

UNIVERSIDADE LUTERANA DO BRASIL

ULBRA – *CAMPUS* GUAÍBA

CURSO DE SISTEMAS DE INFORMAÇÃO



**VERIFICAR A PORTABILIDADE DE APLICAÇÕES
.NET QUANDO EXECUTADAS EM AMBIENTES
DIVERSOS (WINDOWS, WEB E MOBILE).**

TRABALHO DE CONCLUSÃO DE CURSO I

ALEXANDRE COUTINHO EVANGELISTA

Luiz Gustavo Galves Mählmann
Orientador

Guaíba, junho de 2008.

DADOS DE IDENTIFICAÇÃO

Acadêmico(a): Alexandre Coutinho Evangelista

E-mail: alex.evangelista@gmail.com

Professor(a) Orientador(a): Luiz Gustavo Galves Mählmann

E-mail: mahlmann@gmail.com

Título do Projeto: Verificar a portabilidade de aplicações .NET quando executadas em ambientes diversos (Windows, Web, Mobile).

Período de realização: 14/03/2008 – 30/06/2008

SUMÁRIO

1	DEFINIÇÃO DO TEMA	6
1.1	Tema.....	6
1.2	Delimitação do Tema	6
2	PROBLEMA DE PESQUISA	6
3	HIPÓTESES DE SOLUÇÃO	7
4	OBJETIVOS	7
5	JUSTIFICATIVA	8
6	FUNDAMENTAÇÃO TEÓRICA	8
6.1	Linguagens de programação anteriores a plataforma .NET	9
6.1.1	<i>Linguagens de Baixo Nível</i>	9
6.1.2	<i>Linguagens de Alto Nível</i>	10
6.2	A Plataforma JAVA	11
6.3	A Plataforma .NET	12
6.4	.NET Framework	12
6.4.1	<i>Versões do .NET Framework</i>	13
6.4.2	<i>CLR - Common Language Runtime</i>	13
6.4.3	<i>CTS – Common Type System</i>	14
6.4.4	<i>CLS – Common Language Specification</i>	14
6.4.5	<i>BCL – Base Class Library</i>	14
6.4.6	<i>MSIL – Microsoft Intermediate Language</i>	15
6.4.7	<i>Metadata</i>	15
6.4.8	<i>JIT – “Just in Time”</i>	15
6.4.9	<i>GC – Garbage Collector</i>	16
6.5	.NET Compact Framework	16
6.6	Linguagens Suportadas pela Plataforma .NET	16
6.7	Web Services	17
6.7.1	<i>SOAP – Simple Object Access Protocol</i>	17
6.7.2	<i>XML – eXtensible Markup Language</i>	18
6.7.3	<i>WSDL – Web Service Description Language</i>	19
7	METODOLOGIA	20
7.1	MVC – Model-View-Controller	20

7.2 Diagrama de Entidade-Relacionamento.....	21
7.3 UML	21
7.4 Analisando as Versões do .NET Framework quanto a sua Portabilidade..	22
7.4.1 <i>Portabilidade</i>	22
7.4.2 <i>Diferenças entre os ambientes.....</i>	23
7.4.3 <i>Similaridades entre os ambientes</i>	23
7.5 Estrutura de Desenvolvimento da Aplicação.....	23
7.6 Tecnologias Utilizadas.....	26
8 RESULTADOS	27
9 TRABALHO DE CONCLUSÃO DE CURSO – II	32
10 REFERÊNCIAS	32

ÍNDICE DE FIGURAS

FIGURA 1: PROCESSO DE COMPILAÇÃO EM APLICAÇÕES .NET.....	13
FIGURA 2: WEB SERVICE "HELLOWORLD".....	17
FIGURA 3: SOLICITAÇÃO E REPOSTA SOAP DO MÉTODO "HELLOWORLD".....	18
FIGURA 4: RETORNO DO MÉTODO "HELLOWORLD".....	19
FIGURA 5: FASES DE FUNCIONAMENTO DOS WEB SERVICES.....	19
FIGURA 6: COMUNICAÇÃO ENTRE AS CAMADAS DO MVC.....	21
FIGURA 7: ESTRUTURA DE CAMADAS DA APLICAÇÃO.....	24
FIGURA 8: ESTRUTURA DO MODELO NO MICROSOFT VISUAL STUDIO 2008.....	25
FIGURA 9: ESTRUTURA DO ASP.NET MVC FRAMEWORK NO MICROSOFT VISUAL STUDIO 2008.....	27
FIGURA 10: CADASTRO DE PESSOAS – AMBIENTE WINDOWS.....	28
FIGURA 11: CADASTRO DE PESSOAS – AMBIENTE WEB.....	28
FIGURA 12: CADASTRO DE PESSOAS – AMBIENTE MOBILE PARA SMARTPHONE.....	29
FIGURA 13: CADASTRO DE PESSOAS – AMBIENTE MOBILE PARA POCKETPC.....	29
FIGURA 14: FIGURA – EXEMPLO DE CÓDIGO DO AMBIENTE WINDOWS.....	30
FIGURA 15: WEB SERVICE WSPESSOAS COM SUA LISTA DE MÉTODOS.....	30
FIGURA 16: RETORNO DA REQUISIÇÃO DO MÉTODO "LISTA" NO FORMATO XML.....	31

1 DEFINIÇÃO DO TEMA

1.1 Tema

O tema proposto envolve a verificação da portabilidade que uma aplicação pode ter quando for executada em mais de um ambiente (*Windows, Web e Mobile*), utilizando a mesma plataforma de desenvolvimento, abordando as respectivas similaridades e diferenças entre os mesmos.

1.2 Delimitação do Tema

O escopo do trabalho envolve a verificação da portabilidade de um único sistema, utilizando mais de um ambiente para execução, suas características, similaridades, diferenças, compatibilidades e incompatibilidades.

No ambiente *Desktop*, será utilizada a plataforma *Windows*, assim como no *Web*; já no *Móvel*, será utilizada a plataforma *Windows Mobile* (YANG, 2007).

Será feito um comparativo a respeito da evolução dos *frameworks* (THAI, 2001) existentes na plataforma .NET (THAI, 2001), através da análise das respectivas características, atualizações e diferenças até a versão 3.5, disponível no ambiente de desenvolvimento Microsoft Visual Studio 2008 (TROELSEN, 2007).

Será desenvolvida uma aplicação para demonstrar os estudos realizados. Tanto a aplicação, quanto os estudos serão baseados na linguagem de programação C#.NET (TROELSEN, 2007).

2 PROBLEMA DE PESQUISA

Atualmente, desenvolver uma aplicação para o ambiente *Windows* com diversas funcionalidades e disponibilizar um módulo desta aplicação para que seja executada em outro ambiente, não é nenhuma novidade. Porém, essa situação deve ocasionar um retrabalho considerável para o desenvolvedor, inviabilizando a reutilização de código por se tratar de ambientes e até mesmo em utilizar linguagens de programação diferentes.

Com isso, dificilmente, o profissional que desenvolveu a aplicação *Windows* será o mesmo que desenvolverá o módulo para *Web* ou para um dispositivo móvel.

Diante deste cenário, a proposta deste artigo é o desenvolvimento de uma única solução utilizando as potencialidades da plataforma .NET, além da descrição da solução que será desenvolvida, será visto como a plataforma .NET funciona e a maneira que a integração entre os ambientes poderá ser feita.

