UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

**CARLOS EDUARDO FERREIRA MARINS**

# DESENVOLVIMENTO DE UM GUIA ELETRÔNICO DE PROGRAMAÇÃO BASEADO EM SERVIÇOS WEB UTILIZANDO A ABORDAGEM REST

São Luís
2012

**CARLOS EDUARDO FERREIRA MARINS**

# DESENVOLVIMENTO DE UM GUIA ELETRÔNICO DE PROGRAMAÇÃO BASEADO EM SERVIÇOS WEB UTILIZANDO A ABORDAGEM REST

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

**Orientador: Dr. Mário Antonio Meireles Teixeira**

São Luís

2012

**CARLOS EDUARDO FERREIRA MARINS**

# DESENVOLVIMENTO DE UM GUIA ELETRÔNICO DE PROGRAMAÇÃO BASEADO EM SERVIÇOS WEB UTILIZANDO A ABORDAGEM REST

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Aprovada em __/__ /____

BANCA EXAMINADORA

_____

**Prof. Dr. Mário Antonio Meireles Teixeira (Orientador)**

Doutor em Ciência da Computação

Universidade Federal do Maranhão

_____

**Prof. Dr. Carlos de Salles Soares Neto**

Doutor em Informática

Universidade Federal do Maranhão

_____

**Prof. Msc. Geraldo Braz Junior**

Mestre em Ciência da Computação

Universidade Federal do Maranhão

A toda a minha família, em especial à minha esposa Fabrícia Brito e ao meu filho Gustavo Marins, ao qual dedico não só este trabalho, mas toda a minha vida.

# AGRADECIMENTOS

Em primeiro lugar agradeço aos meus pais, Tereza e Melquisedeque Marins, pelo carinho e responsabilidade com os quais me criaram, educaram e tornaram um homem capaz de alcançar meus objetivos. Agradeço também aos meus irmãos, Júnior e Viviane, pelo apoio e companheirismo.

Deixo meus sinceros agradecimentos ao meu orientador, Dr. Mário Antonio Meireles Teixeira, por sua disponibilidade, conhecimento e vontade de ajudar.

Agradeço especialmente à minha esposa e companheira Fabrícia Brito, pelo incentivo constante, o conforto nas horas difíceis e a presença nas horas felizes.

Agradeço a todos os meus amigos, que sempre ajudaram a suportar o cansaço daqueles dias mais estressantes.

Também agradeço a todos os meus colegas de trabalho, em especial Wagner, Holanda, Lourilene, Waldelene, Waldenê e Moizanilton pelo suporte contínuo durante toda a graduação e a todos que, de algum modo, deixaram sua contribuição à minha formação ao longo de todos esses anos.

*“Carpe diem quam minimum credula postero.”*

Horácio

# RESUMO

A crescente procura por sistemas interoperáveis e baseados em padrões amplamente disseminados motivou o surgimento de soluções de desenvolvimento de sistemas baseadas em serviços web. Neste seguimento destacam-se a arquitetura orientada a serviços (SOA), baseada na utilização do padrão SOAP para troca de mensagens e WSDL para descrição de serviços, e a arquitetura orientada a recursos (ROA), que é derivada do estilo arquitetural REST. Neste trabalho, a ROA é utilizada para o desenvolvimento de um guia eletrônico de programação (EPG), que disponibiliza as programações atualizadas das principais emissoras de TV brasileiras, de modo que as mesmas possam ser facilmente consumidas por aplicações clientes desenvolvidas em diversas linguagens de programação e executando sobre plataformas heterogêneas. Para exemplificar uma das possíveis aplicações clientes do EPG, é apresentado o desenvolvimento de um aplicativo Android responsável por consumir as informações disponibilizadas pelo serviço web de EPG e disponibilizá-las de forma clara e intuitiva, provendo interatividade e mobilidade ao usuário final.



Palavras-chave: Serviços Web, Arquitetura Orientada a Serviços, REST, EPG, Android, Desenvolvimento para Dispositivos Móveis.

ABSTRACT

The increasing demand for widespread interoperable standards-based systems has motivated the emergence of system development solutions based on web services. In this segment we highlight service-oriented architecture (SOA), based on the use of the SOAP standard for exchanging messages and WSDL for service description, and resource-oriented architecture (ROA), which is derived from REST architectural style. In this work, ROA is used for development of an electronic program guide (EPG), which provides updated schedules of major Brazilian TV stations, so that they can easily be consumed by client applications developed in various programming languages running on heterogeneous platforms. To illustrate one possible application of EPG customers, it is presented the development of an Android application responsible for consuming some information provided by the EPG web service and making it available in a clear and intuitive way, providing interactivity and mobility to end users.

Palavras-chave: Web Services, Service Oriented Architecture, REST, EPG, Android, Development for Mobile Devices.

LISTA DE ILUSTRAÇÕES

# LISTA DE SIGLAS

| | |
|---|---|
| API | Application Programming Interface |
| DTD | Document Type Definition |
| E/S | Entrada e Saída |
| EPG | Electronic Programming Guide |
| FLWOR | For, Let, Where, Order By, Return |
| HATEOAS | Hypermedia as Engine of Application State |
| HTML | HyperText Markup Language |
| HTTP | Hypertext Transfer Protocol |
| IETF | Internet Engineering Task Force |
| JAX-RS | Java API for RESTful web Services |
| JSON | Java Script Object Notation |
| MIME | Multipurpose Internet Mail Extensions |
| REST | Representational State Transfer |
| ROA | Resource-Oriented Architecture |
| RPC | Remote Procedure Call |
| SDK | Software Development Kit |
| SGBD | Sistema de Gerenciamento de Banco de Dados |
| SGML | Standard Generalized Markup Language |
| SMTP | Simple Mail Transfer Protocol |
| SOA | Service-Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| SQL | Structured Query Language |
| TV | Televisão |
| UDDI | Universal Description, Discovery and Integration |
| UML | Unified Modeling Language |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| VM | Virtual Machine |
| W3C | World Wide Web Consortium |
| WADL | Web Application Description Language |
| WSDL | Web Services Description Language |
| XML | Extensible Markup Language |

| | |
|---|---|
| XMLTV | XML Television |
| XPATH | XML Path Language |
| XQUERY | XML Query |

SUMÁRIO

Marins, Carlos Eduardo Ferreira

Desenvolvimento de um guia eletrônico de programação baseado em serviços web utilizando abordagens REST | Carlos Eduardo Ferreira Marins. – São Luis, 2012.

63 f.

Monografia (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Universidade Federal do Maranhão, 2012.

1. Serviços web 2. Arquitetura orientada a serviços 3. REST I. Título

CDU 004.738.5

# 1 INTRODUÇÃO

Os sistemas computacionais evoluíram de sistemas monolíticos para aplicações cada vez mais distribuídas, impulsionados pelo avanço da tecnologia de hardware e meios de comunicação. Na última década, o número de aplicações, sistemas e repositórios de informações coexistindo dentro das empresas e corporações demonstrou um crescimento sem precedentes. Toda essa complexa infraestrutura de software que suporta os processos de negócio nas empresas está, em sua maioria, baseada em sistemas legados e aplicações construídas sobre plataformas e tecnologias heterogêneas.

Seja pela necessidade de melhorar o gerenciamento de informações estratégicas de apoio à tomada de decisões ou de estabelecer novos planos de relacionamento com os clientes, as empresas tem efetuado grandes investimentos na integração de seus sistemas de informação.

Uma das abordagens mais recentes para a interoperabilidade entre sistemas é a utilização de serviços web. O consórcio W3C (*World Wide Web Consortium*) define serviço web como um sistema de software projetado para suportar interação máquina-máquina sobre uma rede, sendo identificado por um URI (*Uniform Resource Identifier*) e cujas interfaces e ligações são capazes de serem definidas, descritas, e descobertas como artefatos XML (*Extensible Markup Language*) (W3C, 2012).

Inicialmente os serviços web foram baseados na arquitetura orientada a serviços (SOA). SOA pode ser definida como uma abordagem arquitetural que visa construir sistemas a partir de um conjunto de componentes fracamente acoplados, denominados serviços, que podem ser combinados dinamicamente. Ela se fundamenta em serviços elementares, suas descrições e operações básicas de publicação, descoberta e associação (Papazoglou and Georgakopoulos, 2003). Para a realização dessas operações, SOA utiliza uma pilha de padrões baseados em XML que inclui SOAP (*Simple Object Access Protocol*), WSDL (*Web Services Description Language*), UDDI (*Universal Description, Discovery and Integration*), entre outros.

Buscando aumentar a escalabilidade e extensibilidade dos serviços web na Internet, Roy Thomas Fielding propôs em sua tese de doutorado um estilo

arquitetural que captura as características da Web, aplicando uma série de restrições arquiteturais. Este estilo denominou-se REST (*Representational State Transfer*). A Arquitetura Orientada a Recursos (ROA) é uma arquitetura que segue o estilo arquitetural REST, herdando dele, todas as restrições e elementos arquiteturais (Richardson and Ruby, 2007).

Com a implantação da TV (Televisão) digital no Brasil tornaram-se muito bem-vindos esforços de pesquisa e produção de softwares que forneçam suporte ou ampliem as possibilidades dessa tecnologia.

Uma extensa variedade de programas e serviços vem sendo disponibilizados através da TV Digital. A grande quantidade de conteúdos digitais disponíveis motiva a criação de aplicações que sejam capazes de coletar informações sobre os programas e serviços e apresentá-las de maneira organizada (Maia, 2011). Nesse contexto, é essencial a existência de aplicações que apresentem para os usuários as diferentes opções de conteúdo oferecidas, de modo a auxiliar sua escolha. Essas aplicações são chamadas de Guias Eletrônicos (Oliveira, 2011).

O principal objetivo deste trabalho é descrever o desenvolvimento de um mecanismo interoperável, baseado em serviços web *RESTful*, que disponibilize a programação atualizada das principais emissoras de TV brasileiras e internacionais, de modo que as mesmas possam ser facilmente consumidas por aplicações clientes desenvolvidas em diversas linguagens de programação e executando sobre plataformas heterogêneas. Para tanto, é arquitetada uma solução para o agrupamento e resumo das informações de programação disponíveis em diversos provedores de conteúdo existentes na internet.

Paralelamente, objetiva-se desenvolver um software cliente que seja capaz de recuperar essas informações de programação e disponibilizá-las através de uma interface amigável e interativa. Este aplicativo deve ser desenvolvido sobre a plataforma Android, com o intuito de facilitar o acesso às informações através da mobilidade provida por dispositivos portáteis como celulares e *tablets*.

O presente documento está organizado da seguinte forma. O Capítulo 2 apresenta uma visão geral da solução para interoperabilidade baseada em serviços web, incluindo algumas definições, a abordagem orientada a serviços, sua arquitetura e principais padrões web envolvidos. Este capítulo também introduz

conceitos relativos ao estilo arquitetural REST, trazendo no final um breve comparativo entre as duas abordagens.

O Capítulo 3, por sua vez, discorre sobre a plataforma de desenvolvimento para aplicativos móveis, o Android, sua arquitetura e principais elementos.

