

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

**UM ESTUDO SOBRE A PLATAFORMA .NET COMO UMA ALTERNATIVA NO
DESENVOLVIMENTO DE SOFTWARE PARA A INTERNET**

Jailson Almeida de Carvalho

NATAL

2002

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
BACHARELADO EM SISTEMAS DE INFORMAÇÃO**

**UM ESTUDO SOBRE A PLATAFORMA .NET COMO UMA ALTERNATIVA NO
DESENVOLVIMENTO DE SOFTWARE PARA A INTERNET**

Jailson Almeida de Carvalho

NATAL

2002

JAILSON ALMEIDA DE CARVALHO

**UM ESTUDO SOBRE A PLATAFORMA .NET COMO UMA ALTERNATIVA NO
DESENVOLVIMENTO DE SOFTWARE PARA A INTERNET**

Monografia apresentada ao curso de
Graduação em Sistemas de Informação da
Faculdade Natalense para o desenvolvimento
do Rio Grande do Norte para obtenção do grau
de Bacharelado em Sistemas de Informação.

Orientador:

Aluizio Ferreira da Rocha Neto

NATAL

2002

RB-13.749

M

FARN

BIBLIOTECA

Nº Reg M000217

04 05 05

004.4

C324e

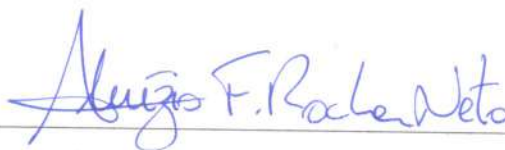
2002

LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Um Estudo Sobre a Plataforma .NET como uma Alternativa no Desenvolvimento de
Software para a Internet

Jailson Almeida de Carvalho

Monografia apresentada e aprovada em 16 de DEZEMBRO de 2002 pela banca
examinadora composta pelos seguintes membros:



Prof. Aluizio Ferreira da Rocha Neto

Orientador



Profa. Livia Maria Martins da Silva

Examinador



Prof. Fellipe Araújo Aleixo

Examinador

Natal, 16 de DEZEMBRO de 2002

*Aos meus pais dedico este trabalho, como
resultado de meu esforço e também do
grande apoio que deles recebi em todos os
projetos de minha vida.*

AGRADECIMENTOS

Venho agradecer a todos que contribuíram para a realização deste trabalho, aqueles que me passaram o conhecimento e o apoio necessário para realizá-lo.

Agradeço a Deus pela força que sempre me concebeu, nos momentos que sempre precisei e encontrei a sua palavra para me confortar.

Aos meus pais que foram de fundamental importância para conclusão deste projeto de vida, pelo apoio de todas as formas, a quem devo toda a minha gratidão e amor.

A minha esposa e filhos, pela compreensão, pela paciência colocada a meu inteiro dispor nos momentos difíceis da vida.

Aos meus professores e minha coordenadora, pelos ensinamentos ministrados e pelo apoio em alguns momentos difíceis que passei, em especial ao meu orientador Aluizio Ferreira Rocha Neto, pela presteza e suporte que me ofereceu para a realização e conclusão deste projeto que será de grande contribuição na minha vida profissional.

E não poderia de esquecer dos meus amigos, em particular Ana Angélica, Daniele Lobo, Mounarte Leitão, Rodrigo Bruno e Vanessa Araújo, que sempre estiveram ao meu lado.

RESUMO

Integrar informação e comunicação, fazendo com que toda a Internet esteja a serviço do usuário é a missão da nova geração de software lançada pela Microsoft chamada de .NET. Neste trabalho, apresentaremos um estudo sobre esta nova plataforma de desenvolvimento, expondo a sua arquitetura e suas linguagens, a fim de possibilitar uma melhor análise.

ABSTRACT

Information and communications association, make the all Internet was a the usuary service is the mission's new generation software Microsoft denomination .NET. This work related a study the encrease plataform, expound his arquitetury and languages, possibility the best analysis.

SUMÁRIO

1. INTRODUÇÃO.....	8
1.1 PROBLEMA DE PESQUISA.....	8
1.2.1 Padrões Abertos	9
1.2 ORGANIZAÇÃO DO TRABALHO.....	11
2. PLATAFORMA .NET	12
2.1 NET FRAMEWORK.....	12
2.2 VANTAGENS DA .NET FRAMEWORK.....	14
2.3 LINGUAGENS DO .NET FRAMEWORK.....	15
2.3.1 A Linguagem C#.....	15
2.3.2 A Linguagem Visual Basic.NET.....	17
2.3.3 A Tecnologia ASP.NET	19
3 WEB SERVICES – SOFTWARE COMO SERVIÇO.....	21
3.1 UMA VISÃO INICIAL	21
3.2 O QUE É SOAP?.....	21
3.3 ARQUITETURA DO <i>WEB SERVICE</i>	24
3.4 SOLUÇÃO DA ARQUITETURA	24
3.5 DESCRIÇÃO DOS WEB SERVICES: OS ARQUIVOS SDL	27
4 .NET VERSUS JAVA.....	29
5 CONCLUSÕES	30
REFERÊNCIAS BIBLIOGRÁFICAS	31

ÍNDICE DE FIGURAS

Figura 1: Compilação na plataforma .NET, MSIL é produzido.....	13
Figura 2: Os objetos podem ser acessados por qualquer pessoa na <i>Web</i>	25
Figura 3: Escrevendo um <i>Web Service</i>	25
Figura 4: TimeService.asmx em um navegador.....	27
Figura 5: Mostrando os segundos.....	27

1. INTRODUÇÃO

1.1 APRESENTAÇÃO

Este trabalho tem como objeto de análise um estudo sobre a plataforma .NET, as linguagens que compõem o framework .NET, assim como uma comparação entre .NET e a plataforma Java.

1.1 PROBLEMA DE PESQUISA

No mundo atual, a informação é colocada à disposição dos indivíduos através de tecnologias cada vez mais inovadoras. As novas tecnologias da informação e comunicação ampliam o potencial cognitivo humano levando a um grande contingente de agentes a interligar-se em um mesmo processo de construção de conhecimentos.

O desenvolvimento de sistemas baseando-se em protocolos e padrões abertos, acoplados a ferramentas e serviços na Internet permite que a integração entre computadores e redes de comunicação, seja de formas mais abrangentes e produtivas, expandindo seu horizonte de atuação para mares nunca antes navegados, à medida que ferramentas e tecnologias vão sendo incorporadas ao nosso dia-a-dia.

O surgimento de uma nova área de tecnologia é sempre um momento de expectativa e intimidador para empresas e desenvolvedores. Os desenvolvedores querem aprendê-la o mais rápido possível, já que representa uma tecnologia nova e interessante que promete facilitar suas vidas.

Integrar informação e comunicação, fazendo com que toda a Internet esteja a serviço do usuário é a missão da nova geração de software, a missão da plataforma Microsoft .NET.

Para se tentar ter uma idéia do que é a .NET podemos inicialmente dizer que o .NET é a resposta da Microsoft à linguagem Java e à plataforma J2EE (*Java 2 Enterprise Edition*) da Sun Microsystems. Mesmo para quem prefere se manter independente da plataforma Windows, a força da gigante de software Microsoft não pode ser ignorada.

