

# Arquitetura para integração de sistemas utilizando Web Services

Lucas Keller, Guilherme Bertoni Machado

Curso de Análise e Desenvolvimento de Sistemas  
Faculdade de Tecnologia Senac RS (FATEC/RS)  
Porto Alegre – RS – Brasil

contato@lucaskeller.net, gb.machado@sinprors.org.br

**Abstract.** *To integrate systems you can choose different ways of transferring data between them. This work is a synchronization of data between two systems, and this synchronization is provided by an architecture based on Web Services with a structure framework for the abstraction of Web Service clients as well as the persistence of synchronized data. It is used to instance, as a source of data management system of a company, and as the destination of the synchronization of the new company website, which necessarily needs to show different information present within the management system, without having its own manager content. The architecture proposed in this paper aims to provide greater flexibility in implementing the integration between these systems and thus increase the profitability of a project that needs to apply this type of technology.*

**Resumo.** *Para integrar sistemas pode-se optar por diferentes maneiras de transferência de dados entre eles. Neste trabalho é feita uma sincronização de dados entre dois sistemas, e esta sincronização é proporcionada por uma arquitetura baseada em Web Services, com uma estrutura de framework responsável pela abstração de clientes de Web Service, assim como a persistência de dados sincronizados. É utilizado, a fim de exemplo, como fonte de dados um sistema de gestão de uma empresa, e como destino da sincronização o novo website desta empresa, que necessariamente precisa demonstrar diversas informações presentes dentro do sistema de gestão, sem precisar possuir um próprio gerenciador de conteúdo. A arquitetura proposta neste trabalho tem como objetivo prover maior agilidade na implementação da integração entre estes sistemas, e conseqüentemente aumentar a rentabilidade de um projeto que necessite aplicar este tipo de tecnologia.*

## 1. Introdução

A necessidade de interoperabilidade entre sistemas, reaproveitamento de código e demais obstáculos proporcionados pela expansão sistemática das organizações, fomentou o desenvolvimento de algumas tecnologias que permitem a comunicação entre sistemas de informação, entre elas estão: *Web Services*, *CORBA (Common Object Request Broker Architecture)*, *DCOM (Distributed Component Object Model)*, *JavaRMI (Java Remote Method Invocation)*.

Os obstáculos oferecidos pela Internet como *firewalls*, ISPs (*Internet Service Providers*) com configurações diversas, *proxys*, etc. favoreceram o uso de *Web services* como meio mais viável de integração entre sistemas distribuídos neste ambiente.

O principal objetivo deste trabalho é possibilitar um possível aumento na agilidade de implementação de um projeto *WEB*, diminuindo a carga de trabalho sobre os clientes de um *Web Service* através de uma biblioteca desenvolvida neste trabalho, denominada *WSProxy*, para abstração da camada de integração. E também da utilização de um *framework* que foi customizado em função da nova fonte de dados que alimentará o banco de dados do projeto.

Basicamente, o *WSProxy* é uma aplicação com a função de leitura e interpretação de um *Web Service*. E tem a missão de suprir a necessidade da concepção de um cliente para cada serviço.

O *framework* que foi customizado é uma solução própria já utilizada em larga escala em projetos *WEB*, tem separação de camadas de aplicação e outros recursos interessantes, porém somente dava suporte à programação do *CRUD* (*Create, Retrieve, Update and Delete*) através de interface *WEB* própria. Um método para entrada dos dados oriundos da integração foi criado e acrescentado ao *script* de automação (que é responsável pela geração de código automático, espelhando-se em um banco de dados previamente criado), com o intuito de prover a sincronização das informações dos dois sistemas.

Iniciando o detalhamento das tecnologias envolvidas na arquitetura, na seção 2.1 são abordados os conceitos de *Web Services*. Já na seção 2.2 pode-se conferir a fundamentação do protocolo SOAP responsável pela formatação da troca de mensagens entre *Web Services*. Seguindo, o WSDL é apresentado na seção 2.3, este que é responsável pela descrição de *Web Services*, e de fundamental importância para a aplicação desenvolvida nesse trabalho, pois através das informações disponibilizadas por esta tecnologia será possível o descobrimento automatizado das propriedades de cada serviço.

Dentro das seções 3 e 4 a estrutura, fluxo e funcionamento da arquitetura é explorada. Primeiramente, na seção 3, é enfatizada a forma correta do uso do WSDL nesta arquitetura. A seção 4, principal foco do trabalho, contém o detalhamento do lado do cliente, com a explicação completa do funcionamento da *WSProxy* e do *framework* aplicado para persistência de dados.

O tipo e validação escolhido para a arquitetura proposta neste trabalho, os resultados obtidos após a validação, o plano de melhorias e as conclusões sobre a aplicação desenvolvida com o resultado da aplicação desta arquitetura podem ser conferidos na seção 5, 6, 7 e 8, respectivamente, e por fim as referências bibliográficas.

## **2. Conceitos Básicos**

Nesta seção serão apresentados os conceitos básicos das tecnologias aplicadas no trabalho prático, apresentando a fundamentação teórica de *Web Services*, SOAP e WSDL.

## 2.1. Web Services

De acordo com a W3C, *Web Service* é um sistema desenvolvido para permitir a interoperabilidade entre máquinas em uma rede computacional. Possui uma interface descrita em um formato legível por uma máquina (mais especificamente, WSDL). Outros sistemas podem interagir com o *Web Service* através desta descrição usando mensagens SOAP, normalmente transportadas pelo protocolo HTTP com os dados serializados no formato XML (*Extensible Markup Language*) e em conjunto com outros padrões web relacionados (W3CWS, 2009).

