Filipe Madeira da Silva

Trabalho de Conclusão de Curso: SOA - Arquitetura Orientada a Serviços

Este texto apresenta um estudo a respeito da Arquitetura Orientada a Serviços, ou SOA, como ela é conhecida.

Orientador:
Prof. Dr. Fábio Kon

Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

São Paulo, SP - Brasil

2006

Sumário

Resumo

Nas últimas décadas, a tecnologia da informação apresentou inovações marcantes.

No início, tínhamos computadores que utilizavam válvulas, ocupando uma sala inteira, e programados com o auxílio de cartões perfurados.

Agora, temos computadores que cabem na palma da mão e conceitos cada vez mais avançados para o desenvolvimento de software.

Podemos perceber que a tecnologia da informação apresenta fases bem definidas e a fase atual é marcada, sem dúvidas, pelo surgimento e popularização da Internet.

O conceito chave no momento é a comunicação dos sistemas, utilizando, principalmente, a Internet e víamos, até há pouco tempo, muitos dizendo que a orientação a objetos havia surgido para ser a solução definitiva para a programação voltada à Internet.

Certamente, a orientação a objetos é um conceito muito poderoso e que facilita a resolução de inúmeros problemas, no entanto, não podemos assumir que ela é suficiente para solucionar qualquer caso de desenvolvimento de software.

No momento, muito tem-se falado a respeito de um novo conceito, a orientação a serviços.

Da mesma forma que ocorreu com a orientação a objetos, há aqueles que apostam neste conceito como sendo algo definitivo. Certamente, vemos que a orientação a serviços oferece alguns conceitos interessantes e extremamente úteis, mas, a qualquer momento, poderemos nos deparar com um paradigma com mais recursos que surja para completar ou mesmo substituir a orientação a serviços

Aqueles que apoiam com entusiasmo a orientação a serviços encontram-se, principalmente, no mundo corporativo, onde um número cada vez maior de empresas passa a dedicar maiores recursos à orientação a serviços e, mais especificamente, à Arquitetura Orientada a Serviços (ou SOA, do acrônimo em inglês).

Este estudo se propõe a analisar esta nova vertente da Engenharia de Software e o impacto que o desenvolvimento e popularização deste paradigma trará ao desenvolvimento de software.

Espera-se que, ao final do texto, o leitor seja capaz de compreender exatamente o que é a arquitetura e o que ela propõe, assim como as suas vantagens e possíveis desvantagens.

O estudo é baseado principalmente na minha experiência de um ano de estágio na Accenture do Brasil que, como muitas outras grandes companhias de tecnologia da informação, tem dedicado recursos crescentes à orientação a serviços.

Durante o estágio, pude compreender no dia-a-dia como é feita a implementação de uma solução utilizando a orientação a serviços e pude participar de treinamentos que

auxiliaram enormemente neste entendimento.

O projeto do qual eu participo é responsável por implantar uma solução de grande porte para a integração de negócios da empresa de telefonia Telefônica, de São Paulo.

É um pouco desta experiência, adquirida durante este ano, que eu tento passar através deste texto que foi escrito como Trabalho de Conclusão de Curso da minha graduação em Ciência da Computação pelo Instituto de Matemática e Estatística da Univesidade de São Paulo.

Parte I

Fundamentos

1 Introdução

1.1 Fundo histórico

A medida que o tempo passa, nos deparamos com uma presença cada vez maior de sistemas de computação no nosso dia-a-dia.

Sistemas de computação simples, como um editor de textos, e sistemas complexos, como o sistema de cobrança de uma empresa de cartão de crédito estão sendo utilizados a todo momento e tornam-se cada vez mais presentes no nosso dia-a-dia.

No entanto, independente da complexidade do sistema, o ponto chave atualmente é a interação entre sistemas.

Com o crescente uso da Internet, presenciamos uma necessidade também crescente de comunicação e interação entre as partes envolvidas.

A utilização dos sistemas de computação tornou-se simples. Agora, precisamos tornar estes sistemas mais autônomos para facilitar o uso destes no dia-a-dia.

Basta ver que, se nos propusermos a analisar o que é a Rede (*Web*) atualmente, concluiremos que ela é uma fonte imensa de informação. Entretanto, tal informação está muito mal organizada.

Uma pessoa mais alheia ao mundo da computação poderia discordar de tal afirmação já que, comparado há alguns anos, encontramos informações e notícias muito mais facilmente.

Entretanto, a questão é que cada vez menos estaremos dispostos a catalogar informações na Rede e tais informações serão utilizadas de forma automatizada por processos, e não por humanos[1].

A idéia é que a Rede deixe de fornecer apenas informação pura e simples e passe a

[1] O conceito envolvido refere-se à ideia de Rede Pragmática (*Pragmatic Web*).

fornecer serviços (*Web Services*) que sejam úteis aos humanos e aos sistemas que a estão utilizando.

1.2 Orientação a Serviços

A orientação a serviços surgiu para suprir tal necessidade, unindo diversos conceitos e práticas sob um único paradigma.

Tal paradigma é conhecido como Arquitetura Orientada a Serviços ou **SOA** (do acrônimo em inglês) e, em essência, trate-se de uma paradigma criado com o objetivo desenvolver sistemas modularizados, o que traz diversos benefícios ao produto final.

Ao invés de desenvolvermos grandes aplicações como um único bloco, podemos encapsular algumas funções importantes e disponibilizá-las na forma de serviços em uma rede.