Vamos encontrar no .NET os mesmos princípios da plataforma Java: máquina virtual; memória automática (*Garbage Collection*); *bytecode* portátil; linguagens dinâmicas e reflexivas (C# e outras); um *framework* orientado a objetos unificado e extremamente

abrangente; suporte a uma variedade de necessidades como distribuição, segurança, acesso a dados, ou desenvolvimento para *Web*.

Várias diferenças vêm da opção da Microsoft por uma plataforma amarrada ao Windows. Muitas bibliotecas do .NET são finas cascas sobre APIs existentes do Windows. Quem conhece as funções da biblioteca Win32, classes da MFC (*Microsoft Foundation Class*) e similares, já tem alguma familiaridade com a .NET. A desvantagem? O porte para outros sistemas operacionais (SO) é possível em tese, mas seria muito difícil e o resultado ineficiente. Ou seja, na prática, desenvolver para a .NET significa desenvolver para os novos SOs Microsoft, como o Windows XP, o Pocket PC e o WinCE.NET. Estes dois últimos voltados para as plataformas de palmtops e celulares.

1.2.1 Padrões Abertos

Muito dificilmente uma aplicação a ser desenvolvida, ou mesmo utilizada dentro de uma empresa, tem vida própria. Normalmente é necessário interagir com outras aplicações e trocar informações com as mesmas. Estas outras aplicações podem estar implementadas há muito tempo, até mesmo em um mainframe, por exemplo.

Toda nova aplicação deve levar em consideração a interação com estas aplicações da melhor forma possível. Interoperabilidade é a palavra chave aqui. A palavra pode estar um pouco batida, mas a realidade é que muito pouco foi efetivamente realizado para que esta integração aconteça de maneira simples e sem traumas.

Segundo TOLOMELLI (2002), a plataforma Microsoft .NET tem a interoperabilidade como um dos seus alicerces, e para que essa integração seja possível, faz-se necessário trabalhar com padrões amplamente adotados pela indústria, e especificado por um órgão independente de grande reconhecimento. Desta forma a estratégia foi desenhada sobre a linguagem XML (*eXtensible Markup Language*).

Para quem não conhece em detalhes XML, é uma linguagem baseada em *tags* (como o HTML) desenvolvida com o objetivo de trocar dados e informações entre aplicativos. Por ser texto, simplesmente fica fácil que XML seja entendido em qualquer sistema operacional (todos eles entendem texto) e facilidade a implementação de soluções independente de plataforma onde os aplicativos estejam rodando. Hoje já temos soluções XML rodando em mainframes e trocando informações com aplicativos rodando em micro computadores.

A escolha por XML não foi por acaso. A linguagem é controlada pelo W3C (World Wide Web Consortium), consórcio responsável pela definição dos principais padrões

utilizados na Internet como HTTP, HTML, entre outros. XML tem recebido um grande suporte da indústria e os principais fabricantes de software já oferecem suporte ao padrão.

O XML está nascendo para ser esta língua franca da Internet, mas nem todas aplicações falam XML ou são facilmente adaptadas para falar. Por exemplo, se você tem um sistema SAP R/3 funcionando na sua empresa ele troca informações através do iDoc, um formato proprietário de arquivos. Todavia para enviar dados para um fornecedor a sua empresa utiliza EDI no formato EDIFACT.

Todas as ferramentas .NET foram construídas como base no padrão XML. Assim, já nasceram falando a linguagem da Internet e estão preparadas para se comunicar. O modelo de programação da plataforma .NET baseado XML introduz o conceito de Serviços Web. Ao contrário da situação atual, na qual os sites são criados de forma artesanal, com um desenvolvimento extremamente complexo para garantir sua operação com outros sites, no .NET fornece um mecanismo de construção de sites e serviços que permite a interação com quaisquer outros sites ou serviços.

Outro protocolo padrão a ser adotado amplamente na plataforma é o SOAP (*Simple Object Access Protocol*, protocolo de acesso a objetos simples). O SOAP nasceu em cima do XML para resolver um problema encontrado por aqueles que pretendem criar um ambiente de aplicativos distribuídos através da Internet. Os modelos de objeto existentes hoje no mercado como COM+ (*Component Object Model Plus*) da Microsoft, CORBA (*Common Object Request Broker Architecture*) da OMG ou EJB (*Enterprise JavaBeans*) da Sun Microsystems, trabalham de forma similar no que diz respeito à forma de requisição de um objeto remoto. Qualquer um destes formatos exige uma conexão permanente entre o cliente e o servidor durante a execução dos métodos do objeto servidor, o que pode ocorrer durante toda a execução da aplicação. Este tipo de conexão não pode ser garantido, pelo menos de uma maneira consistente, através da Internet, pois não sabemos exatamente onde estão localizados esses objetos.

Quando falarmos em *Web Services*, um dos novos conceitos apresentados pela plataforma .NET, estamos falando da implementação Microsoft do protocolo SOAP. Dentro do Visual Studio.NET existe ainda a possibilidade de encapsular os componentes COM+ encontrados em uma empresa e disponibilizá-los como *Web services* para o mundo, aproveitando o investimento feito no desenvolvimento passado dentro da corporação. Estes componentes podem ainda utilizar a tecnologia COM TI (*COM Technology Integration*) e o HIS 2000 (*Host Integration Server*, antigo SNA Server) que permitem a execução de

transações CICS em mainframe como uma transação de componentes COM+ em plataforma baixa, novamente re-aproveitando o desenvolvimento passado em plataforma alta.

A nova linguagem C# foi apresentada pela Microsoft juntamente com a estratégia .NET, e o Framework .NET que é à base da tecnologia .NET para desenvolvimento de aplicativos.

TOLOMELLI (2002) afirma que a Microsoft, com o intuito de oferecer a maior flexibilidade possível aos aplicativos desenvolvidos em .NET, enviou ao ECMA (*European Computers Manufactures Association*, associação europeia de fabricantes de computadores), órgão de padronização equivalente a ISO (*International Standards Organization*) na Europa, a especificação dos itens citados acima para que sejam avaliados e transformados em padrão aberto da indústria, de forma que empresas interessadas possam estar desenvolvendo as suas próprias versões do Framework .NET para rodar em outros sistemas operacionais que não os fornecidos pela Microsoft, como MacOS, Solaris ou Linux. As aplicações na plataforma .NET são compiladas para uma linguagem intermediária denominada MSIL (*Microsoft Intermediate Language*), que por sua vez é compilada pelo Framework .NET para código nativo do sistema operacional onde será executada. Desta forma o mesmo aplicativo pode ser executado em qualquer sistema operacional que tenha um Framework desenvolvido para ele, mantendo a independência de plataforma da sua aplicação .NET.

1.2 ORGANIZAÇÃO DO TRABALHO

O Capítulo 2 abordará a proposta da plataforma .NET e dos conceitos de programação envolvidos nela, o que ela traz de novidade e suas quais são as suas vantagens.

O Capítulo 3 aborda o novo conceito de programação distribuída voltado para a Internet – os *Web Services*.

O Capítulo 4 tratará de uma comparação da plataforma .NET versus J2EE que também é uma tecnologia na produção de softwares para a *Web*.