Pode-se abstrair o conceito de *Web Services* para o mundo não digital, o comparando ao processo de compra de matéria prima por uma fábrica, por exemplo. A fábrica tem um papel específico, fabricar o produto, porém necessita de alguns insumos para realizar esta tarefa, para isso possui um canal de comunicação com seu fornecedor de matéria prima, que por sua vez tem domínio do processo de aquisição da matéria prima, conservação e distribuição da mesma. A fábrica solicita o tipo, quantidade e orçamento através de um protocolo para o fornecedor, que por sua vez retorna o valor total da compra e as formas de pagamento em resposta ao mesmo protocolo, assim, a empresa decide a forma mais adequada de pagamento e fornece o pagamento, recebendo como retorno a quantidade e qualidade especificada da matéria prima.

Estruturalmente é um serviço disponibilizado na internet, com entradas e saídas de dados descritas através de WSDL (*Web Service Definition Language*), com seu acesso feito através do protocolo SOAP (*Simple Object Access Protocol*) todos utilizando a estrutura XML para organização das informações e também intercâmbio de dados. Pode-se ainda registrar este serviço em algum local centralizado através do protocolo UDDI (*Universal Description Discovery & Integration*), que de acordo com OASIS (2008), fornece uma infra-estrutura que permite fixar definições de descrição (*describing*), descoberta (*discovering*) e integração de serviços *Web*.

## 2.2. SOAP

SOAP é um protocolo destinado à troca de informações em um ambiente distribuído e descentralizado. Usa o XML para definir um framework que disponibiliza maneiras de construção de mensagens que possam trafegar através de diversos protocolos, e foi desenvolvido para funcionar independente de qualquer modelo de programação ou implementação específica (W3CSOAP, 2008).

O modelo atualmente utilizado para comunicação entre aplicações de diferentes arquiteturas utiliza RPC (*Remote procedure call*), ou seja, uma tecnologia que permite um software executar rotinas ou procedimentos em outro equipamento ligado a mesma rede através de protocolos do tipo DCOM e CORBA por exemplo. Em um ambiente WEB este tipo de protocolo representa um problema de compatibilidade e segurança, firewalls e servidores de *proxy* normalmente bloqueiam este tipo de tráfego (RFC5531, 2009).

O SOAP não possui este impeditivo, pois é baseado em uma estrutura de dados que trafega sobre HTTP (*Hypertext Transfer Protocol*), o que proporciona ao protocolo alta disponibilidade e compatibilidade, uma vez que o HTTP é suportado por todos os browsers e servidores *Web*, e está cada vez mais presente em uma infinidade de

dispositivos HTTP, de acordo com a RFC2616 (2009). É um protocolo de nível de aplicação que é utilizado para distribuição de *hipermídia* por sistemas de informação. É usado desde 1990 pela rede mundial de computadores (Internet).

### 2.3. WSDL

De acordo com a W3C, WSDL é um formato XML para descrever *Web Services* como um conjunto de canais de comunicação operando sob mensagens contendo qualquer tipo de informação estruturada. As operações e mensagens são descritas de uma maneira abstrata, e depois vinculadas a um protocolo de rede e formato para assim definirem o funcionamento de um canal de comunicação (W3CWSDL, 2008).

Em outras palavras o WSDL é um protocolo que define as entradas, saídas e funcionalidades de um *Web Service*, através dele saberemos como acessar os métodos ou procedimentos do sistema que está disponibilizando o serviço, assim como que tipo de dados receberemos do mesmo.

### 3. Estrutura e Fluxo do Sistema

A estrutura básica utilizada neste trabalho, conta com dois sistemas heterogêneos, sendo um deles o sistema de gestão de uma empresa e o seu novo *website*, sendo que este consumirá as informações já existentes no sistema de gestão, economizando esforço de trabalho no desenvolvimento de sistemas de gerenciamento de conteúdo e inserção de dados dentro do *website*.

Para compor um cenário perfeito para implementação da sincronização entre o sistema de gestão de uma empresa e o seu novo *website*, necessita-se de uma documentação formal mínima que demonstre as funcionalidades do *website* e o que ele necessita de informações do sistema de gestão. Os dados que serão obtidos através de sincronização precisarão ser necessariamente documentados para a equipe que desenvolve o *website* não inclua no plano de desenvolvimento do CMS (*Content Management System* – Sistema Gerenciador de Conteúdo) do *website* CRUD (*Create, Retrieve, Update and Delete* – Criação, Obtenção, Atualização e Exclusão) para estes dados.

Para que o WSDL possa ser utilizado da maneira esperada, nesta documentação já estão descritos desde os nomes de cada serviço até o tipo de dado contido em cada campo da coleção de dados que o *web service* retornará ao cliente. Esta tipagem precisa ser respeitada, visto que, após a aquisição das informações a aplicação chamará classes para persistência de dados, que ao chegarem ao repositório online podem ser repudiados.

Este cenário não é um impeditivo de uma situação onde os desenvolvedores da solução *WEB* não tenham documentação formal do *Web Service*. Desde que a Empresa X forneça um WSDL completo e válido, contendo todos os tipos de dados e *operations*, a equipe tem perfeitas condições de criar aplicação da arquitetura proposta, porém com eventuais customizações na biblioteca do *framework*, principalmente a respeito dos padrões de nomenclatura de chaves primárias, e nomes das *operations*, visto que a WSPProxy na versão deste trabalho está diretamente ligada aos padrões do *framework* utilizado pelo *website*.

Para demonstração de como um serviço é disponibilizado, estruturado, e igualmente provar como a sua leitura pode ser fácil, levando-se em conta que é descrito através de uma estrutura textual intuitiva, o serviço será quebrado em partes devidamente comentadas nas próximas subsecções.

