estão protocolos como *Transmission Control Protocol* (TCP), HTTP, *Simple Mail Transfer Protocol* (SMTP) entre outros. O Web Service pode ser construído sobre quase todos os protocolos de transporte (SNELL; TIDWELL; KULCHENKO, 2002, p. 8, tradução nossa).

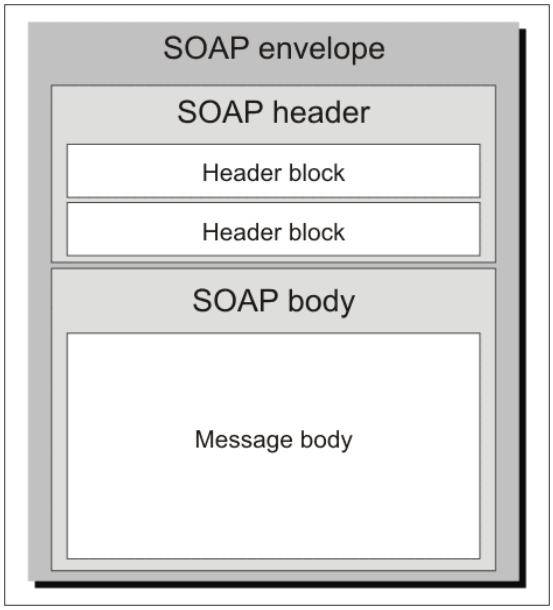
A camada de rede (*network*) da pilha de tecnologias do Web Services é exatamente a mesma do modelo de rede de TCT/IP. É sobre esta camada que os protocolos de transporte trafegam.

4.2 SOAP

SOAP é um protocolo para troca de mensagens em ambiente distribuído, baseado em definições XML. Utilizado para acessar Web Services, seu papel básico é de definir mecanismos que expressem o modelo semântico das aplicações, assim como o modelo em que os dados são decodificados. Esse protocolo empacota as chamadas e retornos aos métodos dos Web Services, sendo utilizado principalmente sobre HTTP (RECKZIEGEL, 2006).

A estrutura de uma mensagem SOAP é bastante simples: se resume em um envelope contendo um cabeçalho (opcional), e um corpo.

A figura 10 demonstra a estrutura de uma mensagem SOAP.



Fonte: Snell,Tidwell e Kulchenko (2002).
Figura 10 – Estrutura de mensagem SOAP

A sintaxe XML definida para uma mensagem SOAP é baseada nas definições do W3C. Estas definições contêm os esquemas XML para a criação de uma mensagem SOAP (SNELL; TIDWELL; KULCHENKO, 2002, p. 14, tradução nossa).

Segundo Leopoldo (2003), os três elementos básicos do SOAP são definidos como:

a) envelope: parte obrigatória de uma mensagem SOAP, funciona como um recipiente para todos os outros elementos da mensagem. Para garantir a chegada da mensagem, é preciso que o envelope contenha informações especificas do protocolo de transporte. Especificamente no protocolo HTTP, existe um cabeçalho indicador de endereço de entrega;
b) cabeçalho: o cabeçalho SOAP é parte opcional. Ele é usado para definir meta-dados que podem prover um contexto para a mensagem ou redirecionar o processamento da mensagem. Um exemplo prático do uso de cabeçalho é para autenticação em casos que as credenciais sejam requeridas. Apesar de opcional, caso haja o cabeçalho, ele deve ser o primeiro a aparecer após a abertura da *tag* do envelope;
c) corpo SOAP: parte obrigatória da mensagem, que guarda dados específicos da chamada de um método particular, como o nome e parâmetros de entrada, saída e resultados obtidos pelo método. Seu conteúdo depende do tipo da mensagem, se ela é uma requisição ou uma resposta.

Para envio e recepção de parâmetros nas mensagens, o SOAP tem em sua especificação suporte a tipos de dados baseados na especificação do XML *Schema Definition*

(XSD). Esta especificação define os tipos primitivos de dados, assim como sua estrutura hierárquica. Além dos tipos primitivos mais comuns como *boolean*, *byte*, *int*, *string*, entre outros, o usuário pode definir tipos, desde que o mesmo possa ser representado em um XSD (LEOPOLDO, 2002).

4.3 WSDL

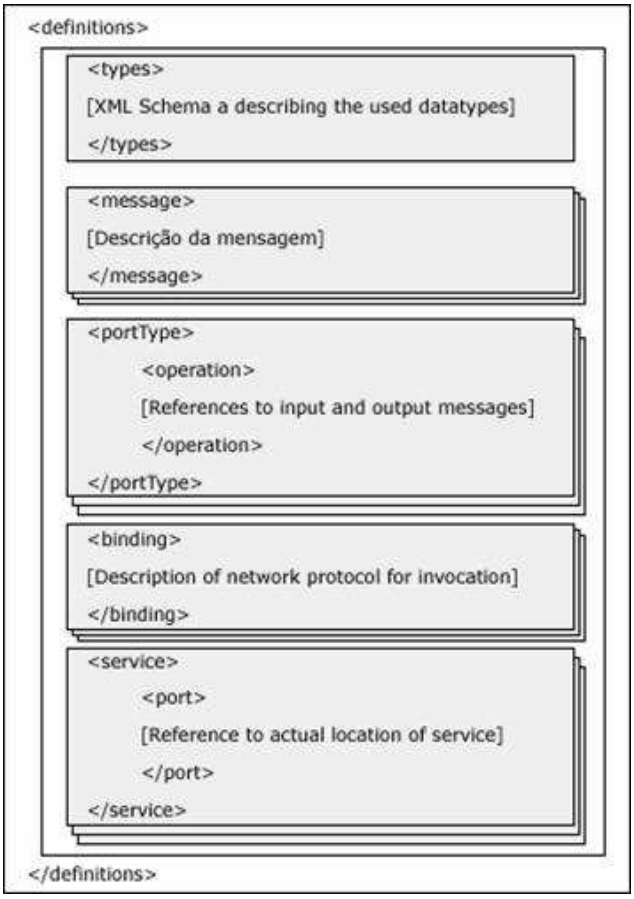
Web Service Definition Language (WSDL) define um sistema para descrição de serviços. Através desta linguagem são descritos os serviços e interfaces oferecidos por uma determinada aplicação, assim como a sua localização. Como outras tecnologias para Web Services, sua especificação também é baseada no XML (RECKZIEGEL, 2006).

Em um documento WSDL existem elementos que formam sua especificação. Estes elementos servem para definir os parâmetros de um determinado serviço.

Para Reckziegel (2006), os elementos deste documento estão definidos como:

a) *types* (tipos): container de definição de tipos de dados;
b) *message* (mensagem): define parâmetros de entrada e saída de um serviço;
c) *operation* (operações): são as definições dos métodos, relação entre parâmetros de entrada e saída;
d) *port type* (tipo de porta): descreve o agrupamento lógico das operações, ou seja, das definições dos métodos;
e) *binding* (vínculo): especifica o protocolo e formato de dado a ser usado para um dado *port type*;
f) *service* (serviço): define a localização real do serviço. É uma coleção que pode conter várias portas, e cada uma é especificada para um tipo de vínculo.

Estes elementos podem ser melhor compreendidos se observada a figura 11, que demonstra os principais elementos de um documento WSDL.



Fonte: Reckziegel (2006).
Figura 11 – Principais elementos de um WSDL

Como em qualquer XML, o WSDL também precisa de um elemento raiz. Este elemento, como pode ser visto na figura 10, chamasse *definitions*.

O WSDL utiliza *namespaces* para aumentar a reutilização dos elementos/componentes definidos em seu documento. Estes *namespaces* são espaços para nomes definidos no interior de um XML, o que permite sua unicidade de nomes.

Os principais *namespaces* usados em um documento WSDL estão indicados no Quadro 1.

Prefixo	**URI do *namespace***	**Descrição**
WSDL	http://schemas.xmlsoap.org/wsdl	*Namespace* de WSDL para *Framework* WSDL
http	http://schemas.xmlsoap.org/wsdl/http	*Namespace* de WSDL para WSDL HTTP *GET* & vínculo *POST*
MIME	http://schemas.xmlsoap.org/wsdl/mime	*Namespace* de WSDL para vínculo MIME de WSDL
XSD	http://www.w3.org/2001/XMLShema	*Namespace* do esquema conforme definido pelo esquema de XSD

Fonte: adaptado de Seely (2002).
Quadro 1 – Prefixos de *namespaces* mais usados no WSDL.

Quando observado um documento WSDL é preciso lembrar que ele é apenas um

documento XML. Ele contém elementos que derivam de seu *namespace* e *schema*, definido por um XSD (SEELY, 2002, p. 167, tradução nossa).

5 DESENVOLVIMENTO DO TRABALHO

O desenvolvimento deste trabalho consiste em um aplicativo para dispositivo móvel (*Pocket PC*), um Web Service para distribuição e sincronização dos dados e um site que, consultando o Web Service, possa ser acessado tanto por celulares como por *desktop*. O objetivo de agregar todas estas tecnologias é ir de encontro com a falta de serviços para dispositivos móveis de ponta a ponta, ou seja, em uma ponta a empresa prestadora de serviço utilizando um dispositivo móvel e na outra ponta um usuário deste serviço consultando informações. Para colocar em prática esta visão de mercado, foi escolhida uma área que pudesse se beneficiar destas tecnologias. A área escolhida é a área de entrega de encomendas, pois retrata bem a possibilidade de agregar dispositivos móveis, assim como a possibilidade de usuários destes serviços de entrega poderem também usar dispositivos móveis para consulta de informações sobre sua entrega.

A figura 12 exemplifica o desenvolvimento do trabalho.

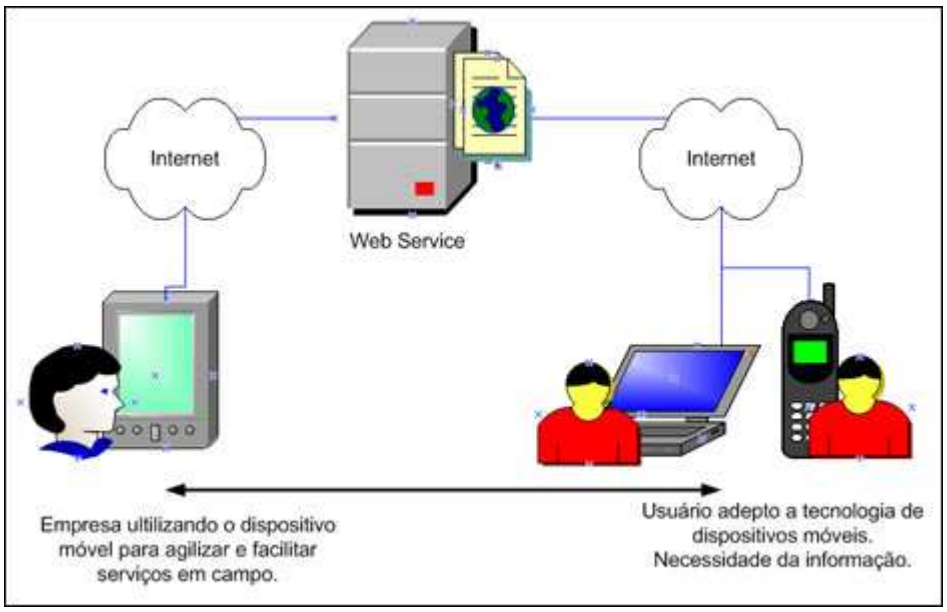


Figura 12 – Representação dos componentes deste trabalho

Para agregar ainda mais valor à pesquisa, houve preocupação em desenvolver todo o trabalho com base em uma única tecnologia. Este desafio foi de encontro com a promessa das novas tecnologias que trabalham com a idéia de “plataformas”, permitindo que com uma mesma linguagem e de forma quase que transparente, seja possível desenvolver aplicativos para dispositivos e ambientes diferentes. Para este trabalho a tecnologia de desenvolvimento usada é a plataforma .NET.

O aplicativo para dispositivo móvel (*Pocket PC*), foi desenvolvido utilizando a tecnologia .NET C*ompact Framework* da plataforma .NET. Com auxílio da ferramenta de desenvolvimento *Visual Studio* 2005 (VS 2005), o sistema foi desenvolvido e testado com o uso de um emulador de *Pocket PC*.

O Web Service utilizado para consulta e atualização de dados foi desenvolvido com o ASP.NET, outra tecnologia que também faz parte da plataforma .NET. O Web Service é encarregado de atualizar o dispositivo móvel com as entregas a serem entregues, assim como receber informações sobre as entregas realizadas.

Além dos serviços de atualização de dados do *Pocket PC*, no Web Service há também serviços para os usuários/clientes consultarem suas mercadorias. Para estes usuários foram desenvolvidas páginas WEB que utilizam os serviços do Web Service. Desenvolvidas com o ASP.NET, estas páginas podem ser acessadas por qualquer dispositivo com acesso a internet, até mesmo celulares.

5.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos funcionais (RF), requisitos não funcionais (RNF) e regras de negócio (RN) listados abaixo, representam os requisitos do trabalho como um todo. Destacando quando um determinado requisito ou regra de negócio se refere ao usuário-entregador ou ao usuário-cliente.

Os requisitos são:

a) ter acesso restrito com usuário e senha no aplicativo do usuário-entregador (RF);
b) a senha do usuário-entregador deverá trafegar até o Web Service de forma ilegível (RF);
c) verificar a autenticidade do usuário-cliente nas solicitações dos serviços no Web Service, consultando seu código de cliente e CPF (RF);
d) o aplicativo do usuário-entregador deverá consultar o Web Service para receber informações sobre as encomendas a serem entregues (RF);
e) permitir a busca pela encomenda no dispositivo móvel do usuário-entregador, para que seja dada a situação atual da encomenda (RF);
f) permitir que o usuário-cliente consulte o histórico das encomendas;
g) permitir que, ao entregar a mercadoria, o usuário-entregador solicite a assinatura

de recebimento do destinatário no próprio *Pocket PC*, caso seu *status* seja "entregue" (RF);

h) permitir que o usuário-entregador descarregue as informações no Web Service após o final das entregas (RNF);
i) permitir a consulta das encomendas pela internet, tanto por computador como por celular (RNF);
j) permitir que o usuário-cliente, ao consultar sua entrega (tanto por celular como por *descktop*), possa ver a assinatura do receptor caso ela já tenha sido entregue. (RF);
k) armazenar informações no dispositivo móvel do usuário-entregador com o uso de *eXtensible Markup Language* (XML) (RNF);
l) caso a encomenda já tenha passado por três tentativas de entrega sem sucesso, a mesma automaticamente se tornará "não entregue" de forma definitiva, retirando-a das próximas entregas (RN);
m) o aplicativo para usuário-entregador e usuário-cliente não deverão fazer qualquer referência ou acesso ao banco de dados das encomendas. Todas as solicitações serão feitas ao Web Service (RF);
n) desenvolver utilizando tecnologias que fazem parte da plataforma de desenvolvimento .NET (RNF).