O Capítulo 4 descreve o desenvolvimento do guia eletrônico de programação propriamente dito, especificando sua arquitetura através de diagramas UML (*Unified Modeling Language*) e detalhando cada componente da aplicação individualmente.

Finalmente, o Capítulo 5 destaca os principais resultados obtidos e propõe trabalhos futuros relacionados.

# 2 SERVIÇOS WEB

O ambiente corporativo mundial tem se tornado cada vez mais dinâmico e globalizado. Empresas se expandem, diversificam seus negócios, são incorporadas por outras, participam de fusões, em resumo, estão cada vez mais inter-relacionadas e dependentes umas das outras. Toda essa necessidade de comunicação é dificultada pelo fato de que as empresas já possuem uma complexa infraestrutura de tecnologia da informação com interface proprietária, o que compromete a troca de informações estratégicas entre sistemas de diferentes empresas. Mesmo uma empresa isolada pode possuir uma grande quantidade de sistemas e componentes de software responsáveis por áreas particulares de negócio. Tais sistemas são basicamente constituídos de aplicações construídas em cima de padrões proprietários, com quase proibitivas restrições de interoperabilidade.

É nesse contexto que surge a arquitetura orientada a serviços (SOA). Tendo como principal objetivo promover a interoperabilidade entre sistemas de informação, independentemente da tecnologia, plataforma ou paradigma em que foram desenvolvidos, SOA propõe a utilização de padrões abertos da Internet para conectar componentes e sistemas heterogêneos. A tecnologia que melhor se adaptou aos princípios da arquitetura orientada a serviço foram os *Web Services*.

Um *Web Service*, ou serviço web, é definido pelo W3C como sendo uma aplicação de software identificada por um URI, cujas interfaces e ligações são capazes de serem definidas, descritas, e descobertas como artefatos XML. Um serviço web suporta interações diretas com outros agentes de software usando mensagens baseadas em XML trocadas via protocolos da Internet (W3C, 2012).

Os benefícios dos serviços web incluem a dissociação entre as interfaces de serviços e suas implementações, a possibilidade de associação dinâmica a serviços, e um aumento da interoperabilidade entre linguagens e plataformas (Ferris e Farrell, 2003).

## 2.1 Arquitetura Orientada a Serviços

A Arquitetura Orientada a Serviços (SOA) pode ser definida como uma abordagem arquitetural que visa construir sistemas a partir de um conjunto de componentes fracamente acoplados, denominados serviços, que podem ser combinados dinamicamente. Estes serviços são componentes de software auto-contidos, com granularidade que permita uma alta coesão e um alto índice de reusabilidade.

Além da interoperabilidade, a arquitetura orientada a serviços promove maior flexibilidade, na medida em que facilita o desenvolvimento ágil de soluções que acompanham a fluidez das transformações ocorridas no âmbito de negócios. Essa agilidade no desenvolvimento se deve em parte à composição de serviços básicos já existentes para construção de novos serviços, e em parte ao fato de que os serviços exibem uma interface pública bem definida, logo, alterações podem ser realizadas em um serviço individual sem comprometer o funcionamento de outros serviços que dependam dele, desde que não se mude sua interface. Os elementos básicos da arquitetura orientada a serviços são ilustrados na Figura 2.1.

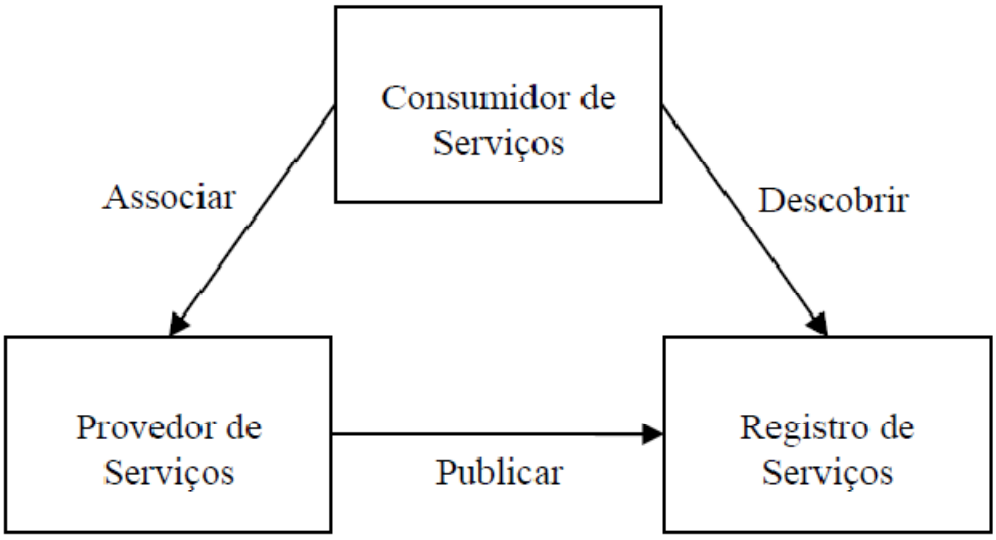


**Figura 2.1** – Arquitetura Orientada a Serviços. Fonte: [Maciel, 2007]

Todo serviço deve possuir uma especificação formal de sua interface e dos procedimentos necessários para que os consumidores de serviços possam se conectar a ele, a qual é tratada como contrato na literatura. Essa descrição é formalizada utilizando-se o padrão WSDL e o provedor de serviços pode, eventualmente, publicá-la em um repositório de serviços.

Um repositório de serviços facilita a descoberta de serviços e a aquisição de informações de uso, como localização física, dados do provedor, contatos pessoais, preço, categoria, potenciais beneficiários, taxas de utilização, restrições técnicas, questões de segurança e níveis de disponibilidade de serviços (Krafizig et al, 2004). Desta forma, o repositório ou registro de serviços provê uma maneira uniforme de o consumidor descobrir quais serviços disponíveis estão aptos a satisfazer seus interesses. Uma vez descoberto o serviço, o consumidor pode associar-se a ele para utilizar os recursos, dados ou funcionalidades fornecidas.

A seqüência publicar/descobrir/associar e os elementos provedor, consumidor e registro de serviços constituem a essência da arquitetura orientada a serviços. Não obstante, podem existir implementações SOA sem a figura do registro de serviços, caso em que a associação deve ser feita diretamente entre o consumidor e o provedor do serviço.

## 2.2 Padrões

### 2.2.1 XML

XML (*eXtensible Markup Language*) é uma linguagem de marcação derivada da *Standard Generalized Markup Language* (SGML), que tem por objetivo descrever dados e documentos sob um formato de texto simples, padronizado e facilmente transportável via protocolos padrões da Internet (Benz e Durant, 2003).

A estrutura de documentos XML é baseada no conceito de elementos ou *tags*. Elementos são porções de informação delimitadas por estruturas identificadas e que possuem valor semântico para a aplicação. Eles são iniciados por um identificador (tag) escrito entre sinais de maior e menor, como em <book>, e tem como delimitador final o marcador inicial, acrescido de uma barra após o sinal de menor, como em </book>. Para caracterizar ou especificar elementos, XML possui o conceito de atributos, que nada mais são que pares chave/valor colocados dentro do marcador inicial, antes do sinal de menor, como em <book isbn=”2585”>.

Todo documento XML deve possuir obrigatoriamente um único elemento mais externo denominado raiz, dentro do qual todos os outros elementos podem ser

aninhados. Essa característica se deve ao fato de que os documentos XML são construídos de modo hierárquico formando uma árvore de elementos.

Documentos XML devem obedecer a uma série de regras sintáticas que visam eliminar ambigüidades e torná-los passíveis de processamento através do uso de analisadores sintáticos (*parsers)*. Um documento é dito bem-formado quando todas essas restrições sintáticas são atendidas. Por outro lado, outras regras estruturais podem ser adicionadas a documentos XML através do uso de *Document Type Definition* (DTD) ou XML *Schema*. Quando um documento é bem-formado e atende todas as restrições estruturais impostas a ele, é classificado como um documento XML válido.

XML tornou-se o padrão para troca de informações na internet, recomendado pelo *World Wide Web Consortium* (W3C) por suas características como simplicidade, interoperabilidade, expressividade, portabilidade, separação entre dados e apresentação, aplicabilidade e flexibilidade.

#### 2.2.1.1 XML *Namespaces*

Documentos XML utilizam, freqüentemente, elementos já existentes definidos em outras aplicações XML. Embora essa característica seja desejada, na medida em que permite o reuso de elementos, ela introduz dois problemas de ordem prática. O primeiro é o reconhecimento, que se trata de como identificar a qual fonte pertence um elemento. O processador XML precisa reconhecer o elemento e identificar sua origem, a fim de tratá-lo adequadamente e validar sua estrutura de acordo com as regras do documento original. O segundo problema diz respeito à ocorrência, num mesmo documento, de dois ou mais elementos com o mesmo identificador, e provenientes de fontes diferentes. Conhecido como colisão, esse problema impossibilita o processador XML de descobrir quais regras ou comportamentos devem ser atribuídos a cada um dos elementos conflitantes.

Para solução dos dois problemas citados, XML oferece o conceito de *namespaces* (domínios). *Namespaces* constituem um método para separar e identificar nomes duplicados de elementos em um documento XML, além de comumente serem usados como identificadores para descrever os tipos de dados e outras informações. Outra opção é seu uso como palavras-chave únicas, indicando

que semânticas de processamento específicas devem ser interpretadas quando os documentos são processados (Newcomer, 2002).

A estratégia utilizada para resolução dos problemas parte da adoção de qualificadores, que são prefixos únicos anexados aos elementos. Na qualificação de elementos XML é utilizado um nome qualificado (*QName*), uma combinação de identificador de *Namespace* e nome local do elemento (Maciel, 2007). Deste modo, é possível tanto identificar a origem de um determinado elemento XML, sanando o problema do reconhecimento, quanto distinguir entre dois elementos com mesmo nome, resolvendo problemas de conflito.

Para a definição de *Namespaces* são utilizados URIs (*Uniform Resource Identifier*), que são cadeias de caracteres responsáveis por identificar unicamente um recurso disponível na rede, normalmente a Internet. Na pratica é usado um subconjunto dos URIs denominado URL (*Uniform Resource Locator*), que além de identificar um recurso, também fornece informações de como e onde acessá-lo.

#### 2.2.1.2 DTD

XML pode ser considerada uma metalinguagem, à medida que possui recursos para a definição de gramáticas que caracterizam linguagens para classes de documentos específicos, com conjunto de elementos, atributos e regras de composição bem determinados (Teixeira, 2012). A primeira forma de associar a um documento XML um conjunto de regras sobre sua formação e estruturação foram os DTDs.

DTDs foram inicialmente definidos como parte da linguagem SGML e usam uma gramática formal para descrever a estrutura e sintaxe de um documento XML, incluindo possíveis valores permitidos para cada elemento, a ordem em que os elementos devem estar dispostos, quais elementos podem ser aninhados, quais elementos ou atributos devem, obrigatoriamente, ser definidos, etc. A partir das regras definidas no DTD é possível automatizar a validação da estrutura de um documento XML usando analisadores sintáticos, o que visa garantir a coerência do documento em relação à aplicação específica em que está sendo utilizado.