### 3.1. Complex types

Quando precisamos trafegar via SOAP tipos de dados que não sejam tipos primitivos das linguagens de programação como: inteiro, *string*, *double*, eles precisam ser explícitos no corpo do WSDL. Tipos de dados não primitivos, normalmente são dados que são uma coleção (*Array*) ou que compõem uma, como por exemplo, os tipos “AreaNegocio” e “ColecaoAreaNegocio” visualizados na Figura 1, mais especificamente nos nodos “<xsd:complexType name=“AreaNegocio”>” e “<xsd:complexType name=“ColecaoAreaNegocio”>”. Pode-se observar na leitura simples do XML que o tipo “AreaNegocio” é composto de 4 elementos “AreaNegocioID”, “Nome”, “DescricaoResumida” e “Exclui”, respectivamente. E o tipo de dado “ColecaoAreaNegocio” é uma coleção de dados “AreaNegocio”, previamente declarado. Além do tipo primitivo de cada elemento, pode-se definir outras propriedades que podem ser úteis para a integração, neste caso foi declarado o tamanho de cada elemento de “AreaNegocio” com a propriedade “size” (RICHARDS, 2006).

```

<definitions targetNamespace="urn:Sincronizacao">
- <types>
- <xsd:schema targetNamespace="urn:Sincronizacao">
  <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
  <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/" />
- <xsd:complexType name="AreaNegocio">
- <xsd:all>
  <xsd:element name="AreaNegocioID" type="xsd:string" size="10"/>
  <xsd:element name="Nome" type="xsd:string" size="10"/>
  <xsd:element name="DescricaoResumida" type="xsd:string" size="10"/>
  <xsd:element name="Exclui" type="xsd:string" size="1"/>
</xsd:all>
</xsd:complexType>
- <xsd:complexType name="ColecaoAreaNegocio">
  <xsd:attribute ref="SOAP-ENC:arrayType" wsdl:arrayType="tns:AreaNegocio[]"/>
</xsd:complexType>
</xsd:schema>
</types>
+ <message name="AreaNegocioRequest"></message>
+ <message name="AreaNegocioResponse"></message>
+ <portType name="urn:Sincronizacao:PortType"></portType>
+ <binding name="urn:Sincronizacao:Binding" type="tns:urn:Sincronizacao:PortType"></binding>
+ <service name="urn:Sincronizacao:"></service>
</definitions>

```

Figura 1 – Tipos de dados do serviço

Como neste trabalho não será criado o WSDL, este exemplo é somente ilustrativo e criado através da biblioteca nuSOAP. Esta biblioteca do PHP possui métodos para criação do WSDL e de tipos de dados chamados *complexTypes*, que são justamente os que não fazem parte do universo de dados primitivos das linguagens de programação, já citados no parágrafo anterior (NUSOAP, 2009).

### 3.2. Messages

Seguindo na interpretação do XML, o WSDL precisa mostrar quais são os formatos de mensagem que devem ser parametrizados para o serviço e que tipo de mensagem é retornado para o cliente, estas duas informações ficam contidas nos nodos *message*.

Na Figura 2 pode-se visualizar os nodos *message* de requisição e resposta do serviço “AreaNegocio” expandidos, mais especificamente “<message name=“AreaNegocioRequest”>” e “<message name=“AreaNegocioResponse”>”, respectivamente. Assim, podemos visualmente interpretar que a mensagem de requisição deverá possuir 3 elementos ou partes, “Usuario”, “Senha” e “AreaNegocioID”, respectivamente, todos do tipo *String*. E a resposta obtida ou *Response* do serviço, caso o cliente obtenha sucesso na requisição, será um tipo de dado “ColecaoAreaNegocio”, que foi previamente definido no corpo do WSDL através de *ComplexTypes* (RICHARDS, 2006).

```
- <definitions targetNamespace="urn:Sincronizacao">
+ <types></types>
- <message name="AreaNegocioRequest">
  <part name="Usuario" type="xsd:string"/>
  <part name="Senha" type="xsd:string"/>
  <part name="AreaNegocioID" type="xsd:string"/>
</message>
- <message name="AreaNegocioResponse">
  <part name="return" type="tns:ColecaoAreaNegocio"/>
</message>
+ <portType name="          PortType"></portType>
+ <binding name="          Binding" type="tns:          PortType"></binding>
+ <service name="          "></service>
</definitions>
```

Figura 2 – Parâmetros e tipos de retorno dos serviços

### 3.3. Port Types

Na Figura 3 está a primeira informação necessária à aplicação que fará a interpretação dos dados do serviço, os serviços disponibilizados com suas respectivas ações de entrada e saída de dados, o nodo “*portType*” oferece este tipo de informação.

```

- <definitions targetNamespace="urn:Sincronizacao">
+ <types></types>
+ <message name="AreaNegocioRequest"></message>
+ <message name="AreaNegocioResponse"></message>
- <portType name="PortType">
- <operation name="AreaNegocio">
  <documentation>Sincroniza as areas de negócio do sistema</documentation>
  <input message="tns:AreaNegocioRequest"/>
  <output message="tns:AreaNegocioResponse"/>
</operation>
</portType>
+ <binding name="Binding" type="tns:PortType"></binding>
+ <service name=" "></service>
</definitions>

```

**Figura 3 – Listagem dos serviços disponíveis**

Pode-se notar através da estrutura de um WSDL, que para melhor compreensão visual de sua estrutura a leitura ideal deve ser feita de baixo para cima, pois primeiramente são declarados todos os dados necessários para o funcionamento do serviço como tipos de dados e tipos de mensagem, e após isto temos a declaração das portas do *Web Service*, ou seja, os serviços propriamente ditos, fazendo a chamada dos itens contidos no XML.

Esta estrutura se repetirá para cada um dos serviços que venha a utilizar tipos de dados complexos ou não, o que proporciona o uso de algoritmos para automatização da leitura e compreensão das informações necessárias para um consumo autônomo do mesmo.

#### **4. A Arquitetura**

Nesta seção é descrito como funciona a arquitetura proposta, descrevendo a funcionalidade do serviço e os requisitos da biblioteca que faz a abstração dos clientes de *Web Service*, assim como a estrutura do *framework* utilizado para organização da persistência dos dados sincronizados no *website*.

