Daniel Mourão Felizardo

Arquitetura Orientada a Serviços: Proposta Preliminar de Serviços para Atendimento ao Consumidor de uma Empresa de Energia Elétrica

Belo Horizonte

2016

Daniel Mourão Felizardo

Arquitetura Orientada a Serviços: Proposta Preliminar de Serviços para Atendimento ao Consumidor de uma Empresa de Energia Elétrica

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para obtenção do grau de Especialista em Informática.

Universidade Federal de Minas Gerais

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Especialização em Informática: Engenharia de Software

Orientador: Marco Túlio de Oliveira Valente

Belo Horizonte

2016

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Felizardo, Daniel Mourão.

F316a Arquitetura orientada a serviços: proposta preliminar de serviços para atendimento ao consumidor em uma empresa de energia elétrica . / Daniel Mourão Felizardo. Belo Horizonte, 2016.
xvi, 64 f.: il.; 29 cm.

Monografia (especialização) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador Marco Túlio de Oliveira Valente.

1. Computação. 2. Engenharia de software. 3. Software - Desenvolvimento. 4. Programação orientada a objetos (Computação). 5. Sistemas operacionais distribuídos (Computadores). I. Orientador. II. Título.

CDU 519.6*32(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
ESPECIALIZAÇÃO EM INFORMÁTICA: ÁREA DE CONCENTRAÇÃO ENGENHARIA DE SOFTWARE

ARQUITETURA ORIENTADA A SERVIÇOS: PROPOSTA PRELIMINAR DE SERVIÇOS PARA ATENDIMENTO AO CONSUMIDOR EM UMA EMPRESA DE ENERGIA ELÉTRICA

DANIEL MOURÃO FELIZARDO

Monografia apresentada aos Senhores:

Prof. Marco Túlio de Oliveira Valente
Orientador
DCC - ICEx - UFMG

Profa. Mariza Andrade da Silva Bigonha
DCC - ICEx - UFMG

Belo Horizonte, 28 de junho de 2016

Este trabalho é dedicado aos meus filhos que sentiram a minha ausência e minha esposa que supriu a minha falta com excelência e amor.

Resumo

A percepção do cliente quanto ao desenvolvimento de novas tecnologias conflita com a expectativa de sua aplicação nos negócios. É comum o cliente reclamar maior agilidade da equipe de TI para adoção de novidades enquanto, a TI está sustentando um legado de produtos que não raramente possuem os mesmos requisitos replicados. Alinhar a expectativa do negócio com os limites técnicos e financeiros da TI, diminuindo o custo de propriedade, simplificando a arquitetura, aumentando o reúso de software é o que tornou atrativo a adoção da Arquitetura Orientada a Serviços. **Palavras-chave**: Arquitetura Orientada a Serviços. SOA. Sistemas Distribuídos.

Abstract

The business' perception of new Technologies development conflicts with the expectation of application in their business. Often the customer complains agility to adopt innovations while IT is supporting a legacy of products that not infrequently have replicated requirements. Align business expectations to technical and financial restrictions of IT, lowering ownership costs, simplifyed architecture and increasing software reuse lead IT to Service Oriented Architecture approach. **Keywords**: Service Oriented Architecture. SOA. Distributed Systems.

Lista de ilustrações

Lista de abreviaturas e siglas

SOA	*Service Oriented Architecture*, Arquitetura Orientada a Serviços
TI	Tecnologia da Informação
URA	Unidade de Resposta Audível
SMS	*Short Message Service*, Serviço de Mensagem Curta
API	*Application Programming Interface*, Interface de Programação de Aplicações
BPM	*Business Process Management*, Gerenciamento de Processo de Negócio
SOAP	*Simple Object Access Protocol*, Protocolo Simples de Acesso a Objetos
RPC	*Remote Procedure Call*, Chamada de Procedimento Remoto
REST	*Representational State Transfer*, Transferência de Estado Representacional
WSDL	*Web Services Description Language*, Linguagem de Descrição de Serviços *Web*.
HTTP	*Hypertext Transfer Protocol*, Protocolo de Transferência de Hipertexto
XML	*eXtensible Markup Language*, linguagens de marcação para necessidades especiais.
URI	*Uniform Resource Identifier*, Identificador Uniforme de Recursos
URL	*Uniform Resource Locator*, Localizador Padrão de Recursos
SGML	*Standard Generalized Markup Language*, Metalinguagem de marcação de documentos.
XSD	*XML Schema Definition*, Documento que descreve uma estrutura de um esquema de dados.
ROI	*Return on Investment*, Retorno do Investimento.
JSON	*Java Script Object Notation*, formato para troca de informações.
CIM	*Commom Information Model*
IEEE	*Institute of Electrical and Electronics Engineers*

Sumário

1 Introdução

Em um mundo cada vez mais conectado e diversificado as pessoas se conectam por meio das mais diversas plataformas de dispositivos, sistemas operacionais, redes sociais e aplicativos. Esta rede de conexões forma as mais diversas possibilidades de combinações. Não raro, em um ambiente corporativo, o surgimento de diversas aplicações com requisitos idênticos e comportamento diferente. Este sintoma ocorre devido a necessidade de cobertura e alcance nas diversas plataformas. Este fenômeno não é algo desejável uma vez que aumenta o custo e a complexidade do ambiente.

Quando o cenário descrito acima se concretiza, quer seja, ignorando o reúso, ou quando existente, é um reúso não planejado e executado frequentemente via clonagem de código fonte, temos um cenário onde o custo de manutenção desses programas cresce proporcionalmente à quantidade de programas criados. Outro problema é que cada aplicação pode ser mantida por equipes ou fornecedores diferentes, criando assim a possibilidade de um mesmo requisito ser entendido e implementado de maneira distinta criando dois programas com ciclo de vida próprio.

Este trabalho tem como base o cenário de uma empresa do setor de serviços, ou seja, a atividade fim não é o desenvolvimento de *software*, que enfrenta o desafio de aproximar cada vez mais do cliente, fornecendo diversos canais de autoatendimento e, principalmente, a contínua redução de custos.

A figura 1 ilustra um cenário atual e é possível imaginar o transtorno causado quando

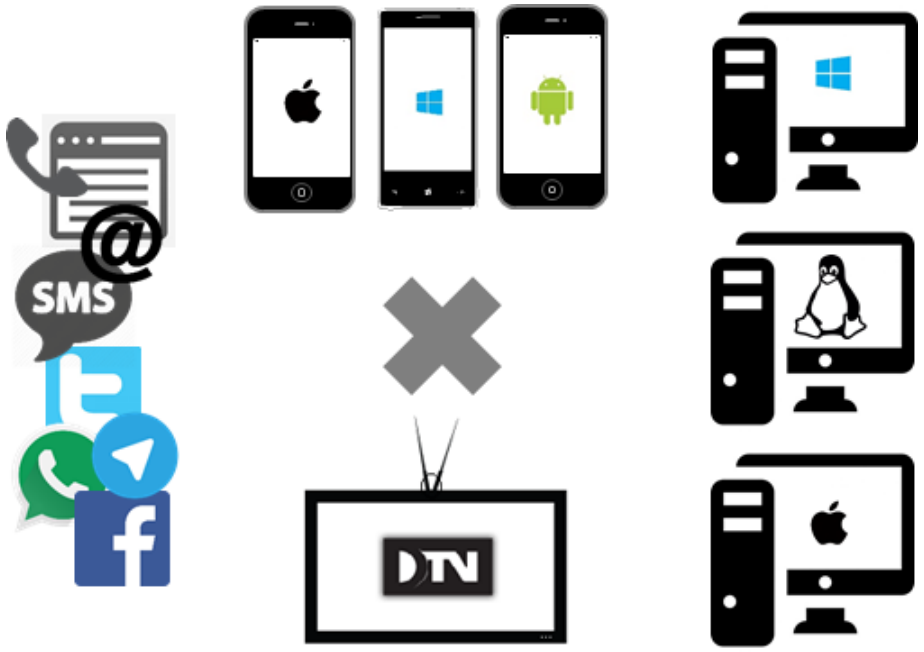

Figura 1 – Diversidade de Plataformas e Canais disponibilizados para o cliente.

um requisito de negócio é modificado, afetando três plataformas móveis, um aplicativo *web* o qual deve ser testado nas principais versões de diversos navegadores, aplicativo para TV Digital, URA, SMS, Twitter, Facebook, Telegram, entre outros diferentes canais.

No mundo corporativo é fundamental justificar todo o investimento executado. O retorno do investimento, *ROI*, é uma das técnicas utilizadas para verificar a viabilidade de um determinado projeto. Outra questão é o teste de *software* que pode ser facilmente justificado quando se calcula o custo de não testar. Porém, como justificar o teste de um mesmo requisito em várias plataformas? Este ponto certamente será questionado pelo corpo executivo.

Outro ponto de atenção neste projeto são as interfaces com os sistemas legados de atendimento, relacionamento e faturamento que podem possuir vícios de implementação como, por exemplo, expor o funcionamento interno de suas funções, faltar detalhes na documentação e até mesmo a inexistência de documentação, falhar na falta de padronização de uso e de comportamento esperado. Esses fatores contribuem aumentando a dificuldade de desenvolvimento e integrações de novos sistemas, por necessitar de conhecimento tanto do negócio quanto da tecnologia envolvida. Dos problemas citados a falha na documentação do *software* é a questão que mais impacta no desenvolvimento, frequentemente as funções e atributos não são facilmente compreendidos e podem demandar a necessidade de executar um projeto de engenharia reversa para tentar entender o *software* em uso.

O desafio proposto, que será estudado neste trabalho, é a criação de um arcabouço de *API*, utilizando uma Arquitetura Orientada a Serviços, *SOA*, para o desenvolvimento de novas aplicações de atendimento ao consumidor. O principal benefício esperado é o incentivo ao reúso de *software* na companhia, consequentemente a redução de custo. Outros benefícios esperados são: (1) a possibilidade de simplificação da arquitetura e do relacionamento entre os sistemas de atendimento, representada pela Figura 2, e (2) facilitar a criação de novas aplicações de maneira rápida, combinando os recursos das *API* desenvolvidas. Espera-se como resultado, o ganho de agilidade na simplificação das interfaces e no uso de nomeclatura que remete aos componentes do negócio. Estes serviços criados deverão ser reaproveitáveis, autônomos e agregáveis, entre outras características de serviços de uma Arquitetura *SOA*. Além disso, devem ocultar sua complexidade interna de funcionamento e preferencialmente ser de fácil compreensão pelo público, que não conhece profundamente o negócio da empresa e também, nem sempre, conhecem as tecnologias envolvidas.

Também são esperados benefícios secundários com o reaproveitamento de código como, por exemplo, na possibilidade do reaproveitamento do *software* proposto por outras companhias do setor, e também, a oportunidade de integração com sistemas de terceiros como, por exemplo, sistemas de fornecedores ou de consumidores, com o devido cuidado e controle de acesso aos dados e de segurança da informação.

Este trabalho também explora, como objetivo central, a adoção e os conceitos da Arquitetura Orientada a Serviços, *SOA*, bem como conceitos de serviços *web*, orquestração de serviços,

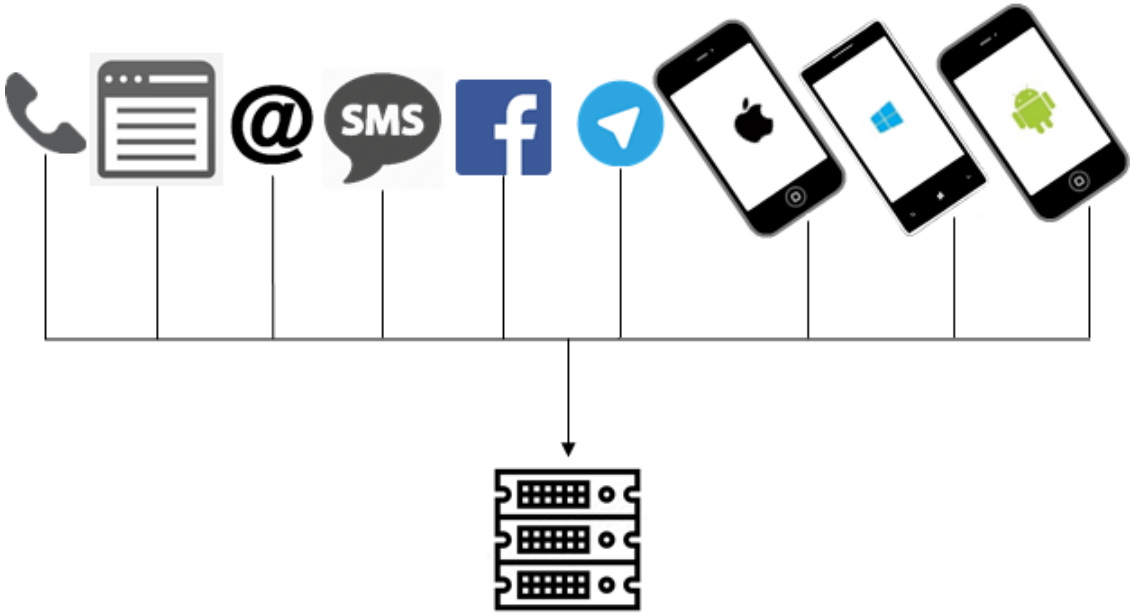

Figura 2 – Racionalização dos ativos de TI.

SOAP, *REST*, *WSDL*, barramento de serviços e *BPM*. Na sequência são apresentados o contexto, as oportunidades, os riscos e as decisões de projeto tomadas durante a elaboração desse trabalho.