3 HIPÓTESES DE SOLUÇÃO

O presente estudo precisa avaliar as seguintes hipóteses para chegar a uma solução adequada:

- Hipótese 1: A aplicação desenvolvida é portátil entre os ambientes *Windows*, *Web* e *Mobile*, utilizando o mesmo *framework*, pois as classes disponíveis nas duas variações de *framework* são as mesmas;
- Hipótese 2: A aplicação desenvolvida é portátil para os ambientes *Windows*, *Web* e *Mobile*, utilizando o .NET *Framework* (THAI, 2001) além do *Compact Framework* (YANG, 2007), pois algumas classes de determinado *framework* não podem ser referenciadas para a utilização em outro;
- Hipótese 3: É necessário que seja desenvolvida uma aplicação para cada ambiente, pois algumas classes de determinado *framework* não podem ser referenciadas para a utilização em outro, além disso, existe a incompatibilidade de componentes.
- Hipótese 4: É necessário que sejam desenvolvidas as interfaces para cada ambiente, sendo utilizada uma camada de controle e outra de regras de negócio, sendo assim, essas camadas podem ser acessadas por todos os ambientes.

4 OBJETIVOS

O objetivo deste trabalho é realizar um estudo que verifique a portabilidade de uma aplicação, para que a mesma possa ser executada em mais de um ambiente, facilitando a reutilização de código além de anular o retrabalho, pois a aplicação será portátil para que seja executada em outros ambientes.

Especificamente, a pesquisa irá focar a eficiência e eficácia do emprego de uma linguagem de programação em uma única aplicação, mas que essa seja executada independente da plataforma escolhida seja ela no próprio *desktop*, através de um navegador de internet ou até mesmo de um dispositivo móvel.

5 JUSTIFICATIVA

Com a constante evolução tecnológica em que vivemos, onde, semanalmente, novidades não param de surgir, novos sistemas vêm facilitando nosso dia-a-dia, por isso precisamos ficar sempre atentos ao que está acontecendo e o que está para acontecer.

Existem aplicações em diversas áreas de atuação muito bem consolidadas no mercado, os dispositivos móveis já não são mais uma novidade e sim uma realidade que vêm tomando uma dimensão cada vez maior. Sua infinidade de recursos faz com que aplicações possam ser usadas em qualquer local, sem que seja necessário o usuário estar em casa ou no seu trabalho.

A maior motivação para a realização deste trabalho é poder unir os pontos fortes que cada ambiente proporciona em uma única aplicação. Desse modo, será possível desenvolver uma aplicação e adaptar a sua adequação a outros ambientes, propiciando uma economia de esforço considerável para o desenvolvedor.

Além disso, espera-se que este trabalho possa contribuir para que novas soluções nesse âmbito possam surgir, agregando ambientes e diferentes plataformas para que não nos tornemos dependentes de um determinado *hardware* e tecnologia.

6 FUNDAMENTAÇÃO TEÓRICA

Para embasar as propostas que serão apresentadas neste trabalho, foi levantado o estado da arte sobre a plataforma .NET, os *frameworks* que a plataforma possui, além de um detalhamento sobre seus componentes e a utilização de *Web Services* (SHORT, 2002).

6.1 Linguagens de programação anteriores a plataforma .NET

Antes de falar sobre a plataforma em questão deste trabalho, não se pode esquecer como tudo isso começou. Com isso, será possível falar sobre as evoluções que ocorreram em relação às linguagens de programação até chegar ao cenário atual.

Hoje em dia, existe um grande número de linguagens de programação, algumas já extintas e outras ainda utilizadas. Algumas são interpretadas de forma que o computador entende diretamente a instrução, já outras, passam por algumas etapas intermediárias de forma que o computador entenda as instruções.

Podemos classificar as linguagens de programação em duas categorias, as linguagens de baixo e alto nível. (DEITEL, 2001)

6.1.1 Linguagens de Baixo Nível

São as linguagens de programação mais próximas da linguagem da máquina, que é a única que o processador consegue entender. Cada computador tem a sua própria linguagem de máquina, já definida pela arquitetura do *hardware*. O código das linguagens de baixo nível consiste em uma seqüência de números, possui dependência do *hardware*, com isso um determinado código só poderá ser executado para um determinado tipo de equipamento. Diante dessas características, até mesmo um programa editor de texto não conseguia entender um código escrito neste tipo de linguagem, inviabilizando completamente a compreensão do código para qualquer pessoa.

Posteriormente, surge a segunda geração das linguagens de baixo nível. Da linguagem natural, que era apenas uma seqüência de números, que poderiam ser utilizadas abreviações de palavras, fazendo, basicamente, as mesmas operações da geração anterior, porém, ela se tornou mais fácil de compreender, conhecida como linguagem *Assembly*. (DEITEL, 2001)

Para que os programas desenvolvidos na linguagem *Assembly* fossem entendidos pelo computador, era necessária a utilização de um tradutor, conhecido como *Assembler*, ele fazia a conversão do código feito em *Assembly* para a linguagem de máquina.

Este processo facilitou a compreensão do código, mas ele apenas substituiu a utilização de números, para abreviações das operações que poderiam ser feitas,

contudo exigia ainda uma série de instruções de código para executar uma simples tarefa.

6.1.2 Linguagens de Alto Nível

Como foi dito acima, o número de instruções que era necessário desenvolver para executar simples operações não acompanhava a velocidade em que os computadores evoluíam, sendo assim era necessário uma maneira em que o processo de desenvolvimento fosse mais rápido e claro para o desenvolvedor. As linguagens de alto nível se diferenciavam justamente por essas razões, pois simples instruções poderiam refletir em operações significativas, que ao invés de abreviações, era possível utilizar expressões da língua inglesa, proporcionando maior clareza na hora de escrever e entender o código.

O programa que fazia a conversão da linguagem de alto nível para a linguagem de máquina era chamado de compilador. Compilar um programa exigia mais tempo, ainda mais por que alterações no código poderiam ser necessárias durante o processo de desenvolvimento. Com a utilização de interpretadores, eles podiam executar o código diretamente, sem a necessidade de compilá-los, o código era compilado apenas no final do seu desenvolvimento.

Existem centenas de linguagens de programação, sendo que algumas são mais conhecidas como o Fortran (*Formula Translator*) usado para realizar complexos cálculos matemáticos, logo depois veio o COBOL (*Common Business Oriented Language*), utilizado em aplicações comerciais, que exigiam precisão e eficiência na manipulação de grande quantidade de dados. A linguagem Pascal surgiu no final dos anos 60, e seu intuito era a utilização no âmbito acadêmico, devido sua facilidade de aprendizagem. (DEITEL, 2001)

Em 1973 surge a linguagem C, embora tenha utilizado conceitos de linguagens já existentes, ela foi reconhecida como a linguagem de desenvolvimento do sistema operacional UNIX, mas, hoje em dia, grande parte dos sistemas operacionais existentes foi desenvolvida em C. Outra vantagem desta linguagem é a independência de seu código em relação ao *hardware*, provando a importância que portabilidade de uma linguagem de programação pode proporcionar. (DEITEL, 2001)

No início dos anos 80 surge um aprimoramento da linguagem C, chamada de C++, além de melhorias significativas, foi proporcionado algo de extrema importância que é a programação orientada a objetos. O C++ é uma linguagem híbrida, assim

pode ser utilizado o método de programação da linguagem C, orientada a objetos ou até mesmo as duas maneiras em um mesmo programa. (DEITEL, 2001)

Construir aplicações em pouco tempo, com baixo custo e sem falhas é uma tarefa difícil, pois as exigências de novas e poderosas soluções estão em constante crescimento. A utilização da programação orientada a objetos é uma tendência natural, por se tratar de uma abordagem fácil de entender, o desenvolvimento e suas respectivas correções são facilitadas, resultando um ganho considerável de produtividade.

No início da década de 90 a Microsoft apresentou a linguagem *Visual Basic*, sua origem é a linguagem BASIC (*Beginner's All-Purpose Symbolic Instruction Code*), que na década de 60 tinha como objetivo fazer com que iniciantes na área de desenvolvimento se familiarizassem com a programação. (DEITEL, 2001)

O *Visual Basic* foi criado para facilitar o desenvolvimento de aplicações para o sistema operacional *Windows*. Mesmo que a origem da linguagem tenha sido o BASIC, elas são bem distintas, pois o *Visual Basic* oferecia o desenvolvimento de interfaces gráficas, tratamento de erros, programação orientada a objetos além de acessar recursos do próprio *Windows* através de API's. (DEITEL, 2001)

6.2 A Plataforma JAVA

Para falarmos da plataforma .NET, não pode-se esquecer de outra poderosa plataforma –JAVA (DEITEL, 2001) – que oferece inúmeros recursos. JAVA como uma linguagem de programação, é orientada a objetos. A sua criação foi baseada no C e C++ (DEITEL, 2001), priorizando a organização e a reutilização de classes e componentes, JAVA como uma plataforma, tem um aspecto importante que é a independência de plataforma.

