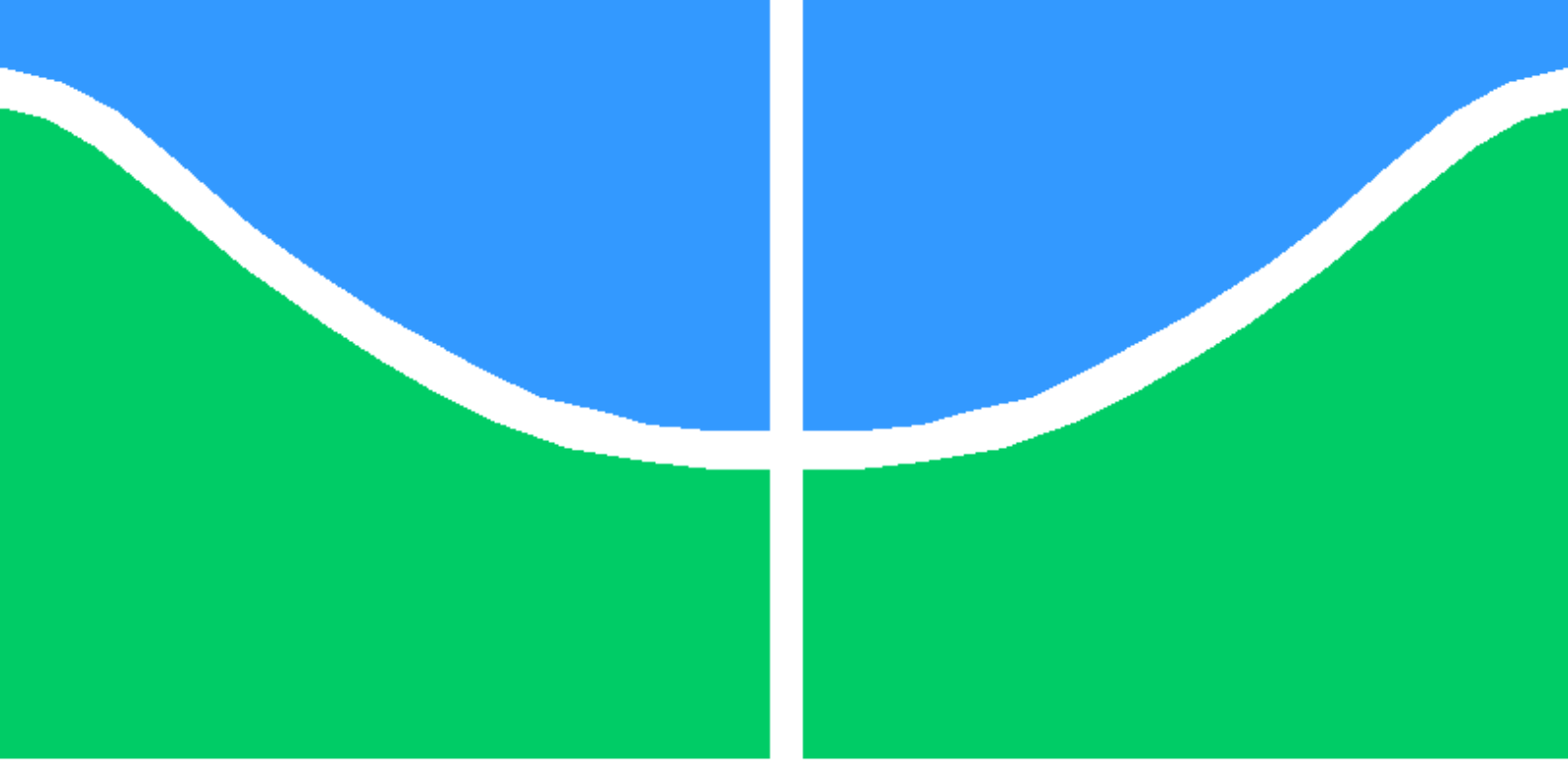

# ARQUITETURA ORIENTADA A SERVIÇOS PARA COMÉRCIO ELETRÔNICO NO SISTEMA BRASILEIRO DE TV DIGITAL

MANOEL CAMPOS DA SILVA FILHO

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

**UNIVERSIDADE DE BRASÍLIA**
**FACULDADE DE TECNOLOGIA**
**DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

# ARQUITETURA ORIENTADA A SERVIÇOS PARA COMÉRCIO ELETRÔNICO NO SISTEMA BRASILEIRO DE TV DIGITAL

**MANOEL CAMPOS DA SILVA FILHO**

DISSERTAÇÃO DE MESTRADO ACADÊMICO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA.