1.1 Motivações

A redução de custo com a racionalização dos ativos de TI é o principal motivador deste projeto. Este projeto também tem como motivadores secundários a diminuição da dependência de fornecedores de *software* e as necessidades de domínio e controle sobre a regra de negócio. No cenário atual existem algumas aplicações desenvolvidas com forte dependência de fornecedores e de suas tecnologias que impossibilitam suas manutenções.

Há de se considerar também o cenário de mudanças nos sistemas legados, que impactam diretamente os diversos aplicativos existentes, ocasionando a necessidade da execução de alterações e testes em uma grande quantidade de *software*.

Uma outra grande barreira, que motivou a execução desse projeto, é a do aprendizado e desenvolvimento de novas aplicações, integradas aos sistemas legados, devido a dificuldade de interface com estes sistemas, a falta de documentação técnica e de padronização das interfaces. Esses fatores dificultam o entendimento e afetam a produtividade dos desenvolvedores. Frequentemente, os desenvolvedores, precisam executar um trabalho de engenharia reversa para tentar entender o funcionamento desse legado.

Outros importantes motivadores para esse trabalho dizem respeito: (1) ao direcionamento do negócio que visa maior transparência nos processos, delegando um maior controle e acompanhamento pelo consumidor, bem como permitindo a integração entre os sistemas da companhia

e sistemas de terceiros. (2) A possibilidade de fornecer ao negócio resposta rápida na adesão da companhia à novas tecnologias e plataformas e possibilitar via da criação de um mediador, serviço *web*, absorção, em um ponto único, das mudanças demandadas pelo negócio.

Aliado aos fatores descritos, a possibilidade de implementar uma nova cultura de desenvolvimento na companhia, a Arquitetura Orientada a Serviços, foram os motivadores dessa proposta de trabalho.

1.2 Contribuições

As principais contribuições desse trabalho são:

- Especificação de um modelo de dados canônico para o subconjunto de atendimento ao cliente para uma empresa de distribuição de energia elétrica.
- Análise e implementação de um caso de uso e comparação com o modelo anterior.
- Algumas decisões de projetos.

1.3 Organização do Texto

Capítulo 2 apresenta os conceitos de Arquitetura Orientada a Serviços, um breve histórico sobre integrações entre sistemas e o arcabouço tecnológico, seus padrões, protocolos e conceitos que permitiram o desenvolvimento da Arquitetura Orientada a Serviços. Nesse capítulo são apresentados os conceitos e propostas que devem nortear o desenvolvimento deste trabalho.

Capítulo 3 apresenta a metodologia do trabalho e suas limitações.

Capítulo 4 descreve o processo de desenvolvimento da arquitetura e as decisões de projetos, bem como os desafios e problemas encontrados durante o desenvolvimento. Nesse capítulo também é apresentado os componentes desenvolvidos, produtos e tecnologias utilizadas.

Capítulo 5 apresenta a conclusão e considerações sobre contribuições e trabalhos futuros.

2 Referenciais Teóricos

Este capítulo aborda os referenciais teóricos e o contexto tecnológico que permitiu a interoperabilidade entre sistemas via serviços *web* e um breve histórico do desenvolvimento destas tecnologias.

2.1 Integração Entre Sistemas

Inicialmente, a integração e a comunicação entre processos de sistemas de computador, que estão em contextos diferentes, ou seja, usam um conjunto de endereçamento de memória distintos e estão, normalmente, executando em computadores diferentes, foram baseadas na Chamada Remota de Procedimento, RPC, descrita RFC707 (WHITE, 1975).

O modelo proposto, ilustrado na Figura 3, esconde a complexidade da camada de transporte de rede para o programador, que utiliza um *stub* cliente para a invocação do método remoto. Deste ponto, a implementação da *RPC*, faz o trabalho de transporte e execução do lado do servidor e o retorno da resposta.

Diversas implementações foram desenvolvidas, incompatíveis entre si, dependentes de plataforma, sistema operacional e até linguagem de programação, por exemplo, as implementações da Microsoft, o MSRPC, DCOM e o COM+ que são produtos *RPC* proprietários para Windows (MICROSOFT, 2003) (HORSTMANN; KIRTLAND, 1997), Java RMI que é uma

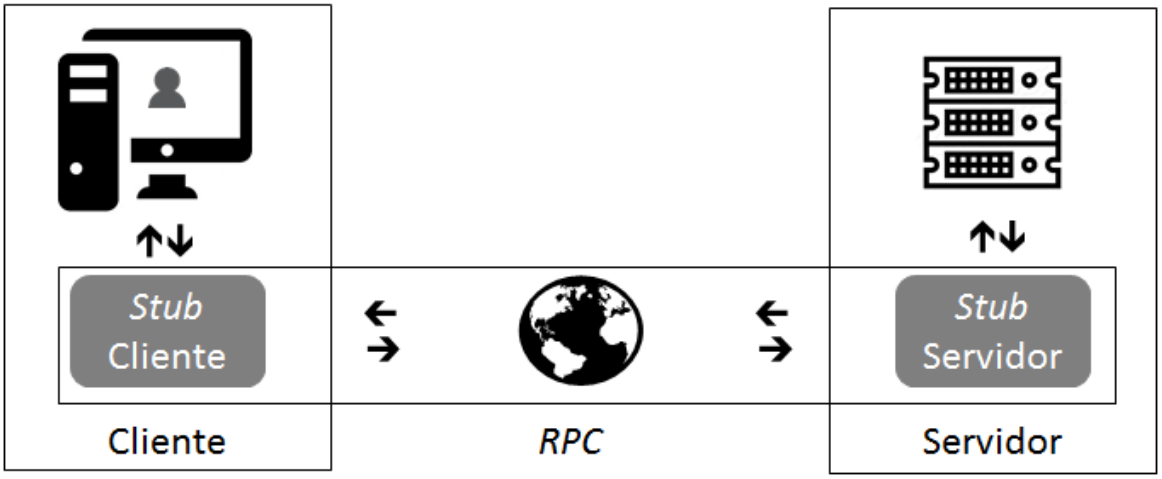

Figura 3 – Simplificação do modelo *RPC*. As chamadas entre o programa e o *stub*, tanto do lado do cliente quanto do lado do servidor, funcionam como uma chamada de procedimento local. A comunicação entre os *stubs* é de responsabilidade da implementação *RPC* utilizada, essa comunicação é efetuada por trocas de mensagens via de uma pilha de protocolos de rede.

implementação de *RPC* para linguagem de programação Java e CORBA que é uma implementação independente de plataforma e linguagem de programação porém, dependente do protocolo proprietário CORBA (CERAMI, 2002).

Há pelo menos duas limitações nas implementações de *RPC* citadas acima. A primeira limitação é a dependência de tecnologia, e em um mundo cada vez mais diversificado, torna incompreensível a adoção de uma tecnologia com esta característica. Outra limitação dessas implementações é a propriedade, algumas destas implementações são dependente de um fornecedor ou plataforma proprietária.

Na próxima seção são exploradas as tecnologias e o contexto que permitiu o surgimento de integrações via serviços *web* e o desenvolvimento da Arquitetura Orientada a Serviços.

2.2 Arquitetura Orientada a Serviço

Este capítulo apresenta o contexto tecnológico que permitiu a popularização da Arquitetura Orientada a Serviços, arquitetura esta que permitiu e popularizou a interoperabilidade entre sistemas via *Web*.

2.2.1 Arquitetura de *Software* e Serviços

Antes de falar sobre a Arquitetura Orientada a Serviços e as tecnologias envolvidas é importante falar sobre os termos Arquitetura e Serviço.

Existem várias definições sobre o que é Arquitetura de *Software*. Segundo Perry e Wolf (1992) é um conjunto de regras de construção, decisões de projeto e elementos que têm como alvo um objetivo e que devem ter respeitadas suas restrições. Krafzig et al. (2004) dizem que são conjuntos de regras que descrevem os componentes e suas relações, é a estrutura do sistema. Fowler (2002) separa arquitetura em duas definições, a primeira se refere aos componentes do sistema e suas interações e a segunda são as decisões difíceis de serem mudadas.

Serviço é uma atividade que executa uma necessidade de negócio. É um item unitário, independente, autocontido e que pode ser facilmente reutilizado em diversos contextos, ou seja, os serviços podem ser combinados para formar outros serviços mais complexos. Um serviço deve estabelecer um contrato, tem entrada, saída, restrições e comportamentos bem definidos.

Arquitetura Orientada a Serviços é uma decisão arquitetural, focada no desenvolvimento serviços *web*, serviços estes que representam as atividades do negócio. Esta arquitetura tem como finalidade central aproximar o negócio e a TI, mediante o desenvolvimento de interfaces que representam não mais funções de software e sim serviços do negócio.

2.2.2 Arcabouço Tecnológico

Algumas tecnologias contribuíram para a popularização dos serviços *web* e a facilitação das integrações entre sistema, assim permitindo o desenvolvimento da Arquitetura Orientada a Serviços.

2.2.2.1 *Hypertext Transfer Protocol*

O Protocolo de Transferência de Hipertexto, do inglês *Hypertext Transfer Protocol*, *HTTP* (FIELDING et al., 1999), é um protocolo da camada de aplicação que segundo Berners-Lee, Fielding e Frystyk (1996) tem a função de construir sistemas de informação de hipermídia, distribuídos e colaborativos. É a base da *World Wide Web* que iniciou nos anos 1980 e tornou-se extremamente popular no anos 1990. O *HTTP* é um protocolo aberto, ou seja, este protocolo pode ser implementado por qualquer pessoa, simplesmente seguindo os padrões descritos para sua construção.

O *HTTP* funciona como um protocolo de troca de mensagens genéricas e não guarda estado de funcionamento. Este protocolo funciona como um intermediário que abstrai a camada de rede, funciona via de requisição e de resposta, é um ambiente de comunicação cliente-servidor (FIELDING et al., 1999).

2.2.2.2 *eXtensible Markup Language*

Algumas aplicações, de maneira rudimentar, consumiam informações na *Web* buscando por páginas de hipertexto e retirando de seu conteúdo a informação desejada. O programador poderia percorrer o arquivo de hipertexto até uma posição especifica para extrair a informação desejada. Este método de extração de informação é claramente instável, uma vez que as páginas são dinâmicas.

A Linguagem de marcação extensível, do inglês *eXtensible Markup Language*, *XML*, é uma linguagem de marcação aberta que padroniza a troca de informação na Web. O *XML* é um subconjunto da *SGML* e foi projetado para facilitar o desenvolvimento e interoperabilidade entre sistemas (YERGEAU et al., 2004).

O *XML* trouxe alguns benefícios interessantes para o desenvolvimento e integração entre sistemas. O *XML* permite a criação de documentos que permite estruturar a informação. Os documentos *XML* podem ser validados via descrição de seu conteúdo, inclusive quanto a formação de estruturas de dados que são fundamentais para desenvolvimento de sistemas. O *XML* permite a criação de tipos complexos, não somente estruturas simples de texto. Apesar de possuir uma estrutura relativamente simples os metadados do *XML* criam um documento extremamente verboso, ou seja, adiciona uma sobrecarga de informação que aumenta o tamanho da informação que deve ser trafegada. Conforme pode ser visto no exemplo a seguir, que

especifica um documento em *XML* com informações de um logradouro simples, onde menos de 25% do conteúdo é ocupado pela informação e o restante faz parte da formatação do documento.

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <endereco>
3   <logradouro>Av. Barbacena</logradouro>
4   <numero>1000</numero>
5   <complemento/>
6   <cidade>Belo Horizonte</cidade>
7   <estado>Minas Gerais</estado>
8   <cep>30190-131</cep>
9 </endereco>
```

Outro formato para troca de informação entre sistemas que tem se popularizado na última década é o *JSON* (CROCKFORD, 2006). Por ser menos verboso e mais simples ganhou espaço frente ao *XML*.

2.2.2.3 *Simple Object Access Protocol*

O Protocolo Simples de Acesso a Objetos, do inglês *Simple Object Access Protocol*, *SOAP*, definido como protocolo de comunicação aberto e como escolha natural foram adotados como base o *XML* e *HTTP*.

O *SOAP* foi criado como um protocolo aberto, para troca de mensagens de chamada de procedimento remoto e sua implementação sobre uma plataforma consolidada como o *HTTP* e o *XML*, lhe garantiu maior facilidade na adoção dado que são protocolos e formatos populares na *Web* (CURBERA et al., 2002).

O *SOAP* possui um envelope, estrutura simples, conforme Figura 4, formado pelas seguintes estruturas:

- Envelope: é uma estrutura *XML* obrigatória, raiz da mensagem *SOAP*, que define qual o conteúdo da mensagem e quem será o destinatário que deverá processar a mensagem, com informação do *namespace* e dos tipos de dados utilizados. O envelope *SOAP* é formado pelas seguintes estruturas:
- – Cabeçalho (*header*): estrutura opcional que define como a mensagem deverá ser processada pelo destinatário
  – Corpo (*body*): é o conteúdo propriamente dito da mensagem ou então o elemento de falha (*SOAP Fault*).

Exemplo de comunicação *SOAP*:

- Consulta (*request*):

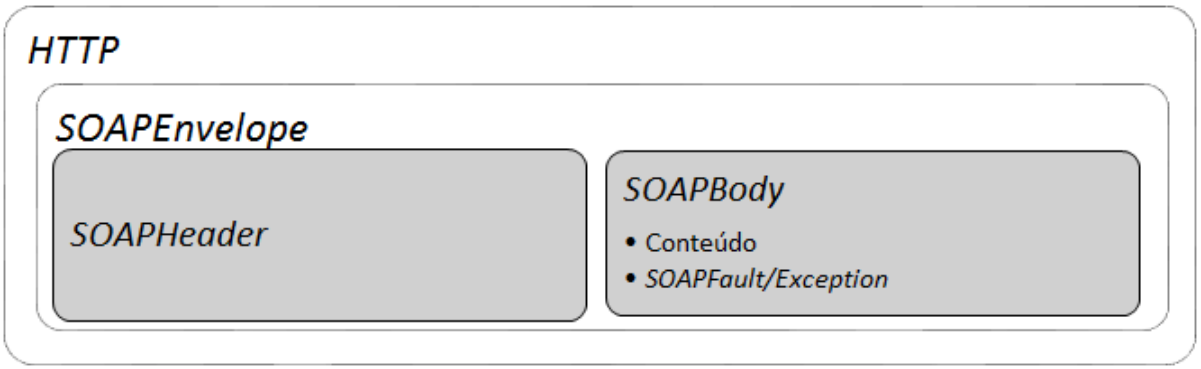

Figura 4 – Representação da estrutura do envelope *SOAP* e a pilha de protocolos: *HTTP* e *SOAP*.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  POST /calculadora HTTP/1.1
3  Host: www.teste.com.br
4  Content-Type: application/soap+xml; charset=utf-8
5  Content-Length: ???
6
7  <?xml version="1.0"?>
8  <soap:Envelope
9    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
10 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
11   <soap:Body
12     xmlns:m="http://www.teste.com.br/calculadora">
13     <m:getDivisao>
14       <m:dividendo>4</m:dividendo>
15       <m:divisor>2</m:divisor>
16     </m:getDivisao>
17   </soap:Body>
18 </soap:Envelope>
```