5.2 ESPECIFICAÇÃO

Na especificação técnica do sistema é apresentado o diagrama de classe, diagrama de caso de uso e diagrama de atividades, baseados na linguagem de modelagem UML. Além dos diagramas, algumas especificações serão textualizadas.

5.2.1 Diagrama de Caso de Uso

Apresenta-se na figura 13 o diagrama de caso de uso mostrando, de uma perspectiva de usuário, as funções e serviços oferecidos. O diagrama abrange toda a estrutura do trabalho, usando retângulos para separar os casos de uso que se relacionam com o lado do entregador e os que estão ligados ao lado cliente. O Web Service está referenciado como um ator por

receber e agir sobre ações dos outros atores, seja em forma de consulta ou envio de informações.

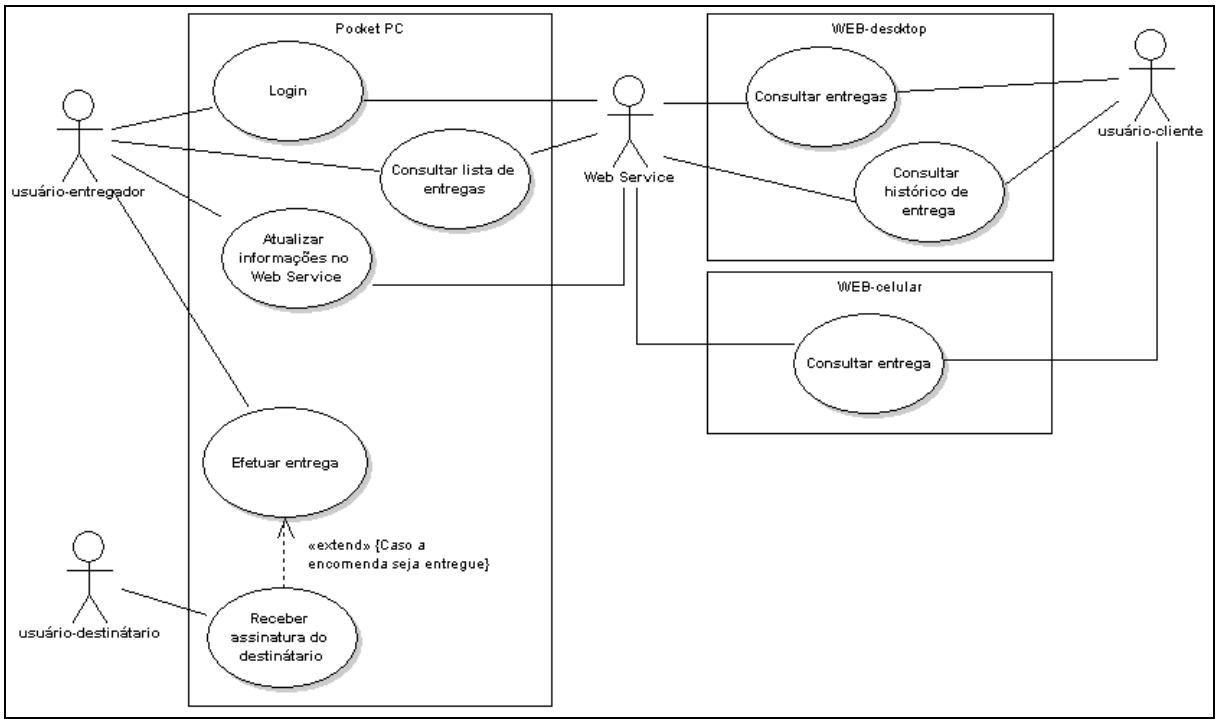


Figura 13 – Diagrama de caso de uso

5.2.1.1 Login

O usuário-entregador deverá se logar antes de acessar outras funcionalidades do sistema. O usuário-entregador informa seu usuário, sua senha e clica no botão acessar. Antes de enviar os dados para validação a senha é criptografada impedindo que ela trafegue de forma legível. Após criptografia da senha os dados são então enviados ao Web Service para que seja verificada a autenticidade do usuário. O Web Service compara os dados e retorna o resultado da validação. Com o resultado positivo do acesso, o sistema mostrará as opções disponíveis para o usuário-entregador, caso contrário será apresentada uma mensagem indicando o não sucesso do login e solicita novamente o usuário e senha. Para este caso de uso há o pré-requisito do acesso ao Web Service.

A figura 14 representa o diagrama de atividades deste caso de uso.

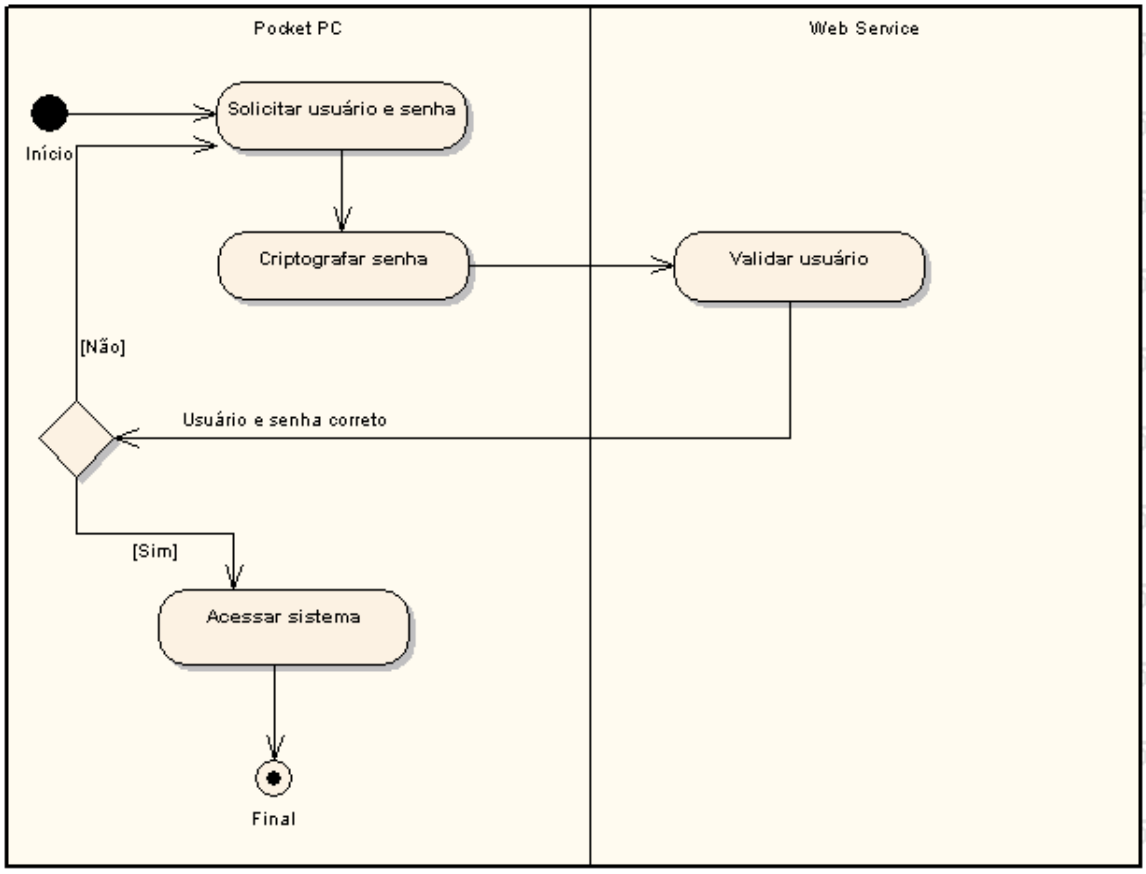


Figura 14 – Diagrama de atividade: *login*

5.2.1.2 Consultar lista de entregas

O usuário-entregador solicita a lista de entregas a serem efetuadas. O sistema verifica a existência do arquivo XML contendo a listagem de entregas. Caso não exista solicita ao Web Service a listagem de entregas que estão vinculadas ao entregador, fazendo assim com que o login seja um pré-requisito para este caso de uso. Caso exista, informa ao usuário a já existência de uma listagem e questiona se deseja carregar uma nova ou carregar a listagem já existente. Após o recebimento da listagem de entregas ela é armazenada em arquivo XML no dispositivo móvel. Apenas são recebidas informações das encomendas que estão com seu status: “tramitando” ou “ñ entregue (haverá próxima tentativa)”. Ao fim do recebimento da lista de entregas ou do carregamento da listagem já existente, o usuário-entregador é informado sobre o sucesso da operação. Para este caso de uso há o pré-requisito do acesso ao Web Service.

A figura 15 representa o diagrama de atividades deste caso de uso.

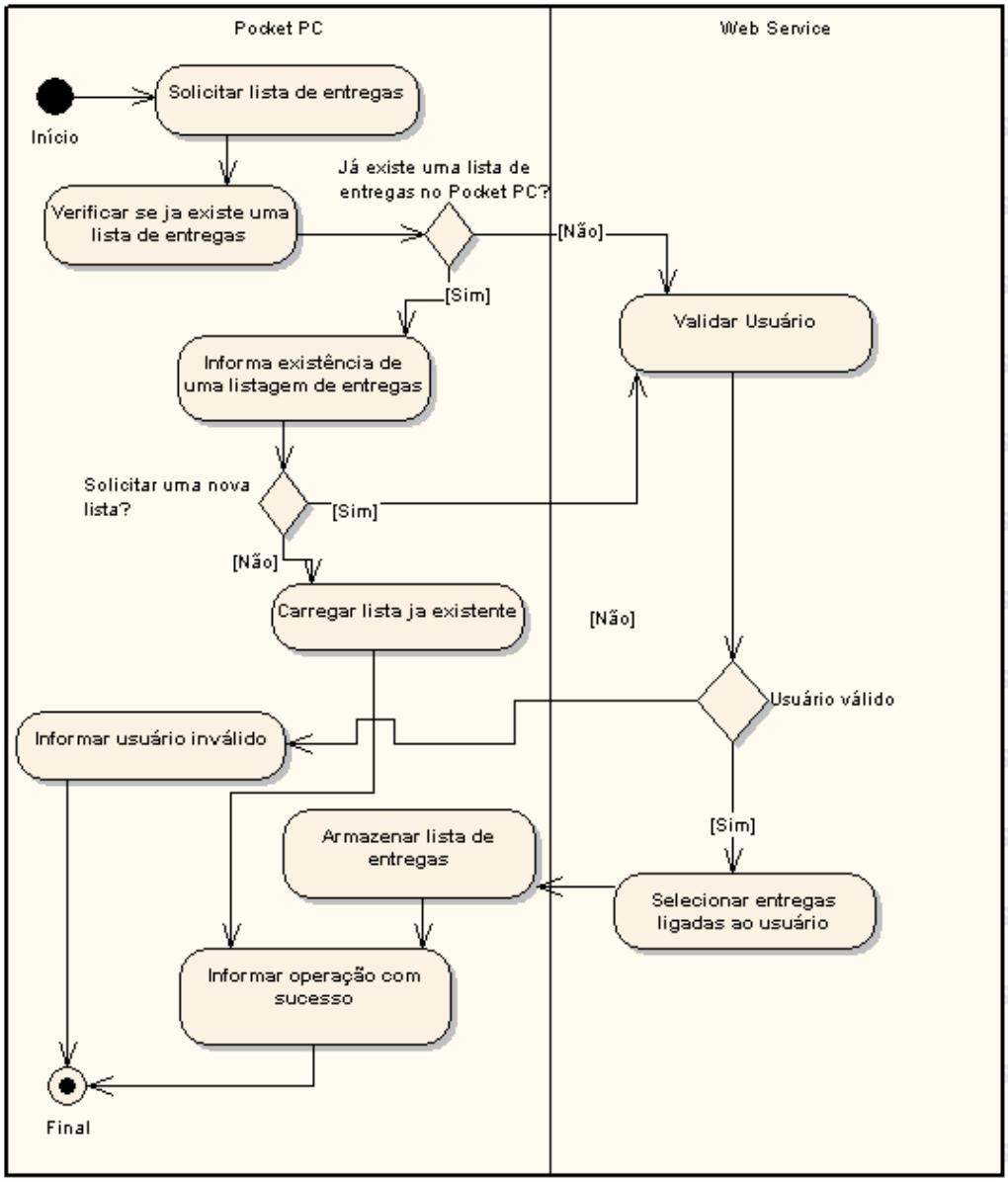


Figura 15 – Diagrama de atividades: consultar lista de entregas

5.2.1.3 Efetuar entrega

Neste caso de uso o usuário-entregador determina se efetua ou não a entrega. O sistema carrega a listagem de entregas que ainda não foram efetuadas. O usuário-entregador escolhe o número da entrega a ser efetuada e o sistema mostra ao usuário a descrição da encomenda, nome do remetente, nome do destinatário e endereço de entrega. Estas informações serão de somente leitura. Após conferir as informações o usuário terá uma opção de “Entregar” ou “Não entregar”. Caso a opção escolhida seja a “Entregar” o sistema mostrará ao usuário-entregador uma área para solicitar a assinatura do destinatário. Após a

assinatura, o usuário-entregador deverá clicar em salvar para que seja armazenada a entrega como “entregue” e a assinatura do remetente. Caso a opção escolhida seja a de “Não entregar” será apresentada uma lista de possíveis motivos para não efetuar a entrega da mercadoria. O usuário-entregador escolherá então uma das seguintes opções listadas: “Ausente”, “Mudou-se”, “Endereço insuficiente”, “Desconhecido(a)”, ”Recusado” e “Não encontrado”. Após escolher a opção que melhor descreve o motivo da não entrega da encomenda, o usuário-entregador deverá clicar em salvar para armazenar os dados informados. Todas as alterações efetuadas serão apenas atualizadas no dispositivo móvel, permitindo que seja possível coletar dados de forma desconectada do Web Service.

A figura 16 representa o diagrama de atividades deste caso de uso.