O Capítulo 5 fará as considerações finais sobre a tecnologia .NET.

2. PLATAFORMA .NET

2.1 NET FRAMEWORK

Segundo SANTA'ANNA (2001) o .NET Framework oferece infra-estrutura de desenvolvimento, distribuição e execução de aplicações, componentes e *Web Services*. Ele é dividido em três partes principais: a *Common Language Runtime* (CLR), ASP.NET e as classes do Framework a qual é uma grande biblioteca extensível onde incorpora as funcionalidades de uma plataforma operacional. Do ponto de vista do desenvolvedor, significa API unificada, flexibilidade e interoperabilidade, sem rebaixar o nível de programação, ou seja, uma aplicação em Visual Basic.NET utiliza todo potencial do Framework para executar tarefas que em Visual Basic 6 somente seria possível com as API do Windows. Juntamente ao Framework, uma versão reduzida, o .NET Compact Framework, disponibilizará a infra-estrutura no desenvolvimento para dispositivos móveis, tais como, smart phones e PDAs (*Personal Digital Assistant*). Uma versão especial e com código aberto do .NET Framework para FreeBSD está sendo criada em conjunto com o Corel, onde provavelmente mais novidades surgirão neste segmento.

A Microsoft definiu a *Common Language Specification* (CLS), a qual é a regra que todas as linguagens (ou compiladores) devem seguir na exposição de itens de visibilidade externa, entre elas: classes, interfaces, métodos, campos, propriedades e eventos. Desta forma, um grande número de linguagens de programação é encontrado na plataforma .NET, ao contrário da plataforma J2EE que atualmente suporta apenas o Java e o desenvolvedor é obrigado a aprendê-la juntamente com a sua biblioteca de classes. A filosofia da plataforma .NET é exatamente oposta, ou seja, possuir o maior número de desenvolvedores de diversas linguagens, inclusive Java, cujo compartilhem um mesmo sistema de tipos comum e a mesma biblioteca de classes. Além disso, esta característica permite o *cross-language* para as linguagens as quais são classificadas como *CLS-compliant*. O desenvolvedor pode escolher e trabalhar com sua linguagem favorita na produção do *assembly*, unidade funcional de compartilhamento e reutilização na CLR, pode ser *.exe* ou *.dll*. Isso permite que uma classe em Visual Basic.NET pode herdar de uma classe em C#, o qual, por sua vez, receberá uma classe escrita em Cobol, no que possibilita times de desenvolvimento integrem ou unifiquem seus *frameworks*, bibliotecas de classes ou *object models* e compartilhem o ambiente

integrado de desenvolvimento do Visual Studio.NET e suas ferramentas (Visio UML, XML Editor, Crystal Reports, entre outras).

O .NET é baseado em um ambiente CLR (*Common Language Runtime*) que gerencia todos os aspectos de runtime do código. O termo “common” no nome significa que todos os componentes .NET são executados no mesmo runtime, independente da linguagem em que foram desenvolvidos. O CLR pode ser comparado a um cobertor aconchegante que envolve o código, fornecendo gerenciamento de memória, um ambiente seguro em que ele possa se desenvolver, transparência na localização de objetos, gerenciamento simultâneo e acesso aos serviços do sistema operacional subjacente. Como o CLR gerencia todos esses aspectos do comportamento de um objeto, o código do CLR é chamado de código gerenciado.

Na primeira versão do Framework encontramos os compiladores para Visual Basic (*vbc.exe*), C# (*csc.exe*), Visual C/C++ (*cl.exe /CLR*) e JScript (*jsc.exe*), os quais produzem um *Intermediate Language* (IL), instruções são transformadas em código nativo pelo compilador *Just-In-Time* (JIT) da CLR, no momento de execução ou *deploy* e colocadas num *cache* para reutilização, tudo gerenciado pelo mecanismo de execução comum. A

Figura 1 demonstra este processo.

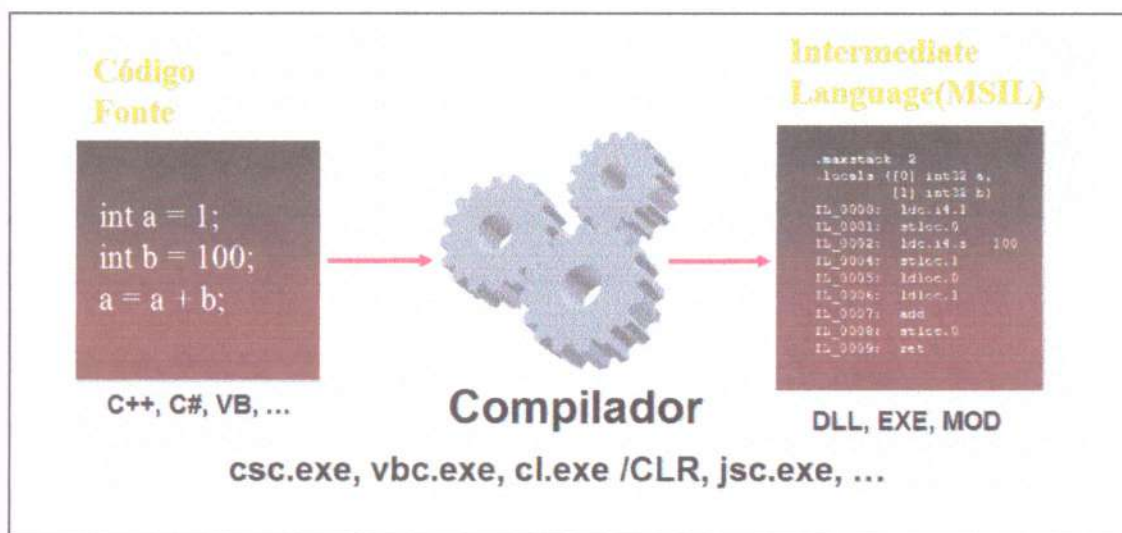


Figura 1: Compilação na plataforma .NET, MSIL é produzido

Os compiladores produzem um código, muito semelhante ao *macro-assembly*, chamado de *Microsoft Intermediate Language* (MSIL), porém a estrutura do código é baseada em instruções reduzidas e operações através da *stack*. A vantagem é que este código funcionará em múltiplas plataformas onde o *runtime* estiver presente. O Framework fornece o

MSIL Assembler, *ilasm.exe*, onde gera um *portable executable* (PE) estendido com informações necessárias para OS Loader localizar o *entry-point* do Runtime Host.

Apesar de algumas semelhanças com a JVM (*Java Virtual Machine*), o CLR é um sofisticado mecanismo de execução. Através dos compiladores just-in-time (JITters), o MSIL transforma-se em código nativo, colocado em *cache* e executado. O grupo de instruções do MSIL foi projetado com foco em diversos compiladores, diferenciando-o dos moldes tradicionais de *byte code* ou *tree code*. Portanto, o código executado não é interpretado assim como é usual na JVM, código nativo é requisito na plataforma .NET.