Embora cumpra com relativa eficácia a tarefa de validação de documentos XML, esse mecanismo apresenta limitações que sugerem a adoção de alternativas.

Entre as desvantagens pode-se citar: não utiliza sintaxe XML para definição das regras de formação de documentos, o que aumenta a complexidade de escrita além de exigir a utilização de outro interpretador; não suporta XML Namespaces, o que restringe a capacidade de reutilização; não oferece suporte a tipos de dados, o que prejudica a integração com bancos de dados e a aplicação de restrições mais elaboradas.

#### 2.2.1.3 XML Schema

XML *Schema* é a linguagem de definição de esquema recomendada pelo W3C, expressa em XML 1.0, que se destina a descrever a estrutura e restringir o conteúdo de documentos escritos em XML. Ao aplicar um esquema, torna-se possível trocar informações entre as aplicações com maior confiança e reduz-se a necessidade de programação para testar e confirmar a estrutura de um documento, ou para confirmar que os dados numa determinada parte do documento correspondem a um tipo de dados particular (Wyke e Watt, 2002).

Em contraponto ao DTD, XML *Schema* possui suporte a XML *Namespaces*, possui maior controle sobre o número de ocorrências de um determinado elemento, possibilita a reutilização de código, possui uma maior compatibilidade com bancos de dados relacionais e possui uma rica hierarquia de tipos de dados, permitindo, inclusive, a criação de tipos customizados pelo desenvolvedor.

Um dos pontos que garantiram o sucesso da tecnologia XML foi o fato de que ao se enviar o arquivo XML e seu respectivo XML *Schema*, transmitem-se os dados propriamente ditos, sua estrutura e uma forma de validação que pode ser facilmente aplicada utilizando-se um analisador XML comum.

### 2.2.2 WSDL

Como comentado anteriormente, um dos pilares da arquitetura orientada a serviços é a publicação de uma interface de serviço bem definida. Neste contexto, a Linguagem de Descrição de Serviços Web (WSDL) foi criada para descrever e publicar os formatos e protocolos de um serviço web de maneira padronizada (Newcomer, 2002). WSDL é escrita em XML e seus elementos contêm a descrição

dos dados e procedimentos relativos a um serviço web, utilizando um ou mais esquemas XML, de modo a permitir que tanto o emissor quanto o receptor entendam os dados que estão sendo trocados.

Mais formalmente, WSDL é um formato XML para descrever serviços de rede como um conjunto de pontos de acesso que funcionam através de mensagens contendo informação orientada ao documento ou orientada ao procedimento. As operações e as mensagens são descritas de forma abstrata e ligadas concretamente ao protocolo de transporte e ao formato da mensagem para definir um ponto de acesso. Pontos de acesso concretos e relacionados são combinados em pontos de acesso abstratos (serviços). WSDL é extensível para permitir que a descrição de pontos de acesso e suas mensagens não levem em consideração quais formatos de mensagem ou protocolos de transporte estão sendo usados na comunicação (WEERAWARANA, 2005).

#### 2.2.2.1 Estrutura WSDL

A estrutura geral de um documento WSDL é composta pelos elementos mostrados através da Figura 2.2.

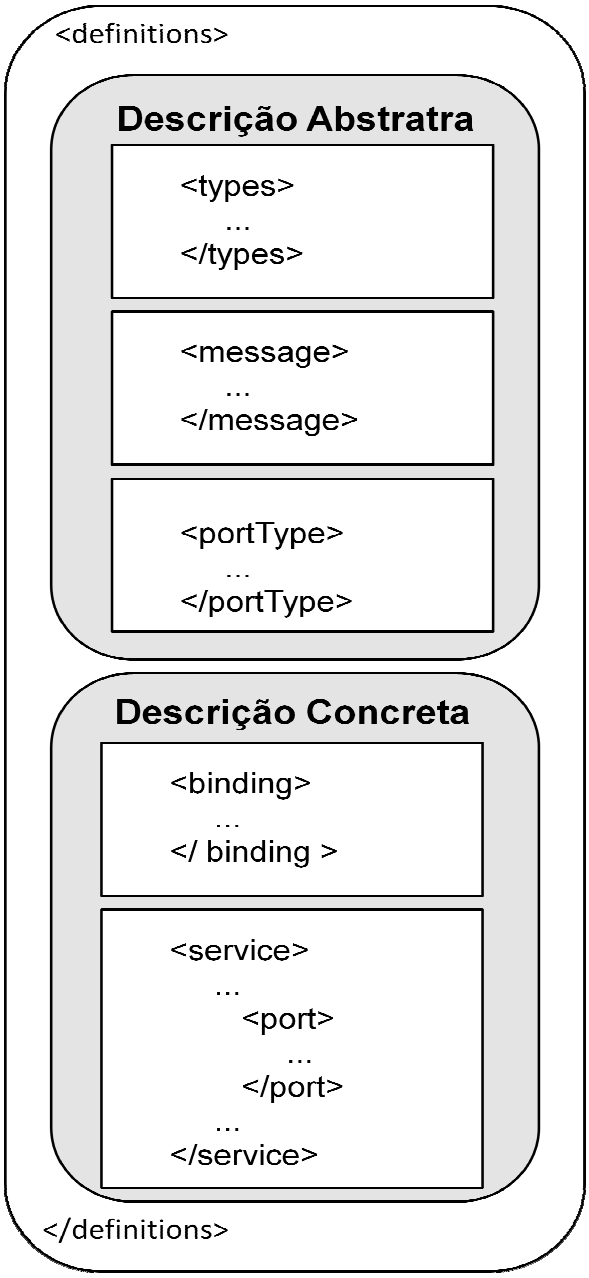


**Figura 2.2** – Estrutura do documento WSDL

A primeira consideração a ser feita a respeito da estrutura de um documento WSDL é a clara separação entre o projeto do serviço web e sua implementação, ou seja, são reservados elementos específicos para a descrição abstrata do serviço, como quais os tipos de dados utilizados, quais os parâmetros de entrada e saída e quais operações estão disponíveis. Outro conjunto de elementos, por sua vez, é responsável por definir o protocolo e o endereço onde o serviço estará disponibilizado. Devido a essa divisão estrutural, um mesmo serviço pode ser disponibilizado através de endereços e protocolos diferentes.

A raiz de todo documento WSDL é o elemento *<definitions>,* onde são declarados os *namespaces* que serão utilizados ao longo do documento.

Os elementos responsáveis pela descrição abstrata do serviço web são *<type>*, *<message>* e *<portType>*.

O elemento *<type>* engloba certos tipos predefinidos usados na descrição da interface, ou seja, permite especificar os tipos que serão trocados nas mensagens de ida e volta do serviço. Esquemas XML são permitidos para definir estruturas de dados complexas.

O elemento *<message>* descreve as mensagens trocadas entre o serviço e o consumidor, em termo dos elementos de dados que as compõem. Cada mensagem pode conter um ou mais elementos *<part>*, que definem o conteúdo da mensagem, representando os parâmetros que são passados e a resposta que o serviço retorna.

A interface propriamente dita é representada através do elemento *<portType>*. Este elemento é composto por um conjunto de operações *(<operation>*), as quais possuem uma documentação da funcionalidade oferecida e fazem referência às mensagens descritas em *<message>*, classificando-as em parâmetros de entrada e saída.

Como parte da especificação concreta do serviço, o elemento *<binding>* associa as operações contidas em *<portType>* a protocolos de transporte específicos, como SOAP, HTTP (*Hypertext Transfer Protocol*) ou SMTP (*Simple Mail Transfer Protocol*), por exemplo. Completando as informações de como acessar o serviço, existem os elementos *<service>* e *<port>* que indicam o endereço onde o serviço está disponível.

### 2.2.3 SOAP

SOAP é um protocolo baseado em XML, criado para promover a troca de informações entre componentes de um ambiente de computação distribuída. Embora seu principal objetivo seja a definição de chamadas a procedimentos remotos (RPC) transportáveis via HTTP, SOAP pode ser usado em uma variedade de sistemas de mensagens e pode ser entregue através de uma variedade de protocolos de transporte. Sua especificação define nada mais que um simples envelope, baseado em XML, para a informação sendo transferida e um conjunto de

regras para tradução de aplicações e tipos de dados dependentes de plataforma em representações XML (Tidwell et al, 2001).

O protocolo é composto por partes distintas. A primeira parte é o envelope, usado para descrever o conteúdo da mensagem e algumas informações sobre como processá-la. A segunda parte consiste de regras para codificar instâncias de tipos de dados específicos. A última parte descreve a aplicação do envelope e as regras de codificação de dados para representar chamadas e respostas RPC, incluindo o uso de HTTP como camada de transporte (Englander, 2002).

#### 2.2.3.1 Envelope SOAP

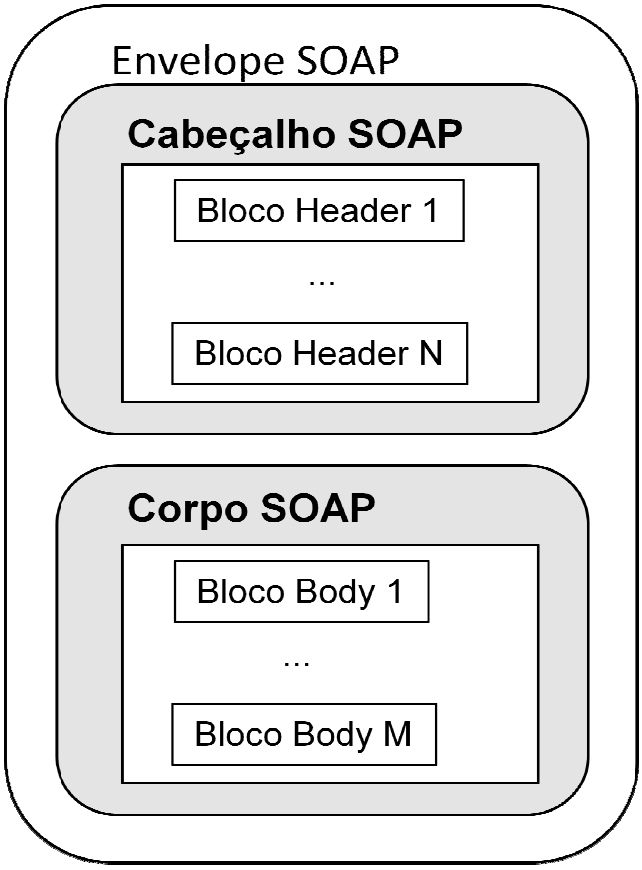


**Figura 2.3** – Estrutura da mensagem SOAP

Todas as mensagens SOAP são encapsuladas em um elemento XML chamado *<envelope>*, que é a raiz de um documento SOAP. O elemento *<envelope>* possui como subelementos *<header>* e *<body>*.

