

Capítulo 13

WebServices

O que são WebServices

Para explicar de maneira adequada o significado da tecnologia *WebServices*, é interessante fazer uma breve introdução aos problemas atuais no mundo da tecnologia. Com o grande avanço de sistemas operacionais, bancos de dados, hardware, software, enfim, todo o tipo de tecnologia que envolve o mundo dos negócios, vem surgindo a necessidade de compartilhamento de informações entre parceiros comerciais, governo e sociedade e até mesmo entre departamentos de uma empresa. Acontece que interligar diferentes plataformas, bancos de dados, operações, entre outros conceitos, é uma tarefa muito complicada e trabalhosa.

Imagine o seguinte cenário (*figura 13.1*) :

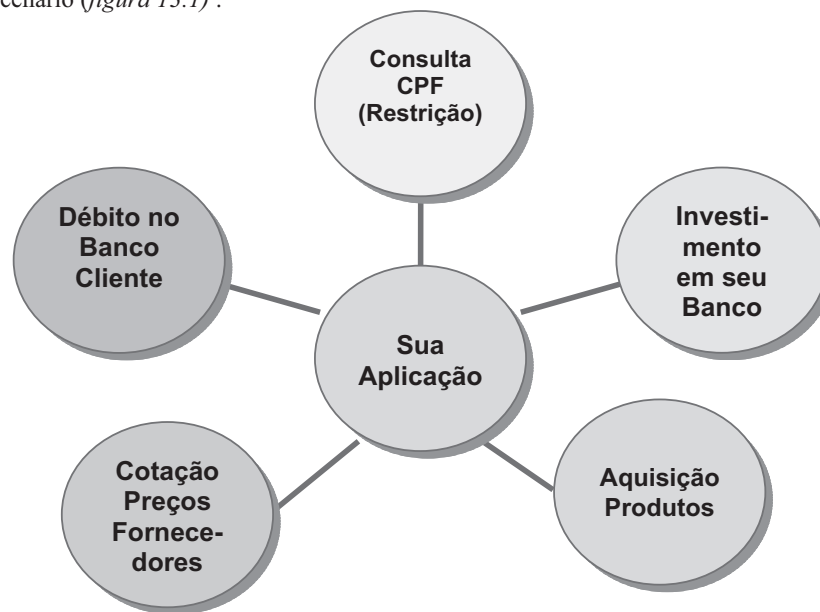


Figura 13.1 Cenário de uma aplicação

A *figura 13.1* ilustra uma situação comum nos dias de hoje, onde temos diversas operações com tecnologias e parceiros diferentes. Sem dúvida que a maioria das operações poderá ser desenvolvida com um grande trabalho em equipe, exigindo um bom trabalho de pesquisa junto aos parceiros e inúmeras horas de desenvolvimento.

É justamente neste cenário que surgiu a tecnologia *WebServices*, para impulsionar os sistemas legados e facilitar a integração de parceiros comerciais.

Pegando carona na primeira fase do *WebServices*, foram implementadas e idealizadas novas funções, com o intuito de promover e consolidar operações semelhantes. A *figura 13.2* ilustra este cenário.

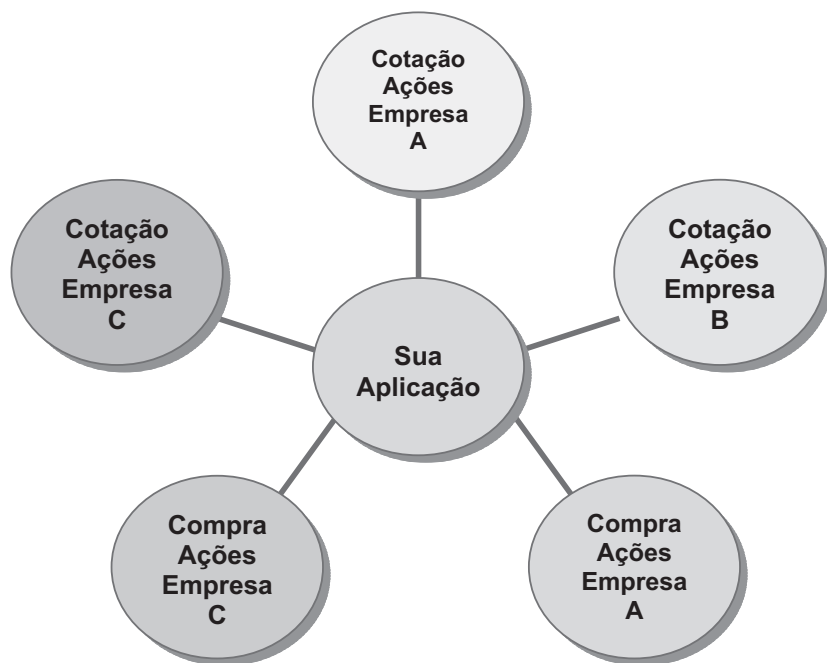


Figura 13.2 Cenário de *WebServices* semelhantes

Analisando a *figura 13.2*, temos uma aplicação que utiliza *WebServices* de diferentes fornecedores para fazer cotações de ações, assim como efetuar a aquisição. Um exemplo bastante utilizado, é a cotação do câmbio atual. Veja o cenário na *figura 13.3*.

Imagine o cenário ilustrado pela *figura 13.3* onde sua aplicação necessita ter 24 horas/dia (lembre-se de que o mundo não pára) a cotação do câmbio de diferentes países, onde seria uma catástrofe imaginar a quebra dos servidores.

O cenário da *figura 13.3*, ilustra uma operação de cotação de câmbio, utilizando *WebServices* semelhantes, onde não pode haver queda de servidores, pois a missão é crítica. Bem, do seu lado tudo sem problemas, pois existe uma infra-estrutura ideal para este tipo de operação. Mas no lado do provedor de informações? Imagine que você esteja utilizando apenas um *WebService* e o mesmo quebra a conexão? O que fazer? Com a segunda fase da tecnologia, você pode mapear ilimitados *WebServices* semelhantes para fazer o mesmo serviço, ou seja, quando cair o primeiro, passa para o segundo; na queda do segundo, para o terceiro e assim por diante, podendo até retornar ao primeiro *WebService*.

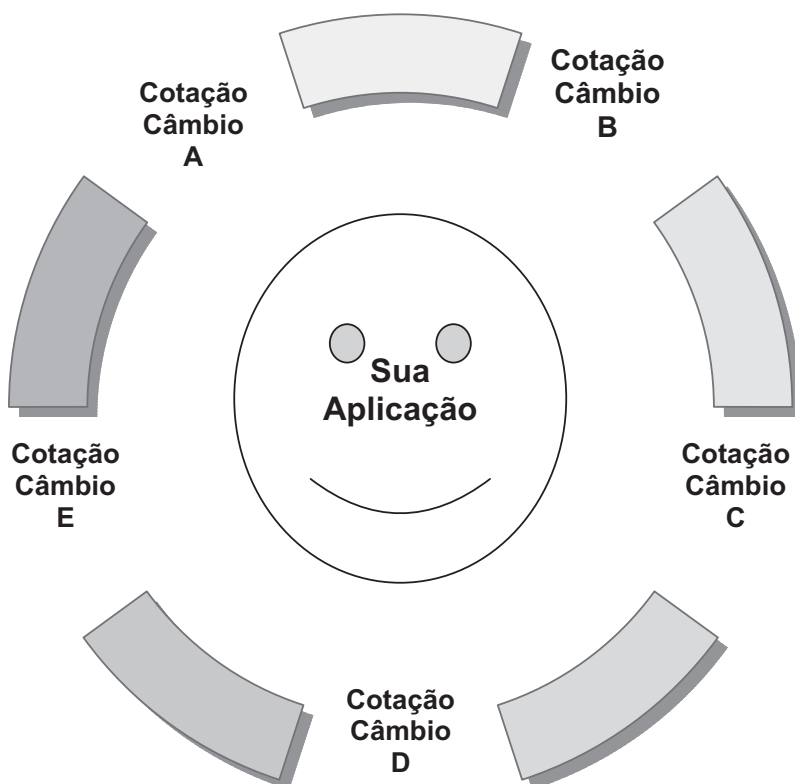


Figura 13.3 Cotação câmbio atual

Para concluir o conceito de *WebServices*, vamos imaginar um cenário mais simples, onde precisamos integrar informações de diferentes departamentos e filiais, que foram desenvolvidos em plataformas diferentes. Um bom exemplo para isso são os bancos que estão adquirindo outros bancos em todo o mundo e que utilizam conceitos e plataformas diferentes de trabalho.

A *figura 13.4* ilustra o Banco Facunte, adquirindo outros dois bancos com diferentes tecnologias.

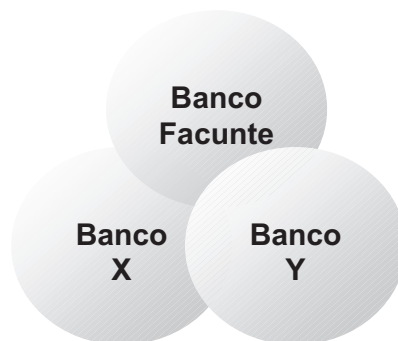


Figura 13.4 Banco com diferentes plataformas

Tecnologias do nosso cenário exemplo

Banco	S.O.	Banco Dados	Terminais
Facunte	Unix	DB2	Terminais Linux
X	NT	SQL Server	Windows
Y	Solaris	Oracle	Terminais Linux

Integrar informações de diferentes plataformas não é novidade e, como já foi mencionado, é possível, mas muito trabalhoso. A proposta da tecnologia *WebServices*, neste cenário, é o de facilitar a troca de informações, fazendo o Banco Facunte entender e tratar as informações dos Bancos X e Y, independente do banco de dados, sistema operacional ou outro fator não citado.

Com algumas linhas de programação e um bom planejamento, as informações essenciais serão interligadas facilmente. Meus amigos, isso é *WebServices*!

Em resumo, *WebServices*, é um padrão não-proprietário, que possibilita o processamento distribuído em sistemas heterogêneos. E acredito que muitos de vocês neste ponto estejam ansiosos para produzir o seu primeiro *WebService*. Antes, devemos conhecer os padrões que fazem parte da tecnologia.

Padronização

Imagine IBM, Microsoft, Borland, Oracle, Sun e outros gigantes da área de tecnologia, planejando, desenvolvendo e decidindo juntos uma mesma tecnologia. É isso que acontece com o *WebServices*. E para facilitar ainda mais o seu uso e aplicabilidade, foram adotados padrões já consagrados, além do desenvolvimento de outros padrões bastante simples. Vejamos os padrões:

XML

XML (Extensive Markup Language) ou Linguagem de Marcação Extensível, que consiste em uma série de regras, dividindo o documento em partes lógicas e hierárquicas. Atualmente é utilizada para trabalhar em conjunto com diversas tecnologias, seja no padrão de gravação de um arquivo, no transporte de informações, até mesmo em definições de montagem de veículos. Trabalhando em conjunto com o *WebServices*, fornece estrutura para o documento de regras e serviços (WSDL), pacote de dados e ocorrências.

Veja um exemplo de XML de dados:

```
<?xml version="1.0"?>
<LISTA_CLIENTES>
  <Cliente>
    <RazaoSocial>Banco Facunte</RazaoSocial>
    <Cidade>São Paulo</Cidade>
    <Estado>SP</Estado>
  </Cliente>
  <Cliente>
    <RazaoSocial>Global Education</RazaoSocial>
    <Cidade>São Paulo</Cidade>
    <Estado>SP</Estado>
  </Cliente>
  <Cliente>
    <RazaoSocial>Clube Delphi</RazaoSocial>
    <Cidade>Rio de Janeiro</Cidade>
    <Estado>RJ</Estado>
  </Cliente>
</LISTA_CLIENTES>
```

WSDL

WSDL – Web Service Definition Language ou Linguagem de Definições de WebServices. O WSDL é um documento criado no padrão XML, com o objetivo de fornecer informações sobre a definição de um *WebService*. Esta definição consiste em métodos, parâmetros e serviços fornecidos pelo *WebService*. Veremos na prática um documento WSDL, bem como a sua forma de utilização.

SOAP

Soap – Simple Object Access Protocol ou Protocolo Simples de Acesso a Objetos. Em poucas palavras, o SOAP é o protocolo utilizado para troca de informações através de objetos criados em diversas linguagens de programação, como Delphi, Java, C++, C#, VB.NET, entre outras. Trafega através da porta 80 de um servidor HTTP, facilitando assim o trabalho com servidores protegidos por *Firewall*, onde a porta 80, apesar de fortemente monitorada, permite o tráfego de informações.

UDDI

UDDI – Universal Definition Discovery Interface ou Interface para Descoberta de Definições Universais. Para facilitar a compreensão do UDDI, farei uma comparação com os serviços de busca na Internet. Quando queremos procurar algum documento, artigo, software, na Internet, normalmente utilizamos serviços de busca, como o Google (meu preferido), Yahoo!, Hotbot, Cadê?, entre outros. Imagine que para localizar um *WebServices* é bastante semelhante, com alguns parâmetros diferenciados. O UDDI na realidade é um grande catálogo de *WebServices*, oferecido por diversas empresas, seja ela envolvida na padronização, como IBM e Borland, ou até mesmo empresas independentes, como o SALCentral.COM (www.salcentral.com). Com isso poderemos localizar e mapear serviços semelhantes, como o exemplo que vimos neste capítulo, sobre a cotação de câmbios.

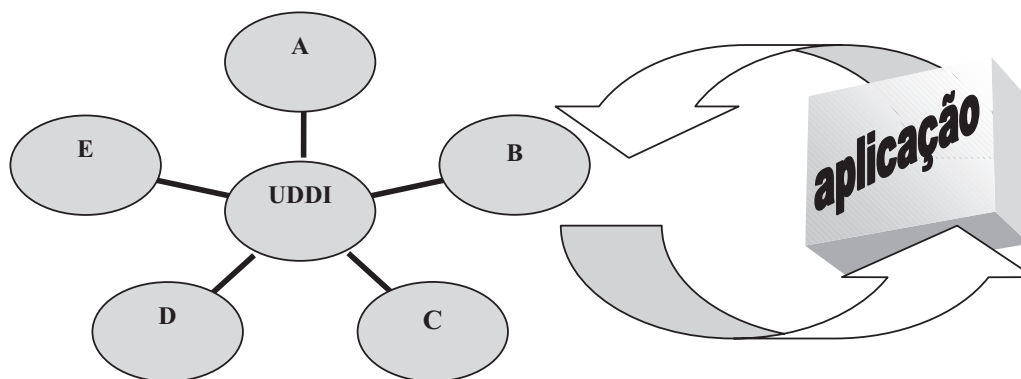


Figura 13.5 Aplicação utilizando o serviço UDDI

A figura 13.5 ilustra uma aplicação utilizando os serviços do UDDI. Neste cenário foram localizados e mapeados *WebServices* semelhantes, de maneira que a aplicação tenha objetos redundantes, aumentando a eficácia e a segurança. Isso demonstra a flexibilidade da tecnologia, onde as empresas poderão escolher os serviços (*WebServices*) mais adequados à sua aplicação.

Sugestões de desenvolvimento

CPF Restrições de Crédito

Empresas que oferecem serviços e informações sobre restrições de crédito poderão desenvolver *WebServices* para fornecer tais informações diretamente nas aplicações de seus clientes. O mais interessante é que a cobrança será automática, pois cada cliente terá uma chave criptografada e a empresa poderá cobrar por número de acessos.

Rastreamento de Cargas

Empresas como FedEx e DHL já fornecem seus próprios *WebServices*. Correios, empresas de logística e transporte, poderão desenvolver *WebServices* para rastrear as *cargas e encomendas* enviadas. Além disso, poderão fornecer informações de estimativa de chegada dinâmica, ou seja, imagine uma carga que teve um acréscimo na previsão de chegada, devido a uma greve no porto, ou até mesmo um problema no trajeto; com isso a informação poderá ser fornecida dinamicamente ao cliente, onde o mesmo tomará as devidas providências, como um adiamento de reunião ou de produção.