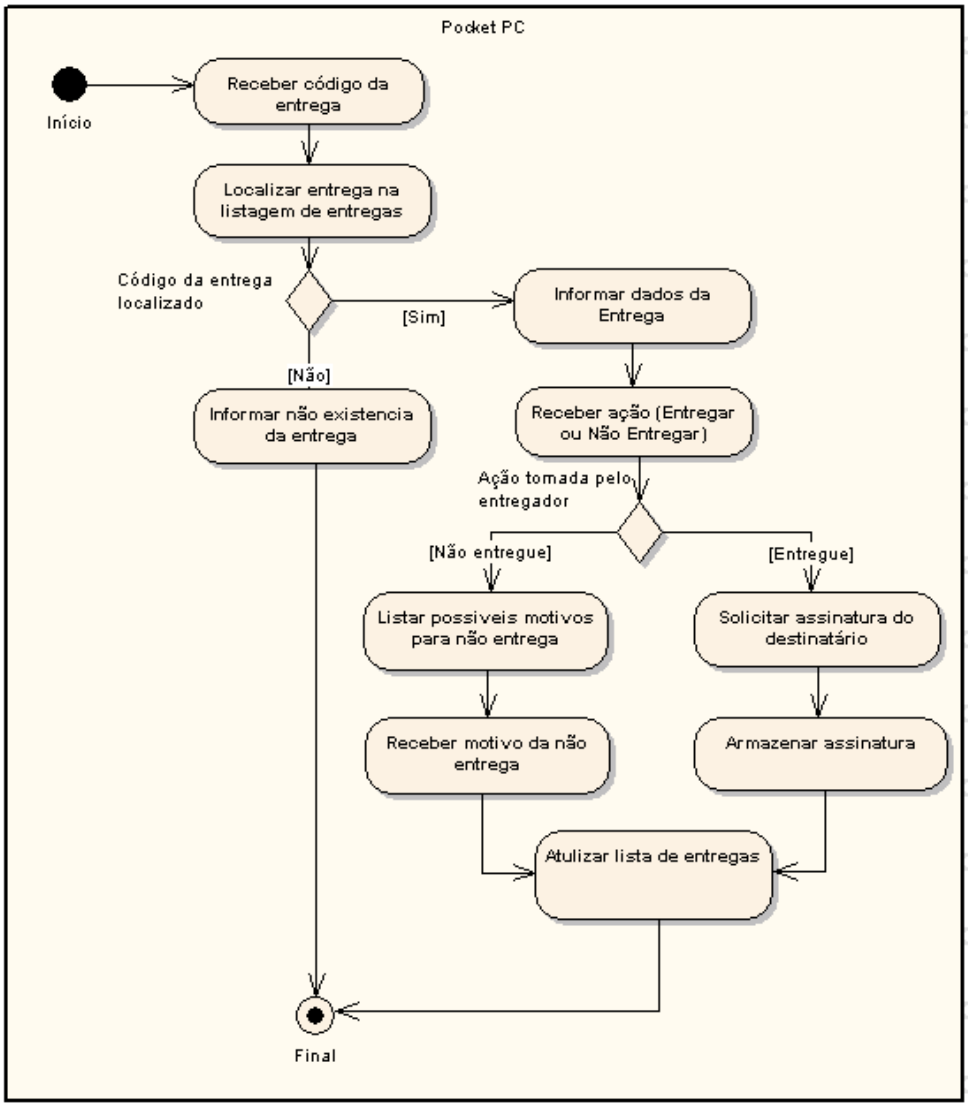


Figura 16 – Diagrama de atividades: efetuar entrega

5.2.1.4 Atualizar informações no Web Service

O usuário-entregador clica no botão “Enviar lista de entregas”, o sistema carrega todas as entregas em que houve atualização no seu status de entrega. Estas entregas efetuadas são enviadas ao Web Service. Para cada entrega o Web Service adicionará um registro de histórico, para futuras consultas. Caso a entrega já tenha dois registros de histórico (significando que já houveram duas tentativas de entrega), o sistema verifica se a entrega atual foi feita com sucesso ou não. Caso não tenha sido efetuada com sucesso, a entrega se torna “não entregue” de forma permanente, não aparecendo mais nas próximas listagens de entregas. Além de seu status se tornar permanente “não entregue”, será adicionada na observação da entrega a frase: “Tentativa de entrega encerrada.”. Caso a entrega tenha sido efetuada com sucesso, o Web Service receberá a assinatura do cliente em forma de pontos que constituem linhas e as transformará em imagem, armazenando no servidor. O nome da imagem será formado pelo código do cliente e código da entrega. Ao terminar de processar todas as entregas, uma mensagem indicando o término será mostrado ao usuário-entregador. Após este caso de uso, as informações já estarão disponíveis para consulta pelo usuário-cliente com os dados atualizados.

A figura 17 representa o diagrama de atividades deste caso de uso.

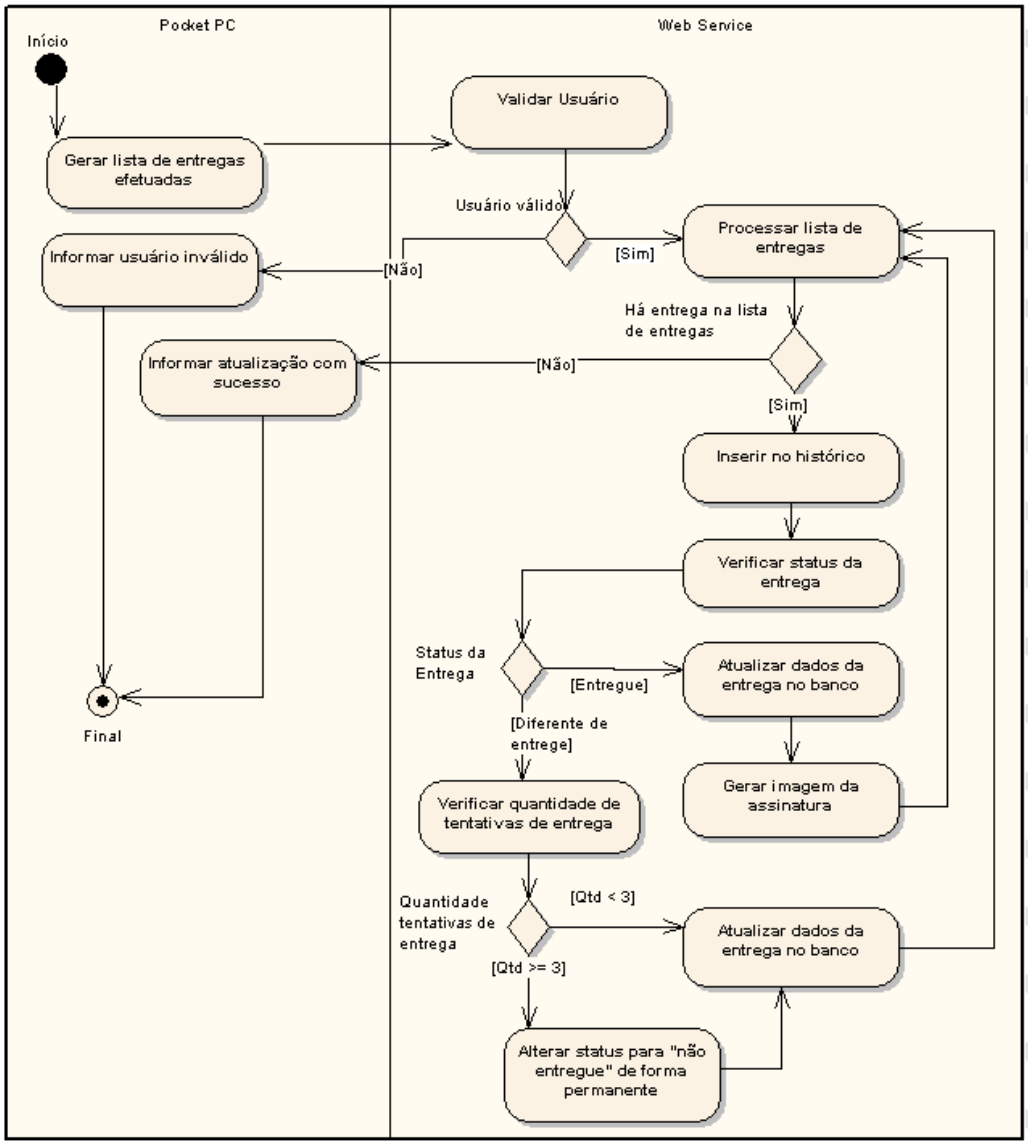


Figura 17 – Diagrama de atividades: atualizar informações no Web Service

5.2.1.5 Consultar Entregas

O usuário-cliente informa seu código de cliente e CPF e clica em enviar. O Web Site envia os dados ao Web Service que valida ou não os dados do cliente. Caso os dados não sejam válidos, o usuário-cliente é informado do erro. Caso contrário, serão listadas todas as entregas pertinentes ao código e CPF informado. Os dados resultantes da solicitação serão mostrados em uma grid com o código da entrega, destinatário, descrição da entrega e status atual. Além destas informações, cada registro terá um *link* chamado “detalhe”. Optando por ver detalhes de uma determinada entrega, o Web Site mostrará todas as informações ligadas á entrega escolhida. Junto às informações detalhadas sobre a entrega também é mostrada a

assinatura do destinatário, caso a entrega tenha sido efetuada com sucesso. As informações listadas tem caráter informativo, não podendo ser alterado.

A figura 18 representa o diagrama de atividades deste caso de uso.

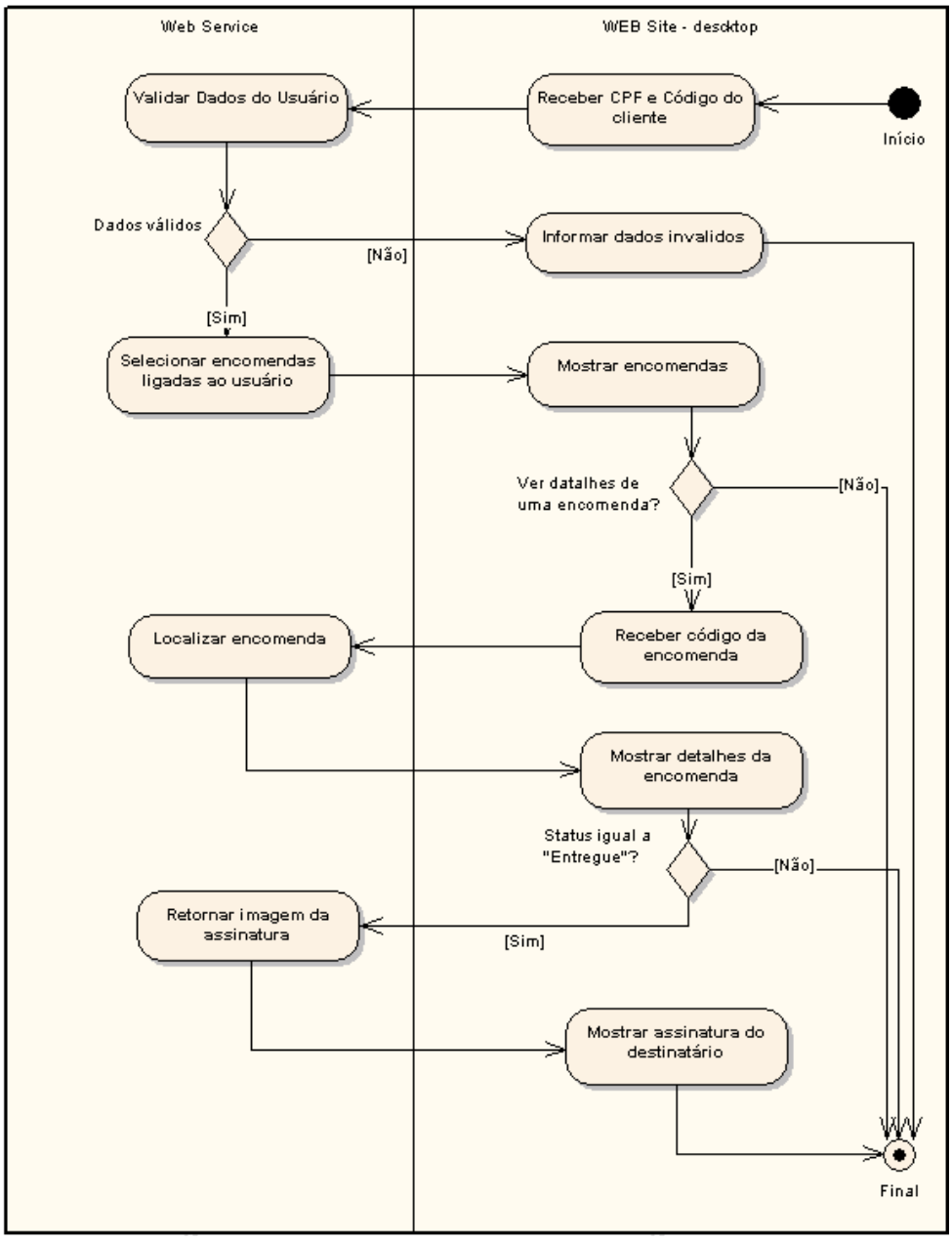


Figura 18 – Diagrama de atividades: consultar entregas

5.2.1.6 Consultar Histórico da Entrega

Após efetuar seu acesso ao Web Site, o usuário-cliente encontrará um campo para informar o código da entrega que deseja visualizar o histórico. O Web Site solicita ao Web

Service os dados históricos que dizem respeito ao usuário e código da entrega. Caso o código da entrega não esteja ligado ao cliente solicitante, uma mensagem será exibida informando o erro. Caso contrário serão listadas em forma de tabela as informações sobre o histórico da encomenda, as informações listadas serão: código da entrega, data do histórico, status e observação.

A figura 19 representa o diagrama de atividades deste caso de uso.

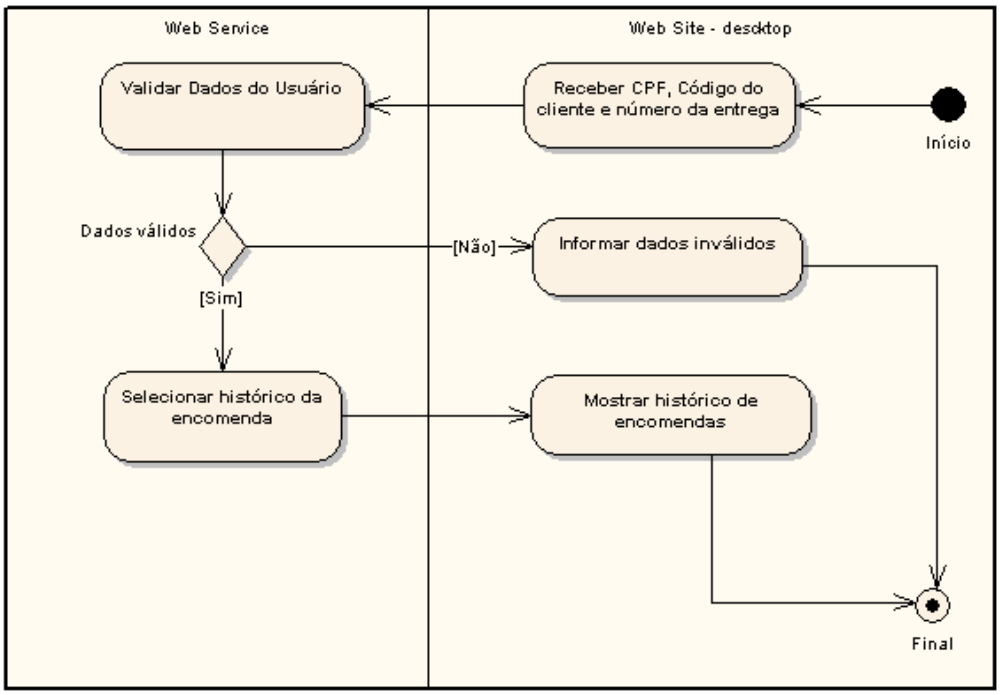


Figura 19 – Diagrama de atividades: consultar histórico da entrega

5.2.1.7 Consultar Entrega

Este caso de uso representa o acesso pelo Web Site específico por celulares com acesso a internet. O usuário-cliente que deseja consultar sua entrega pelo celular deverá informar seu código de cliente, CPF e código da encomenda. O Web Site envia as informações ao Web Service solicitando as informações pertinentes ao número da entrega informado. Se as informações enviadas forem válidas, serão mostrados ao usuário os detalhes da entrega solicitada, assim como a assinatura do destinatário, caso a entrega tenha sido efetuada com sucesso. Se as informações não forem válidas, uma mensagem será exibida ao usuário informando o erro.