O elemento <*header*> é opcional, mas quando existir deve ser o primeiro subelemento de <*envelope*>. O cabeçalho é um mecanismo de extensão que permite o registro de qualquer tipo de informação que se encontra fora do escopo semântico da mensagem no corpo, mas é, no entanto, útil ou mesmo necessária para processar a mensagem corretamente. Este elemento pode ser usado para prover mecanismos de autenticação, gerenciamento de transação, autorização de pagamento, etc.

O elemento corpo, identificado pelo marcador <*body*>, é obrigatório e contém a solicitação ou resposta SOAP propriamente dita. Neste elemento pode-se encontrar uma mensagem de estilo RPC que contém o nome do método e seus parâmetros, ou uma mensagem de sentido único e as suas partes interessadas, ou uma falha e seus detalhes (Englander, 2002). Como mecanismo para reportar ao emissor a ocorrência de algum erro ou falha no processamento de mensagens, o corpo possui um subelemento denominado <*fault*>.

## 2.3 Abordagem REST

*Representational State Transfer* (REST) é um estilo arquitetural híbrido para sistemas hipermídia distribuídos, derivado de vários estilos arquiteturais de software em rede (Fielding, 2000), que define um conjunto de princípios que podem ser aplicados na construção de sistemas com uma arquitetura orientada a recursos (ROA). Sistemas construídos segundo os princípios REST são chamados aplicativos *RESTful*.

REST está baseado no conceito de recursos que são identificados e disponibilizados através de URIs. Essas URIs são acessíveis a partir de links que retornam as representações dos recursos. Quando o cliente acessa uma dessas representações ele passa a ocupar um novo estado em relação à aplicação como um todo. Cada representação pode conter links para outros recursos, o que permite ao cliente navegar pela aplicação a partir da transição entre estados, daí o termo *Representational State Transfer.*

A elaboração de REST foi baseada em estilos arquiteturais como cliente/servidor, *stateless*, *cache*, código sob demanda e interface uniforme.

O estilo cliente/servidor define a separação das entidades participantes da computação em dois papéis: um servidor, responsável por disponibilizar um conjunto de serviços, e um cliente, que faz uso destes serviços.

Em REST, toda comunicação deve ser *stateless*, ou seja, qualquer informação necessária para atender uma requisição deve estar contida na própria requisição. Isso implica que toda informação de contexto e de estado da sessão está contida exclusivamente no cliente (Fielding, 2000).

A restrição arquitetural *cache*, exige que as respostas enviadas a clientes sejam marcadas, implícita ou explicitamente, como passíveis ou não de *cache*. Caso positivo, o cliente pode armazenar a resposta para usá-la em caso de novas requisições equivalentes, diminuindo, assim, o trafego na rede e melhorando o desempenho percebido pelo usuário.

Através da restrição opcional código sob demanda, REST permite que funcionalidades do cliente sejam estendidas através do download e execução de código na forma de *applets* ou *scripts*. Embora essa característica possa simplificar o cliente, reduzindo o número de recursos a serem pré-implementados, também pode acarretar em perda de visibilidade (Fielding, 2000).

A principal distinção entre REST e outro estilos arquiteturais para Web é a ênfase na utilização de uma interface uniforme (Fielding, 2000). Em decorrência desta restrição arquitetural, o conjunto de métodos para cada elemento do sistema é conhecido e padronizado, sendo que a cada execução de um método, sua semântica é visível. Em termos mais específicos, o princípio da interface uniforme define que o servidor deve ser capaz de determinar o que deve ser feito ao receber uma requisição HTTP em uma URI apenas pela observação do método presente nessa requisição (Pautasso et al. 2008). Aplicando o princípio de generalização da engenharia de software à interface dos componentes, a arquitetura é simplificada e a visibilidade melhorada.

### 2.3.1 Arquitetura orientada a recursos

A Arquitetura Orientada a Recursos (ROA) é uma arquitetura que segue o estilo arquitetural REST, herdando dele, todas as restrições e elementos arquiteturais (Richardson and Ruby, 2007).

ROA é uma arquitetura voltada para serviços web, sendo implementada a partir de tecnologias da Web como o protocolo HTTP e a linguagem de marcação XML.

O HTTP é um protocolo da camada de aplicação para sistemas hipermídia, colaborativos e distribuídos, baseado no modelo de comunicação requisição/resposta que pode ser utilizado para realizar diferentes tarefas. O HTTP é um protocolo sem manutenção de estado (*stateless*) e define como é feita a troca de mensagens entre o cliente e o servidor; ou seja, esse protocolo define como um cliente requisita recursos em um servidor e como este responde a tais requisições (França et al, 2011). Como o HTTP possibilita a utilização de diferentes representações, as aplicações podem ser construídas independentemente da forma como os dados serão transferidos (RFC2616, 1999).

#### 2.3.1.1 Recursos e identificadores

Um recurso é qualquer coisa que seja importante o suficiente para ser referenciado como algo em si (Richardson and Ruby, 2007). Em uma aplicação *RESTful* um recurso pode ser um registro em um banco de dados, um arquivo em disco, um serviço, uma coleção de outros recursos, etc.

Em ROA, todos os recursos devem ser identificados unicamente através de URIs. É importante que elas sigam uma estrutura intuitiva e que variem de forma previsível, pois devem refletir os relacionamentos entre os recursos que identificam. Outra função das URIs é endereçar adequadamente os recursos na Web.

#### 2.3.1.2 Representação

Todo recurso possui uma representação, que nada mais é que o formato das mensagens trocadas entre o servidor e o cliente através dos métodos HTTP. Representações são usadas para capturar o estado atual ou previsto dos recursos (Fielding, 2000).

Um recurso pode possuir várias representações, como texto simples, XML, HTML (*HyperText Markup Language*) ou JSON (*Java Script Object Notation*), por exemplo. A escolha da melhor representação depende da necessidade do cliente. A

negociação de conteúdo pode ser efetuada através de metadados enviados nos cabeçalhos das requisições, por exemplo, o cabeçalho *Accept* pode ser usado em uma requisição para restringir o tipo MIME (*Multipurpose Internet Mail Extensions*) obtido na resposta (Silvestre e Polônia, 2008).

#### 2.3.1.3 Vinculação de recursos

A dinâmica de sistemas *RESTful* é conseguida através da navegação entre recursos, que, por sua vez, é alcançada com a utilização de enlaces embutidos nas representações.

Quando o servidor envia uma representação com enlaces, ele está enviando para o cliente uma série de URIs com estados possíveis da aplicação. Esta propriedade é perfeitamente representada através de grafos, onde os nós são os recursos e as arestas são os enlaces ligando-os (Silvestre e Polônia, 2008).

O princípio da vinculação de recursos está relacionado com o uso da abordagem HATEOAS (*Hypermedia as Engine Of Application State*). Nessa abordagem, as aplicações são consideradas como uma máquina de estado onde cada página representa um estado e os links representam todas as possíveis transições de estado a partir do estado corrente.

#### 2.3.1.4 Interface Uniforme

A interface uniforme define o conjunto de operações possíveis para cada recurso. Em ROA, os métodos das requisições HTTP (*GET, POST, PUT* ou *DELETE*) são utilizados para indicar ao provedor do recurso a ação que deve ser realizada (França et al, 2011).

• *GET*: Obtêm a representação de um recurso, assim como os meta-dados associados.

• *DELETE*: Remove um recurso.

• *PUT*: Cria ou atualiza um recurso na URI especificada.

• *POST*: Cria um recurso subordinado ou anexa informações a um recurso, respondendo com a URI do novo recurso.

• *OPTIONS*: Lista quais os métodos que podem ser executados no recurso.

• *HEAD*: Idêntico ao *GET*, mas omitindo a representação.

## 2.4 WS-* versus REST

WS-* é o nome dado ao conjunto de padrões e especificações relacionados ao desenvolvimento de serviços web tradicionais. Além dos padrões citados neste trabalho como SOAP e WSDL, também podem ser citados: UDDI, *WS-Notification, WS-Addressing, WS-Transfer, WS-Policy, WSSecurity, WS-Trust, WS-ReliableMessaging, WS-Transfer, WS-I Basic Profile, WSTransaction*, entre outros.

A Figura 2.4 exibe um comparativo entre a abordagem REST e a WS-*, focando nos padrões utilizados por cada uma delas.

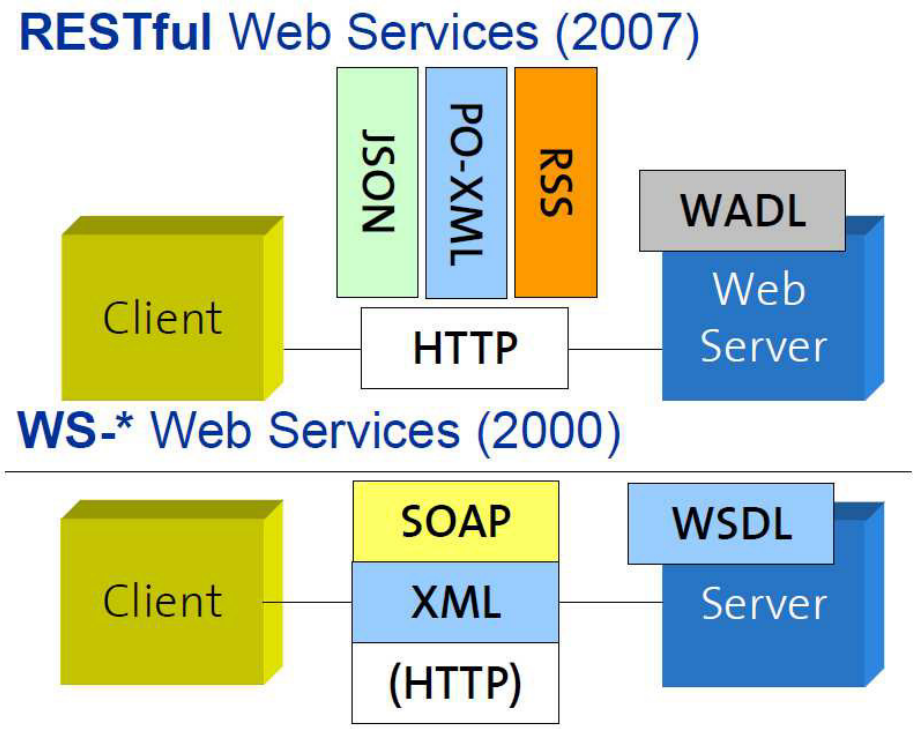


**Figura 2.4** – Comparativo RESTful x WS-* Web Services. Fonte: [Pautasso et al, 2008]

Uma das principais diferenças entre as duas abordagens refere-se à utilização do padrão SOAP para transporte das mensagens em serviços web tradicionais, o que garante maior transparência e independência de protocolo, visto que a mesma mensagem, no mesmo formato, pode ser transportada através de

diversos protocolos. A abordagem REST, por outro lado, é considerada mais simples, pois utiliza padrões W3C e IETF (Internet Engineering Task Force) bem conhecidos (HTTP, XML, URI, MIME) e necessitam de infraestrutura que já se tornou pervasiva (Pautasso et al. 2008).