Identificação de Veículos

Os DETRANs de todo o país poderiam desenvolver *WebServices*, com inúmeros objetivos. O mais importante seria a identificação prévia do veículo, onde despachantes, compradores e vendedores de veículos, guardas e operadores de trânsito, saberiam imediatamente informações do veículo em questão. Através de Palmtops ou celulares, guardas de trânsito poderiam controlar com mais facilidade as multas em veículos infratores. Fabricantes de veículos poderiam checar com maior facilidade informações sobre um grupo de chassis, para um possível *recall*, preparando uma estratégia junto à sua equipe de mecânicos para um melhor atendimento por região.

Cotação de Ações

As operadoras de ações poderiam desenvolver *WebServices*, afim de fornecer informações diretamente nas aplicações de seus clientes, onde os mesmos poderiam concluir negócios mais facilmente, e armazenando a informação diretamente em seu próprio banco de dados.

Cotação de Moedas

Já existem diversos *WebServices* catalogados oferecendo o serviço de cotação de moedas. Os mais avançados oferecem funções para conversão, onde a aplicação fornece as moedas e valores e o *WebService* retorna o resultado. Existem outros que fazem atualização monetária baseada em múltiplas moedas. Isso é muito interessante em países como o nosso, que já trocou de moeda uma dezena de vezes.

TEF (Transferência Eletrônica de Fundos)

Este é um assunto muito delicado, pois envolve segurança. A tecnologia de *WebServices* está amadurecendo nesta questão, tanto que o SPB (Sistema de Pagamentos Brasileiro) está utilizando a tecnologia para o transporte de algumas informações. O pacote trafega numa rede muito segura, e altamente criptografada, evitando ações de hackers mal-intencionados. Mesmo assim é uma sugestão muito interessante, que facilita o trâmite das informações entre instituições.

Controle de Estoque

Tenho certeza que será muito bem utilizado e aplicado este tipo de *WebService*. Poderemos desenvolver *WebServices* que serão utilizados por nossos clientes, vendedores e também fornecedores. Os clientes utilizarão para fazer cotações e fechamento de pedidos através de suas próprias aplicações. Os vendedores, tanto externos como internos, utilizarão para consultar o saldo no estoque e efetuar os seus pedidos. Já os fornecedores poderão enviar orçamentos para produtos que estão chegando no nível de estoque mínimo, assim como os próprios compradores da empresa.

CEPs

Os Correios, uma das empresas mais respeitadas do país, poderiam desenvolver um *WebService*, oferecendo as informações de sua base de dados on-line, diretamente na aplicação do cliente. O objetivo não é apenas consultar e corrigir informações sobre o CEP; pode-se ampliar o serviço para o cálculo de envio de encomendas, e os prazos estipulados.


RG

Um dos assuntos polêmicos em nosso país. Cada Estado tem sua forma de estabelecer regras para os números de R.G.s. Será que um dia iremos acordar e cada cidadão brasileiro terá sua identificação própria, sem duplicidades? Isso só será possível através de uma integração entre os municípios de todo o país. Quando isso for possível, o Governo poderá oferecer *WebServices* com os mais variados objetivos. Um bom exemplo para isso, seria o Censo. Sem dúvida nenhuma teríamos um Censo com um percentual próximo do máximo e com muita rapidez.

CNAB

Quem aí já passou pelo suado processo de montagem de arquivos no “padrão” (eu disse padrão?) CNAB? Cada banco tem um formato de arquivo, um número diferente de propriedades a serem preenchidas, *headers psicodélicos* e linhas de registros com informações absurdas e repetitivas. Criando um *WebService*, *padrão (aqui sim, seria um padrão)*, os clientes poderiam transmitir e receber informações de cobrança com uma facilidade enorme. O mais interessante disso tudo acontece quando o cliente opta por outra instituição para fazer a sua cobrança e não precisa alterar nenhuma linha de programação, apenas algumas propriedades, como por exemplo, o código da instituição financeira.

Serviços de UDDI e catálogos alternativos





	<p>http://uddi.org UDDI principal, responsável pela especificação do UDDI.</p>
<p>Web Services and UDDI Universal Description, Discovery, and Integration</p>	<p>http://uddi.ibm.com UDDI da IBM, com opções de registros de <i>WebServices</i> corporativos e para testes.</p>
<p>Microsoft® uddi Business Registry Node</p>	<p>http://uddi.microsoft.com UDDI da Microsoft, com diversas opções de busca, registro simples de novos <i>WebServices</i>, ferramentas para desenvolvedores, entre outros serviços. Excelente.</p>




	http://uddi.sap.com UDDI da SAP. É necessário entrar na área UDDI Discovery para registrar, procurar e publicar <i>WebServices</i> .
	http://www.webservicelist.com/ Fantástico catálogo de <i>Webservices</i> independente, com divisão por categoria, busca de semelhantes, palavras chaves, entre outros serviços.
	http://www.xmethods.com Excelente catálogo de <i>Webservices</i> , com os mais populares, até <i>Webservices</i> comerciais.

Nos serviços aqui apresentados, existe uma imensidão de *WebServices*, divididos nas mais variadas categorias. Desde simples consultas a cotação de moedas, até serviços na área de saúde. Mais adiante iremos aprender a importar estes serviços (*WebServices*) em nossa aplicação.

Delphi x WebServices

Neste tópico iremos aprender os conceitos de *WebServices* integrado com o Delphi. O Delphi possui um conjunto de componentes específico para a tecnologia de *WebServices*. Na seção *WebServices* da paleta de componentes, encontramos os seguintes *objetos*:

COMPONENTE (OBJETO)	DESCRIÇÃO
 HTTPRIO	Utiliza para obter uma referência para uma interface registrada. Gera em memória uma tabela de métodos, a fim de fornecer a implementação da interface.
 HTTPReqResp	Executa métodos de chamada na interface, de maneira a enviar e receber mensagens no padrão SOAP. Normalmente utilizado em conjunto com o HTTPRIO, implementando os métodos Get e Post.
 OpToSOAPDOMConvert	Normalmente instanciado em tempo de execução pelo componente HTTPRIO, tem como principal função fazer um <i>parser</i> nos métodos de chamada.
 SOAPConnection	Utilizado para conectar a aplicação aos servidores implementados no padrão <i>WebService</i> .

COMPONENTE (OBJETO)	DESCRIÇÃO
 HTTPSoapDispatcher	Responsável por responder as chamadas no padrão SOAP.
 WSDLHTMLPublish	Responsável por publicar um documento WSDL com todo o descritivo do <i>WebService</i> .
 HTTPSoapPascalInvoker	Utilizado para interpretar uma mensagem padrão SOAP, e executar o método correspondente.

WebService Exemplo

Vamos criar nosso primeiro *WebService* para aprender melhor seu conceito.

Através das opções *File/New...*, seção *WebServices*, selecione a opção *SOAP Server Application* (figura 13.6).

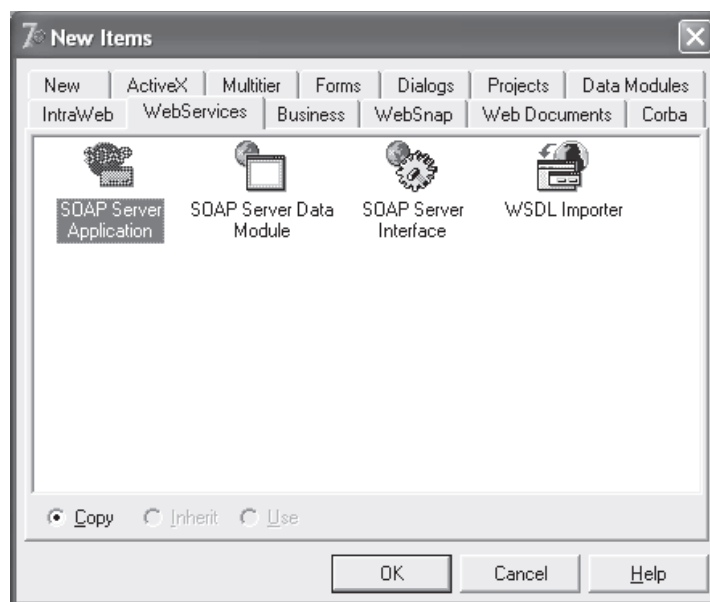


Figura 13.6 Nova aplicação *WebService*

Em seguida, selecione a opção *CGI* para o tipo da aplicação servidora *SOAP* (figura 13.7).



Figura 13.7 Tipo da aplicação servidora

Em seguida, o Delphi pergunta se deseja criar uma *Interface SOAP* padrão (figura 13.8). Em nosso primeiro exemplo, vamos criar tal *Interface*, a fim de conhecer sua implementação.



figura 13.8 Criação da Interface

Em seguida (figura 13.9), devemos informar os dados da nova *Interface*.

Nos campos *Service Name* e *Unit identifier* coloque *wsexemplo*. Com isso estamos criando uma *Interface* com o nome *wsexemplo*, e gravando a *unit* com o mesmo nome. Em *Code generation* selecione as opções *Generate Comments* e *Generate Sample Methods*. Com isso estamos gerando exemplos de métodos e comentários.



Figura 13.9 Identificação do serviço

Clique em OK para finalizar. Vamos gravar nossa aplicação.

Unit WebModule	un_ws1.pas
Unit Implementação WsExemplo	wsexemploImpl.pas
Unit Interface WsExemplo	wsexemploIntf.pas
Projeto	ws1.dpr

Vamos analisar o que o nosso amigo Delphi criou.

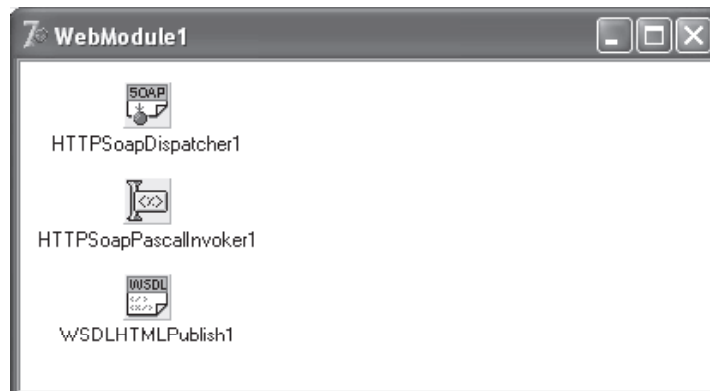


Figura 13.10 WebModule1

A *figura 13.10* ilustra nosso WebModule com três componentes no padrão WebService (HTTPSoapDispatcher, HTTPSoapPascalInvoker e WSDLHTMLPublish). Vejamos sua implementação.

```
procedure TWebModule1.WebModule1DefaultHandlerAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  WSDLHTMLPublish1.ServiceInfo(Sender, Request, Response, Handled);
end;
```

A única função do nosso WebModule1 é a criação de um handler para o WebService e publicação do documento WSDL. Na unit *wsexemploIntf* estamos definindo a Interface de nossa aplicação.

```
{Invokable interface Iwsexemplo }

unit wsexemploIntf;

interface

uses InvokeRegistry, Types, XSBuiltIns;

type

  TEnumTest = (etNone, etAFew, etSome, etAlot);

  TDoubleArray = array of Double;

  TMyEmployee = class(TRemotable)
  private
    FLastName: AnsiString;
    FFirstName: AnsiString;
    FSalary: Double;
  published
    property LastName: AnsiString read FLastName write FLastName;
    property FirstName: AnsiString read FFirstName write FFirstName;
    property Salary: Double read FSalary write FSalary;
```

```

end;

{ Invokable interfaces must derive from IInvokable }
Iwsexemplo = interface(IInvokable)
['{A391DC0F-CDA7-4929-97B8-DAECA7C2CF18}']

    { Methods of Invokable interface must not use the default }
    { calling convention; stdcall is recommended }
    function echoEnum(const Value: TEnumTest): TEnumTest; stdcall;
    function echoDoubleArray(const Value: TDoubleArray): TDoubleArray; stdcall;
    function echoMyEmployee(const Value: TMyEmployee): TMyEmployee; stdcall;
    function echoDouble(const Value: Double): Double; stdcall;
end;

implementation

initialization
    { Invokable interfaces must be registered }
    InvRegistry.RegisterInterface(TypeInfo(Iwsexemplo));

end.

```

Esta *unit* na realidade está seguindo as regras da O.O. (Orientação a Objeto), onde definimos uma *Interface* como base, para que possamos implementar nossas classes. Além disso estamos registrando a *Interface* no modelo *SOAP*. Repare que os métodos criados são apenas exemplos de implementação, que solicitamos previamente, justamente para estudar e analisar. Na *unit wsexemploImpl*, temos a implementação da *Interface* que vimos anteriormente. Veja o código.

```

{ Invokable implementation File for Twsexemplo which implements Iwsexemplo }

unit wsexemploImpl;

interface

uses InvokeRegistry, Types, XSBuiltIns, wsexemploIntf;

type

    { Twsexemplo }
    Twsexemplo = class(TInvokableClass, Iwsexemplo)
    public
        function echoEnum(const Value: TEnumTest): TEnumTest; stdcall;
        function echoDoubleArray(const Value: TDoubleArray): TDoubleArray; stdcall;
        function echoMyEmployee(const Value: TMyEmployee): TMyEmployee; stdcall;
        function echoDouble(const Value: Double): Double; stdcall;
    end;

implementation

function Twsexemplo.echoEnum(const Value: TEnumTest): TEnumTest; stdcall;
begin
    { TODO : Implement method echoEnum }
    Result := Value;
end;

function Twsexemplo.echoDoubleArray(const Value: TDoubleArray): TDoubleArray;
stdcall;
begin

```

```

{ TODO : Implement method echoDoubleArray }
Result := Value;
end;

function Twsexemplo.echoMyEmployee(const Value: TMyEmployee): TMyEmployee; stdcall;
begin
{ TODO : Implement method echoMyEmployee }
Result := TMyEmployee.Create;
end;

function Twsexemplo.echoDouble(const Value: Double): Double; stdcall;
begin
{ TODO : Implement method echoDouble }
Result := Value;
end;

initialization
{ Invokable classes must be registered }
InvRegistry.RegisterInvokableClass(Twsexemplo);

end.

```

Acredito que deu para perceber, que os métodos apenas retornam os mesmos valores informados. Para compreender melhor, vamos fazer uma pequena alteração no método **echoDouble**. Substitua a linha de retorno, pelo código que segue em negrito:

```

function Twsexemplo.echoDouble(const Value: Double): Double; stdcall;
begin
{ TODO : Implement method echoDouble }
Result := Value * 3;
end;

```

Grave a aplicação. Antes de compilar, vamos definir o diretório para geração do nosso *WebService*.

Através das opções *Project/Options.../Directories_Conditionals*, configure a opção *Output Directory*, apontando para o seu diretório cgi-bin (figura 13.11).

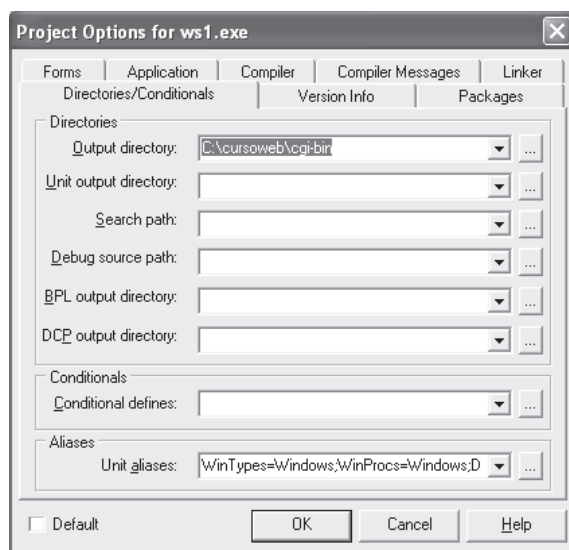


Figura 13.11 Configuração do diretório

Grave novamente a aplicação.

Agora vamos compilar a aplicação. Vamos executar a aplicação no browser para analisar o seu conteúdo. Digite: <http://localhost/cgi-bin/ws1.exe/>. A *figura 13.12* ilustra o resultado da primeira fase de nossa aplicação.