APROVADA POR:

Prof. Dr. Paulo Roberto de Lira Gondim, ENE/UnB
Orientador

Prof. Dr. Divanilson Rodrigo Campelo, ENE/UnB
Examinador interno

Prof. Dr. Díbio Leandro Borges, CIC/UnB
Examinador externo

**BRASÍLIA, 16 DE JUNHO DE 2011.**

**FICHA CATALOGRÁFICA**
MANOEL CAMPOS DA SILVA FILHO
**Arquitetura orientada a serviços para comércio eletrônico no Sistema Brasileiro de TV Digital**
**2011xv, 107p., 201x297 mm**
(ENE/FT/UnB, Mestre, Engenharia Elétrica, 2011)
Dissertação de Mestrado - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Elétrica

## REFERÊNCIA BIBLIOGRÁFICA

MANOEL CAMPOS DA SILVA FILHO (2011) Arquitetura orientada a serviços para comércio eletrônico no Sistema Brasileiro de TV Digital. Dissertação de Mestrado em Engenharia Elétrica, Publicação 439/2011, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 107p.

## CESSÃO DE DIREITOS

AUTOR: Manoel Campos da Silva Filho
TÍTULO: Arquitetura orientada a serviços para comércio eletrônico no Sistema Brasileiro de TV Digital.
GRAU: Mestre ANO: 2011



---

Manoel Campos da Silva Filho
Universidade de Brasília, Faculdade de Tecnologia, Departamento de Engenharia Elétrica, 70910-900, Brasília-DF, Brasil.

# Resumo

## Arquitetura orientada a serviços para comércio eletrônico no Sistema Brasileiro de TV Digital

Esta dissertação descreve uma arquitetura orientada a serviços para provimento de comércio eletrônico pela TV Digital, por meio do Sistema Brasileiro de TV Digital (SBTVD), desenvolvida para o sub-sistema Ginga-NCL do *middleware* Ginga. A arquitetura proposta utiliza serviços de diferentes provedores (nas áreas de telecomunicações, logística e outros) para compor uma estrutura de *T-Commerce*. Tais serviços são desenvolvidos considerando aspectos de interoperabilidade, utilizando o protocolo SOAP, para o qual é apresentada uma implementação, juntamente com o HTTP, como base para o desenvolvimento de toda a arquitetura e um dos objetivos principais do projeto.

Com a arquitetura elaborada, uma aplicação cliente, desenvolvida em NCL e Lua, é apresentada como prova de conceito do uso das implementações dos protocolos e da arquitetura proposta. Tal aplicação utiliza o *framework* LuaOnTV para a construção da interface gráfica de usuário para a TV Digital, o qual foi estendido neste trabalho, com as melhorias sendo apresentadas ao longo do mesmo.

O trabalho ainda apresenta um conjunto de aplicações desenvolvidas a partir dos *frameworks* construídos, que complementam as funcionalidades da aplicação de T-Commerce, como leitor de RSS e rastreamento de encomendas.

A partir do ambiente de desenvolvimento montado para a construção das aplicações, contendo a implementação de referência do sub-sistema Ginga-NCL do *middleware* Ginga, nativamente instalada, foi gerada uma distribuição Linux que permite que tal ambiente seja instalado em qualquer computador ou máquina virtual, para permitir o desenvolvimento de arquitetura semelhante ou extensão da arquitetura proposta.

**Palavras-chave:** SBTVD, Ginga, Ginga-NCL, *Web Services*, HTTP, SOAP, SOA, Lua, NCL, NCLua, NCLua HTTP, NCLua SOAP, LuaOnTV 2.0, *E-Commerce*, *T-Commerce*

# Capítulo 2

# Revisão bibliográfica

## 2.3 A Tecnologia de *Web Services*

Há alguns anos as aplicações corporativas eram desenvolvidas principalmente em um modelo cliente/servidor de duas camadas, existindo uma aplicação *desktop* que fazia acesso direto a um banco de dados. Pela natureza destas aplicações, que precisam ser instaladas em cada computador onde serão executadas, existe um grande esforço para atualizar todos os computadores com novas versões do sistema. Mesmo que haja um processo automatizado para esta atualização, isto demanda recursos como tempo e largura de banda. Com o avanço das tecnologias de comunicação de dados, o avanço da *Web* e suas linguagens de programação e, principalmente, com o advento da tecnologia AJAX, atualmente é possível desenvolver aplicações de *Internet* ricas (*Rich Internet Applications*, RIA) com componentes visuais que vão além dos componentes básicos oferecidos pela linguagem HTML. Isto permitiu que muitas empresas migrassem seus sistemas *desktop* para a plataforma *Web*, sem perder funcionalidades existentes naquele tipo de interface.