A arquitetura propõe padrões e práticas de desenvolvimento para possibilitar que os serviços (*web services*) possam interagir adequadamente em um ambiente tão heterogêneo quanto a Internet, por exemplo, ou em outra rede qualquer.

Como qualquer arquitetura baseada em componentes, SOA mantém a independência de cada um deles e define, apenas, como será feita a comunicação entre eles.

Os padrões que têm sido utilizados para coordenar esta comunicação têm surgido sob orientação do W3C e definem desde a linguagem padrão de comunicação (XML) até o modo como um serviço deve ser publicado para tornar-se acessível através da rede (UDDI). Estes e outros padrões importantes para a compreensão do funcionamento da arquitetura, como SOAP e WSDL, serão abordados com o decorrer do texto.

Parte II

A Arquitetura

2 *Características de uma arquitetura orientada a serviços*

Segundo a Accenture, SOA é:

> *"Uma arquitetura que define como funções de negócios distintas, implementadas por sistemas autônomos, devem operar conjuntamente para executar um processo de negócio."*

Assim, SOA é um paradigma de desenvolvimento de software que visa permitir que os componentes de um processo de negócio sejam integrados mais facilmente.

Um componente de um processo de negócio é uma atividade comum, algo que é realizado frequentemente naquele processo de negócio específico. Definimos esta atividade como *função de negócio*.

Assim, o objetivo é implementar um sistema que represente o negócio do cliente, dividindo este negócio em processos e estes em atividades (funções de negócio).

Algumas características importantes da arquitetura serão detalhadas nas seções a seguir.

2.1 Baixo acoplamento (*loose coupling)*

Cada atividade (função de negócio) é implementada como um serviço, ou seja, um componente independente que poderá ser utilizado quantas vezes forem necessárias em partes diversas do sistema.

Assim, vemos que SOA é, essencialmente, uma arquitetura baseada em componentes, onde cada componente preserva a sua independência e interage, apenas, através de interfaces bem definidas.

Este é o conceito conhecido como *baixo acoplamento* e que esta presente na orientação a serviços.

2.2 Neutralidade de implementação

Outro ponto importante a se ressaltar é que não há limitações prévias em relação ao modo de implementar um serviço. Não há limitações em relação às tecnologias, linguagens ou plataformas a serem utilizadas, por exemplo.

O desenvolvedor deve, apenas, especificar adequadamente o que o seu serviço faz e informações como tipos de dados de entrada e saída. Assim, outros desenvolvedores poderão fazer o uso adequado deste serviço baseado nesta interface que foi especificada e que será a única parte realmente visível do serviço.

Detalhes de implementação específicos de um componente (serviço) não são importantes para o restantes do sistema e, por isso, não devem estar disponíveis.

2.3 Interoperabilidade

Para compôr a especificação citada na seção anterior, e que servirá de interface para o serviço, um conjunto de regras e padrões devem ser seguidos. Tais padrões e regras foram desenvolvidos para especificar tudo que é importante para que um serviço torne-se acessível e utilizável através de uma rede.

Sendo assim, vemos que a arquitetura propõe uma quase completa liberdade de desenvolvimento e, com isso, alcança-se a *interoperabilidade*, que é a possibilidade de sistemas coexistirem e comunicarem-se independente de fabricantes ou tecnologias.

A interoperabilidade é uma característica muito importante pois permite produzir soluções muito mais flexíveis e de melhor qualidade já que não encontramos dificuldades de comunicação entre sistemas diversos, implementados de acordo com necessidades e limitações igualmente diversas.

