UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

Relatórios Técnicos
do Departamento de Informática Aplicada
da UNIRIO
nº 0007/2010

Avaliação de Evoluções de Serviços em Conexões Ponto-a-Ponto

Henrique Prado Sousa
Leonardo Guerreiro Azevedo
Flávia Santoro
Fernanda Baião

Departamento de Informática Aplicada

UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Av. Pasteur, 458, Urca - CEP 22290-240
RIO DE JANEIRO – BRASIL

Projeto de Pesquisa

Grupo de Pesquisa Participante

Patrocínio

PETROBRAS

Relatórios Técnicos do DIA/UNIRIO, No. 0007/2010
Editor: Prof. Sean W. M. Siqueira Setembro, 2010

Avaliação de Evoluções de Serviços em Conexões Ponto-a-Ponto*

Henrique Prado Sousa, Leonardo Guerreiro Azevedo, Flávia Santoro, Fernanda Baião

Núcleo de Pesquisa e Prática em Tecnologia (NP2Tec)
Departamento de Informática Aplicada (DIA) – Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{henrique.souza, azevedo, flavia.santoro, fernanda.baiao}@uniriotec.br

Abstract. The best strategy for SOA deployment depends on the right understanding and analysis of different existing supporting tools. There are a lot of tools, available from different vendors, such as IBM, HP, Oracle/BEA, SAP, Microsoft etc. To choose tools which fit enterprise requirements is not an easy task. This work presents the analysis of technologies that support service point-to-point connection, the first step in a SOA deployment, as well as explicit the impacts of changes using those technologies.

Keywords: SOA, consumer and provider point-to-point connection, technologies for service development.

Resumo. A definição da melhor estratégia para implantação de uma arquitetura orientada a serviços depende do correto entendimento e análise das diferentes ferramentas do mercado. Existem muitas ferramentas, disponíveis por diferentes fornecedores, tais como IBM, HP, Oracle/BEA, SAP, Microsoft etc. A escolha das ferramentas que melhor atende às necessidades da organização não é uma tarefa simples. Este trabalho apresenta a análise de diferentes tecnologias para suporte à conexão ponto-a-ponto de serviços, forma inicial para introduzir SOA em uma organização, bem como os impactos das evoluções nas implementações dos serviços utilizando tais tecnologias.

Palavras-chave: SOA, conexão ponto-a-ponto entre consumidor e provedor, tecnologias para desenvolvimento de serviços.

---

* Trabalho patrocinado pela Petrobras.

Sumário

Figuras

1 Introdução

Service-Oriented Architecture (SOA) é o tópico mais comentado por fornecedores de TI. Fornecedores como IBM, HP, BEA, Oracle, SAP, Microsoft dentre outros estão investindo fortemente neste tópico e desenvolvendo uma série de ferramentas para apoiar a arquitetura orientada a serviços [Hurwitz *et al.*, 2009, 2007]. Dessa forma, é importante entender bem as tecnologias bem como os padrões subjacentes a fim de definir a melhor estratégia para implantação e manutenção de uma iniciativa SOA na organização.

Este relatório tem o objetivo de avaliar as principais questões relacionadas à evolução na implementação de serviços, considerando a conexão ponto-a-ponto entre consumidor e provedor. Estas questões foram levantadas durante reuniões de equipe do projeto e a partir de estudos da literatura.

A fim de fazer um estudo das diferentes tecnologias utilizadas, os testes foram realizados considerando as seguintes plataformas de desenvolvimento: Workshop for WebLogic Platform, Eclipse/Axis, framework .Net. Também são apresentados os detalhes de implementações de clientes nestes ambientes.

Este relatório foi produzido pelo Projeto de Pesquisa em SOA como parte das iniciativas dentro do contexto do Projeto de Pesquisa do Termo de Cooperação entre NP2Tec/UNIRIO e a Petrobras/TIC-E&P/GDIEP.

Todos os serviços utilizados neste relatório, bem como os testes aplicados, foram implementados utilizando a ferramenta BEA Workshop for WebLogic Platform.

Esse relatório está organizado em 7 capítulos, sendo o capítulo 1 a presente introdução. No capítulo 2 são apresentados estudos das tecnologias. No capítulo 3 são apresentadas avaliações práticas realizadas usando o Workshop for WebLogic Platform para implementação do serviço e da aplicação cliente que invoca o serviço. Enquanto que os capítulos 4 e 5 apresentam os resultados obtidos utilizando as tecnologias Eclipse/Axis e .Net para implementação da aplicação cliente. No capítulo 6 são apresentadas as conclusões do trabalho e, no capítulo 7, as referências bibliográficas.

2 Estudo tecnologias

2.1 XML Beans

XMLBeans [Apache, 2009a] é uma ferramenta que permite acessar arquivos no formato XML através de objetos Java. O objetivo desta tecnologia é obter vantagem no uso de arquivos do tipo XML e XML Schema através de um mapeamento de seus atributos para a linguagem Java, possibilitando a manipulação natural do arquivo XML, instanciado como um objeto. A partir do XML Schema, é possível compilar classes e interfaces Java que permitem a manipulação simples da instância do arquivo XML, por exemplo, através de funções como get e set.

O arquivo XML Schema possui a extensão XSD e pode ser encontrado em diferentes versões, entretanto, o XML Beans suporta qualquer versão de definição do XML Schema e garante a integridade do documento em seu formato XML após manipulações em seu conteúdo. A Figura 1 apresenta um exemplo de XML Schema. O

esquema contém um elemento raiz *purchase-order* que é composto por quatro tipos de elementos: *customer, date, line-item, e shipper*. O elemento *customer* é do tipo *customer* e possui 2 atributos: *name* e *address*, ambos do tipo string. O elemento *date* é do tipo *dateTime*. O elemento *line-item* é do tipo *line-item* e possui *description*, *per-unit-ounces*, *price*, e *quantity*. O elemento *shipper* é do tipo *shipper* e possui *name* e *per-ounce-rate*.

```
xs:schema targetNamespace="http://openuri.org/easypo"
    xmlns:po="http://openuri.org/easypo"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

    <xs:element name="purchase-order">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="customer" type="po:customer"/>
                <xs:element name="date" type="xs:dateTime"/>
                <xs:element name="line-item" type="po:line-item"
minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="shipper" type="po:shipper"
minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="customer">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="line-item">
        <xs:sequence>
            <xs:element name="description" type="xs:string"/>
            <xs:element name="per-unit-ounces" type="xs:decimal"/>
            <xs:element name="price" type="xs:double"/>
            <xs:element name="quantity" type="xs:int"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="shipper">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="per-ounce-rate" type="xs:decimal"/>
        </xs:sequence>
    </xs:complexType>

   </xs:schema>
```

Figura 1 – Exemplo de um XML Schema (XSD)

As classes geradas pelo mapeamento do arquivo XSD oferecem construtores para a grande maioria de suas funcionalidades, permitindo o máximo aproveitamento dos recursos oferecidos pelo XML Schema, sem restrições a apenas um subconjunto de suas funcionalidades. Esse fator é importante, uma vez que frequentemente não é possível obter o controle sobre algumas características do XML Schema que se deseja manipular em Java. Além disso, caso a instância de um XML tenha seu padrão modificado, o conjunto de informações originais se mantém imutável e disponível ao desenvolvedor. Isso é importante, porque em certas ocasiões, é necessário realizar modificações estruturais para solucionar problemas em aplicações específicas.

O desenvolvedores da tecnologia afirmam que durante o projeto do XMLBeans, foi dedicada atenção especial a performance e que testes informais aplicados em ambientes utilizando XMLBeans obtiveram ótimo desempenho.

A Figura 2 apresenta um exemplo de instância do XML Schema apresentado na Figura 1.

```
<po:purchase-order xmlns:po="http://openuri.org/easypo">
    <po:customer>
        <po:name>Gladys Kravitz</po:name>
        <po:address>Anytown, PA</po:address>
    </po:customer>
    <po:date>2003-01-07T14:16:00-05:00</po:date>
    <po:line-item>
        <po:description>Burnham's Celestial Handbook, Vol
1</po:description>
        <po:per-unit-ounces>5</po:per-unit-ounces>
        <po:price>21.79</po:price>
        <po:quantity>2</po:quantity>
    </po:line-item>
    <po:line-item>
        <po:description>Burnham's Celestial Handbook, Vol
2</po:description>
        <po:per-unit-ounces>5</po:per-unit-ounces>
        <po:price>19.89</po:price>
        <po:quantity>2</po:quantity>
    </po:line-item>
<po:shipper>
        <po:name>ZipShip</po:name>
        <po:per-ounce-rate>0.74</po:per-ounce-rate>
    </po:shipper>
</po:purchase-order>
```

Figura 2 – Exemplo de instância de um XML Schema

A especificação do XML Schema oferece um rico modelo de dados que permite expressar uma sofisticada estrutura e restrições nos dados. Por exemplo, um XML Schema pode forçar o controle sobre como a data é ordenada em um documento, ou restrições em valores particulares (por exemplo, uma data de aniversário deve ser considerada somente após 1990). Infelizmente, a habilidade de forçar regras como esta não é possível em Java sem escrever o respectivo código. XML Beans obedece às restrições do esquema.

3 Avaliações práticas utilizando Workshop for Weblogic Platform

Esta seção apresenta as avaliações práticas realizadas.

3.1 Implementação do serviço e cliente utilizados nos testes

O serviço utilizado como ilustração nas seções seguintes e o cliente utilizado para os testes foram implementados de acordo com o passo a passo descrito em [Azevedo *et al.*, 2009]. A versão do *plugin* Axis utilizada para desenvolver o cliente no Workshop é 1.2.1 v.200602062208.

3.1.1 Passos necessários para atualizar estrutura do objeto retornado pelo serviço

Para inserir ou remover algum atributo do objeto sendo retornado pelo serviço, é necessário atualizar o conjunto de arquivos a seguir:

- Classe POJO – Classe que descreve o objeto a ser retornado.
- Arquivo XSD – É o XML Schema que define a estrutura do arquivo XML correspondente ao mapeamento do objeto Java para XML.
- Arquivo de tipos JAR – Arquivo com classes Java, gerado a partir do XML Schema.
- Classe de controle do banco de dados (dbControl) – Classe que executa o SQL para obter dados a partir de um banco de dados.

O passo a passo para atualizar atributos do objeto retornado pelo serviço é apresentado a seguir. Na seção 3.2 é apresentado um exemplo de remoção de atributo e na seção 0 é apresentado um exemplo de adição de atributo.

O primeiro passo é modificar a classe POJO, inserindo ou excluindo o atributo desejado. Caso seja incluído um atributo, deverão ser incluídos os métodos get e set para este atributo.

O segundo passo é atualizar o XML Schema (arquivo XSD) para contemplar as alterações realizadas na classe POJO. Então o elemento referente ao atributo da classe deverá ser apagado ou inserido neste arquivo.

O terceiro passo é gerar os arquivos JAR com as classes Java correspondentes aos elementos do esquema.

O quarto passo é atualizar a classe de controle do banco de dados (dbControl) para contemplar as variáveis que receberão os dados do banco de dados para preencher os atributos do objeto instanciado no comando SQL correspondente.

O quinto e último passo é atualizar o arquivo WSDL, que contemplará no contrato do serviço as modificações realizadas.

A aplicação destes passos pode ser verificada nos dois exemplos seguintes que demonstram a adição e remoção de atributos do objeto de retorno de um serviço.