Outro ponto importante diz respeito à necessidade de descrição dos serviços web tradicionais através do padrão WSDL. Em REST, graças ao uso de URIs e hyperlinks, tem-se mostrado que é possível descobrir recursos web sem a utilização de uma abordagem baseada em registro compulsório em um repositório centralizado (Pautasso et al. 2008). Não obstante, existem iniciativas para a definição de um padrão para descrição de serviços baseados em REST como a WADL (*Web Application Description Language*).

Também vale destacar que, em REST, o servidor não armazena nenhuma informação de estado, existe uma interface uniforme baseada nos verbos HTTP e os recursos são acessados através de URIs, enquanto serviços baseados em WS-* permitem a definição de informações contextuais no servidor, não possuem interface uniforme, facultando o uso de quaisquer operações, as quais são publicadas e acessadas como métodos remotos.

# 3 PLATAFORMA ANDROID

## 3.1 O que é Android

Com a popularização massiva dos celulares e dispositivos móveis com cada vez maior poder de processamento, a computação demonstra uma tendência natural para aplicações baseadas em plataformas para ambientes móveis. Uma das mais recentes e bem sucedidas é a plataforma Android.

A plataforma Android foi lançada em 2008 pela Google, por meio do consórcio Open Handset Alliance e, segundo (Android developers, 2012), é uma pilha de software para dispositivos móveis que inclui um sistema operacional, middleware e aplicações denominadas críticas. O Android SDK (*Software Development Kit*) fornece as ferramentas e APIs (*Application Programming Interfaces*) necessárias para começar a desenvolver aplicações na plataforma Android usando a linguagem de programação Java.

A plataforma Android é um ambiente para dispositivos móveis que inclui um sistema operacional baseado no kernel 2.6 do Linux, um ambiente rico para desenvolvedores, além de suporte às tecnologias presentes na maioria dos dispositivos móveis (tela sensível ao toque, sistema de localização, acelerômetro, etc) e funcionalidades convencionais de telefones celulares (Sousa, 2011).

## 3.2 Arquitetura

A arquitetura da plataforma Android é organizada em camadas que possuem atribuições específicas e gerenciam seus próprios processos. A Figura 3.1 ilustra a pilha de software da plataforma.

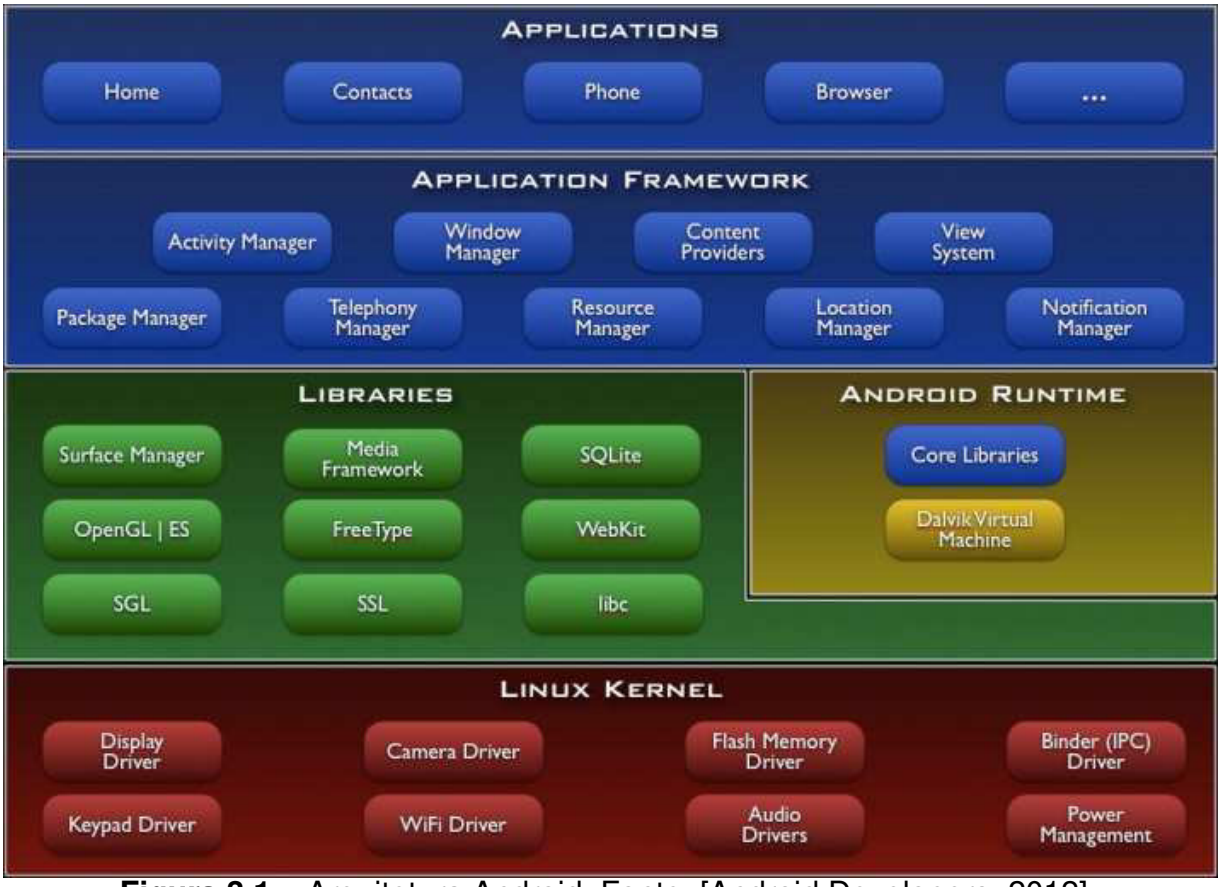


**Figura 3.1** – Arquitetura Android. Fonte: [Android Developers, 2012]

A camada superior oferece um conjunto de aplicações de alto nível que são acessadas diretamente pelo usuário final, dentre as quais destacam-se clientes de email, calendário, mapas, navegador, jogos e aplicações de terceiros. Logo abaixo da camada de aplicação, estão localizados os frameworks de aplicação, que incluem os programas que gerenciam as funções básicas do dispositivo como alocação de recursos, gerenciamento do telefone, oferecem informações sobre localização, notificações, alarmes etc.

Logo abaixo na pilha, Android inclui bibliotecas escritas em C/C++, bibliotecas de multimídia, visualização de camadas 2D e 3D, funções para navegadores web, funções de aceleradores de hardware, renderização 3D, funções para gráficos,fontes bitmap e vetorizadas e funções de acesso a banco de dados SQLite, dentre outras funcionalidades expostas aos desenvolvedores através dos frameworks de aplicação

No mesmo nível das bibliotecas está o ambiente de execução do Android (*Android Runtime*) que possui um conjunto de bibliotecas do núcleo java e instâncias da máquina virtual Dalvik, comentada a seguir neste capítulo.

Na base da pilha localiza-se o kernel do Linux, responsável pelo gerenciamento de memória, de processos e de E/S, pilha de rede, *drivers* de dispositivos, entre outros. É nesta camada que ocorre a abstração entre o hardware e o restante da pilha de software.

### 3.3 Máquina Virtual Dalvik

A plataforma Android possui uma máquina virtual própria, otimizada para dispositivos com poucos recursos computacionais e projetada especificamente para uso em ambientes embarcados. Embora seja muitas vezes referida como uma máquina virtual Java, isto não é estritamente preciso, visto que o bytecode em que opera não é bytecode Java. Em vez disso, uma ferramenta chamada dx, incluída no SDK Android, transforma os arquivos de classes compiladas por um compilador Java convencional em arquivos em formato .dex (Dalvik, 2012). Todos os arquivos .dex gerados são empacotados em um único arquivo com extensão .apk, que é o formato padrão para distribuição de aplicações Android.

Dalvik possui um modelo em que cada aplicação é executada sob uma instância da máquina virtual (VM), logo, cada aplicação associada a uma VM tem o seu espaço de endereçamento separado (Ferreira, 2010). Outra característica marcante é o fato desta VM usar o kernel Linux do dispositivo para lidar com funcionalidades de baixo nível, incluindo segurança, threading e gerenciamento de processos e memória (Meier, 2010).

### 3.4 Principais elementos

Existem quatro tipos principais de componentes Android: *activities*, *services*, *broadcast receivers* e *content providers*.

#### 3.4.1 Activity

Atividades representam interfaces visuais da aplicação através das quais ocorre toda a interação com o usuário. Em geral, uma aplicação Android é composta por várias atividades, sendo que uma delas é carregada no inicio da aplicação, a

qual é denominada atividade principal. A partir da atividade principal, outras atividades podem ser acionadas, cada uma com funções específicas, colaborando para a dinâmica do sistema.

Apenas uma atividade pode estar ativa por vez, por isso, cada vez que uma nova atividade começa, a atividade anterior é interrompida e colocada em uma pilha (*back stack*). Mais detalhes a respeito do ciclo de vida de atividades serão discutidos adiante.

O conteúdo visual de uma atividade é controlado por visões (*Views*), objetos derivados da classe *android.view.View*. Cada visão é reponsável por um espaço retangular da janela. Visões podem ser alternadas e modificadas a qualquer momento pela atividade.

#### 3.4.1.1 Ciclo de Vida das Atividades

Iniciar uma atividade é um procedimento muito custoso, visto que envolve a criação de novos processos Linux, alocação de memória para todos os objetos de interface gráfica, tradução de todos os objetos para *layouts* XML e a criação de toda a tela (Gargenta, 2011). Para evitar desperdício de processamento, iniciando uma atividade várias vezes, Android gerencia o ciclo de vida de todas as atividades via *activity manager*. O ciclo de vida das atividades corresponde aos estados em que elas podem estar no decorrer do tempo e as transições entre esses estados, como visto na Figura 3.2.