3 *Exemplo de arquitetura orientada a serviços*

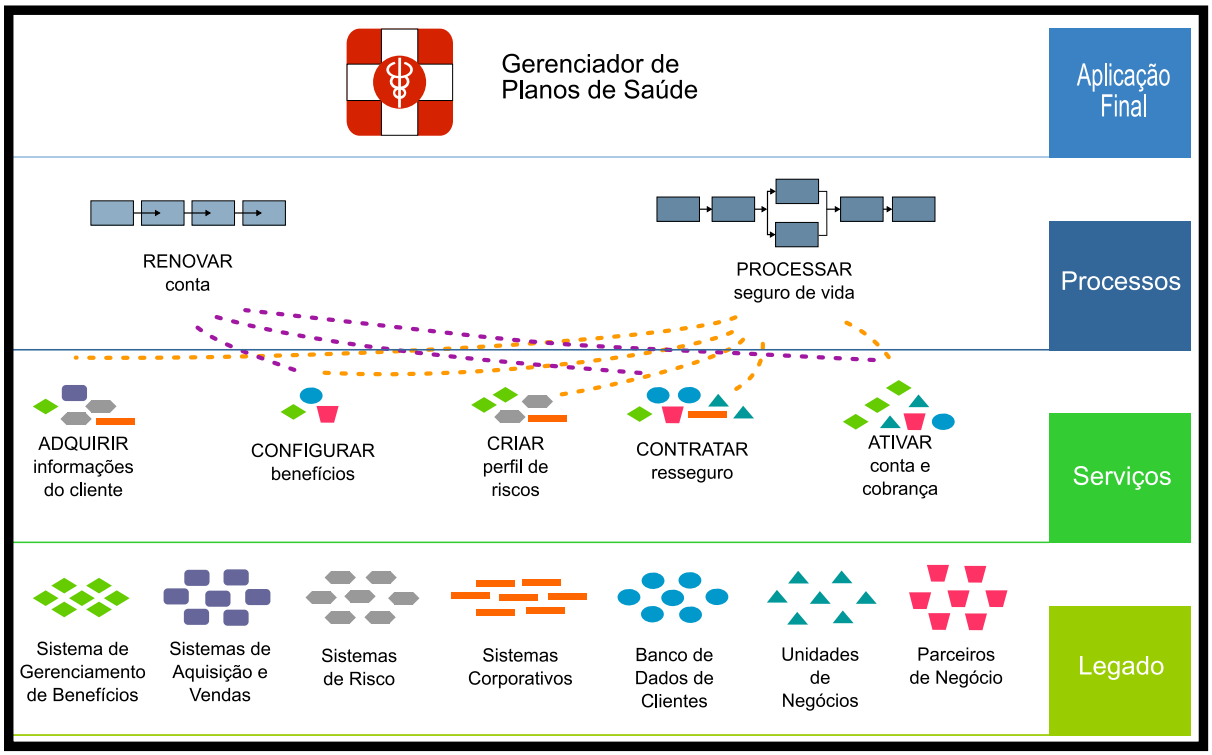

Figura 1: Exemplo de uma arquitetura baseada em serviços para o caso de um sistema gerenciador de planos de saúde.

Na figura acima, temos um exemplo simplificado do modo como um sistema gerenciador de planos de saúde poderia ser implementado utilizando os conceitos relacionados à orientação a serviços.

No nível inferior, temos todos os sistemas que o cliente atualmente utiliza. Estes sistemas são comumente chamados de *legado* e representam soluções desenvolvidas há algum tempo e que não estão, necessariamente, de acordo com a realidade atual do negócio do cliente.

Um negócio é algo extremamente dinâmico, que muda frequentemente ao longo do tempo. Já os sistemas não apresentam este dinamismo devido à forma extremamente rígida com que costumas ser desenvolvidos. Sendo assim, não há como os sistemas evoluirem de modo a acompanhar a evolução do negócio que eles se propõe a atender.

Contudo, geralmente, tais sistemas são vitais para a operação atual do negócio e o cliente, simplesmente, não tem como desfazer-se deles.

É neste contexto que aplicamos a orientação a serviços. Cada função de negócio que pretendemos disponibilizar no novo sistema será implementada como um serviço independente baseado nos sistemas já existentes.

Criamos serviços para acessarem tais sistemas e retirarem todas as informações que necessitamos. Também é comum o novo sistema passar a operar os sistemas antigos de uma forma camuflada, automatizando operações que costumavam ser feitas manualmente.

Neste exemplo, vemos que o legado é representado por estes sistemas:

- sistema de gerenciamento de benefícios;
- sistemas de aquisição e vendas;
- sistema de riscos;
- sistemas corporativos;
- banco de dados de clientes;
- unidades de negócios e
- parceiros de negócios.

Após analisar os sistemas existentes, podemos implementar as funções de negócios desejadas, que estão representadas no nível imediatamente superior ao nível do legado na figura. Neste caso, tais funções são:

- adquirir informações do cliente;
- configurar benefícios;
- criar perfil de riscos;
- contratar resseguros e

- ativar conta e cobrança.

Tendo implementado estes serviços, o próximo passo é fazer uso das funções de negócio que, agora, estão disponíveis. Tais funções são organizadas de forma a construirmos os processos de negócio desejados.

Podemos definir esta fase com uma etapa de montagem. A maior parte do trabalho já foi realizada e, agora, estamos, apenas, juntando as peças disponíveis para construirmos o conjunto esperado.

A solução final, por vezes chamada de aplicação composta e representada no nível superior da figura, é composta por um conjunto de processos de negócio.

Cada processo de negócio é montado de acordo com o metodo descrito a pouco e, no exemplo, temos dois processos, que estão representados imediatamente abaixo do nível da aplicação final. Tais processos são:

- renovar contas e
- processar seguro de vida.

4 Benefícios da orientação a serviços

Mesmo em um exemplo tão simples como o anterior pode-se notar diversos benefícios de uma arquitetura que segue o modelo proposto pela orientação a serviços.

O serviço que representa a função *configurar benefício*, por exemplo, foi utilizado em ambos os processos desenvolvidos.

Apesar de não podermos determinar o processo específico para o qual este serviço foi inicialmente desenvolvido, ou mesmo se ele foi idealizado para ser utilizado em ambos os processos, presenciamos, em qualquer caso, a reutilização de um serviço em partes diversas do sistema.

Em geral, a criação de serviços resulta em um bom índice de reaproveitamento, o que possibilita a redução de custos de projeto e desenvolvimento, e diminui o tempo total do projeto.

SOA possibilita isto, apenas, porque este é um paradigma focado, através dos serviços, nos valores e processos de negócio. A aplicação torna-se, assim, tão ágil quanto o negócio em si e qualquer mudança no negócio é facilmente transmitida para o sistema, rearranjando as funcionalidades que já estão disponíveis através dos serviços e desenvolvendo a lógica restante.

A seguir, temos o detalhamento de alguns benefícios importantes que são, geralmente, perseguidos pelas pesquisas na área de Engenharia de Software e que são alcançados quando implantamos uma solução baseada em SOA.

4.1 Reúso de código

O reúso de código é um tema recorrente no campo de pesquisa de Engenharia de Software. No entanto, o pleno reúso é difícil de ser atingido principalmente devido a

incompatibilidades de plataformas e linguagens.

Com a orientação a serviços, como vimos, não nos deparamos com tal dificuldade pois há plena interoperabilidade entre os sistemas.