Partindo da premissa que o serviço já estará implementado conforme a documentação, agora é necessário desenvolver os clientes que realizarão requisições ao *Web Service* para obtenção dos dados, que posteriormente serão persistidos no repositório de dados a ser sincronizado.

Para execução desta tarefa, a arquitetura proposta neste trabalho será aplicada desde a abstração de clientes de *Web Service* através da biblioteca *WSPProxy*, até a aplicação de um *framework*, que possibilita com a padronização de algumas operações básicas, geração de código automático utilizando a ferramenta *MyGeneration* (MYGENERATION, 2009), que gera as classes de regras de negócio e acesso a dados. O diagrama de casos de uso da arquitetura é ilustrado pela figura 4.

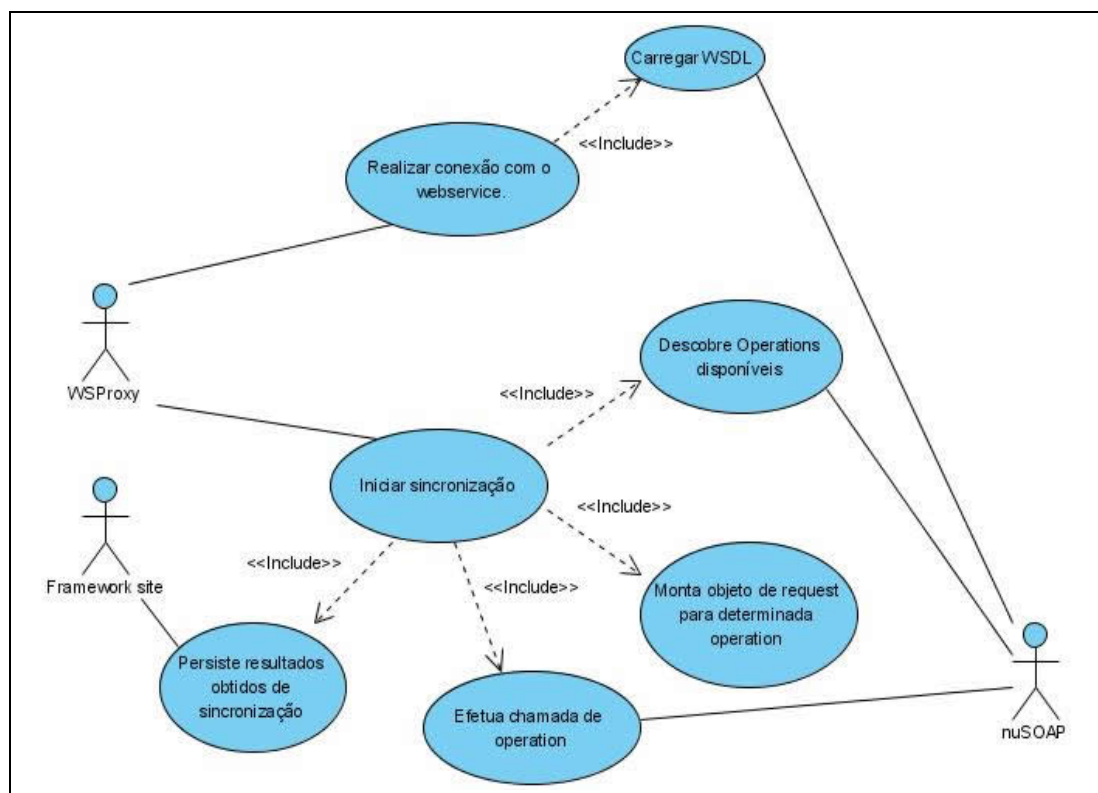


Figura 4. Diagrama de Casos de Uso da Arquitetura

## 4.1. Biblioteca WSPProxy

A biblioteca WSPProxy é uma das principais saídas deste trabalho. Consiste em uma classe, que por sua vez, estende a classe principal da biblioteca NuSOAP, de onde utiliza o processamento do envelope SOAP, leitura e validação do WSDL e diversas outras ações referentes a operações básicas em *Web Services*, deixando a cargo da biblioteca desenvolvida no trabalho a parametrização de *operations*, tratamento de erros e SOAP *faults*, persistência automática de dados de retorno do *Web Service* e o registro (*log*) das operações executadas.

### 4.1.1. Requisitos da biblioteca

A ferramenta efetua a leitura e interpretação de um WSDL, assim como chamada automática de classes para persistência de dados recebidos via sincronização.

### 4.1.2. Percorrendo as principais funcionalidades da biblioteca

Nas subseções subsequentes, o fluxo de execução das funcionalidades básicas da biblioteca WSPProxy é abordado. Desde a criação de uma instância da classe WSPProxy, até a ligação que a mesma tem com o restante da arquitetura, no caso, diretamente com o *framework* utilizado nas persistências de dados.

#### 4.1.2.1. Iniciando uma sincronização

Para dar início a uma sincronização, ou seja, a parametrização de algum tipo de serviço, a fim de que dados sejam carregados para dentro do repositório de dados do *website*, ou

aplicação *WEB*, existe o método chamado “iniciarSincronização” dentro da classe *WSProxy*.

Porém, antes mesmo da chamada do método que iniciará a rotina, dentro do construtor da classe *WSProxy*, algumas ações já são executadas, são elas:

- Criação de um *SOAP Client*, ou seja, uma interface a qual fará as requisições no formato SOAP (responsabilidade da biblioteca *nuSOAP*);
- Carga do WSDL através de um método chamado “*loadWSDL*” que após evocado, habilita uma série de outros recursos para a leitura e interpretação do WSDL, utilizados em alguns momentos dentro da biblioteca *WSProxy*.