2.2 VANTAGENS DA .NET FRAMEWORK

A principal idéia da .NET Framework é usar um modelo baseado em componentes como a única maneira de programar para o sistema operacional. Este é o modelo mais flexível e produtivo entre todos os disponíveis, bastante semelhante do Delphi ou Visual Basic. Isto, no entanto não é uma tarefa fácil:

- O modelo de objetos deve é definido no próprio sistema operacional;
- Suporta várias linguagens de programação;
- Diversos conceitos como propriedades e eventos são suportados nativamente, não apenas os métodos e campos, como normalmente ocorre;
- Criar instâncias e até mesmo herdar uma classe da outra deve ser permitido, mesmo que só tenhamos o código binário disponível e não saibamos sequer a linguagem de desenvolvimento da classe original;
- Os objetos devem é “autodocumentados”; ou seja, incluir informações detalhadas dos tipos que estão lá dentro. Este recurso não só facilita a chamada das classes, mas também é fundamental para permitir a validação do uso e manutenção da integridade do sistema de tipos em tempo de execução;
- O gerenciamento de memória é feito pelo sistema operacional, para que um programa possa “passar um objeto” para outro programa sem dificuldades e não precisar se preocupar com alocação de memória;
- Existir uma preocupação com controle de versões; um programa pode precisar de uma versão mais antiga e outro de uma versão mais nova da mesma classe; ambos devem ser satisfeitos, mesmo que estejam rodando ao mesmo tempo.

Os executáveis .NET são independentes do sistema operacional. Basta que haja um “sistema de runtime” que possa compilar os programas para a CPU real e que as classes necessárias estejam implementadas para o programa rodar. Nada nos programas compilados amarra a alguma CPU ou sistema operacional.

2.3 LINGUAGENS DO .NET FRAMEWORK

Neste capítulo falaremos das novas linguagens apresentadas pela Microsoft que formam a base de sustentação da plataforma .NET.

2.3.1 A Linguagem C#

O C# (denomina-se “C sharp”), uma linguagem de programação moderna, orientada a objeto e construída para explorar o poder dos serviços da Internet baseados em XML na plataforma .NET. Com herança do sistema de desenvolvimento Visual C++, o C# possibilitará a milhares de desenvolvedores de C e C++ a utilização de habilidades para construir, de maneira rápida, sofisticadas aplicações baseadas em XML para a plataforma .NET. Para simplificar a integração e interoperabilidade, a Microsoft está trabalhando com ECMA, um grupo que estabelece padrões internacionais para criar um padrão C#, possibilitando a muitos fornecedores a entrega da linguagem e ferramentas de suporte.

Com seu design funcional orientado a objeto, o C# é uma ótima escolha para que os desenvolvedores construam uma ampla gama de aplicações e componentes de alta performance para a *Web*, com menos linhas de código e pouca chance de ocorrência de erros.

A sintaxe do C# é muito parecida com a do C++. A princípio, quando uma construção do C++ não oferece problemas, ela é usada. Este é o caso, por exemplo, nas declarações de variáveis e “*loops*”. Mas o C# modifica bastante o C++ e não tem a pretensão de manter a compatibilidade, apenas a “familiaridade”.(GORDON, 2002)

As principais características conforme GORDON (2002) do C# são as seguintes:

- Todas as variáveis e código são declarados no escopo de classes. É possível, contudo, declarar tipos (“*structs*” e enumerações) fora do escopo de classes.
- Tipagem forte. As enumerações são tipos próprios e incompatíveis com outras enumerações. Existe um tipo lógico (*bool*) incompatível com inteiros. Os tipos intrínsecos são: lógico, inteiros de vários tamanhos pré-definidos (8, 16, 32 e 64 bits, com e sem sinal), ponto flutuante IEEE de 4 e 8 bytes, *string* e decimal. Só existe um

único tipo “*char*”, também incompatível com inteiros. Tanto o “*char*” como a “*string*” armazenam apenas caracteres *Unicode* (16 bits por caractere). O tipo “*decimal*” é armazenado como uma mantissa binária de 96 bits e um expoente na base 10, para um total de 128 bits. A precisão do decimal é de pelo menos 28 dígitos decimais, o que evita a maioria dos erros de arredondamento comuns aos formatos de ponto flutuante em binário. Existe também “*structs*”, boas para serem usadas em situações “leves”, como por exemplo, uma coordenada (X, Y), quando o custo em memória e tempo de execução de uma classe seria grande e desnecessário.

- Os objetos e “*arrays*” são necessariamente alocados dinamicamente no “*heap*” com o uso do operador “*new*”.
- O índice dos “*arrays*” começa com zero e sua faixa é sempre verificada em tempo de execução.
- O C# inicializa a maioria das variáveis com zero e efetua diversas verificações de lógica, como se uma variável foi atribuída antes de ser usada, se um parâmetro de saída foi atribuído e se um inteiro teve sua faixa violada.
- Todas as conversões de tipo (“*cast*”) são validadas em função do tipo real da variável em tempo de execução, sem exceções.
- O operador “.” é usado em diversos lugares, quando em C++ seriam usados “.”, “::” e “->”.
- Existe um outro tipo de loop além dos oriundos do C (*for*, *while*, *do..while*), o “*foreach*”, usado para varrer todos os elementos de um *array* ou “coleção”.
- O “*switch*” elenca opções mutuamente exclusivas, por definição, e pode ser usado com strings. O “*break*” depois de cada opção é obrigatório.
- O único mecanismo de tratamento de erros do C# é a *exception*.
- Não existem macros, mas existe compilação condicional (*#ifdef*, etc).
- Os *templates* não são suportados, pelo menos por enquanto. Talvez seja possível criar um mecanismo semelhante aos *templates* no futuro. De qualquer forma, o C# tem um suporte bastante abrangente a “*reflections*”, o que pode substituir *templates* em várias situações.
- O C# suporta sobrecarga de funções e de operadores, como o C++, mas não tem argumentos “*default*”.

- O C# possui operadores de conversão, mas existe uma sintaxe para indicar se a conversão deve ser implícita ou explícita. O construtor não é usado como operador de conversão.

O C# é um C++ “limpo”, com várias boas idéias comuns em outras linguagens e algumas novas, como “*boxing*”, “*delegates*”, “*garbage collection*” e “*attributes*”. Ela é muito atraente para programadores que desejam migrar para a plataforma “.NET”.

2.3.2 A Linguagem Visual Basic.NET

O Visual Basic.NET é a nova versão do Visual Basic, mas invés de simplesmente adicionar algumas características novas ao Visual Basic 6.0, a Microsoft fez uma reengenharia completa do produto para podermos mais facilmente escrever aplicações distribuídas tais como n-tier na *Web* e em empresas. É o avanço de uma linguagem orientada a eventos e baseada em objetos, O Visual Basic 6, para uma linguagem orientada a objetos e focada no desenvolvimento de componentes. No Visual Basic.NET o essencial é a orientação a objetos e a biblioteca de classes para torná-lo diferencial, visto que todas linguagens modernas dependem destes itens.

O VB.NET tem dois novos tipos de formulários (*Windows Forms* e *Web Forms*); além de uma versão nova do ADO (focada no acesso de data *sources* desconectados); assim como um novo alinhamento da linguagem, removendo os *keywords* do legado, melhorando a segurança dos tipos, e expondo as construções de baixo nível que os desenvolvedores avançados requerem.