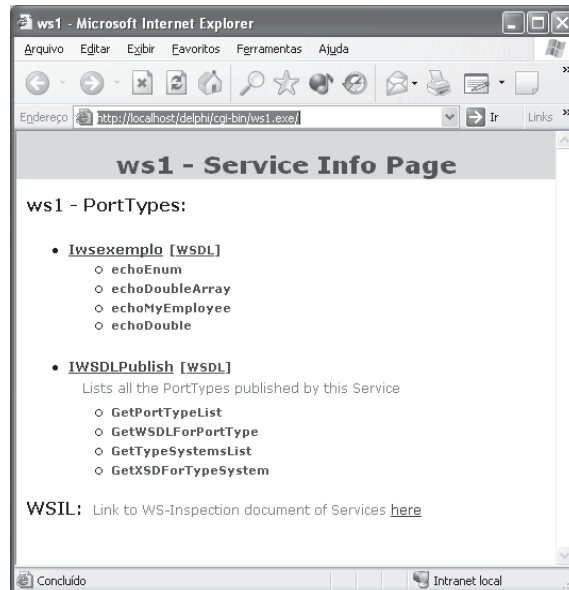


Figura 13.12 Aplicação ws1

Repare que temos todos os métodos listados na *Interface Iwsexemplo* (echoEnum, echoDoubleArray, echoMyEmployee, echoDouble). O documento gerado está de acordo com o padrão estabelecido pelo W3C-UDDI (órgão responsável pelo padrão *WebService*). Clicando no link **WSDL** da *Interface Iwsexemplo* será apresentado o seguinte documento **WSDL**.

```
<?xml version="1.0" encoding="utf-8" ?>
-      <definitions                                xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"                                name="Iwsexemploservice"
targetNamespace="http://tempuri.org/"                                xmlns:tns="http://tempuri.org/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:ns1="urn:wsexemploIntf">
- <types>
- <xs:schema targetNamespace="urn:wsexemploIntf" xmlns="urn:wsexemploIntf">
- <xs:simpleType name="TEnumTest">
- <xs:restriction base="xs:string">
- <xs:enumeration value="etNone" />
- <xs:enumeration value="etAFew" />
- <xs:enumeration value="etSome" />
- <xs:enumeration value="etAlot" />
- </xs:restriction>
- </xs:simpleType>
- <xs:complexType name="TDoubleArray">
- <xs:complexContent>
- <xs:restriction base="soapenc:Array">
- <xs:sequence />
- <xs:attribute ref="soapenc:arrayType"                                n1:arrayType="xs:double[]"
xmlns:n1="http://schemas.xmlsoap.org/wsdl/" />
- </xs:restriction>
- </xs:complexContent>
- </xs:complexType>
- <xs:complexType name="TMyEmployee">
- <xs:sequence>
```

```

<xs:element name="LastName" type="xs:string" />
<xs:element name="FirstName" type="xs:string" />
<xs:element name="Salary" type="xs:double" />
</xs:sequence>
</xs:complexType>
</xs:schema>
</types>
- <message name="echoEnum0Request">
  <part name="Value" type="ns1:TEnumTest" />
</message>
- <message name="echoEnum0Response">
  <part name="return" type="ns1:TEnumTest" />
</message>
- <message name="echoDoubleArray1Request">
  <part name="Value" type="ns1:TDoubleArray" />
</message>
- <message name="echoDoubleArray1Response">
  <part name="return" type="ns1:TDoubleArray" />
</message>
- <message name="echoMyEmployee2Request">
  <part name="Value" type="ns1:TMyEmployee" />
</message>
- <message name="echoMyEmployee2Response">
  <part name="return" type="ns1:TMyEmployee" />
</message>
- <message name="echoDouble3Request">
  <part name="Value" type="xs:double" />
</message>
- <message name="echoDouble3Response">
  <part name="return" type="xs:double" />
</message>
- <portType name="Iwsexemplo">
- <operation name="echoEnum">
  <input message="tns:echoEnum0Request" />
  <output message="tns:echoEnum0Response" />
</operation>
- <operation name="echoDoubleArray">
  <input message="tns:echoDoubleArray1Request" />
  <output message="tns:echoDoubleArray1Response" />
</operation>
- <operation name="echoMyEmployee">
  <input message="tns:echoMyEmployee2Request" />
  <output message="tns:echoMyEmployee2Response" />
</operation>
- <operation name="echoDouble">
  <input message="tns:echoDouble3Request" />
  <output message="tns:echoDouble3Response" />
</operation>
</portType>
- <binding name="Iwsexemplobinding" type="tns:Iwsexemplo">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
- <operation name="echoEnum">
  <soap:operation soapAction="urn:wsexemploIntf-Iwsexemplo#echoEnum" style="rpc" />
- <input message="tns:echoEnum0Request">
  <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="urn:wsexemploIntf-Iwsexemplo" />
  </input>
- <output message="tns:echoEnum0Response">

```

```

    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="urn:wsexemploIntf-Iwsexemplo" />
  </output>
</operation>
- <operation name="echoDoubleArray">
  <soap:operation
    soapAction="urn:wsexemploIntf-Iwsexemplo#echoDoubleArray"
    style="rpc" />
  <input message="tns:echoDoubleArray1Request">
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="urn:wsexemploIntf-Iwsexemplo" />
  </input>
  <output message="tns:echoDoubleArray1Response">
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="urn:wsexemploIntf-Iwsexemplo" />
  </output>
</operation>
- <operation name="echoMyEmployee">
  <soap:operation
    soapAction="urn:wsexemploIntf-Iwsexemplo#echoMyEmployee"
    style="rpc" />
  <input message="tns:echoMyEmployee2Request">
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="urn:wsexemploIntf-Iwsexemplo" />
  </input>
  <output message="tns:echoMyEmployee2Response">
    <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="urn:wsexemploIntf-Iwsexemplo" />
  </output>
</operation>
- <operation name="echoDouble">
  <soap:operation soapAction="urn:wsexemploIntf-Iwsexemplo#echoDouble" style="rpc" />
- <input message="tns:echoDouble3Request">
  <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="urn:wsexemploIntf-Iwsexemplo" />
</input>
- <output message="tns:echoDouble3Response">
  <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="urn:wsexemploIntf-Iwsexemplo" />
</output>
</operation>
</binding>
- <service name="Iwsexemploservice">
- <port name="IwsexemploPort" binding="tns:Iwsexemplobinding">
  <soap:address location="http://localhost/delphi/cgi-bin/ws1.exe/soap/Iwsexemplo" />
</port>
</service>
</definitions>

```

Parece complicado, não é?

Vamos analisar algumas partes do documento para compreender melhor seu funcionamento.

```

- <types>
- <xs:schema targetNamespace="urn:wsexemploIntf" xmlns="urn:wsexemploIntf">
- <xs:simpleType name="TEnumTest">
- <xs:restriction base="xs:string">
  <xs:enumeration value="etNone" />
  <xs:enumeration value="etAFew" />
  <xs:enumeration value="etSome" />

```



```
<xs:enumeration value="etAlot" />
</xs:restriction>
```

Passando pelo bloco das definições iniciais (`<?xml version="1.0" ...`), chegamos no bloco de definições de tipos e métodos. Neste bloco são definidos todos os métodos da nossa *Interface*, com o descritivo completo. Repare que a classe *TEnumTest* é descrita com perfeição.

No bloco que segue, são descritos os métodos *Request* e *Response* da *Interface* em questão.

```
- <message name="echoEnum0Request">
  <part name="Value" type="ns1:TEnumTest" />
</message>
- <message name="echoEnum0Response">
  <part name="return" type="ns1:TEnumTest" />
</message>
```

A seguir, temos o bloco que define o nome da porta (*Port*) e as operações *Request* e *Response*, descritas no bloco anterior.

```
- <portType name="Iwsexemplo">
- <operation name="echoEnum">
  <input message="tns:echoEnum0Request" />
  <output message="tns:echoEnum0Response" />
</operation>
```

Em seguida temos o bloco que “*envelopa*” e define a camada de transporte dos métodos.

```
- <binding name="Iwsexemplobinding" type="tns:Iwsexemplo">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
- <operation name="echoEnum">
  <soap:operation soapAction="urn:wsexemploIntf-Iwsexemplo#echoEnum" style="rpc" />
- <input message="tns:echoEnum0Request">
  <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="urn:wsexemploIntf-Iwsexemplo" />
  </input>
- <output message="tns:echoEnum0Response">
  <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  namespace="urn:wsexemploIntf-Iwsexemplo" />
  </output>
```

O último bloco finaliza o documento, declarando o nome do serviço, bem como o nome da porta e a sua camada de transporte.

```
- <service name="Iwsexemploservice">
- <port name="IwsexemploPort" binding="tns:Iwsexemplobinding">
  <soap:address location="http://localhost/delphi/cgi-bin/ws1.exe/soap/Iwsexemplo" />
</port>
</service>
</definitions>
```

Agora vamos criar uma aplicação cliente para testar nosso primeiro *WebService*. Através das opções *File/New Application* crie uma nova aplicação, e grave os arquivos como segue:

Unit	un_teste_ws1.PAS
Projeto	teste_ws1.DPR

Agora vamos importar a *Interface* em nossa aplicação. Através das opções *File/New.../WebServices* (figura 13.13), selecione a opção *WSDL Importer*.

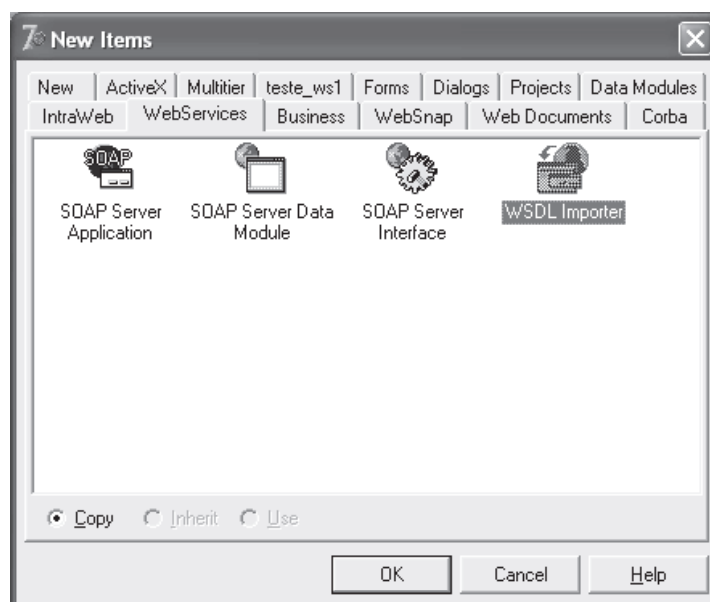


Figura 13.13 WSDL importer

Em seguida, como ilustra a *figura 13.14*, digite o endereço que segue.

```
http://localhost/cgi-bin/ws1.exe/wsdl/Iwsexemplo
```

Este endereço faz a chamada ao *documento WSDL* da *Interface Iwsexemplo*.

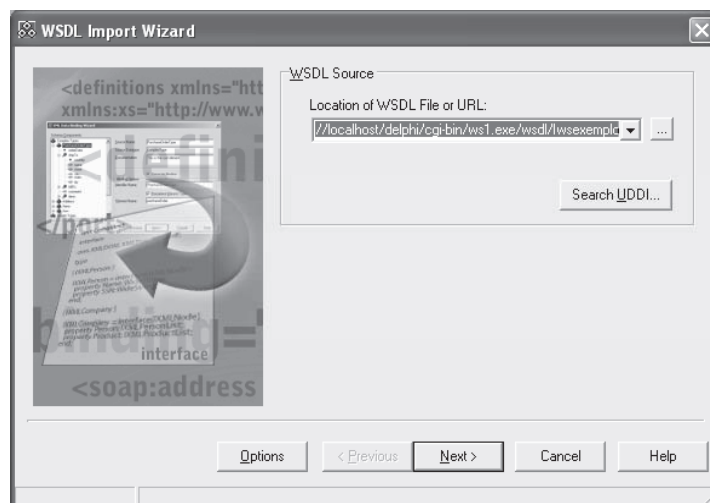


Figura 13.14 Importação da Interface através do WSDL

Este procedimento está importando toda a *Interface* para a nossa aplicação. Com isso teremos acesso aos métodos definidos. É interessante destacar que um *WebService* poderá ter inúmeras *Interfaces*, onde poderemos importar apenas as que condizem com a nossa necessidade. Aperte o botão *Next* para avançar à próxima fase. A *figura 13.15* ilustra a *Interface* gerada pelo assistente.

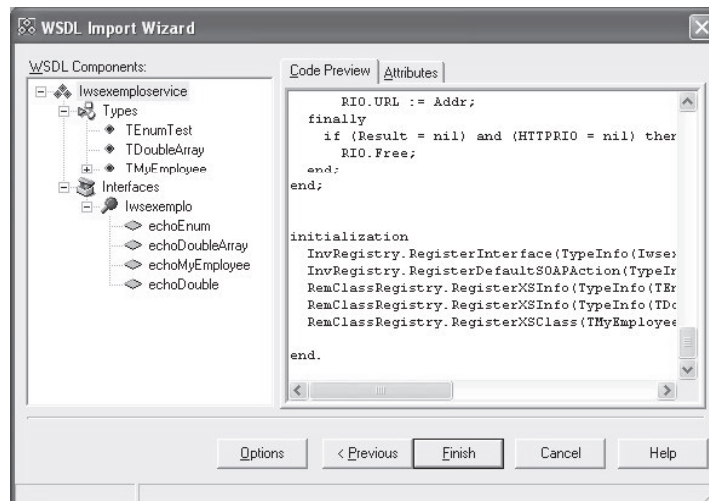



Figura 13.15 Interface gerada pelo assistente

Para concluir, aperte a tecla *Finish*. O assistente gerou uma *Unit* com toda a *Interface* implementada. Grave a *Unit* com o nome *Iwsexemplo1.pas*.

O que acabamos de fazer, na realidade, foi a importação de uma *Interface* para facilitar o uso do *WebService*. Agora com o foco na *unit un_teste_ws1*, insira a *unit Iwsexemplo1.pas* gerada pelo assistente.


```
implementation
uses Iwsexemplo1;
```


Neste ponto iremos configurar o acesso para este formulário. Insira um objeto do tipo *THTTPRio* e configure as propriedades que seguem, respeitando a sequência apresentada, caso contrário, uma exceção ocorrerá.


OBJETO		
		
THTTPRio		
Objeto	Propriedade	Valor
HTTPRio1	Name	HR1
	WSDLLocation	http://localhost/cgi-bin/ws1.exe/wsdl/Iwsexemplo
	Service	Iwsexemploservice
	Port	IwsexemploPort

Com isto configuramos o objeto de acesso à *Interface*, informando a localização do documento *WSDL* (*WSDLLocation*), o serviço (*Service*), e a porta (*Port*).

Agora insira os objetos que seguem, configurando suas respectivas propriedades.

OBJETO		
	TEdit	
Objeto	Propriedade	Valor
edValor	Name	edValor
	Left	32
	Text	deixe em branco
	Top	24
	Width	120

OBJETO		
	TButton	
Objeto	Propriedade	Valor
Button1	Name	Button1
	Caption	Calcula
	Left	176
	Top	24
	Width	125

OBJETO		
	Tlabel	
Objeto	Propriedade	Valor
IbResultado	Name	IbResultado
	Caption	0
	Left	32
	Top	64

A figura 13.16 ilustra o formulário da nossa aplicação.

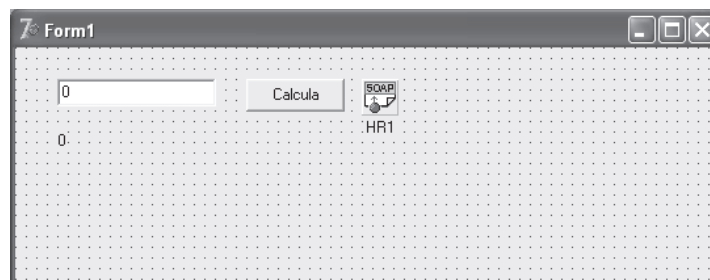


Figura 13.16 Formulário da aplicação

Agora vamos codificar a aplicação. Insira o código que segue no evento *OnClick* do objeto *Button1*.

```
var
    Iexemplo: Iwsexemplo;
begin
```

```

Iexemplo:= HR1 as Iwsexemplo;
lbResultado.Caption:=FloatToStr(Iexemplo.echoDouble( StrtoFloat(edValor.Text)));
end;

```

O código é bastante simples, onde estamos definindo um objeto do tipo *Iwsexemplo*. Em seguida estamos instanciando o objeto a partir do nosso *HTTPRIO* (*HR1*), adotando o modelo *Iwsexemplo*. E por fim, apresentamos o resultado em nosso objeto *lbResultado*, através da função **IExemplo.echoDouble**.