Essa mudança de paradigma *desktop* para *Web* vem seguida ainda de outra tendência: a dos sistemas distribuídos. Estes são sistemas que comumente utilizam a arquitetura cliente/-servidor, no entanto, são compostos de três ou mais camadas. Segundo [Sommerville 2011]:

> "um sistema distribuído é uma coleção de computadores independentes que aparece para o usuário como um único sistema coerente."

A Figura 2.1 apresenta uma arquitetura de um sistema distribuído em quatro camadas. As mesmas são descritas a seguir.

- Camada de Apresentação: pode contar tanto com aplicações *desktop* leves, conhecidas como *thin client* (cliente magro/leve), possuindo apenas uma interface gráfica que faz

acesso a um servidor de aplicação, por meio de chamadas de procedimento remoto; quanto com um *browser*, que faz acesso a um servidor *Web*, responsável por gerar as interfaces HTML.

- Camada *Web*: composta por um ou mais servidores *Web*, responsáveis por gerar a interface HTML para os clientes *Web*. As aplicações *desktop* não acessam esta camada, tendo ligação direta com a Camada de Aplicação.

- Camada de Aplicação: composta por um ou mais servidores de aplicação onde as regras de negócios da aplicação estão implementadas. Desta forma, alterações na lógica de negócios não requer a atualização dos clientes. Os servidores de aplicação adicionam tolerância a falhas e balanceamento de carga ao sistema.

- Camada de Dados: composta por um ou mais servidores de banco de dados, acessíveis apenas pelos servidores de aplicação, o que garante transparência e independência de banco de dados aos clientes.

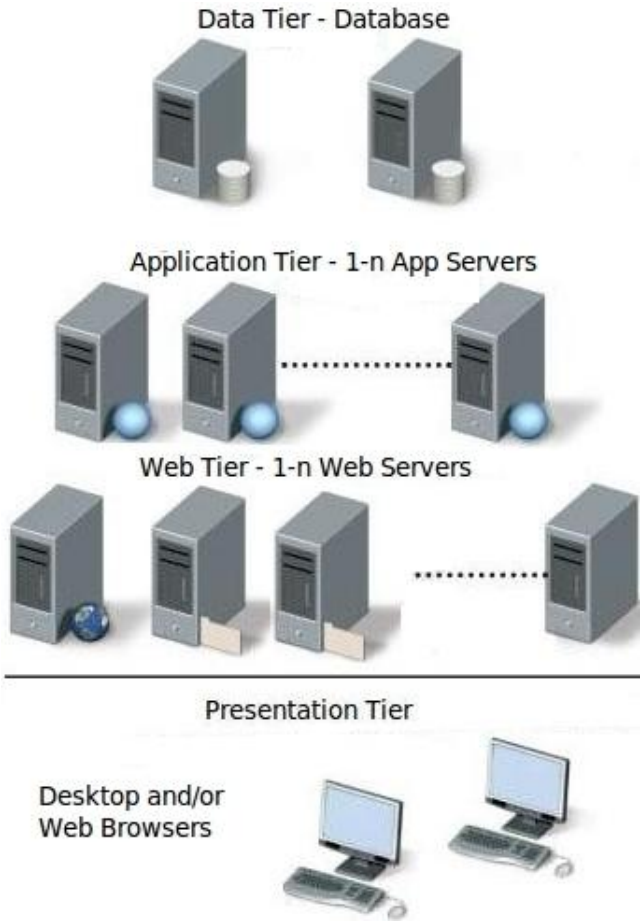


Figura 2.1: Arquitetura de Aplicação Distribuída (adaptada de [InterSystems 2010]).