Estas características abrem novas portas para o desenvolvedor Visual Basic: com *Web forms* somado ao ADO.NET podemos desenvolver rapidamente *Web Sites* com alta escalabilidade; com o suporte a herança, linguagem ganha novas características de programação verdadeiramente orientada a objetos; os *Windows Forms* suportam de forma nativa a acessibilidade e a herança visual; e distribuir suas aplicações é agora tão simples quanto copiar seus executáveis e componentes de um diretório para outro.

Agora o Visual Basic.NET é inteiramente integrado com as outras ferramentas de desenvolvimento do Microsoft Visual Studio.NET. Não somente podemos desenvolver componentes das aplicações em linguagens de programação diferentes, como suas classes podem herdar classes escritas em outras linguagens usando a herança entre linguagens. Com o *debugger* unificado, você pode eliminar erros das aplicações em múltiplas linguagens, independentes de estarem funcionando em máquinas locais ou remotas.

Havia duas opções a serem consideradas no início do projeto do Visual Basic.NET – adaptar o código já existente para funcionar dentro da estrutura .NET ou uma mudança mais radical. Para que se pudesse ter as características pedidas pelos usuários (por exemplo, herança) e habilitar o acesso completo a nova plataforma, além de assegurar o avanço do Visual Basic na geração seguinte de aplicações *Web*. A decisão escolhida foi construir do início uma plataforma nova.

Um dos objetivos principais era assegurar que o código Visual Basic tivesse total interoperabilidade com o código escrito em outras linguagens, tais como Microsoft Visual C# ou Microsoft Visual C++, e trazer para o desenvolvedor Visual Basic, de forma simples, toda a potência da estrutura .NET, sem recorrer aos *workarounds* de programação necessários tradicionalmente para fazer o trabalho com Windows APIs. O Visual Basic.NET tem agora compatibilidade total com Visual C++ em relação a tipos de variáveis, *arrays*, *user-defined types* e classes. (LOWY, 2002)

Quando o projeto é atualizado, a linguagem é modificada por qualquer mudança de sintaxe e seus formulários do Visual Basic 6.0 são convertidos para Windows Forms. Na maioria dos casos, você terá que fazer algumas mudanças no seu código, depois da atualização. Isso é necessário por que certos objetos e características não possuem equivalência no Visual Basic.NET, ou podem ter uma equivalência tão dissimilar para uma atualização automática. Depois da atualização, você pode também alterar sua aplicação para levar vantagem em algumas das novas características do Visual Studio.NET.

Para ajudar a fazer as mudanças, depois que o projeto é atualizado, Visual Basic.NET adiciona um *Upgrade Report* no projeto, especificando qualquer problema, e insere comentários no seu código atualizado alertando para afirmações que necessitaram ser alteradas.

Componentes compilados com Visual Basic.NET tem sutis diferenças para componentes compilados em Visual Basic 6.0. Para iniciantes, porque objetos do Visual Basic.NET são liberados através *garbage collection*, quando objetos são explicitamente destruídos, pode haver um atraso antes deles serem realmente removidos da memória. Há diferenças adicionais como quando as *variant/objects* mudam de descrição neste documento. O resultado da combinação dessas diferenças são aquelas aplicações de Visual Basic.NET que terão condutas de tempo de execução similar, mas não iguais às aplicações do Visual Basic 6.0.

2.3.3 A Tecnologia ASP.NET

O principal objetivo da arquitetura .NET é permitir ao usuário o fácil acesso a seus aplicativos e dados em qualquer lugar, a qualquer hora e usando qualquer dispositivo. Para concretizar esta visão existem vários componentes, de servidores a ferramentas de desenvolvimento.

É uma das tecnologias mais interessantes da plataforma .NET, que juntamente ao Visual Studio.NET trás o desenvolvimento RAD e orientado a componentes para a *Web*. Sobre ele vem todo o suporte para desenvolvimento com *Web Forms*, *Web Controls* e principalmente XML *Web Services*. *Mobile Web Forms* também faz parte do contexto e seu propósito é o desenvolvimento de aplicações com suporte múltiplo dispositivos e tudo isso pode ser a partir de uma só fonte. No ASP.NET o desenvolvimento “flui” como em Visual Basic, e seu ponto forte trata-se do código ser compilado, a lógica de negócio isolada em componente e com layout de tela separada usualmente com a extensão *.aspx*. Ele torna o desenvolvimento de aplicativos baseados em *Web* tão simples como o desenvolvimento de aplicativos em Visual Basic. O programador não tem sequer que conhecer HTML ou “scripts de cliente” (usualmente JScript) para criar aplicativos sofisticados.(NOYES,2002)

A segurança – especialmente a autenticação e a autorização – é muito mais facial de ser implementada com ASP.NET do que com ASP. A autenticação é o processo pelo qual um usuário comprova a sua identidade ao sistema de um computador. A autorização é o processo de restringir o acesso de um usuário aos recursos, baseado na identidade.

Algumas das vantagens da utilização do ASP.NET:

- O modelo de desenvolvimento é muito mais parecido com o Visual Basic do que com outras ferramentas HTML.
- O código (arquivo “.cs”) é separado da apresentação (arquivo “.aspx”). O “*Web designer*” pode mudar a aparência da aplicação sem afetar o código.
- O seu programa ter performance de código compilado e seu fonte está seguro; seu provedor só enxerga os fontes do seu código se você quiser.
- Os componentes mantêm automaticamente o estado sem uma única linha de código. Imagine quanto código seria necessário apenas para manter o estado do “*ListBox*”.
- Temos o poder de uma linguagem “de gente grande” como o C# de forma a permitir a escrita de código mais seguro e poderoso. Não existe mais a necessidade de escrever

componentes COM apenas para complementar coisas difíceis de fazer em VBScript ou JScript.

- Podemos usar o poder do C# e VS.NET para melhor organizar o código e dividi-lo entre os diversos membros de uma equipe

O ASP.NET é um ambiente produtivo para o desenvolvimento de aplicativos baseados em HTML. Ele torna o desenvolvimento de aplicativos *Web* tão simples como o desenvolvimento de aplicativos Windows usando uma ferramenta RAD como o Visual Basic. (LOWY, 2002)

3 WEB SERVICES – SOFTWARE COMO SERVIÇO

3.1 UMA VISÃO INICIAL

Hoje em dia, a maioria dos sites *Web* são como ilhas isoladas de dados e lógica. Eles não interoperam com outros sites comerciais, e se o fazem, cria esta interoperabilidade exige grandes investimentos em desenvolvimento personalizado para ter os dois sites trabalhando juntos. Se estes dois sites quiserem interoperar com um terceiro, seria necessário um novo ciclo de desenvolvimento personalizado. Adicione a este contexto a necessidade de um poderoso *firewall* corporativo para proteger o seu site e rede, e você verá o processo doloroso de estabelecer uma interação útil entre os sites.

O .NET promete aliviar esta dor e facilitar a interoperação não só de dois, mas de muitos sites utilizando os *Web Services*.

Os *Web Services* removem o obstáculo do *firewall* à interoperabilidade traduzindo as instruções de linguagens específicas que seriam bloqueadas por um *firewall* em texto HTTP que o atravessará.

Um *Web Services* é uma aplicação lógica, programável, acessível, que usa os protocolos padrão da internet, para que se torne possível a comunicação transparente de máquina-para-máquina e aplicação-para-aplicação.