Para reutilizar um serviço, precisamos, apenas, encontrá-lo (o que é feito através de um diretório de serviços, como será posteriormente explicado) e utilizá-lo.

4.2 Redução de redundâncias de funcionalidades

É comum vermos uma função de negócio importante ser implementada em um sistema e ter que ser replicada em diversos outros sistemas. Isto ocorre, principalmente, devido a incompatibilidades de plataforma e linguagens já que cada sistema foi implementado de forma autônoma em relação aos outros.

Utilizando SOA, podemos eliminar tais redundâncias pois uma única atividade implementada como um serviço poderá ser compartilhada com qualquer sistema. A atividade não é inerente a nenhum sistema e, sim, à rede na qual o serviço foi disponibilizado.

4.3 Redução do custo de manutenção

A redução do custo de manutenção se dá à mesma medida que a redução de redundâncias de funcionalidades.

Antes, se uma falha na lógica de uma função fosse detectada ou se tal função tivesse que ser alterada devido a mudanças no negócio em si, teríamos que realizar tal alteração em todos os sistemas nos quais a função estivesse presente.

Agora, com SOA, precisamos mudar, apenas, o único serviço que todas as aplicações utilizam, reduzindo, assim, o custo e o tempo de manutenção.

4.4 Conclusão

Os itens enumerados acima são apenas algumas das vantagens alcançadas ao utilizarmos uma arquitetura de componentes, mais especificamente, a arquitetura orientada a serviços, para desenvolver uma aplicação complexa.

Um projeto de integração de um negócio de grande porte é algo muito complexo

e demorado. Sendo assim, se pudermos desenvolver o sistema com um alto índice de reaproveitamento, baixa redundância de funcionalidades e conseqüentes reduções de custo de produção e de tempo de desenvolvimento, teremos uma solução que atenderá melhor ao cliente. E isto é alcançado com a utilização de SOA.

Além disso, temos como vantagens importantes a possibilidade de adeqüar de o aplicativo de forma fácil às mudanças no negócio do cliente, e as reduções do tempo e do custo de manutenção da aplicação, citadas anteriormente.

Certamente, SOA está mostrando-se como um paradigma que pode ser muito vantajoso tanto aos desenvolvedores quanto aos clientes.

5 *Padrões utilizados*

Como dito anteriormente, diversas regras e padrões devem ser seguidos para montarmos um sistema baseado em SOA.

O ciclo de vida de um serviço envolve diversas fases e cada uma destas fases tem um padrão a ser adotado para que o serviço seja corretamente especificado. Um serviço deve seguir padrões com relação ao formato do protocolo de mensagens, assim como com relação à forma de descrevê-lo ou torná-lo disponível em um serviço de diretórios.

Apesar de algumas fases admitirem uma série de padrões, alguns desses padrões assumiram um posto de destaque e tornaram-se padrões *de facto* para a implementação de uma arquitetura orientada a serviços.

Assim, o mais comum é encontrarmos serviços implementados utilizando **XML** como a linguagem responsável por representar tipos de dados e compôr as mensagens, **SOAP** como o protocolo de troca de mensagens XML, **HTTP** como o protocolo responsável pelo envio das mensagens, **WSDL** para descrever o serviço e **UDDI** para listar os serviços na rede.

Os protocolos citados, assim como alguns outros que são responsáveis por funções auxiliares, serão descritos a seguir.

5.1 eXtensible Markup Language (XML)

XML ou Linguagem de Marcadores Extensíveis é uma linguagem muito utilizada atualmente em diversas aplicações pois, devido ao fato de ser possível criar marcadores de acordo com a necessidade corrente, permite que representemos muitos tipo de dados.

A facilidade disponibilizada pela extensibilidade dos marcadores torna possível criarmos marcadores que realmente trazem um valor semântico à mensagem e, assim, facilitam imensamente o tratamento dos dados.

XML é uma linguagem baseada em texto e é totalmente independente de plataforma. Deste modo, esta linguagem é ideal para a troca de mensagens entre plataformas, que é algo muito comum de ocorrer entre serviços, que, provalvelmente, estão em plataformas distintas.

Com XML, a informação é organizada de uma forma hierárquica, parecida com uma árvore, onde os marcadores mais internos são imediatamente dependentes do marcador que está no nível superior a estes.

Este tipo de estrutura, aliada aos marcadores com valor semântico, permite que os algoritmos responsáveis por analisar a mensagem sejam simples e eficientes.

Além disso, utilizando XML, criamos mensagens que são facilmente interpretadas tanto por humanos quanto por computadores, como explicado anteriormente, e podemos incluir praticamente qualquer informação em qualquer linguagem, devido ao suporte a *Unicode*, que é um formato de representação de caracteres que permite incluirmos os caracteres ocidentais comuns, caracteres não-romanos, símbolos matemáticos, entre outros.

A seguir temos um exemplo de representação de tipos de dados utilizando XML. O exemplo segue o esquema de esquemas sugerido pelo W3C e conhecido como XSD.

5.1.1 Exemplo de arquivo XSD

XSD (XML Schema Definition) é um esquema baseado em XML utilizado para descrever tipos de dados, assim como as relações entre tipo de dados distintos. Além disso, através do XSD também é feita a validação da representação do dado.