Segundo [Coulouris, Dollimore e Kindberg 2007], um sistema distribuído traz benefícios como: permitir a heterogeneidade de componentes, possibilitar a escalabilidade, ser tolerante a falhas, dentre outros.

Neste contexto, *Web Services* (WSs) proveem um *framework* extensível para comunicação de aplicação para aplicação, que utiliza protocolos *Web* existentes e baseados em padrões XML abertos [Curbera et al. 2002]. Eles são a maior tecnologia para publicação de serviços na *Web* e têm significativa adoção em áreas como integração de aplicações, computação distribuída de larga escala e cooperação *business-to-business* (B2B) [KOPECKÝ e Simperl 2008]. Os mesmos são instalados na Camada de Aplicação, como mostrado na Figura 2.1, e proveem um conjunto de padrões para acesso a serviços pela *Internet*, sendo uma tecnologia estabelecida no mercado e cada vez mais adotada por empresas [Lausen e Haselwanter 2007].

*Web Services* permitem o desenvolvimento baseado em componentes, acessíveis por meio da *Internet*. Eles são componentes reutilizáveis, que as aplicações, independentemente da linguagem em que foram implementadas, podem utilizar sem se preocupar em como eles foram desenvolvidos [Vilas et al. 2007]. Desta forma, a construção de aplicações a partir de Web Service segue o processo de desenvolvimento de *software* conhecido como componentização, como apresentado em [Sommerville 2011].

Diferentemente de tecnologias como *Common Object Request Broker Architecture* (CORBA), *Distributed Component Object Model* (DCOM), *Component Object Model Plus* (COM+) e Java *Remote Method Invocation* (RMI), WSs usam protocolos *Web* e formatos de dados ubíquos, como HTTP e XML [Vilas et al. 2007].

O objetivo dos *Web Services* é alcançar a interoperabilidade entre aplicações, usando padrões *Web* (*Web standards*). Eles usam um modelo de integração fracamente acoplado para permitir a integração flexível de sistemas heterogêneos em uma variedade de domínios, incluindo *Business-to-Consumer* (*B2C*), *Business-to-Business* (*B2B*) e *Enterprise Application Integration* (*EAI*) [OASIS 2007].

Pela própria natureza distribuída da *Web*, o uso de WSs vem ao encontro da integração entre aplicações heterogêneas, executando em diferentes plataformas, desenvolvidas por diferentes empresas. Cada vez mais empresas disponibilizam serviços, públicos ou privados, para serem utilizados por terceiros, permitindo a interoperabilidade entre diferentes sistemas. Além disto, pelo fato de WSs serem transportados, principalmente, por protocolo HTTP, não sofrem com problemas de portas bloqueadas em *Firewalls* que outras tecnologias, como as mencionadas anteriormente, encontram. Isto facilita ainda mais a integração entre sistemas hospedados em diferentes redes.

### 2.3.1 Protocolos de Comunicação

Dada a natureza distribuída e heterogênea da *Web*, mecanismos de comunicação devem ser independentes de plataforma, seguros e leves quanto possível. A linguagem XML é um padrão ~~altamente~~ que foi utilizado para codificação e intercâmbio de dados entre sistemas, sendo totalmente independente de plataforma. Por esse motivo, a mesma é utilizada no protocolo de comunicação usado por WSs SOAP.

O *Simple Object Access Protocol* (SOAP), é um protocolo padronizado pelo *World Wide Web Consortium* (W3C), como protocolo de comunicação para WSs. O SOAP é um protocolo, baseado em XML, para a troca de mensagens e chamadas de procedimentos remotos (*Remote Procedure Call*, RPC). No lugar de definir um novo protocolo para empacotar as mensagens, SOAP utiliza protocolos *Web* existentes como HTTP e SMTP.

Ele é um protocolo leve, adequado para comunicação em um ambiente distribuído e descentralizado [Vilas et al. 2007].

Uma mensagem SOAP, também denominada "envelope SOAP", é composta por um arquivo XML, contendo um elemento *header* e um *body*. A Listagem 2.1 mostra a estrutura de um envelope SOAP.