3.1.2 Arquivos originais utilizados nos testes

O código que define a classe POJO, XML Schema, código do DBControl e o WSDL, antes de se realizar as alterações (remoção e adição de atributos) são apresentados nas Figura 3, Figura 4, Figura 5 e Figura 6.

```java
package br.com.petrobras.beans;

import java.math.BigDecimal;

public class UnidadeOperativa {
    private BigDecimal codigoUnidadeOperativa;
    private String siglaUnidadeOperativa;
    private String nomeUnidadeOperativa;
    private BigDecimal codigoUnidadeOperativaAGP;

    public BigDecimal getCodigoUnidadeOperativa() {
        return codigoUnidadeOperativa;
    }
    public void setCodigoUnidadeOperativa(BigDecimal codigoUnidadeOperativa) {
        this.codigoUnidadeOperativa = codigoUnidadeOperativa;
    }
    public BigDecimal getCodigoUnidadeOperativaAGP() {
        return codigoUnidadeOperativaAGP;
    }
    public void setCodigoUnidadeOperativaAGP(BigDecimal codigoUnidadeOperativaAGP) {
        this.codigoUnidadeOperativaAGP = codigoUnidadeOperativaAGP;
    }
    public String getNomeUnidadeOperativa() {
        return nomeUnidadeOperativa;
    }
    public void setNomeUnidadeOperativa(String nomeUnidadeOperativa) {
        this.nomeUnidadeOperativa = nomeUnidadeOperativa;
    }
    public String getSiglaUnidadeOperativa() {
        return siglaUnidadeOperativa;
    }
    public void setSiglaUnidadeOperativa(String siglaUnidadeOperativa) {
        this.siglaUnidadeOperativa = siglaUnidadeOperativa;
    }

}
```

Figura 3 – Codificação da classe POJO

```xml
<?xml version="1.0"?>
<xs:schema targetNamespace="http://controls.beans/unidadeoperativa" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns="http://controls.beans/unidadeoperativa" elementFormDefault="qualified"
xmlns:wld="http://www.bea.com/2002/10/weblogicdata" attributeFormDefault="unqualified">

    <xs:complexType name="unidadeoperativa">
        <xs:sequence>
            <xs:element name = "codigoUnidadeOperativa" type="xs:decimal"  nillable="true" minOccurs="0" maxOccurs="1" />
            <xs:element name = "siglaUnidadeOperativa" type="xs:string" nillable="true" minOccurs="0" maxOccurs="1" />
            <xs:element name = "nomeUnidadeOperativa" type="xs:string" nillable="true" minOccurs="0" maxOccurs="1" />
            <xs:element name = "codigoUnidadeOperativaAGP" type="xs:decimal" nillable="true" minOccurs="0" maxOccurs="1" />
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="lstUnidadeOperativa">
        <xs:sequence>
            <xs:element name="UnidadeOperativa" type="unidadeoperativa" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>

    <xs:element name="UnidadesOperativasList" type="lstUnidadeOperativa" />

</xs:schema>
```

Figura 4 – Codificação do XML Schema

```
package br.com.petrobras.controls;

import org.apache.beehive.controls.system.jdbc.JdbcControl;

@ControlExtension
@JdbcControl.ConnectionDataSource(jndiName = "UnidadeOperativaDS")
public interface UnidadeOperativaDbControl extends JdbcControl {

    static final long serialVersionUID = 1L;

    @JdbcControl.SQL(statement="  select  UNOP_CD_UNID_OPER as codigoUnidadeOperativa, " +
            "          UNOP_SG_UNID_OPER as siglaUnidadeOperativa," +
            "          UNOP_NM_UNID_OPER as nomeUnidadeOperativa," +
            "          UNOP_CD_AGP as codigoUnidadeOperativaAGP " +
            "  FROM unid_operativa  " +
            "  Where 1=1 " +
            "  {sql: p_where} " +
            "  {sql: p_groupby} " +
            "  {sql: p_having} " +
            "  {sql: p_orderby} ")



    UnidadeOperativa[] getUnidadeOperativa(String p_where, String p_groupby,
            String p_having, String p_orderby) throws SQLException;

}
```

Figura 5 – Codificação da classe de controle do banco de dados (dbControl)

```
<?xml version='1.0' encoding='UTF-8'?>
<s0:definitions name="UnidadeOperativaServiceServiceDefinitions"
targetNamespace="http://br/com/uniriotec/services" xmlns=""
xmlns:s0="http://schemas.xmlsoap.org/wsdl/"
xmlns:s1="http://br/com/uniriotec/services"
xmlns:s2="http://schemas.xmlsoap.org/wsdl/soap/">
  <s0:types>
    <xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
targetNamespace="http://controls.beans/unidadeoperativa"
xmlns="http://controls.beans/unidadeoperativa"
xmlns:wld="http://www.bea.com/2002/10/weblogicdata"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:complexType name="unidadeoperativa">
        <xs:sequence>
          <xs:element maxOccurs="1" minOccurs="0"
name="codigoUnidadeOperativa" nillable="true" type="xs:decimal"/>
          <xs:element maxOccurs="1" minOccurs="0"
name="siglaUnidadeOperativa" nillable="true" type="xs:string"/>
          <xs:element maxOccurs="1" minOccurs="0"
name="nomeUnidadeOperativa" nillable="true" type="xs:string"/>
          <xs:element maxOccurs="1" minOccurs="0"
name="codigoUnidadeOperativaAGP" nillable="true" type="xs:decimal"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="lstUnidadeOperativa">
        <xs:sequence>
          <xs:element maxOccurs="unbounded" minOccurs="0"
name="UnidadeOperativa" type="unidadeoperativa"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="UnidadesOperativasList"
type="lstUnidadeOperativa"/>
    </xs:schema>
    <xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
targetNamespace="http://br/com/uniriotec/services"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="getUnidadeOperativa">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="p_siglaUnidadeOperativa"
type="xs:string"/>
          </xs:sequence>
```

```
        </xs:complexType>
      </xs:element>
      <xs:element name="getUnidadeOperativaResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element ref="unid:UnidadesOperativasList"
xmlns:unid="http://controls.beans/unidadeoperativa"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </s0:types>
  <s0:message name="getUnidadeOperativa">
    <s0:part element="s1:getUnidadeOperativa" name="parameters"/>
  </s0:message>
  <s0:message name="getUnidadeOperativaResponse">
    <s0:part element="s1:getUnidadeOperativaResponse"
name="parameters"/>
  </s0:message>
  <s0:portType name="UnidadeOperativaService">
    <s0:operation name="getUnidadeOperativa"
parameterOrder="parameters">
      <s0:input message="s1:getUnidadeOperativa"/>
      <s0:output message="s1:getUnidadeOperativaResponse"/>
    </s0:operation>
  </s0:portType>
  <s0:binding name="UnidadeOperativaServiceServiceSoapBinding"
type="s1:UnidadeOperativaService">
    <s2:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <s0:operation name="getUnidadeOperativa">
      <s2:operation soapAction="" style="document"/>
      <s0:input>
        <s2:body parts="parameters" use="literal"/>
      </s0:input>
      <s0:output>
        <s2:body parts="parameters" use="literal"/>
      </s0:output>
    </s0:operation>
  </s0:binding>
  <s0:service name="UnidadeOperativaServiceService">
    <s0:port binding="s1:UnidadeOperativaServiceServiceSoapBinding"
name="UnidadeOperativaServiceSoapPort">
      <s2:address
location="http://localhost:7001/DesenvolvimentoServicos/UnidadeOperativaService"/>
    </s0:port>
 </s0:service>
</s0:definitions>
```

Figura 6 – Codificação do WSDL

3.2 Teste de remoção de atributo do objeto retornado pelo serviço

Este teste visa alterar o serviço, removendo um atributo do POJO retornado, e analisando o seu comportamento em relação à ausência do valor deste atributo.

3.2.1 Removendo atributo do objeto retornado pelo provedor

No teste de remoção de um atributo na classe POJO foram executados os seguintes passos:

1. Remover o atributo e seus respectivos métodos "get" e "set" da classe POJO. Para o teste, foi removido o atributo "codigoUnidadeOperativaAGP". O código removido da codificação original (apresentada na Figura 3) é apresentado na Figura 7.

```
private BigDecimal codigoUnidadeOperativaAGP;

public BigDecimal getCodigoUnidadeOperativaAGP() {
```

```
    return codigoUnidadeOperativaAGP;
}
public void setCodigoUnidadeOperativaAGP(BigDecimal codigoUnidadeOperativaAGP)
{
   this.codigoUnidadeOperativaAGP = codigoUnidadeOperativaAGP;
}
```

Figura 7 – Código removido da classe POJO

2. Remover a "tag" de configuração referente ao atributo removido na classe POJO, no "XML Schema". O código removido da codificação original (apresentada na Figura 4) é apresentado na Figura 8.

```
<xs:element name = "codigoUnidadeOperativaAGP" type="xs:decimal"
nillable="true" minOccurs="0" maxOccurs="1" />
```

Figura 8 – Código removido do XSD

3. Gerar os arquivos "JAR Types" contendo as classes Java correspondentes aos elementos do esquema. Para isso, deve-se clicar com o botão direito do mouse no XML Schema (arquivo XSD) e ir em "Web Services/Generate Types JAR files..." conforme ilustra a Figura 9.

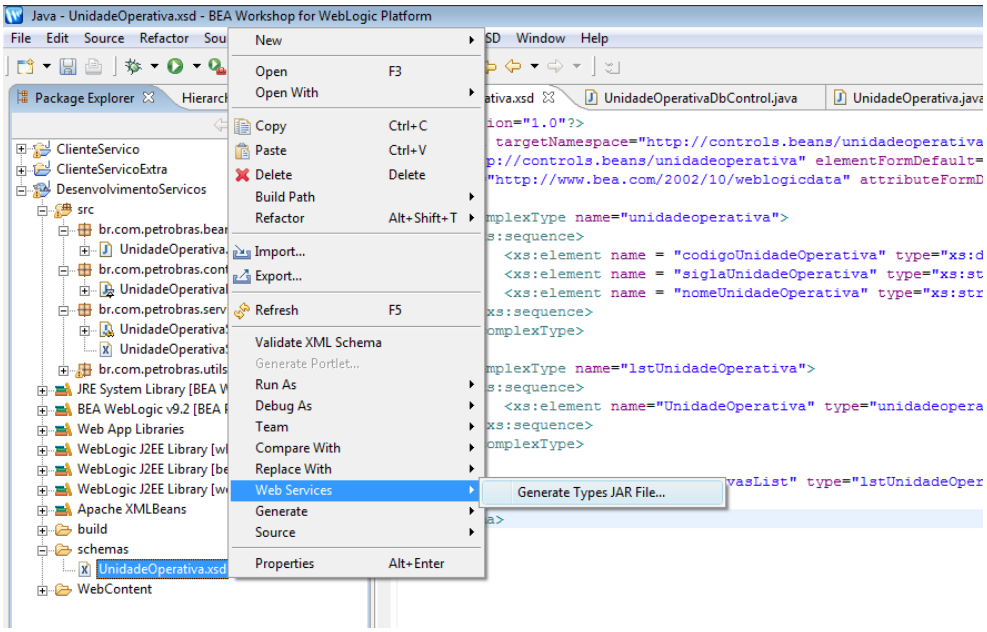

Figura 9 – Geração dos arquivos JAR Types

4. Atualizar a classe de controle do banco de dados (dbControl), retirando a variável da consulta SQL que recebe o dado respectivo ao atributo que está sendo excluído. O código removido da codificação original (apresentada na Figura 5) foi:

   " UNOP_CD_AGP as codigoUnidadeOperativaAGP " +

   Obs.: É necessário apagar a ", " antes do "from" no comando SQL ao apagar a linha citada acima.

5. Atualizar o arquivo WSDL, que contemplará no contrato do serviço as modificações realizadas. Observe que este arquivo não influencia a execução do cliente. É interessante gerá-lo a fim de comparar as diferenças entre o WSDL

considerado pelo cliente e o que descreve o serviço do provedor. Para isso, deve-se clicar com o botão direito no serviço implementado em Java e ir em "Generate WSDL", conforme a Figura 10.

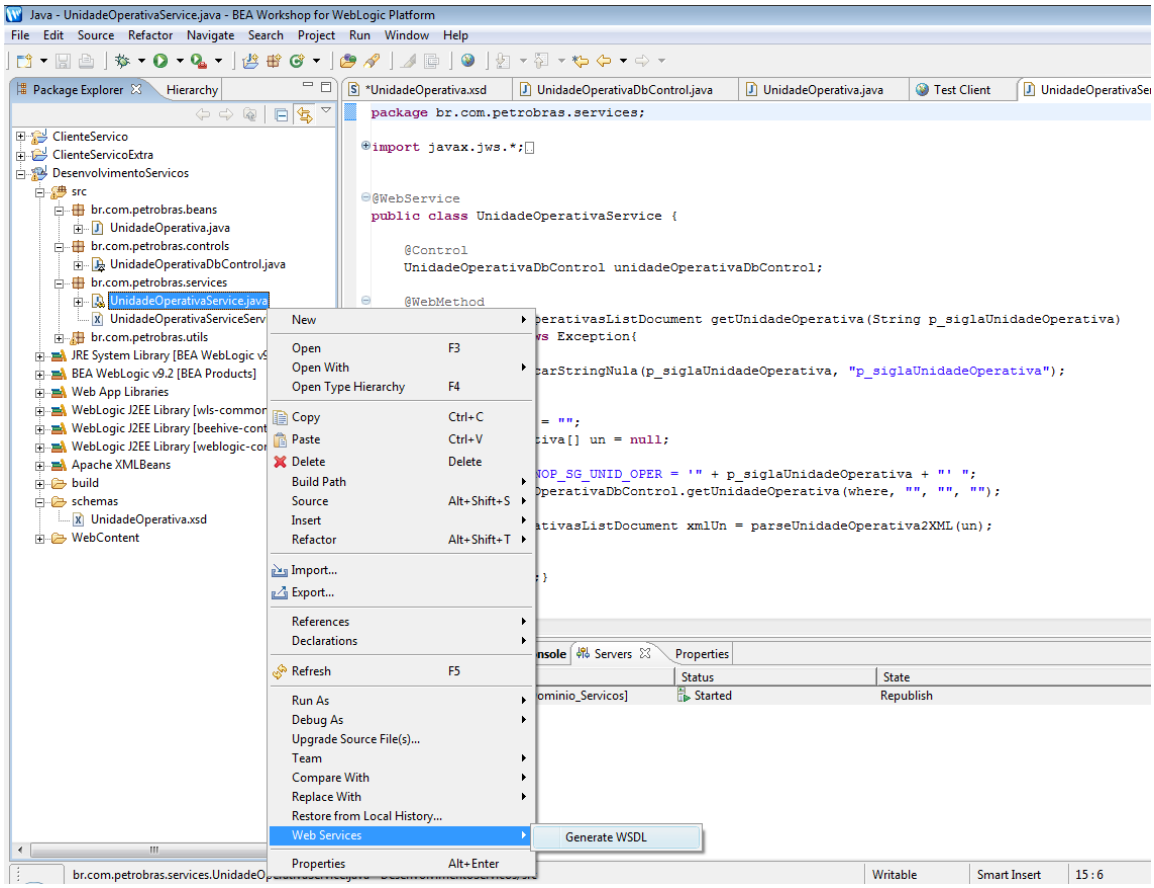

Figura 10 – Geração do arquivo WSDL

Após ter executado todos os passos, o teste pode ser realizado executando o cliente para o serviço.