A figura 20 representa o diagrama de atividades deste caso de uso.

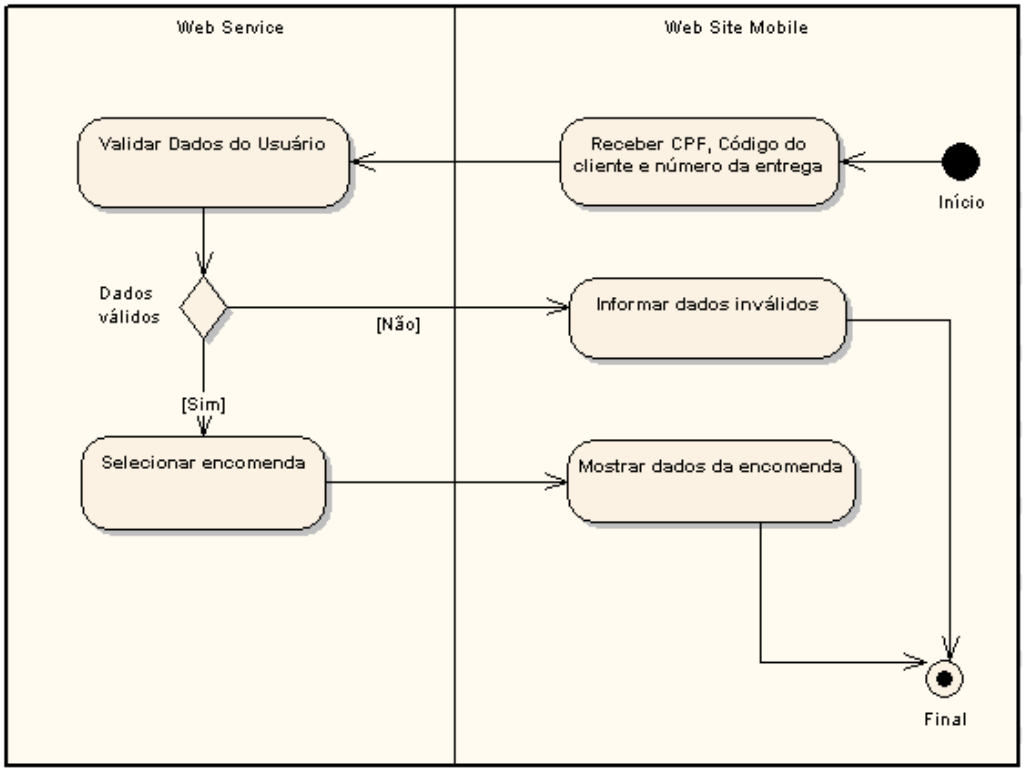


Figura 20 – Diagrama de atividades: consultar entrega

5.2.2 Diagrama de modelo de classe

O desenvolvimento deste trabalho representa a construção de três aplicativos, o aplicativo para *Pocket PC*, Web Service e WEB Site separados em diferentes diagramas. O diagrama de classe que representa os aplicativos do *Pocket PC* e Web Service é mostrador na figura 21. O diagrama que representa o WEB Site é apresentado na figura 22 e figura 23.

No primeiro diagrama são especificadas apenas as classes relacionadas ao negócio. Não mostrando as classes relacionadas a telas, componentes ou que apenas intermedeiam a comunicação ao banco de dados ou Web Service.

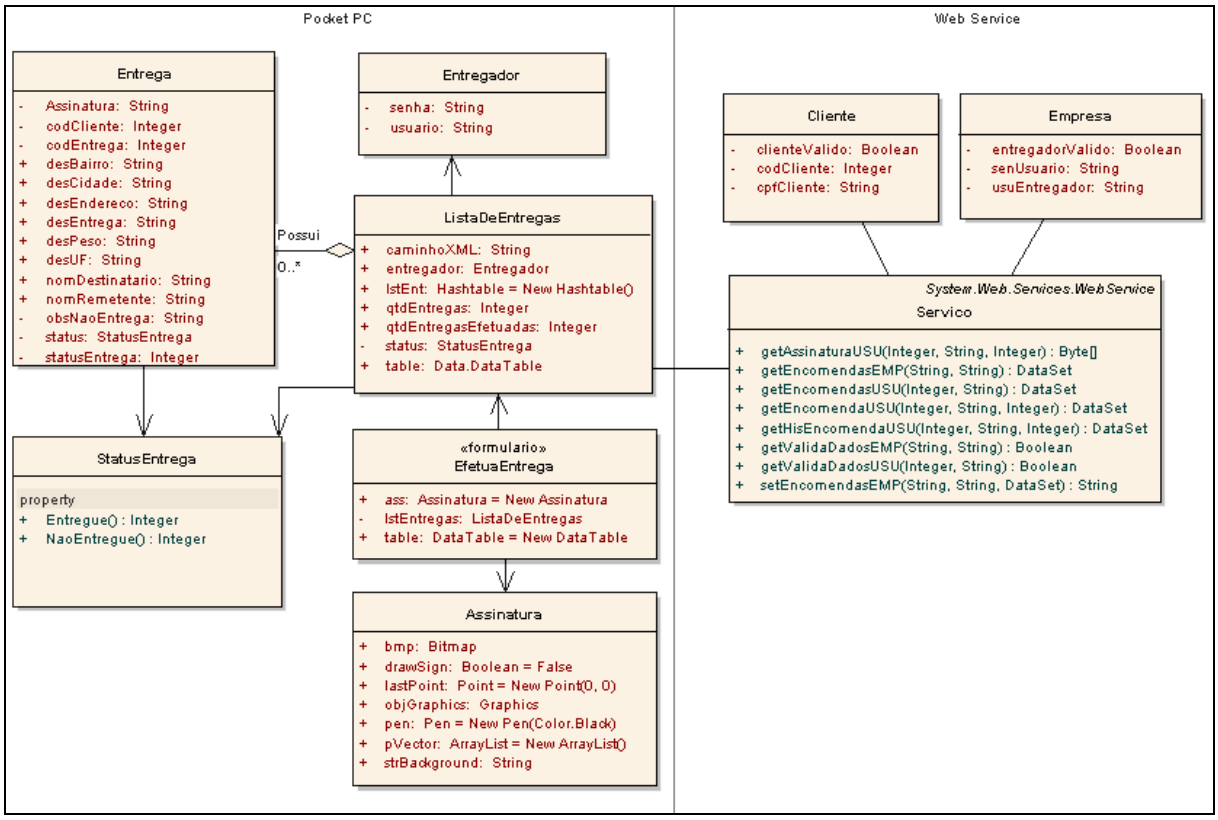


Figura 21 – Diagrama de Classe

Explicando melhor as classes implementadas:

a) Entrega: é a classe responsável por identificar uma entrega. É nesta classe que constam os dados da entrega, assim como o atributo que determina se houve ou não sua entrega. Métodos desta classe:

- New: método construtor para criação desta classe (em VB.NET o método construtor é definido pelo nome “New”). Recebe como parâmetro o código do cliente (codCli: *Integer*), código da entrega (codEnt: *Integer*), descrição da entrega (desEnt: *String*), descrição do peso da entrega (desPes: *String*), nome do remetente (nomRem: *String*), nome do destinatário (nomDes: *String*), descrição do endereço (desEnd: *String*), nome da cidade (desCid: *String*), sigla do estado (desUF: *String*) e nome do bairro (desBai: *String*),
- efetuaEntrega: método responsável por colocar o objeto em status de entregue. Recebe como parâmetro a assinatura do receptor, o dado da assinatura é formado por pontos que representam linhas, este parâmetro é do tipo *string* (assinatura: *String*),
- naoEfetuaEntrega: método responsável por colocar o objeto em status de não entregue. Recebe como parâmetro a observação que identifica o motivo da não entrega da mercadoria (obs: *String*),

- getAssinatura, getCodCliente, getCodEntrega e getStatusEntrega: métodos que como os próprios nomes definem, retornam respectivamente a assinatura (*String*), código do cliente (*Integer*), código da entrega (*Integer*) e status da entrega (*Integer*);

b) ListaDeEntregas: esta classe contém métodos pertinentes ao gerenciamento das entregas e métodos que efetuam a solicitação e envio de encomendas ao Web Service. A ligação direta com o Web Sevice representa a visão da aplicação de forma a abstrair as classes intermediárias e recursos da tecnologia .NET que fazem acesso via HTTP. O arquivo XML também é gerenciado por esta classe. Principais métodos:
   - geraTab: método que cria as colunas do atributo “table”, este atributo é do tipo *DataTable* e representa o arquivo XML em memória,
   - addEntrega: recebe como parâmetro o objeto entrega (objEntrega: Entrega). Insere o objeto no atributo “lstEnt” e atualiza o atributo “table“,
   - getListaDeEntregas: método sem parâmetros de entrada, responsável por retornar o atributo “table” com os dados das entregas,
   - getEntrega: recebe como parâmetro o número da entrega (codEnt: *Integer*). Verifica a existência da encomenda informada e caso exista retorna o objeto entrega correspondente. Caso não exista retorna “vazio”,
   - salvarListaXML: sem parâmetros de entrada este método é responsável por serializar em arquivo o parâmetro “table” que está em memória para o dispositivo móvel. Usa como referência para o local onde o arquivo deve ser criado o atributo “caminhoXML”,
   - carregarListaXML: este método é responsável por ler o arquivo XML que é referenciado pelo atributo “caminhoXML”. Para cada entrega representada no XML é criado um objeto “entrega” e adicionado ao atributo “lstEnt”. O atributo “table” também é atualizado,
   - efetuaEntrega: recebe como parâmetro um *boolean* (que informa se a entrega foi realizada com sucesso ou se não foi possível ser realizada), o código da entrega (codEntrega: *Integer*), assinatura do destinatário (assinatura: *String*), caso a entrega tenha sido efetuada com sucesso, e motivo da não entrega (obsNaoEntregua: *String*), caso a entrega não tenha sido realizada. Este método localiza na lista a entrega referenciada e executa o método “efetuaEntrega” ou “naoEfetuaEntrega” do objeto “entrega”,

- receberListaWebService: método responsável por receber do Web Service a lista de entregas ligada ao usuário logado. Este método instancia a classe serviço do Web Service e aguarda sua resposta. Recebendo o parâmetro do tipo *DataTable* o método verifica cada registro recebido e cria os objetos do tipo entrega, populando a lista de entregas. Com todas as entregas recebidas, o método "salvarListaXML" é executado,
- atualizarListaWebService: este método é responsável por instanciar a classe de serviço do Web Service. O método prepara um parâmetro do tipo *DataSet* contendo os dados do XML. Este parâmetro é populado com as entregas que tiveram seu status modificado. Após instanciar o serviço do Web Sevice e enviar os dados, este método aguarda a confirmação do envio pelo Web Sevice e após a confirmação exclui o arquivo XML que estava em disco, além de zerar a lista em memória e os atributos "qtdEntregas" e "qtdEntregasEfetuadas";

c) Entregador: esta classe armazena os dados de usuário e senha do entregador. O objeto é criado após o usuário logar. Métodos desta classe:
   - New: o método construtor desta classe recebe como parâmetro usuário (usuario: *String*) e senha (senha: *String*) e armazena nos respectivos atributos,
   - encriptaSenha: recebe como parâmetro o atributo senha (senha: *String*), este método criptografa a senha usando um algoritmo de criptografia *Hash*. Após a criptografia, o método armazena a senha criptografada no atributo senha desta classe,
   - validarDados: método responsável por instanciar a classe serviço do Web Sevice e invocar o método responsável por validar os dados de usuário e senha. O método retorna a resposta do Web Service (*boolean*);

d) Status: classe usada para identificar o código condizente com o status. Contém apenas duas propriedade: Entregue e Não Entregue;

e) Assinatura: esta classe herda as propriedades da classe Control do .NET. Ela foi criada para que seja possível capturar os movimentos do cursor assim que for precionado dentro de uma determinada área. Esta classe foi criada com auxílio de um algoritmo criado por Behera (2004) na linguagem C#, além da migração seu código foi adaptado para a realidade deste trabalho. Alguns dos métodos desta classe:
   - onMouseDown: método que sobrepõe o método da classe herdada (Control) para controlar o momento em que o usuário clicou sobre a área delimitada para

assinatura. Neste método é setado a um atributo o valor *true*, indicando que o usuário clicou sobre a tela. Neste momento é armazenada a posição X e Y do click,

- onMouseMove: este método sobrepõe o método na classe herdada para controlar o movimento do mouse. Durante a movimentação do mouse sobre a área delimitada são armazenadas as coordenadas do cursor em um *array* de pontos. Com a última posição do cursor armazenada, mais a posição atual do cursor, são criadas linhas adicionadas a um atributo *bitmap*,
- onMouseUp: sobrepõe o método da classe herdada, é utilizado para alterar o atributo que indica que o usuário está assinando para *false*,
- clear: método com o objetivo de reiniciar o controle permitindo que o usuário possa começar a sua assinatura novamente. Este método limpa o *array* de pontos e re-cria o atributo *bitmap* executando o método initMemoryBitmap,
- initMemoryBitmap: este método cria um objeto do tipo *bitmap* atribuindo valores de largura, altura e uma imagem inicial indicando o local de assinatura.
- getAssinatura: método responsável por fazer a leitura do *array* contendo os pontos X e Y que formarão a assinatura. Este método cria uma *string* para ser enviada por Web Service separando cada linha (X1 Y1 e X2 Y2) por um caractere delimitador;

f) EfetuaEntrega: Esta classe é herdade pelo formulário de entrega. Esta apenas representada no diagrama para referenciar a classe que utiliza a classe Assinatura;

g) Sevico: esta classe tem como objetivo especificar quais métodos estão disponíveis para acesso por aplicativos que queiram consultar ou enviar informações ao Web Service. Está é a classe usada para especificar o arquivo WSDL. Ela não contém atributos, apenas as chamadas de métodos que por sua vez instanciam as classes Cliente ou Empresa, classes estas, que contém as regras de negócio e acesso a banco de dados. Os métodos desta classe foram nomeados com um sufixo “USU” e “EMP”, apenas para denominar respectivamente os métodos ligados ao usuário (cliente) e a empresa (empresa de entregas):