Listagem 2.1: Estrutura de um envelope SOAP [Curbera et al. 2002]

```
1  <SOAP:Envelope
2    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
3    <SOAP:Header>
4         <!--        conteúdo do header aqui        -->
5    </SOAP:Header>
6
7    <SOAP:Body>
8         <!--        conteúdo do body aqui        -->
9    </SOAP:Body>
10 </SOAP:Envelope>
```

#### 2.3.1.1 Troca de Mensagens SOAP

Como exemplo para esta seção, vamos utilizar um sistema *Web* de uma companhia aérea, como apresentado em [Curbera et al. 2002]. O sistema disponibiliza um *Web Service* (WS) para a compra de passagens. Uma empresa que vende pacotes turísticos pode utilizar este WS para integrar o seu sistema com o da companhia aérea, e assim, fazer todo o processo de registro da passagem dentro do seu próprio sistema. Desta forma, o sistema da empresa de turismo deve enviar um envelope SOAP para o WS disponibilizado pela companhia aérea, contendo os dados do cliente e dados da passagem a ser adquirida, como data/hora, número do voo e do assento escolhido pelo cliente. A Listagem 2.2 mostra um exemplo de tal envelope SOAP, empacotado numa mensagem HTTP.

Listagem 2.2: Envelope SOAP transportado via HTTP [Curbera et al. 2002]

```
1  POST /travelservice
2  SOAPAction: "http://www.acme-travel.com/checkin"
3  Content-Type: text/xml; charset="utf-8"
4  Content-Length: nnnn
5
6  <SOAP:Envelope
7      xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
8      <SOAP:Body>
9          <et:eTicket xmlns:et=
10             "http://www.acme-travel.com/eticket/schema">
11             <et:passengerName first="Joe" last="Smith"/>
12             <et:flightInfo airlineName="AA"
13                  flightNumber="1111"
14                  departureDate="2002-01-01"
15                  departureTime="1905"/>
16         </et:eTicket>
17      </SOAP:Body>
18 </SOAP:Envelope>
```

Observe que o envelope SOAP da Listagem 2.2 não possui um *header*. O mesmo é um elemento opcional e utilizado para prover informações específicas da aplicação, como credenciais para acesso ao serviço e informações de cobrança, dentre outras[Point 2010].

#### 2.3.1.2 Chamadas de Procedimento Remoto Usando SOAP

Para que SOAP possa usar RPC para chamar procedimentos remotos, é necessário especificar alguns detalhes para o protocolo RPC, por exemplo:

- como valores de tipos são transportados entre a representação SOAP (XML) e a representação da aplicação, e vice-versa (para indicar, por exemplo, como deve ser feita a conversão de uma classe Java para XML e vice-versa), e

- onde as várias partes do protocolo RPC são transportadas (identificação de objeto, nome da operação e parâmetros de entrada e saída).

A especificação XML *schema* do W3C [11] provê uma linguagem padrão para definir a estrutura do documento e os tipos de dados da estrutura do XML. Isto é, dado um tipo básico como *integer* ou um tipo complexo como uma *struct*, XML *schema* oferece uma forma padrão de escrever dados destes tipos em um documento XML. Para habilitar o transporte de valores tipados, SOAP assume um sistema de tipos baseado em XML *schema* e define sua codificação em XML. Usando este estilo de codificação pode-se produzir uma codificação XML para qualquer tipo de dado estruturado. Parâmetros RPC e retornos são especificados usando esta codificação.

Usando o mesmo WS da companhia aérea, a empresa de turismo deseja obter informações a respeito de um determinado voo. Assim, seu sistema pode enviar um envelope SOAP ao WS da companhia aérea. O envelope da Listagem 2.3 mostra uma chamada RPC, encapsulada dentro de um envelope SOAP. O envelope permite a invocação do método remoto *GetFlightInfo*, que recebe dois parâmetros: uma *string* contendo o nome da companhia aérea, e um inteiro contendo o número do voo. Tal método retorna um valor estruturado (uma *struct*) com dois campos: o número do portão de embarque e a situação do voo.

No envelope SOAP, a chamada para *GetFlightInfo* é um elemento XML com atributos que incluem informações sobre a codificação (note a referência para o XML *schema*). Os elementos filhos são os argumentos do método: *airlineName* e *flightNumber*. Os tipos são definidos no atributo *type*, onde *xsd* refere-se as definições de um XML *schema*. Quando o servidor SOAP recebe o envelope, ele converte os valores *string*, dos atributos *airlineName* e *flightNumber*, para seus respectivos tipos *string* e inteiro, chamando o método *GetFlightInfo* passando estes valores.