3.2.2 Invocação pelo cliente do método do serviço que teve o atributo removido

Após a remoção do atributo, executamos o cliente. Nesta execução o cliente gerado a partir do WSDL do serviço original não foi alterado, ou seja, o cliente estava considerando que o método do serviço retornaria todos os atributos, inclusive aquele que foi removido.

A resposta de retorno do serviço (Figura 11), configurada propositalmente neste teste para faltar um atributo esperado pelo cliente ocasionou uma mensagem de erro comunicando a ausência do elemento. Entretanto, o cliente emitiu a resposta normalmente representando o valor ausente como "null".

```
<WS data binding error>Could not find element property 'codigoUnidadeOperativaAGP' on
{http://controls.beans/unidadeoperativa}unidadeoperativa.
  Available elements are:
{http://controls.beans/unidadeoperativa}codigoUnidadeOperativa
{http://controls.beans/unidadeoperativa}siglaUnidadeOperativa
{http://controls.beans/unidadeoperativa}nomeUnidadeOperativa

******************************************************
1
AS
BRA
null
```

```
**************************************************
```

Figura 11 – Resposta do cliente na ausência de um parâmetro

3.3 Teste de adição de atributo no objeto retornado pelo serviço

Este teste visa implementar no servidor, um serviço que responda à solicitação do cliente com um atributo a mais do que ele espera e analisar o seu comportamento em relação a presença de algum parâmetro extra. Ou seja, estamos considerando neste teste que o serviço do provedor foi evoluído e em seu método foi adicionado um atributo, enquanto que o cliente permaneceu apontando para a versão anterior do serviço, a qual não contém o atributo novo.

3.3.1 Adicionando atributo do objeto retornado pelo provedor

No teste de adição de um atributo na classe POJO foram executados os seguintes passos:

1. Adicionar o atributo e seus respectivos métodos "get" e "set" da classe POJO. Para o teste, foi adicionado o atributo "variavelAdicionada". O código adicionado à codificação original (Figura 3) é apresentado na Figura 12.

```
private BigDecimal variavelAdicionada;
public BigDecimal getVariavelAdicionada() {
    return variavelAdicionada;
}
public void setVariavelAdicionada(BigDecimal variavelAdicionada) {
    this.variavelAdicionada = variavelAdicionada;
}
```

Figura 12 – Código adicionado na classe POJO

2. Adicionar a "tag" de configuração referente ao atributo adicionado na classe POJO, no "XML Schema". O código adicionado à codificação original (Figura 4) é apresentado na Figura 13.

```
<xs:element name = "variavelAdicionada" type="xs:decimal"
nillable="true" minOccurs="0" maxOccurs="1" />
```

Figura 13 - Código adicionado ao XSD

3. Gerar os arquivos JAR contendo as classes Java correspondentes aos elementos do esquema (Figura 9).
4. Atualizar a classe de controle do banco de dados (dbControl), adicionando a variável da consulta SQL que recebe o dado respectivo ao atributo que está sendo adicionado. Neste caso, para não necessitar alterar a tabela original no banco de dados, foi configurado um valor fixo apenas representativo. O comando SQL ficou conforme a Figura 14.

```
@JdbcControl.SQL(statement="  select  UNOP_CD_UNID_OPER as codigoUnidadeOperativa, " +
        "          UNOP_SG_UNID_OPER as siglaUnidadeOperativa," +
        "          UNOP_NM_UNID_OPER as nomeUnidadeOperativa," +
        "          UNOP_CD_AGP as codigoUnidadeOperativaAGP, " +
        "          1000 as variavelAdicionada" +
        "  FROM unid_operativa  " +
        "  Where 1=1 " +
        "  {sql: p_where} " +
        "  {sql: p_groupby} " +
        "  {sql: p_having} " +
        "  {sql: p_orderby} ")
```

Figura 14 – Comando SQL após inserção da nova variável

5. Atualizar o arquivo WSDL, que contemplará no contrato do serviço as modificações realizadas (Figura 10). Observe que este arquivo não influencia a execução do cliente. É interessante gerá-lo a fim de comparar as diferenças entre o WSDL considerado pelo cliente e o que descreve o serviço do provedor.

Para testar a inclusão da variável, o serviço foi executado do lado do provedor, obtendo o resultado no formato XML conforme apresentado na Figura 15.

Service Response

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Header />
  <soapenv:Body>
  <m:getUnidadeOperativaResponse xmlns:m="http://br/com/petrobras/services">
   <UnidadesOperativasList xmlns="http://controls.beans/unidadeoperativa">
    <UnidadeOperativa>
     <codigoUnidadeOperativa>1</codigoUnidadeOperativa>
     <siglaUnidadeOperativa>BRA</siglaUnidadeOperativa>
     <nomeUnidadeOperativa>AS</nomeUnidadeOperativa>
     <codigoUnidadeOperativaAGP>1</codigoUnidadeOperativaAGP>
     <variavelAdicionada>1000</variavelAdicionada>
    </UnidadeOperativa>
   </UnidadesOperativasList>
  </m:getUnidadeOperativaResponse>
 </soapenv:Body>
 </soapenv:Envelope>
```

Figura 15 – XML de resposta do serviço

Após ter executado todos os passos, o teste pode ser realizado executando o cliente para o serviço.

3.3.2 Invocação pelo cliente do método do serviço

Após a adição do atributo, executamos o cliente. Nesta execução o cliente gerado a partir do WSDL do serviço original não foi alterado, ou seja, o cliente estava considerando que o método do serviço retornaria apenas os atributos anteriores, sem considerar o atributo que foi adicionado.

A resposta de retorno do serviço, configurada propositalmente neste teste para ter um atributo a mais do que o esperado pelo cliente, ocasionou uma mensagem de erro comunicando a ausência de um método "get" para o elemento considerado como extra. Entretanto, o cliente emitiu a resposta normalmente com todos os valores esperados (Figura 16).

```
<WS data binding error>Could not find getter for property 'VariavelAdicionada' on
beans.controls.unidadeoprativa.Unidadeoperativa
******************************************************
1
AS
BRA
```

```
**************************************************
```

Figura 16 - Resposta do cliente na presença de um parâmetro extra

3.3.3 Conclusão

A atualização do serviço para inclusão ou exclusão de novos atributos, não ocasiona erros durante a troca de mensagens entre o servidor e o cliente quando está sendo utilizado XMLBeans. Caso a mensagem do serviço possua mais informações do que o esperado pelo cliente, não afetará os procedimentos realizados por ele, uma vez que as informações de seu interesse estão presentes e serão lidas. Caso a mensagem do serviço possua menos informações, o cliente automaticamente preenche os atributos com valores nulos, caso não seja programado algum valor padrão para o dado atributo.

Isso demonstra que um mesmo serviço poderá suprir informações para diferentes clientes que possuam necessidade de um subconjunto de informações dentro de um conjunto disponibilizado no serviço. Obviamente o tráfego de informações extra não é o ideal, entretanto, para pequenas variações entre clientes, que são supridos pelo mesmo subconjunto de informações, diferindo apenas de alguns atributos para satisfazer seus interesses específicos, torna-se desnecessário desenvolver novo serviço para atender as estas demandas separadamente e o cliente não precisa ser alterado para considerar a evolução do serviço.

Por outro lado, o cliente deve estar ciente de que a evolução do serviço do provedor pode acarretar a ele receber menos informações. Neste caso, é importante que o cliente valide os dados recebidos a fim de não ocorrer uma falha pelo fato de um dado estar com valor nulo quando deveria estar preenchido.

3.4 Teste de inclusão de método no serviço

O objetivo deste teste é verificar se ocorre algum erro no cliente ao incluir novo método em um serviço do provedor, sem atualizar o cliente.

3.4.1 Inserção do novo método

O método *getUnidadesOperativas* foi inserido na classe que representa o serviço (UnidadeOperativaService.java). O código após inserção do novo método é apresentado na Figura 17.

```
package br.com.petrobras.services;

import javax.jws.*;

@WebService
public class UnidadeOperativaService {

    @Control
    UnidadeOperativaDbControl unidadeOperativaDbControl;

    public UnidadesOperativasListDocument getUnidadeOperativa(String p_siglaUnidadeOperativa)

    public UnidadesOperativasListDocument getUnidadesOperativas(String[] p_siglasUnidadeOperativa)

      private UnidadesOperativasListDocument parseUnidadeOperativa2XML( UnidadeOperativa[] un ) throws Exception {
}
```

Figura 17 – Classe que representa o serviço

Ao executar o serviço com o novo método no servidor, não ocorreram erros. O antigo e o novo método foram oferecidos para execução, conforme apresentado na Figura 18.

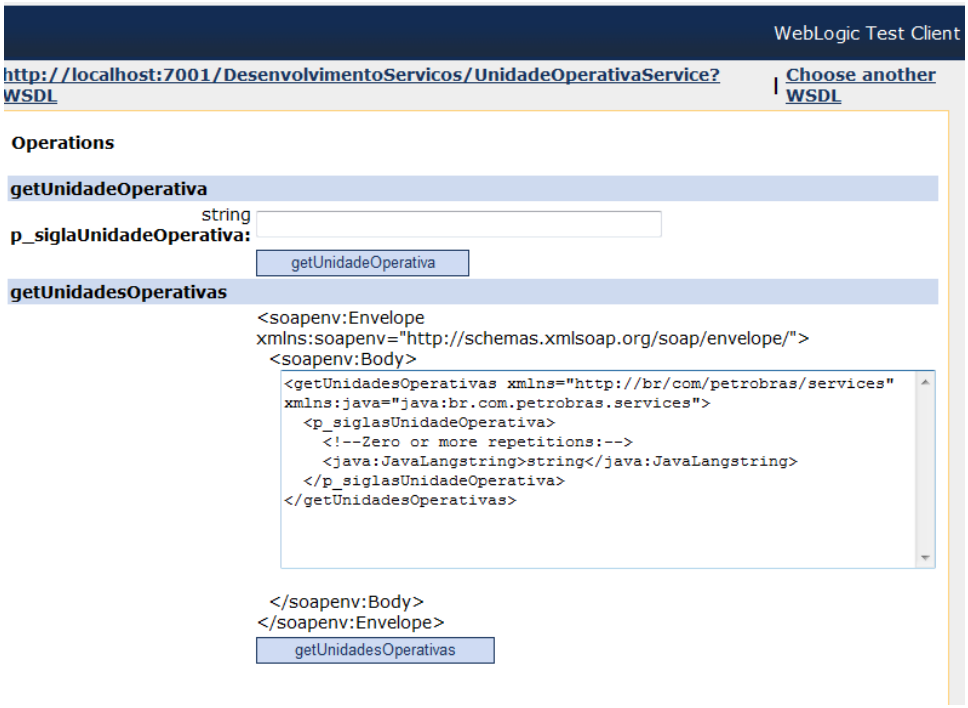

Figura 18 – Serviços ofertados pelo provedor

Ao testar o novo método, o resultado foi conforme o esperado (Figura 19).

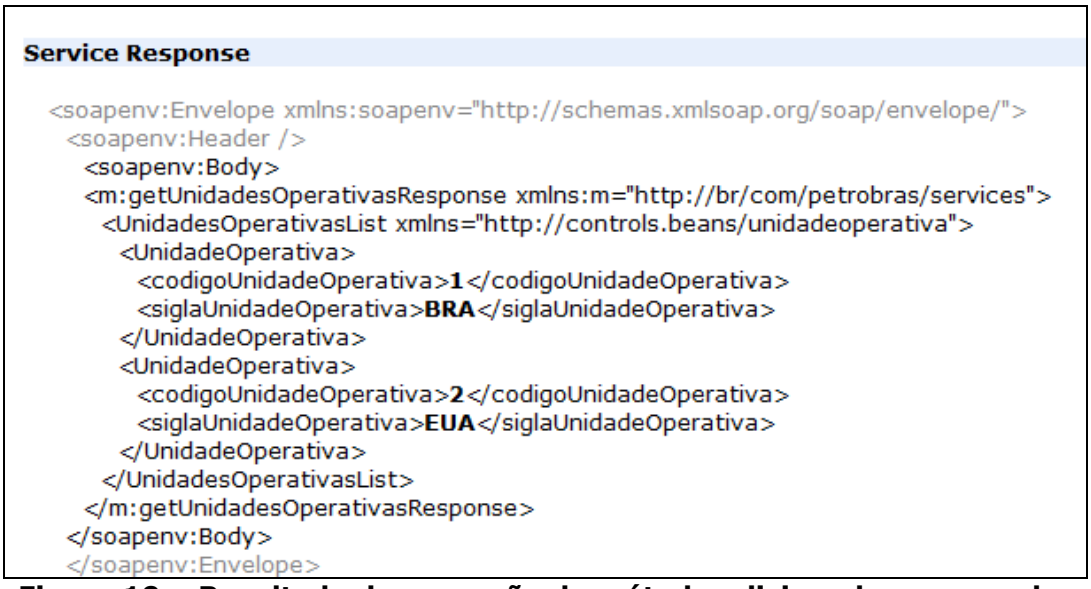

Figura 19 – Resultado da execução do método adicionado ao provedor

3.4.2 Invocação pelo cliente do método do serviço

Ao executar o cliente invocando o método antigo com o serviço do provedor disponibilizando um novo método, não ocorreram erros. O resultado obtido foi conforme o esperado (Figura 20). Observe que o cliente possui duas operações no portType UnidadeOperativaService (Figura 21) e o cliente está apontando para um portType com apenas uma operação (Figura 22). No entanto, isto não ocasiona problemas na invocação do serviço.

Figura 20 – Resultado da execução do serviço pelo cliente