Imediatamente após a criação de uma nova instância da classe *WSProxy*, podemos em seguida chamar o método “iniciarSincronização”, que dá início ao processo de sincronização. Pode-se tanto iniciar a sincronização de um único serviço, ou deixar a cargo da biblioteca detectar todos os serviços existentes e automaticamente tentar efetuar a sincronização dos mesmos, pois o WSDL fornece as informações necessárias para a detecção das *operations* e dos tipos de dados necessários para a persistência das informações no banco do cliente.

A detecção automática das *operations*, ou serviços, existentes no *Web Service* é proporcionada pelo método “*getOperations*” (conforme a Figura 5) do objeto “*wSDL*” do *nuSOAP* (classe que a *WSProxy* estende).

```
214 function iniciarSincronizacao($operation=null)
215 {
216     if($this->debugMode) $this->debug->LogSincronizacao->DataHoraInic
217
218     if ($operation !== null)
219     {
220         $this->nusoapCall($operation);
221     }
222     else
223     {
224         $colecacaoOperations = $this->wSDL->getOperations();
225
226         foreach ($colecacaoOperations as $operation => $OperationData)
227         {
228             $this->nusoapCall($operation);
229         }
230     }
231 }
```

**Figura 5 – Método “*getOperations*” utilizado para detecção automática de todas *operations* do *WebService*.**

Já com as *operations* carregadas, a Figura 6 mostra as informações das *operations* disponibilizados com seus respectivos dados de entrada e saída, além de diversas outras informações não relevantes para esta fase.

```

[SincronizarAreaNegocio] => Array
(
    [name] => SincronizarAreaNegocio
    [binding] => ERPBinding
    [endpoint] => http://127.0.0.1/tcc/erp/Solucao/wsdl.php
    [soapAction] => http://localhost/tcc_proxy/erp/Solucao/wsdl.php#areanegocio
    [style] => rpc
    [input] => Array
        (
            [use] => encoded
            [namespace] => http://localhost/tcc_proxy/erp/Solucao/wsdl.php
            [encodingStyle] => http://schemas.xmlsoap.org/soap/encoding/
            [message] => SincronizarAreaNegocioRequest
            [parts] => Array
                (
                    [usuario] => http://www.w3.org/2001/XMLSchema:string
                    [senha] => http://www.w3.org/2001/XMLSchema:string
                    [areanegocioid] => http://www.w3.org/2001/XMLSchema:string
                )
            )
        )
    [output] => Array
        (
            [use] => encoded
            [namespace] => http://localhost/tcc_proxy/erp/Solucao/wsdl.php
            [encodingStyle] => http://schemas.xmlsoap.org/soap/encoding/
            [message] => SincronizarAreaNegocioResponse
            [parts] => Array
                (
                    [retorno] => http://localhost/tcc_proxy/erp/Solucao/wsdl.php:colecacaoareanegocio
                )
            )
        )
    [transport] => http://schemas.xmlsoap.org/soap/http
    [documentation] =>
)

```

**Figura 6 – Operation**

#### 4.1.2.2. Preparar parâmetros de uma *operation*

As *operations* do tipo *request-response*, *one-way*, *solicit/response* de um *Web Service* podem exigir a entrada de dados para ativar alguma ação dentro do serviço, como validação de permissões de acesso, quantidade de elementos que se quer retornar, tipo de elementos, etc.

Através do método “nusoapCall”, que em sua primeira ação efetua a chamada de um outro método “prepararParametrosOperation”, é possível a elaboração de um conjunto de parâmetros para cada *operation* executada. Porém, no exemplo utilizado neste trabalho, será fixado um conjunto de valores para todas as *operations*.

Na Figura 7 podemos observar o método responsável pela montagem da lista de parâmetros de uma *operation*.

```

86     function prepararParametrosOperation($operation)
87     {
88         /**
89          * Switch para servicos one-way ou com parametros especiais
90          */
91
92         $operation = strtolower(str_replace("Sincronizar", "", $operation));
93
94         switch ($operation)
95         {
96             default:
97
98                 $colecaoParametros = array
99                 (
100                     'usuario'      => 'usuario',
101                     'senha'        => 'senha',
102                     $operation.'id' => '0'
103                 );
104
105                 break;
106         }
107
108         return $colecaoParametros;
109     }

```

Figura 7 – Método “prepararParametrosOperation”

#### 4.1.2.3. Execução ou chamada de uma operation

Logo após preparar todos os parâmetros de uma determinada *operation*, resta a ação de executá-la, ou seja, efetuar a requisição via SOAP de uma *operation*. A formatação da mensagem no protocolo SOAP, assim como a sua serialização e requisição de transferência para o servidor web através do protocolo HTTP, fica a cargo da biblioteca nuSOAP, que já implementa todos os protocolos e padrões para este tipo de transação.

Dentro da classe WSPProxy o método “nusoapCall” implementa a chamada da *operation* através do método “call” da classe “nusoap\_client”, enviando através de parametrização o nome da *operation* que se deseja sincronizar e o conjunto de parâmetros previamente organizados.

A Figura 8 demonstra esta etapa de preparação e chamada dentro do método “nusoapCall”.

```

139     function nusoapCall($operation)
140     {
141         /**
142          * Monta o array de parametros da operation
143          */
144         $colecaoParametros = $this->prepararParametrosOperation($operation);
145
146         /**
147          * Chamada da Operation
148          */
149         $resultado = $this->call
150         (
151             $operation,
152             $colecaoParametros
153         );
154     }

```

Figura 8 – Preparação de parâmetros da *operation* e chamada da mesma.