O mais importante, até aqui, é justamente a compreensão de como conseguimos implementar o *WebService* em nossa aplicação.

Vamos testar nossa aplicação. Compile e execute a aplicação, informando um número no campo e pressionando o botão. A *figura 13.17* ilustra nossa aplicação em tempo de execução.

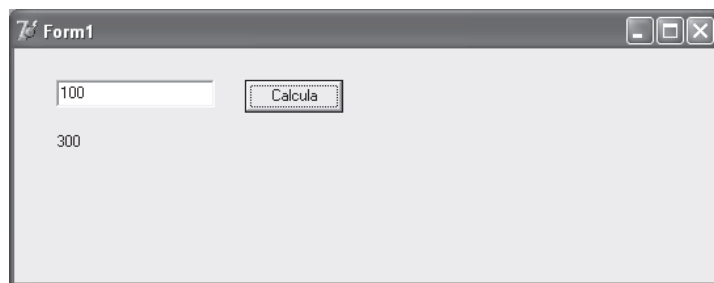


Figura 13.17 Aplicação em tempo de execução

Perceba que, na primeira vez que pressionamos o botão, existe um *delay*, que é justamente o tempo de conexão com o *WebService*.

Repita a operação e perceba que já não existe mais o *delay*. Amigos, no próximo tópico iremos desenvolver nosso próprio *WebService*.

Listagem 13.1 *un_teste_ws1.pas (formulário principal)*

```

unit un_teste_ws1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, InvokeRegistry, Rio, SOAPHTTPClient, StdCtrls;

type
  TForm1 = class(TForm)
    HR1: THTTPIO;
    edValor: TEdit;
    lbresultado: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

```

```

implementation
  uses Iwsexemplo1;
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  Iexemplo: Iwsexemplo;
begin
  Iexemplo:= HR1 as Iwsexemplo;
  lbresultado.Caption:=FloatToStr(Iexemplo.echoDouble(StrtoFloat(edValor.Text)));
end;

end.

```

Listagem 13.2 Iwsexemplo1.pas (interface importada para a aplicação)

```

// *****
// The types declared in this file were generated from data read from the
// WSDL File described below:
// WSDL      : http://localhost/delphi/cgi-bin/ws1.exe/wsdl/Iwsexemplo
// Encoding  : utf-8
// Version   : 1.0
// (29/08/2002 12:08:27 - 1.33.2.5)
// *****

unit Iwsexemplo1;

interface

uses InvokeRegistry, SOAPHTTPClient, Types, XSBuiltIns;

type

  // *****
  // The following types, referred to in the WSDL document are not being represented
  // in this file. They are either aliases[@] of other types represented or were
  // referred
  // to but never[!] declared in the document. The types from the latter category
  // typically map to predefined/known XML or Borland types; however, they could also
  // indicate incorrect WSDL documents that failed to declare or import a schema
  type.
  // *****
  // !:double      - "http://www.w3.org/2001/XMLSchema"
  // !:string      - "http://www.w3.org/2001/XMLSchema"

  TMyEmployee      = class;                                { "urn:wsexemploIntf" }

  { "urn:wsexemploIntf" }
  TEnumTest = (etNone, etAFew, etSome, etAlot);

  TDoubleArray = array of Double;                          { "urn:wsexemploIntf" }

  // *****
  // Namespace : urn:wsexemploIntf
  // *****
  TMyEmployee = class(TRemotable)

```

```

private
    FLastName: WideString;
    FFirstName: WideString;
    FSalary: Double;
published
    property LastName: WideString read FLastName write FLastName;
    property FirstName: WideString read FFirstName write FFirstName;
    property Salary: Double read FSalary write FSalary;
end;

// ***** //
// Namespace : urn:wsexemploIntf-Iwsexemplo
// soapAction: urn:wsexemploIntf-Iwsexemplo#%operationName%
// transport : http://schemas.xmlsoap.org/soap/http
// style      : rpc
// binding    : Iwsexemplobinding
// service    : Iwsexemploservice
// port       : IwsexemploPort
// URL        : http://localhost/delphi/cgi-bin/ws1.exe/soap/Iwsexemplo
// ***** //
Iwsexemplo = interface(IInvokable)
['{A21A137B-E1B0-488A-810D-790FBDEA675A}']
    function echoEnum(const Value: TEnumTest): TEnumTest; stdcall;
    function echoDoubleArray(const Value: TDoubleArray): TDoubleArray; stdcall;
    function echoMyEmployee(const Value: TMyEmployee): TMyEmployee; stdcall;
    function echoDouble(const Value: Double): Double; stdcall;
end;

function GetIwsexemplo(UseWSDL: Boolean=System.False; Addr: string=''; HTTPRIO:
THHTTPRIO = nil): Iwsexemplo;

implementation

function GetIwsexemplo(UseWSDL: Boolean; Addr: string; HTTPRIO: THHTTPRIO):
Iwsexemplo;
const
    defWSDL = 'http://localhost/delphi/cgi-bin/ws1.exe/wsdl/Iwsexemplo';
    defURL  = 'http://localhost/delphi/cgi-bin/ws1.exe/soap/Iwsexemplo';
    defSvc  = 'Iwsexemploservice';
    defPrt  = 'IwsexemploPort';
var
    RIO: THHTTPRIO;
begin
    Result := nil;
    if (Addr = '') then
    begin
        if UseWSDL then
            Addr := defWSDL
        else
            Addr := defURL;
    end;
    if HTTPRIO = nil then
        RIO := THHTTPRIO.Create(nil)
    else
        RIO := HTTPRIO;
    try

```

```

    Result := (RIO as Iwsexemplo);
    if UseWSDL then
    begin
        RIO.WSDLLocation := Addr;
        RIO.Service := defSvc;
        RIO.Port := defPrt;
    end else
        RIO.URL := Addr;
    finally
        if (Result = nil) and (HTTPRIO = nil) then
            RIO.Free;
        end;
    end;
end;

initialization
    InvRegistry.RegisterInterface(TypeInfo(Iwsexemplo), 'urn:wsexemploIntf-Iwsexemplo',
    'utf-8');
    InvRegistry.RegisterDefaultSOAPAction(TypeInfo(Iwsexemplo), 'urn:wsexemploIntf-
    Iwsexemplo#%operationName%');
    RemClassRegistry.RegisterXSInfo(TypeInfo(TEnumTest), 'urn:wsexemploIntf',
    'TEnumTest');
    RemClassRegistry.RegisterXSInfo(TypeInfo(TDoubleArray), 'urn:wsexemploIntf',
    'TDoubleArray');
    RemClassRegistry.RegisterXSClass(TMyEmployee, 'urn:wsexemploIntf', 'TMyEmployee');
end.

```

Segundo WebService

Neste exemplo vamos criar nosso *WebService* sem utilizar os exemplos de métodos gerados pelo assistente.

Nosso *WebService* consiste em fazer um simples cálculo, apresentando um número aproximado de dias já vividos, a partir de uma idade informada. Exemplo: 28 anos = 10.220 dias aproximadamente (28 x 365), onde 28 = idade informada, 365 = dias do ano. Através das opções *File/New...*, seção *WebServices*, selecione a opção *SOAP Server Application* (figura 13.18).

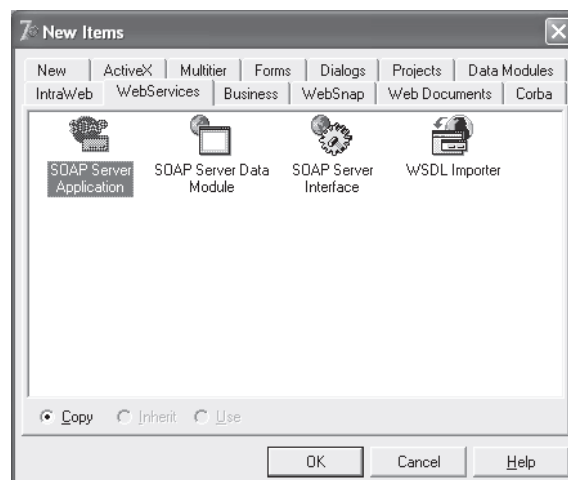


Figura 13.18 Nova aplicação *WebService*

Em seguida, selecione a opção *CGI* para o tipo da aplicação servidora *SOAP* (figura 13.19).



Figura 13.19 Tipo da aplicação servidora

Em seguida, o Delphi pergunta se deseja criar uma *Interface SOAP* padrão (figura 13.20). Neste caso, seleciona *No*, pois iremos criar “manualmente”.



Figura 13.20 Criação da Interface

Vamos gravar nossa aplicação.

Unit WebModule	un_ws2.PAS
Projeto	ws2.DPR

Agora vamos criar a *Interface* de nossa aplicação. A partir da versão 7, existe o assistente para a criação da *Interface*. Na realidade faz a mesma operação que dispensamos no diálogo anterior (figura 13.20).

Através das opções *File/New..WebServices*, selecione o assistente *SOAP Server Interface* (figura 13.21), e em seguida pressione **OK**.

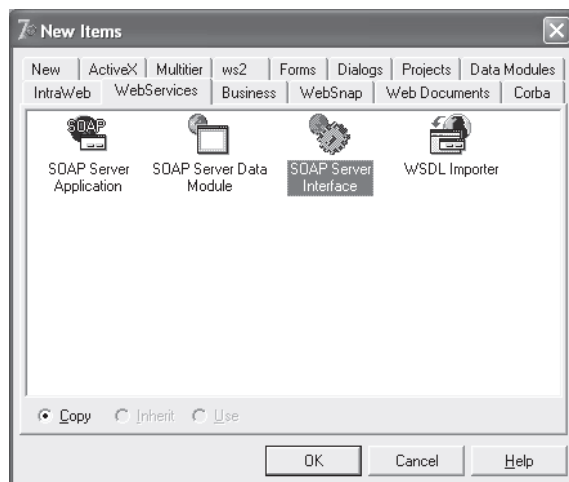


Figura 13.21 Criando a Interface do Webservice

Em seguida (figura 13.22) devemos informar os dados da nova Interface.

Nos campos *Service Name* e *Unit identifier* coloque *wsIdade*. Com isso estamos criando uma Interface com o nome *wsIdade*, e gravando a *unit* com o mesmo nome.

Em *Code generation* desmarque as duas opções (*Generate Comments* e *Generate Sample Methods*).

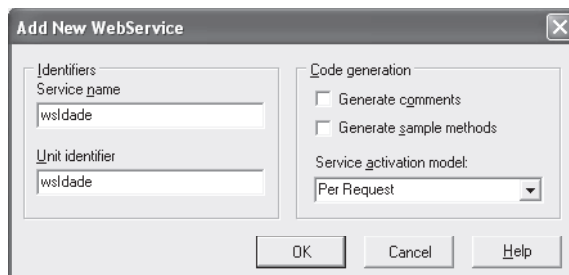


Figura 13.22 Dados complementares da Interface

O assistente criou duas novas *units*: *wsIdadeIntf.pas* e *wsIdadeImpl.pas*.

Vamos gravar nossa aplicação.

Unit wsIdadeIntf	<i>wsIdadeIntf.pas</i>
Unit wsIdadeImpl	<i>wsIdadeImpl.pas</i>

Vamos analisar as *units* criadas.

wsIdadeIntf.pas

```
unit wsIdadeIntf;

interface

uses InvokeRegistry, Types, XSBuiltIns;

type

  IwsIdade = interface(IInvokable)
```

```
[ '{A808C7C4-86BC-446C-B329-60F22BD82A87}' ]  
end;  
  
implementation  
  
initialization  
  InvRegistry.RegisterInterface(TypeInfo(IwsIdade));  
  
end.
```

Esta é a *unit* para definição da *Interface* do nosso *WebService*. No bloco que segue, definimos os métodos. Repare que foi criado um identificador único (GUID). Poderíamos fazer esta *Interface* sem o uso do assistente, e para a criação do GUID, basta pressionar as teclas *CTRL-G* no editor.

```
type  
  
  IwsIdade = interface(IInvokable)  
    [ '{A808C7C4-86BC-446C-B329-60F22BD82A87}' ]  
  end;
```

A seguir temos o registro da *Interface* do nosso *WebService*..

```
implementation  
  
initialization  
  InvRegistry.RegisterInterface(TypeInfo(IwsIdade));
```

Agora veremos a *unit wsIdadeImpl*.

wsIdadeImpl.pas

```
unit wsIdadeImpl;  
  
interface  
  
uses InvokeRegistry, Types, XSBuiltIns, wsIdadeIntf;  
  
type  
  
  TwSIdade = class(TInvokableClass, IwsIdade)  
    public  
    end;  
  
implementation  
  
initialization  
  InvRegistry.RegisterInvokableClass(TwsIdade);  
  
end.
```

Esta é a *unit* para implementação dos métodos definidos na *Interface* (*unit wsIdadeIntf*). Repare no bloco a seguir, que esta *unit* faz uso da anterior.

```
uses InvokeRegistry, Types, XSBuiltIns, wsIdadeIntf;
```

No bloco que segue, os métodos são definidos como *public*.

```
type
```

```
TwsIdade = class(TInvokableClass, IwsIdade)
public
end;
```

Bem, agora vamos definir o método de nosso *WebService*. Selecione a *unit wsIdadeIntf*, e insira o método *QuantosDias*, como segue.

```
IwsIdade = interface(IInvokable)
['{A808C7C4-86BC-446C-B329-60F22BD82A87}']

function QuantosDias(idade: Integer):integer; stdcall;

end;
```

Repare que estamos fazendo uma chamada explícita *stdcall*, padrão em *WebServices*. Grave a *unit*. Agora, selecione a *unit wsIdadeImpl* e insira o método *QuantosDias*, como segue;

```
public
function QuantosDias(idade: Integer):integer; stdcall;
end;
```

Na seção *implementation*, digite o código a seguir.

```
function TwsIdade.QuantosDias(idade: integer):integer;
begin
    Result:=idade * 365;
end;
```

Está pronto nosso *WebService*.

Agora vamos desenvolver dois clientes com tecnologias diferentes. O primeiro será no padrão *desktop*, e o outro uma aplicação *CGI*. Antes de compilar, vamos definir o diretório para geração do nosso *WebService*. Através das opções *Project/Options.../Directories_Conditionals*, configure a opção *Output Directory*, apontando para o seu diretório *cgi-bin* (figura 13.23).

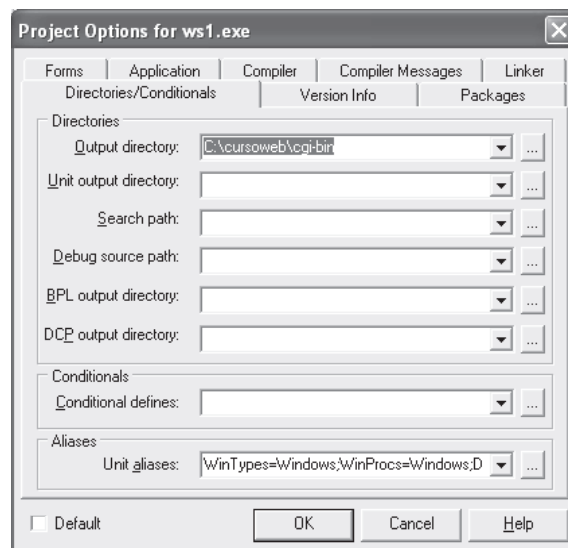


Figura 13.23 Configuração do diretório

Compile a nossa aplicação para que possamos analisar o documento **WSDL**. No browser digite **http://localhost/cgi-bin/ws2.exe**. A figura 13.24 ilustra o assistente WSDL de nossa aplicação.