- Resposta (*response*):

```
1  HTTP/1.1 200 OK
2  Content-Type: application/soap+xml; charset=utf-8
3  Content-Length: ???
4
5  <?xml version="1.0"?>
6  <soap:Envelope
7    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
8  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
9    <soap:Body
10     xmlns:m="http://www.teste.com.br/calculadora">
11     <m:getDivisaoResponse>
12       <m:quociente>2</m:quociente>
```

```
13      </m:getDivisaoResponse>
14    </soap:Body>
15  </soap:Envelope>
```

- Falha (*SOAP fault*) em caso de passar um divisor igual a 0:

```
1  <?xml version="1.0"?>
2  <soap:Envelope
3    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
4  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
5    <soap:Body>
6      <soap:Fault>
7        <faultcode>ZeroDiv</faultcode>
8        <faultstring>Erro Divisao por Zero.</faultstring>
9      </soap:Fault>
10   </soap:Body>
11 </soap:Envelope>
```

2.2.2.4 *Web Services Description Language*

A Linguagem de Descrição de Serviços *web*, do inglês *Web Services Description Language*, *WSDL*, assim como o *SOAP*, é uma linguagem baseada em *XML* que permite descrever o funcionamento de um serviço *web*. O documento escrito com esta linguagem descreve tudo o que o consumidor precisa conhecer para utilizar o serviço *web* como, por exemplo, o endereço do serviço, porta, nome da interface, operações e o esquema da estrutura e dados, definidos em documentos de definição de esquemas *XSD* (CHRISTENSEN et al., 2001).

A seguir é apresentado um exemplo de documento *XSD* que descreve a estrutura de um endereço que é uma estrutura formada por outros esquemas: logradouro, bairro, município e seguimento. Além destes tipos complexos esta estrutura é formada por outro tipo primitivos como nome do logradouro, número, complemento, referência, estado, pais e outros.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema targetNamespace="http://exemplo.com.br/soa/dados/Endereco/v1"
3    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4    xmlns:Q1="http://exemplo.com.br/soa/dados/Logradouro/v1"
5    xmlns:Q2="http://exemplo.com.br/soa/dados/Bairro/v1"
6    xmlns:Q3="http://exemplo.com.br/soa/dados/Municipio/v1"
7    xmlns:Q4="http://exemplo.com.br/soa/dados/SeguimentoLogradouro/v1">
8    <xsd:import schemaLocation="SeguimentoLogradouro.xsd"
9      namespace="http://exemplo.com.br/soa/dados/SeguimentoLogradouro/v1">
10   </xsd:import>
11   <xsd:import schemaLocation="Municipio.xsd"
12     namespace="http://exemplo.com.br/soa/dados/Municipio/v1">
13   </xsd:import>
```

```
14   <xsd:import schemaLocation="Bairro.xsd"
15     namespace="http://exemplo.com.br/soa/dados/Bairro/v1">
16   </xsd:import>
17   <xsd:import schemaLocation="Logradouro.xsd"
18     namespace="http://exemplo.com.br/soa/dados/Logradouro/v1">
19   </xsd:import>
20   <xsd:complexType name="Endereco">
21     <xsd:sequence>
22       <xsd:element name="identificador" type="xsd:string"
23         minOccurs="0" maxOccurs="1"></xsd:element>
24       <xsd:element minOccurs="0" name="logradouro"
25         type="Q1:Logradouro" />
26       <xsd:element minOccurs="0" name="numero"
27         type="xsd:string" />
28       <xsd:element minOccurs="0" name="complemento"
29         type="xsd:string" />
30       <xsd:element minOccurs="0" name="referencia"
31           type="xsd:string" />
32       <xsd:element minOccurs="0" name="estado"
33          type="xsd:string" />
34       <xsd:element minOccurs="0" name="pais"
35         type="xsd:string" />
36       <xsd:element minOccurs="0" name="domicilioFiscal"
37         type="xsd:string" />
38       <xsd:element minOccurs="0" name="codigoLocalidade"
39         type="xsd:string" />
40       <xsd:element minOccurs="0" name="enderecoCompleto"
41         type="xsd:string" />
42       <xsd:element minOccurs="0" name="bairro"
43         type="Q2:Bairro" />
44       <xsd:element minOccurs="0" name="municipio"
45         type="Q3:Municipio" />
46       <xsd:element minOccurs="0" name="seguimento"
47         type="Q4:SeguimentoLogradouro" />
48     </xsd:sequence>
49   </xsd:complexType>
50 </xsd:schema>
```

O próximo exemplo ilustra um documento *WSDL* onde o *XSD* acima é um dos esquemas inseridos.

Ainda podemos perceber neste documento as operações criadas para o serviço de endereço, e em cada função o esquema de dados relacionado:

- associarEnderecoContato
- buscarEnderecoPorCEP

- buscarEnderecoPorInstalacao
- criarEnderecoContato
- listarEnderecoContato
- listarTipoEndereco

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  name="Endereco"
  targetNamespace="http://exemplo.com.br/soa/infra/Endereco"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://exemplo.com.br/soa/infra/Endereco"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:documentation>
    <wsdl:appinfo source="WMQI_APPINFO">
      <MRWSDLAppInfo imported="true">
        <generatedXSD location="ServicoEndereco_InlineSchema1.xsd" />
        <binding hasEncoding="false" imported="true"
          name="EnderecoSOAP"
          originalBindingStyle="document" />
      </MRWSDLAppInfo>
    </wsdl:appinfo>
  </wsdl:documentation>
  <wsdl:types>
    <xsd:schema
      targetNamespace="http://exemplo.com.br/soa/infra/Endereco"
      xmlns:ibmSchExtn="http://www.ibm.com/schema/extensions">
      <xsd:include schemaLocation="ServicoEndereco_InlineSchema1.xsd" />
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="listaEnderecoContatoRequest">
    <wsdl:part element="tns:listaEnderecoContatoRequest"
    name="parameters" />
  </wsdl:message>
  <wsdl:message name="listaEnderecoContatoResponse">
    <wsdl:part element="tns:listaEnderecoContatoResponse"
    name="parameters" />
  </wsdl:message>
  <wsdl:message name="criaEnderecoContatoRequest">
    <wsdl:part element="tns:criaEnderecoContatoRequest"
    name="parameters" />
  </wsdl:message>
  <wsdl:message name="criaEnderecoContatoResponse">
    <wsdl:part element="tns:criaEnderecoContatoResponse"
    name="parameters" />
```

```
41  </wsdl:message>
42  <wsdl:message name="associaEnderecoContatoRequest">
43    <wsdl:part element="tns:associaEnderecoContatoRequest"
44    name="parameters" />
45  </wsdl:message>
46  <wsdl:message name="associaEnderecoContatoResponse">
47    <wsdl:part element="tns:associaEnderecoContatoResponse"
48      name="parameters" />
49  </wsdl:message>
50  <wsdl:message name="buscaEnderecoPorCEPRequest">
51    <wsdl:part element="tns:buscaEnderecoPorCEPRequest"
52    name="parameters" />
53  </wsdl:message>
54  <wsdl:message name="buscaEnderecoPorCEPResponse">
55    <wsdl:part element="tns:buscaEnderecoPorCEPResponse"
56    name="parameters" />
57  </wsdl:message>
58  <wsdl:message name="listaTipoEnderecoRequest">
59    <wsdl:part element="tns:listaTipoEnderecoRequest"
60    name="parameters" />
61  </wsdl:message>
62  <wsdl:message name="listaTipoEnderecoResponse">
63    <wsdl:part element="tns:listaTipoEnderecoResponse"
64    name="parameters" />
65  </wsdl:message>
66  <wsdl:message name="buscarEnderecoPorInstalacaoRequest">
67    <wsdl:part element="tns:buscarEnderecoPorInstalacaoRequest"
68    name="parameters" />
69  </wsdl:message>
70  <wsdl:message name="buscarEnderecoPorInstalacaoResponse">
71    <wsdl:part element="tns:buscarEnderecoPorInstalacaoResponse"
72    name="parameters" />
73  </wsdl:message>
74  <wsdl:message name="EnderecoFalha">
75    <wsdl:part name="parameters" element="tns:EnderecoFalha">
76    </wsdl:part>
77  </wsdl:message>
78  <wsdl:portType name="EnderecoPortType">
79    <wsdl:operation name="listarEnderecoContato">
80      <wsdl:input message="tns:listaEnderecoContatoRequest" />
81      <wsdl:output message="tns:listaEnderecoContatoResponse" />
82      <wsdl:fault name="EnderecoFalha" message="tns:EnderecoFalha">
83      </wsdl:fault>
84    </wsdl:operation>
85    <wsdl:operation name="criarEnderecoContato">
86      <wsdl:input message="tns:criaEnderecoContatoRequest" />
87      <wsdl:output message="tns:criaEnderecoContatoResponse" />
```

```
88        <wsdl:fault name="EnderecoFalha" message="tns:EnderecoFalha">
89        </wsdl:fault>
90      </wsdl:operation>
91      <wsdl:operation name="associarEnderecoContato">
92        <wsdl:input message="tns:associaEnderecoContatoRequest" />
93        <wsdl:output message="tns:associaEnderecoContatoResponse" />
94        <wsdl:fault name="EnderecoFalha" message="tns:EnderecoFalha">
95        </wsdl:fault>
96      </wsdl:operation>
97      <wsdl:operation name="buscarEnderecoPorCEP">
98        <wsdl:input message="tns:buscaEnderecoPorCEPRequest" />
99        <wsdl:output message="tns:buscaEnderecoPorCEPResponse" />
100       <wsdl:fault name="EnderecoFalha" message="tns:EnderecoFalha">
101       </wsdl:fault>
102     </wsdl:operation>
103     <wsdl:operation name="listarTipoEndereco">
104       <wsdl:input message="tns:listaTipoEnderecoRequest" />
105       <wsdl:output message="tns:listaTipoEnderecoResponse" />
106       <wsdl:fault name="EnderecoFalha" message="tns:EnderecoFalha">
107       </wsdl:fault>
108     </wsdl:operation>
109     <wsdl:operation name="buscarEnderecoPorInstalacao">
110       <wsdl:input message="tns:buscarEnderecoPorInstalacaoRequest" />
111       <wsdl:output message="tns:buscarEnderecoPorInstalacaoResponse" />
112       <wsdl:fault name="EnderecoFalha" message="tns:EnderecoFalha" >
113       </wsdl:fault>
114     </wsdl:operation>
115   </wsdl:portType>
116   <wsdl:binding name="EnderecoSOAP" type="tns:EnderecoPortType">
117     <soap:binding style="document"
118       transport="http://schemas.xmlsoap.org/soap/http" />
119     <wsdl:operation name="listarEnderecoContato">
120       <soap:operation
121       soapAction
122       ="http://exemplo.com.br/soa/infra/Endereco/listarEnderecoContato" />
123       <wsdl:input>
124         <soap:body use="literal" />
125       </wsdl:input>
126       <wsdl:output>
127         <soap:body use="literal" />
128       </wsdl:output>
129       <wsdl:fault name="EnderecoFalha">
130         <soap:fault name="EnderecoFalha" use="literal" />
131       </wsdl:fault>
132     </wsdl:operation>
133     <wsdl:operation name="criarEnderecoContato">
134       <soap:operation
```

```
135         soapAction
136         ="http://exemplo.com.br/soa/infra/Endereco/criarEnderecoContato" />
137       <wsdl:input>
138         <soap:body use="literal" />
139       </wsdl:input>
140       <wsdl:output>
141         <soap:body use="literal" />
142       </wsdl:output>
143       <wsdl:fault name="EnderecoFalha">
144         <soap:fault name="EnderecoFalha" use="literal" />
145       </wsdl:fault>
146     </wsdl:operation>
147     <wsdl:operation name="associarEnderecoContato">
148       <soap:operation
149         soapAction=
150         "http://exemplo.com.br/soa/infra
151 ␣␣␣␣␣␣␣␣/Endereco/associarEnderecoContato"/>
152       <wsdl:input>
153         <soap:body use="literal" />
154       </wsdl:input>
155       <wsdl:output>
156         <soap:body use="literal" />
157       </wsdl:output>
158       <wsdl:fault name="EnderecoFalha">
159         <soap:fault name="EnderecoFalha" use="literal" />
160       </wsdl:fault>
161     </wsdl:operation>
162     <wsdl:operation name="buscarEnderecoPorCEP">
163       <soap:operation
164         soapAction
165         ="http://exemplo.com.br/soa/infra/Endereco/buscarEnderecoPorCEP" />
166       <wsdl:input>
167         <soap:body use="literal" />
168       </wsdl:input>
169       <wsdl:output>
170         <soap:body use="literal" />
171       </wsdl:output>
172       <wsdl:fault name="EnderecoFalha">
173         <soap:fault name="EnderecoFalha" use="literal" />
174       </wsdl:fault>
175     </wsdl:operation>
176     <wsdl:operation name="listarTipoEndereco">
177       <soap:operation
178         soapAction
179         ="http://exemplo.com.br/soa/infra/Endereco/listarTipoEndereco" />
180       <wsdl:input>
181         <soap:body use="literal" />
```

```
182         </wsdl:input>
183         <wsdl:output>
184           <soap:body use="literal" />
185         </wsdl:output>
186         <wsdl:fault name="EnderecoFalha">
187           <soap:fault name="EnderecoFalha" use="literal" />
188         </wsdl:fault>
189       </wsdl:operation>
190       <wsdl:operation name="buscarEnderecoPorInstalacao">
191         <soap:operation
192         soapAction=
193         "http://exemplo.com.br/soa/infra
194 ␣␣␣␣␣␣/Endereco/buscarEnderecoPorInstalacao"
195         />
196         <wsdl:input>
197           <soap:body use="literal" />
198         </wsdl:input>
199         <wsdl:output>
200           <soap:body use="literal" />
201         </wsdl:output>
202         <wsdl:fault name="EnderecoFalha">
203           <soap:fault name="EnderecoFalha" use="literal" />
204         </wsdl:fault>
205       </wsdl:operation>
206     </wsdl:binding>
207     <wsdl:service name="EnderecoService">
208       <wsdl:port binding="tns:EnderecoSOAP" name="EnderecoBinding">
209         <soap:address
210         location="http://localhost:7800/infra/endereco/enderecoService" />
211       </wsdl:port>
212     </wsdl:service>
213 </wsdl:definitions>
```