O documento a seguir é uma amostra de um pedido que transita ao longo do fluxo de um processo de baixa de um acesso de uma rede de serviços, como o presenciado por mim durante a implantação do processo de baixa de acessos para a Telefônica.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <xsd:schema
3     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4     targetNamespace="http://Businessitems">
5     <xsd:element name="Pedido" type="tns:Pedido"/>
6     <xsd:complexType name="Pedido">
7        <xsd:sequence>
8           <xsd:element name="Id" type="xsd:string"/>
```

```
9           <xsd:element name="Posicao" type="xsd:string"/>
10          <xsd:element name="Header" type="tns:Header"/>
11          <xsd:element name="AcessoA" type="tns:Acesso_A"/>
12          <xsd:element name="AcessoZ" type="tns:Acesso_Z"/>
13          <xsd:element name="FlagsPedido" type="tns:FlagsPedido"/>
14          <xsd:element name="DadosAlocacao"
15                       type="tns:DadosAlocacao"
16                       maxOccurs="unbounded"/>
17       </xsd:sequence>
18    </xsd:complexType>
19    <xsd:complexType name="Header">
20       <xsd:sequence>
21          <xsd:element name="TipoPedido" type="xsd:string"/>
22          <xsd:element name="Operacao" type="xsd:string"/>
23          <xsd:element name="Status" type="xsd:string"/>
24          <xsd:element name="ProjetoEspecial" type="xsd:string"/>
25          <xsd:element name="SequencialCircuito" type="xsd:string"/>
26       </xsd:sequence>
27    </xsd:complexType>
28    <xsd:complexType name="Acesso_A">
29       <xsd:complexContent>
30          <xsd:extension base="tns:Acesso"/>
31       </xsd:complexContent>
32    </xsd:complexType>
33    <xsd:complexType name="Acesso" abstract="true">
34       <xsd:sequence>
35          .
36          .
37          .
38       </xsd:sequence>
39    </xsd:complexType>
40    <xsd:complexType name="Cliente" abstract="true">
41       <xsd:sequence>
42          .
43          .
```

```
44            .
45         </xsd:sequence>
46      </xsd:complexType>
47      <xsd:complexType name="DadosInstalacao" abstract="true">
48         <xsd:sequence>
49            .
50            .
51            .
52         </xsd:sequence>
53      </xsd:complexType>
54      <xsd:complexType name="Comp" abstract="true">
55         <xsd:sequence>
56            <xsd:element name="DadosTec"
57                         type="tns:DadosTec"
58                         maxOccurs="20"/>
59         </xsd:sequence>
60      </xsd:complexType>
61      <xsd:complexType name="DadosTec" abstract="true">
62         <xsd:sequence>
63            <xsd:element name="NomeAtributo" type="xsd:string"/>
64            <xsd:element name="ValorAtributo" type="xsd:string"/>
65            <xsd:element name="UnidadeAtributo" type="xsd:string"/>
66         </xsd:sequence>
67      </xsd:complexType>
68      <xsd:complexType name="Roteador" abstract="true">
69         <xsd:sequence>
70            <xsd:element name="NomeRoteador" type="xsd:string"/>
71            <xsd:element name="TipoRoteador" type="xsd:string"/>
72            <xsd:element name="DadosRoteador" type="xsd:string"/>
73         </xsd:sequence>
74      </xsd:complexType>
75      <xsd:complexType name="DadosConsultaInventario">
76         <xsd:sequence>
77            <xsd:element name="Circuito"
78                         type="tns:Circuito"
```

```
                    maxOccurs="20"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Circuito">
      <xsd:sequence>
         <xsd:element name="Elemento" type="tns:Elemento"/>
      </xsd:sequence>
   </xsd:complexType>
   <xsd:complexType name="Elemento">
      <xsd:sequence>
         .
         .
         .
      </xsd:sequence>
   </xsd:complexType>

         .
         .
         .

   <xsd:complexType name="CPE">
      <xsd:sequence>
         <xsd:element name="FlagCPE" type="xsd:string"/>
         <xsd:element name="FlagCPETelefonica" type="xsd:string"/>
         <xsd:element name="SenhaFornecida" type="xsd:string"/>
         <xsd:element name="NIP" type="xsd:string"/>
         <xsd:element name="NecessarioGerencia" type="xsd:string"/>
         <xsd:element name="Item"
                     type="tns:Item"
                     minOccurs="0"
                     maxOccurs="unbounded"/>
         <xsd:element name="DadosTeste" type="tns:DadosTeste"/>
      </xsd:sequence>
   </xsd:complexType>

```