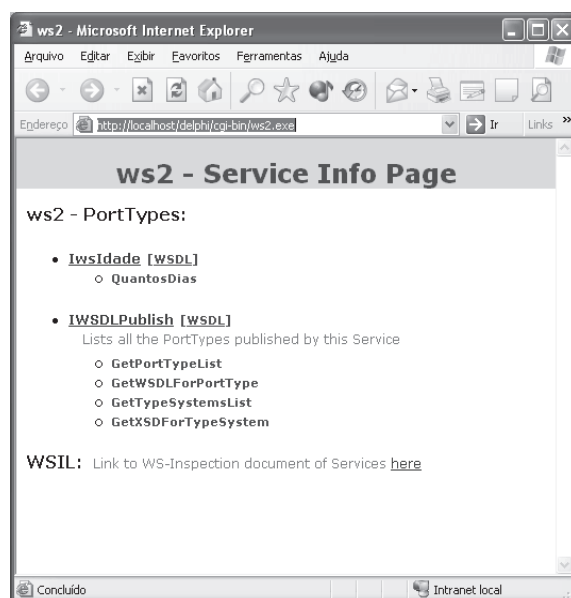


Figura 13.24 Página de informação (assistente WSDL).

Repare que o nosso método **QuantosDias** está publicado. Clique no **WSDL** do método para visualizar a implementação. O seguinte documento **WSDL** foi gerado.

```
<?xml version="1.0" encoding="utf-8" ?>
-      <definitions                                xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"                                name="IwsIdadeservice"
targetNamespace="http://tempuri.org/"                                xmlns:tns="http://tempuri.org/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
- <message name="QuantosDias0Request">
  <part name="idade" type="xs:int" />
</message>
- <message name="QuantosDias0Response">
  <part name="return" type="xs:int" />
</message>
- <portType name="IwsIdade">
- <operation name="QuantosDias">
  <input message="tns:QuantosDias0Request" />
  <output message="tns:QuantosDias0Response" />
</operation>
</portType>
- <binding name="IwsIdadebinding" type="tns:IwsIdade">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
- <operation name="QuantosDias">
  <soap:operation soapAction="urn:wsIdadeIntf-IwsIdade#QuantosDias" style="rpc" />
- <input message="tns:QuantosDias0Request">
  <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:wsIdadeIntf-IwsIdade" />
  </input>
- <output message="tns:QuantosDias0Response">
  <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:wsIdadeIntf-IwsIdade" />
  </output>
</operation>
```

```

</binding>
- <service name="IwsIdadeservice">
- <port name="IwsIdadePort" binding="tns:IwsIdadebinding">
  <soap:address location="http://localhost/delphi/cgi-bin/ws2.exe/soap/IwsIdade" />
</port>
</service>
</definitions>

```

Agora vamos criar a primeira aplicação cliente para o nosso *WebService*. Através das opções *File/New Application* cria uma nova aplicação e grave os arquivos como segue:

Unit	un_teste_ws2.PAS
Projeto	teste_ws2.DPR

Agora vamos importar a *Interface* em nossa aplicação. Através das opções *File/New.../WebServices* (figura 13.25), selecione a opção *WSDL Importer*.

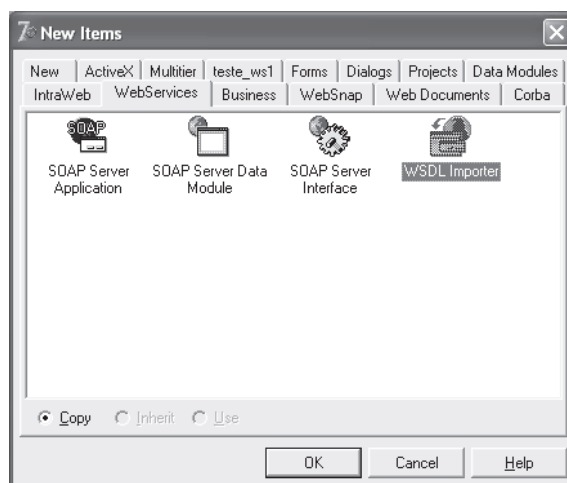


Figura 13.25 WSDL importer

Em seguida, como ilustra a figura 13.26, digite o endereço que segue.

```
http://localhost/cgi-bin/ws2.exe/wsdl/IwsIdade
```

Este endereço faz a chamada ao documento *WSDL* da *Interface IwsIdade*.

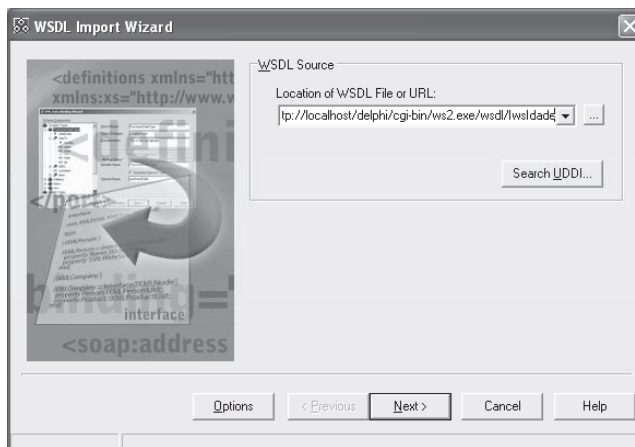



Figura 13.26 Importando a Interface


O assistente gerou uma *Unit* com toda a *Interface* implementada. Grave a *Unit* com o nome *IwsIdade1.pas*. Agora com o foco na *unit un_teste_ws2*, insira a *unit IwsIdade1.pas* gerada pelo assistente.


```
implementation
uses IwsIdade1;
```


Neste ponto iremos configurar o acesso para este formulário. Insira um objeto do tipo *THTTPrIo* e configure as propriedades que seguem, respeitando a seqüência apresentada, caso contrário, uma exceção ocorrerá.

OBJETO		
	THTTPrIo	
Objeto	Propriedade	Valor
HTTPrIo1	Name	HR1
	WSDLLocation	http://localhost/cgi-bin/ws2.exe/wsd/IwsIdade
	Service	IwsIdadeservice
	Port	IwsIdadePort

Com isto configuramos o objeto de acesso à *Interface*, informando a localização do documento *WSDL* (*WSDLLocation*), o serviço (*Service*), e a porta (*Port*). Agora insira os objetos que seguem, configurando suas respectivas propriedades.

OBJETO		
	TEdit	
Objeto	Propriedade	Valor
edValor	Name	edIdade
	Left	32
	Text	deixe em branco
	Top	24
	Width	120

OBJETO		
	TButton	
Objeto	Propriedade	Valor
Button1	Name	Button1
	Caption	Calcula
	Left	176
	Top	24
	Width	125

OBJETO		
	Tlabel	
Objeto	Propriedade	Valor
lbResultado	Name	lbDias
	Caption	Dias
	Left	32
	Top	64

A figura 13.27 ilustra o formulário da nossa aplicação.

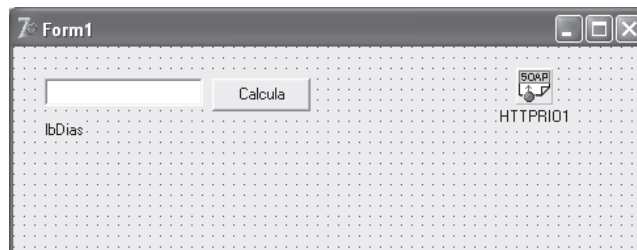


Figura 13.27 Formulário da aplicação

Agora vamos codificar a aplicação. Insira o código que segue no evento *OnClick* do objeto *Button1*.

```
var
  IIidade: IwsIdade;
begin
  IIidade:=HR1 as IwsIdade;
  lbDias.Caption:='Você já viveu aproximadamente
'+IntToStr(IIidade.QuantosDias(StrToInt(edIdade.Text)))+ ' dias';
end;
```

Muito parecido com o nosso primeiro teste, estamos definindo um objeto do tipo *IwsIdade*, em seguida estamos instanciando o objeto a partir do nosso *HTTPrIo* (*HR1*), adotando o modelo *IwsIdade*. E por fim, apresentamos o resultado em nosso objeto *lbDias*, através da função *IIidade.QuantosDias*.

Fácil? Acredito que neste ponto já deu para compreender o método de implementação de um *WebService*. Agora vamos utilizar o mesmo *WebService* numa aplicação do tipo *CGI*. A partir do Delphi, selecione as opções *File/New/Other...* e em seguida a opção *Web Server Application*, como ilustra a figura 13.28.

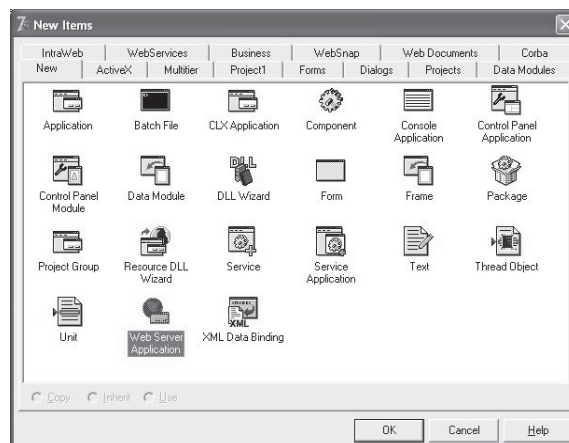


Figura 13.28 Opção *Web Server Application*

Na janela seguinte selecione a opção *CGI Stand-Alone executable* (figura 13.29).

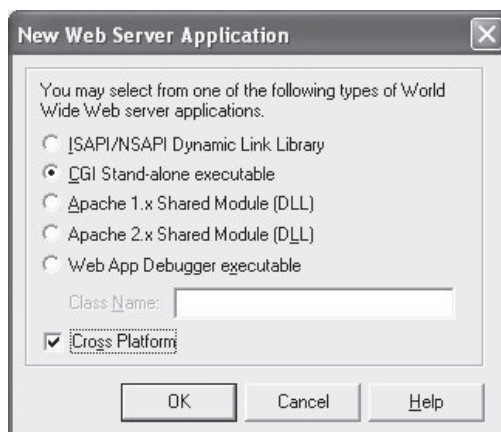


Figura 13.29 Seleção do tipo da aplicação

Em seguida teremos o nosso *WebModule* (figura 13.30).

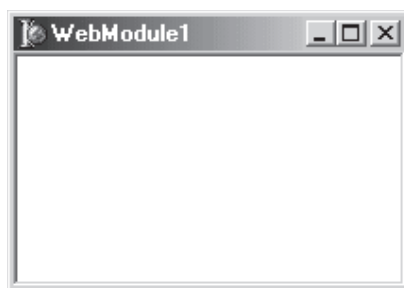



Figura 13.30 WebModule

Neste ponto ao invés de importar o *WSDL*, vamos apenas adicionar a *unit* que já importamos em nossa primeira aplicação: *IwsIdade*;

```
implementation
uses IwsIdade1;
```

Neste ponto iremos configurar o acesso para o nosso *CGI*. Insira um objeto do tipo *THTTPRio* e configure as propriedades que seguem, respeitando a seqüência apresentada, caso contrário, uma exceção ocorrerá.

OBJETO		
		
THTTPRio		
Objeto	Propriedade	Valor
HTTTPRio1	Name	HR1
	WSDLLocation	http://localhost/cgi-bin/ws2.exe/wsdllwsIdade
	Service	IwsIdadeservice
	Port	IwsIdadePort

Agora vamos criar nossa *Action*. Através do duplo-clique no *WebModule*, acesse o editor de *ActionItems* (figura 13.31).

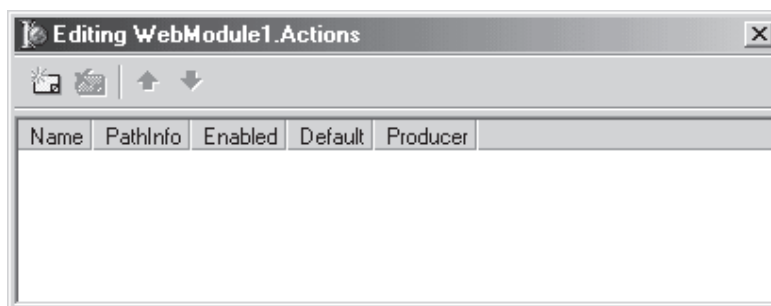


Figura 13.31 Editor *ActionItems*

Clique no primeiro botão do editor para inserir uma nova *Action* (figura 13.32).

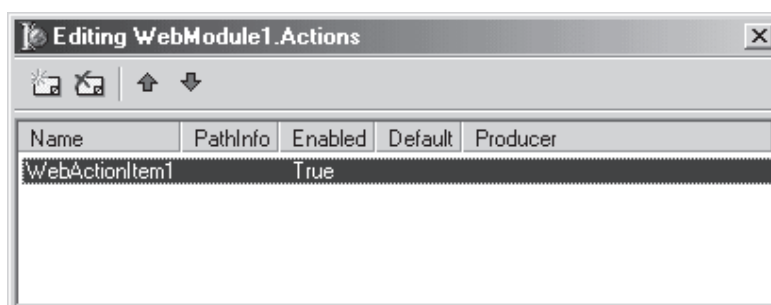



Figura 13.32 *ActionItem*

Em seguida altere as seguintes propriedades.

OBJETO		
		
TWebActionItem		
Objeto	Propriedade	Valor
padrao	Default	True
	Name	dias
	PathInfo	/dias

Esta será nossa *Action padrão*, ou seja, caso o usuário não digite nada, além do nome da nossa aplicação, esta *Action* será executada. No evento *OnAction* coloque o seguinte código:

```
var
  IIidade: IwsIdade;
begin
  IIidade:=HR1 as IwsIdade;
  Response.Content:='Você já viveu aproximadamente
'+InttoStr(IIidade.QuantosDias(StrtoInt(Request.QueryFields.Values['idade'])))+'
dias';
end;
```

Aqui estamos utilizando o método *Response.Content* para apresentar ao usuário a mensagem criada através do parâmetro *Idade*, utilizando o método *QuantosDias* do nosso *WebService*.

Antes de compilar vamos definir o diretório para geração do nosso *WebService*. Através das opções *Project/Options.../Directories_Conditionals*, configure a opção *Output Directory*, apontando para o seu diretório *cgi-bin* (figura 13.33).

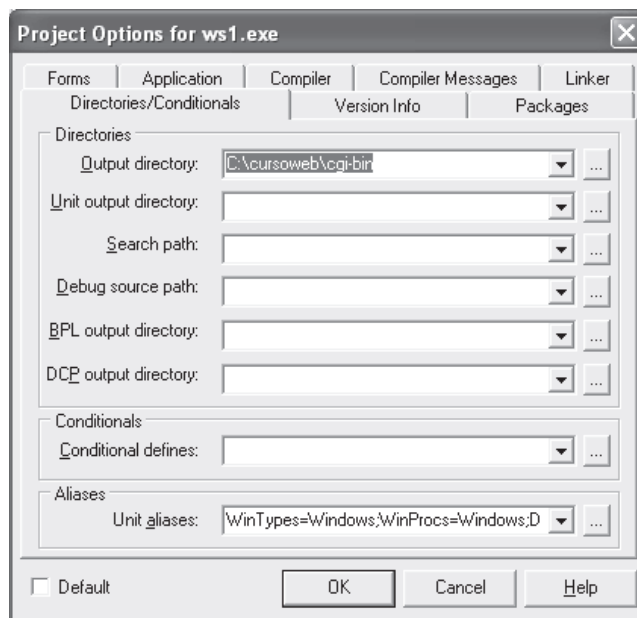


Figura 13.33 Configuração do diretório

Grave os arquivos como segue.

Unit	un_cgi_ws2.PAS
Projeto	teste_cgi_ws2.DPR

Agora vamos testar a nossa aplicação. No browser digite o seguinte endereço:

`http://localhost/delphi/cgi-bin/teste_cgi_ws2.exe/dias?idade=55`

A figura 13.34 ilustra nosso CGI em tempo de execução.