- getAssinaturaUSU: recebendo como parâmetro código do cliente (cod_cliente: *Integer*), CPF do cliente (cpf_cliente: *String*) e código da entrega que a assinatura está vinculada (cod_entrega: *Integer*). Instancia a classe cliente para fazer a verificação dos dados do cliente e receber um *array* de bytes da imagem da assinatura que corresponde ao código do cliente e entrega. Este método

retorna um *array* de bytes,

- getEncomendasUSU: este método retorna um *DataSet* (em formato XML) com as entregas ligadas ao cliente. Recebe como parâmetro o código do cliente (cod_cliente: *Integer*) e CPF do cliente (cpf_cliente: *String*). Instancia a classe cliente para validação dos dados e executar o método getEnomendas,
- getEncomendaUSU: este método retorna apenas os dados de uma determinada entrega. Recebe como parâmetro o código do cliente (cod_cliente: *Integer*), CPF do cliente (cpf_cliente: *String*) e código da entrega (cod_entrega: *Integer*). Instancia a classe cliente para validação dos dados e executar o método getEncomenda,
- getHisEncomendaUSU: recebe como parâmetro o código do cliente (cod_cliente: *Integer*), CPF do cliente (cpf_cliente: *String*) e código da entrega (cod_entrega: *Integer*). Método responsável por retornar o histórico de uma determinada entrega (*DataSet*: XML). Instancia a classe cliente para validação dos dados e executar o método getHisEncomenda,
- getValidaDadosUSU: este método recebe como parâmetro os dados do cliente (cod_cliente: *Integer*, cpf_cliente: *String*). Retorna um *boolean* indicando a validade dos dados informados,
- getEncomendasEMP: retorna entregas ligadas ao usuário entregador. Este método recebe como parâmetros os dados do entregador (usuario: *String*, senha: *String*), executa o método getEncomendas da classe Empresa e retorna seu resultado (*DataSet*: XML),
- getValidaDadosEmp: método responsável por validar dados do usuário. Recebe como parâmetro os dados do usuário e retorna um *boolean* indicando validade dos dados informados,
- setEncomendasEMP: este método recebe como parâmetro os dados do usuário entregador (usuario: *String*, senha: *String*) e a lista de entregas efetuadas (dados: *DataSet*);

h) Cliente: classe com os métodos pertinentes ao usuário cliente. Esta classe contém as regras de negócio e o acesso ao banco de dados ligados ao cliente. Métodos:

- New: o método construtor desta classe recebe como parâmetro o código do cliente (cod_cliente: *Integer*) e CPF do cliente (cpf_cliente: *String*). Armazena nos atributos da classe os parâmetros recebidos. É ainda no método construtor que é feita a consulta ao banco de dados sobre a validade das informações.

Caso os dados sejam válidos, o atributo clienteValido recebe *true*, caso contrário recebe *false*. Em todos os métodos o atributo clienteValido é consultado para verificar a autenticidade dos dados do usuário que criou o objeto,

- getValido: este método apenas retorna o valor do atributo clienteValido (*booblean*),
- getEncomenda: recebe como parâmetro o número da entrega (cod_entrega: *Integer*). Método responsável por retornar as informações sobre uma determinada entrega ligada aos atributos da classe codCliente e cpfCliente. Retorna os dados em um *DataSet* (XML),
- getEncomendas: não há parâmetros para este método. Retorna informações de todas as entregas ligadas aos dados do cliente contidos no atributo codCliente e cpfCliente. O retorno é um *DataSet* (XML),
- getAssinatura: recebe como parâmetro o código da entrega (cod_entrega: *integer*) e o caminho físico no servidor das imagens das assinaturas (caminhoImg: *String*). O método verifica se o código da entrega está ligado aos atributos codCliente e cpfCliente. Validando a ligação com os dados, o método localiza a assinatura no caminho especificado mais o nome formado pelo "Cód.Cliente_CódEntrega.jpg". Localizado o arquivo, o método carrega a imagem em memória (*MemoryStream*) e retorna os *bytes* da imagem em um array de *bytes*,
- getHisEncomenda: recebe como parâmetro o código da entrega (cod_entrega: *Integer*). Retorna o histórico das tentativas de entregas deste código de entrega (*DataSet*: XML);

i) Empresa: classe com os métodos pertinentes ao usuário entregador. Esta classe contém as regras de negócio e o acesso ao banco de dados ligados à empresa entregadora.

- New: o método construtor desta classe recebe como parâmetro o usuário (usu_entregador: *string*) e a senha (sem_entregador: *string*) do usuário entregador. Atribui aos atributos usuEntregador e senEntregador os parâmetros recebidos. Faz a validação dos dados junto ao banco de dados e atribui o resultado da validação ao atributo entregadorValido (*boolean*),
- getEncomendas: método sem parâmetro. Após verificar se atributo entregadorValido esta como *true*, são selecionadas todas as entregas ligadas a

este usuário (*DataSet*: XML),

- setEncomendas: este método recebe como parâmetro um XML (*DataSet*) com a lista de entregas efetuadas. O método faz a leitura do *DataSet* e para cada entrega extrai as informações para chamar o método setEncomenda,
- setEncomenda: este método tem como parâmetro o usuário (usuário: *String*), senha (senha: *String*), código do remetente (cod_cliente: *Integer*), código da entrega (cod_entrega: *String*), status da entrega (cod_status: *Integer*), observação da entrega (obs_entrega: *String*) e assinatura do destinatário (des_assinatura: *String*). É verificado se este usuário entregador é responsável pela entrega. Caso não seja, o método retorna *false*, indicando a não execução do método. É neste método que é feita a verificação da quantidade de tentativas de entregas (caso ela não tenha sido entregue), inserção na tabela de histórico e alteração do status atual da entrega. Ao final, o método executa o método setAssinatura caso a entrega tenha sido efetuada. Retorna *true* ao chegar ao fim com sucesso,
- setAssinatura: método responsável por transformar o dado recebido em forma de pontos de vetor em imagem. Recebe como parâmetro o código do cliente, código da entrega e a assinatura (assinatura: *String*). O parâmetro assinatura é quebrado em conjuntos de informações cada um contendo: “x1,y1” e “x2,y2”, onde cada conjunto destes indica uma traço feito pelo usuário destinatário. Ao fim da leitura de todos os pontos, a imagem gerada em memória é serializada e transformada em um arquivo “.JPG” com o nome criado com base no código do cliente e código da entrega,
- getMimMax: método criado para auxiliar na criação do arquivo de imagem, retorna os maiores e menores pontos da imagem.

A seguir é mostrado o projeto do Web Site onde foi usado o modelo de especificação de projetos WEB chamado *Web Application Extension* (WAE), consultado em Conallen (2003), possibilitando assim identificar as páginas WEB e suas interações.

Na figura 22 é possível observar a ligação entre a aplicação WEB e os arquivos StyleSheet.css e funcoes.js, respectivamente arquivos destinados ao estilo visual da página e as funções *Java Script* utilizadas no Web Site. Desta mesma forma é observada a ligação entre o aplicativo WEB e o Web Service.

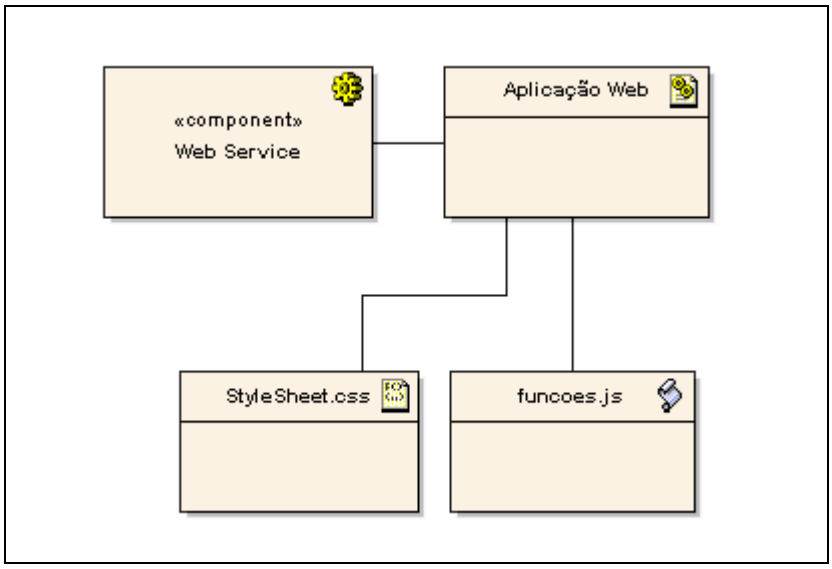


Figura 22 - Diagrama aplicativo WEB

Na figura 23 a aplicação WEB é detalhada, demonstrando as interações entre os arquivos, classes e componentes WEB.

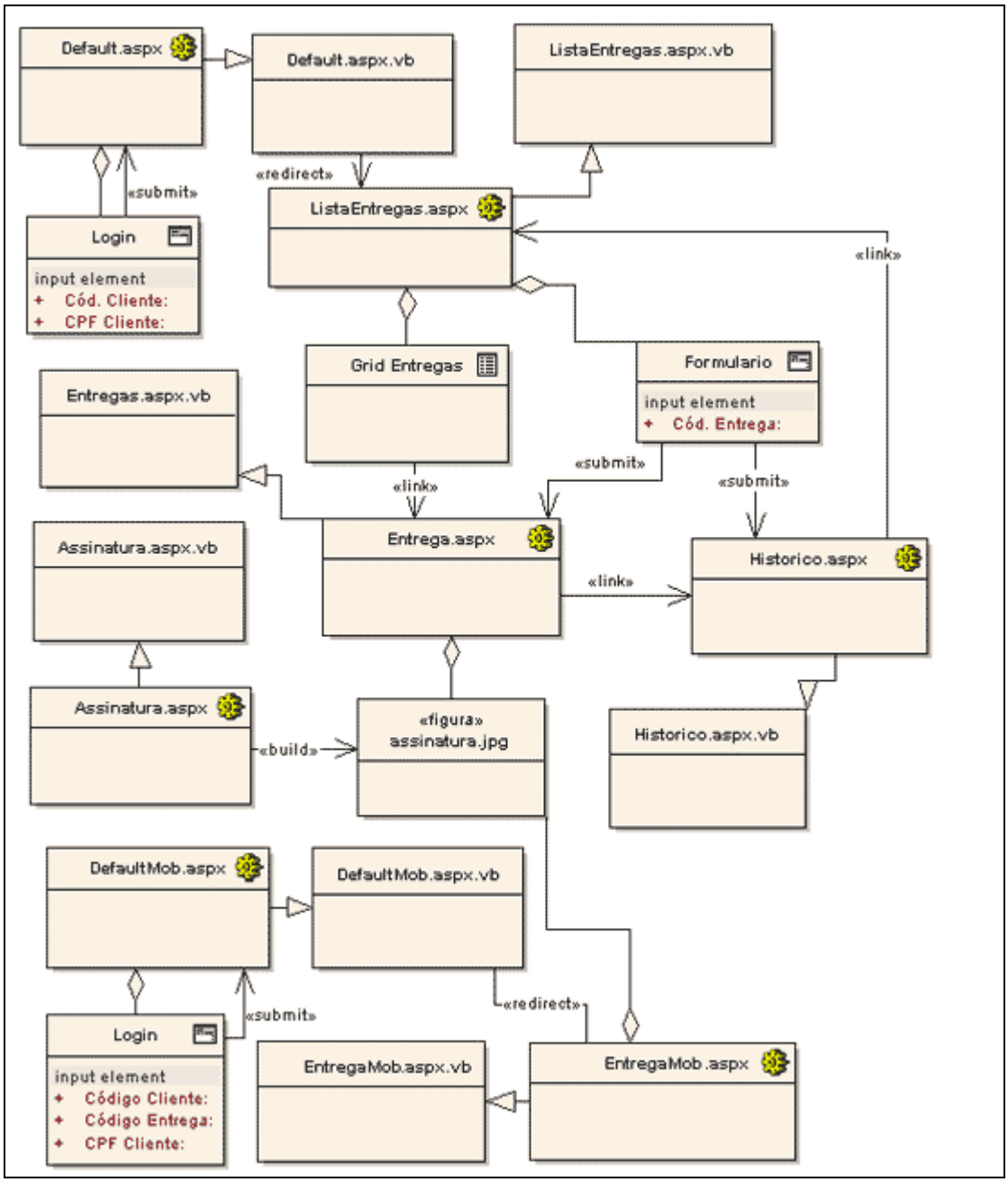


Figura 23 - Diagrama detalhado do aplicativo WEB

Para explicar melhor o diagrama demonstrado na figura 23, serão descritas as páginas na ordem em que elas são executadas.

A página “default.aspx” é a página inicial a ser executada no Web Site destinado a *desktop*. Como todas as páginas .ASPX, esta página herda sua classe .ASPX.VB, executa o método “Page_Load” (método padrão na inicialização de páginas .NET) e retorna o resultado em HTML. Nesta página há um formulário agregado destinado ao login. Preenchendo os campos, este formulário posta os dados para própria página que por sua vez executa o método “validaCliente”, método que consulta os dados do usuário no Web Service. Com a validação efetuada com sucesso, o usuário é redirecionado para ListaEntregas.aspx.

A página ListaEntregas.aspx faz em seu “Page_Load” (método herdado da sua classe .aspx.vb) a consulta ao Web Service para receber um XML com todas as entregas do usuário logado. Estas informações são usadas para popular um *grid* que está agregado na página, que contém um link para a página Entrega.aspx. Ainda na página ListaEntregas.aspx, há um formulário para agilizar o acesso direto a página Entrega.aspx e Histórico.aspx.

Entrega.aspx é a página onde o usuário vê detalhes de uma entrega específica, recebendo como parâmetro o código da entrega. Em sua inicialização é solicitado ao Web Service informações sobre esta entrega, que após a consulta são listadas ao usuário. Nesta página há agregada uma imagem gerada pela página Assinatura.aspx, esta imagem representa a assinatura do destinatário.

A página Assinatura.aspx recebe como parâmetro o código da entrega, código do cliente e CPF do cliente. A classe herdada pela página contém em seu método “Page_Load” a solicitação ao Web Service da imagem que representa a assinatura recebida na entrega. Com o A*rray* de *Bytes* recebido o método altera a propriedade “ContentType” da página para "image/jpg", e devolve dados em binário, ao invés de HTML.

Histórico.aspx é a página destinada a listar as tentativas de entregas realizadas para uma determinada entrega. Tendo *link* para o usuário voltar a ver sua lista de entregas.

No caso de acesso por celular, foram desenvolvidas duas páginas que tem como objetivo, após o processamento no servidor, devolver páginas com conteúdo WML, linguagem interpretada por browsers de celular.