Com o documento acima é possível montar o esqueleto da função, ele é utilizado para a criação do *stub*, e então, a partir dele, é possível consumir o serviço descrito por meio de um sistema ou um navegador *web*.

2.2.2.5 *Universal Description, Discovery and Integration*

A *UDDI*, *Universal Description, Discovery na Integration*, é um padrão que define como um serviço *web* deve ser publicado e descoberto. Em resumo é um serviço *web* que informa os esquemas *XSD*, os *WSDL*, *SOAP* e também informações de negócio, responsáveis e política do serviço (COMMITTEE et al.,).

Resumidamente em um diretório *UDDI* é possível que um fornecedor de serviço registre suas informações de contato, descritivo do serviços, informações técnicas de uso e os esquemas

e modelos (COMMITTEE et al.,).

2.2.2.6 Serviços *Web*

A equação para formação de um serviço *Web* é a soma de *HTTP, XML, SOAP, WSDL* e *UDDI*, sendo que este último não é obrigatório.

O motivo do sucesso dos serviços web em relação aos demais serviços *RPC* citados no início do capítulo é a independência de sistemas operacionais, plataformas e linguagens de programação. Outro fator de sucesso é que todo os protocolos e linguagens envolvidas são abertos e de amplo conhecimento. Podemos dizer que os serviços *web* aproveitaram o sucesso da *Web*, sua heterogeneidade e independência. Hoje todo o mundo está conectado via *Web* e nada mais natural e simples do que utilizar uma plataforma consolidada para integração entre sistemas.

2.2.3 Arquitetura Orientada a Serviços: O Que é *SOA*?

Do inglês *Service Oriented Architecture*, *SOA*, possui várias definições como, por exemplo, segundo o GARTNER (2014) é um paradigma e disciplina que aproxima a TI do negócio. *SOA* auxilia na diminuição de custos, consistência e agilidade no desenvolvimento. Permite a criação de serviços de negócio interoperáveis que podem ser reutilizados e compartilhados. A IBM (2016) também tem uma definição muito próxima, *SOA* é uma abordagem arquitetural de TI centrada no negócio que garante a integração do negócio. Segundo a Microsoft (2016), *SOA* é a orientação de serviços por meio de integração entre sistemas distintos. Para MacKenzie et al. (2006) SOA é um paradigma para organização e distribuição de responsabilidades que pode estar debaixo de diferentes domínios.

Há diversas definições sobre *SOA*, algumas até se contradizem, mas o que todas são unanimes em dizer é que SOA tem a finalidade de auxiliar de aproximar o negócio da TI por meio da construção de serviços que referenciam ao negócio.

Como abordado nos itens anteriores, *SOA* não é uma tecnologia, não é uma metodologia e *SOA* não são serviços *web*, portanto *SOA* não pode ser comprado em uma prateleira de um grande fornecedor e implantado em sua companhia.

Segundo Arsanjani et al. (2009), orientação a serviços é um paradigma de computação distribuída que molda o negócio. *SOA* não é uma aplicação. *SOA* é dirigida para o negócio, para o crescimento sustentável, agregação de valores, agilidade e redução de custo (ERL, 2005).

Ainda segundo Arsanjani et al. (2009) as prioridades em um projeto *SOA* são: Agregação de valor ao negócio, foco na estratégia do negócio no lugar dos objetivos específicos do projeto, interoperabilidade essencial e não integração customizada, compartilhamento de serviços, flexibilidade e refatoramento contínuo.

Nos princípios do *Manifesto SOA* (ARSANJANI et al., 2009) destaca-se que produtos, plataformas, padrões e tecnologias não garantem a implantação do *SOA* e que pode ser desenvolvido usando quantas tecnologias e padrões forem necessários, mas é importante a busca por padrões de indústria e da comunidade, alcançar a uniformidade exterior, redução de dependências entre serviços e assim, diminuir o impacto de mudanças, abstraindo o modelo de negócio e organizar cada serviço e função dentro de cada módulo da organização.

Em resumo *SOA* não trouxe nenhuma novidade ao mundo da computação porém, contribuiu com uma significativa melhoria a interoperabilidade entre sistemas abortando não a tecnologia, mas a sua aplicação.

2.2.4 Serviço

Serviço é algo que presta uma função para seu consumidor, é uma atividade, uma ação atômica, serviços podem ser orquestrados utilizando um conjunto de outros serviços. Um exemplo abstrato de serviço é o fornecimento de energia elétrica, temos a geração, a transmissão, a distribuição e a comercialização. Para o consumidor não importa a complexidade do setor de energia, o que ele precisa é da energia, simplesmente ligar uma tomada. Considerando estes aspectos tem-se os seguintes princípios de desenho de serviços (ERL, 2009).

- Reúso: o fator principal que influencia a necessidade do reúso para o negócio é a redução de custo(OLIVEIRA[1]; RAMOS; JUNIOR,). Este é um importante fator para adoção de *SOA* nas organizações, possibilitar a reúso de serviços para construção de novos produtos aumentando assim a agilidade da entrega (ALFéREZ GERM'N H E PELECHANO, 2011).

- Possuir um contrato formal de serviço: documentos que descrevem o que o serviço faz, utilizam padrões como *WSDL, XSD, UDDI* e *SOAP* possibilitam a descoberta do serviço e sua utilização. Padrões de contrato dentro da organização devem ser definidos e disciplinados.

- Baixo acoplamento e alta coesão: os serviços devem ser independentes de outros serviços para seu funcionamento e do outro lado devem ter sua função muito bem definida. Estes dois requisitos afetam diretamente a reusabilidade do serviços e por consequência a possibilidade de fazer composição entre serviços.

- Abstração da Lógica de Funcionamento: o usuário deve ser capaz de utilizar o serviço sem entender a lógica de construção ou a complexidade de funcionamento.

- Composição: os serviços devem ser capazes de serem agregados e combinados para resolver um problema maior, formando assim um serviço composto.

- Autonomia: os serviços controlam a sua própria lógica.

- *Stateless*: evitar alocação de recurso por período longo e não guardar estado de funcionamento. O serviço não deve executar um longo processamento, deve funcionar rapidamente enviando uma requisição para o software produtor e também devolver rapidamente uma resposta, para isso podemos desenvolver serviços síncronos, quando o cliente fica aguardando a resposta e assíncronos, quando o cliente volta em outro momento para buscar uma resposta.

- Descoberta: é um item um pouco controverso, como um serviço pode ser encontrado? As pessoas podem até encontrar serviços em catalogo ou diretórios de serviço só que conversas comerciais são necessárias para o entendimento. A descoberta ou o mapeamento de serviços devem ocorrer, por meio de documentação, contrato e descrição do serviço para possibilitar o reúso de software e a agilidade no desenvolvimento de novas aplicações, evitando a replicações de serviços.

2.3 Considerações Finais

Neste capítulo, foram apresentados os conceitos e arcabouço tecnológico necessário para a construção de uma Arquitetura Orientada a Serviços.

Das tecnologias para a construção de serviços *web* apresentadas nenhuma possui restrição de propriedade, ou seja, ao contrário das tecnologias de interoperabilidade entre sistemas também apresentadas que possuíam restrições tecnológicas e de propriedade.

Foram apresentados ainda os conceitos de *SOA*, definições de arquitetura de *software*, como decisão de modelagem e organização de componentes de *sofware* difíceis de serem mudadas, e de serviços, que é uma atividade de negócio, conceitos que definem o modelo arquitetural adotado neste trabalho.

No próximo capítulo será apresentado a metodologia utilizada neste trabalho.

3 Metodologia

Este trabalho tem natureza exploratória e conforme citado na Seção 1.2, tem o objetivo de expor algumas decisões de projeto, a definição parcial de um modelo de dados e a criação de um serviço *web*.

Inicialmente foi realizada uma pesquisa bibliográfica e então confeccionado o referencial teórico, Capítulo 2, para embasar a construção e as decisões de projeto. Este referencial teórico foi buscado em artigos, publicações, referências de fornecedores de soluções (comercial) e livros. Foram priorizados artigos recentes e outros notoriamente referenciados.

No próximo capítulo será apresentado o resultado da execução deste projeto com base nas decisões tomadas que referenciam o contexto teórico levantado no Capítulo 2.

4 Propostas de Serviços

Neste capítulo será abordado o processo criativo de construção de serviços *SOA* e algumas decisões de projeto tomadas durante o projeto.

4.1 Modelo Arquitetural

O primeiro passo do trabalho foi a definição do modelo arquitetural que seria adotado. Foram definidos os componentes principais da arquitetura que podem ser separados em:

- Sistemas Legados: aplicações consolidadas que possuem grande parte das regras de negócio codificadas.
- Repositório de Regras de Negócio: um sistema de armazenamento de decisões e regras de negócio que podem ser reaproveitadas por diversas aplicações e serviços. Neste componente as mensagens dos serviços são decodificadas e processadas, retornando assim um resultado ou uma decisão que poderá mudar o fluxo dos processos dos serviços de negócio.
- Serviços de TI: aplicações de infraestrutura que não implementam uma serviço de negócio mas sim de sistema, por exemplo, um serviço de assinatura digital ou envio de mensagens de SMS.
- Serviços de Negócio: são os serviços que representam as atividades do negócio, por exemplo, impressão de fatura ou histórico de atendimento. Tanto nos Serviços de Negócio quanto nos serviços de TI não existe calculo ou processamento de dados, estas atividades ficam nas funções implementadas nos serviços legados ou então no repositório de regras de negócio.
- BPM: os Serviços de Negócio podem disparar o inicio de um processo de negócio em um sistema de BPM, *Business Process Management*. Estes processos de negócio podem conter serviços de consultas que podem ser expostos no Barramento de Serviços.

Na Figura 5 podemos ter uma ideia quanto a disposição e comunicação entre os componentes da arquitetura proposta. Um componente de software utilizado e não retratado no diagrama da Figura 5 é o serviço de registro de serviços, sistema onde são armazenados a biblioteca de serviços publicados e utilizados de terceiros, além da função de registrador de serviços ele é utilizado como um *proxy* para serviços de terceiros, isolando em apenas um ponto as referências das *URL* utilizadas.