Para desenvolvermos um *Web Service* utilizamos tecnologias pensadas sob esta visão, como o SOAP (Simple Object Access Protocol), WSDL (*Web Service Description Language*), e HTTP (HyperText Transfer Protocol), usadas para passar mensagens através das máquinas. Estas mensagens podem variar muito na complexidade, desde a chamada de um método, às submissões da ordem de compra.

3.2 O QUE É SOAP?

O SOAP é o protocolo simples do acesso do objeto. SOAP é baseado em XML e descreve um formato de messaging para comunicação de máquina-à-máquina. Contem também diversas seções opcionais que descrevem os atendimentos do método (RPC) e que detalham a emissão de mensagens do SOAP sobre o HTTP.

O padrão SOAP foi criado em cima da linguagem XML, de modo que as definições dos métodos e argumentos do objeto servidor somente são apresentados através de um arquivo XML no momento da chamada do método, garantindo que qualquer alteração no

objeto servidor seja verificada antes da chamada. Além disso, SOAP só exige a necessidade de conexão com o objeto servidor no momento da chamada do método específico.

SOAP é também um padrão do W3C e adotado por diversas empresas entre elas a Ariba e a IBM. Esta última inclusive ajudou na especificação inicial juntamente com a Microsoft e já possui um site com diversos objetos SOAP disponíveis e que podem ser chamadas remotamente. Algumas das implementações destes objetos está rodando em diversos servidores *Web* diferentes, e no site XMethods possui soluções para implementações de SOAP para programadores Perl e Java.

Assim é possível desenvolver um componente SOAP dentro do Visual Studio.NET, utilizando Visual Basic.NET ou C#, por exemplo, e acessar este objeto através de um script Perl no seu servidor *Web*. A recíproca também é verdadeira. Você pode escrever um componente SOAP em Java rodando num servidor Apache em Linux, e acessar este objeto de uma aplicação .NET rodando num dispositivo com Pocket PC ou Windows CE. É este tipo de interatividade que deve ocupar a mente dos gerentes de projetos e CIOs do hoje em dia, pensando no futuro das empresas onde trabalham.

Este é um pedido típico do SOAP (incluindo o cabeçalho HTTP) para um método EchoString nomeado atendimento do RPC, que faça o exame de uma string como um parâmetro:

```
POST /test/simple.asmx HTTP/1.1
Host: 131.107.72.13
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://soapinterop.org/echoString"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns="http://soapinterop.org/"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body
    soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <tns:echoString>
```

```

    <inputString>string</inputString>
  </tns:echoString>
</soap:Body>
</soap:Envelope>

```

Como você pode ver, o nome do método é codificado como o XML: <tns:echoString>, e o parâmetro é codificados como <inputString>.

O método C # que corresponde a isto é:

```
public String echoString(String inputString);
```

Está é a resposta do servidor:

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:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns="http://soapinterop.org/"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <tns:echoStringResponse>
      <Return>string</Return>
    </tns:echoStringResponse>
  </soap:Body>
</soap:Envelope>

```

E assim caminham as informações. Através do Visual Studio.Net podemos criar *Web Services* utilizando qualquer linguagem .Net com grande facilidade.

3.3 ARQUITETURA DO *WEB SERVICE*

Inicialmente vamos observar outra vertente do uso dos *WebServices*, constatando a existência de uma tendência aonde venhamos a utilizar vários aplicativos que acessem serviços na internet (*WebServices*) ao invés de browser utilizaremos programas como, por exemplo, o Napster, que permite buscas em discos de milhares de usuários, que compartilham musicas através de um serviço.

Para que possamos desenvolver este tipo programas, é necessária uma forma rápida e fácil de escrever o código que se comunica com outros programas sobre o Internet. A idéia não é nova; temos um grande número de tecnologias existentes que permitem este tipo de uma comunicação: RPC, DCOM, e os serviços da fila de mensagens (MSMQ). Cada uma destas técnicas é quase completa, porém todas têm uma falha fatal: trabalham somente com sistemas similares. O MSMQ fala somente com outro MSMQ, um cliente de DCOM somente a um servidor DCOM, e assim em diante.

Precisamos encontrar uma forma de que um programa possa trocar mensagens com outro, de forma independente, não somente do sistema operacional, mas também das instruções internas dos programas.

3.4 SOLUÇÃO DA ARQUITETURA

A única maneira para tratar o número enorme de entidades heterogêneas na Internet é usar o denominador comum mais baixo. Ou seja, quando os bytes são transferidos de um site para outro, o processo necessita utilizar algum padrão que todos na Internet, suportem. O protocolo mais comum de transferência na *Web* é o HTTP. O padrão entre plataformas para pegarmos uma informação, codificá-la, transferi-la sobre HTTP é o XML. A Microsoft uniu estas idéias e desenvolveu o conceito de *WebService* - uma maneira "seamless" de objetos em um servidor aceitarem pedidos dos clientes usando o HTTP e o XML.

Para criar um *WebService*, simplesmente escrevemos uma aplicação servidora Microsoft.NET como se ele fosse acessado por clientes locais, informando que desejamos que ele seja disponível como um cliente *Web*, e deixe que, o ASP.NET faça o resto. Isto automaticamente se encaixa acima de uma infra-estrutura pré-fabricada que aceite pedido entrante através do HTTP e os trace os mesmos para chamar o objeto.

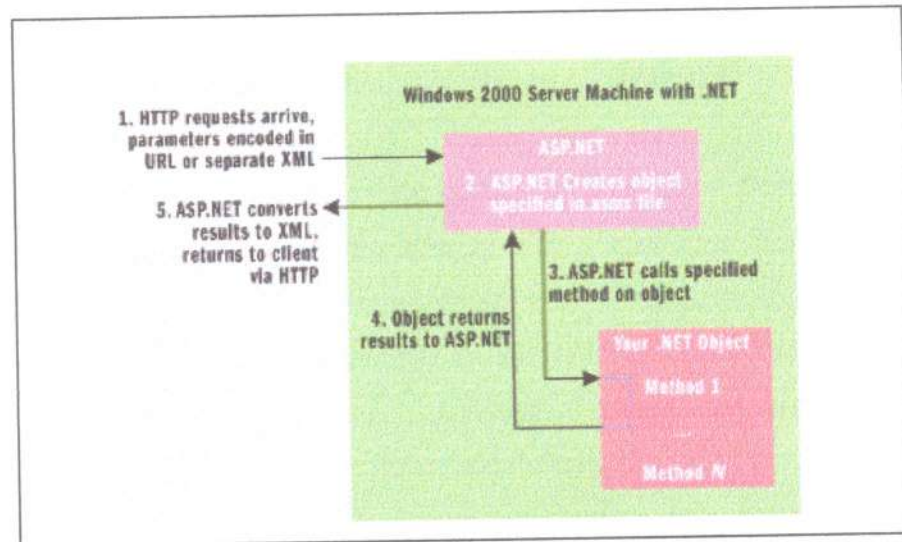


Figura 2: Os objetos podem ser acessados por qualquer pessoa na Web

No lado do cliente, o .NET fornece as classes do *proxy* que têm o fácil acesso, funções baseadas em *WebServices* fornecidos por todos os servidores que aceitam pedidos do HTTP, como mostrado na Figura 3. Uma ferramenta de desenvolvimento lê a descrição do *WebService* e gera uma classe *proxy* que possui funções na linguagem utilizada para desenvolver o cliente. Quando o cliente chama uma destas funções, a classe do *proxy* gera um pedido do HTTP e emite-o ao servidor. Quando a resposta volta do servidor, a classe do *proxy* analisa gramaticalmente os resultados e retorna-os da função. Isto permite que sua função baseada em cliente interaja “*seamlessly*” com todo o *WebServer* que se comunica via HTTP e XML.

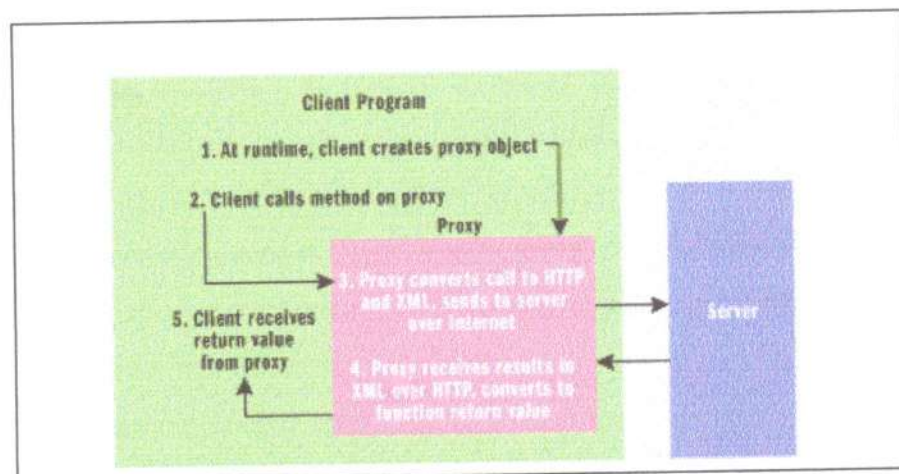


Figura 3: Escrevendo um Web Service

Escrever o *WebService* muito simples; pode-se usar o notepad. O programa começa com a saudação padrão `<%@...%>` do ASP.NET. Nela, a diretiva orientadora do *WebService* diz a ASP.NET que o código na página deve ser exposto como um *WebService*. O atributo da

língua diz ao ASP que linguagem compilar o código da página. O ASP.NET usará Visual Basic.NET para compilar o código.

O resto da página contém o código que executa a classe. Primeiramente a diretriz orientadora das importações, uma característica nova do Visual Basic.NET, diz ao compilador para importar os "namespaces". O termo "namespace" é conceitualmente idêntico a uma referência em no projeto em Visual Basic 6.0. O ASP.NET compilará o código deste exemplo "just-in-time" quando o pedido chegar de um cliente. Os nomes que seguem as importações dizem ao compilador que grupos de funcionalidade devem ser incluídos na referência. Veremos uma *keyword brandnew* (Herança) na extremidade da linha: Herança *WebService*. Isto representa uma das principais características do Visual Basic.NET - suporte a um conceito de orientação-objeto chamado herança.

Quando se diz que a nova classe *TimeService* herda da classe *WebService*, está se informando ao compilador para fazer o exame de toda a classe providenciada pelo *WebService* (conhecido como a classe baixa) e incluí-lo em sua nova classe de *TimeService* (sabida como a classe derivada). Podendo haver reutilização do código da classe que herdada, podemos adicionar, alterar, e cancelar as funcionalidades dessa classe.

A classe do *WebService* é fornecida pelo novo .NET *runtime library*, assim como muitos objetos usados Visual Basic 6.0 são providenciados pelo sistema operacional. *WebService* possui toda a arquitetura pré-fabricada requerida para segurar os pedidos entrantes do HTTP para distribuí-los aos métodos apropriados ao objeto.

Para definir os métodos e as funções da classe. Vamos pensar outra vez, de forma similar ao Visual Basic 6.0, mas com uma novidade: temos que adicionar um atributo novo, `<WebMethod(>`, a cada método que se queira que seja exposto aos clientes da *Web*. Isto diz ao ASP.NET que o método deve ser exposto aos clientes como um serviço *Web*. Apenas sua classe pode conter métodos públicos e privados, assim podendo conter métodos que estão expostos aos clientes da *Web* ou não.

Podemos acessar o código gerado através do IE 5.5, teremos a Figura 4. ASP.NET detecta o acesso à página do próprio *WebService* (ao contrário de um dos métodos dentro do serviço), e responde com uma página da informação sobre o serviço. Isto lhe mostra todos os métodos que ASP.NET encontrou na classe (neste caso, somente *GetTime*).

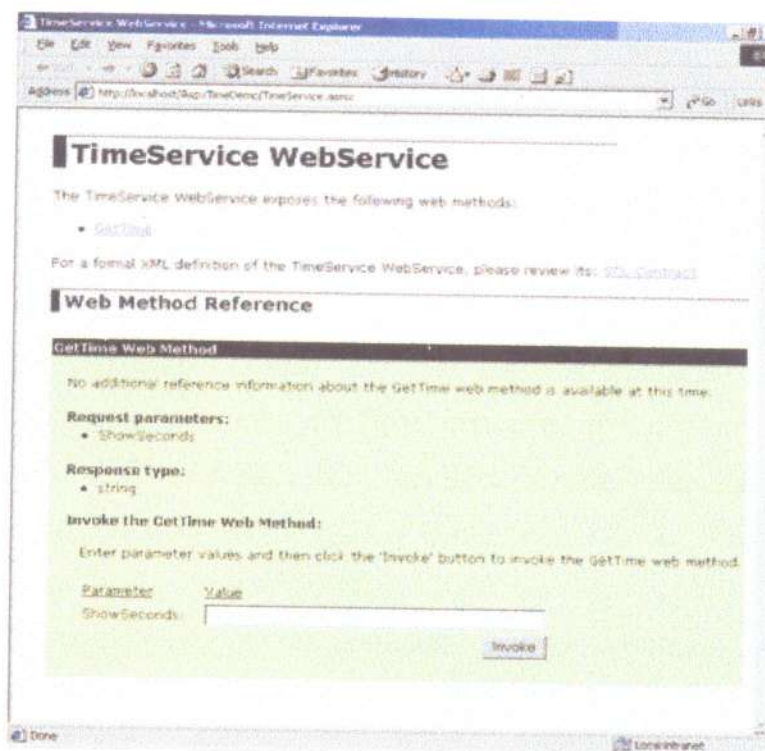


Figura 4: TimeService.asmx em um navegador

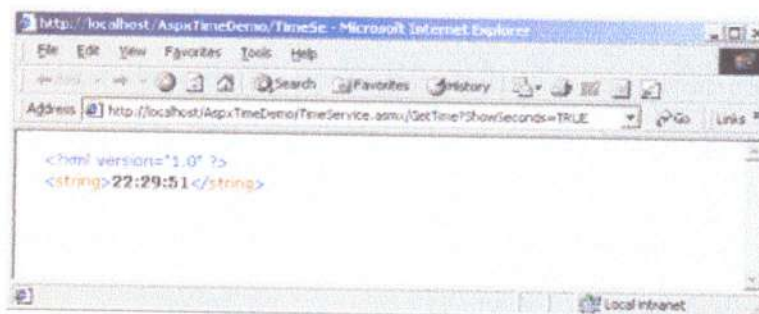


Figura 5: Mostrando os segundos

3.5 DESCRIÇÃO DOS WEB SERVICES: OS ARQUIVOS SDL