Para acessar a página inicial destinada a celulares deve ser executado o arquivo: DefaultMob.aspx. Seguindo a mesma idéia das outras páginas descritas anteriormente, esta também herda uma classe .aspx.vb. A página agrega um formulário de login que solicita o código do cliente, CPF do cliente e código da entrega. Este formulário posta os dados para a própria página que após validar os dados junto ao Web Service, redireciona o usuário para a página EntregaMob.aspx.

A página EntregaMob.aspx recebe como parâmetro os dados do formulário preenchido anteriormente e executa uma solicitação ao Web Service. Tendo o retorno das informações sobre a entrega solicitada, os dados são mostrados ao usuário. Agregada a esta página também está a imagem da assinatura do destinatário, gerada pela página Assinatura.aspx.

5.3 IMPLEMENTAÇÃO

Neste capítulo serão apresentados tópicos pertinentes as técnicas, ferramentas e a operacionalidade dos aplicativos desenvolvidos neste trabalho.

5.3.1 Técnicas e ferramentas utilizadas

Neste tópico serão apresentadas as principais ferramentas e técnicas utilizadas no desenvolvimento deste trabalho. Abordando a ferramenta de desenvolvimento utilizada para implementação, banco de dados, emuladores e exemplificação das técnicas utilizadas e código implementado.

5.3.1.1 Visual Studio .NET

O Visual Studio .NET (VS.NET) é um ambiente de desenvolvimento integrado (IDE) da Microsoft para o desenvolvimento, teste, depuração e documentação de programas. Ambiente de desenvolvimento que permite a utilização de diversas linguagens de programação, além de possuir ferramentas de edição e manipulação de diversos tipos de arquivo (DEITEL et al, 2004, p. 28).

Além da possibilidade da escolha e inserção de novas linguagens (que façam parte da plataforma .NET), é possível no VS.NET desenvolver para diferentes plataformas, seja WEB, seja um serviço, um Web Service ou dispositivos móveis na própria ferramenta sem o uso de outros softwares.

Uma das características do ambiente é a possibilidade de criar “Soluções” e para cada solução adicionar vários projetos. Tanto um projeto WEB ou *desktop* pode fazer parte de uma mesma solução. Projeto contém as interfaces de usuário e arquivos-fonte, assim como outros arquivos.

Quando criado um novo projeto é possível escolher qual o tipo de projeto, como mostrado da figura 24.

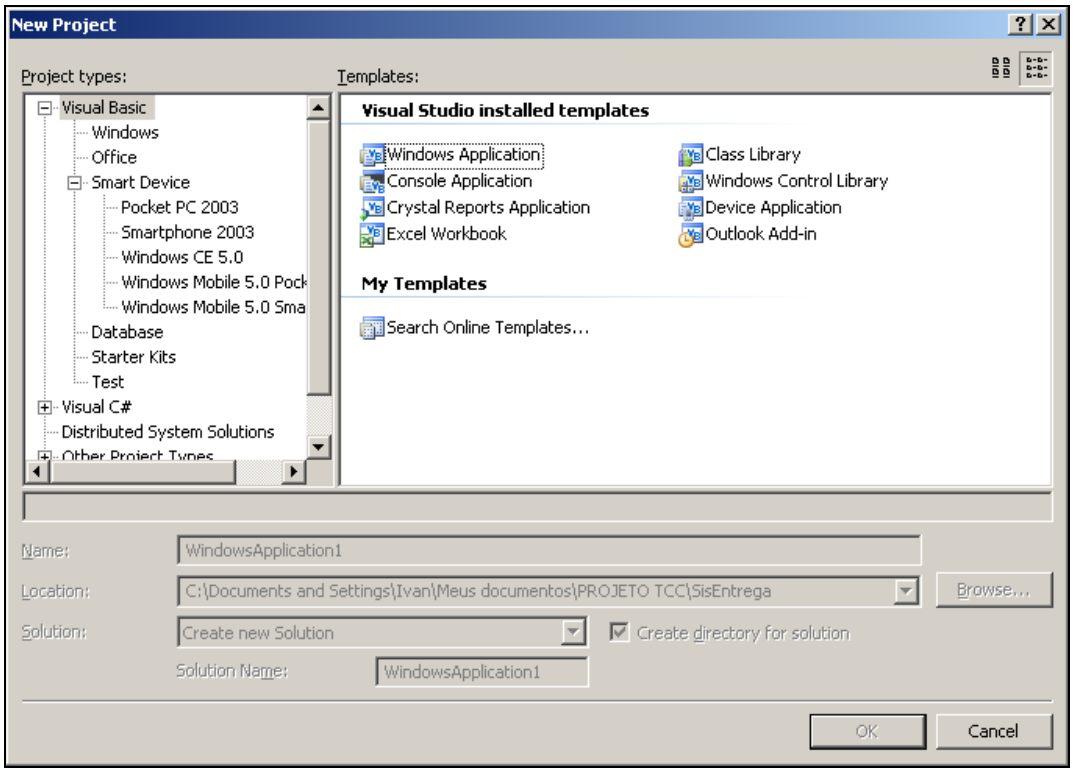


Figura 24 - Caixa de diálogo de um novo projeto.

O ambiente acompanha ferramentas de teste e alguns emuladores, possibilitando por exemplo, que seja testado um Web Site sem o desenvolvedor ter um Servidor Web em seu equipamento.

5.3.1.2 Banco de Dados SQL Server

Para armazenamento dos dados no Web Service foi utilizado o SQL Server. Desenvolvido pela Microsoft, o SQL Server é um sistema gerenciador de banco de dados relacional com arquitetura cliente/servidor.

Para garantir a estabilidade e escalabilidade em trabalhos de missões críticas, o SQL Server possui uma arquitetura robusta e preparada, permitindo que o software bata recordes digitais de armazenamento e gerenciamento de dados (GAVIN, 2004).

Dentre as diversas funcionalidades e características do SQL Server, é possível descrever algumas como o nativo suporte a XML, permitindo que solicitações tenham como retorno dados no formato XML, assim como é possível usar o padrão de dados para inserir, atualizar e excluir dados. Esta possibilidade de trabalhar com os dados diretamente no

formato XML facilita a integração com outros softwares.

No SQL Server também é possível desenvolver funções além das já fornecidas, utilizando a linguagem Transact-SQL é possível desenvolver funções que retornam um valor simples ou até mesmo uma tabela.

O software se destina a pequenas aplicações mono usuário ou até mesmo a aplicações WEB com milhares de transações simultâneas.

5.3.1.3 Emuladores

Durante o desenvolvimento deste trabalho foram usados emuladores que pudessem representar de forma real o comportamento dos aplicativos nos dispositvos móveis.

Para emular o uso de um *Pocket PC* foi utilizado o *Microsoft Device Emulator*, aplicativo para *desktop* usado para efetuar testes, debug e rodar o aplicativos em *Pocket PC*, *Pocket PC Phone Edition* e *SmatPhone* em tempo real sem a utilização de um dispositivo fisicamente ligado ao computador. Entre as principais características deste emulador estão o suporte a ferramenta de desenvolvimento VS.NET e a possibilidade de salvar o estado atual do dispositivo (*Saving State*), possibilitando configurá-lo apenas uma vez. Este emulador emula aplicativos nas versões Windows *Móbile*, Windows CE e Pocket PC 2003, ficando por escolha do desenvolvedor a versão que deseja executar.

Para emular o acesso Web por celular foram utilizados dois emuladores: *Microsoft Mobile Explorer* e *Openwave Simulator*. Ambos os emuladores tem como principal objetivo interpretar códigos no formato WML e mostrar o resultado em seus micro browsers. Além de interpretar o código WML é possível também trocar os *Skins* dos emuladores, podendo chegar o mais próximo da realidade de um determinado celular. Em especial o *Openwave Simulator* tem em seu emulador um *Console Emulator*, um console que permite observar todas as solicitações e o resultado das mesmas em código, além da forma em interface já interpretada.

Na figura 25 é mostrado os emuladores usados para o desenvolvimento deste trabalho.

Figura 25 - Emuladores usados neste trabalho

5.3.2 Implementação

A implementação deste trabalho representa o desenvolvimento e integração de três aplicativos distintos. O aplicativo para *Pocket PC* que representa o lado empresa coletando/atualizando dados, o aplicativo para Web Site representando o lado cliente consultado informações ligadas a ele, e o aplicativo Web Service responsável por esta intermediação entre os aplicativos que visão o lado empresa e outro visando o lado cliente.

Tanto o aplicativo para *Pocket PC*, como o Web Site, foram desenvolvidos separando o código responsável por manipulação dos dados, do código da interface. Já o Web Service, por não ter interfaces mas sim uma classe que representa todos os métodos disponíveis para invocação, foi desenvolvido com outras duas classes para separar de forma distinta os métodos/serviços que são pertinentes ao cliente ou a empresa.

Para o desenvolvimento do Web Service sobre a tecnologia .NET, foi criado um arquivo .asmx que é a referência para consulta de outros aplicativos. Aplicativos ou desenvolvedores que queiram obter informações, como o WSDL deste Web Service, devem solicitar este arquivo. Na figura 26 é mostrado conteúdo deste arquivo, que contém apenas uma tag.

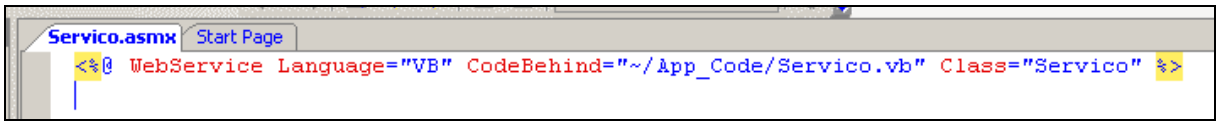


Figura 26 – Conteúdo do arquivo .asmx

Esta tag do arquivo .asmx, indica qual linguagem é usada na classe de serviço (*language*), o caminho do arquivo que contém a classe (*CodeBehind*), e qual o nome da classe que contém os métodos liberados para consulta ou envio de informações. É esta classe que contém os serviços disponíveis pelo Web Service.

Para que este arquivo seja executado basta fazer uma solicitação HTTP utilizando o endereço que representa o local onde este Web Service está hospedado, mais o nome do arquivo .asmx, exemplo: www.dominio.com.br/arquivo.asmx.

A classe que está referenciada no arquivo .asmx (Serviço.vb) tem como pré-requisito a importação de *Name Spaces* que contém classes auxiliares para a criação e uso do Web Service.

Todos os métodos desta classe que devem estar disponíveis para consulta e devem fazer parte do arquivo de definição do Web Service, terão como prefixo uma tag indicando que o método é um *Web Method*.

A figura 27 mostra um exemplo do conteúdo da classe Servico e alguns métodos com a tag *Web Method*.

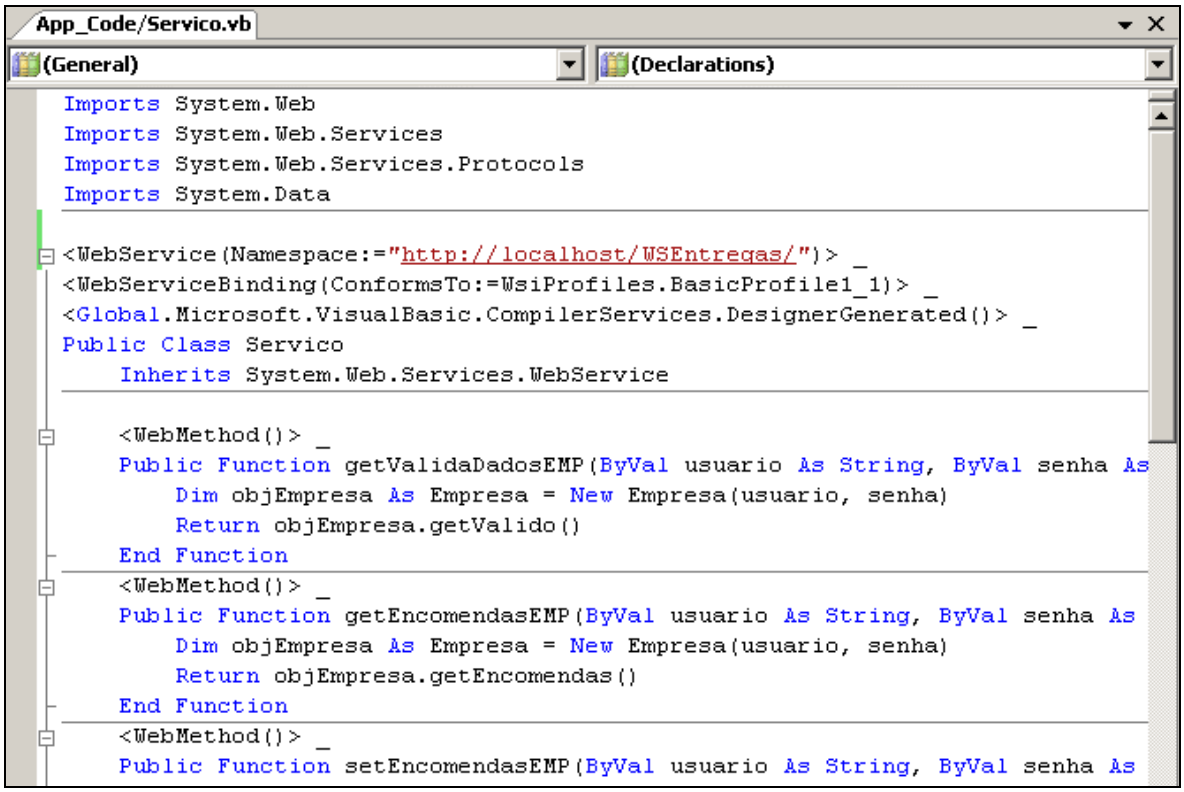


Figura 27 – Exemplo do código da classe Servico

Cada um dos *Web Methods* será definido no WSDL. Como pesquisado nas fundamentações teóricas o WSDL é responsável por descrever o Web Service e é formado por elementos que formam sua especificação. Cada método é especificado neste WSDL, mas para exemplificar o resultado, no quadro 2 é mostrado partes de cada um destes elementos com base na especificação do método “getValidaDadosEMP”.

```
<?xml version="1.0" encoding="utf-8" ?>
- <wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://localhost/WSEntregas/"
    xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://localhost/WSEntregas/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  ...
- <s:element name="getValidaDadosEMP">
  - <s:complexType>
    - <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="usuario" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="senha" type="s:string" />
      </s:sequence>
    </s:complexType>
  </s:element>
  ...
- <wsdl:message name="getValidaDadosEMPSoapIn">
    <wsdl:part name="parameters" element="tns:getValidaDadosEMP" />
  </wsdl:message>
- <wsdl:message name="getValidaDadosEMPSoapOut">
    <wsdl:part name="parameters" element="tns:getValidaDadosEMPResponse" />
  ...
- <wsdl:portType name="ServicoSoap">
  - <wsdl:operation name="getValidaDadosEMP">
      <wsdl:input message="tns:getValidaDadosEMPSoapIn" />
      <wsdl:output message="tns:getValidaDadosEMPSoapOut" />
    </wsdl:operation>
  ...
- <wsdl:binding name="ServicoSoap" type="tns:ServicoSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  - <wsdl:operation name="getValidaDadosEMP">
      <soap:operation soapAction="http://localhost/WSEntregas/getValidaDadosEMP" style="document" />
    - <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
    - <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  ...
- <wsdl:service name="Servico">
  - <wsdl:port name="ServicoSoap" binding="tns:ServicoSoap">
      <soap:address location="http://localhost:1033/WEB%20SERVICE/Servico.asmx" />
    </wsdl:port>
```

Quadro 2 – Exemplo de partes dos elementos do WSDL resultante

Com o uso desta especificação qualquer aplicativo poderá utilizar o Web Service.