A plataforma JAVA oferece uma série de pacotes e definições para que os desenvolvedores possam criar soluções robustas, incluindo recursos de conectividade de banco de dados, rede, *web* e interfaces para aplicações *desktop*.

Ela utiliza uma máquina virtual, chamada JVM (*Java Virtual Machine*) (DEITEL, 2001), que garante que uma aplicação desenvolvida em JAVA possa ser executada sem problemas no sistema operacional.

Diante dessas características podemos citar um problema nesta plataforma: ela não oferece integração com outras linguagens de programação, sendo que o princípio da plataforma é justamente que se utilize uma única linguagem para tudo.

Dessa forma, não existe a integração de aplicações desenvolvidas em outras linguagens de programação com as que foram desenvolvidas em JAVA.

6.3 A Plataforma .NET

A plataforma .NET foi desenvolvida pela Microsoft durante 4 anos e seu lançamento oficial ocorreu em junho de 2000, a estratégia desta plataforma era de explorar ao máximo a integração de serviços através da *Web*, fazendo que tanto aplicações *Web* ou *Desktop* se comuniquem de uma forma simples, independente do sistema operacional e linguagem de programação.

Um diferencial significativo que a plataforma .NET oferece é o suporte a uma série de linguagens de programação, com isso uma mesma solução pode ser desenvolvida em mais de uma linguagem de programação ou até mesmo utilizar uma solução existente e integrar ela em um novo projeto, economizando em retrabalho.

6.4 .NET Framework

Um *framework* é uma estrutura pré-definida que contém um conjunto de componentes de software oferecendo uma estrutura padrão para o desenvolvimento de uma aplicação.

O .NET *Framework* oferece suporte a mais de 20 linguagens de programação, independente da linguagem escolhida, sendo ela suportada por este *framework* o resultado da aplicação será o mesmo.

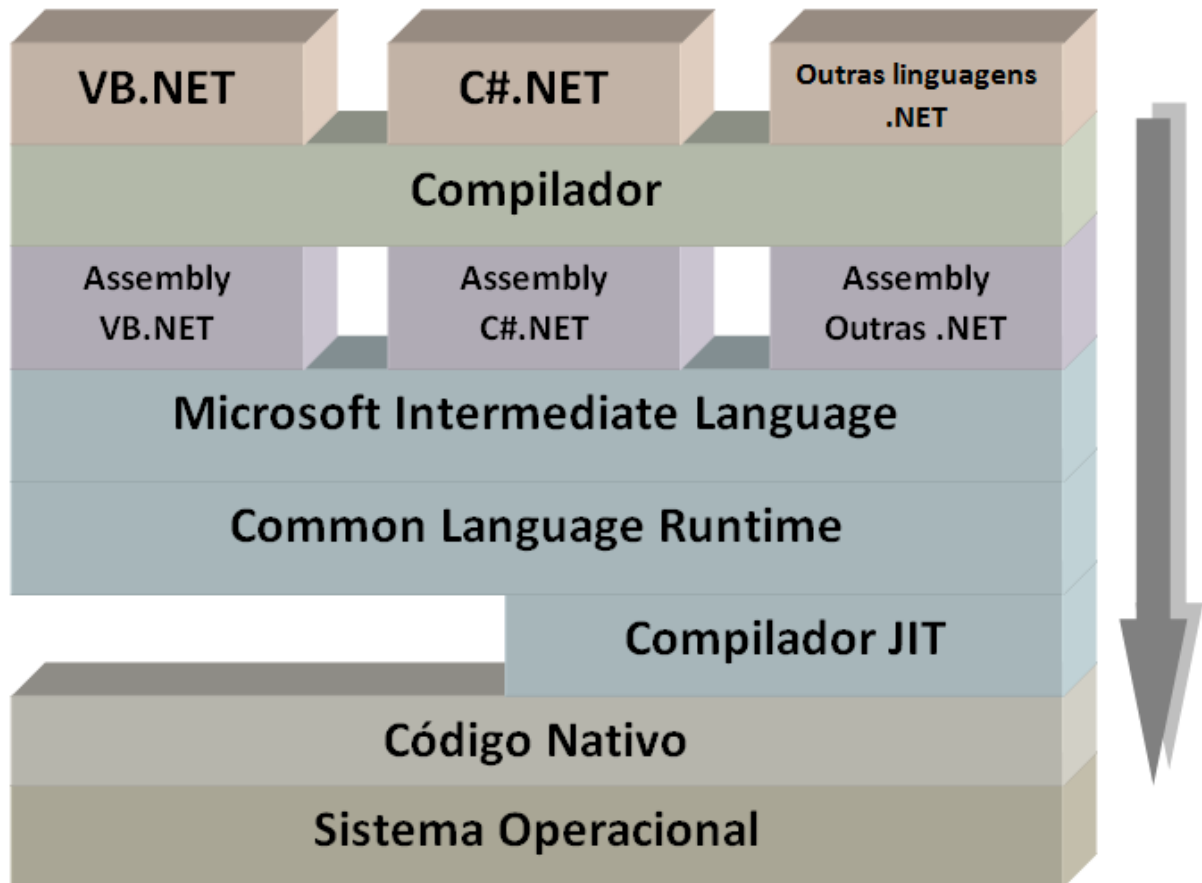


Figura 1: Processo de compilação em aplicações .NET

6.4.1 Versões do .NET Framework

As versões do .NET Framework passaram por algumas evoluções, desde a versão 1.0 até a 3.5, pois novas funcionalidades, principalmente em relação a classes, tecnologias, linguagens de programação e padrões de desenvolvimento, alguns aspectos foram atualizados, outros adicionados.

6.4.2 CLR - Common Language Runtime

O CLR . (THAI, 2001) é o ambiente que faz a execução das aplicações desenvolvidas dentro do .NET Framework, é a máquina virtual similar ao JVM (Java Virtual Machine), ou seja, está entre a aplicação desenvolvida e o sistema operacional.

Apesar de estas aplicações executarem no ambiente Windows, elas não são aplicações Win32 (THAI, 2001). Assim que uma aplicação é executada o Win32 chama o runtime .NET fazendo com que ele assuma o controle da aplicação executada, podendo gerenciar a memória, diminuindo a chance de ocorrer erros

durante sua execução. O CLR é o ambiente de execução da plataforma .NET, portanto independente da linguagem escolhida ele é o mesmo.

6.4.3 CTS – *Common Type System*

O CTS (THAI, 2001) é parte integrante do CLR e é responsável pela especificação dos tipos dentro da plataforma .NET, como se fosse uma padronização para que todas as linguagens possam se integrar entre as demais suportadas pela plataforma.

A plataforma .NET trata todas as linguagens suportadas de forma igual, uma classe escrita em C# deve ser equivalente a uma classe escrita em VB.NET. As linguagens devem seguir esses conceitos para que possam ser consideradas compatíveis com a plataforma.

6.4.4 CLS – *Common Language Specification*

O CLS (THAI, 2001) é um conjunto de regras que devem ser seguidas por linguagens que queiram se tornar compatíveis com a plataforma .NET. Isso é essencial para que o CLR entenda o código independente da linguagem escolhida pelo desenvolvedor, a obrigatoriedade de seguir estas regras é que no caso de seu descumprimento, compromete o princípio da independência e interoperabilidade da linguagem de programação, que é a grande vantagem que a plataforma oferece em relação as demais plataformas existentes.