Problemas diversos podem ocorrer neste momento, falhas de conexão de rede, má formatação dos parâmetros da *operation*, acesso negado ao serviço, mensagem SOAP mal formada, etc. Para detectar estes tipos de problemas após a chamada da *operation*, um tratamento para os erros está implementado e são gravados em um objeto contendo informações de *log* de operações da transação, que posteriormente, conforme a opção do usuário pode ser apresentada ou não. A Figura 9 demonstra como estes erros são tratados.

```
155      /**
156       * Trata as soap faults
157       */
158      if ($this->fault)
159      {
160          if($this->debugMode) $this->debug->LogSincronizacao->Mensagem = $resultado["faultstring"];
161      }
162      else
163      {
164          /**
165           * Verifica a existencia de erros
166           */
167          $erro = $this->getError();
168
169          if ($erro)
170          {
171              if($this->debugMode) $this->debug->LogSincronizacao->Mensagem = $erro;
172          }
173          else
174          {
175              /**
176               * Condicional sem erros - inicia a persistência
177              */
```

Figura 9 - Tratamento de erros após a chamada da *operation*

#### 4.1.2.4. Receber as coleções de dados e iniciar persistência dos dados.

Esta funcionalidade da aplicação, concentra as ações que encaminham os dados recolhidos do *Web Service* para persistência. O método responsável por esta operação é chamado “persisteResultado”, o mesmo é evocado logo após não ocorrerem erros depois da execução do método “call” da classe nuSOAP, dentro do método “nusoapCall”, conforme ilustrado na Figura 10.

```
175      /**
176       * Condicional sem erros - inicia a persistência
177       */
178      $colecaoResultado = $resultado;
179
180      if (count($colecaoResultado) > 0)
181      {
182          $this->persisteResultado($colecaoResultado, $operation);
183      }
```

Figura 10 – Chamada do método “persisteResultado” dentro do método “nusoapCall”.

#### 4.1.2.5. Persistência de dados

A coleção de resultados obtidos através do *Web Service* é inserida no banco de dados do cliente através do método “persisteResultado”. Dentro deste método começa a ser utilizada a estrutura do *framework* (descrito na seção 4.2) aplicado na construção do *website*, pois dinamicamente o nome de uma classe de Regras de Negócio (RN) é montada e instanciada, e em seguida, cada um dos resultados é submetido à persistência.

A existência deste método é possível em função de padronização de nomenclatura das classes de RN obtida através da automação de código promovida pela ferramenta *MyGeneration*, onde cada classe de RN e Acesso a Dados (AD) correspondem à uma tabela do banco de dados da aplicação, mais explicações sobre esta automação estão disponíveis na seção 4.2.

A Figura 11 demonstra como a classe RN é dinamicamente incluída no documento e instanciada, para em seguida ser ativada através do método “Sincronizar”, que encaminha os dados à persistência no banco de dados.

```
247  /**
248   * Salvar colecao de dados no website
249   *
250   * @param array $colecacaoResultado
251   * @param string $entidade
252   */
253  function persisteResultado($colecacaoResultado,$operation)
254  {
255      $entidade = strtolower(str_replace("Sincronizar","", $operation));
256
257      if (file_exists($this->caminhoClasse."/RN/" . $entidade . "RN.php"))
258      {
259          /**
260           * Include da classe a ser instanciada dinamicamente
261           */
262          eval("require_once '". $this->caminhoClasse."/RN/" . $entidade . "RN.php'");
263
264          /**
265           * Instancia a classe RN dinamicamente
266           */
267          eval("\$" . $entidade . "RN = new " . $entidade . "RN();");
268
269          /**
270           * Transforma a colecao de retornos do web service em um array de objetos
271           */
272          $colecacaoResultado = $this->transformarArrayParaObjeto($colecacaoResultado);
273
274          if ($this->debugMode) $this->debug->LogSincronizacao->colecacaoRegistros = array();
275
276          foreach ($colecacaoResultado as $chave => $resultado)
277          {
278              eval("\$retornoSalvar = " . "\"" . $entidade . "RN->Sincronizar(\$resultado);");
279          }
280      }
281  }
```

**Figura 11 – Destaque para as principais funcionalidades do método “persisteResultado”, responsável pela persistência dos dados obtidos através do WebService.**

A figura 12 apresenta o diagrama de sequência da arquitetura, que representa o diagrama de caso de uso da arquitetura proposta.