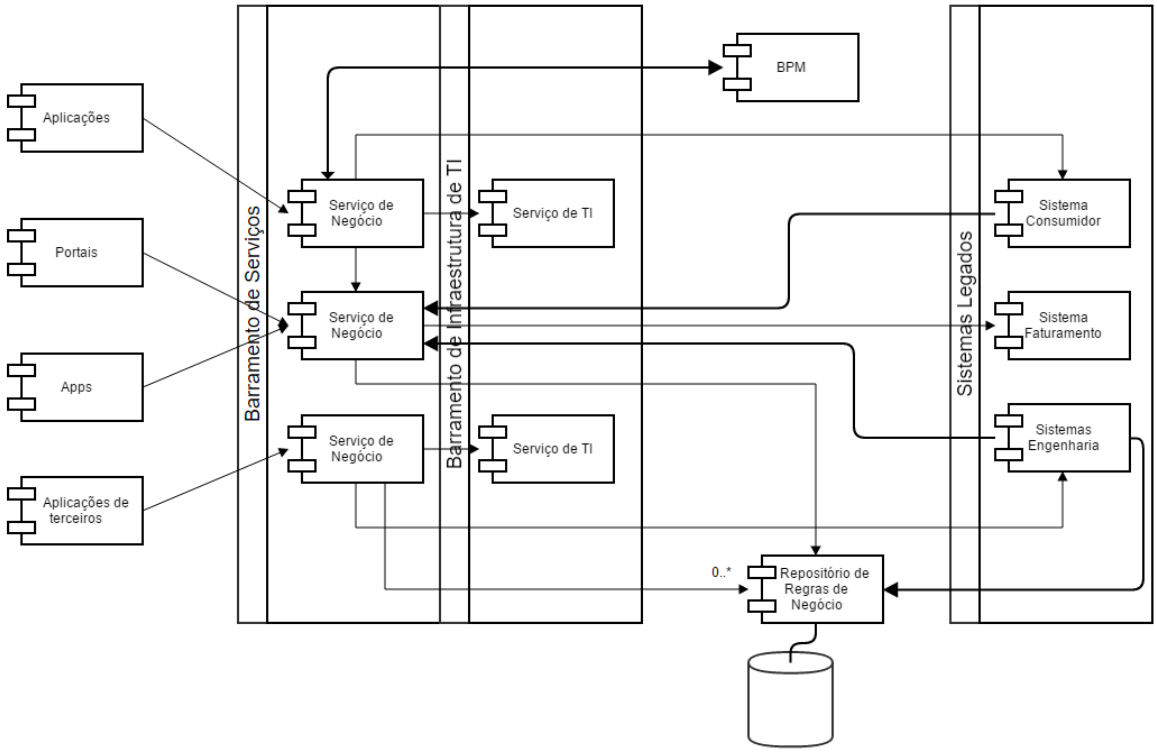

Figura 5 – Representação dos componentes na arquitetura proposta.

4.2 Definição do Modelo de Dados

Assim que definido que seria adotado como principal decisão arquitetural a Orientação a Serviços, e como este modelo arquitetural aborda a estrutura do negócio, foi necessário definir um modelo de dados que representa o negócio. Partindo da definição desse modelo de dados, ou seja, da criação ou reúso de uma modelo de dados que descreve a estrutura do negócio de uma empresa de distribuição de energia é que será possível definir os serviços e suas operações.

Antes de definir este modelo de dados, foi estudado a possibilidade de utilizar o modelo descrito pelo *CIM-IEC61968* (USLAR et al., 2012), que define todos os requisitos que um sistema de gerenciamento de distribuição de energia elétrica deve possuir.

A utilização da *CIM-IEC61968* foi, o primeiro tema polêmico do projeto, uma vez que é um modelo que descreve a indústria de distribuição de energia elétrica com o ponto de vista do *IEEE*. O *IEEE* é uma organização norte americana e portanto é natural que o modelo de negócio descrito espelha a realidade dos Estados Unidos.

Não é que o modelo brasileiro seja incompatível ou muito diferente do modelo norte americano, porém alguns detalhes impediram o uso integral da *CIM-IEC61968* como, por exemplo, as estruturas internas dos objeto descritos, os atributos que descrevem um endereço, uma pessoa ou uma empresa são diferentes. Outro ponto de dificuldade foi o uso dos termos em

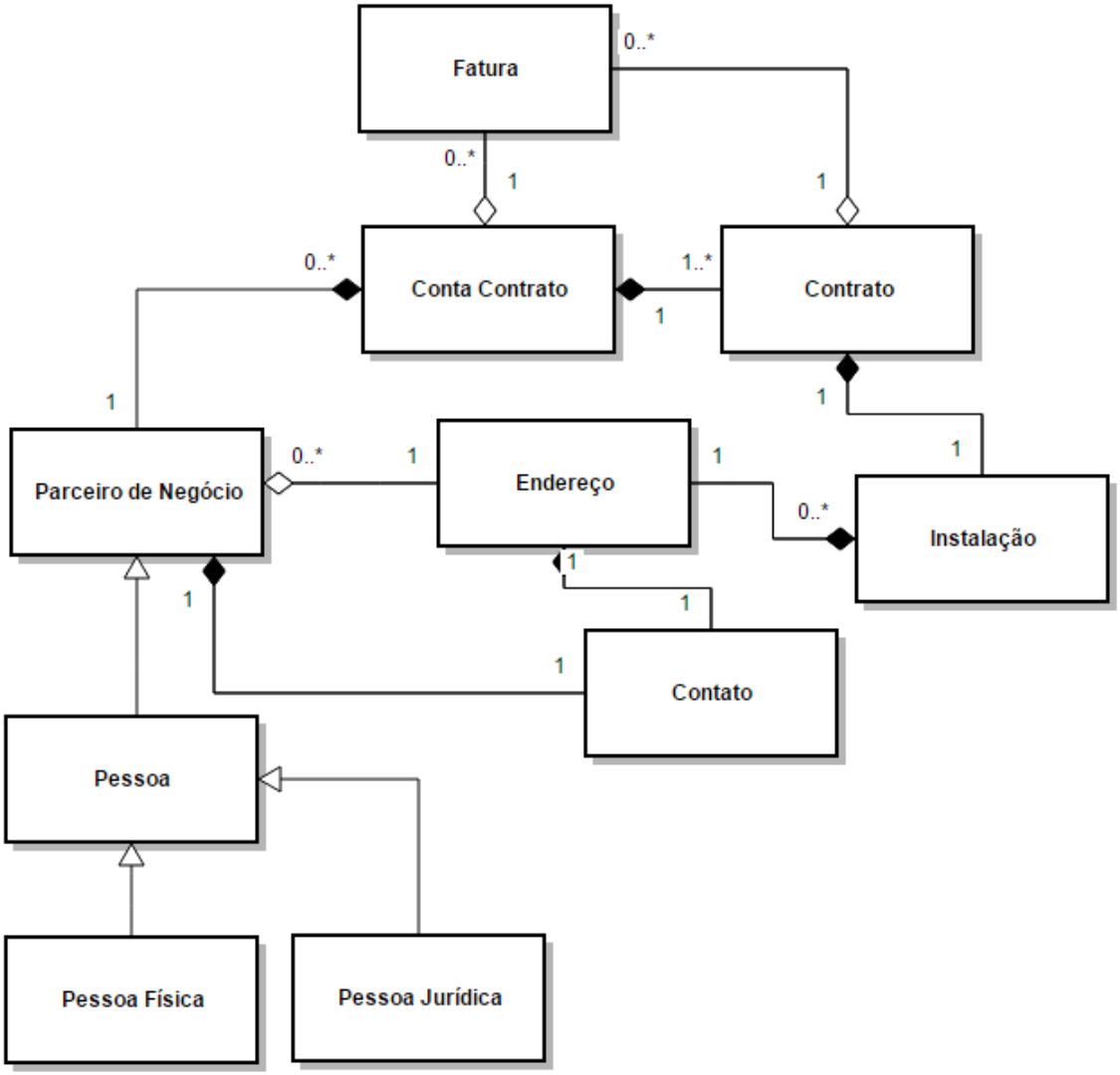

Figura 6 – Modelo Canônico Parcial.

inglês, principalmente as referências da realidade local, que estavam dificultando a interação com o cliente interno.

Portanto, a partir das questões mencionados, foi decidido utilizar a *CIM-IEC61968* como base para um novo modelo em português com aditamento das particularidades da realidade brasileira.

4.2.1 Modelo Proposto

O modelo apresentado nesta seção é parte do modelo global, com a visão apenas do negócio de comercialização de energia elétrica. Utilizando como base o *CIM-IEC61968*, traduzido para o português e adaptado, foi então formatado o modelo descrito na Figura 6.

Este modelo, estruturado conforme a *CIM-IEC61968*, foi amplamente discutido com a

área de negócio e aprovado.

Veja que no modelo proposto, Figura 6, há elementos de herança, composição e agregação que são elementos da Orientação por Objetos. Estes elementos foram definidos nos esquemas *XSD*, como podem ser vistos a seguir.

- Herança entre Pessoa e Pessoa Física: verifique que o tipo Pessoa, abstrato, é um *complexType* com o tipo *abstract* definido como verdadeiro.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema
3 targetNamespace="http://distribuicao.com.br/soa/dados/Pessoa/v1"
4   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
5   <xsd:complexType abstract="true" name="Pessoa">
6     <xsd:sequence></xsd:sequence>
7   </xsd:complexType>
8 </xsd:schema>
```

- A abstração de Pessoa e Pessoa Física: verifique que o tipo PessoaFisica, subclasse de Pessoa, é uma extensão de Pessoa onde são definidos os atributos de PessoaFisica como, por exemplo, nome, sobrenome e CPF.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema
3 targetNamespace="http://distribuicao.com.br/soa/dados/PessoaFisica/v1"
4   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5   xmlns:pes="http://distribuicao.com.br/soa/dados/Pessoa/v1">
6   <xsd:import namespace="http://distribuicao.com.br/soa/dados/Pessoa/v1"
7   schemaLocation="Pessoa.xsd"/>
8   <xsd:complexType name="PessoaFisica">
9     <xsd:complexContent>
10      <xsd:extension base="pes:Pessoa">
11        <xsd:sequence>
12          <xsd:element name="nome"
13          type="xsd:string" minOccurs="0" />
14          <xsd:element name="sobrenome"
15          type="xsd:string" minOccurs="0"/>
16          <xsd:element name="CPF" type="xsd:string"/>
17        </xsd:sequence>
18      </xsd:extension>
19    </xsd:complexContent>
20  </xsd:complexType>
21 </xsd:schema>
```

- A abstração entre Pessoa e Pessoa Jurídica: verifique que o tipo PessoaJuridica, subclasse de Pessoa, é uma extensão de Pessoa onde são definidos os atributos de PessoaJuridica como, por exemplo, CNPJ, razão social e nome fantasia.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
targetNamespace="http://cemig.com.br/soa/dados/PessoaJuridica/v1"
  xmlns:pes="http://cemig.com.br/soa/dados/Pessoa/v1"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:import namespace="http://cemig.com.br/soa/dados/Pessoa/v1"
  schemaLocation="Pessoa.xsd"/>
  <xsd:complexType name="PessoaJuridica">
    <xsd:complexContent>
      <xsd:extension base="pes:Pessoa">
        <xsd:sequence>
          <xsd:element name="nomeFantasia"
          type="xsd:string" minOccurs="0" />
          <xsd:element name="razaoSocial"
          type="xsd:string" minOccurs="0"/>
          <xsd:element name="CNPJ"
          type="xsd:string"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

- Elemento de Agregação de Parceiro de Negócio com Pessoa e Contato: para um Parceiro de Negócio, nome sugerido para consumidor, existir ele deve ser composto de Pessoa, Contato e outros itens. A escolha do elemento de agregação é devido a relação de Todo-Parte, onde Contato não faz sentido sem Parceiro de Negócio.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
targetNamespace="http://distribuidora.com.br/soa/dados/ParceiroNegocio/v1"
  xmlns:contato="http://distribuidora.com.br/soa/dados/Contato/v1"
  xmlns:doc="http://distribuidora.com.br/soa/dados/Documento/v1"
  xmlns:end="http://distribuidora.com.br/soa/dados/Endereco/v1"
  xmlns:pes="http://distribuidora.com.br/soa/dados/Pessoa/v1"
  xmlns:pesF="http://distribuidora.com.br/soa/dados/PessoaFisica/v1"
  xmlns:pesJ="http://distribuidora.com.br/soa/dados/PessoaJuridica/v1"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:Q2="http://distribuidora.com.br/soa/dados/SegmentoMercado/v1">
  <xsd:import schemaLocation="SegmentoMercado.xsd"
  namespace="http://distribuidora.com.br/soa/dados/SegmentoMercado/v1">
```