Listagem 2.3: Chamada SOAP RPC empacotada em HTTP [Curbera et al. 2002]

```
1  POST /travelservice
2  SOAPAction: "http://www.acme-travel.com/flightinfo"
3  Content-Type: text/xml; charset="utf-8"
4  Content-Length: nnnn
5
6  <SOAP:Envelope
7    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
8    <SOAP:Body>
9      <m:GetFlightInfo
10       xmlns:m="http://www.acme-travel.com/flightinfo"
11       SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
12       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
13       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
14          <airlineName xsi:type="xsd:string">UL</airlineName>
15          <flightNumber xsi:type="xsd:int">506</flightNumber>
```

[11] http://www.w3.org/XML/Schema

```
16      </m:GetFlightInfo>
17    </SOAP:Body>
18  </SOAP:Envelope>
```

A Listagem 2.4 mostra a resposta à requisição SOAP enviada anteriormente. Neste caso, a resposta contém um valor estruturado, com os campos *gate* (número do portão de embarque) e *status* (situação do voo).

Implementações de SOAP existem para diversas linguagens, como C/C++, Java, Perl e Lua[12], que automaticamente geram e processam mensagens SOAP. Assumindo que as mensagens estão em conformidade com as especificações do protocolo SOAP, elas podem ser trocadas por serviços implementados em diferentes linguagens e plataformas, permitindo uma total interoperabilidade entre sistemas heterogêneos, algo que não é sempre verdade para sistemas que utilizam a arquitetura CORBA, que possui objetivos semelhantes ao SOAP.

Listagem 2.4: Resposta de requisição SOAP empacotada em HTTP [Curbera et al. 2002]

```
1  HTTP/1.1 200 OK
2  Content-Type: text/xml; charset="utf-8"
3  Content-Length: nnnn
4
5  <SOAP:Envelope
6    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
7    <SOAP:Body>
8      <m:GetFlightInfoResponse
9        xmlns:m="http://www.acme-travel.com/flightinfo"
10       SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
11       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
12       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
13          <flightInfo>
14            <gate xsi:type="xsd:int">10</gate>
15            <status xsi:type="xsd:string">ON TIME</status>
16          </flightInfo>
17      </m:GetFlightInfoResponse>
18    </SOAP:Body>
19  </SOAP:Envelope>
```

### 2.3.2 Descrição de Serviços

O protocolo SOAP define uma linguagem padrão para uso de WSs, mas ele não informa quais mensagens devem ser trocadas para que tenhamos sucesso no uso de um determinado serviço. Para isto, existe a *Web Service Description Language* (WSDL), uma linguagem também baseada em XML. Um documento WSDL descreve a interface de um WS, permitindo que as aplicações que consumirão o serviço disponibilizado,saibam quais informações

[12]Uma implementação do protocolo SOAP foi feita para uso em aplicações de TV digital desenvolvidas em NCL e Lua, e será apresentada no Capítulo 5.

devem incluir em um envelope SOAP. Logo, podem chamar um determinado procedimento remoto disponibilizado pelo WS. O WSDL descreve um WS como uma coleção de *end points* que podem trocar certas mensagens [Curbera et al. 2002]. Um *end point* é um elemento no WSDL que define a ligação entre a definição de um determinado serviço e o seu endereço de rede, permitindo o acesso ao mesmo.

As mensagens SOAP anteriormente mostradas não poderiam ter sido construídas se não conhecêssemos o documento WSDL que descreve o serviço que desejamos utilizar. Informações como o nome do método a ser acessado e o nome e tipos dos parâmetros de entrada e saída foram obtidos a partir do WSDL do serviço [Curbera et al. 2002].

#### 2.3.2.3 Usando o WSDL

Para usuários e desenvolvedores, o WSDL provê uma descrição formal das interações cliente/servidor. Desenvolvedores podem usar o documento WSDL como entrada para uma aplicação que gere funções *proxy*[14], utilizadas no cliente, para acesso aos métodos do WS. Tais funções *proxy* também podem ser geradas dinamicamente, em tempo de execução, por rotinas implementadas no cliente que interpretam o WSDL e geram as chamadas SOAP necessárias. Em ambos os casos, as funções *proxy* têm a finalidade de esconder do cliente toda a complexidade envolvida no processo de chamada dos métodos remotos.