```
<s0:portType name="UnidadeOperativaService">
    <s0:operation name="getUnidadesOperativas" parameterOrder="parameters">
      <s0:input message="s1:getUnidadesOperativas"/>
      <s0:output message="s1:getUnidadesOperativasResponse"/>
    </s0:operation>
    <s0:operation name="getUnidadeOperativa" parameterOrder="parameters">
      <s0:input message="s1:getUnidadeOperativa"/>
      <s0:output message="s1:getUnidadeOperativaResponse"/>
    </s0:operation>
  </s0:portType>
```

Figura 21 – Configuração do PortType do provedor

```
<s0:portType name="UnidadeOperativaService">
    <s0:operation name="getUnidadeOperativa" parameterOrder="parameters">
      <s0:input message="s1:getUnidadeOperativa"/>
      <s0:output message="s1:getUnidadeOperativaResponse"/>
    </s0:operation>
  </s0:portType>
```

Figura 22 – Configuração de PortType do servidor considerado pelo cliente

3.4.3 Conclusão

A inserção de novos métodos no servidor não produz erro na execução normal do cliente cujo portType do serviço referenciado não foi atualizado. Logicamente, se o cliente tentar invocar o método novo do serviço, para o qual ele não possui descrição, ocorrerá erro.

3.5 Teste de valor default

O objetivo deste teste é avaliar o uso de valores default no XSD do provedor e do cliente.

Inicialmente foi realizado o teste de inclusão de valores default no XSD do provedor. A Figura 23 apresenta a atribuição de valores default para os quatro elementos. Em seguida, foram realizadas as seguintes modificações: a consulta do DBControl que retorna os dados do POJO foi alterada para retornar null para o atributo *nomeUnidadeOperativa* (Figura 24), e foi removido da consulta o retorno do atributo *siglaUnidadeOperativaAGP*.

Era esperado que o valor default fosse atribuído ao atributo tanto na ausência de um dado, caracterizado pelo valor "null", como quando o SQL não retornasse o valor para um atributo. No entanto, ao testar o serviço no provedor (usando o TestClient), foi retornado o XML apresentado na Figura 25, o qual não contém o atributo *nomeUnidadeOperativa e siglaUnidadeOperativaAGP*. Foram realizados vários testes, inclusive com outros parâmetros no XSD, mas nenhum atributo dos elementos foi considerado. Observamos que esta característica está na forma de uso XMLBeans, o qual deve ser melhor estudado para entender o que está ocorrendo.

```
<xs:element name = "codigoUnidadeOperativa" type="xs:decimal"
nillable="false" minOccurs="0" maxOccurs="1" default="9999" />
```

```
<xs:element name = "siglaUnidadeOperativa" type="xs:string"
nillable="false" minOccurs="0" maxOccurs="1" default="SiglaPadrao" />


<xs:element name = "nomeUnidadeOperativa" type="xs:string"
nillable="true" minOccurs="0" maxOccurs="1" default="NomePadrao"/>


<xs:element name = "siglaUnidadeOperativaAGP" type="xs:string"
nillable="true" minOccurs="0" maxOccurs="1" default="SiglaAGPPadrao"/>
```

Figura 23 – Configuração de parâmetros no XML Schema

```
      @JdbcControl.SQL(statement="        select        UNOP_CD_UNID_OPER    as
codigoUnidadeOperativa, " +
                    "          UNOP_SG_UNID_OPER as siglaUnidadeOperativa," +
                    "          null as nomeUnidadeOperativa" +
                    "  FROM unid_operativa     " +
                    "  Where 1=1 " +
                    "  {sql: p_where} " +
                    "  {sql: p_groupby} " +
                    "  {sql: p_having} " +
                    "  {sql: p_orderby} ")
```

Figura 24 – Comando SQL do serviço

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header />
    <soapenv:Body>
    <m:getUnidadeOperativaResponse
xmlns:m="http://br/com/uniriotec/services">
      <UnidadesOperativasList
xmlns="http://controls.beans/unidadeoperativa">
        <UnidadeOperativa>
          <codigoUnidadeOperativa>1</codigoUnidadeOperativa>
          <siglaUnidadeOperativa>BRA</siglaUnidadeOperativa>
        </UnidadeOperativa>
      </UnidadesOperativasList>
    </m:getUnidadeOperativaResponse>
  </soapenv:Body>
  </soapenv:Envelope>
```

Figura 25 – XML de resposta do servidor

Pode-se verificar que o servidor ignora os atributos com valor nulo e não inclui no XML. O atributo que não recebeu valor através da consulta também foi ignorado.

Ao executar o serviço através da chamada realizada pelo cliente, foi obtida a resposta apresentada na Figura 26.

```
*******************************************************
1
null
BRA
null
*******************************************************
```

Figura 26 – Resposta emitida pelo cliente ao executar o serviço

O servidor ignorou o valor nulo e não inseriu o valor padrão configurado, emitindo o XML para o cliente sem alguma definição para o parâmetro nulo. Em relação ao atributo sem valor, foi preenchido o campo deste parâmetro também como nulo, o que demonstra que o valor padrão também não foi aplicado neste caso.

Neste teste o cliente também considerou a restrição do valor default para valor nulo recebido do provedor, conforme pode ser visto nas configurações do cliente apresentada na Figura 27.

```
<xs:element          default="9999"          maxOccurs="1"          minOccurs="0"
name="codigoUnidadeOperativa" nillable="false" type="xs:decimal"/>
<xs:element       default="SiglaPadrao"       maxOccurs="1"       minOccurs="0"
name="siglaUnidadeOperativa" nillable="false" type="xs:string"/>
<xs:element       default="NomePadrao"       maxOccurs="1"       minOccurs="0"
name="nomeUnidadeOperativa" nillable="false" type="xs:string"/>
```

Figura 27 – XSD do cliente com valor default

3.5.1 Conclusão

O parâmetro do XML Schema que representa o valor padrão para um atributo não funcionou conforme o esperado. Ao repassar o valor nulo para este parâmetro, o valor padrão configurado não preencheu automaticamente o valor do parâmetro. O uso do XMLBeans deve ser estudado para entender melhor o porquê de isto estar ocorrendo.

3.6 Testes utilizando objeto como parâmetro para método do serviço

Para a realização dos testes apresentados nas seções 3.6.2 e 3.6.3 é criada uma nova classe que será utilizada como parâmetro do método do serviço, ou seja, os parâmetros utilizados para invocar o serviço serão encapsulados em um objeto. O objeto de retorno não foi alterado, continua sendo a classe POJO "UnidadeOperativa".

3.6.1 Encapsulamento de atributos do método em uma classe

A nova classe foi chamada de "ConsultaUnidadeOperativa" (Figura 28), e possui como atributos os campos "nome" e "sigla" e os respectivos métodos *getters* e *setters*. Essa classe foi inserida no projeto do cliente e do servidor.

```
package br.com.petrobras.beans;

public class ConsultaUnidadeOperativa {
    String nome;
    String sigla;
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getSigla() {
        return sigla;
    }
    public void setSigla(String sigla) {
        this.sigla = sigla;
    }
}
```

Figura 28 – Nova classe que será utilizada como parâmetro de chamada do serviço

No servidor, o método "getUnidadeOperativa" foi modificado para receber o objeto como parâmetro. O método realiza a busca pela sigla da unidade operativa, invocando o método "getUnidadeOperativa" do Database Control. O código deste método é apresentado na Figura 29.

```
@WebService
public class UnidadeOperativaService {

    @Control
    UnidadeOperativaDbControl unidadeOperativaDbControl;

    @WebMethod
    public UnidadesOperativasListDocument getUnidadeOperativa(ConsultaUnidadeOperativa consulta)
                throws Exception{

        if (consulta != null)
        {
            String where = "";
            UnidadeOperativa[] un = null;

            where="AND UNOP_SG_UNID_OPER = '" + consulta.getSigla() + "' ";
            un = unidadeOperativaDbControl.getUnidadeOperativa(where, "", "", "");

            UnidadesOperativasListDocument xmlUn = parseUnidadeOperativa2XML(un);

            return xmlUn;
        }

        return null;
    }
```

Figura 29 – Método que representa o serviço recebendo a nova classe como parâmetro

Uma instância de "ConsultaUnidadeOperativa" é criada pelo cliente para armazenar o valor do atributo necessário para execução do serviço (Figura 30) e então o objeto é passado como parâmetro na chamada do serviço. Quando o serviço é executado, ele extrai o valor do atributo do objeto e realiza a consulta. Ao obter o resultado da consulta em objetos POJO da classe "UnidadeOperativa", ele transforma os objetos em XML utilizando o método parseUnidadeOperativa2XML e retorna o arquivo XML resultante para o consumidor do serviço (Figura 29).

```java
package Main;

import beans.controls.unidadeoperativa.Unidadeoperativa;

public class WSCliente {

    public static UnidadeOperativaService webservice;
    public static UnidadeOperativaServiceServiceLocator locator;

    public static void main(String[] args) {
        String wsdlurl =    "http://localhost:7001/Servidor/UnidadeOperativaService?WSDL";

        ConsultaUnidadeOperativa unidadeOperativa = new ConsultaUnidadeOperativa();
        unidadeOperativa.setSigla("BRA");

        try{
            locator = new UnidadeOperativaServiceServiceLocator();
            locator.setUnidadeOperativaServiceSoapPortEndpointAddress(wsdlurl);
            webservice = locator.getUnidadeOperativaServiceSoapPort();

            System.out.println("*****************************************************");

            Unidadeoperativa[] lstUN = webservice.getUnidadeOperativa(unidadeOperativa);

            for (int i = 0; i < lstUN.length; i++) {
                Unidadeoperativa un = lstUN[i];
                System.out.println("Código: " + un.getCodigoUnidadeOperativa());
                System.out.println("Nome: " + un.getNomeUnidadeOperativa());
                System.out.println("Sigla: " + un.getSiglaUnidadeOperativa());
                System.out.println("Indicador: " + un.getIndicadorAtiva());
                System.out.println("Código AGP: " + un.getCodigoUnidadeOperativaAGP());
                System.out.println("*****************************************************");
            }

        } catch (Exception ex){
            ex.printStackTrace();
        }
    }
}
```

Figura 30 – Instanciação da classe "ConsultaUnidadeOperativa"

Ao executar o serviço, foi obtida a seguinte resposta (Figura 31):

```
*****************************************************
Código: 1
Nome: AS
Sigla: BRA
Indicador: PT
Código AGP: 1
*****************************************************
```

Figura 31 – Resultado da execução do serviço

Como visto, a resposta foi retornada conforme o esperado.

Os testes a seguir executam os mesmos passos dos testes apresentados nas seções 3.2 e 3.3 , correspondentes à remoção e adição de atributos sendo que agora na classe POJO que o serviço recebe como parâmetro, ao invés de ser na classe dos objetos retornados pelo método do serviço.

3.6.2 Teste de remoção de atributo no POJO que encapsula os parâmetros do serviço

Neste teste, foi removido o atributo "nome" e seus métodos "get" e "set" da classe "ConsultaUnidadeOperativa", no servidor. O cliente não sofreu nenhuma alteração,

mantendo toda a implementação como se o servidor não tivesse sido alterado. Esse teste simula uma evolução do método no servidor, sem alteração do cliente.

Ao executar o serviço no cliente, foi obtido o resultado apresentado na Figura 32. A mensagem "WS data binding error>" foi exibida no console no momento da criação, no cliente da instância que irá invocar o método do serviço, ou seja, ao criar o *locator* para serviço. No entanto, apesar da exibição desta mensagem de erro, o serviço foi invocado e a resposta à consulta foi retornada normalmente.

```
<WS data binding error>Could not find element property 'Nome' on
{java:br.com.uniriotec.beans}ConsultaUnidadeOperativa.
  Available elements are:
{java:br.com.uniriotec.beans}Sigla

******************************************************
Código: 1
Nome: AS
Sigla: BRA
Indicador: PT
Código AGP: 1
******************************************************
```

Figura 32 - Resultado da chamada, por cliente desenvolvido no Workshop, de método do serviço com um atributo a menos na classe do servidor

Portanto, se um atributo de um objeto POJO for removido de um serviço e o cliente estiver utilizando XMLBeans para invocar o serviço, não ocorrerá erro de execução no cliente e o cliente receberá um aviso de que o serviço do provedor foi evoluído.

3.6.3 Teste de inclusão de atributo no POJO que encapsula os parâmetros do serviço

Ao contrário do teste apresentado na seção anterior, este teste corresponde à, inclusão de um atributo no POJO que encapsula os atributos do método do serviço. Neste caso, foi incluído o atributo "novoAtributo" e seus métodos *getters* e *setters* na classe "ConsultaUnidadeOperativa", no servidor. O cliente não sofreu nenhuma alteração, mantendo toda a implementação como se o servidor não tivesse sido alterado. Esse teste simula uma evolução do método no servidor, sem alteração do cliente.