O CLS reconhecendo que uma determinada linguagem é compatível com a plataforma .NET, gera um código intermediário, sendo que o mesmo não é um código *assembly*, que é dependente da arquitetura do processador, tornando-se assim um código único, mesmo que as aplicações tenham sido desenvolvidas com linguagens diferentes. O CLR interpreta de uma única maneira, provendo o funcionamento e a compatibilidade entre as linguagens compatíveis com a plataforma .NET.

6.4.5 BCL – *Base Class Library*

A BCL (THAI, 2001) é a biblioteca de classes da plataforma .NET, nela, contém classes básicas para que uma aplicação possa ser desenvolvida, o seu uso facilita o processo de desenvolvimento, pois existe a possibilidade da reutilização de

componentes de software, simplificando e agilizando toda a elaboração de uma aplicação.

Nessa biblioteca, é possível encontrar classes referentes a interfaces, manipulação de arquivos, gerenciamento de memória, funcionalidades de *web*, rede entre outras. As classes estão organizadas de forma hierárquica dentro de uma estrutura chamada *namespace*.

6.4.6 MSIL – Microsoft Intermediate Language

A MSIL (THAI, 2001) também conhecida como IL é a linguagem intermediária, que é gerada no momento em que o código é compilado, como foi dito anteriormente na CLS, faz com que o código ganhe a independência da linguagem de programação e da plataforma em que a aplicação será utilizada.

6.4.7 Metadata

Uma aplicação desenvolvida na plataforma .NET é auto-explicativa, portanto ela não necessita do registro do *Windows* para armazenar informações em relação à aplicação. Em um METADATA (THAI, 2001) consta informação referente à versão da aplicação, isso permite que duas aplicações com o mesmo nome possam ser utilizadas sem que ocorra nenhum tipo de conflito.

O CLR verifica no METADATA a versão que deve ser executada, facilitando a manutenção de aplicações em ambiente de produção, pois existia uma dificuldade principalmente na atualização e manutenção de DLL's. Os componentes que tinham versões diferentes, usados com descuido em uma atualização, poderiam gerar uma incompatibilidade entre versões, inviabilizando o funcionamento correto da aplicação.

6.4.8 JIT – “Just in Time”

O JIT (THAI, 2001) faz a conversão do código IL para o código específico da arquitetura do processador onde a aplicação .NET está sendo executada.

Atualmente, existem três tipos de JIT's que são:

- *Pré-JIT*: Compila todo o código da aplicação de uma única vez e armazena no *cache*, caso seja necessária a sua execução novamente.

- *Econo-JIT*: Compilador específico para o uso de dispositivos móveis como *Smartphones* (YANG, 2007) e *PocketPC's* (YANG, 2007). Ele compila o código sob demanda, liberando mais memória para que o dispositivo possa utilizar as demais funcionalidades sem sobrecarregá-lo.
- *Normal-JIT*: Compila o código sob demanda, armazenando o mesmo no *cache*, assim, se houver uma nova chamada do mesmo método, ele não vai precisar ser compilado novamente.

6.4.9 GC – Garbage Collector

O GC (THAI, 2001) é um recurso muito importante, pois ele faz o gerenciamento da memória, liberando espaços que já não estão mais em uso, sem que o desenvolvedor tenha que se preocupar com este tipo de situação.

Quando este gerenciamento é feito diretamente pelo desenvolvedor, as aplicações ganham em desempenho e eficiência. Na plataforma .NET, esse procedimento é chamado de *unsafe code* ou seja, código inseguro, portanto, o ideal é que o GC fique responsável por este tipo de ação.

6.5 .NET Compact Framework

O *.NET Compact Framework* (YANG, 2007) é a materialização do *.NET Framework*, mas é específico para dispositivos móveis. Devido às diferenças de *hardware* entre um computador e um dispositivo móvel, foi necessário criar um *framework* exclusivamente para este segmento de dispositivo.

O funcionamento da CLR e BCL, que são os principais componentes do *framework*, segue o mesmo princípio, porém na BCL existem classes que foram criadas especificamente para dispositivos móveis, assim como outras classes foram retiradas por fugirem do contexto destes dispositivos.

6.6 Linguagens Suportadas pela Plataforma .NET

Neste trabalho será utilizada a linguagem C# (TROELSEN, 2007). Seguem abaixo algumas das linguagens que atualmente são suportadas pela plataforma .NET (DEITEL, 2001),

- C#
- C++

- Visual Basic
- Jscript
- Cobol
- Small Talk
- Perl
- Pascal
- Python
- Oberon
- APL
- Haskell
- Mercury
- Scheme

6.7 Web Services

Um dos principais objetivos da plataforma .NET é a integração de aplicações *web* com *aplicações desktop*. Isso é possível graças à utilização de *Web Services* (SHORT, 2002), pois com seu uso é possível fazer com que aplicações de linguagens e ambientes diferentes se comuniquem, para isso os *Web Services* possuem tecnologias em relação à descoberta de serviços, descrição das funcionalidades e transporte dos dados.

Abaixo segue o exemplo de um *Web Service* publicado com o método "HelloWorld".

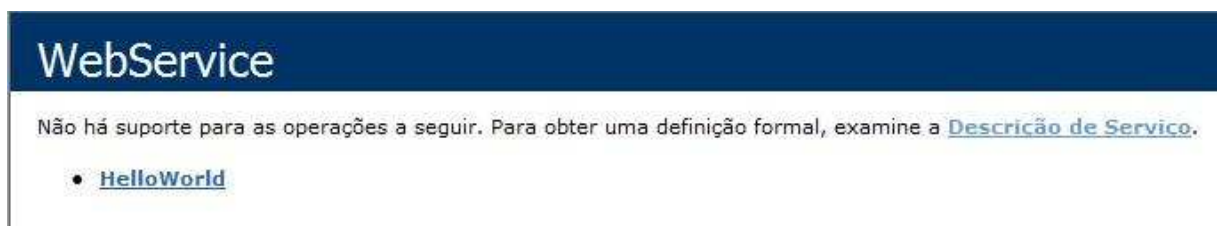


Figura 2: Web Service "HelloWorld"

6.7.1 SOAP – Simple Object Access Protocol

O SOAP (SHORT, 2002) é baseado em XML (SHORT, 2002) e faz comunicação da máquina cliente com a máquina servidor através de mensagens.

Trata-se do principal componente dos *Web Services*, isso porque com ele é possível detectar erros de interoperabilidade entre as aplicações envolvidas.

As mensagens são estruturadas por XML e possui um elemento *Envelope* com sub elementos de controle e com o corpo da mensagem.

O sub elemento de controle é chamado de *Header*, nele há informações referente a configurações. No sub elemento que possui o corpo da mensagem chamado *Body*, essas informações são referentes à requisição da mensagem e à resposta. Caso ocorra uma falha, ela irá retornar o elemento *Fault* com a descrição do erro.

Para dar continuidade ao exemplo na chamada do método "HelloWorld", a figura abaixo mostra uma requisição e sua respectiva resposta através do SOAP.

SOAP 1.1

O exemplo a seguir mostra uma solicitação e uma resposta SOAP 1.1. Os **espaços reservados**

```
POST /WebService/WebService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/HelloWorld"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" />
  <soap:Body>
    <HelloWorld xmlns="http://tempuri.org/" />
  </soap:Body>
</soap:Envelope>

HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" />
  <soap:Body>
    <HelloWorldResponse xmlns="http://tempuri.org/" />
      <HelloWorldResult>string</HelloWorldResult>
    </HelloWorldResponse>
  </soap:Body>
</soap:Envelope>
```

Figura 3: Solicitação e resposta SOAP do método "HelloWorld".

6.7.2 XML – eXtensible Markup Language

XML (SHORT, 2002) é uma linguagem de marcação de dados, estes organizados de forma hierárquica. Por ser um padrão aberto, sua portabilidade é plena, portanto é possível trocar informações de uma aplicação para outra.