```
14    </xsd:import>
15    <xsd:import namespace="http://distribuidora.com.br/soa/dados/Endereco/v1"
16    schemaLocation="Endereco.xsd"/>
17    <xsd:import namespace="http://distribuidora.com.br/soa/dados/Contato/v1"
18    schemaLocation="Contato.xsd"/>
19    <xsd:import
20    namespace="http://distribuidora.com.br/soa/dados/Documento/v1"
21    schemaLocation="DocumentoFiscal.xsd"/>
22    <xsd:import
23    namespace="http://distribuidora.com.br/soa/dados/Pessoa/v1"
24    schemaLocation="Pessoa.xsd"/>
25    <xsd:import
26    namespace="http://distribuidora.com.br/soa/dados/PessoaFisica/v1"
27    schemaLocation="PessoaFisica.xsd"/>
28    <xsd:import
29    namespace="http://distribuidora.com.br/soa/dados/PessoaJuridica/v1"
30    schemaLocation="PessoaJuridica.xsd"/>
31    <xsd:complexType name="ParceiroNegocio">
32      <xsd:sequence>
33        <xsd:element minOccurs="1" name="pessoa" type="pes:Pessoa"/>
34        <xsd:element name="identificador" type="xsd:string"/>
35        <xsd:element name="telefone" type="xsd:string"/>
36        <xsd:element name="email" type="xsd:string"/>
37        <xsd:element name="identificadorRedeSocial" type="xsd:string"/>
38        <xsd:element name="categoria" type="xsd:string"/>
39        <xsd:element name="tipo" type="xsd:string"/>
40        <xsd:element name="segmentoMercado" type="Q2:SegmentoMercado"/>
41        <xsd:element minOccurs="0" name="endereco"
42        type="end:Endereco"/>
43        <xsd:element maxOccurs="unbounded" minOccurs="1" name="contato"
44        type="contato:Contato"/>
45      </xsd:sequence>
46    </xsd:complexType>
47  </xsd:schema>
```

O produto final deste processo são as definições de esquemas: *XSD*.

4.3 Serviços Candidatos

Uma vez estruturado o modelo de dados utilizando a *CIM-IEC61968*, foram levantados os serviços candidatos também utilizando como base este mesmo *CIM*.

Para definição dos serviços de negócio candidatos, foi então abordado uma metodologia *top-down*, onde após analisar cada processo de negócio e desses processos extrair os serviços candidatos e por meio de um contínuo processo de refatoramento, cada serviço foi quebrado em serviços menores até o momento em que este serviço ainda fazia sentido para o negócio, este

processo de refatoração deverá possibilitar maior reaproveitamento de código e possibilitar a criação de novos serviços por meio de composição.

Após a definição de um serviço candidato o próximo passo é a definição da interface desse serviço. Neste processo não deve ser considerado detalhes da implementação, somente devem ser definidas as operações, entradas e saídas a partir dos termos utilizados pelo negócio.

4.3.1 Serviço Identificar Parceiro de Negócio

- Para parceiro de negócio foi neste momento definida uma operação chamada de identificaParceiroNegocio que é análoga ao momento em que um consumidor é consultado em um sistema de gestão de consumidor. No documento a seguir percebe-se a definição dos parâmetros de *request* e *response*, onde no primeiro, consulta, são definidos os parâmetros de entrada e no segundo, retorno, os objetos esperados de saída.

- Nos parâmetros de entrada foram definidos os campos em que um atendente costuma utilizar para consulta como, por exemplo, CPF, CNPJ, número do cliente, medidor de energia ou instalação e *e-mail*, em caso em que o atendente é uma *URA* pode ser utilizado o número de telefone.

- Percebe-se que foi definido um retorno que é um elemento construído no passo anterior, Parceiro de Negócio.

- Outro elemento de retorno é um objeto do tipo *fault* que é utilizado para levantar exceção de sistema de tipo específico de *fault* pertencente a Parceiro de Negócio chamado de identificarParceiroNegocioFalha.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema
3  targetNamespace="http://distribuidora.com.br/soa/negocio/ParceiroNegocio/"
4       xmlns:Q1="http://distribuidora.com.br/soa/dados/ParceiroNegocio/v1"
5       xmlns:ibmSchExtn="http://www.ibm.com/schema/extensions"
6       xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
7       xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
8       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
9       xmlns:Q2="http://distribuidora.com.br/soa/dados/Protocolo/v1"
10      xmlns:Q3="http://distribuidora.com.br/soa/dados/Falha/v1">
11      <xsd:import schemaLocation="../../xsd/Falha.xsd"
12      namespace="http://distribuidora.com.br/soa/dados/Falha/v1">
13      </xsd:import>
14      <xsd:import schemaLocation="../../xsd/Protocolo.xsd"
15      namespace="http://distribuidora.com.br/soa/dados/Protocolo/v1">
16      </xsd:import>
17      <xsd:import schemaLocation="../../xsd/ParceiroNegocio.xsd"
18      namespace="http://distribuidora.com.br/soa/dados/ParceiroNegocio/v1">
```

```
19      </xsd:import>
20      <xsd:element ibmSchExtn:docRoot="true"
21      name="identificacaoParceiroNegocioRequest">
22          <xsd:complexType>
23              <xsd:sequence>
24                  <xsd:element minOccurs="0"
25                  name="canalAtendimento" type="xsd:string" />
26                  <xsd:element name="CPF" type="xsd:string" />
27                  <xsd:element name="CNPJ" type="xsd:string" />
28                  <xsd:element name="identificadorParceiroNegocio"
29                  type="xsd:string" />
30                  <xsd:element name="identificadorInstalacao"
31                  type="xsd:string" />
32                  <xsd:element name="identificadorMedidor"
33                  type="xsd:string" />
34                  <xsd:element name="identificadorRedeSocial"
35                  type="xsd:string" />
36                  <xsd:element name="email" type="xsd:string" />
37                  <xsd:element name="telefone" type="xsd:string" />
38              </xsd:sequence>
39          </xsd:complexType>
40      </xsd:element>
41      <xsd:element ibmSchExtn:docRoot="true"
42      name="identificacaoParceiroNegocioResponse">
43          <xsd:complexType>
44              <xsd:sequence>
45                  <xsd:element minOccurs="0"
46                  name="protocoloPai" type="Q2:Protocolo" />
47                  <xsd:element
48                  name="parceiroNegocio" type="Q1:ParceiroNegocio" />
49              </xsd:sequence>
50          </xsd:complexType>
51      </xsd:element>
52      <xsd:element name="identificarParceiroNegocioFalha">
53          <xsd:complexType>
54              <xsd:sequence>
55                  <xsd:element
56            name="falha" type="Q3:Falha"></xsd:element>
57              </xsd:sequence>
58          </xsd:complexType>
59      </xsd:element>
60  </xsd:schema>
```

- A partir da definição das entradas e saída a construção da operação pode ser executada por meio da definição do que é enviado no *request* e devolvido no *response* e os objetos de *fault*.

- Também é definido o endereço de chamada do serviço *Web* por meio do *address location*.
- A operação identificaParceiroNegocio foi atribuída ao objeto ParceiroNegocio, análogo a uma operação e a definição de uma classe em orientação por objeto.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="ParceiroNegocio"
targetNamespace="http://distribuidora.com.br/soa/negocio/ParceiroNegocio/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://distribuidora.com.br/soa/negocio/ParceiroNegocio/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:documentation>
        <wsdl:appinfo source="WMQI_APPINFO">
            <MRWSDLAppInfo imported="true">
                <generatedXSD
                location="ParceiroNegocio_InlineSchema1.xsd" />
                <binding hasEncoding="false"
                imported="true" name="ParceiroNegocioSOAP"
          originalBindingStyle="document" />
            </MRWSDLAppInfo>
        </wsdl:appinfo>
    </wsdl:documentation>
    <wsdl:types>
        <xsd:schema
        targetNamespace=
        "http://distribuidora.com.br/soa/negocio/ParceiroNegocio/"
            xmlns:Q1=
            "http://distribuidora.com.br/soa/dados/ParceiroNegocio/v1"
            xmlns:ibmSchExtn="http://www.ibm.com/schema/extensions">
            <xsd:include
            schemaLocation="ParceiroNegocio_InlineSchema1.xsd" />
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="identificarParceiroNegocioRequest">
        <wsdl:part element="tns:identificacaoParceiroNegocioRequest"
      name="parameters" />
    </wsdl:message>
    <wsdl:message name="identificarParceiroNegocioResponse">
        <wsdl:part element="tns:identificacaoParceiroNegocioResponse"
      name="parameters" />
    </wsdl:message>
    <wsdl:message name="identificarParceiroNegocioFalha">
        <wsdl:part name="parameters"
        element="tns:identificarParceiroNegocioFalha"></wsdl:part>
    </wsdl:message>
    <wsdl:portType name="ParceiroNegocioPortType">
```