Ao executar o cliente, foi obtido o resultado apresentado na Figura 33. O resultado ocorreu conforme o esperado, o que demonstra que a inclusão de variáveis na classe que encapsula parâmetros do serviço não causa nenhum impacto na invocação do serviço pelo cliente. Por outro lado, nenhuma informação foi retornada informando que o serviço provido evoluiu.

```
******************************************************
Código: 1
Nome: AS
Sigla: BRA
Indicador: PT
Código AGP: 1
******************************************************
```

Figura 33 – Resultado da chamada, por cliente desenvolvido no Workshop, de método do serviço com um atributo a mais na classe do servidor

Portanto, se for adicionado um atributo ao POJO e o cliente estiver utilizando XMLBeans para invocar o serviço, não ocorrerá erro de execução no cliente e o cliente não receberá um aviso de que o serviço do provedor foi evoluído.

3.7 Problema encontrado na realização dos testes utilizando o Workshop for WebLogic Platform – Inteiro nulo

Ao executar o teste do cliente no servidor com a ferramenta TestCase do Workshop for WebLogic Platform, a ferramenta exibe a interface apresentada na Figura 34. É possível verificar os valores representativos (que a ferramenta insere automaticamente) dados aos tipos de cada variável no local onde se deve preencher o valor correto para a variável. Por exemplo: "string" para variáveis do tipo "String" e "1000.00" para variáveis do tipo "BigDecimal".

Figura 34 – Interface do serviço oferecida pela ferramenta

Alguns dos valores dos parâmetros do serviço (variáveis da mensagem) podem ser opcionais ao serviço, podendo ser apagadas do XML de requisição do serviço, como apresentado na Figura 35.

Figura 35 – XML com valores opcionais apagados

Então, ao excluir os valores inseridos pela ferramenta para as variáveis opcionais, esperou-se que a ferramenta interpretasse automaticamente valores nulos para estes campos, entretanto, ocorre um erro ao excluir o valor das variáveis do tipo "BigDecimal", como apresentado na Figura 36. Esse erro é gerado por um "bug" conhecido como TrimTrailingZero.

Service Response

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header />
    <soapenv:Body>
    <soapenv:Fault>
      <faultcode>soapenv:Server</faultcode>
      <faultstring>String index out of range: -1</faultstring>
```

Figura 36 – Erro ao solicitar o serviço sem valor atribuído para variáveis do tipo "BigDecimal"

4 Testes realizados com cliente desenvolvido no Eclipse

Os testes realizados nas seções anteriores utilizaram clientes desenvolvidos na ferramenta Workshop for WebLogic Platform, da BEA. Os testes realizados nesta seção são os mesmos e utilizam as mesmas classes, porém os clientes foram desenvolvidos na ferramenta Eclipse, versão GANYMEDE (http://www.eclipse.org/ganymede/), utilizando o Axis para criar automaticamente arquivos do cliente a partir do WSDL, necessitando somente desenvolver a classe principal para configurar a chamada do serviço.

4.1 Desenvolvimento do cliente a partir do Eclipse

Os passos necessários para o desenvolvimento do cliente no Eclipse são:

1. Criar novo "Dynamic Web Project" em "File/New/ Dynamic Web Project (Figura 37).

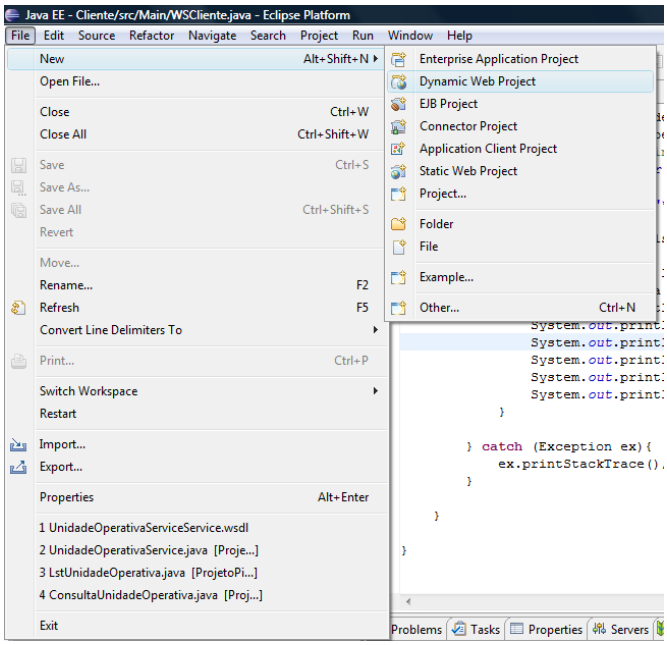

Figura 37 – Criando no projeto no Eclipse

2. Inserir o nome do projeto e clicar em "Finish" (Figura 38).

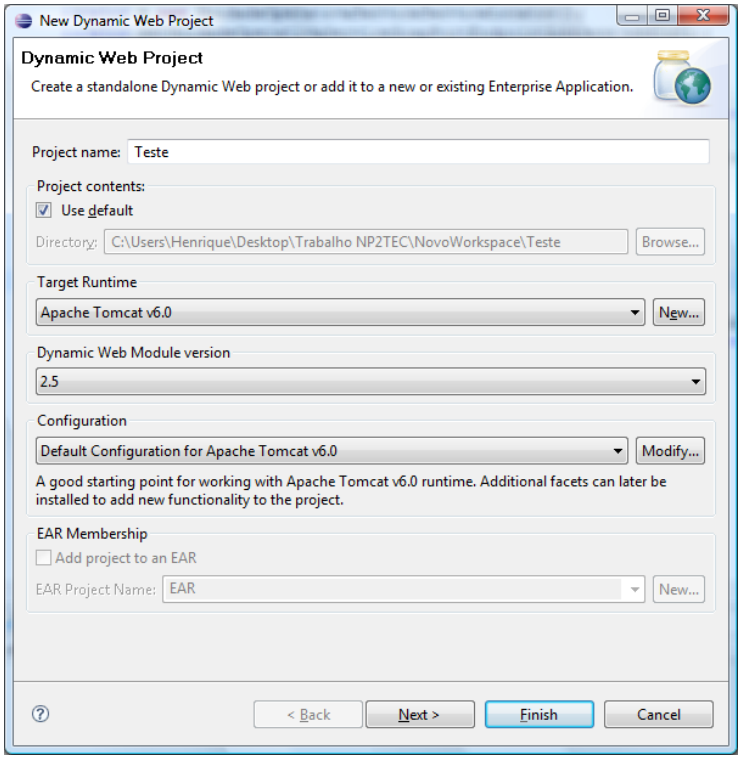

Figura 38 – Finalizando a criação do novo projeto no Eclipse

3. Copiar o arquivo WSDL para dentro da pasta WebContent (Figura 39).

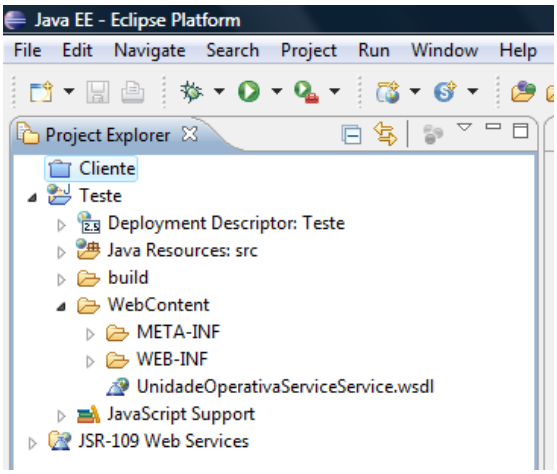

Figura 39 – Copiando o arquivo WSDL para a pasta WebContent

4. Criar o cliente a partir do WSDL, clicando com o botão direto no arquivo e ir em "Web Services/Generate client" (Figura 40).

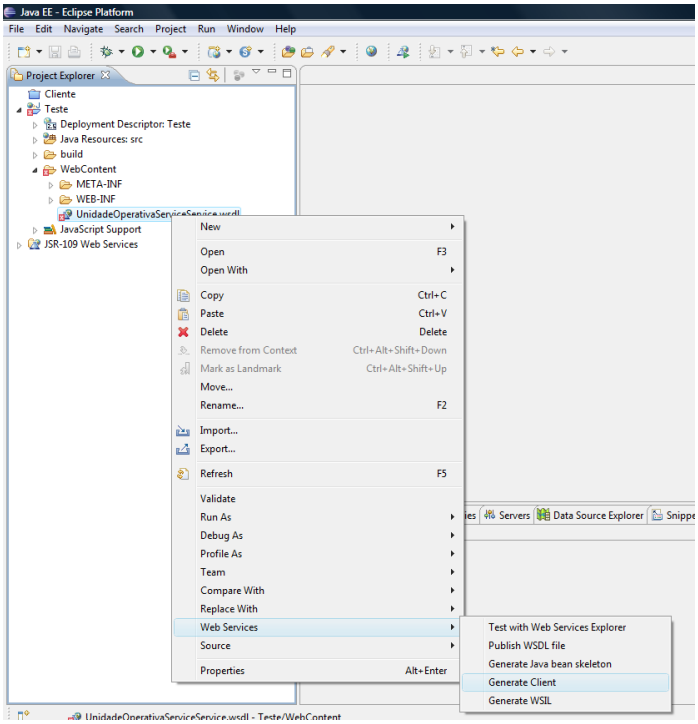

Figura 40 – Criando codificação do cliente a partir do WSDL no Eclipse

5. Na próxima janela, clicar em "Finish" (Figura 41). Talvez seja necessário atualizar a pasta do projeto para que seja visualizado todos os arquivos. Para isso, pressione F5, após selecionar o projeto no Project Explorer.

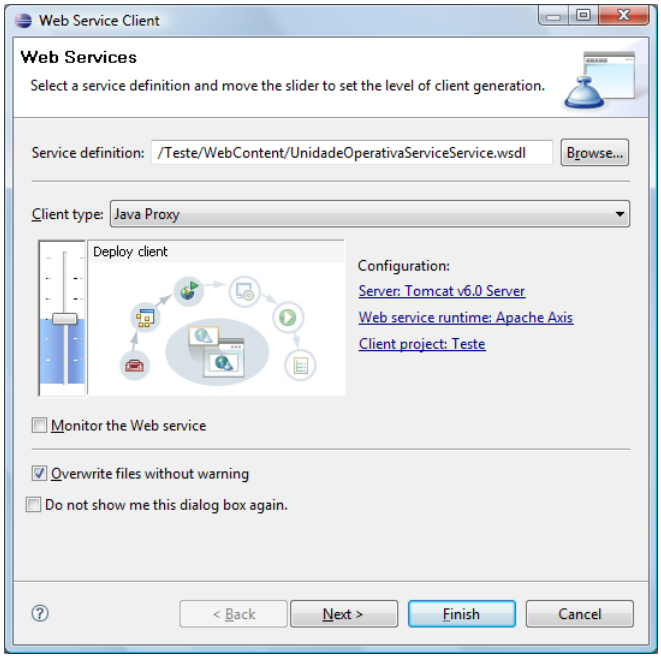

Figura 41 – Finalizando a configuração para o criação da codificação do cliente

6. Implementar a classe principal do cliente, onde deverá possuir a chamada do serviço e o tratamento dos dados de retorno, dentro da pasta "src. A figura abaixo ilustra um exemplo desta classe (Figura 42):

```
package Main;

import beans.controls.unidadeoperativa.Unidadeoperativa;
import beans.uniriotec.com.br.ConsultaUnidadeOperativa;
import br.com.uniriotec.services.UnidadeOperativaService;
import br.com.uniriotec.services.UnidadeOperativaServiceServiceLocator;

public class WSCliente {

	public static UnidadeOperativaService webservice;
	public static UnidadeOperativaServiceServiceLocator locator;

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		String wsdlurl =
	"http://localhost:7001/Servidor/UnidadeOperativaService?WSDL";

		ConsultaUnidadeOperativa unidadeOperativa = new
                                        ConsultaUnidadeOperativa();
		unidadeOperativa.setSigla("BRA");

		try{
			locator = new UnidadeOperativaServiceServiceLocator();
	locator.setUnidadeOperativaServiceSoapPortEndpointAddress(wsdlurl);
			webservice =
                                locator.getUnidadeOperativaServiceSoapPort();
			System.out.println("**********************");

			Unidadeoperativa[] lstUN =
                                webservice.getUnidadeOperativa(unidadeOperativa);

			for (int i = 0; i < lstUN.length; i++) {
				Unidadeoperativa un = lstUN[i];
				System.out.println("Código: " +
                                        un.getCodigoUnidadeOperativa());
				System.out.println("Nome: " +
                                        un.getNomeUnidadeOperativa());
				System.out.println("Sigla: " +
                                        un.getSiglaUnidadeOperativa());
				System.out.println("Indicador: " +
                                        un.getIndicadorAtiva());
				System.out.println("Código AGP: " +
                                        un.getCodigoUnidadeOperativaAGP());
				System.out.println("**********************");