No exemplo citado anteriormente, a chamada do método "HelloWorld" retorna o seguinte código em XML.

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">Hello World</string>
```

Figura 4: Retorno do método "HelloWorld"

6.7.3 WSDL – Web Service Description Language

Com o WSDL (SHORT, 2002) é possível descrever o serviço, estas informações estão relacionadas às operações e aos parâmetros necessários para que a comunicação entre as aplicações ocorra.

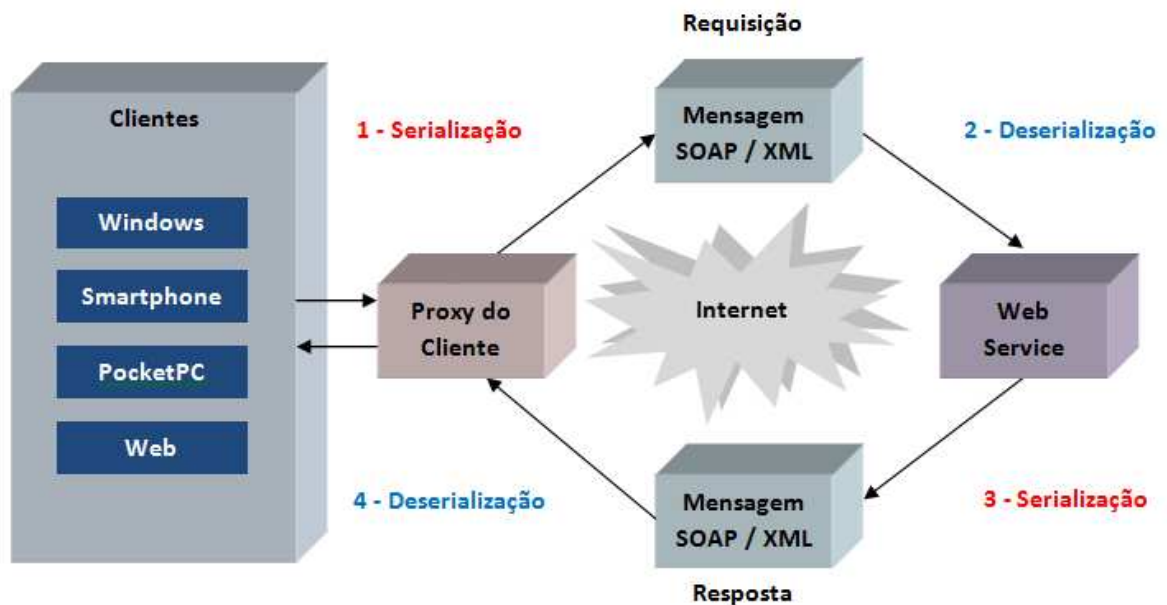


Figura 5: Fases de funcionamento dos Web Services

A figura acima mostra as fases do funcionamento de um *Web Service*.

- Fase 1: Requisição codificada feita pela aplicação através do SOAP.
- Fase 2: O *Web Service* decodifica as informações que recebeu e acessa o método requisitado.
- Fase 3: A resposta do método é codificada através do SOAP.
- Fase 4: A aplicação recebe a solicitação decodificada.

Diante da exigência de fazer com que uma aplicação *Windows*, *Web* e *Mobile* se comuniquem é inevitável que o desenvolvimento de *Web Services* seja necessário, com ele, será possível reaproveitar o mesmo *Web Service* na integração dos três ambientes.

7 METODOLOGIA

Para elaboração do trabalho, será necessária a adoção do padrão MVC (GAMMA, 1994) com o intuito de estruturar as camadas da aplicação, além de facilitar a comunicação entre as mesmas.

Por se tratar de uma aplicação orientada a objetos, torna-se inevitável a utilização da UML (PIOLINE, 2005) para realizar a modelagem do projeto, assim como do Diagrama Entidade-Relacionamento (CHEN, 1990) para a planificação do Bando de Dados da aplicação.

Para verificar se uma aplicação pode ser portátil em mais de um ambiente, será necessária uma análise quanto suas similaridades e diferenças para definir o nível de portabilidade que a aplicação irá atender.

A elaboração de um padrão de desenvolvimento será importante para avaliar a integração dos ambientes na aplicação, pois será feita uma descrição detalhada de como serão criadas as camadas e de que forma elas irão se relacionar.

7.1 MVC – Model-View-Controller

O padrão MVC (GAMMA, 1994) é utilizado há algum tempo, pois ele foi criado para a construção de interfaces no ambiente *Smalltalk* (GAMMA, 1994).

Utilizar esse padrão no projeto consiste em dividir a aplicação em três camadas. Com esta divisão, a implementação e manutenção da aplicação ficam facilitadas, pois, caso haja a necessidade de alterar alguma interface, as regras de negócios não serão afetadas, pois estarão em outra camada. Podemos dizer que estas camadas são divididas em:

- *Modelo*: É a principal camada do padrão, ela contempla as regras de negócio da aplicação, que são as suas funcionalidades. Nela, consiste a parte do armazenamento, manipulação e geração dos dados além de ser independente da camada de visualização.

- *Visualização*: É a camada responsável pela visualização das informações da aplicação. Ali, estão contidas suas interfaces, que são responsáveis pela entrada e saída das informações.
- *Controle*: É a camada intermediária entre o modelo e a visualização, ela é responsável pelo controle dos eventos da aplicação.

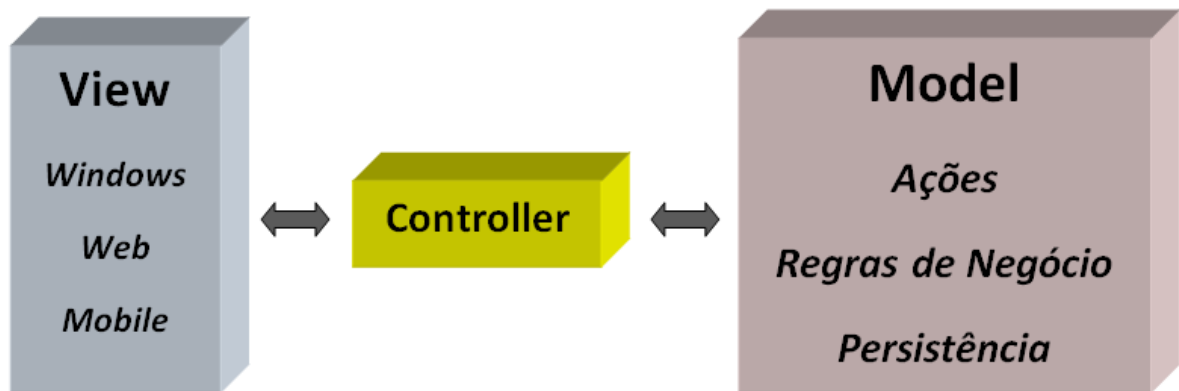


Figura 6: Comunicação entre as camadas do MVC.

7.2 Diagrama de Entidade-Relacionamento

Na solução proposta será utilizado um banco de dados relacional, portanto, a modelagem será realizada através de um diagrama entidade-relacionamento (CHEN, 1990). Ele é a representação gráfica dos relacionamentos entre as tabelas e as associações existentes entre seus respectivos atributos.

7.3 UML

UML é a linguagem de modelagem unificada (*Unified Modelling Language*) (PIOLINE, 2005). Através de seus diagramas ela facilita a especificação, visualização e documentação de modelos de sistemas orientados a objeto. Seus diagramas são divididos em duas categorias: Diagramas Estruturais e Diagramas de Comportamento.

São usados diagramas estruturais para capturar a organização física dos elementos dentro do seu sistema, como um objeto se relaciona a outro. Os diagramas estruturais utilizados neste trabalho são:

Diagrama de Classe: Usa classes e interfaces para capturar detalhes sobre as entidades que compoem seu sistema e as relações estáticas entre eles.

Diagramas de comportamento focam os comportamentos dos elementos em um sistema. É possível usar diagramas de comportamento para capturar exigências, operações e a alteração de estado dos elementos. Os diagramas de comportamento utilizados neste trabalho são:

Diagrama de Atividades: Captura o fluxo de um comportamento ou atividade para o próximo. É semelhante a um fluxograma clássico, entretanto, é mais expressivo.

Diagrama de Caso de Uso: Capturam as exigências funcionais do sistema, mostrando os atores que vão interagir com o sistema e os casos de uso com os respectivos relacionamentos.