```
43          <wsdl:operation name="identificarParceiroNegocio">
44              <wsdl:input message="tns:identificarParceiroNegocioRequest" />
45              <wsdl:output
46              message="tns:identificarParceiroNegocioResponse" />
47              <wsdl:fault name="ParceiroNegocioFalha"
48              message="tns:identificarParceiroNegocioFalha"></wsdl:fault>
49          </wsdl:operation>
50      </wsdl:portType>
51      <wsdl:binding name="ParceiroNegocioSOAP"
52      type="tns:ParceiroNegocioPortType">
53          <soap:binding style="document"
54        transport="http://schemas.xmlsoap.org/soap/http" />
55          <wsdl:operation name="identificarParceiroNegocio">
56              <soap:operation
57          soapAction=
58          "http://distribuidora.com.br/soa/negocio
59 ␣␣␣␣␣␣␣␣/parceiroNegocio/identificarParceiroNegocio" />
60              <wsdl:input>
61                  <soap:body use="literal" />
62              </wsdl:input>
63              <wsdl:output>
64                  <soap:body use="literal" />
65              </wsdl:output>
66              <wsdl:fault name="ParceiroNegocioFalha">
67                  <soap:fault name="ParceiroNegocioFalha" use="literal" />
68              </wsdl:fault>
69          </wsdl:operation>
70      </wsdl:binding>
71      <wsdl:service name="ParceiroNegocio">
72          <wsdl:port binding="tns:ParceiroNegocioSOAP"
73          name="ParceiroNegocioBinding">
74              <soap:address
75          location=
76          "http://localhost:7800/negocio/comercial
77 ␣␣␣␣␣␣␣␣/parceiroNegocio/parceiroNegocioService" />
78          </wsdl:port>
79      </wsdl:service>
80  </wsdl:definitions>
```

4.3.2 Serviço Listar Débitos

- Foi definida uma operação chamada de Listar Débito que é o serviço em que um consumidor solicita ao atendente as faturas que devem ser pagas. Se existem faturas que devem ser pagas, devem existir outras faturas que ou foram pagas, ou não devem ser pagas.

- Este serviço foi um serviço candidato a ser refatorado em outros serviços e um desse

serviço é o Histórico de Faturamento.

- Uma vez definido o serviço Histórico de Faturamento que deverá buscar todas as faturas no sistema de faturamento, o serviço Listar Débito deverá consumir e tratar a listagem original. Para definir qual o requisito uma fatura deve possuir para ser um débito foram construídas no repositório de regras de negócio estas definições:
  - Regra 1 - Débito é toda fatura que não possui data de pagamento.
  - Regra 2 - Débito não pode possuir data de vencimento com mais de 5 anos.
  - Regra 3 - Débito não pode estar em cobrança judicial.
- Estas regras e outras foram definidas e inseridas no repositório de regras para uso futuro na implementação deste serviço.
- Então os elementos de entrada, saída e falha são definidos e a operação Listar Débito é adicionada ao Serviço junto com as demais operações descobertas como, por exemplo, Histórico de Faturamento que será utilizado pelo Listar Débito por meio da orquestração de serviço.

O processo de criação de Serviços *web*, para um projeto *SOA*, devem respeitar os seguintes requisitos:

- Reúso: todo serviço e operação devem ser pensados em seu reúso. Um serviço ou operação deve ser refatorado à exaustão para que novos serviços e operações possam ser criados e reaproveitados.
- Possuir um contrato formal de serviço: por meio da construção do WSDL e XSD, bem como a definição de seus limites e funcionamento.
- Baixo acoplamento e alta coesão: uma operação deve ser atômica e resolver uma ação de negócio.
- Abstração da Lógica de Funcionamento: a entrada das operações de serviços e a definição de seus objetos devem respeitar o modelo de negócio, jamais uma operação ou atributo tecnológico deve ser exposto na definição do contrato.
- Composição: os serviços devem ser capazes de ser agregados e combinados para resolver um problema maior formando um serviço composto.
- Autonomia: os serviços controlam a sua lógica, devem ser independentes.
- *Stateless*: os serviços devem ser atômicos, devem iniciar uma requisição, processar, responder e terminar, não devem persistir seu estado de funcionamento. É como no funcionamento do protocolo *HTTP*, ele não tem estado de funcionamento.

Os processos de criação dessas interfaces e dos modelos de dados são processos de criação, onde é fundamental a participação de um arquiteto de dados junto ao arquiteto de sistemas. Uma definição errada nesta etapa dificilmente é contornada durante o projeto, causando prejuízo e retrabalho.

A criação das funções e dos serviços devem sempre respeitar o modelo de dados definido e cada serviço deve responder positivamente as seguintes perguntas:

- O serviço é uma função do negócio?
- O serviço é independente?
- Posso reutilizar? Não preciso refatorar?
- Se eu entregar o serviço para um cliente, ele é capaz de entender e utilizar?

O produto final deste processo são as definições dos serviços *Web*: *WSDL*, que junto aos *XSD* possibilitam a criação de *Stubs* no sistema consumidor.

4.4 Desenvolvimento do Software

Algumas restrições foram impostas ao projeto durante a execução como, por exemplo:

- Nenhuma função dos sistemas legados poderiam ser alteradas e nenhuma nova solicitada por este projeto.
- Nenhuma função dos sistemas legados poderiam ser executadas diretamente via chamada remota de procedimento. Esta função seria de responsabilidade dos serviços *Web* existentes.
- As funções expostas dos sistemas legados poderiam ser alteradas devido ao projeto de fusão de processos de negócio e de *software* que a empresa enfrentava no momento desse projeto.

Para evitar os prováveis impactos dos riscos impostos pelas restrições citadas acima, algumas decisões de projeto foram tomadas:

- A decisão fundamental foi a definição das interfaces dos serviços tomadas seguindo a filosofia da Arquitetura Orientada a Serviços: as interfaces refletem o negócio e não a implementação do sistema, toda transformação para adequação aos sistemas legados seriam de responsabilidade da implementação do serviço *Web*.
- Outra importante decisão foi a definição do modelo canônico de dados e a organização do projeto baseado neste modelo de dados criou uma camada que segregou a implementação dos legados, dos novos serviços de negócio desenvolvidos.

- Outro decisão que trouxe enorme benefício foi o constante refatoramento tanto de funções de negócio quanto de TI que possibilitou a fácil remoção, substituição e inclusão de novos comportamento e funcionalidades, por exemplo, para trocar um componente de assinatura digital, ou de armazenamento de documentos, basta alterar o componente responsável pelo processo.
- Devido a diversidade de padrões nas aplicações legadas foi definido um padrão corporativo para os formatos de dados que seriam adotados pelo projeto, por exemplo, as datas teriam o padrão de brasileiro definido por DD/MM/AAAA.
- Toda complexidade interna e de regra de negócio seriam ocultadas pelas novas interfaces, expondo somente informações relevantes de negócio.
- Maximizar o paralelismo nas execuções das tarefas orquestradas e transformar o máximo de operações possíveis em operações assíncronas para poupar recursos computacionais. Alguns serviços internos de negócio, que não tem influência direta no processo solicitado foram retirados do fluxo principal de execução, com isso consegue-se diminuir o tempo de processamento.

4.4.1 Serviço Identificar Parceiro de Negócio

A função usada para identificar o parceiro de negócio nesse projeto é a mesma função já utilizada nos atuais canais de atendimento, é um serviço *web* do tipo faz tudo, nessa função é possível encontrar vários objetos do modelo de dados como, por exemplo, Parceiro de Negócio, Conta Contrato, Contrato, Instalação, Fatura, Contato, entre outros. Para possibilitar uma comparação, a operação atual possui 68 parâmetros de entrada e a nova operação apenas 8 parâmetros e cada um desses parâmetros possibilitam a identificação de apenas um parceiro.

Outro ponto importante é que cada atendimento deve ser identificado por um protocolo e cada protocolo possui um ou mais protocolos auxiliares. Estes protocolos auxiliares possuem função apenas interna, para medição e estatística do negócio e não são funções do negócio. No atual modelo a criação do protocolo, protocolo auxiliar e do serviço de negócio eram expostos para o consumidor da função. O resultado disso é que cada desenvolvedor interpreta e executa de maneira distinta estas operações, por exemplo, as Figuras 7 e 8 ilustram duas implementações distintas para o mesmo requisito.

Os modelos citados pelas Figuras 7 e 8 expõem a complexidade interna da execução de uma operação que deveria ser atômica. No modelo da Figura 7, se a aplicação encerrar antes do serviço de identificação do parceiro tem-se um protocolo aberto sem a execução do serviço e na Figura 8, se a aplicação encerrar antes da execução do serviço de identificação do parceiro o problema ocorre de maneira diferente, tem-se um protocolo encerrado sem a execução do serviço. Vários outros problemas podem ocorrer com esta exposição e para corrigir na implementação *SOA* sugerida, veja o fluxo mostrado na Figura 9.

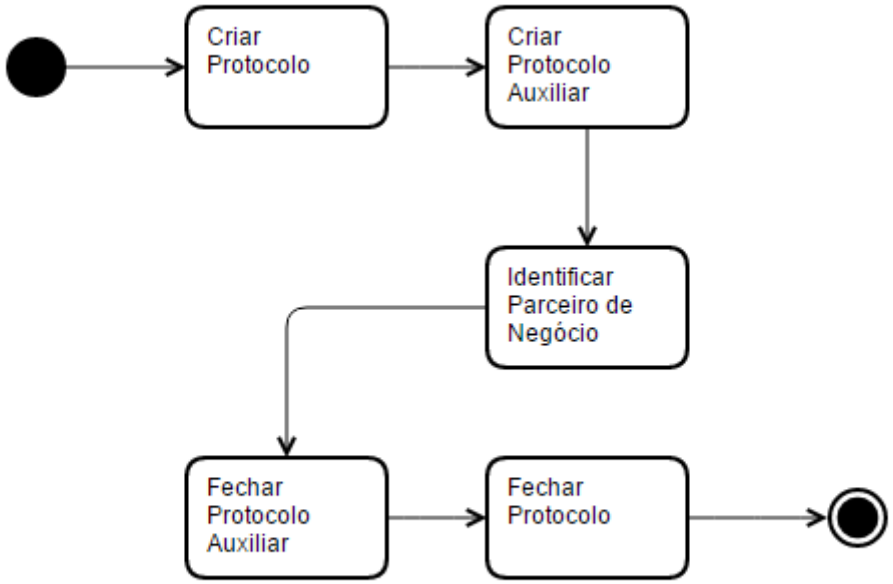

Figura 7 – Implementação de Identificar Parceiro de Negócio do portal *web*.

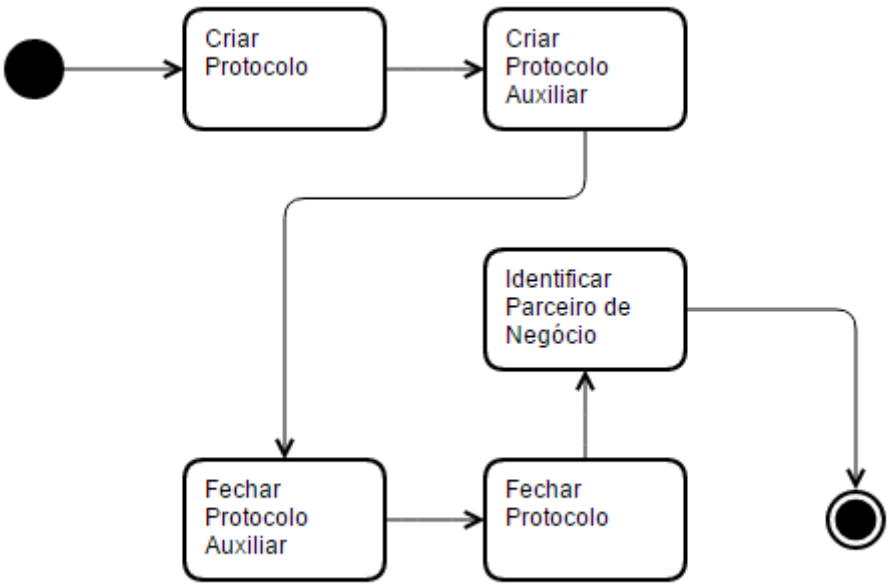

Figura 8 – Implementação de Identificar Parceiro de Negócio em aplicativos móveis.

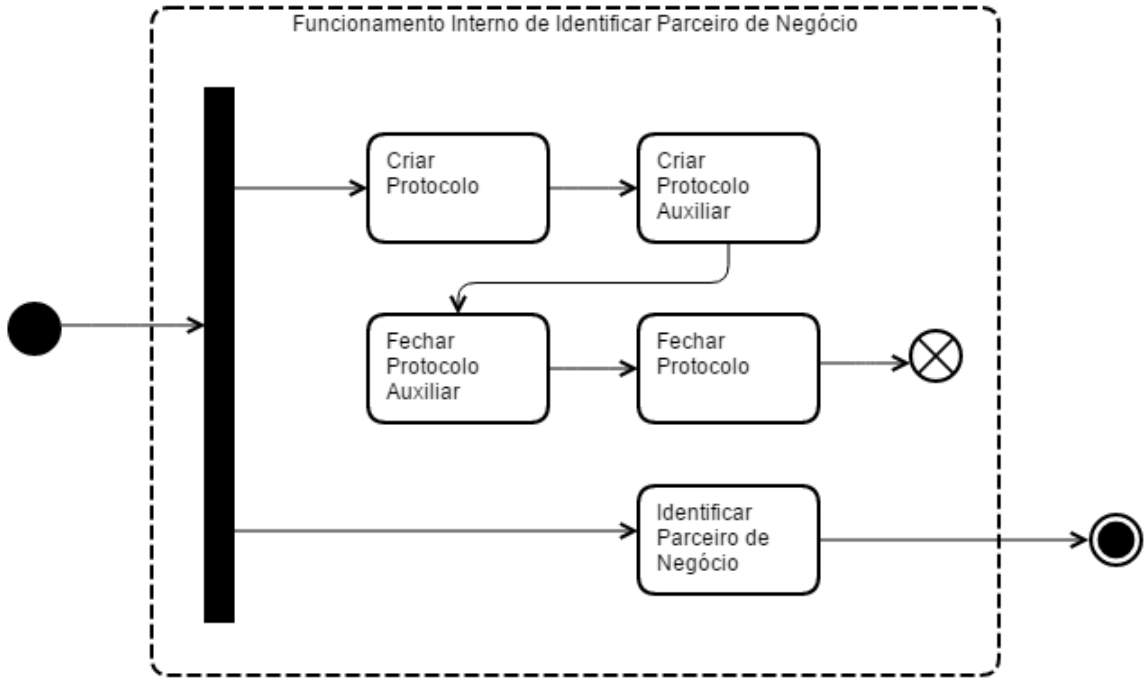

Figura 9 – Nova implementação de Identificar Parceiro de Negócio.

No novo fluxo, observa-se que o cliente deverá executar somente a operação de negócio Identificar Parceiro de Negócio, é uma função atômica, esta função será responsável internamente por toda a orquestração de funções do processo de negócio. Também é possível notar o paralelismo, há dois fluxos de execução, um responsável pelo processo do protocolo e o outro pela execução do serviço de negócio, para o cliente do serviço o tempo de execução agora é o tempo da operação principal e não mais das 5 operações citadas.

- Exemplo de envelope *SOAP* usado para requisição da operação de identificacaoParceiroNegocio do serviço de ParceiroNegocio:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:par="http://distribuidora.com.br/soa/negocio/ParceiroNegocio/">
  <soapenv:Header/>
  <soapenv:Body>
    <par:identificacaoParceiroNegocioRequest>
      <CPF></CPF>
      <CNPJ></CNPJ>
      <identificadorParceiroNegocio></identificadorParceiroNegocio>
      <identificadorInstalacao></identificadorInstalacao>
      <identificadorMedidor></identificadorMedidor>
      <identificadorRedeSocial></identificadorRedeSocial>
```

```
13        <email></email>
14        <telefone></telefone>
15      </par:identificacaoParceiroNegocioRequest>
16    </soapenv:Body>
17  </soapenv:Envelope>
```

- Exemplo de envelope *SOAP* de retorno da operação de identificacaoParceiroNegocio do serviço de ParceiroNegocio. É possível observar os objetos do modelo de dados como, por exemplo, Parceiro de Negócio com Pessoa Física, Endereço e outros.

```
1  <soapenv:Envelope
2    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
3    <soapenv:Body>
4      <io3:identificacaoParceiroNegocioResponse
5        xmlns:io3="http://distribuidora.com.br/soa/negocio/ParceiroNegocio/"
6        xmlns:in="http://distribuidora.com.br/soa/dados/PessoaFisica/v1"
7        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
8        <parceiroNegocio>
9          <pessoa xsi:type="in:PessoaFisica">
10           <nome></nome>
11           <sobrenome></sobrenome>
12           <CPF></CPF>
13         </pessoa>
14         <identificador></identificador>
15         <telefone></telefone>
16         <email></email>
17         <identificadorRedeSocial/>
18         <categoria/>
19         <tipo/>
20         <segmentoMercado>
21           <tipo></tipo>
22           <descricao></descricao>
23         </segmentoMercado>
24         <docFiscal>
25           <item>
26             <identificador></identificador>
27             <tipo></tipo>
28             <descricao></descricao>
29           </item>
30         </docFiscal>
31         <endereco>
32           <logradouro>
33             <nome></nome>
34           </logradouro>
35           <numero></numero>
36           <pais></pais>
```

```
37              <bairro>
38                <nome></nome>
39              </bairro>
40              <municipio>
41                <nome></nome>
42              </municipio>
43              <seguimento>
44                <cep></cep>
45              </seguimento>
46            </endereco>
47          </parceiroNegocio>
48        </io3:identificacaoParceiroNegocioResponse>
49    </soapenv:Body>
50  </soapenv:Envelope>
```

- Parte do envelope *SOAP* de retorno da função de identificação de parceiro. Observe que vários objetos não fazem parte da entidade Parceiro de Negócio como, por exemplo, instalação, contrato, faturas, entre outras, além da falta de padronização e significância dos nomes dos atributos como, por exemplo, BPKIND, GPART, VTREF, OPBEL e WAERS.

```
1  <soapenv:Envelope
2    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
3    <soapenv:Body>
4      <out:identParceiroResponse
5        xmlns:out="http://distribuidora.com.br/soa/negocio/IdentParceiro/">
6        <Y_PARCEIRO>
7          <PARTNER></PARTNER>
8          <TYPE></TYPE>
9          <BPKIND></BPKIND>
10         <NAME_LAST></NAME_LAST>
11         <NAME_FIRST></NAME_FIRST>
12         <STREET></STREET>
13         <HOUSE_NUM></HOUSE_NUM>
14                  (...)
15       </Y_PARCEIRO>
16       <YT_INSTALACAO/>
17       <YT_ENDERECO/>
18       <Y_ERRO></Y_ERRO>
19       <Y_DOC_ATUALIZADO></Y_DOC_ATUALIZADO>
20       <Y_NUM_DEBITOS></Y_NUM_DEBITOS>
21       <YT_DETALHES_DEBITOS>
22         <item>
23           <GPART></GPART>
24           <VKONT></VKONT>
25           <VTREF></VTREF>
26           <ANLAGE></ANLAGE>
```

```
27              <OPBEL></OPBEL>
28              <ZIMPRES></ZIMPRES>
29              <WAERS></WAERS>
30              <FAEDN></FAEDN>
31              <BETRW></BETRW>
32            </item>
33          </YT_DETALHES_DEBITOS>
34          <YT_DOC_FISCAL/>
35          <YT_DOC/>
36          <YT_NOTAS/>
37          <YT_DETALHES_CC/>
38          (...)
39      </out:identParceiroResponse>
40    </soapenv:Body>
41  </soapenv:Envelope>
```

4.4.2 Serviço Listar Débitos

A operação listar débito do serviço de faturas é um exemplo de refatoração e reúso de código, esta função foi refatorada em listar débito e outra função genérica de listagem de fatura. A operação Listar Faturas é uma operação que recupera todas as faturas de uma conta contrato, enquanto a operação Listar Débitos retorna somente as faturas que podem ser pagas. Observa-se na Figura 10 o reúso da função Listar Faturas, além do processamento da informação externa ao barramento, por meio de um serviço de regra de negócio.