```
114             .
115             .
116             .
117
118       <xsd:complexType name="AcessosBalanceamento">
119          <xsd:sequence>
120             <xsd:element name="IdAcesso"
121                          type="xsd:string"
122                          maxOccurs="6"/>
123          </xsd:sequence>
124       </xsd:complexType>
125    </xsd:schema>
```

A **linha 1** do documento representa a declaração. Nesta linha temos os atributos *version*, que informa que o documento segue a especificação 1.0, e *encoding*, que define o tipo de codificação de caracteres utilizado, no caso ISO-8859-1 (*Latin-1/West European*).

Na **linha 2**, temos a definição do esquema. O atributo **xmlns:xsd** define que os elementos e tipos de dados usados vêm do *namespace* "http://www.w3.org/2001/XMLSchema". *Namespace* é um ambiente onde o nome das variáveis é único.

Na **linha 5**, vemos a definição do primeiro elemento do documento. Este é o elemento principal utilizado em todo o sistema mencionado e os atributos *name* e *type* informam o nome e o tipo do elemento, respectivamente.

Neste caso, o elemento **pedido** é um tipo complexo e será definido na **linha 6**.

Nas **linhas 7 e 17**, temos o marcador *sequence*. Isto significa que todos os elementos internos a estes marcadores deverão manter a ordem em que eles aparecem no XML.

Nas **linhas 8 a 16**, temos os elementos que compõe este tipo complexo. Eles são definidos da mesma forma que foi descrita anteriormente.

Na **linha 106**, podemos notar alguns atributos importantes de um elemento: *minOccurs*, que define a ocorrência mínima do elemento, no caso zero, o que significa que ele não precisa estar presente no XML; e *maxOccurs*, que define a ocorrência máxima, no caso *unbounded* (ilimitado), que significa que não há um limite de vezes que o elemento *Item* pode aparecer na mensagem.

5.2 Simple Object Acess Protocol (SOAP)

SOAP é um protocolo baseado em XML que é utilizado para definir um modo uniforme de transmitir dados representados no formato XML.

Ele é um protocolo para troca de mensagens de via única e que não guarda informações sobre interações anteriores (*stateless*). É um paradigma bastante simples, no entanto, as aplicações podem implementar interações mais complexas, tais como requisição/resposta e requisição/múltiplas respostas.

Este protocolo não é responsável pelo envio em si da mensagem mas, sim, por um conceito conhecido como envelope. O protocolo SOAP encapsula a mensagem e agrega informações tais como a quem se destina a mensagem, exatamente como um envelope de uma carta.

Um uso comum para o protocolo SOAP é na chamada de funções remotas (RPC ou *Remote Procedure Call*), onde precisamos informar, junto com a requisição para execução desta função, o local onde a função será encontrada, o nome da função, os argumentos e o valor de retorno.

A principal desvantagem do protocolo reside no fato dele ser baseado em texto. Por não ser um protocolo binário, o seu uso fica empedido em diversos casos.

5.3 HyperText Transfer Protocol (HTTP)

HTTP é um protocolo da camada de aplicação que tornou-se o protocolo padrão de transferência de dados na Internet (*World-Wide Web*).

Originalmente, o protocolo foi desenvolvido para o intercâmbio de páginas HTML mas, por ser um protocolo do tipo requisição-resposta, tem sido utilizado em diversas outras ocasiões.

Uma requisição utilizando HTTP consiste em um cliente iniciando uma conexão TCP (*Transfer Control Protocol*), protocolo da camada de transporte responsável por dividir as mensagens em pacotes e enviá-las através de uma rede, com um servidor em um ponto distinto da rede.

A requisição é enviada a uma porta específica onde o servidor já monitora a entrada de requisições e, após processar a requisição, o servidor envia uma mensagem de resposta que será a informação requisitada ou alguma mensagem de erro.

Uma variação comum do HTTP é o HTTPS (*HTTP over Secure Socket Layer*). Neste caso, a camada de segurança (SSL) criptografa a mensagem antes do envio.

5.4 Web Services Description Language (WSDL)

WSDL é um protocolo também baseado em XML e responsável por descrever um serviço, ou seja, por descrever a interface do serviço, que é a única parte visível do mesmo.

Através da descrição no formato determinado pelo protocolo WSDL, temos acesso a todas as informações necessárias para realizar a comunicação com um certo serviço. As informações importantes para a comunicação são os formatos das mensagens de entrada e saída a serem trocadas nas interações com o serviço, as operações suportadas pelo mesmo, os tipos de dados utilizados nestas interações, o endereço onde o serviço pode ser encontrado e as interfaces de ligação (*binding*).

Quando os tipos de dados utilizados pelo serviço são tipos complexos, estes são descritos através do esquema XML (XSD) citado anteriormente.

Resumidamente, WSDL é responsável por revelar o que um serviço pode fazer, onde ele está, como invocá-lo e quais tipos de dados enviar ao serviço. Ele descreve completamente a interface de um serviço.

O documento a seguir é um exemplo da interface de um serviço utilizado no projeto para a Telefônica. Trata-se de um serviço bem simples e a sua descrição, através do documento wsdl, reflete isto.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <wsdl:definitions
3     xmlns:bons1="http://Businessitems"
4     xmlns:tns="http://Utilitario/interfaces/InterfacePedido"
5     xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
6     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
7     name="InterfacePedido"
8     targetNamespace="http://Utilitario/interfaces/InterfacePedido">
9     <wsdl:types>
10       <xsd:schema
11          targetNamespace="http://Utilitario/interfaces/InterfacePedido"
12          xmlns:bons1="http://Businessitems"
```

```
13          xmlns:tns="http://Utilitario/interfaces/InterfacePedido"
14          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
15          <xsd:import
16             namespace="http://Businessitems"
17             schemaLocation="../xsd-includes/http.Businessitems.xsd"/>
18          <xsd:element name="operation1">
19             <xsd:complexType>
20                <xsd:sequence>
21                   <xsd:element
22                      name="Input"
23                      nillable="true"
24                      type="bons1:Pedido"/>
25                </xsd:sequence>
26             </xsd:complexType>
27          </xsd:element>
28          <xsd:element name="operation1Response">
29             <xsd:complexType>
30                <xsd:sequence>
31                   <xsd:element
32                      name="Output"
33                      nillable="true"
34                      type="bons1:Pedido"/>
35                </xsd:sequence>
36             </xsd:complexType>
37          </xsd:element>
38       </xsd:schema>
39    </wsdl:types>
40    <wsdl:message name="operation1RequestMsg">
41       <wsdl:part
42          element="tns:operation1"
43          name="operation1Parameters"/>
44    </wsdl:message>
45    <wsdl:message
46       name="operation1ResponseMsg">
47       <wsdl:part
```

```
48            element="tns:operation1Response"
49            name="operation1Result"/>
50      </wsdl:message>
51      <wsdl:portType name="InterfacePedido">
52         <wsdl:operation name="operation1">
53            <wsdl:input
54               message="tns:operation1RequestMsg"
55               name="operation1Request"/>
56            <wsdl:output
57               message="tns:operation1ResponseMsg"
58               name="operation1Response"/>
59         </wsdl:operation>
60      </wsdl:portType>
61   </wsdl:definitions>
```

É interessante destacar neste documento a presença dos elementos *operation1* e *operation1Response* nas **linhas 18 e 28**, respectivamente.