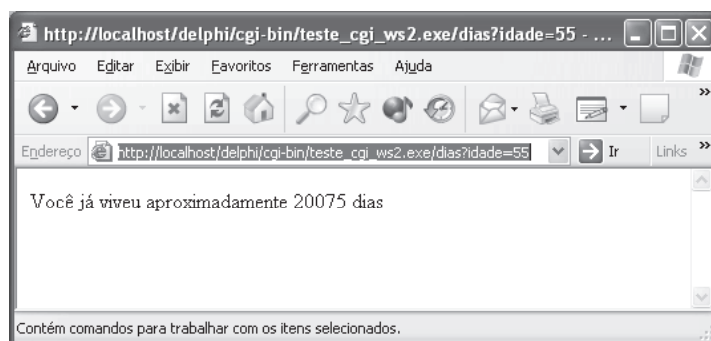


Figura 13.34 CGI em execução

Para testar o CGI, estamos passando através do parâmetro *Idade* (*?idade=*), a idade desejada para o cálculo. Nosso CGI extrai através do método *Request.QueryFields* o valor do parâmetro *Idade*, e faz o cálculo utilizando o método *QuantosDias* do nosso *Webservice*. Com isso concluímos o nosso projeto.

Listagem 13.3 un_teste_ws2.pas (teste modelo desktop)

```

unit un_teste_ws2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, InvokeRegistry, Rio, SOAPHTTPClient, StdCtrls;

type
  TForm1 = class(TForm)
    HR1: THTTPIO;
    edIdade: TEdit;
    Button1: TButton;
    lbDias: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
  uses IwsIdade1;

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  IIidade: IwsIdade;
begin
  IIidade:=HR1 as IwsIdade;
  lbDias.Caption:='Você já viveu aproximadamente
'+InttoStr(IIidade.QuantosDias(StrToInt(edIdade.Text)))+ ' dias';
end;

end.

```

Listagem 13.14 IwsIdade1 (implementação da Interface Idade)

```

// ***** //
// The types declared in this file were generated from data read from the
// WSDL File described below:
// WSDL      : http://localhost/delphi/cgi-bin/ws2.exe/wsdl/IwsIdade
// Encoding  : utf-8
// Version   : 1.0
// (28/08/2002 19:06:33 - 1.33.2.5)
// ***** //

unit IwsIdade1;

interface

uses InvokeRegistry, SOAPHTTPClient, Types, XSBuiltIns;

```

```

type
    // ***** //
    // The following types, referred to in the WSDL document are not being represented
    // in this file. They are either aliases[@] of other types represented or were
referred
    // to but never[!] declared in the document. The types from the latter category
    // typically map to predefined/known XML or Borland types; however, they could also
    // indicate incorrect WSDL documents that failed to declare or import a schema
type.
    // ***** //
    // !:int          - "http://www.w3.org/2001/XMLSchema"

    // ***** //
    // Namespace : urn:wsIdadeIntf-IwsIdade
    // soapAction: urn:wsIdadeIntf-IwsIdade#QuantosDias
    // transport : http://schemas.xmlsoap.org/soap/http
    // style      : rpc
    // binding    : IwsIdadebinding
    // service    : IwsIdadeservice
    // port       : IwsIdadePort
    // URL        : http://localhost/delphi/cgi-bin/ws2.exe/soap/IwsIdade
    // ***** //
    IwsIdade = interface(IInvokable)
    ['{28CE8152-2421-53DD-F897-0FAAB7C4FB3B}']
        function  QuartosDias(const idade: Integer): Integer; stdcall;
    end;

function GetIwsIdade(UseWSDL: Boolean=System.False; Addr: string=''; HTTPRIO:
THHTTPRIO = nil): IwsIdade;

implementation

function GetIwsIdade(UseWSDL: Boolean; Addr: string; HTTPRIO: THHTTPRIO): IwsIdade;
const
    defWSDL = 'http://localhost/delphi/cgi-bin/ws2.exe/wsdl/IwsIdade';
    defURL  = 'http://localhost/delphi/cgi-bin/ws2.exe/soap/IwsIdade';
    defSvc  = 'IwsIdadeservice';
    defPrt  = 'IwsIdadePort';
var
    RIO: THHTTPRIO;
begin
    Result := nil;
    if (Addr = '') then
    begin
        if UseWSDL then
            Addr := defWSDL
        else
            Addr := defURL;
    end;
    if HTTPRIO = nil then
        RIO := THHTTPRIO.Create(nil)
    else
        RIO := HTTPRIO;
    try
        Result := (RIO as IwsIdade);
    
```

```

    if UseWSDL then
    begin
        RIO.WSDLLocation := Addr;
        RIO.Service := defSvc;
        RIO.Port := defPrt;
    end else
        RIO.URL := Addr;
    finally
        if (Result = nil) and (HTTPRIO = nil) then
            RIO.Free;
        end;
    end;
end;

initialization
    InvRegistry.RegisterInterface(TypeInfo(IwsIdade), 'urn:wsIdadeIntf-IwsIdade', 'utf-8');
    InvRegistry.RegisterDefaultSOAPAction(TypeInfo(IwsIdade), 'urn:wsIdadeIntf-IwsIdade#QuantosDias');

end.

```

Listagem 13.15 un_cgi_ws2 (teste modelo CGI)

```

unit un_cgi_ws2;

interface

uses
    SysUtils, Classes, HTTPApp, InvokeRegistry, Rio, SOAPHTTPClient;

type
    TWebModule1 = class(TWebModule)
        HR1: THHTTPRIO;
        procedure WebModuleIdiasAction(Sender: TObject; Request: TWebRequest;
            Response: TWebResponse; var Handled: Boolean);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    WebModule1: TWebModule1;

implementation
    uses IwsIdade1;
{$R *.dfm}

procedure TWebModule1.WebModuleIdiasAction(Sender: TObject;
    Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
    IIdade: IwsIdade;
begin
    IIdade:=HR1 as IwsIdade;
    Response.Content:='Você já viveu aproximadamente
'+InttoStr(IIdade.QuantosDias(StrToInt(Request.QueryFields.Values['idade'])))+'
dias';

```

```
end;
```

```
end.
```

WebServices com Banco de Dados

Neste exemplo vamos criar nosso *WebService* que retorna informações do nosso banco de dados. Nosso *WebService* consiste em fazer uma pesquisa em nossa tabela de clientes e retornar a Razão Social. Imagine que o código do cliente é uma espécie de C.G.C, onde qualquer contribuinte, seja pessoa física ou jurídica, teria acesso aos dados da empresa, em sua própria aplicação.

Através das opções *File/New...*, seção *WebServices*, selecione a opção *SOAP Server Application* (figura 13.35).

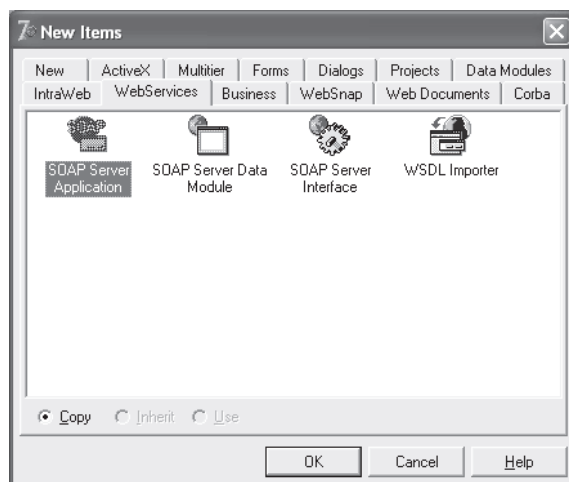


Figura 13.35 Nova aplicação *WebService*

Em seguida selecione a opção *CGI* para o tipo da aplicação servidora *SOAP* (figura 13.36).

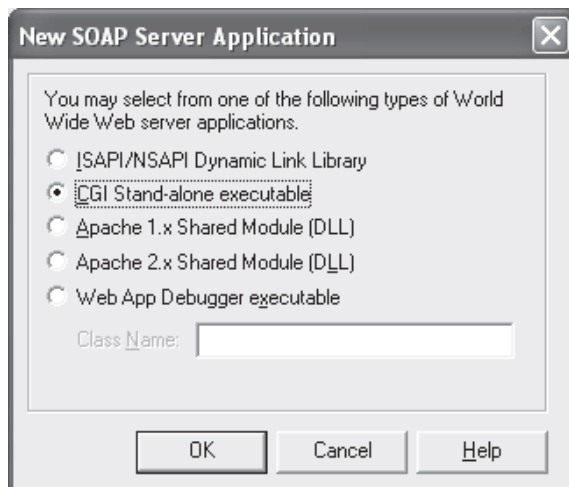


Figura 13.36 Tipo da aplicação servidora

Na janela de diálogo que segue (figura 13.37), não confirme a criação da *Interface*.

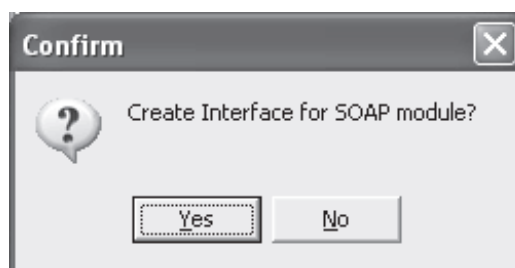


Figura 13.37 Criação da Interface

Vamos gravar nossa aplicação.

Unit WebModule	un_ws3.PAS
Projeto	ws3.DPR

Agora vamos inserir um *SOAP Data Module* em nossa aplicação. Através das opções *File/New..WebServices*, selecione a opção *SOAP Server Data Module* (figura 13.38), e em seguida pressione OK.

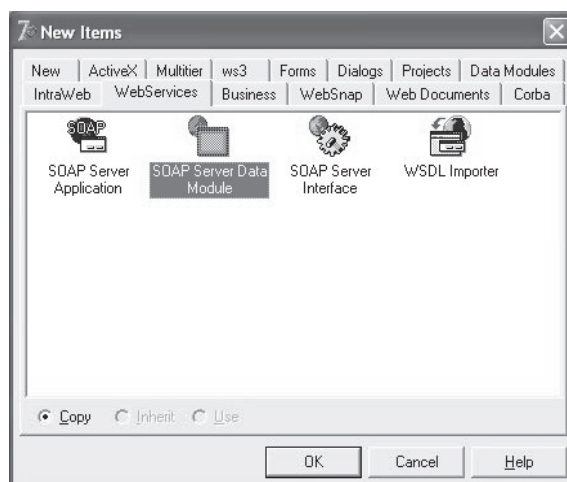


Figura 13.38 Soap Server Data Module

Em seguida (figura 13.39) informe o nome do nosso *Soap Server Data Module*, como *ws3dm*.

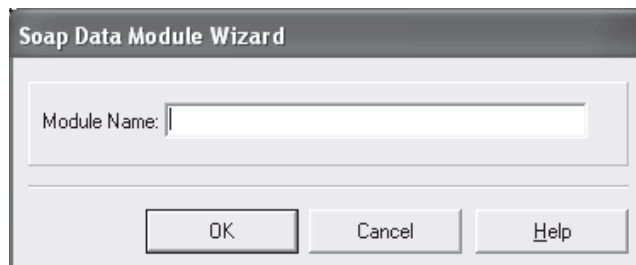


Figura 13.39 Module Name

Grave a *unit* como *un_ws3_dm.pas*.

Insira um objeto do tipo *TSQLConnection*. Através do duplo-clique, já na tela de configuração, aponte para a nossa conexão Clientes, criada anteriormente. Vamos relembrar os atributos da conexão.

PROPRIEDADE	VALOR
CommitRetain	False
Database	<i>localhost:c:\cursoweb\clientes.gdb</i>
Password	a famosa masterkey
UserName	o famoso SYSDBA
Name	ConexaoBD

Altere também a propriedade *LoginPrompt* para false.

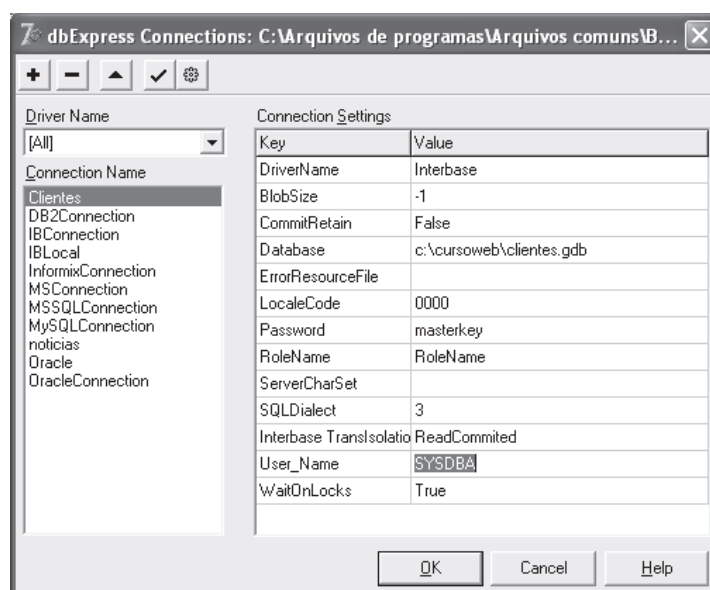


Figura 13.40 Configuração da Conexão

Agora vamos inserir o objeto para manipular nossa tabela de clientes. Insira um objeto do tipo *TSQLDataSet*, e altere as seguintes propriedades:

PROPRIEDADE	VALOR
SQLConnection	ConexaoBD
CommandText	select * from TBCLIENTE where COD_CLIENTE=:pcodigo

Configure o tipo do parâmetro *pcodigo* para *Integer*.

Agora vamos codificar a *Interface* do nosso *WebService*. Na *unit un_ws3_dm*, insira o código que segue, adequando ao código já existente.

```

type
  Iws3dm = interface(IAppServerSOAP)
    ['{7D59A6B9-A6C8-4FC4-B105-C92A0CDA478B}']
    function MostraRazao(codigo:integer):String; stdcall;
  end;

  Tws3dm = class(TSoapDataModule, Iws3dm, IAppServerSOAP, IAppServer)
    ConexaoBD: TSQLConnection;
    SQLClientes: TSQLDataSet;
  private

  public
    function MostraRazao(codigo:integer):String; stdcall;
  end;

```

Repare que criamos o método *MostraRazao*, que implementaremos agora na seção *implementation*. Agora, na seção *implementation*, insira completamente o código a seguir:

```

function Tws3dm.MostraRazao(codigo:integer):String;
begin
  { configura o parâmetro }
  SQLClientes.ParamByName('pcodigo').Value:=codigo;
  SQLClientes.Open;

  if not(SQLClientes.Eof) then
    Result:=SQLClientes.FieldValues['RAZAO_SOCIAL']
  else
    Result:='Cliente inexistente';

  SQLClientes.Close;

end;

```

Vamos analisar o código. No bloco a seguir, estamos atribuindo o código transmitido através do *Client* em nosso parâmetro e abrindo o *DataSet*.

```

SQLClientes.ParamByName('pcodigo').Value:=codigo;
SQLClientes.Open;

```

Em seguida, verificamos a existência do cliente, e em caso afirmativo retornamos o valor do campo *Razao_Social*, caso contrário, retornamos a mensagem “Cliente inexistente!”. E para finalizar, fechamos o nosso *DataSet*.

```

if not(SQLClientes.Eof) then
  Result:=SQLClientes.FieldValues['RAZAO_SOCIAL']
else
  Result:='Cliente inexistente';

SQLClientes.Close;

```

O nosso *WebService* está prontinho pra ser utilizado. Agora vamos criar nossas aplicações *Client*. Através das opções *File/New Application* crie uma nova aplicação e grave os arquivos como segue:

Unit	un_teste_ws3.PAS
Projeto	teste_ws3.DPR

Agora vamos importar a *Interface* em nossa aplicação. Através das opções *File/New.../WebServices* (figura 13.41), selecione a opção *WSDL Importer*.