A fim permitir que os programadores desenvolvam as aplicações dos clientes que usam este *WebService*, é preciso fornecer as informações necessárias durante os projetos e o processo de programação. O ASP.NET fornece um serviço descritivo. Quando se compila um *WebService*, o ASP.NET produz um arquivo que lista os protocolos suportados pelo serviço, pelos métodos fornecidos e pelos parâmetros requeridos por esse serviço. O arquivo é codificado em XML utilizando um vocabulário chamado Service Description Language (SDL). Pode-se considerar o arquivo SDL de um *WebService* como um contrato, porque lista as funcionalidades que o serviço é capaz e lhe informa como pedi-las.

O arquivo SDL é um tanto complexo, assim para facilitar o entendimento foram separadas algumas parcelas para discutir os seus princípios; O elemento `serviceDescription` é a raiz do arquivo. Tudo dentro é parte da descrição do *WebService*. Existe um atributo que dá um nome ao *WebService*, neste caso, *TimeService*. A raiz contém dois tipos interessantes de sub-elementos. O primeiro é uma descrição do protocolo, que diz a um colaborador interessado no cliente que protocolos o serviço suporta e como codificar os dados do pedido e da resposta para esse protocolo. Qualquer pessoa ou ferramenta de desenvolvimento que olha o SDL saberá que o serviço nomeado *TimeService* pode ser alcançado através deste protocolo. O arquivo cheio de SDL contém também as entradas que dizem que pode ser alcançada através da operação HTTP Post e também através do Simple Object Access Protocol (SOAP) - um híbrido do HTTP/XML.

Olhando dentro da descrição do protocolo, pode-se ver que o serviço contém um método chamado `GetTime`, que fica na URL especificada. O pedido requer um único parâmetro chamado `ShowSeconds`.

O fato que usando a operação HTTP GET, a operação implica que ele está sendo passado para o formulário em forma de uma string. A resposta volta em XML, outra vez no formulário, como uma string.

A segunda parte interessante no arquivo SDL é o elemento `<schema>` de W3C-standard. Este com a definição abstrata do serviço, sem consideração para como é alcançado em todo o protocolo particular ou uniforme onde vive.

Podemos pensar nisso como uma definição de interface em um tipo de biblioteca; descrevendo uma série de métodos, mas não uma execução deles.

4 .NET VERSUS JAVA

Tecnicamente, as tecnologias J2EE (Java 2 Enterprise Edition) e Microsoft .Net são bem parecidas. Ambas fornecem arquiteturas de componentes para a construção de sistemas distribuídos, oferecendo recursos similares para o desenvolvimento de aplicações Internet e Web Services. As duas tecnologias usam também o mesmo conceito de máquina virtual, em que as linguagens de programação são compiladas para um código intermediário, com velocidade de execução muito próximas.

4.2 AS PRINCIPAIS DIFERENÇAS DO .NET E J2EE

.NET	J2EE
- .NET é uma plataforma inteiramente nova. - Produto da Microsoft que implementa uma série de serviços - Dependência de plataforma: MS Windows - Programador retorna um Visual Studio.Net - .NET Framework retorna o que o cliente escolhe (browser, palm) - Somente componentes stateless - Suporta a múltiplas linguagens (C#, VB, C++) - Suporta OO entre componentes de linguagens diferentes (herança, polimorfismo) - Poder de execução da Microsoft - Ferramentas RAD – Independência de dispositivos do cliente e Web Service - Mais simples de construir - Menor custo de aquisição	- A plataforma Java encontra-se disponível no mercado há mais sete anos. - Especificação mantida por um consórcio de empresas liderados pela SUN - Dependência de plataforma: Java - Programador escolhe o que o cliente retorna (XML/HTML) - Os componentes podem manter “estado” entre as chamadas do cliente (componentes Stateful) - Suporta Somente Java - Grande suporte da indústria de software - Atrasado em relação a Web Services - Sistemas que precisam rodar em múltiplas plataformas - Mais opções de serviço e mais flexibilidade

5 CONCLUSÕES

Foi observado por meio desse trabalho que estamos caminhando para um cenário compatibilidade entre sistemas e dispositivos. Não podemos mais focar isoladamente cada sistema ou cada decisão. Todo e qualquer sistema deverá estar preparado para interagir com outros, para oferecer e receber informações de outros sistemas da própria ou de terceiros, e também usufruir os recursos disponibilizados.

O cenário atual é favorável à J2EE, mas que muitas vezes a vantagem é somente de já haver comprovado alguma qualidade, sendo perfeitamente possível que a .NET venha a possuir a mesma vantagem, porém essa plataforma terá um longo desafio pela frente. O ponto positivo desse cenário é que os padrões XML e Web Service oferecem um grau inédito de interoperabilidade entre as duas plataformas.

Não importa quem vai ficar com a fatia maior do mercado, e sim o benefício que cada uma trará para facilitar a construção de sistemas que precisam cada vez mais está integrados, num mundo que se torna cada vez mais heterogêneo.

REFERÊNCIAS BIBLIOGRÁFICAS

- GORDON, Alan. Integre as tecnologias .NET. **Revista .NetMagazine**. n. 1, fev. 2002.
- LOWY, Juval. Os Web Services transpõem o Firewall.. **Revista .NetMagazine**. n. 1, fev. 2002.
- LOWY, Juval. Tome um novo rumo com o .NET. **Revista .NetMagazine**. n. 1, fev. 2002.
- LOWY, Juval. Disponível em: <www.componentware.net> Acesso em 16 de nov.2002.
- MICROSOFT. Disponível em: <<http://microsoft.com>> Acesso em 11 de nov.2002.
- NOYES, Brian. Migre para o .NET. **Revista .NetMagazine**. n. 1, fev. 2002.
- O'KELLY, Peter. Conheça a estratégia .NET. **Revista .NetMagazine**, n. 1, fev. 2002.
- PORTALPROGRAMAÇÃO. Disponível em: <<http://www.portaldaprogramacao.com>> acesso em 11 de nov.2002.
- PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO – PUC. Disponível em : <<http://www.puc.br>> Acesso em 08 nov.2002.
- Revista Developers Magazine, número 62, título **Web Development**, Outubro de 2001.
- Revista Developers Magazine, número 66, título **J2EE x .NET – Aonde Está o Futuro do Mercado**, Fevereiro de 2002.
- S2. Disponível em: <www.s2.com.br> Acesso em 16 de nov.2002.
- TOLOMELLI, Leonardo. Disponível em: <<http://microsoft.com/brasil/net/default.asp>> Acesso em 11 nov.2002.
- UNIVERSIDADE DE SÃO PAULO – USP. Disponível em: <http://www2.usp.br/> Acesso em 08 de nov.2002.
- W3. Disponível em: <<http://www.w3.org/TR/SOAP/>> Acesso em 16 de nov.2002.