Diagramas de Interação: Usados para mostrar a interação formada pelos objetos e seus relacionamentos, incluindo as mensagens trocadas entre os mesmos. Os tipos de diagramas de interação são os de Seqüência e os de Colaboração, eles são equivalentes, podendo ser um convertido em outro, entretanto, não é uma regra que eles visualizem exatamente as mesmas informações.

7.4 Analisando as Versões do .NET Framework quanto a sua Portabilidade

Como o desenvolvimento da aplicação envolve o ambiente para dispositivos móveis, é necessária a utilização de um *framework* específico para este tipo de *hardware*, portanto aspectos de portabilidade, diferenças e similaridades entre os ambientes e seus respectivos *frameworks* serão vistos abaixo.

7.4.1 Portabilidade

Segundo (IEEE 90), portabilidade é a facilidade com que um sistema ou um componente pode ser transferido de um *hardware* ou *software* para outro ambiente.

Para concluirmos que uma aplicação é portátil, é preciso definir qual o seu grau de portabilidade, a medição é feita em relação a dois aspectos:

- As diferenças entre o ambiente original e o novo ambiente que a aplicação será desenvolvida.
- A capacidade do desenvolvedor em reutilizar classes e métodos comuns para dois ou mais ambientes, diminuindo o esforço exigido.

7.4.2 Diferenças entre os ambientes

A utilização da plataforma .NET facilitou o uso de uma única linguagem de programação para o desenvolvimento nos três ambientes, portanto a linguagem C# (TROELSEN, 2007) é portátil nos ambientes já mencionados. As diferenças que existem entre eles, basicamente, estão nas interfaces gráficas e nas classes que são específicas para cada tipo de plataforma.

Essas diferenças ocorrem porque estamos utilizando dois *frameworks* diferentes, o .NET Framework, que é utilizado para os ambientes *Windows* e *Web*, já para os dispositivos móveis, é utilizado o .NET *Compact Framework*. Neste *framework* existem classes específicas para este tipo de hardware, além de classes específicas para os ambientes *Windows* e *Web* terem sido removidas, devido à limitação que estes dispositivos têm em relação aos computadores.

7.4.3 Similaridades entre os ambientes

Nos ambientes *Windows* e *Web*, como será utilizado o .NET *Framework*, com exceção da parte gráfica, praticamente, todas as classes serão reutilizáveis. Da mesma forma no ambiente *Mobile*, a aplicação para *Smartphone* e *Pocket PC*, como utilizam o .NET *Compact Framework*, suas classes poderão ser reutilizadas.

Para que possa ser feita a integração de aplicações de um *framework* para o outro, é necessária a utilização de *Web Services*, pois neles são criadas classes que são comuns entre os ambientes, como o acesso ao banco de dados, por exemplo.

Com a elaboração de *Web Services* o ciclo de integração entre os ambientes é completo podendo assim avaliar o grau de portabilidade que a aplicação tem.

7.5 Estrutura de Desenvolvimento da Aplicação

Conforme as definições apresentadas acima, procurou-se desenvolver um modelo padrão para estruturar o desenvolvimento da aplicação. Esse modelo está definido em três camadas, a fim de organizar o papel dos módulos da aplicação e de verificar como os ambientes irão se comunicar.

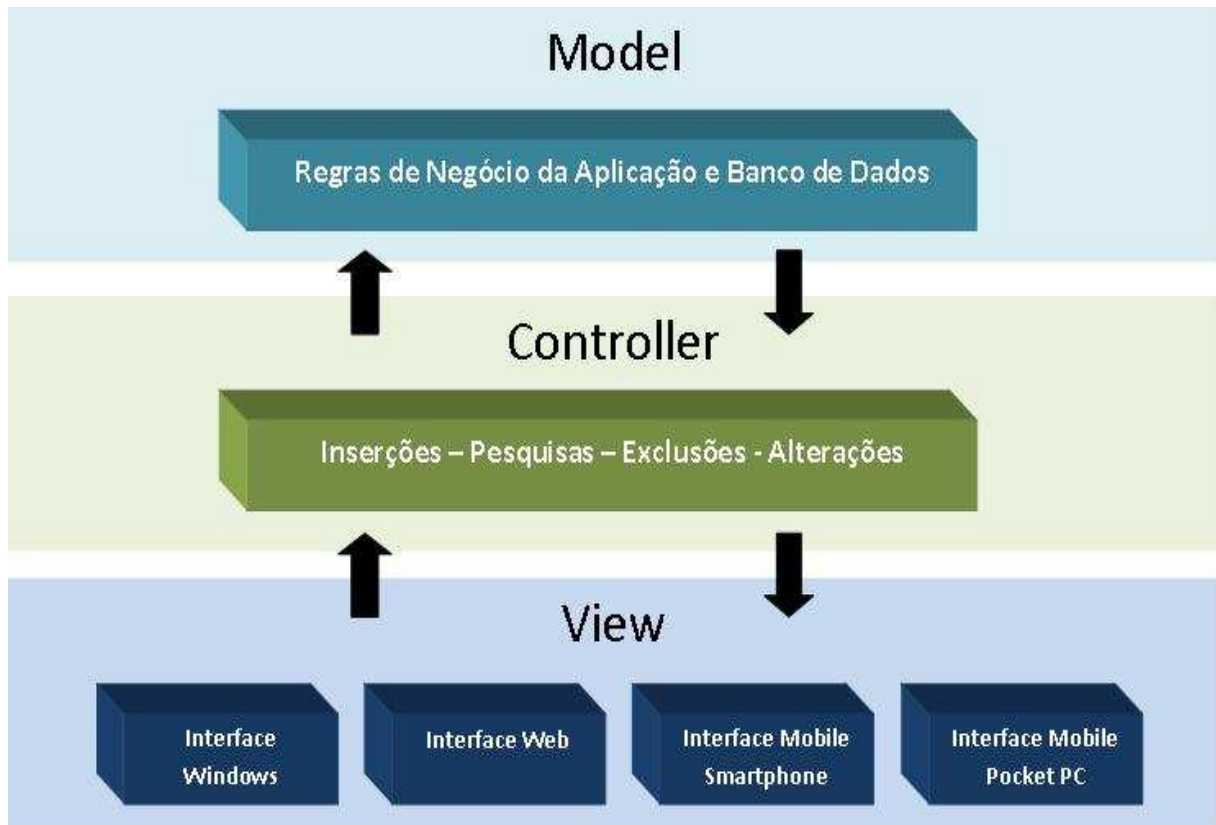


Figura 7: Estrutura de camadas da aplicação.

Para chegar neste modelo, tomou-se como base o princípio do MVC, dividindo a aplicação em uma camada de interface, outra de controle e uma camada responsável pela parte lógica da aplicação e de acesso a dados.

Durante o desenvolvimento do modelo, foi possível verificar que *.NET Framework* e o *.NET Compact Framework* não podiam ser referenciados de forma direta, inviabilizando a depuração do código. A referência poderia ser feita apenas adicionando a DLL do respectivo framework já compilada. Nesse caso, a agilidade do desenvolvimento ficaria comprometida e confusa. Além disso, teria que referenciá-las para cada ambiente resultando em retrabalho.

Com essas dificuldades, criou-se um método que facilitou o entendimento da função de cada camada da aplicação, abaixo segue o detalhamento do modelo desenvolvido:

- Foi definida a camada de interface para cada ambiente, o seu papel é realizar a chamada da camada de controle, posteriormente, aguarda o resultado e apresenta na interface para que o usuário visualize.

- Para a camada de controle, deve ser criado um *Web Service*, contendo os métodos de cada objeto, com suas respectivas ações como inserções, consultas, alterações e exclusões.
- Para a camada de regra de negócios e banco de dados, devem ser criadas pelo menos três DLLs, sendo que essas utilizarão o *.NET Framework*. A utilização do *.NET Compact Framework* será necessária caso a aplicação *Mobile* necessite de uma funcionalidade específica que o *.NET Framework* não atenda, abaixo seguem as três DLLs que devem ser criadas:
 - Banco de Dados: Deve ser criada contendo os dados da conexão com o banco de dados, como nome do servidor, nome da base de dados, usuário e senha. Com essas informações declaradas nesta DLL ela é visível em toda a aplicação.
 - Regras de Negócio: Deve ser criada a parte que contém a lógica da aplicação, além das operações de banco de dados.
 - Objetos: Nesta DLL devem ser criadas as classes de cada objeto e seus respectivos atributos e métodos.

A figura abaixo mostra a estrutura das camadas utilizando a ferramenta Microsoft Visual Studio 2008.

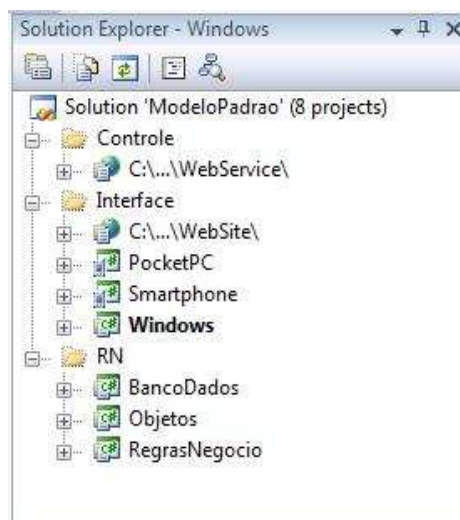


Figura 8: Estrutura do modelo no Microsoft Visual Studio 2008.

Definido o papel de cada camada, é importante detalhar como funciona a ligação entre elas, seguindo, abaixo, o detalhamento das referências de cada uma:

- A camada de interface deve referenciar apenas o *Web Service*, que faz o papel da camada de controle;
- A camada de controle deve referenciar as DLL's de conexão com o banco de dados, regras de negócio e objetos.

Com isso, a ligação entre as camadas é consolidada, criando independência entre as mesmas, proporcionando que equipes diferentes desenvolvam a mesma aplicação, sendo que cada equipe desenvolva sua respectiva camada.

Toda a camada de regras de negócio e controle é comum para os ambientes, ou seja, se houver qualquer alteração na regra de negócios, esta irá servir para todos os ambientes envolvidos, evitando redundância de código, além de diminuir o retrabalho durante a fase desenvolvimento.

7.6 Tecnologias Utilizadas

Para realização deste modelo foram utilizadas as seguintes tecnologias e recursos:

Ambiente de Desenvolvimento

- Microsoft Visual Studio 2008;
- Servidor IIS (*Internet Information Services*);
- Microsoft SQL Server 2005;

Sistemas Operacionais

- Microsoft Windows Vista;
- Microsoft Windows XP;
- Microsoft Windows Mobile 5.0;

Navegadores de Internet

- Internet Explorer 7.0;
- Mozilla Firefox.

8 RESULTADOS

Após o desenvolvimento do modelo, no qual pode-se estabelecer como a estrutura da aplicação será constituída, foi possível chegar a importantes conclusões.

Em relação à plataforma .NET após realizar um estudo sobre ela e seus respectivos *frameworks* e componentes que os integram, conclui-se que é uma plataforma de fácil utilização, principalmente, no que diz respeito ao foco deste trabalho, que é a integração de aplicações unindo tecnologias diferentes.

Embora a proposta do trabalho seja utilizar apenas uma linguagem de programação, vale ressaltar a independência da plataforma também em relação às demais linguagens suportadas, além de explorar muito bem a utilização de serviços na *web*.

Comprovando os recursos citados acima, foi preciso elaborar um modelo que dividisse a aplicação em camadas com a finalidade de estruturar e organizar o projeto. Para isso, foi utilizado o conceito do modelo MVC, entretanto não foi possível utilizar o *framework* disponibilizado pela Microsoft chamado de ASP.NET MVC Framework (MVC, 2008), pois o mesmo é voltado para aplicações *Web*.

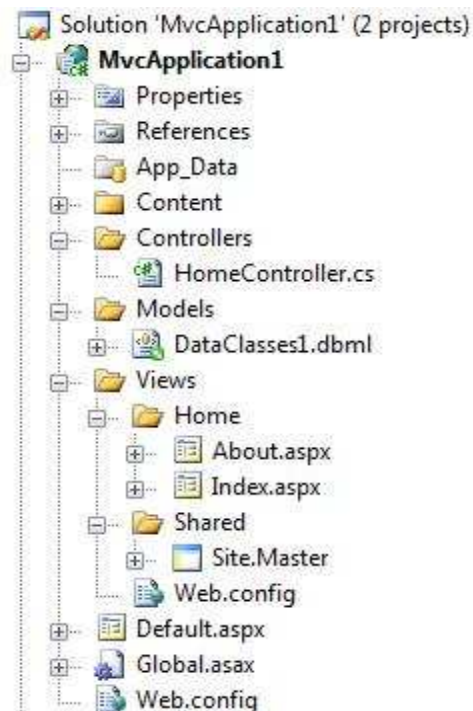


Figura 9: Estrutura do ASP.NET MVC Framework no Microsoft Visual Studio 2008.

Mesmo com a viabilidade de referenciar este *framework* nos demais ambientes estudados, a utilização de apenas parte do framework deixaria a comunicação entre as camadas com um nível de complexidade maior.

Utilizar um modelo próprio, considerando aspectos conceituais do modelo MVC, facilitou o entendimento da comunicação entre as camadas.

Outro aspecto é a utilização de *Web Services* para a camada de controle da aplicação, com isso, os ambientes dependem de comunicação com a internet, caso contrário o funcionamento da sua aplicação é comprometido.

Portanto, pode-se concluir que a união desses ambientes em uma única aplicação, traz diversos benefícios para todas as partes envolvidas. Desse modo, será possível desenvolver uma aplicação e adaptar apenas o que for necessário para outros ambientes, propiciando uma economia de esforço considerável para o desenvolvedor.

Através deste modelo, foi possível desenvolver uma aplicação para testes, sendo possível validar a integração entre os ambientes, além do seu funcionamento.

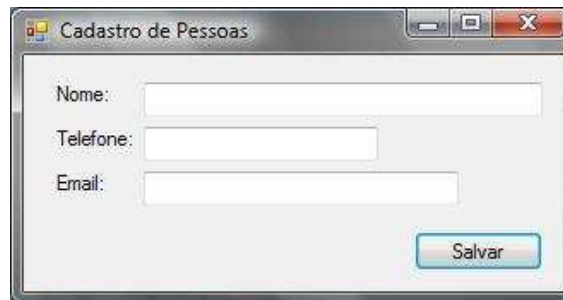


Figura 10: Cadastro de Pessoas – Ambiente Windows.

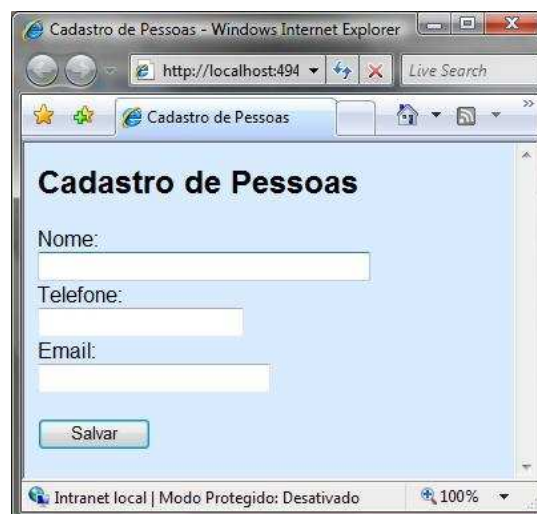


Figura 11: Cadastro de Pessoas – Ambiente Web.



Figura 12: Cadastro de Pessoas – Ambiente Mobile para Smartphone.



Figura 13: Cadastro de Pessoas – Ambiente Mobile para PocketPC.

A aplicação de teste consiste no cadastro de uma pessoa, para tal foi desenvolvida a interface de cadastro para cada ambiente como mostram as figuras 10, 11, 12 e 13.

Como as interfaces têm a mesma funcionalidade em todos os ambientes, a figura abaixo mostra o código do botão “Salvar” para o ambiente *Windows*.