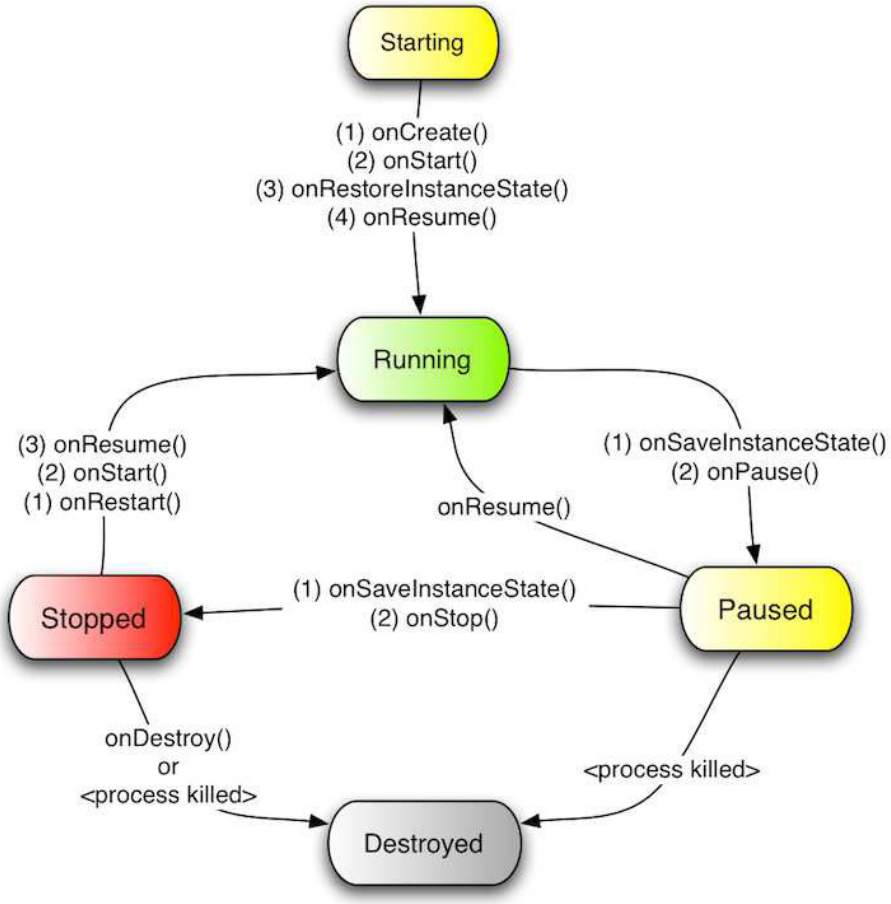


**Figura 3.2** – Ciclo de vida das atividades. Fonte: [Oreilly, 2012]

Os estados possíveis para uma atividade são *starting* (iniciando), *running* (executando), *paused* (em pausa), *stopped* (parado) e *destroyed* (destruído). A cada mudança de estado são enviadas chamadas a métodos de *callback* que podem ser implementados pelos desenvolvedores para controlar o estado ou prover funcionalidades.

Quando uma atividade não existe na memória ela se encontra no estado iniciando, após ser carregada, ela passa ao estado executando. Neste estado a atividade está em primeiro plano, pronta para receber interações do usuário, ou seja, está no topo da pilha de execução.

Caso a atividade perca o foco da interação com o usuário, mas continue visível na tela, ela passa ao estado em pausa. Qualquer atividade neste estado ainda mantém alta prioridade para obtenção de memória e recursos, já que a interação pode ser retomada muito em breve.

Caso uma atividade deixe de ser visível ao usuário, isto é, tenha sido sobreposta por outra atividade, ela passa ao estado parado. Neste estado a atividade continua mantendo suas informações, mas perde prioridade na obtenção de recursos do sistema e pode ser removida da memória a qualquer momento, quando passa ao estado destruído.

### 3.4.2 Services

A classe *Services* é usada para executar um serviço em segundo plano, geralmente vinculado a algum processo que deve ser executado por tempo indeterminado e possui um alto consumo de recursos, memória e unidade central de processamento (Ferreira, 2010).

Como serviços não possuem uma interface visual, geralmente são iniciados a partir de atividades. Quando um serviço é acessado através do método *startService()*, o serviço executa totalmente dissociado da atividade que o iniciou. Neste caso, o serviço, normalmente, realiza uma tarefa e não retorna nenhum resultado para o componente que o invocou. Outra opção é iniciar o serviço através do método *bindService()*. Neste caso, o serviço é dito vinculado e oferece uma interface cliente-servidor que permite aos componentes interagir com ele, enviar pedidos e obter resultados.

### 3.4.3 Broadcast Receiver

*Broadcast Receivers* (receptor de transmissão) constituem uma implementação Android para um mecanismo produtor/consumidor, ou mais precisamente, um padrão do tipo observador. Através desse mecanismo uma aplicação pode registrar um receptor associado a um evento, de modo que no momento em que o evento ocorre o método *onRecieve()* é disparado.

Muitos receptores de transmissão iniciam através do sistema, como anúncios de bateria fraca ou alterações no idioma, no entanto, aplicações também podem iniciar transmissões que podem ser recebidas por componentes da própria aplicação ou por processos totalmente diferentes.

### 3.4.4 Content Provider

Por padrão, o Android executa cada aplicativo em sua própria *sandbox* para que todos os dados que pertençam a um aplicativo sejam totalmente isolados de outros aplicativos no sistema (Gargenta, 2011). Os *content providers* (provedores de conteúdo) provém um nível de abstração para que qualquer dado guardado no dispositivo seja acessível por mais de uma aplicação (Ferreira, 2010).

Um provedor de conteúdo usa uma interface padrão na forma de um URI para atender solicitações de dados de outros aplicativos, que não necessitam saber qual provedor de conteúdo estão utilizando. O sistema operacional consulta quais aplicativos estão registrados como provedores de conteúdo para o URI dado, e envia a solicitação para a aplicação adequada (Rogers et al, 2009).

### 3.4.5 Intents

Três dos principais componentes de uma aplicação - atividades, serviços e receptores de transmissão - são ativados através de mensagens, chamadas de *Intents* (intenções). Mensagens *Intent* são funcionalidades para a ligação em tempo de execução de componentes da mesma aplicação ou de aplicações diferentes (Android developers, 2012).

*Intents* são mensagens assíncronas que são enviadas ao sistema operacional indicando a intenção de realizar alguma ação. Elas podem indicar o componente que deverá tratar a requisição, a ação que deve ser realizada, os dados sobre os quais a ação deve ser realizada, parâmetros extras e categorias que auxiliam o Android na escolha de qual componente é o mais adequado para receber a *Intent*.

## 3.5 Interface Gráfica

O desenvolvimento de interfaces gráficas no Android é baseado na utilização de elementos *View* e *ViewGroup* estruturados hierarquicamente como uma árvore de componentes. Os elementos *View* correspondem às folhas da árvore e representam os componentes gráficos da aplicação (*widgets*) como *TextView*, *Button*, *EditText*, *Checkbox*, etc. Ao passo que os elementos *ViewGroup* constituem

os ramos da árvore e representam *containers* de *widgets*, especificando regras para disposição dos mesmos na tela. Como exemplos de *ViewGroups* tem-se *LinearLayout*, *FrameLayout*, *AbsoluteLayout*, *RelativeLayout*, *TableLayout*, etc.

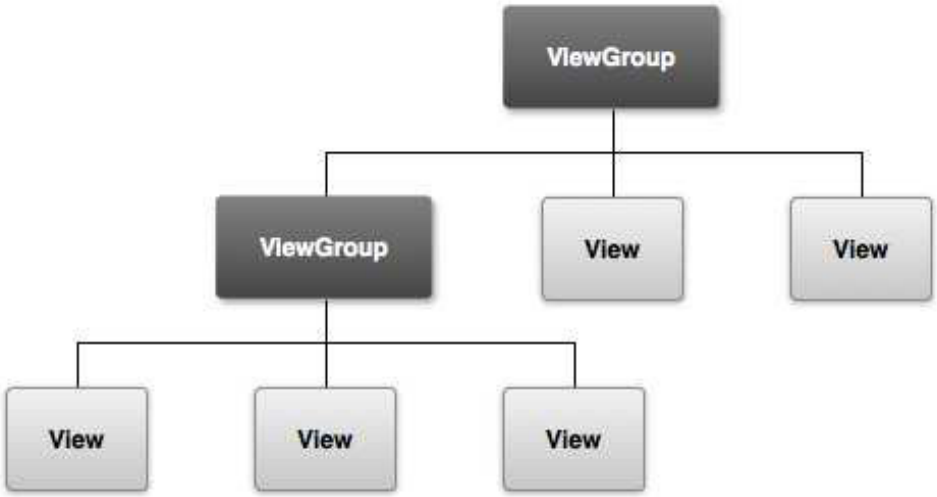


**Figura 3.3** – Hierarquia dos elementos *View* e *ViewGroup.* Fonte: [Android Developers, 2012]

A plataforma Android oferece duas maneiras distintas para a construção de interfaces gráficas. A abordagem declarativa envolve o uso de documentos XML para descrever como a interface deve se parecer, similar à construção de páginas da Web utilizando HTML (Gargenta, 2011). A abordagem programática, por outro lado, permite ao desenvolvedor criar interfaces utilizando código java, de maneira análoga à construção de telas usando *Java AWT* ou *Java Swing*.

Normalmente o desenvolvimento para andorid preconiza o uso das duas abordagens, sendo a primeira recomendada para a descrição de todas as características estáticas relacionadas ao layout da página e a segunda indicada para definir o comportamento e a interação com o usuário.

# 4 O GUIA ELETRÔNICO DE PROGRAMAÇÃO

Este capítulo descreve o desenvolvimento e a arquitetura de um serviço de guia eletrônico de programação baseado em um serviço web *RESTful* e um cliente Android, o qual é responsável por fornecer uma interface amigável ao usuário final.

Inicialmente são introduzidas duas ferramentas utilizadas neste trabalho, o formato XMLTV (*XML Television*) e o Sistema de Gerenciamento de Banco de Dados (SGBD) DB2 *PureXML*.

## 4.2 XMLTV

XMLTV é um conjunto de soluções para obtenção, manipulação e compartilhamento de informações de programação de TV de vários canais, através da definição de um formato padronizado baseado em XML. O formato XMLTV foi originalmente desenvolvido por Ed Avis e é atualmente mantido pelo XMLTV Project (XMLTV, 2012).

O elemento raiz dos documentos XMLTV é o elemento *<tv>*, que possui como atributos um conjunto de metadados relacionados ao documento como um todo. Dentro deste elemento, existem dois tipos básicos de elementos: *<channel>* e *<programme>*.

O elemento *<channel>* armazena informações sobre os canais, como identificador, descrição, *site* e logomarca. Este elemento pode aparecer inúmeras vezes em um mesmo documento, geralmente no início.