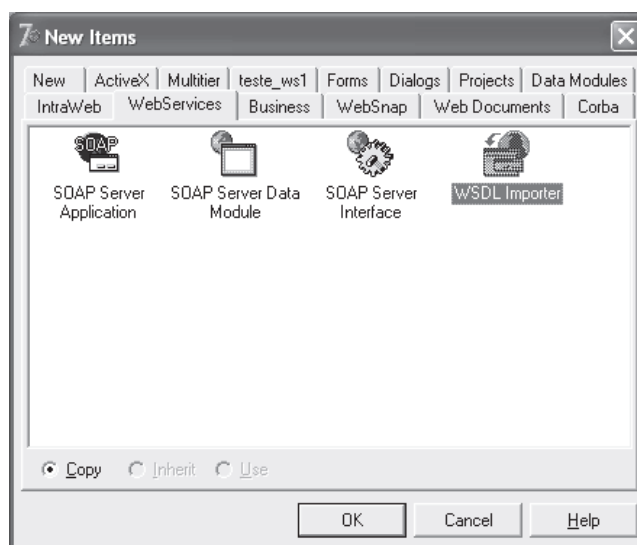


Figura 13.41 WSDL importer

Em seguida, como ilustra a *figura 13.42*, digite o endereço que segue.

```
http://localhost/cgi-bin/ws3.exe/wsdl/Iws3DM
```

Este endereço faz a chamada ao *documento WSDL* da *Interface Iws3DM*.

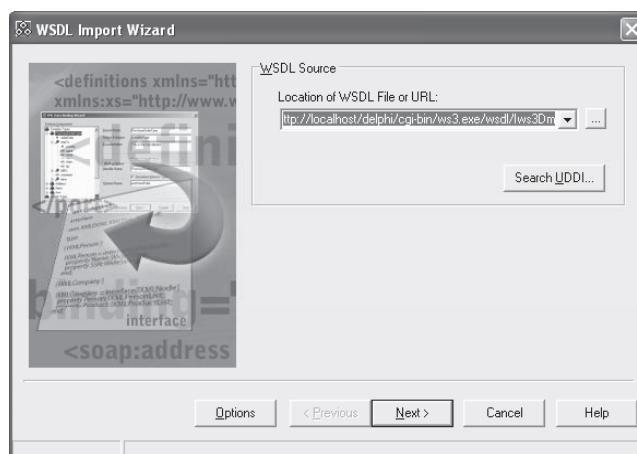



Figura 13.42 Importando a Interface

O assistente gerou uma *Unit* com toda a *Interface* implementada. Grave a *Unit* com o nome *Iws3Dm1.pas*.


Agora com o foco na *unit un_teste_ws3*, insira a *unit Iws3DM1.pas* gerada pelo assistente.


```
implementation
uses Iws3DM1;
```


Neste ponto iremos configurar o acesso para este formulário. Insira um objeto do tipo *THTTTPRIO* e configure as propriedades que seguem, respeitando a seqüência apresentada, caso contrário, uma exceção ocorrerá.

OBJETO		
	THTTPRio	
Objeto	Propriedade	Valor
HTTPRio1	Name	HR1
	WSDLLocation	http://localhost/cgi-bin/ws3.exe/wsd/lws3dm
	Service	lws3dmService
	Port	lws3DmPort

Com isto configuramos o objeto de acesso à *Interface*, informando a localização do documento *WSDL* (*WSDLLocation*), o serviço (*Service*), e a porta (*Port*). Agora insira os objetos que seguem, configurando suas respectivas propriedades.

OBJETO		
	TEdit	
Objeto	Propriedade	Valor
edCodigo	Name	edCodigo
	Left	32
	Text	deixe em branco
	Top	24
	Width	120

OBJETO		
	TButton	
Objeto	Propriedade	Valor
Button1	Name	Button1
	Caption	Pesquisa
	Left	176
	Top	24
	Width	125

OBJETO		
	Tlabel	
Objeto	Propriedade	Valor
lbRazao	Name	lbRazao
	Caption	Cliente
	Left	32
	Top	64

A figura 13.43 ilustra o formulário da nossa aplicação.

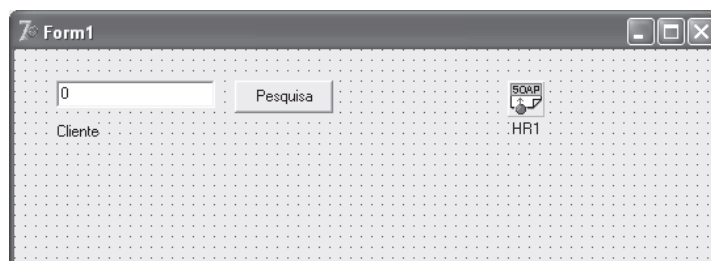


Figura 13.43 Formulário aplicação teste3

Agora vamos implementar o código do formulário. No evento *OnClick* do botão, insira o código que segue:

```
var
IRazao:IWS3DM;
begin
  IRazao:=HR1 as Iws3DM;
  lbRazao.Caption:=IRazao.MostraRazao( StrToInt(
    edCodigo.Text ));
end;
```

Estamos fazendo uma operação bastante simples, passando o parâmetro código para o nosso *Webservice* e apresentando o resultado no objeto *lbRazao*.

As figuras 13.44 e 13.45 ilustram nossa aplicação em tempo de execução.

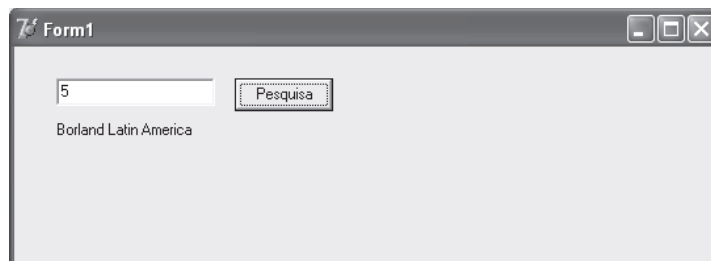


Figura 13.44 Cliente encontrado

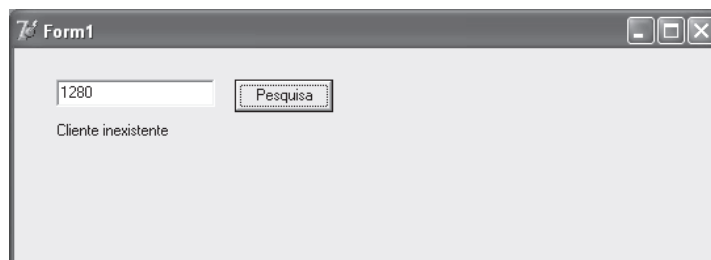


Figura 13.45 Cliente inexistente

Amigos, isto é fantástico, não é?

Agora vamos desenvolver uma aplicação no padrão *CGI* para consultar a razão social de um cliente.

A partir do Delphi, selecione as opções *File/New/Other...* e em seguida a opção *Web Server Application*, como ilustra a figura 13.46.

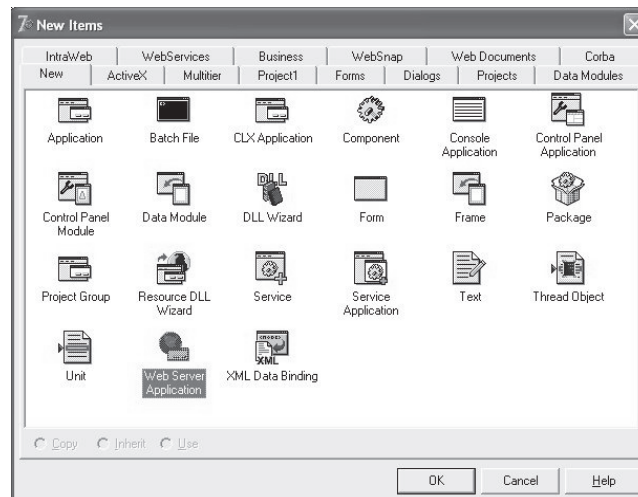


Figura 13.46 Opção *Web Server Application*

Na janela seguinte selecione a opção *CGI Stand-Alone executable* (figura 13.47).

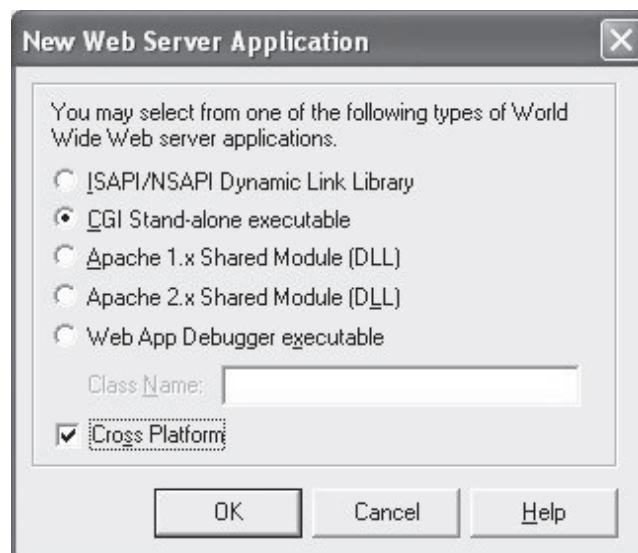


Figura 13.47 Seleção do tipo da aplicação

Em seguida teremos o nosso *WebModule* (figura 13.48).

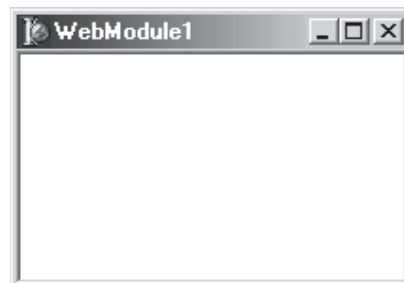



figura 13.48 *WebModule*

Neste ponto ao invés de importar o *WSDL*, vamos apenas adicionar a *unit* que já importamos em nossa primeira aplicação: *Iws3DM*;

```
implementation
uses Iws3DM;
```

Neste ponto iremos configurar o acesso para o nosso *CGI*. Insira um objeto do tipo *THHTPRIO* e configure as propriedades que seguem, respeitando a sequência apresentada, caso contrário, uma exceção ocorrerá.

OBJETO		
	THHTPRIO	
Objeto	Propriedade	Valor
HTTPrIo1	Name	HR1
	WSDLLocation	http://localhost/cgi-bin/ws3.exe/wsdl/lws3dm
	Service	lws3dmService
	Port	lws3DmPort

Agora vamos criar nossa *Action*. Através do duplo-clique no *WebModule*, acesse o editor de *ActionItems* (figura 13.49).

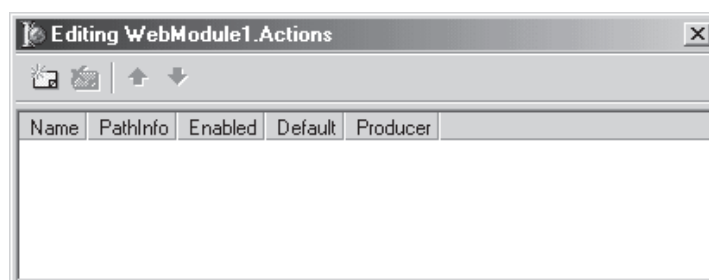


Figura 13.49 editor *ActionItems*

Clique no primeiro botão do editor para inserir uma nova *Action* (figura 13.50).

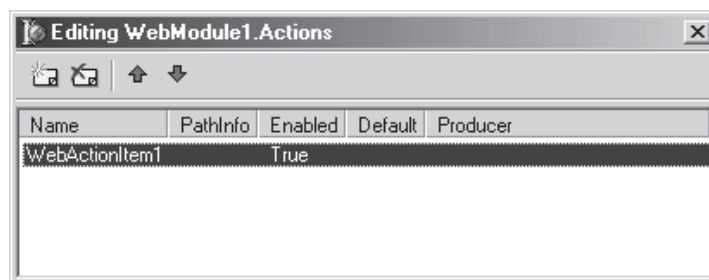



Figura 13.50 *ActionItem*

Em seguida altere as seguintes propriedades.

OBJETO		
 TWebActionItem		
Objeto	Propriedade	Valor
Padrao	Default	True
	Name	consulta
	PathInfo	/consulta

Esta será nossa *Action padrão*, ou seja, caso o usuário não digite nada, além do nome da nossa aplicação, esta *Action* será executada. No evento *OnAction* coloque o seguinte código:

```
var
IRazao: Iws3DM1;
begin
  IRazao:=HR1 as Iws3DM;
  Response.Content:='Cliente =
'+ IRazao.MostraRazao(StrToInt(Request.QueryFields.Values['codigo']));
end;
```

Aqui estamos utilizando o método *Response.Content* para apresentar ao usuário a mensagem criada através do parâmetro *codigo*, utilizando o método *MostraRazao* do nosso *WebService*.

Antes de compilar, vamos definir o diretório para geração do nosso *WebService*. Através das opções *Project/Options.../Directories_Conditionals*, configure a opção *Output Directory*, apontando para o seu diretório *cgi-bin* (figura 13.51).

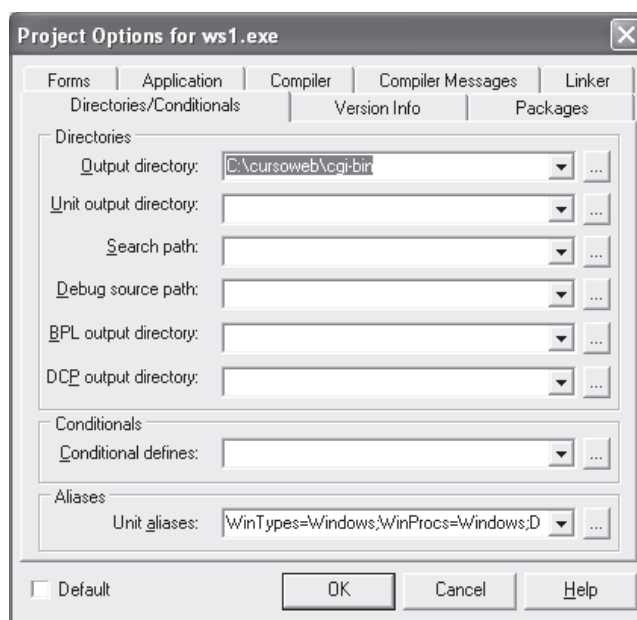


Figura 13.51 Configuração do diretório

Grave os arquivos como segue.

Unit	un_cgi_ws3.PAS
Projeto	teste_cgi_ws3.DPR

Agora vamos testar a nossa aplicação. No browser digite o seguinte endereço:

```
http://localhost/delphi/cgi-bin/teste_cgi_ws3.exe/consulta?codigo=14
```

A figura 13.52 ilustra nosso CGI em tempo de execução.

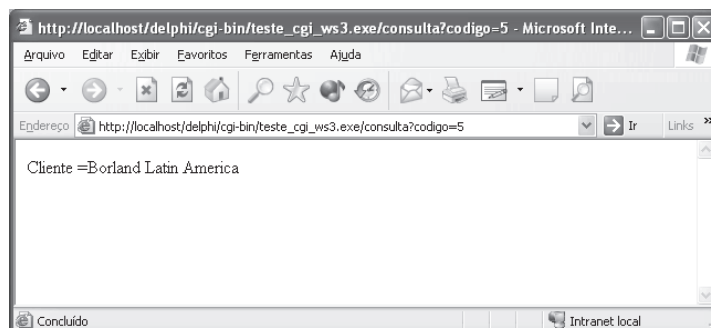


figura 13.52 Resultado da aplicação

Agora deu para perceber o poder do *WebService*, não? No próximo tópico, iremos desenvolver uma aplicação utilizando o protocolo *SOAP* em conjunto com a tecnologia *DataSNAP*.

Listagem 13.16 un_teste_ws3.pas

```
unit un_teste_ws3;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, InvokeRegistry, Rio, SOAPHTTPClient, StdCtrls;

type
  TForm1 = class(TForm)
    edCodigo: TEdit;
    lbRazao: TLabel;
    Button1: TButton;
    HR1: THHTTPRIO;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
  uses iws3dml;
  {$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  IRazao:IWS3DM;
begin
  IRazao:=HR1 as Iws3DM;
```

```

        lbRazao.Caption:=Irazao.MostraRazao(StrToInt(edCodigo.Text));
end;

end.