			}

		} catch (Exception ex){
			ex.printStackTrace();
		}

	}

}
```

Figura 42 – Exemplo de classe de chamada do serviço e tratamento dos dados de retorno

7. Para executar o cliente, clique com botão direto na classe “WSClient” e vá em “Run As/Java Application. O resultado será mostrado no console da ferramenta (Figura 43).

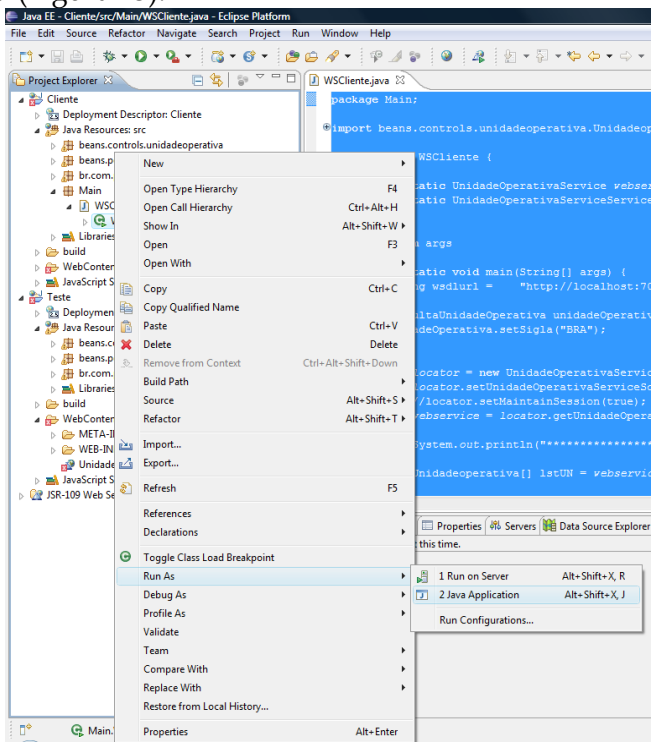

Figura 43 – Execução do cliente no Eclipse

4.2 Execução do cliente desenvolvido no Eclipse

A execução do cliente desenvolvido no Eclipse retornou o resultado da invocação do método, mas também retornou uma mensagem (*WARNING*) dizendo que algumas classes solicitadas não puderam ser encontradas. (Figura 44).

```
26/09/2009 00:51:12 org.apache.axis.utils.JavaUtils
isAttachmentSupported
WARNING: Unable to find required classes (javax.activation.DataHandler
and javax.mail.internet.MimeMultipart). Attachment support is
disabled.
**********************************************************
Código: 1
Nome: AS
Sigla: BRA
Indicador: PT
Código AGP: 1

**********************************************************
```

Figura 44 – Resultado de execução do cliente desenvolvido no Eclipse.

Essa mensagem ocorre devido a ausência do arquivo “mail.jar”, que deve ser importado para o *classpath* do projeto. Este arquivo pode ser baixado no site oficial da Sun (http://java.sun.com/products/javamail/downloads/index.html).

Após a importação deste arquivo, o cliente foi novamente executado, retornando a mesma resposta, porém, sem a mensagem (Figura 45). A seguir, são apresentado os testes realizados no Eclipse considerando a importação deste arquivo.

```
*****************************************************
Código: 1
Nome: AS
Sigla: BRA
Indicador: PT
Código AGP: 1
*****************************************************
```

Figura 45 – Resultado do cliente desenvolvido no Eclipse, após importação do arquivo "mail.jar"

4.3 Teste de remoção de atributo no POJO que encapsula os parâmetros do serviço

Neste teste remove-se um atributo da classe que é parâmetro para o método do servidor, a fim de simular uma atualização de remoção de atributo em uma classe em produção. Este teste é semelhante ao teste apresentado na seção 3.6.2 .

O resultado da execução do serviço, depois da remoção do atributo na classe de consulta no servidor, foi igual ao resultado da execução do cliente em suas configurações normais (Figura 45). No entanto, não foi exibida a mensagem de erro "WS data binding error" em relação à ausência do atributo, como ocorreu na execução do cliente desenvolvido no Workshop for WebLogic Platform (Figura 46).

```
*****************************************************
Código: 1
Nome: AS
Sigla: BRA
Indicador: PT
Código AGP: 1
*****************************************************
```

Figura 46 - Resultado da chamada, por cliente desenvolvido no Eclipse, de método do serviço com um atributo a menos na classe do servidor

4.4 Teste de inclusão de atributo no POJO que encapsula os parâmetros do serviço

Neste teste insere-se um atributo da classe que é parâmetro para o método do servidor, a fim de simular uma inclusão de atributo em uma classe em produção. Este teste é semelhante ao teste apresentado na seção 3.6.3 .

O resultado da chamada do serviço indica que a ferramenta identificou a presença do atributo extra e levantou exceção (*SEVERE: Exception:org.xml.sax.SAXException: Invalid element in beans.controls.unidadeoperativa.Unidadeoperativa – novoAtributo*), caracterizando a falta do atributo no POJO retornado pelo serviço e terminando o programa sem retornar resposta, conforme pode ser visto na Figura 47. A exceção foi levantada quando o método do serviço foi invocado, na linha "Unidadeoperativa[] lstUN = webservice.getUnidadeOperativa(unidadeOperativa)".

```
*****************************************************
15/10/2009 00:36:11 org.apache.axis.client.Call invoke
SEVERE: Exception:
```

```
org.xml.sax.SAXException: Invalid element in
beans.controls.unidadeoperativa.Unidadeoperativa - novoAtributo
      at
org.apache.axis.encoding.ser.BeanDeserializer.onStartChild(BeanDeseria
lizer.java:258)
      at
org.apache.axis.encoding.DeserializationContext.startElement(Deseriali
zationContext.java:1035)
      at
org.apache.axis.message.SAX2EventRecorder.replay(SAX2EventRecorder.jav
a:165)
      at
org.apache.axis.message.MessageElement.publishToHandler(MessageElement
.java:1141)
      at
org.apache.axis.message.RPCElement.deserialize(RPCElement.java:236)
      at
org.apache.axis.message.RPCElement.getParams(RPCElement.java:384)
      at org.apache.axis.client.Call.invoke(Call.java:2467)
      at org.apache.axis.client.Call.invoke(Call.java:2366)
      at org.apache.axis.client.Call.invoke(Call.java:1812)
      at
br.com.uniriotec.services.UnidadeOperativaServiceServiceSoapBindingStu
b.getUnidadeOperativa(UnidadeOperativaServiceServiceSoapBindingStub.ja
va:175)
      at Main.WSCliente.main(WSCliente.java:30)
AxisFault
 faultCode:
{http://schemas.xmlsoap.org/soap/envelope/}Server.userException
 faultSubcode:
 faultString: org.xml.sax.SAXException: Invalid element in
beans.controls.unidadeoperativa.Unidadeoperativa - novoAtributo
 faultActor:
 faultNode:
 faultDetail:
      {http://xml.apache.org/axis/}stackTrace:org.xml.sax.SAXException
: Invalid element in beans.controls.unidadeoperativa.Unidadeoperativa
- novoAtributo
      at
org.apache.axis.encoding.ser.BeanDeserializer.onStartChild(BeanDeseria
lizer.java:258)
      at
org.apache.axis.encoding.DeserializationContext.startElement(Deseriali
zationContext.java:1035)
      at
org.apache.axis.message.SAX2EventRecorder.replay(SAX2EventRecorder.jav
a:165)
      at
org.apache.axis.message.MessageElement.publishToHandler(MessageElement
.java:1141)
      at
org.apache.axis.message.RPCElement.deserialize(RPCElement.java:236)
      at
org.apache.axis.message.RPCElement.getParams(RPCElement.java:384)
      at org.apache.axis.client.Call.invoke(Call.java:2467)
      at org.apache.axis.client.Call.invoke(Call.java:2366)
      at org.apache.axis.client.Call.invoke(Call.java:1812)
      at
br.com.uniriotec.services.UnidadeOperativaServiceServiceSoapBindingStu
b.getUnidadeOperativa(UnidadeOperativaServiceServiceSoapBindingStub.ja
va:175)
```

```
      at Main.WSCliente.main(WSCliente.java:30)

      {http://xml.apache.org/axis/}hostname:MegaNote

org.xml.sax.SAXException: Invalid element in
beans.controls.unidadeoperativa.Unidadeoperativa - novoAtributo
      at org.apache.axis.AxisFault.makeFault(AxisFault.java:101)
      at org.apache.axis.client.Call.invoke(Call.java:2470)
      at org.apache.axis.client.Call.invoke(Call.java:2366)
      at org.apache.axis.client.Call.invoke(Call.java:1812)
      at
br.com.uniriotec.services.UnidadeOperativaServiceServiceSoapBindingStu
b.getUnidadeOperativa(UnidadeOperativaServiceServiceSoapBindingStub.ja
va:175)
      at Main.WSCliente.main(WSCliente.java:30)
Caused by: org.xml.sax.SAXException: Invalid element in
beans.controls.unidadeoperativa.Unidadeoperativa - novoAtributo
      at
org.apache.axis.encoding.ser.BeanDeserializer.onStartChild(BeanDeseria
lizer.java:258)
      at
org.apache.axis.encoding.DeserializationContext.startElement(Deseriali
zationContext.java:1035)
      at
org.apache.axis.message.SAX2EventRecorder.replay(SAX2EventRecorder.jav
a:165)
      at
org.apache.axis.message.MessageElement.publishToHandler(MessageElement
.java:1141)
      at
org.apache.axis.message.RPCElement.deserialize(RPCElement.java:236)
      at
org.apache.axis.message.RPCElement.getParams(RPCElement.java:384)
      at org.apache.axis.client.Call.invoke(Call.java:2467)
      ... 4 more
```

Figura 47 - Resultado da chamada, por cliente desenvolvido no Eclipse, de método do serviço com um atributo a mais na classe do servidor

4.5 Teste de remoção de atributo no POJO que armazena os dados retornados pelo servidor

Neste teste, remove-se um atributo da classe POJO retornada pelo serviço do servidor, a fim de simular uma remoção de atributo de classe em produção. Este teste é semelhante ao teste apresentado na seção 3.2 .

O resultado da execução do cliente não apresentou erros relevantes ao receber o objeto da classe POJO com um atributo a menos. A ferramenta apenas apresentou o valor "null" para o atributo ausente (Figura 48).

```
******************************************************
Código: 1
Nome: AS
Sigla: BRA
Indicador: PT
Código AGP: null
******************************************************
```

Figura 48 – Resultado da execução do cliente no Eclipse, com um atributo a menos no POJO do servidor

4.6 Teste de inclusão de atributo no POJO que armazena os dados retornados pelo do servidor

Neste teste, inclui-se um atributo da classe POJO retornado pelo serviço do servidor, a fim de simular uma adição de atributo em classe em produção. Este teste é semelhante ao teste apresentado na seção 3.3 .

O resultado da chamada do serviço indica que a ferramenta identificou a presença do atributo extra e levantou exceção (*Exception - Invalid element in beans.controls.unidadeoperativa.Unidadeoperativa*), caracterizando a falta do atributo no POJO retornado pelo serviço e terminando o programa sem retornar resposta, conforme pode ser visto na Figura 49. Esta excessão ocorre no momento da chamada do serviço, na seguinte linha de comando "Unidadeoperativa[] lstUN = webservice.getUnidadeOperativa("BRA");"

```
*****************************************************
12/10/2009 02:33:14 org.apache.axis.client.Call invoke
SEVERE: Exception:
org.xml.sax.SAXException: Invalid element in
beans.controls.unidadeoperativa.Unidadeoperativa - novoAtributo
      at
org.apache.axis.encoding.ser.BeanDeserializer.onStartChild(BeanDeseria
lizer.java:258)
      at
org.apache.axis.encoding.DeserializationContext.startElement(Deseriali
zationContext.java:1035)
      at
org.apache.axis.message.SAX2EventRecorder.replay(SAX2EventRecorder.jav
a:165)
      at
org.apache.axis.message.MessageElement.publishToHandler(MessageElement
.java:1141)
      at
org.apache.axis.message.RPCElement.deserialize(RPCElement.java:236)
      at
org.apache.axis.message.RPCElement.getParams(RPCElement.java:384)
      at org.apache.axis.client.Call.invoke(Call.java:2467)
      at org.apache.axis.client.Call.invoke(Call.java:2366)
      at org.apache.axis.client.Call.invoke(Call.java:1812)
      at
br.com.uniriotec.services.UnidadeOperativaServiceServiceSoapBindingStu
b.getUnidadeOperativa(UnidadeOperativaServiceServiceSoapBindingStub.ja
va:168)
      at Main.WSCliente.main(WSCliente.java:26)
AxisFault
 faultCode:
{http://schemas.xmlsoap.org/soap/envelope/}Server.userException
 faultSubcode:
 faultString: org.xml.sax.SAXException: Invalid element in
beans.controls.unidadeoperativa.Unidadeoperativa - novoAtributo
 faultActor:
 faultNode:
 faultDetail:
      {http://xml.apache.org/axis/}stackTrace:org.xml.sax.SAXException
: Invalid element in beans.controls.unidadeoperativa.Unidadeoperativa
- novoAtributo
      at
org.apache.axis.encoding.ser.BeanDeserializer.onStartChild(BeanDeseria
lizer.java:258)
```