O elemento *<programme>* contém informações detalhadas a respeito de programas individuais. Como principais informações disponibilizadas estão: data e hora do início e fim do programa, canal, título, sinopse, categorias, informações a respeito do áudio e vídeo, elenco, classificação indicativa, etc. Um trecho de um documento XMLTV é apresentado na Figura 4.1.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <tv generator-info-name="Revista Eletronica - Unidade Lorenz Ltda" generator-info-url="http://www.revistaeletronica.com.br">
  - <channel id="GLB">
      <display-name lang="pt">Globo Brasília</display-name>
      <icon src="glb.gif" />
    </channel>
  - <programme start="20120507001000 -0300" stop="20120507045000 -0300" channel="GLB" program_id="0000077928">
      <title lang="pt">Sou Espião</title>
      <title lang="en">I Spy</title>
      <desc>O agente Alex Scott e o campeão mundial de boxe Kelly Robinson se unem para recuperar o sofisticado avião F-22,
        www.revistaeletronica.com.br</desc>
    - <credits>
        <director>Betty Thomas</director>
        <actor>Eddie Murphy</actor>
        <actor>Owen Wilson</actor>
        <actor>Famke Janssen</actor>
        <actor>Malcolm McDowell</actor>
        <actor>Gary Cole</actor>
        <actor>Viv Leacock</actor>
        <actor>Keith Dallas</actor>
        <actor>Tate Taylor</actor>
        <actor>Lynda Boyd</actor>
        <actor>Bill Mondy</actor>
        <actor>Sugar Ray Leonard</actor>
      </credits>
      <date>2002</date>
      <category lang="pt">Filme</category>
      <category lang="pt">Comédia</category>
      <country>EUA</country>
    - <video>
        <colour>yes</colour>
      </video>
    - <rating system="Advisory">
        <value>Programa impróprio para menores de 16 anos</value>
      </rating>
    - <star-rating>
        <value>3/5</value>
      </star-rating>
    </programme>
```

**Figura 4.1** – Trecho de documento XMLTV

No Brasil, a empresa Revista Eletrônica disponibiliza o XMLTV Revel, que é um pacote de arquivos no formato XMLTV com a programação dos principais canais de TV, 100% compatível com o *Windows Media Center*. Os arquivos são distribuídos gratuitamente através do endereço http://xmltv.revistaeletronica.com.br/wp/?page_id=36 (REVEL, 2012).

## 4.3 Banco de dados XML

Documentos XML, como comentado no Capítulo 2, são largamente utilizados para troca de informações entre sistemas devido às suas características como auto-descrição, interoperabilidade, flexibilidade e extensibilidade. Não obstante essa vocação para utilização como transportadora de dados, caso em que os documentos existem geralmente apenas na memória ou trafegando na rede, documentos XML podem constituir também boas soluções para armazenamento de informações.

A persistência de dados através de XML pode se basear na utilização de arquivos gravados em disco ou em coleções de documentos armazenados em um banco de dados. A última abordagem compreende os bancos de dados com suporte a XML, que mapeiam todo conteúdo XML para bancos de dados tradicionais, e os bancos de dados XML nativos, cujos campos são dados XML, e não referências para arquivos XML. Neste trabalho optou-se pela utilização de um banco de dados XML nativo, especificamente o DB2 *PureXML*.

### 4.3.1 DB2 PureXML

O *PureXML* é a nova tecnologia inserida no DB2 versão 9 permitindo que o banco de dados armazene documentos XML bem formados em colunas do tipo de dados XML, mantendo sua forma hierárquica. Deste modo não é mais necessário mapear documentos XML para o banco de dados ou armazená-los em objetos grandes.

Uma vez que os documentos XML estão armazenados no banco de dados em estruturas específicas para os mesmos, eles podem ser manipulados e consultados com muito mais eficiência. O DB2 *pureXML* oferece várias abordagens para a realização de consultas em seus bancos de dados, cada uma delas com uma série de funções específicas que facilitam a recuperação de informações.

### 4.3.2 Consultas XML

DB2 oferece três alternativas para a realização de consultas XML: XPATH (*XML Path Language*), XQuery (*XML QUERY*) e XML/SQL.

Em XPATH um documento XML é visto como uma árvore na qual cada parte do documento é representada por um nó. O resultado é uma hierarquia de nós que representam os elementos em uma estrutura pesquisável (Paz, 2009). Para localizar e extrair uma porção do documento XML, XPATH percorre a árvore retornando as subarvores que satisfazem os predicados ou filtros especificados na consulta.

Consultas envolvendo dados XML também podem ser feitas através de uma linguagem chamada XQuery. Ela é uma linguagem de consultas recomendada pelo W3C usada para encontrar e extrair elementos e atributos de documentos XML, para

tanto utiliza expressões de caminho para navegação pela estrutura hierárquica dos mesmos.

As consultas XQuery podem ser realizadas através de expressões de caminho e expressões FLWOR (*for, let, where, order by, return*). Expressões de caminho utilizam um subconjunto da linguagem XPATH, enquanto a expressão FLWOR, uma das expressões mais poderosos e comumente usadas na linguagem XQuery, é comparável à instrução SELECT-FROM-WHERE na linguagem SQL (*Structured Query Language*).

Por se tratar de um banco de dados híbrido, as instruções SQL convencionais são válidas para recuperar tanto os dados XML quanto os dados das tabelas relacionais (Paz, 2009 apud POLETTO, 2004). No entanto quando a intenção é selecionar porções de um documento XML ou filtrar registros baseando-se no valor de elementos internos ao XML é necessária a utilização de XML/SQL.

XML/SQL é uma versão estendida do SQL padrão que, através das funções *Xmlquery, Xmltable e Xmlexists*, permite a realização de consultas avaliando internamente o documento XML armazenado.

Embora utilizem abordagens e tecnologias diferentes, estes métodos de consulta podem ser combinados para atender as necessidades do desenvolvedor.

## 4.4 Requisitos do Sistema

Este tópico expõe um conjunto de requisitos funcionais e não funcionais nos quais o desenvolvimento do guia eletrônico de programação está pautado.

O principal requisito não funcional preconiza a adoção de uma abordagem de sistemas distribuídos baseada em serviços web que atendam às especificações do estilo arquitetural REST. Adicionalmente, vale destacar:

- Aplicação acessível através da internet;
- Interface com usuário acessível a partir de um dispositivo Android;
- Utilização de um banco de dados XML nativo;
- Interface simples e intuitiva;

Os principais requisitos funcionais podem ser visualizados através do diagrama de casos de uso ilustrado na Figura 4.2.

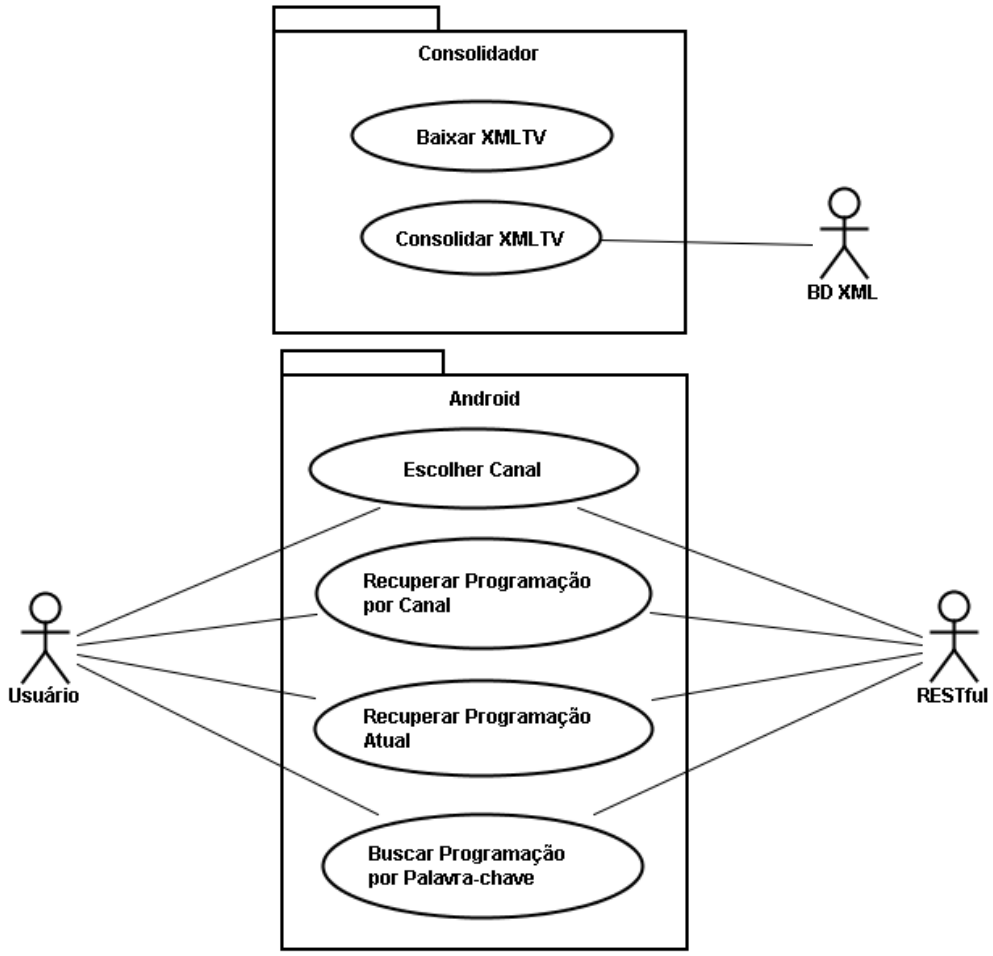


**Figura 4.2** – Diagrama de casos de uso

## 4.5 Arquitetura

A arquitetura do guia eletrônico de programação é composta por três componentes que colaboram para satisfazer o conjunto de requisitos descritos na seção anterior.

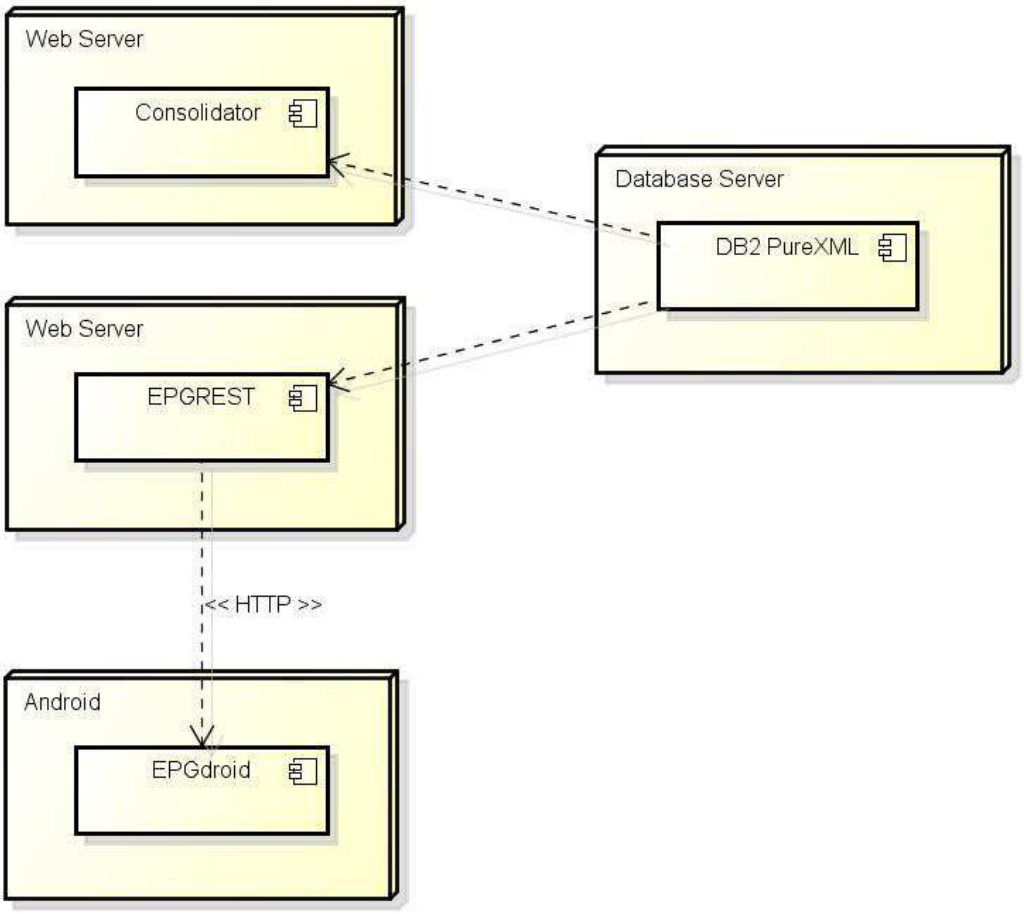


**Figura 4.3** – Arquitetura Geral

### 4.5.1 Componente Consolidator

Normalmente, as informações sobre programas e serviços são enviadas, por radiodifusão ou sob demanda, multiplexadas com outros tipos de dados como, por exemplo, áudio e vídeo (Maia, 2011). No entanto, quando se deseja obter essas informações fora do contexto das televisões e decodificadores é necessária a utilização de fontes de dados alternativas, geralmente disponibilizadas na Web. Para este fim, no Brasil, destacam-se como provedores de informações de programação de TV a sapo.tv e a Revista Eletrônica.

A Revista Eletrônica disponibiliza essas informações através do download de arquivos no formato XMLTV. O componente consolidator tem por objetivo transformar estes arquivos em informação utilizável pelo serviço *RESTful*, seguindo o fluxograma da Figura 4.4.

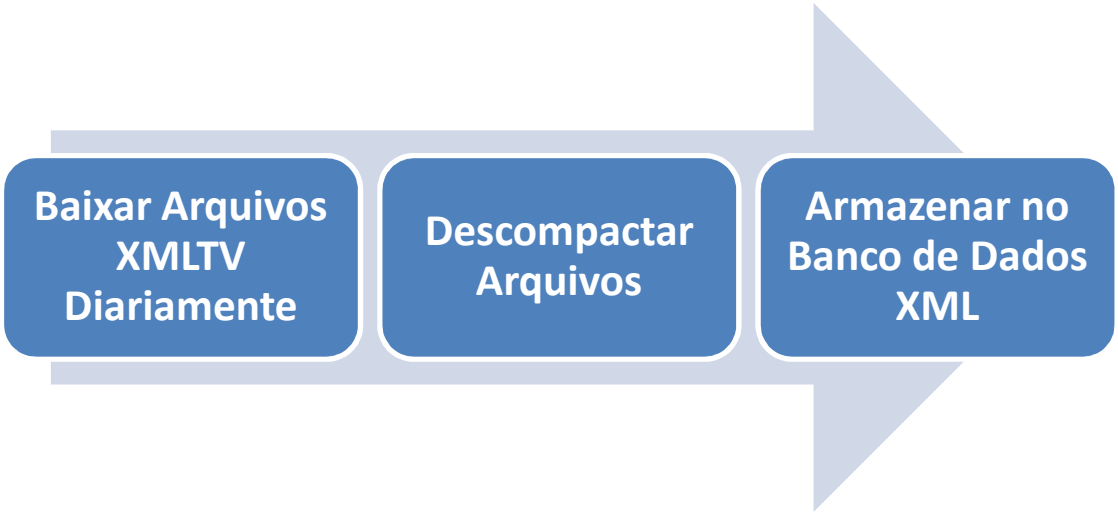


**Figura 4.4** – Fluxograma do componente Consolidator

O primeiro requisito para a obtenção dos dados é a automatização dos downloads dos arquivos disponibilizados pelo site. Para tanto, este componente dispõe das classes *Scheduler* e *FileDownloader*, sendo a primeira responsável pelo agendamento do horário do download e a segunda pela transferência dos arquivos do site para uma área acessível ao módulo. A classe *Unzip*, por sua vez, descompacta o arquivo descarregado gerando um arquivo XMLTV para cada canal.

Para consolidar as informações e permitir o acesso às mesmas por outros componentes, o Consolidator as armazena em uma base de dados XML gerenciada pelo SGBD DB2. As classes responsáveis pela conexão e persistência dos documentos são, respectivamente, a *DB2Connection* e a *XMLPersistence*.

A base de dados modelada para armazenar as informações de programação é simples. Isso se deve ao fato de que todas os dados relativos à programação de TV propriamente dita são armazenados em um campo do tipo XML nativo ao DB2 *PureXML*.

Conforme visto na Figura 4.5, a tabela *Programme* armazena a programação de cada canal por data, enquanto a tabela *Channel* armazena os dados específicos de cada canal. A tabela *XMLTemp* é utilizada internamente para preparação dos dados.

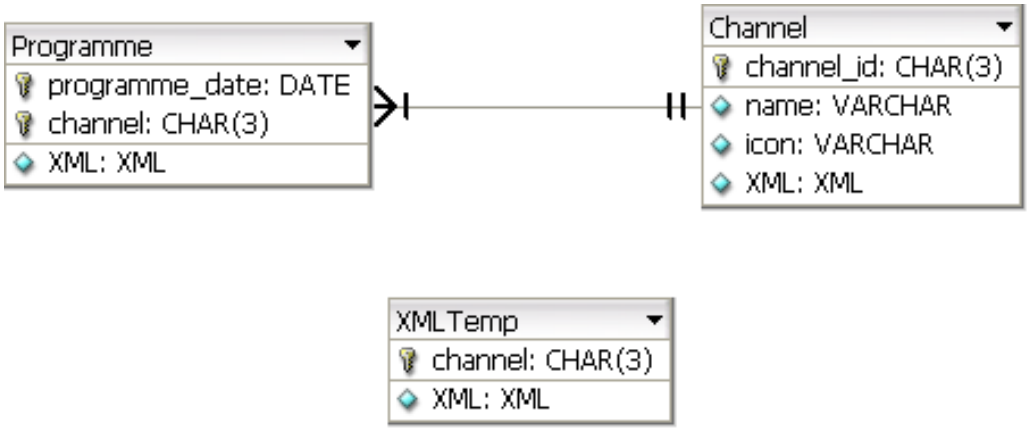


**Figura 4.5** – Tabelas do banco de dados DB2

### 4.5.2 Componente EPGREST

O componente EPGREST tem como função oferecer uma interface uniforme para acesso a informaçoes de programação de TV brasileiras, retornando seus resultados em um formato portátil, baseado em XML e facilmente interpretado por praticamente todas as linguagens de programação modernas.

EPGREST foi desenvolvido a partir dos princípios arquiteturais REST descritos no Capítulo 2, utilizando serviços web para implementar funcionalidades bem definidas e independentes. Devido a estas características, os serviços podem ser consumidos por clientes em diferentes aplicações e processos de negócios, utilizando-se variadas plataformas e tecnologias.

#### 4.5.2.1 Desenvolvimento

Para o desenvolvimento deste componente foi utilizada a tecnologia Java, aliada à especificação JAX-RS (*Java API for RESTful Web Services*) e sua implementação *Jersey*. Como repositório de dados, foi utilizada uma base de dados gerenciada pelo SGBD DB2 9 Pure XML.

Para que o *Jersey* seja executado é necessário um *Web Container*. O contêiner web fornece um ambiente de execução que gerencia o ciclo de vida dos componentes de uma aplicação Web. O EPGREST utiliza o Glassfish com servidor de aplicação Java EE, o qual possui um contêiner web.

JAX-RS é um framework baseado em anotações usado para simplificar o desenvolvimento de serviços web *RESTful*. As anotações vinculam URIs específicas e operações HTTP a métodos individuais de uma classe Java, podendo prover a definição de parâmetros que são extraidos a partir da solicitação HTTP, e especificar o formato de dados para serialização e desserialização de objetos Java. A especificação JAX-RS define também mapeamentos de exceções lançadas pelo aplicativo para códigos de resposta HTTP, além de oferecer facilidades para a negociação de conteúdo HTTP (Burke, 2010).

O Projeto de *Jersey* é uma comunidade de código aberto que desenvolve e mantém uma implementação da referência JSR-311: JAX-RS. *Jersey* implementa suporte para as anotações definidas no JSR-311, tornando mais fácil para os desenvolvedores construir serviços web *RESTful* com Java e a JVM Java (Jersey, 2012).

Dentre as anotações JAX-RS mais importantes pode-se citar:

- *@Path*: Define uma URI para o recurso;
- *@Produces*, *@Consumes*: Define em quais tipos MIME os recursos deverão ser produzidos e consumidos;
- *@GET*, *@POST*, *@DELETE*, *@PUT*: Identifica a qual método HTTP corresponde o método Java;
- *@PathParam*: Usado para injetar valores da URL em algum parâmetro do método Java.

Para exemplificar a utilização das anotações JAX-RS, a Figura 4.6 exibe um trecho de código retirado do EPGREST, responsável por disponibilizar a programação de TV resultante da realização de uma pesquisa baseada no parametro *search*.