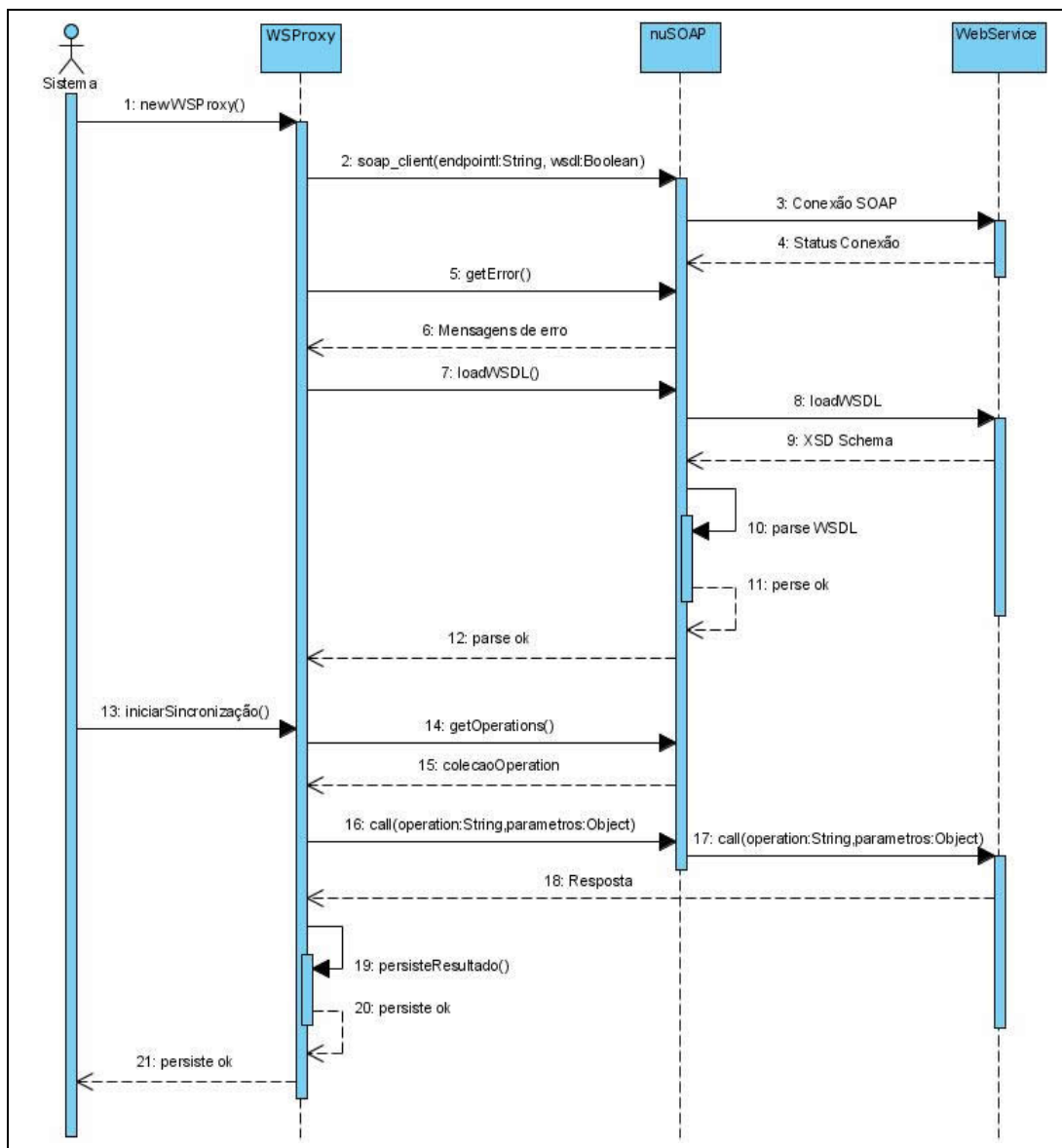


Figura 12. Diagrama de Sequência da Arquitetura

#### 4.2. Framework utilizado no website.

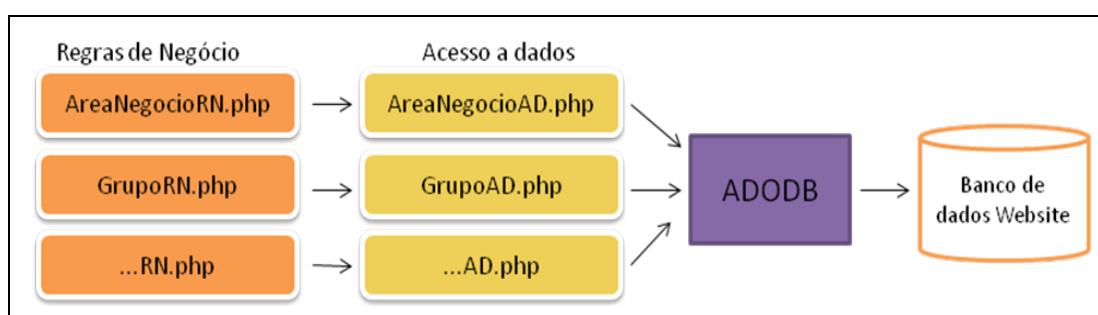
O *framework* utilizado na solução possui um padrão de programação para a camada de acesso a dados e controle das regras de negócio que giram em torno das operações básicas em um banco de dados: inserção, leitura, atualização, exclusão. Praticamente cada tabela do banco de dados que precise deste tipo de operação possui duas classes, uma para controle de regras de negócio (RN) e outra para o acesso a dados (AD).

Exemplificando este modelo, pode-se citar uma tabela que possua os dados de usuários do sistema. Algumas regras de negócio para esta tabela são bem conhecidas, como evitar a inserção de usuários de mesmo *login*, obter os dados dos relacionamentos desta tabela como o perfil do usuário, excluir em cascata os dados deste usuário em tabelas relacionadas, etc.

Este tipo de controle deve ser implementado sempre na camada de regra de negócio, após o tratamento necessário aos dados, instancia-se uma classe de acesso a dados desta RN, onde conterão chamadas de *stored procedures*, ou *queries SQL* para persistência de dados.

Assim sendo, no momento em que o programador necessitar de uma listagem das Áreas de Negócio do Cliente X, simplesmente instanciará uma classe de regra de negócio da respectiva tabela e chamar o método “Pesquisar”, que obterá uma coleção de dados desta tabela. Isso ocorre também na inserção de dados, onde o programador chamará o método “Salvar” passando por parâmetro os dados da tabela e recebendo uma mensagem do sucesso da persistência.

A Figura 13 ilustra o fluxo de dados desta arquitetura.



**Figura 13 – Fluxo de dados do framework do website.**

A criação destas classes manualmente seria inviável para estruturas de dados muito grandes sem a utilização de um gerador de códigos, para esta função a ferramenta *MyGeneration* é utilizada.

Extremamente flexível e escrita em .NET, o *MyGeneration* gera código a partir de *templates* escritos em linguagens C#, VB.NET, JScript e VBScript. Permite uma conexão direta com o banco e dados do *website*, o que proporciona a programação do template o acesso a um mapeamento dos campos das tabelas existentes, o que é essencial para que sejam criados os arquivos de CRUD, que no caso do *framework* em questão são as classes de AD e RN (MYGENERATION, 2009).

## 5. Validação

Este trabalho foi desenvolvido com o objetivo de suprir uma necessidade de mercado real, onde uma Empresa X necessitava administrar os dados de seus produtos em um local centralizado. Este local é o ERP da empresa, onde se concentram as informações mais atualizadas e consistentes.