[14]Tais funções *proxy* são escritas ou geradas na linguagem utilizada pela aplicação cliente (que consome o *Web Service*). Possuem a mesma assinatura dos métodos remotos, tornando transparente para o cliente a chamada aos métodos no WS. São responsáveis por gerar o envelope SOAP para enviar ao servidor onde o WS é hospedado.

### 2.3.4 *Web Services* REST

Como visto anteriormente, SOAP é o padrão W3C para *Web Services*, ~~amplamente~~ que foi utilizado na integração de aplicações heterogêneas. No entanto, existe uma outra vertente para construção de *Web Services* denominada REST. O REST é o acrônimo para *Representational State Transfer*. Ele é um estilo arquitetural para construção de aplicações distribuídas que também permite a integração de aplicações heterogêneas, mas não é um padrão W3C.

O estilo REST foi apresentado pela primeira vez no trabalho de dissertação de Roy Fielding[Fielding 2000], sendo baseado em padrões *Web* como HTTP e URL. Ele consiste em prover serviços *Web* a partir de um recurso *Web* acessível por meio de uma URL. A interação com tal serviço é feita por meio de uso dos métodos HTTP como *GET*, *POST* e *DELETE* para passagem de parâmetros ao serviço. Tal estilo é bastante simplificado em relação ao SOAP, pois todos os dados a serem passados ao serviço vão diretamente na URL do mesmo, por meio de uma requisição HTTP. Desta forma, o tamanho de uma requisição REST é bastante reduzido em relação ao SOAP.

REST não define um padrão de formato das mensagens de resposta, no entanto, pode-se perfeitamente utilizar XML para isto. Atualmente outros padrões de formato de dados são bastante utilizados como o *JavaScript Object Notation* (JSON). Desta forma, pode-se utilizar qualquer padrão que se desejar, como por exemplo, o formato de dados definido pela

linguagem Lua, sendo bastante útil em aplicações para o Sistema Brasileiro de TV Digital, desenvolvidas em tal linguagem, por utilizar um formato nativo da mesma.

O estilo REST atualmente se tornou um padrão de fato e é amplamente utilizado para integração de serviços *Web* com aplicações de diferentes tipos, como é o caso dos serviços disponibilizados pelo *microblog Twitter*. Este permite que desenvolvedores construam aplicações nas mais diferentes plataformas (*Web, desktop, mobile*) que se integrem com o serviço de *microblog*.

O REST também segue a arquitetura cliente/servidor usada no protocolo SOAP. Segundo [Welling 2006], o estilo é denominado REST (em português, Transferência de Estado Representacional), pois o cliente obtém uma representação de um recurso através de uma URL, o que faz com que a aplicação cliente entre em um determinado estado. O cliente pode selecionar um link que foi incluído no primeiro recurso para, por exemplo, obter informações detalhadas. Como o link direciona para outro recurso, a aplicação cliente obtém uma nova representação de um recurso e entra em um novo estado.

Como citado, a grande vantagem do REST é sua simplicidade, ~~no entanto, por não haver um padrão de passagem de parâmetros e formato de dados para a respostas das requisições, tal estilo pode não ser adequado para a construção de aplicações seguindo o padrão de arquitetura orientada a serviços, onde uma aplicação pode utilizar serviços de diferentes provedores. Usando REST, cada provedor de serviços pode definir sua forma de passar os parâmetros e o formato de dados da mensagem de retorno, assim a aplicação que integra com todos os provedores precisará implementar diversos formatos de entrada e saída de dados. Com o SOAP isto não ocorre, pois todos os provedores seguem um padrão bem definido e a aplicação cliente implementa apenas tal padrão.~~

Todas as informações riscadas acima não são mais verdade, uma vez que o formato de troca de dados normalmente utilizado por serviços REST é JSON. Existe uma estrutura bem definida para tais tipos de serviços, por mais que não sejam padronizados por órgãos como o W3C.

### 2.3.5 Arquitetura Orientada a Serviços

Segundo [Sommerville 2011], uma arquitetura orientada a serviços (*Service Oriented Architecture* - SOA) é uma forma de desenvolvimento de sistemas distribuídos onde os componentes de tal sistema são serviços *stand-alone* (autônomos), executando em computadores geograficamente distribuídos. Em tal arquitetura, ainda segundo [Sommerville 2011], usar *Web Services* é uma forma de as organizações tornarem suas informações acessíveis. Com isto, é possível haver a integração entre empresas, mesmo que por meio de sistemas heterogênos completamente diferentes.