Estes elementos são os que definem os tipos de dados de entrada e saída. Neste caso, o tipo de dado utilizado tanto como entrada quanto como saída é o tipo *pedido*, descrito anteriormente.

Nas **linhas 51 a 60**, temos o marcador *portType*, que define a operação realizada pelo serviço, que consiste nos tipos de mensagens de entrada e saída, basicamente.

5.5 Universal Description, Discovery and Integration (UDDI)

UDDI também é um protocolo baseado em XML e é utilizado para descrever, descobrir e gerenciar os serviços na rede. Através deste protocolo, existe uma maneira padronizada dos serviços descreverem a si próprios e, assim, tornarem-se disponíveis a quem possa utilizá-los.

UDDI provê um diretório de serviços, algo que pode ser comparado com uma "lista telefônica de serviços"tanto pela utilidade quanto pela forma como as informações estão organizadas.

Ao publicarmos um serviço, temos que disponibilizar diversas informações para que

um provável usuário deste serviço possa encontrá-lo de forma mais fácil.

Sendo assim, há três componentes básicos de registro de um serviço que estão explicitados abaixo:

- Páginas brancas: aqui, há informações relacionadas com os desenvolvedores tais como informações de contato, por exemplo;
- Páginas amarelas: aqui, as informações estão relacionadas às categorias de atuação do serviço, como numa lista de páginas amarelas clássica; e
- Páginas verdes: informações técnicas sobre o serviço.

Disponibilizando tais informações quando um serviço é publicado, ele fica muito mais acessível pois um potencial usuário deste serviço poderá encontrá-lo fazendo buscas pelos desenvolvedores ou pela categoria de indústria a qual se destina, por exemplo.

Deste modo, as principais normas utilizadas numa arquitetura foram descritas. No entanto, estas não são as únicas normas seguidas e, sim, as mais comumente utilizadas.

Como exemplo, podemos citar outros protocolos de rede responsáveis pelo transporte de mensagens, além do HTTP e do HTTPS, como FTP (*File Transfer Protocol*) e SMTP (*Simple Mail Transfer Protocol*). Com objetivos similares ao UDDI, temos WS-Inspection, WS-Metadata Exchange e WS-Resource FrameWork, por exemplo.

A seguir, temos alguns outros padrões relacionados com camadas auxiliares da arquitetura de um serviço.

5.6 Mensageria

A camada de mensageria é responsável pela confiabilidade da mensagem, ou seja, por assegurar que a mensagem chegue ao destinatário e retorne adequadamente.

Algumas especificações relacionadas a esta camada estão nas seções a seguintes.

5.6.1 WS-Addressing

Especificação que permite que sistemas de mensagens suportem transmissões de mensagens de uma maneira neutra através de redes que incluem pontos de processamento

como *firewalls* e *gateways*, por exemplo. Anteriormente, esta especificação já foi conhecida por WS-Routing, WS-Referral e SOAP Routing Protocol (SOAP-RP).

5.6.2 SOAP Message Transmission Optimization Mechanism (MTOM)

MTOM descreve um mecanismo de otimização da transmissão da mensagem SOAP através de uma re-codificação seletiva de porções da mensagem para otimizar o tratamento.

5.6.3 WS-Reliable Messaging

Especificação criada com o propósito de criar um modelo genérico de assegurar a entrega de uma mensagem para um serviço.

5.7 Segurança

Segurança é, sem dúvidas, um fator relevante para a utilização de serviços. Sendo assim, cada camada da arquitetura necessita ser segura para manter a segurança do sistema todo.

Os requisitos de segurança variam desde privacidade, confiabilidade até integridade de dados, para assegurar a realização de um negócio virtualmente na rede.

A seguir, temos especificações relacionadas a segurança.

5.7.1 WS-Security

Este é um protocolo de comunicação que provê meio de aplicar segurança aos serviços.

O protocolo foi, originalmente, desenvolvido pela IBM, Microsoft e VeriSign, e contém especificações para reforçar a integridade e a confidencialidade na troca de mensagens entre os serviços.

5.7.2 XML-Encryption

Criptografar mensagens XML é essencial para manter a confidencialidade da informação, como dados bancários e de cartão de crédito, tanto durante o envio quanto

ao armazenar os dados.

Em geral, o esquema de chave pública é utilizado para criptografar as mensagens. Neste esquema, existe um par de chaves: uma para criptografar e outra para decriptografar.

Parte III

Conclusão

6 Visão geral sobre SOA

SOA é um paradigma recente mas que aplica conceitos difundidos há bastante tempo.

Como vimos ao longo do texto, SOA tem inúmeras vantagens, assim como qualquer arquitetura baseada em componentes.

Quando dividimos o sistema que pretendemos implantar em funções de negócios, e implementamos estas funções como serviços, temos um ganho considerável de tempo e uma sensível redução de custos pois o nível de reutilização de código aumenta.