Ao desenvolver o aplicativo do *Pocket PC* e do Web Site foi preciso adicionar ao projeto .NET a referência ao Web Service criado, para que com o WSDL seja possível saber qual a especificação dos métodos a serem consultados. Quando efetuada esta referência automaticamente uma classe denominada “proxy” é criada. Esta classe é responsável por

intermediar as solicitações ao Web Service, ela contém para cada método lido no WSDL métodos de invocação ao serviço.

Com esta classe intermediária gerada pela tecnologia, ao desenvolver é possível instanciar um objeto Web Service quase que de forma transparente, sem que se saiba que este método será na verdade uma solicitação HTTP a um servidor que então retornará o resultado, a não ser pelo fato de ter que haver um tratamento de erro verificando exceções como “System.Net.WebException”, para que seja possível saber quando a exceção reflete um erro na conexão com o Web Service.

No quadro 3 é mostrado como qualquer aplicativo pode efetuar referência ao Web Service na plataforma .NET. No exemplo é mostrada a referência ao método getValidaDadosEMP, método do Web Service que valida o usuário entregador.

```
Public Function validarDados() As Boolean
    Try
        Dim objWS As WebServiceEntrega.Servico = New WebServiceEntrega.Servico
        Return objWS.getValidaDadosEMP(usuario, senha)
    Catch ex As Exception
        Throw ex
    End Try
End Function
```

Quadro 3 - Exemplo invocação de um método no Web Service

A classe “proxy” (aqui chamada de “Servico”), como pode ser visto no quadro 3, invoca o Web Service enviando uma solicitação no padrão SOAP. Ao receber a resposta a classe extrai da resposta, também em SOAP, o valor recebido. Por ser um padrão bem específico, apenas mudando em cada solicitação os valores e parâmetros de cada método solicitado ao Web Service, o próprio .NET se encarrega de empacotar ou desempacotar o SOAP. Como estudado na fundamentação teórica, o SOAP é formado por um envelope onde os parâmetros são passados ou recebidos em seu corpo (*body*). Para o método getValidaDadosEMP (mostrado na figura anterior), a solicitação e a resposta em SOAP é mostrado como exemplo no quadro 4, onde é passado o valor do usuário “ivan.junges” e a senha “psw” criptografada, depois de receber a solicitação e processar, o método relacionado do Web Service devolve o valor “*true*”.

Solicitação em SOAP

```
POST /WSEntregas/Servico.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <getValidaDadosEMP xmlns="http://localhost/WSEntregas/">
      <usuario>ivan.junges</usuario>
      <senha>7bb483729b5a8e26f73e1831cde5b842</senha>
    </getValidaDadosEMP>
  </soap12:Body>
</soap12:Envelope>
```

Resposta em SOAP

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <getValidaDadosEMPResponse xmlns="http://localhost/WSEntregas/">
      <getValidaDadosEMPResult>TRUE</getValidaDadosEMPResult>
    </getValidaDadosEMPResponse>
  </soap12:Body>
</soap12:Envelope>
```

Quadro 4 - Exemplo de solicitação e resposta em SOAP

Como levantado na fundamentação teórica, o .NET *Compact Framework*, tem controles e funções em número reduzido se comparado com as disponíveis para a versão completa da plataforma. No desenvolvimento do aplicativo para *Pocket PC*, tomado como exemplo um aplicativo de entrega de encomendas, foi necessário desenvolver um controle para receber as assinaturas, atendendo assim ao requisito funcional que solicita permitir ao usuário-destinatário assinar no próprio *Pocket PC*. O que ocorre é a não existência de um componente nativo deste tipo no .NET *Compact Framework*.

Por não conter um componente específico, este problema foi solucionado criando em memória um *bitmap* que é criado e atualizado no momento em que o usuário clica (sobre uma área delimitada), mantém pressionado e ao arrastar, suas coordenadas são armazenadas em um *Array List* de pontos, que ao terminar, estes pontos são armazenados em *String*. Esta *String* com os pontos X e Y separados pelo caractere "|" (*pipe*) é enviada ao Web Service, que após a leitura completa dos pontos transforma o conteúdo até então armazenado em memória em imagem no disco do servidor.

No quadro 5 é mostrado um trecho do código onde o evento onMouseMove sobrepõe o

método da classe “Controle” para armazenar os pontos ao *Array List*.