```

public partial class frmPessoas : Form
{
    public frmPessoas()
    {
        InitializeComponent();
    }
    private void btnSalvar_Click(object sender, EventArgs e)
    {
        try
        {
            //Instancia o web service
            Windows.refWSPessoas.wsPessoas wsPes = new Windows.refWSPessoas.wsPessoas();

            //Instancia o objeto pessoa pela referência do web service
            Windows.refWSPessoas.Pessoas oPes = new Windows.refWSPessoas.Pessoas();

            //Atribui os valores para o objeto pessoa
            oPes.Nome = txtNome.Text;
            oPes.Telefone = txtFone.Text;
            oPes.Email = txtEmail.Text;

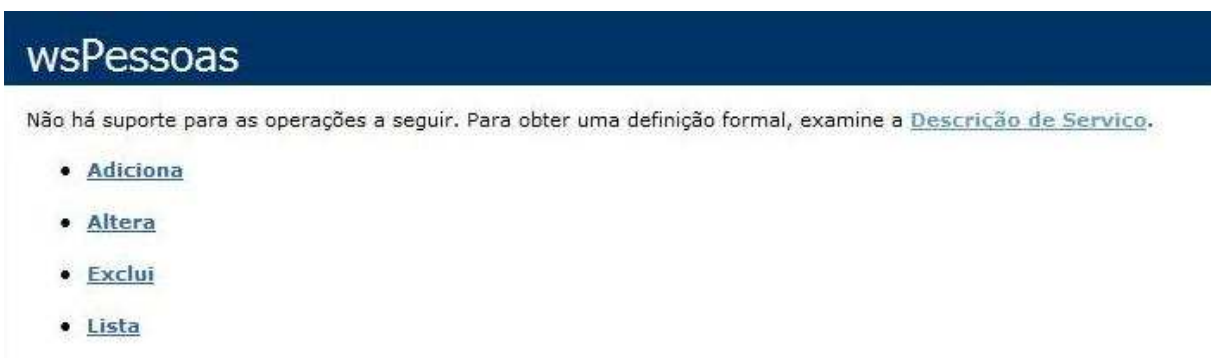
            //Invoca o método adiciona do web service enviando o objeto pessoa
            wsPes.Adiciona(oPes);
        }
        catch
        {
            //Mensagem de Erro
        }
    }
}

```

Figura 14: Figura – Exemplo de código do ambiente Windows.

A figura abaixo mostra os métodos disponíveis que o *Web Service* “wsPessoas” possui, basicamente, ele oferece ações que o usuário pode tomar através da interface, no exemplo, ele pode efetuar inserções, exclusões, pesquisas e alterações no banco de dados no que diz respeito ao objeto pessoa.

Estes métodos podem ser acionados de qualquer ambiente, ou seja, qualquer alteração efetuada no código neste *Web Service* irá servir para todos os ambientes.



The screenshot shows the title bar 'wsPessoas' in a dark blue header. Below it, a message states: 'Não há suporte para as operações a seguir. Para obter uma definição formal, examine a [Descrição de Serviço](#).' Underneath, there is a bulleted list of methods: 'Adiciona', 'Altera', 'Exclui', and 'Lista', each with a blue underline.

Figura 15: Web Service wsPessoas com sua lista de métodos.

Caso fosse feita uma pesquisa ou uma carga da tabela “Pessoas” do banco de dados, o método “Lista” do *Web Service* retornaria em XML os dados da pesquisa,

provando que existe interoperabilidade de aplicações. Como o XML é um padrão aberto, como foi visto anteriormente, mostra a transparência no tráfego dos objetos nos ambientes.

No momento em que o tráfego ocorre, os objetos são serializados, ou seja, codificados durante o processo de envio, tanto na requisição quanto na resposta. A decodificação é feita pelo *Web Service* para que sejam efetuadas as operações necessárias e no retorno, quando chega na aplicação.

```
<?xml version="1.0" encoding="utf-8" ?>
- <ArrayOfPessoas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://tempuri.org/">
- <Pessoas>
  <Nome>Pessoa A</Nome>
  <Telefone>1234</Telefone>
  <Email>pessoaa@dominio.com</Email>
</Pessoas>
- <Pessoas>
  <Nome>Pessoa B</Nome>
  <Telefone>5678</Telefone>
  <Email>pessoab@dominio.net</Email>
</Pessoas>
- <Pessoas>
  <Nome>Pessoa C</Nome>
  <Telefone>9012</Telefone>
  <Email>pessoac@dominio.com.br</Email>
</Pessoas>
</ArrayOfPessoas>
```

Figura 16: Retorno da requisição do método "Lista" no formato XML.

Vale ressaltar que foram feitas inserções nos quatro ambientes, e o tempo de resposta foi satisfatório. Além de inserções simultâneas, também foi feita a carga dos dados em uma lista no próprio formulário de pesquisa das aplicações.

O desenvolvimento da camada de regras de negócio e dados e também da camada de controle, são usadas para qualquer ambiente, sendo necessário o desenvolvimento das interfaces para cada ambiente. Podendo, concluir que existe portabilidade entre os ambientes.

Com a independência da aplicação quanto aos seus ambientes, o usuário final terá maior mobilidade na execução de suas tarefas, além de um ganho de produtividade.

9 TRABALHO DE CONCLUSÃO DE CURSO – II

Para o TCC-II será realizado o desenvolvimento do sistema para o Departamento de Criminalística baseado na metodologia especificada em TCC I.

A elaboração deste sistema será compreendida das seguintes atividades:

- Análise do sistema atual do Departamento de Criminalística.
- Requisitos da aplicação.
- Modelagem da aplicação.
- Desenvolvimento da aplicação com base no modelo desenvolvido no TCC I
- Testes.
- Documentação da aplicação.
- Entrega da aplicação.

10 REFERÊNCIAS

CHEN, Peter. **Modelagem de Dados. A Abordagem Entidade-Relacionamento para Projeto Lógico**, 1990.

DEITEL, Harvey M. **C# How to Program**. [978-0130622211], 2001.

IEEE. **IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries**, 1990.

GAMMA, Erich. **Design Patterns: Elements of Reusable Object-Oriented Software**, 1994.

MVC. **The Official Microsoft ASP.NET Site** – Site oficial do Framework MVC para aplicações ASP.NET, URL: www.asp.net/mvc, visitado em junho de 2008.

PILONE, Dan. PITMAN, Neil. **UML 2.0 in a Nutshell**. [0-596-00795-7], 2005.

THAI, Thuan. LAM, Hoang. **NET Framework Essentials**. [0-596-00165-7], 2001.

TROELSEN, Andrew. **Pro C# 2008 and the .NET 3.5 Platform, 4th Edition**. [978-1-59059-884-9], 2007.

SHORT, Scott. **Building XML Web Services for the .NET Platform**, [0-735-61406-7], 2002.

YANG, Baijian. ZHENG, Pei. NI, Lionel. **Professional Microsoft Smartphone Programming**. [0-471-76293-5], 2007.