Além disso, centralizar a lógica de uma função de negócio em um único serviço, propicia um ambiente muito mais dinâmico, onde mudanças podem ser implantadas rapidamente já que estas precisam ser realizadas em apenas um lugar do sistema.

Vimos, também, que SOA é uma solução interessante para lidar com sistemas antigos (sistemas de legado) que, geralmente, são sistemas pouco adaptados à realidade atual do negócio do cliente. Isso porque podemos encapsular funções realizadas por estes sistemas em novos serviços que podem ser criados para melhorar o modo como o negócio é operado.

Um exemplo concreto é o vivenciado por mim durante implantação do sistema de aprovisionamento de acessos da Telefônica. Antes, haviam inúmeros sistemas de legado que já estavam defasados e eram operados manualmente, o que causava grande quantidade de inconsistências no sistema. Agora, apesar de não termos implantado uma solução completa, pudemos camuflar o acesso a estes sistemas, que agora são realizados de forma quase que totalmente automática pelo nosso sistema. Apenas algumas interações tipicamente humanas foram mantidas sendo que o principal da lógica de negócio relacionada com um aprovisionamento de acesso foi encapsulado e automatizado.

Unindo as vantagens citadas à força das empresas que estão patrocinando a adoção deste novo paradigma, é possível concluir que SOA terá um espaço cada vez maior no dia-a-dia das empresas que implantam sistemas de integração. Realmente, SOA é um dos temas mais comentados atualmente no mercado de tecnologia e as empresas estão

destinando recursos crescentes a este paradigma.

Parte IV

Parte subjetiva

7 O projeto

A idéia de realizar o trabalho de formatura a respeito de orientação a serviços surgiu a medida que o projeto do qual eu participava na Telefônica se desenvolvia.

Eu percebi que SOA era um termo recorrente e que orientação a serviços era algo muito discutido, no entanto, este não é um conceito que é amplamente conhecido atualmente.

Sendo assim, resolvi desenvolver um trabalho de pesquisa a respeito e pude ancorá-lo na minha experiência adquirida no dia-a-dia junto com o restante da equipe e em um curso ministrado pela IBM a respeito de SOA e da ferramenta que utilizamos para implantar a solução baseada em SOA.

A Accenture possui, também, um material extensivo a respeito de SOA na intranet da empresa e pude consultar tais cursos para orientar a minha pesquisa.

8 A graduação

Durante toda a graduação, pude desenvolver a minha capacidade de aprendizado e pesquisa e isto foi muito útil neste momento.

Apesar deste texto tratar-se uma monografia de conclusão de curso, a pesquisa realizada para desenvolvê-la tem sido extremamente útil no dia-a-dia do trabalho na Accenture.

Isto prova que o curso prepara, sim, o aluno para o mercado de trabalho, onde pude constatar que capacidades como as citadas acima são muito mais valorizadas (e realmente são mais úteis) do que conhecimentos específicos relacionados a linguagens ou *frameworks*.

Algumas matérias tiveram destaque nesta formação e eu irei citá-las nas seções a seguir.

8.1 MAC110 Introdução à Computação
MAC122 Princípios de Desenvolvimento de Algoritmos

Apesar de serem matéria iniciais do curso e que, por isso, não cobravam tanto conhecimento teórico, estas matérias foram essenciais para desenvolver a capacidade de aprendizado individual que foi tão útil ao longo do curso e é útil no mercado de trabalho.

Mais do que ensinar a sintáxe das linguagens utilizadas (JAVA e C), estas matérias ensinaram como entender um problema e, então, pensar em uma forma de resolvê-lo (algoritmo). De posse disto, a tarefa de desenvolver o código, independente da linguagem, torna-se simples.

8.2 MAC426 Sistemas de Bancos de Dados

Esta é uma matéria muito importante, sem dúvidas, pois praticamente qualquer aplicação, por mais simples que seja, utiliza um sistema de banco de dados.

Da mesma forma que a orientação geral do curso, nesta matéria não é ensinada uma linguagem específica, como PL/SQL, mas, sim, como entender um sistema de banco de dados para utilizá-lo.

Apesar de não irmos muito fundo na implantação de um sistema de banco de dados nesta matéria, o conteúdo ensinado é extremamente importante e utilizado a todo momento no mercado de trabalho.

8.3 PCS210 Redes de Computadores
MAC448 Programação para Redes de Computadores

Atualmente, as redes de computadores estão presentes em todos os lugares. Assim, entender bem os conceitos envolvidos é essencial para compreender os sistemas que utilizamos e mesmo os que iremos implantar.

Estas matérias propiciaram este entendimento que são extremamente úteis. Na implantação do sistema citado nesta monografia, por exemplo, o aprovisionamento de acessos envolve diversas tecnologias como ATM e FRAME RELAY, as quais eu já conhecia e compreendia devido a estas duas matérias.

8.4 MAC340 Laboratório de Engenharia de Software

Nesta matéria, realizada neste último ano do curso, pude entender e implantar um sistema baseado em componentes. Tendo em vista que SOA trata-se, acima de tudo, de um paradigma relacionado com componenetes, esta matéria foi de extrema importância para a compreensão de grande parte dos conceitos envolvidos nesta monografia.

Referências

1 SERVICE-ORIENTED Computing: Semantics, Processes, Agents. [S.l.]: Chichester: John Wiley and Sons Ltd., 2005.

2 INTRODUCTION to Service Oriented Architecture (SOA). [S.l.], 2005.