```
Protected Overrides Sub OnMouseMove(ByVal e As MouseEventArgs)
    MyBase.OnMouseMove(e)
    'processa os movimentos caso o flag esteja ligado
    If (drawSign) Then
        'desenha a nova linha / segmento de memoria
        objGraphics.DrawLine(pen, lastPoint.X, lastPoint.Y, e.X, e.Y)
        pVector.Add(lastPoint.X & " " & lastPoint.Y & " " & e.X & " " & e.Y)
        'atualizar o ultimo ponto
        lastPoint.X = e.X
        lastPoint.Y = e.Y
        'mostra o bitmap atualizado
        Invalidate()
    End If
End Sub
```

Quadro 5 - Método que armazena as coordenadas da assinatura em memória.

No desenvolvimento do Web Site de consulta, que visa atender o lado cliente, foi utilizado a forma de desenvolvimento *Code Behind*, separando o HTML do código ASP.NET.

Como no desenvolvimento do aplicativo para o *Pocket PC*, as páginas fazem acesso ao Web Service, recebe resposta das solicitações e mostra ao usuário o conteúdo. Em especial a página WEB criada para acesso por celulares, foi desenvolvida utilizando os controles ASP.NET *Móbile Controls*. Estes controles usam uma assinatura própria da plataforma .NET, mas após seu processamento o retorno é em WML, linguagem interpretada pelos celulares. Para exemplificar, a figura 28 mostra um controle ASP.NET e o resultado após a solicitação em um console do emulador de celular.

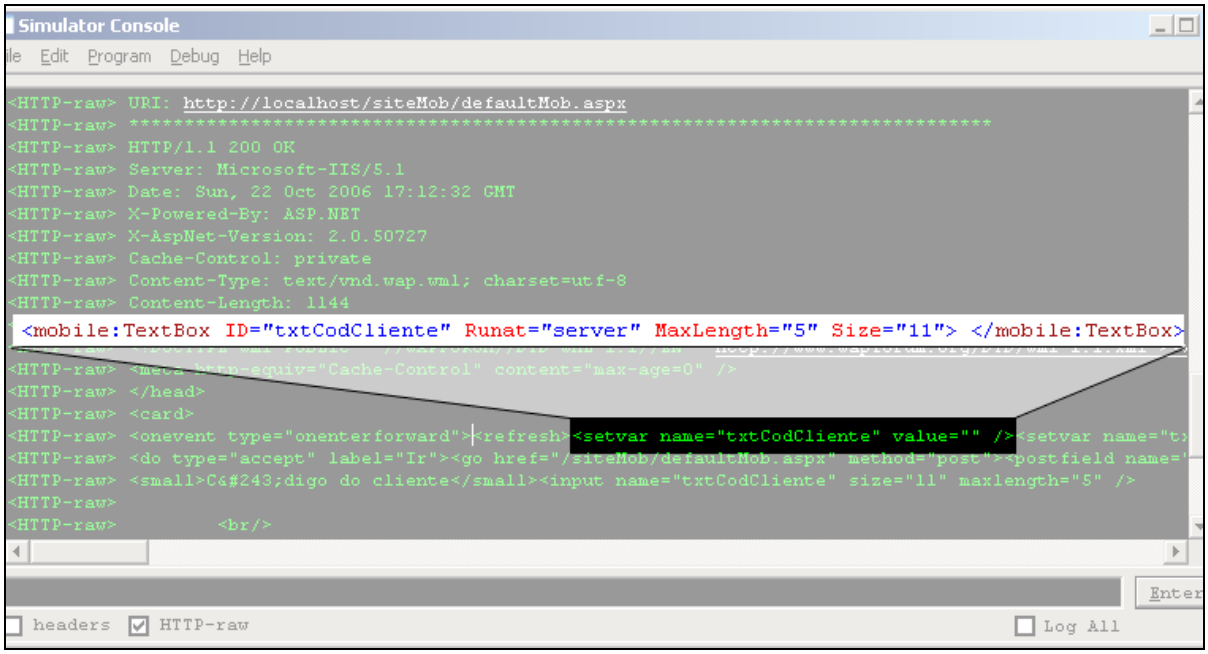


Figura 28 - Resultado do uso do controle ASP.NET *Móbile Controls*

Apenas para exemplificar esta situação, a imagem mostra o controle código do cliente

gerado para WML.

5.3.3 Operacionalidade da implementação

Como o propósito deste trabalho visa a operacionalidade e o estudo das tecnologias que possibilitam a concretização desta idéia (uma empresa coletar informações com o *Pocket PC*, enviar a um Web Service e permitir a consulta por Web Site tanto por *desktop* como por celular), não foram desenvolvidas telas de cadastramento de clientes, empresas, entregas entre outras situações que não são pertinentes ao objetivo deste trabalho. Assim, para efetuar a execução e teste deste trabalho, existe como pré-requisito a base de dados do Web Service estar previamente populada. A seguir será exemplificado cada situação separado por aplicativo.

5.3.3.1 Aplicativo visando a empresa na coleta de dados (Aplicativo do *Pocket PC*).

O primeiro passo para o aplicativo do *Pocket PC* é efetuar o login. A figura 29 mostra um exemplo de acesso na tela de login.

Figura 29 - Tela de login

A tela de login faz a verificação da autenticidade dos dados junto ao Web Service e envia o usuário a tela principal.

Na tela principal, a primeira opção permitida ao usuário é a de receber a lista de entregas. A figura 30 mostra a tela principal recebendo a lista de entregas.

Figura 30 - Recebendo lista de entregas

Após receber do Web Service a lista de entregas, as informações na tela principal são atualizadas, mostrando a quantidade de entregas a serem efetuadas.

Com a listagem carregada no *Pocket PC* é possível utilizar o aplicativo de forma desconectada, pois a próxima operação é a de efetuar entregas.

Clicando sobre o botão efetuar entrega, é apresentada uma tela onde o usuário deverá informar o número da entrega a ser efetuada. Na figura 31 pode ser observada a tela de entrega.

Figura 31 - Tela efetuar entrega.

Depois de verificado os dados da entrega, rolando a barra de rolagem o usuário encontra duas opções de ação, a de efetuar ou não a entrega. Caso decida por efetuar a entrega, um controle solicitando a assinatura é mostrado ao usuário. Caso contrário será mostrada uma lista de motivos para a não entrega. Este processo é exemplificado na figura 32.

Figura 32 - Solicitando assinatura ao destinatário.

Com a assinatura do destinatário, o usuário pode salvar a entrega, armazenando assim

os dados no *Pocket PC*.

Este processo é feito para todas as entregas, e a cada entrega efetuada os números indicando a quantidade de entregas e a quantidade de entregas efetuadas vão sendo atualizados.

Com as entregas efetuadas, chega o momento de atualizar as informações e enviar as assinaturas coletadas ao Web Service. Para isso, é preciso pressionar o botão “Enviar Lista de Entregas” na tela principal do aplicativo. A figura 33 mostra a tela informando o envio com sucesso.

Figura 33 - Envio com sucesso da lista de entregas

Com este processo de envio, conclui-se o processo de recepção e envio de dados ao Web Service do lado Empresa. As informações estão atualizadas para consulta pelo lado Cliente.

5.3.3.2 Aplicativo visando o lado cliente (Web Site destinado à consulta)

O usuário cliente, tem duas formas de consultar se sua entrega foi efetuada com sucesso, por celular ou pelo seu *desktop*. Primeiramente será apresentada a consulta por seu *desktop* com acesso a internet.

Ao acessar o endereço do site onde está hospedado o aplicativo, o usuário verá uma

tela de acesso igual à mostrada na figura 34.

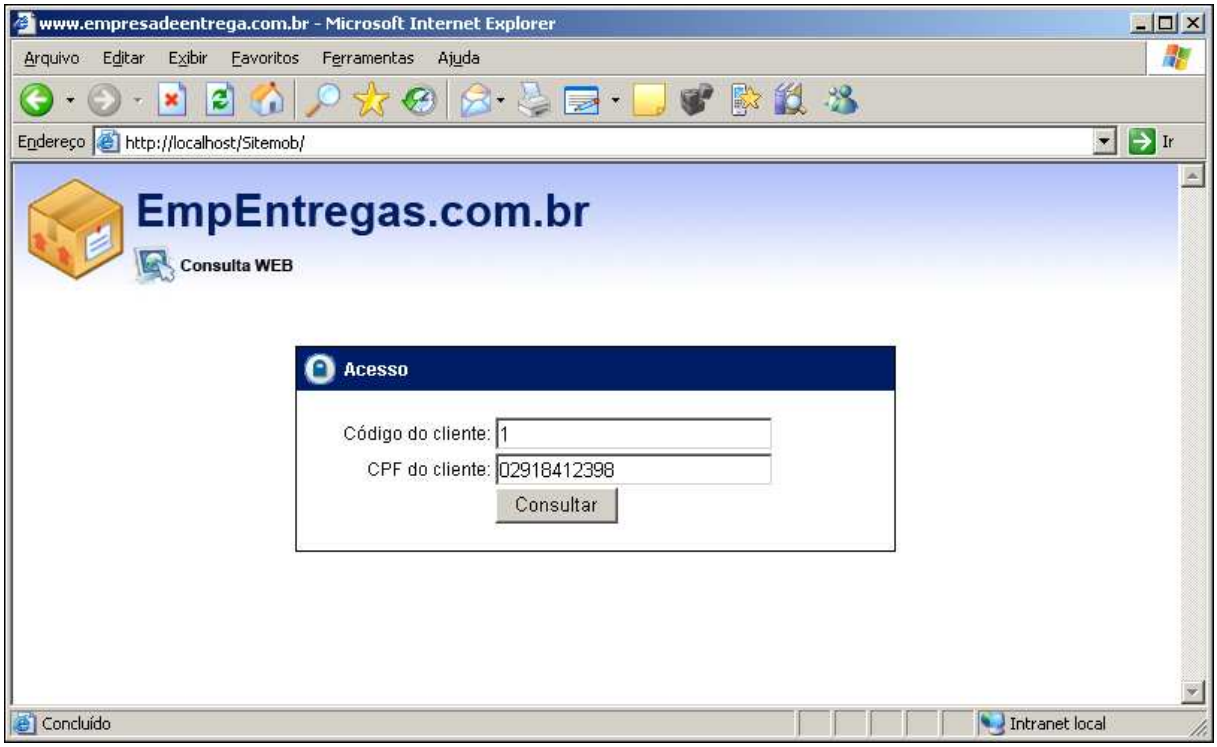


Figura 34 - Tela acesso Web Site

Após validar junto ao Web Service, o site é redirecionado a tela “Lista de Entregas” com todas as entregas ligadas a este cliente. Na figura 35 é possível observar a tela.

Figura 35 - Tela lista de entregas

Novamente a consulta se faz possível com acesso ao Web Service, que foi solicitado e

retornou a listagem vista na tela “Lista de Entregas”.

Clicando sobre o link “Detalhes” de uma determinada entrega, visualiza-se informações relacionadas a ela. Caso tenha sido entregue é possível ver a assinatura do destinatário. Na figura 36 é observado os detalhes de uma entrega que foi efetuada.

Figura 36 - Tela de detalhes da entrega

Além da solicitação dos dados, a própria imagem da assinatura foi solicitada e retornada pelo Web Service. Como a imagem estava armazenada em disco o retorno foi em *bytes*.

Além dos detalhes, é possível ver o histórico da entrega. Caso a entrega tenha tido mais de uma tentativa sem sucesso, é no histórico que esta informação é listada.

O usuário adepto ao uso da internet por celular pode acessar sua entrega com o mesmo endereço de domínio, mudando apenas o final do endereço para “/DefaultMob.aspx”. A figura 37 ilustra a tela de acesso a página para celular.

Figura 37 - Tela de acesso por celular

Após preencher os campos e clicar em consultar, o usuário é redirecionado para a tela de detalhes da entrega que foi solicitada. Como todas as outras telas, esta também não há qualquer acesso direto a base de dados, a não ser a consulta ao Web Service. A figura 38 ilustra o detalhe da entrega, mostrando a assinatura do destinatário.

Figura 38 - Tela de detalhe da entrega para celular

Caso a entrega solicitada não tivesse sido efetuada, a assinatura não estaria aparecendo e a entrega estaria com outro status.

5.4 RESULTADOS E DISCUSSÃO

Como observado na fundamentação teórica, as classes de acesso a dados, gerenciamento de tabela em memória, controle do XML, classes usadas para manipulação de arquivos e imagens, são iguais tanto no desenvolvimento do aplicativo para o *Pocket PC*, como no Web Site e também no Web Service. Com isso é possível confirmar o fato de poder reaproveitar o conhecimento em todos os segmentos onde a plataforma .NET possa ser instalada.

Durante o desenvolvimento do aplicativo *Pocket PC*, foi observada uma grande dificuldade em testar e efetuar *debug* do código, pois a ferramenta de desenvolvimento VS.NET exige muito do computador. Os emuladores que acompanham a ferramenta exigem o uso de muita memória e o processo se torna crítico quando além do teste há a necessidade de "*debug*" passo-a-passo.

A criação de um controle não existente de forma nativa no .NET *Compact Framework* para atender um dos objetivos deste trabalho, comprova a possibilidade de expansão da tecnologia por parte do desenvolvedor. O controle de assinatura apesar de ter se mostrado funcional, durante os testes se mostrou lento pela demora na atualização do *bitmap* em memória.

Com a criação do Web Site usando controles e a forma de desenvolvimento ASP.NET, foi possível observar que as mudanças foram revolucionarias em comparação ao antigo ASP. A forma de desenvolvimento atual chega muito próxima a utilizada para *desktop*, mas ao mesmo tempo houve aumento na dificuldade de aprendizagem do programador que conhece outra tecnologia WEB, pois além de uma nova linguagem, o programador deve também conhecer os novos controles existente no ASP.NET para usufruir da tecnologia, não bastando apenas herdar o conhecimento do HTML.

Praticamente da mesma forma que foi criado o site convencional com ASP.NET foi possível criar um site para celulares, a não ser pela grande redução de componentes disponíveis e a preocupação que é preciso ter com alocação de memória em sessões. Durante o desenvolvimento houve dificuldade no uso de sessões, pois em determinados emuladores de celular o uso deste recurso não ocorreu com sucesso.

No quadro 5 é feita uma comparação com os trabalhos correlatos, levando em consideração algumas características deste trabalho.

Acadêmicos / Item	Este trabalho	Gavin (2004)	Ramos (2004)	Schefer (2004)
Plataforma de desenvolvimento	.NET	.NET	.NET	JAVA
Dispositivo Móvel	Pocket PC e Celular	Pocket PC	SmartPhone	Celular
Uso de Web Service	Sim	Sim	Sim	Não
Criação de Web Site	Acesso por desktop e por celular	Não	Não	Não
Criação de um componente/controle para dispositivo móvel.	Sim. Controle de captura de assinatura	Não	Não	Não
Objetivo	Desenvolver uma arquitetura “Empresa” e “Cliente” com uso de dispositivos móveis e Web Service. Usando como exemplo a área de entrega de encomendas.	Protótipo de um sistema CRM para dispositivo móvel para atender o segmento industrial metalúrgico	Segurança na transmissão de mensagens para dispositivos móveis através de criptografia	Coletar Informações e envia-lás pra um desktop para futura análise dos dados

Quadro 5 - Comparação deste trabalho com trabalhos correlatos

Em relação à comparação apresentada no quadro 5, é importante salientar que apesar de outros dois trabalhos correlatos terem usado Web Service, apenas este efetuou um estudo aprofundado sobre a tecnologia.

6 CONCLUSÕES

Este trabalho atingiu seu objetivo de desenvolver um aplicativo para dispositivo móvel para exemplificar a idéia de uma empresa coletando dados, um WEB Site que pode ser acessado por *desktop* ou celular visando a massa de pessoas (clientes) que necessitam de serviços que possam ser utilizados por seus celulares, e um Web Service para possibilitar a integração das informações. Além do seu objetivo este trabalho também concluiu o desafio de desenvolver todos os três aplicativos de diferente plataforma e hardware com a mesma tecnologia de desenvolvimento.

O desenvolvimento deste trabalho comprovou a viabilidade do uso de dispositivos móveis para ajudar empresas com trabalhos em campo, além de comprovar a facilidade e flexibilidade que é possível oferecer aos seus clientes usuários da internet, seja por computador ou celular. Os estudos realizados e a arquitetura desenvolvida neste trabalho podem ser usados como base para análise e estudo do desenvolvimento de softwares que atendam outras áreas (além da área de entregas, usada como exemplo), mas que também buscam a integração de diferentes arquiteturas.

Com a pesquisa realizada sobre a tecnologia .NET em diferentes plataformas (dispositivo móvel, Web Service e Web Site) juntamente com o desenvolvimento prático deste trabalho, concluiu-se que a plataforma atende as expectativas em relação a portabilidade. Apesar de haver algumas mudanças no paradigma de desenvolvimento ao mudar de arquitetura, a tecnologia .NET apresentou grande interoperabilidade.

Quanto ao uso do Web Service, a tecnologia se mostrou perfeita para integração de softwares diferentes e hardwares fisicamente separados. Com a leitura do WSDL criado, qualquer software pode estar acessando este trabalho e consultando as informações disponíveis, pois o mesmo segue o padrão estudado nas fundamentações teóricas.

Como uso da área de entregas foi usada apenas como exemplo para o desenvolvimento prático, não houve uma grande preocupação com regras de negócio, mas sim com o estudo e integração das diferentes tecnologias e plataformas. Sendo assim, a exploração e a busca por sanar a necessidade de mercado levantada na introdução deste trabalho se torna genérica e não exclusivamente de uma área de atuação.

6.1 EXTENSÕES

Este trabalho ainda pode ser explorado visando à integração de outras tecnologias e Web Services além dos já explorados neste trabalho.

Uma sugestão de extensão para este trabalho ainda usando como exemplo a área de entregas, é a integração do aplicativo *Pocket PC* com *Global Position System* (GPS) e por sua vez o Web Service ao receber informações da posição do *Pocket PC*, fazer consulta a um Web Service com serviços de mapas, como o Microsoft *MapPoint* ou *GoogleMap*.

Outra sugestão seria explorar o uso de outros Web Services, estudando uma solução quando houvesse uma grande quantidade de requisições a um determinado serviço, possibilitando a distribuição entre outros Web Services.

Além destas sugestões, há ainda a possibilidade de tornar este trabalho um sistema de informação completo, usufruindo da arquitetura deste trabalho mas explorando a regra de negócios de uma empresa de entregas.

REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDER, John; HOLLIS, Billy. **Desenvolvendo aplicações web com Visual Basic.NET e ASP.NET**. Tradução Bázan Tecnologia e Lingüistica. São Paulo: Berkeley, 2002.

BARWELL, Fred et al. **Profissional Visual Basic.NET**. 2. ed. Tradução Asiovaldo Griesi. São Paulo: Pearson Education do Brasil, 2004.

BEHERA , Gitansu. **Signature capture in handheld or Pocket PC with C#**. 2004. Disponível em: <http://www.codeproject.com/Purgatory/SignatureCapture_PocketPC.asp?print=true>. Acesso em: 20 set. 2006.

BORGES JR., Maurício P. **Aplicativos móveis:** aplicativos para dispositivos móveis usando C# .NET com a ferramenta Visual Studio .NET e com banco de dados MySQL e SQL Server. Rio de Janeiro: Ciência Moderna, 2005.

CEMBRANELLI, Felipe. **ASP.NET:** guia do desenvolvedor. São Paulo: Novatec, 2003.

CHERRY, Michael; DEMICHILLIE, Greg. **A plataforma de desenvolvimento .NET**. [S.l.]: Microsoft, c2002.

CONALLEN, Jim. **Desenvolvendo aplicações web com UML**. 2. ed. Tradução Altair D. C. de Moraes; Cláudio B. Dias. Rio de Janeiro: Campus, 2003.

D'ANGELO, Fernando. **Microsoft .NET a plataforma Java da Microsoft.** [São Paulo], 2003. Disponível em: <http://www.aspbrasil.com.br/conteudo/detalhesCompleta.aspx?codConteudo=3306>. Acesso em: 23 ago. 2006.

DEITEL, Harvey M. et al. **Visual Basic.NET:** como programar. 2. ed. Tradução Célia Yumi Okano Taniwaki. São Paulo: Pearson Education do Brasil, 2004.

GALVIN, Deleon. **Protótipo de sistema CRM para dispositivos móveis utilizando a tecnologia .NET.** 2004. 90 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

GUIMARÃES, Renato. **ASP.NET – uma mudança radical no desenvolvimento web**. [São Paulo], 2003. Disponível em: <http://www.imasters.com.br/artigo/1624>. Acesso em: 28 ago. 2006.

HADDAD, Renato. **Saiba porquê desenvolver aplicações para telefones celulares com .NET.** [São Paulo], 2003. Disponível em: <http://www.linhadecodigo.com.br/artigos.asp?id_ac=171>. Acesso em: 14 abr. 2006.

LEOPOLDO, Marcus R. B. **Entendendo o simple object access protocol (SOAP).** [São Paulo], 2003. Disponível em: <http://www.msdnbrasil.com.br/secure/sharepedia/arquivos/SOAP.pdf >. Acesso em: 02 out. 2006.

MIRANDA, Luiz H. **Introdução ao mundo móvel.** Goiânia, [2005]. Disponível em: <http://www.devgoiania.net/pocket.aspx>. Acesso em: 10 out. 2006.

MICROSOFT CORPORATION. **Mobile application architecture animated presentation.** [S.l.]: c2006. Disponível em: <http://www.asp.net/mobile/flasharchitecture.aspx?tabindex=3&tabID=44>. Acesso em: 10 out. 2006.

MICROSOFT CORPORATION. **Microsoft .NET Framework e aplicativos web**. Tradução Izabel C. M. Santos, Ana B. Tavares. Rio de Janeiro: Campus, 2001.

MOSIMANN NETTO, Max. **Microsoft .NET Compact Framework :** conheça a plataforma para dispositivos móveis criada pela Microsoft. [São Paulo], 2005. Disponível em: <http://www.linhadecodigo.com.br/artigos.asp?id_ac=646>. Acesso em: 13 mar. 2006.

MOSIMANN NETTO, Max. **Mobilidade e dispositivos moveis**. [São Paulo], 2004. Disponível em: < http://www.linhadecodigo.com.br/artigos.asp?id_ac=206>. Acesso em: 23 maio 2006.

PEKUS. **Dispositivos móveis.** São Paulo, [2004]. Disponível em: <http://www.pekus.com.br/palmtops.htm>. Acesso em: 28 maio 2006.

RAMOS, Robson. **Protótipo de software para envio de mensagens criptografadas para um dispositivo móvel utilizando a plataforma .NET**. 2004. 51 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

RECKZIEGEL, Mauricio. **Entendendo os web services**. [São Paulo], 2006. Disponível em: <http://www.imasters.com.br/artigo/4245/webservices/entendendo_os_webservices/ >. Acesso em: 31 ago. 2006.

ROMELLI, Maycol S. **Desenvolvendo aplicações para Pocket PC utilizando SQL Server CE.** [São Paulo], 2002. Disponível em: <http://www.linhadecodigo.com.br/artigos.asp?id_ac=57>. Acesso em: 05 out. 2006.

SCHAEFER, Carine. **Protótipo de aplicativo para transmissão de dados a partir de dispositivos móveis aplicado a uma empresa de transportes**. 2004. 53 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SEELY, Scott. **SOAP**: cross plataform web service development using XML. Upper Saddle River: Pretice Hall PTR, 2002.

SHODJAI, Payam. **Serviços da web e a plataforma Microsoft**. [São Paulo], 2006. Disponível em: <http://www.microsoft.com/brasil/msdn/Tecnologias/aspnet/EstruturaInterna.mspx>. Acesso em: 31 ago. 2006

SNELL, James; TIDWELL, Doug; KULCHENKO, Pavel. **Programing web services with SOAP.** Sebastopol: O'Reilly & Associates, 2002.

TARIFA, Alexandre; FACUNDE, Emerson; GARCIA, Marcus. **Visual Basic .NET:** desenvolvendo uma aplicação comercial**.** Rio de Janeiro: Brasport Livros e Multimídia, 2005.

TOLOMELLI, Leonardo. **Entrevista exclusiva: Leonardo Tolomelli:** mercado. [São Paulo], 2005. Disponível em: <http://www.imasters.com.br/artigo/3264/mercado/entrevista_exclusiva_leonardo_tolomelli>. Acesso em: 28 maio 2006.