```
	at
org.apache.axis.encoding.DeserializationContext.startElement(Deseriali
zationContext.java:1035)
	at
org.apache.axis.message.SAX2EventRecorder.replay(SAX2EventRecorder.jav
a:165)
	at
org.apache.axis.message.MessageElement.publishToHandler(MessageElement
.java:1141)
	at
org.apache.axis.message.RPCElement.deserialize(RPCElement.java:236)
	at
org.apache.axis.message.RPCElement.getParams(RPCElement.java:384)
	at org.apache.axis.client.Call.invoke(Call.java:2467)
	at org.apache.axis.client.Call.invoke(Call.java:2366)
	at org.apache.axis.client.Call.invoke(Call.java:1812)
	at
br.com.uniriotec.services.UnidadeOperativaServiceServiceSoapBindingStu
b.getUnidadeOperativa(UnidadeOperativaServiceServiceSoapBindingStub.ja
va:168)
	at Main.WSCliente.main(WSCliente.java:26)

	{http://xml.apache.org/axis/}hostname:MegaNote

org.xml.sax.SAXException: Invalid element in
beans.controls.unidadeoperativa.Unidadeoperativa - novoAtributo
	at org.apache.axis.AxisFault.makeFault(AxisFault.java:101)
	at org.apache.axis.client.Call.invoke(Call.java:2470)
	at org.apache.axis.client.Call.invoke(Call.java:2366)
	at org.apache.axis.client.Call.invoke(Call.java:1812)
	at
br.com.uniriotec.services.UnidadeOperativaServiceServiceSoapBindingStu
b.getUnidadeOperativa(UnidadeOperativaServiceServiceSoapBindingStub.ja
va:168)
	at Main.WSCliente.main(WSCliente.java:26)
Caused by: org.xml.sax.SAXException: Invalid element in
beans.controls.unidadeoperativa.Unidadeoperativa - novoAtributo
	at
org.apache.axis.encoding.ser.BeanDeserializer.onStartChild(BeanDeseria
lizer.java:258)
	at
org.apache.axis.encoding.DeserializationContext.startElement(Deseriali
zationContext.java:1035)
	at
org.apache.axis.message.SAX2EventRecorder.replay(SAX2EventRecorder.jav
a:165)
	at
org.apache.axis.message.MessageElement.publishToHandler(MessageElement
.java:1141)
	at
org.apache.axis.message.RPCElement.deserialize(RPCElement.java:236)
	at
org.apache.axis.message.RPCElement.getParams(RPCElement.java:384)
	at org.apache.axis.client.Call.invoke(Call.java:2467)
	... 4 more
```

Figura 49 – Resultado da chamada do serviço no Eclipse, com o POJO configurado com um atributo a mais em relação ao esperado

4.7 Conclusão

A implementação do cliente no Eclipse é relativamente simples. A maioria dos testes realizados ocorreram sem erros, sendo retornado a mensagem (*warning*) : "...org.apache.axis.utils.JavaUtils isAttachmentSupported - WARNING: Unable to find required classes (javax.activation.DataHandler and javax.mail.internet.MimeMultipart). Attachment support is disabled." a qual foi solucionada com a importação do arquivo mail.jar para o *Classpath*. Ocorreu erro apenas no caso em que foi adicionado um atributo na classe do POJO retornado pelo serviço, sem atualizar o cliente.

5 Testes realizados com cliente desenvolvido no .Net

Nesta seção, novamente os testes serão realizados de acordo com os anteriores, entretanto, utilizando o framework .Net da Microsoft, contido no pacote Visual Studio 2005 (http://www.microsoft.com/visualstudio/en-us/default.mspx), foi utilizado para a geração do cliente. A linguagem utilizada será C#, que é umas das linguagens oferecidas pelo Visual Studio.

5.1 Desenvolvimento do cliente utilizando framework .Net

Os passos seguidos para o desenvolvimento do cliente são apresentados a seguir:

1. Criar novo projeto em "File/New/ Project..." (Figura 50).

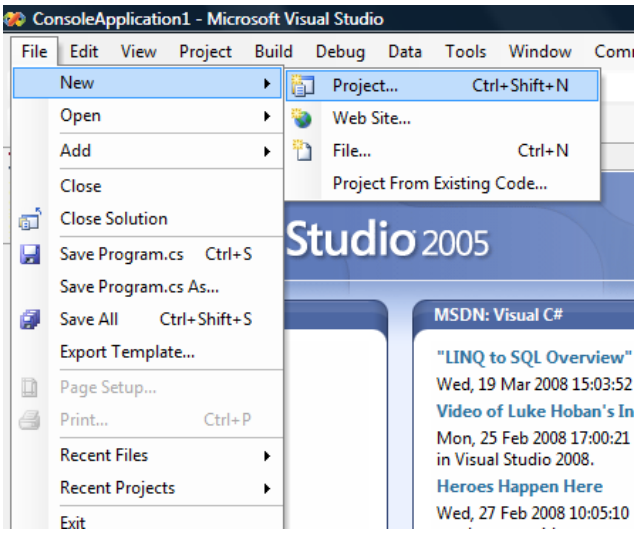

Figura 50 – Criação do projeto no Visual Studio

2. Preencher o nome do projeto. Escolha um tipo de projeto apresentado nos *templates* (Visual Studio installed templates) para utilizar no projeto e clique em "OK". Para a execução dos testes, foi escolhido o *template* "Console Application" (Figura 51).

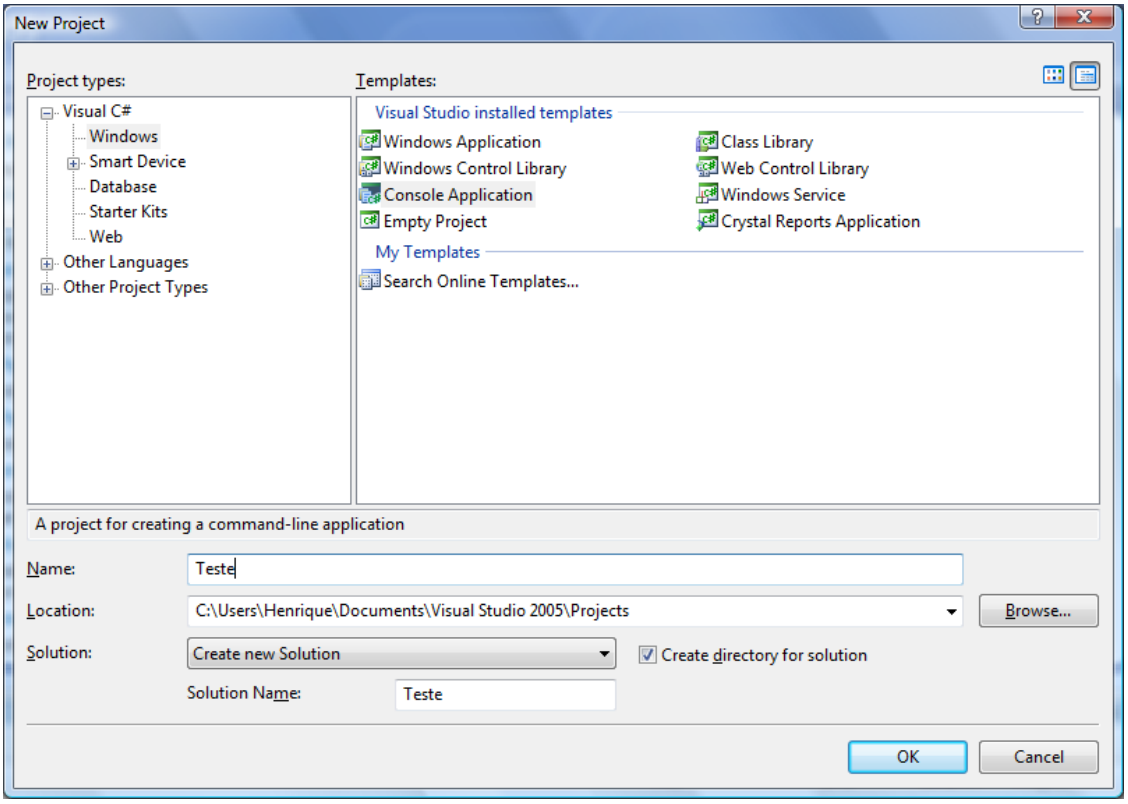

Figura 51 – Escolha do tipo de projeto que será utilizado no desenvolvimento do cliente

3. Em "Solution Explorer", clique com o botão direito em "References" e clique em "Add new web reference..." (Figura 52)

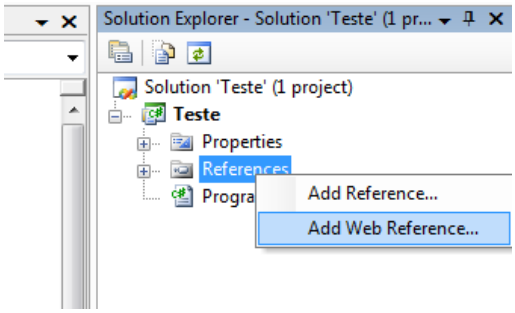

Figura 52 – Adicionando uma referência para o WSDL do serviço no servidor

4. Na próxima janela, em URL, insira o endereço do WSDL publicado e clique em "Go" (Figura 53). Após o reconhecimento do serviço, clique em "Add Reference" (Figura 54). Em "Web reference name" é possível definir um nome para referenciar o WSDL. Para que a ferramenta reconheça o WSDL, ele deverá estar disponível através do servidor executando. Observe que para estes testes, o endereço do WSDL é http://localhost:7001/Servidor/UnidadeOperativaService?WSDL).

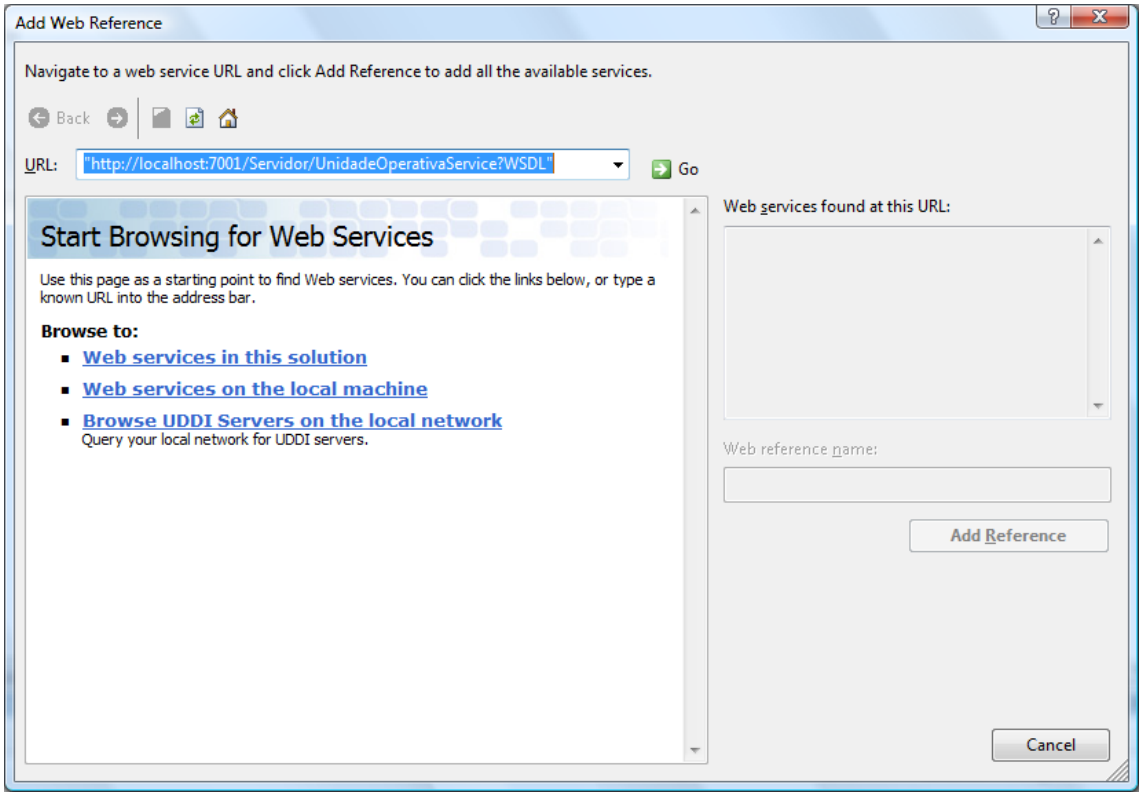

Figura 53 – Inserindo a URL do WSDL que encontra-se no servidor

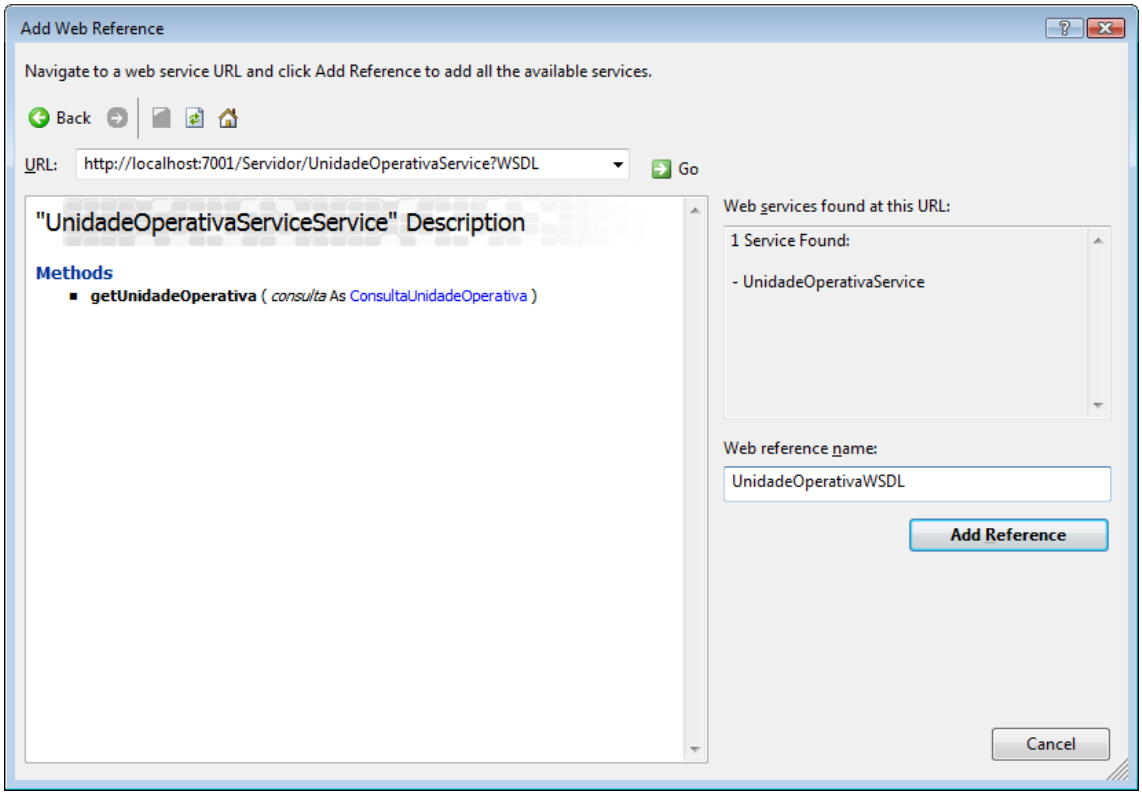

Figura 54 – Reconhecimento do WSDL pela ferramenta

5. Após o passo anterior, o cliente será automaticamente criado. Então deve ser implementada a classe principal do cliente, onde deverá possuir a chamada do serviço e o tratamento dos dados de retorno, no arquivo "program.cs". A Figura 55 ilustra um exemplo de implementação desta classe.