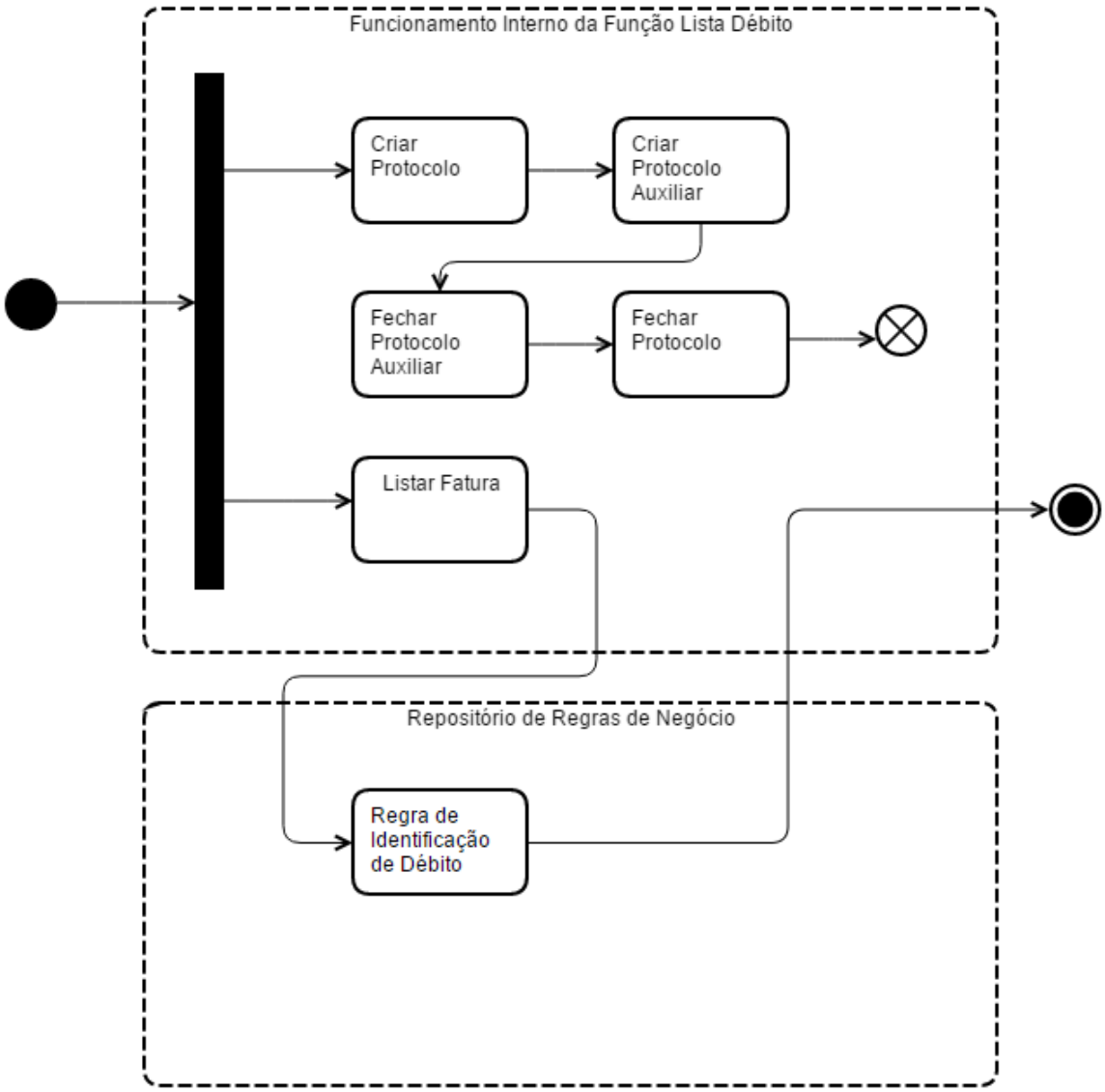

Figura 10 – Implementação *SOA* para operação Listar Débito. Observe que esta operação utiliza a orquestração de uma outra função de negócio que é Listar Fatura.

- No envelope *SOAP* a seguir percebe-se na definição da consulta que o contrato definiu, conforme o modelo de dados, que para listagem de débito, que é uma entidade dependente de um contrato e uma conta contrato, estes dois atributos devem ser informados.

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2 xmlns:deb="http://distribuidora.com.br/soa/negocio/Debito/">
3  <soapenv:Header/>
4  <soapenv:Body>
5   <deb:listaDebitosRequest>
6    <identificadorContrato></identificadorContrato>
7    <identificadorContaContrato></identificadorContaContrato>
```

```
8    </deb:listaDebitosRequest>
9   </soapenv:Body>
10 </soapenv:Envelope>
```

- Abaixo, o envelope *SOAP* de retorno da consulta de débito com um objeto tipo lista contendo débitos.

```
1  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
2   <soapenv:Body>
3    <out24:listaDebitosResponse
4     xmlns:out24="http://distribuidora.com.br/soa/negocio/Debito/">
5     <debitos>
6      <documentoContaContratual></documentoContaContratual>
7      <documentoImpressao></documentoImpressao>
8      <codigoMoeda></codigoMoeda>
9      <dataVencimento></dataVencimento>
10     <valor></valor>
11     <contaContrato>
12      <identificador></identificador>
13     </contaContrato>
14     <contrato>
15      <identificador></identificador>
16     </contrato>
17    </debitos>
18   </out24:listaDebitosResponse>
19  </soapenv:Body>
20 </soapenv:Envelope>
```

4.5 Conclusão

Neste capítulo foi apresentado o processo criativo de construção de um projeto de *software* utilizando a Arquitetura Orientada a Serviços. Da definição de um modelo canônico de dados à descoberta de serviços e da construção dos contratos de serviços *web* até seu funcionamento interno.

Foram também apresentados alguns artefatos gerados e as decisões de projetos em cada etapa do processo de construção:

- Modelo de Dados: artefato que define a ontologia do negócio de distribuição de energia elétrica, seus componentes, suas descrições e suas operações.
- Lista de serviços candidatos: lista de serviços que fazem parte da descrição do negócio. Estes serviços são definidos e priorizados sempre junto ao negócio.

- Contratos de Serviços: definição das estruturas de dados e operações, em forma de documentos *WSDL* e *XSD*, os quais mapeiam as classes definidas pelo modelo de dados e as operações descobertas na lista de serviços candidatos.

A lista acima define o alicerce do *sofware* proposto, restando apenas a sua codificação.

No próximo capítulo será apresentada a conclusão deste trabalho.

5 Conclusão

Este trabalho mostrou que a Arquitetura Orientada a Serviços não incluiu nenhuma inovação tecnológica ou um novo paradigma na Ciência da Computação. A Orientação a Serviços é uma abordagem arquitetural que enfatiza a construção de operações e objetos relacionados ao negócio.

O uso do *HTTP, XML, SOAP, WSDL* e do *UDDI*, e também da aquisição de produtos não garantem uma Arquitetura Orientada a Serviços, assim como o uso de Classes não garantem por Orientação a Objetos.

Este trabalho também mostrou como a definição de um modelo de dados, dos contratos de serviços e de suas operações em concordância com o negócio são os caminhos que devem ser trilhados para conseguir o objetivo de construir um arcabouço de serviços *SOA*.

Assim como na Orientação por Objetos, o constante refatoramento e amadurecimento dos serviços e operações garantem o reúso e o retorno do investimento nos produtos de TI.

A criação dos contratos de serviços garantiu a baixa dependência de fornecedores de plataformas. Por ser aberto e definido como padrão para desenvolvimento *SOA*, o uso do *XML, SOAP, WSDL, XSD* e *UDDI* permite definir na fronteira, os contratos dos serviços permitindo a qualquer momento a substituição de um produto de implementação *SOA*, ou mesmo dos consumidores, desde que se implemente seguindo os contratos definidos.

Na opinião do autor, a definição dos contratos utilizando os conceitos e o vocabulário do negócio, facilitou o desenvolvimento por terceiros de novas aplicações de maneira menos traumática já que este desenvolvimento foi independente da tecnologia dos sistemas legados, diminuindo o esforço de entendimento.

O uso de serviços *web* e do *HTTP*, na opinião do autor, facilitou o acesso e o reúso de *software*, em um mundo *Web* é mais natural e compreensível o reúso via da *Web*.

O uso desta arquitetura também mostrou que é aderente à integração com sistemas de *BPM* e que em alguns processos de negócio foi possível o reaproveitamento de operações construídas neste projeto, automatizando partes deste.

Alguns pontos negativos, que poderiam influenciar negativamente o projeto, foram encontrados nas definições de processos de negócio. Quando um processo é mal definido ou não parece correto, pode sugerir a criação de serviços e operações ruins. Se no momento da especificação de um serviço, e não for possível responder positivamente as perguntas a seguir, é porquê este processo deve ser revisto antes do desenvolvimento do serviço:

- O serviço é uma função do negócio?

- O serviço é independente?
- Posso reutilizar?
- Consigo entender o porquê deste serviço?

Estes questionamentos devem ser feitos para o arquiteto de serviço e o cliente e nenhuma dúvida pode ficar sem resposta, uma vez que pode resultar em um serviço ruim e afetar negativamente um projeto.

5.1 Trabalhos Futuros

No processo de desenvolvimento deste projeto, alguns desafios foram resolvidos e certamente algumas icógnitas ainda pairam no ar, por exemplo:

- O controle de acesso aos serviços e dados, e a segurança dos serviços.
- O retorno de investimento deste projeto.
- O uso da tecnologia *REST*.
- O processo de publicação, descoberta e monetização de serviçõs.
- Sobre a taxisonomia de serviços.
- Processo de descoberta e mapeamento de novos serviços.
- Governança de *SOA*.

Estes são alguns dos itens que merecem cuidado especial e estudo mais aprofundado.

Referências

ALFéREZ GERM'N H E PELECHANO, V. Systematic reuse of web services through software product line engineering. In: IEEE. *Web Services (ECOWS), 2011 Ninth IEEE European Conference on*. [S.l.], 2011. p. 192–199. Citado na página 36.

ARSANJANI, A. et al. The soa manifesto. *SOA Manifesto, October*, 2009. Citado 2 vezes nas páginas 35 e 36.

BERNERS-LEE, T.; FIELDING, R.; FRYSTYK, H. *Hypertext transfer protocol–HTTP/1.0*. [S.l.]: May, 1996. Citado na página 25.

CERAMI, E. *Web services essentials: distributed applications with XML-RPC, SOAP, UDDI & WSDL*. [S.l.]: "O'Reilly Media, Inc.", 2002. Citado na página 24.

CHRISTENSEN, E. et al. *Web services description language (WSDL) 1.1*. 2001. Citado na página 28.

COMMITTEE, O. U. S. T. et al. *UDDI 101*. Citado 2 vezes nas páginas 34 e 35.

CROCKFORD, D. *The application/json media type for javascript object notation (json)*. 2006. Citado na página 26.

CURBERA, F. et al. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, IEEE Computer Society, v. 6, n. 2, p. 86, 2002. Citado na página 26.

ERL, T. *Service-oriented architecture: concepts, technology, and design*. [S.l.]: Pearson Education India, 2005. Citado na página 35.

ERL, T. Soa: Princípios de design de serviços. *Pearson Prentice Hall, São Paulo*, v. 200, p. 18–21, 2009. Citado na página 36.

FIELDING, R. et al. *Hypertext transfer protocol–HTTP/1.1*. [S.l.]: RFC 2616, June, 1999. Citado na página 25.

FOWLER, M. *Patterns of enterprise application architecture*. [S.l.]: Addison-Wesley Longman Publishing Co., Inc., 2002. Citado na página 24.

GARTNER. *IT Glossary: SOA*. 2014. Disponível em: <http://www.gartner.com/it-glossary/service-oriented-architecture-soa>. Acesso em: 15 mai 2016. Citado na página 35.

HORSTMANN, M.; KIRTLAND, M. Dcom architecture. *Microsoft Corporation, July*, 1997. Citado na página 23.

IBM. *What is SOA?* 2016. Disponível em: <http://www-01.ibm.com/software/solutions/soa/what-is-soa.html>. Acesso em: 15 mai 2016. Citado na página 35.

KRAFZIG, D. et al. *Service-Oriented Architecture Best Practices (The Coad Series)*. [S.l.]: Prentice Hall PTR, Upper Saddle River, NJ, 2004. Citado na página 24.

MACKENZIE, C. M. et al. Reference model for service oriented architecture 1.0. *OASIS standard*, v. 12, 2006. Citado na página 35.

MICROSOFT. *RPC Technical Reference*. 2003. Disponível em: <http://technet.microsoft.com/en-us/library/cc759499.aspx>. Acesso em: 15 mai 2016. Citado na página 23.

MICROSOFT. *O que é SOA e Processos de Negócios (BP)?* 2016. Disponível em: <https://www.microsoft.com/brasil/servidores/soa/about/>. Acesso em: 15 mai 2016. Citado na página 35.

OLIVEIRA[1], J. A.; RAMOS, F. G.; JUNIOR, J. J. L. D. Reusabilidade em soa: Um mapeamento sistemático da literatura. Citado na página 36.

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 17, n. 4, p. 40–52, 1992. Citado na página 24.

USLAR, M. et al. *The Common Information Model CIM: IEC 61968/61970 and 62325-A practical introduction to the CIM*. [S.l.]: Springer Science & Business Media, 2012. Citado na página 42.

WHITE, J. E. *RFC 707: A high-level framework for network-based resource sharing*. [S.l.]: December, 1975. Citado na página 23.

YERGEAU, F. et al. Extensible markup language (xml) 1.0. *W3C Recommendation, third edition, February*, 2004. Citado na página 25.