Seria inviável o deslocamento de pessoal para atualização via CMS do canal de comunicação WEB, que a Empresa X iniciou o desenvolvimento. E um risco para a segurança da informação permitir um *link* direto ao banco de dados do ERP a uma empresa terceira, que implementaria o *website*. Logo, a utilização de *Web Services* foi eleita, e a forma de organização do fluxo de sincronização definida por este trabalho.

## 6. Resultados obtidos

Os objetivos de agilidade e clareza no momento da implementação da camada de dados do *website* foram alcançados com êxito, pois não exigiram dos desenvolvedores envolvidos grandes conhecimentos sobre *Web Services*, pois a classe WSPProxy permitiu o retorno de dados nativos da linguagem PHP às classes de Regras de Negócio e Acesso a Dados, com arquitetura já difundida na equipe de produção.

Quanto à meta de aumento da rentabilidade do projeto resultante deste esforço minimizado de implementação, foi prejudicada em função da análise de requisitos e documentação técnica de baixa qualidade a respeito do *Web Service*, gerada pela equipe de desenvolvimento do *website*, e pouco amadurecimento no modelo de negócio da Empresa X. Estes fatores geraram retrabalho e constantes modificações na estruturação do modelo de dados da integração.

## 7. Plano de melhorias da Arquitetura

Em algumas situações quando é necessária a sincronização de milhares de dados houve uma queda de desempenho da arquitetura, possivelmente causada pela transferência de uma quantidade muito grande de dados via XML, e a necessidade de processamento através do *script* PHP. Eventualmente uma melhoria na estruturação dos dados transferidos, com algum tipo de compactação será necessária.

A respeito da segurança na transição de informações através dos serviços, podem ser implementadas algumas formas de controle, como um sistema de autenticação, onde cada serviço teria como parâmetros dados de acesso as informações em questão, e caberia ao serviço conferi-las, e aferir ou não a liberação dos dados a requisição. Outra proposta interessante é utilizar um canal SSL (*Secure Socket Layer*) para as operações, tecnologia que permite a transição de dados encriptados através do protocolo HTTP.

Outro tipo de melhoria estimada é a criação dinâmica das classes de regras de negócio e acesso aos dados no momento da execução de um serviço. Isto no caso de a ferramenta sincronizada não utilizar o *framework* proposto neste trabalho para consulta e listagem de dados na *interface* com o usuário, o que economizaria espaço em disco do servidor de aplicação assim como o controle de versão das classes de AD e RN.

E a última modificação proposta é uma *interface* de configuração da WSPProxy, o que permitiria que o desenvolvedor configurasse, por exemplo, parâmetros diferentes para cada *operation* existente no serviço, assim não haveria a necessidade de customização no código da classe.

## 8. Conclusões

A utilização de *Web Services* para integração de sistemas é muito adequada, pois permite uma representação detalhada da estrutura de dados a ser transferida através do protocolo SOAP, e também de sua descrição a outros dispositivos, utilizando WSDL.

O WSDL foi explorado em todo desenvolvimento deste trabalho. A grande vantagem deste documento é permitir a implementação de automações para tráfego de informações, pois descreve minuciosamente toda a operação necessária para troca de mensagens entre duas máquinas, com distribuição baseada em uma tecnologia altamente

difundida atualmente, o XML, garante alta compatibilidade com diversos dispositivos computacionais.

Aplicando a arquitetura proposta neste trabalho, principalmente para sistemas WEB, o ganho em rastreabilidade de problemas na execução das sincronizações foi considerável, pois torna as etapas da sincronização, normalmente disparada pelo *website*, de fácil isolamento, como por exemplo:

- Isolar resposta do *Web Service*;
- Isolar problemas em camada de regras de negócio;
- Isolar problemas no momento da persistência dos dados no banco.

Este tipo de recurso é proporcionado pela separação de tarefas em diferentes classes: a abstração de clientes fica a cargo da WSPProxy, as regras de negócio em classes de “RN” e a persistência de dados em classes de “AD”.

Portanto a facilidade de rastreabilidade de problemas de execução, geração de código automático e redução dos esforços em desenvolvimento, tornam a ferramenta uma boa opção para integração de sistemas, como no caso de validação deste trabalho.

## 9. Referências

MYGENERATION. *MyGeneration Help*. Disponível em: <<http://www.mygenerationsoftware.com/portal/Documentation/MyGeneration/tabid/58/Default.aspx>>. Acessado em 13 de Julho de 2009.

NUSOAP. *Introduction to NuSOAP*. Disponível em: <<http://www.scottnichol.com/nusoapintro.htm>>. Acessado em 13 de Julho de 2009.

OASIS. *UDDI Spec Technical Committee Draft*. Disponível em: <[http://uddi.org/pubs/uddi\\_v3.htm](http://uddi.org/pubs/uddi_v3.htm)>. Acessado em 29 de setembro de 2008.

RFC2616 - *Hypertext Transfer Protocol -- HTTP/1.1* – Disponível em: <<http://www.ietf.org/rfc/rfc2616.txt>>. Acessado em 13 de Julho de 2009.

RFC5531. *RPC: Remote Procedure Call Protocol Specification Version 2*. Disponível em: <<http://tools.ietf.org/html/rfc5531>>. Acessado em 13 de Julho de 2009.

RICHARDS, Robert. *PHP XML AND WEBSERVICES: Master working with XML and Web services using PHP*. New York: Apress, 2006.

W3CSOAP. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. Disponível em: <<http://www.w3.org/TR/soap12-part1/>>. Acessado em 21 de setembro de 2008.

W3CWS. *Web Services Architecture*. Disponível em: <<http://www.w3.org/TR/ws-arch/#whatis>>. Acessado em 13 de Julho de 2009.

W3CWSDL. *Web Services Description Language (WSDL) 1.1*. Disponível em: <<http://www.w3.org/TR/wsdl>>. Acessado em 21 de setembro de 2008.