```

Listagem 13.17 Interface Iws3DM

```

// ***** //
// The types declared in this file were generated from data read from the
// WSDL File described below:
// WSDL      : http://localhost/delphi/cgi-bin/ws3.exe/wsdl/Iws3Dm
// Encoding  : utf-8
// Version   : 1.0
// (01/09/2002 12:34:46 - 1.33.2.5)
// ***** //

unit Iws3Dm1;

interface

uses InvokeRegistry, SOAPHTTPClient, Types, XSBuiltIns, SOAPMidas;

type

    // ***** //
    // The following types, referred to in the WSDL document are not being represented
    // in this file. They are either aliases[] of other types represented or were
    // referred
    // to but never[] declared in the document. The types from the latter category
    // typically map to predefined/known XML or Borland types; however, they could also
    // indicate incorrect WSDL documents that failed to declare or import a schema
    type.
    // ***** //
    // !:string      - "http://www.w3.org/2001/XMLSchema"
    // !:int          - "http://www.w3.org/2001/XMLSchema"

    // ***** //
    // Namespace : urn:un_ws3_dm-Iws3dm
    // soapAction: urn:un_ws3_dm-Iws3dm#MostraRazao
    // transport  : http://schemas.xmlsoap.org/soap/http
    // style      : rpc
    // binding    : Iws3dmbinding
    // service    : Iws3dmservice
    // port       : Iws3DmPort
    // URL        : http://localhost/delphi/cgi-bin/ws3.exe/soap/Iws3Dm
    // ***** //
    Iws3dm = interface(IAppServerSOAP)
    ['{B8148078-61EE-65BD-6196-A55E0A9CA57C}']
        function MostraRazao(const codigo: Integer): WideString; stdcall;
    end;

implementation

initialization

```

```

    InvRegistry.RegisterInterface(TypeInfo(Iws3dm), 'urn:un_ws3_dm-Iws3dm', 'utf-8');
    InvRegistry.RegisterDefaultSOAPAction(TypeInfo(Iws3dm), 'urn:un_ws3_dm-Iws3dm#MostraRazao');
end.

```

Listagem 13.18 un_cgi_ws3

```

unit un_cgi_ws3;

interface

uses
    SysUtils, Classes, HTTPApp, InvokeRegistry, Rio, SOAPHTTPClient;

type
    TWebModule1 = class(TWebModule)
        HR1: THTTPIO;
        procedure WebModule1consultaAction(Sender: TObject;
            Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    WebModule1: TWebModule1;

implementation
    uses Iws3DM1;
{$R *.dfm}

procedure TWebModule1.WebModule1consultaAction(Sender: TObject;
    Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
    IRazao: Iws3DM;
begin
    IRazao:=HR1 as Iws3DM;
    Response.Content:='Cliente
='+IRazao.MostraRazao(StrToInt(Request.QueryFields.Values['codigo']));
end;

end.

```

SOAP DataSnap XML

Neste exemplo vamos criar um *WebService* que fornece um modelo de comunicação de dados muito interessante. O protocolo *SOAP* responsável pelo transporte das informações e a tecnologia *DataSnap* pelo fornecimento e empacotamento dos dados. Teremos ainda, uma aplicação de teste bastante simples.

Bem, vamos iniciar o desenvolvimento do nosso *WebService*. Através das opções *File/New...*, seção *WebServices*, selecione a opção *SOAP Server Application* (figura 13.53).

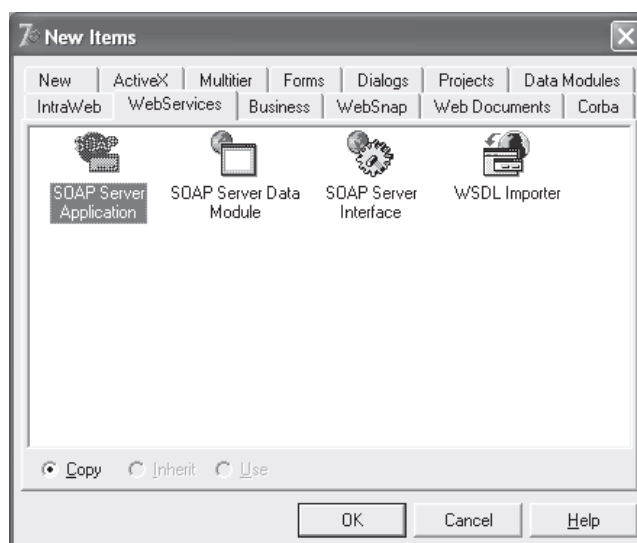


Figura 13.53 Nova aplicação WebService

Em seguida, selecione a opção *CGI* para o tipo da aplicação servidora *SOAP* (figura 13.54).



Figura 13.54 Tipo da aplicação servidora

Na janela de diálogo que segue (figura 13.55), não confirme a criação da *Interface*.

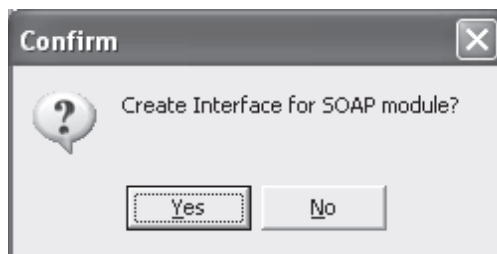


Figura 13.55 Criação da Interface

Vamos gravar nossa aplicação.

Unit WebModule	un_ws4.PAS
Projeto	ws4.DPR

Agora vamos inserir um *SOAP Data Module* em nossa aplicação. Através das opções *File/New..WebServices*, selecione a opção *SOAP Server Data Module* (figura 13.56), e em seguida pressione *OK*.

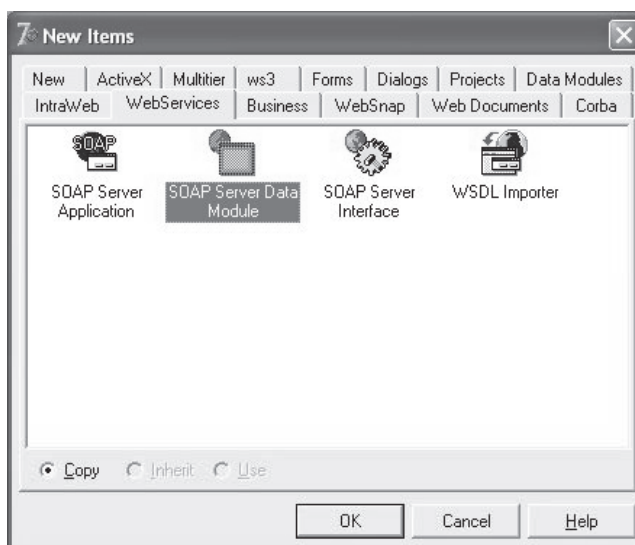


Figura 13.56 Soap Server Data Module

Em seguida (figura 13.57) informe o nome do nosso *Soap Server Data Module*, como *ws4dm*.



Figura 13.57 Module Name

Grave a *unit* como *un_ws4_dm.pas*. Insira um objeto do tipo *TSQLConnection*. Através do duplo-clique, já na tela de configuração, aponte para a nossa conexão *Clientes*, criada anteriormente. Vamos relembrar os atributos da conexão.

PROPRIEDADE	VALOR
CommitRetain	False
Database	<i>localhost:c:\cursoweb\clientes.gdb</i>
Password	a famosa masterkey
UserName	o famoso SYSDBA
Name	ConexaoBD

Altere também a propriedade *LoginPrompt* para false.

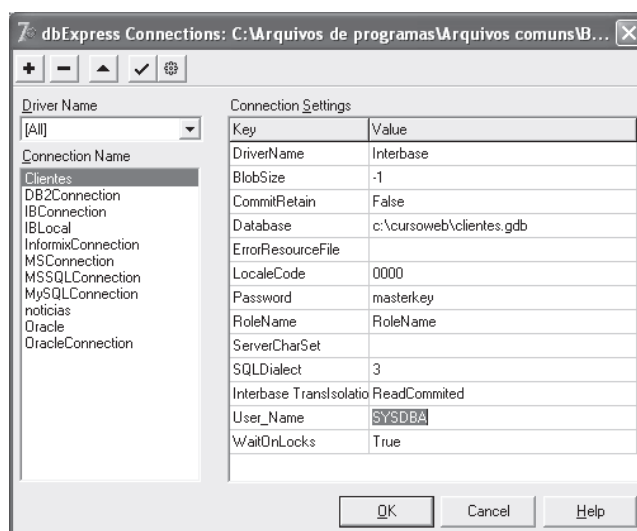


Figura 13.58 Configuração da Conexão

Agora vamos inserir o objeto para manipular nossa tabela de clientes. Insira um objeto do tipo *TSQLDataSet*, e altere as seguintes propriedades:

PROPRIEDADE	VALOR
SQLConnection	ConexaoBD
CommandText	select * from TBCLIENTE
Name	SQLClientes
Active	True

Insira um objeto do tipo *TDataSetProvider* para fornecer os dados de nosso *DataSet* a qualquer cliente conectado em nossa aplicação.

PROPRIEDADE	VALOR
DataSet	SQLClientes
Name	DataSetProvider1

Antes de compilar, vamos definir o diretório para geração do nosso *WebService*.

Através das opções *Project/Options.../Directories_Conditionals*, configure a opção *Output Directory*, apontando para o seu diretório cgi-bin (figura 13.59).

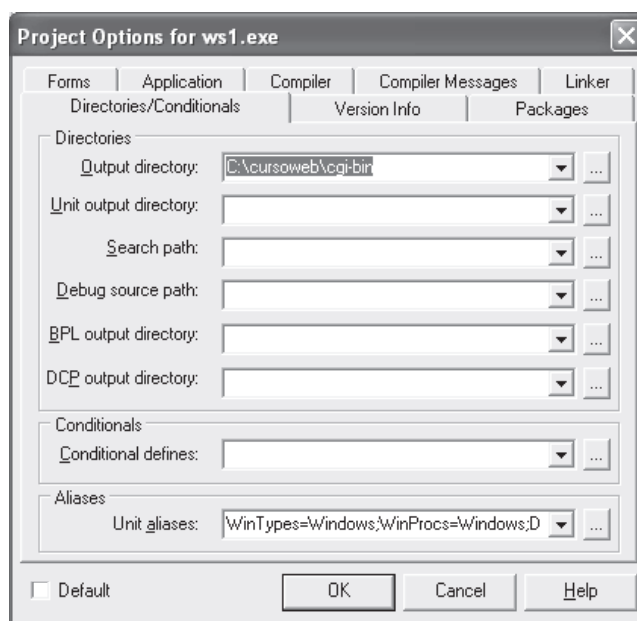


Figura 13.59 Configuração do diretório

Grave e compile a aplicação. Agora vamos construir nossa aplicação *Client*. Através das opções *File/New Application* crie uma nova aplicação e grave os arquivos como segue:

Unit	un_teste_ws4.PAS
Projeto	teste_ws4.DPR

Agora insira um objeto do tipo *TSOAPConnection* e altere as propriedades que seguem:

PROPRIEDADE	VALOR
URL	http://localhost/cgi-bin/ ws4.exe/ soap /Iws4dm
Name	SOAPConnection1
Connected	True

Perceba que estamos utilizando o método *SOAP* do nosso *WebService*. Agora, insira um objeto do tipo *TClientDataSet* e altere as seguintes propriedades.

PROPRIEDADE	VALOR
RemoteServer	SoapConnection1
ProviderName	DataSetProvider1
Active	True

Através do duplo-clique no objeto *ClientDataSet1*, insira os campos, como ilustra a *figura 13.60*.



Figura 13.60 Campos do ClientDataSet1

Agora, selecione todos os campos e arraste para o formulário, dimensionando como sugere a *figura 13.61*.

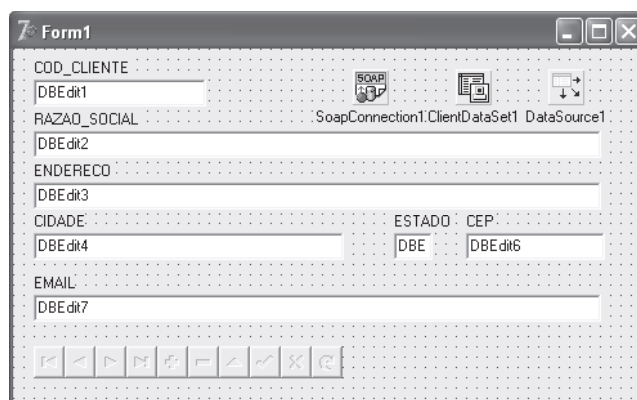


Figura 13.61 Formulário exemplo

Com a propriedade *Active* do *ClientDataSet1* configurada como *True*, execute a aplicação e veja o resultado (*figura 13.62*).

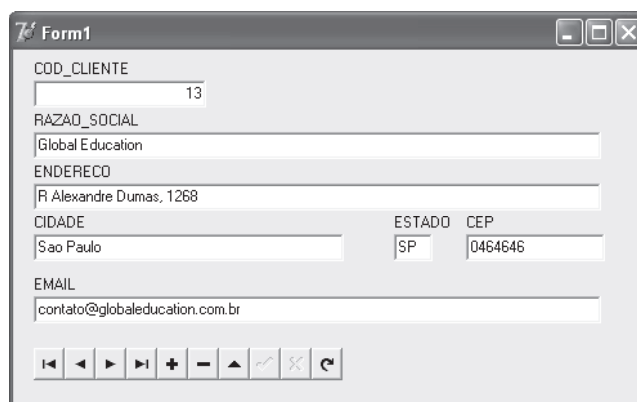


Figura 13.62 Aplicação em execução

Amigos, com isso concluímos nosso capítulo de *WebServices*.

Listagem 13.19 un_ws4_dm.pas

```
Unit un_ws4_dm;

interface

uses SysUtils, Classes, InvokeRegistry, Midas, SOAPMidas, SOAPDm, DBXpress,
    FMTBcd, DB, SqlExpr, Provider;

type
    Iws4dm = interface(IAppServerSOAP)
        ['{CA9392ED-702B-4596-A577-E341279D98D4}']
    end;

    Tws4dm = class(TSoapDataModule, Iws4dm, IAppServerSOAP, IAppServer)
        ConexaoBD: TSQLConnection;
        SQLClientes: TSQLDataSet;
        DataSetProvider1: TDataSetProvider;
    private
    public
    end;

implementation

{$R *.DFM}

procedure Tws4dmCreateInstance(out obj: TObject);
begin
    obj := Tws4dm.Create(nil);
end;

initialization
    InvRegistry.RegisterInvokableClass(Tws4dm, Tws4dmCreateInstance);
    InvRegistry.RegisterInterface(TypeInfo(Iws4dm));
end.
```

Listagem 13.20 un_teste_ws4.pas

```
unit un_teste_ws4;

interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, DB, DBClient, SOAPConn, StdCtrls, Mask, DBCtrls, ExtCtrls;

type
    TForm1 = class(TForm)
        SoapConnection1: TSoapConnection;
        ClientDataSet1: TClientDataSet;
        ClientDataSet1COD_CLIENTE: TIntegerField;
        ClientDataSet1RAZÃO_SOCIAL: TStringField;
        ClientDataSet1ENDERECO: TStringField;
        ClientDataSet1CIDADE: TStringField;
        ClientDataSet1ESTADO: TStringField;
        ClientDataSet1CEP: TStringField;
```

```
ClientDataSet1EMAIL: TStringField;
Label1: TLabel;
DBEdit1: TDBEdit;
DataSource1: TDataSource;
Label2: TLabel;
DBEdit2: TDBEdit;
Label3: TLabel;
DBEdit3: TDBEdit;
Label4: TLabel;
DBEdit4: TDBEdit;
Label5: TLabel;
DBEdit5: TDBEdit;
Label6: TLabel;
DBEdit6: TDBEdit;
Label7: TLabel;
DBEdit7: TDBEdit;
DBNavigator1: TDBNavigator;
private
{ Private declarations }
public
{ Public declarations }
end;

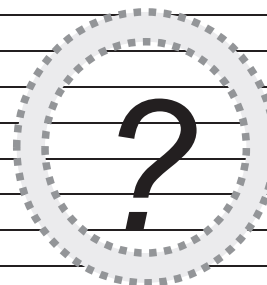
var
Form1: TForm1;

implementation

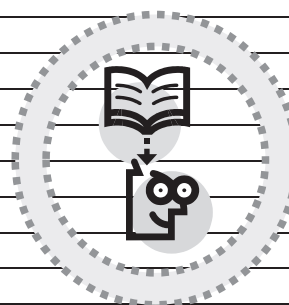
{$R *.dfm}

end.
```

Anotações de Dúvidas



Preciso Revisar



Anotações Gerais