```
using System;
using System.Collections.Generic;
using System.Text;
using Teste.UnidadeOperativaWSDL; //Importação da "Web Reference"
UnidadeOperativaWSDL

namespace Teste
{
    class Program
    {
        static void Main(string[] args)
        {
            UnidadeOperativaServiceService svc = new
                                    UnidadeOperativaServiceService();
            ConsultaUnidadeOperativa con = new
                                    ConsultaUnidadeOperativa();
            con.Sigla = "BRA";
            unidadeoperativa[] uns = svc.getUnidadeOperativa(con);

            foreach (unidadeoperativa un in uns)
            {
                Console.WriteLine("Nome:"+ un.nomeUnidadeOperativa);
                Console.WriteLine("Indicador" + un.indicadorAtiva);
                Console.WriteLine("Sigla" + un.siglaUnidadeOperativa);
                Console.WriteLine("================================");
            }
            Console.ReadLine();
        }
    }
}
```

Figura 55 - Exemplo de classe de chamada do serviço e tratamento dos dados de retorno

6. Para executar o cliente, pressione CTRL + F5.

5.2 Execução do cliente desenvolvido no framework .Net

A execução do cliente desenvolvido utilizando o framework .Net retornou o resultado esperado, conforme pode ser visto na Figura 56.

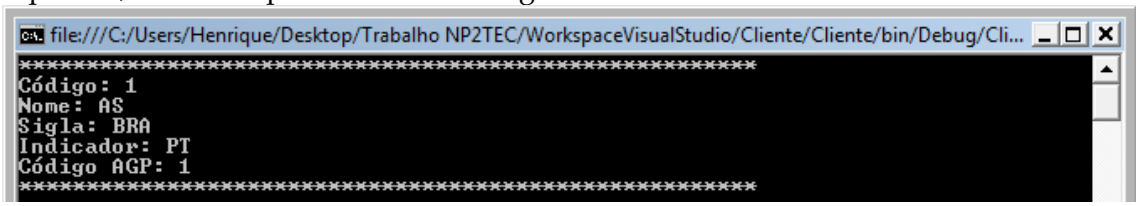

Figura 56 – Resultado da execução do cliente utilizando o framework .Net

5.3 Teste de remoção de atributo na classe que encapsula os parâmetros do serviço

Neste teste remove-se um atributo da classe que é parâmetro para o método do servidor, a fim de simular uma atualização de remoção de atributo em uma classe em produção. Este teste é semelhante ao teste apresentado na seção 3.6.2 .

O resultado da execução do serviço (Figura 57), depois da remoção do atributo na classe de consulta no servidor, foi igual ao resultado da execução do cliente em suas configurações normais, ou seja, a remoção do atributo não afetou no resultado do serviço. No entanto, não foi exibida a mensagem de erro "WS data binding error" em relação à ausência do atributo, como ocorreu na execução do cliente desenvolvido no Workshop for WebLogic Platform (Figura 32).

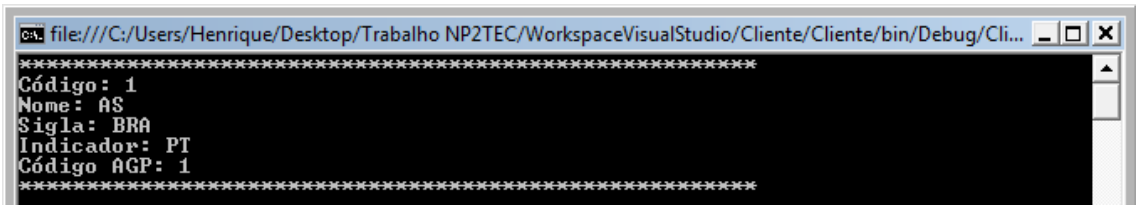

Figura 57 – Resultado da execução do cliente desenvolvido no .Net após remoção de atributo na classe emitida como parâmetro na chamada do serviço

5.4 Teste de inclusão de atributo na classe que encapsula os parâmetros do serviço

Neste teste insere-se um atributo da classe que é parâmetro para o método do servidor, a fim de simular uma inclusão de atributo em uma classe em produção. Este teste é semelhante ao teste apresentado na seção 3.6.3 .

O resultado da execução do serviço (Figura 58), depois da inserção do atributo na classe de consulta no servidor, foi igual ao resultado da execução do cliente em suas configurações normais, ou seja, a inserção do atributo não afetou no resultado do serviço.

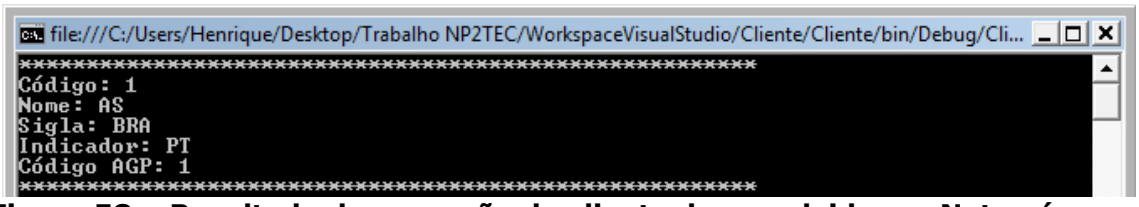

Figura 58 – Resultado da execução do cliente desenvolvido no .Net após inclusão de atributo na classe emitida como parâmetro na chamada do serviço

5.5 Teste de remoção de atributo no POJO que armazena os dados retornados pelo servidor

Neste teste, remove-se um atributo da classe POJO retornada pelo serviço do servidor, a fim de simular uma remoção de atributo de classe em produção. Este teste é semelhante ao teste apresentado na seção 3.2 .

O resultado da execução do serviço (Figura 59), depois da remoção do atributo na classe POJO, no servidor, mostra que foi atribuído o valor nulo ao atributo "codigoUnidadeOperativaAGP", uma vez que o seu respectivo campo de resposta encontra-se vazio.

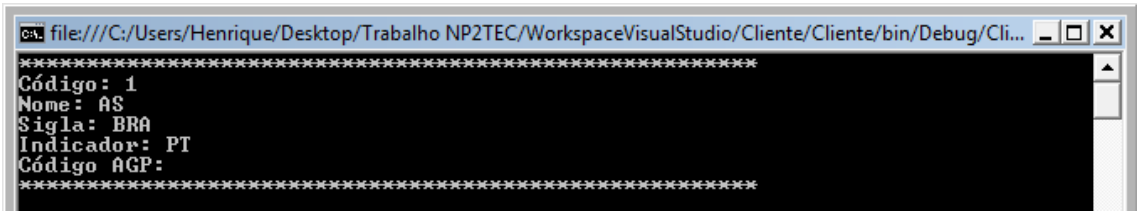

Figura 59 – Resultado da execução do cliente desenvolvido no ,Net após remoção de atributo na classe POJO no servidor

5.6 Teste de inclusão de atributo no POJO que armazena os dados retornados pelo servidor

Neste teste, inclui-se um atributo da classe POJO retornada pelo serviço do servidor, a fim de simular uma remoção de atributo em classe de produção. Este teste é semelhante ao teste apresentado na seção 3.3 .

O resultado da execução do serviço (Figura 60), depois da inserção do atributo na classe POJO no servidor, foi igual ao resultado da execução do cliente quando o servidor encontra-se em suas configurações normais, ou seja, a inserção do atributo não afetou no resultado do serviço, e o atributo novo não foi considerado pelo cliente do serviço.

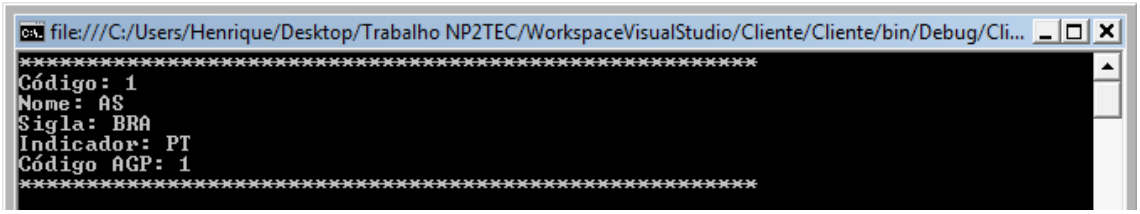

Figura 60 - Resultado da execução do cliente desenvolvido no ,Net após inserção de atributo na classe POJO no servidor

5.7 Conclusão

A implementação do cliente no framework .Net é relativamente simples. Não ocorreu erro em nenhum dos testes realizados. Logo, se o serviço provido tiver a classe retornada ou a classe que encapsula os parâmetros do método alteradas, o cliente não sofrerá nenhum impacto com esta evolução.

6 Conclusões

Os resultados dos testes demonstraram a tolerância das ferramentas utilizadas em relação a atualização assíncrona de atributos de classes que são emitidas pelos serviços.

Em quase todos os casos, a ferramenta emitiu uma mensagem de aviso em relação às alterações que foram identificadas, tais como ausência ou excesso de parâmetros, menos nos clientes desenvolvidos na plataforma .Net, onde não apareceram mensagens em nenhuma de suas respostas. Somente nos testes de inclusão de atributo

no POJO do serviço (ou no POJO retornado ou no POJO passado como parâmetro), realizado com cliente desenvolvido no Eclipse, não se conseguiu obter resposta, e o programa foi finalizado com mensagem de erro.

Em relação às ferramentas utilizadas, o Workshop for WebLogic Platform da BEA apresentou problemas como congelamentos e erros inesperados, enquanto o Eclipse e o .Net não apresentaram nenhuma anomalia durante os testes. Por outro lado, utilizando a nova versão da ferramenta, o Oracle Workshop for Weblogic (versão 10.3), ocorreram menos problemas.

O desenvolvimento dos clientes nas três ferramentas pode ser considerado semelhante, entretanto, Eclipse e Workshop utilizam estritamente a linguagem Java, enquanto o framework .Net pode utilizar qualquer linguagem oferecida em sua *suíte*. Como foi escolhido o C#, observou-se maior complexidade em sua estrutura de classes e código em relação as outras tecnologias.

7 Referências bibliográficas

APACHE XML Project, http://xmlbeans.apache.org/. Acesso em 04 Ago.2009a.

APACHE XML Project, http://xmlbeans.apache.org/docs/2.
0.0/guide/conGettingStartedwithXMLBeans.html. Acesso em 08 de Ago. 2009b.

APACHE XML Project, http://xmlbeans.apache.org/overview.html. Acesso em 08 de Ago. 2009c.

AZEVEDO, L.; SOUSA, H. P.; BAIAO, F.; SANTORO, F. **Desenvolvendo web services no BEA Workshop for WebLogic Platform.** Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT-0014/2009, 2009. Disponível em: http://seer.unirio.br/index.php/monografiasppgi.

HURWITZ, J., BLOOR, R., BAROUDI, C., KAUFMAN, M., Service Oriented Architecture for Dummies, Willey Publishing, Inc.