```java
/**
 * Retorna a representação de uma instancia do recurso
 * @return uma instancia java.lang.String no formato text/xml
 */
@Path("search/{search}")
@GET
@Produces("text/xml")
public String getSearch(@PathParam("search") String search) {
    try {
        return  EPGRecovery.getSearch(search);
    } catch (Exception e) {
        return "<exception>" + e + "</exception>";
    }
}
```

**Figura 4.6** – Exemplo de anotações JAX-RS

#### 4.5.2.2 Serviços

Para a aplicação do guia eletrônico de programação foram definidos dois serviços básicos, *channel* e EPG, ambos possuindo apenas métodos GET.

O serviço *channel* possui somente um recurso, que disponibiliza uma lista com todos os canais que possuem suas programações registradas. O serviço channel é identificado pela URL /*channel*.

O serviço EPG oferece um conjunto de subrecursos, representando documentos XMLTV, relacionados à programação de TV propriamente dita. O primeiro subrecurso recebe como parâmetro um código de canal e uma data, disponibilizando a programação inteira do canal na data especificada. O segundo subrecurso oferece um documento XMLTV contendo os programas, de todos os canais, que estão sendo transmitidos na data e hora atuais. Um terceiro subrecurso realiza uma busca na programação baseando-se em uma string passada como parâmetro. Outros subrecursos, embora não utilizados pelo cliente EPGdroid, estão disponíveis no serviço, entre os quais: a programação de um programa específico em um intervalo de datas,  a programação de um dado canal em um intervalo de datas, todos os programas de uma determinada categoria em um intervalo de datas. As URLs para acesso aos recursos são, respectivamente:

- /EPG/{canal}/{data}
- /EPG/now
- /EPG/search/{palavra-chave}

- /EPG/program/{id_programa}/{data1}/{data2}
- /EPG/{canal}/{data1}/{data2}
- /EPG/category/{categoria}/{data1}/{data2}

Os elementos entre chaves são substituídos pelos parâmetros a serem passados pela URL.

Para a implementação dos serviços foram utilizadas basicamente as classes *ChannelResource*, *EPGResource*, *ChannelRecovery* e *EPGRecovery*. Sendo que as duas primeiras são responsáveis pela definição dos serviços através das anotações JAX-RS e as duas últimas responsáveis pela recuperação dos dados solicitados no banco de dados XML.

Para melhor compreensão da dinâmica dos serviços, a Figura 4.7 exibe um diagrama de sequência para o subrecurso que retorna a programação de um canal em uma data específica.

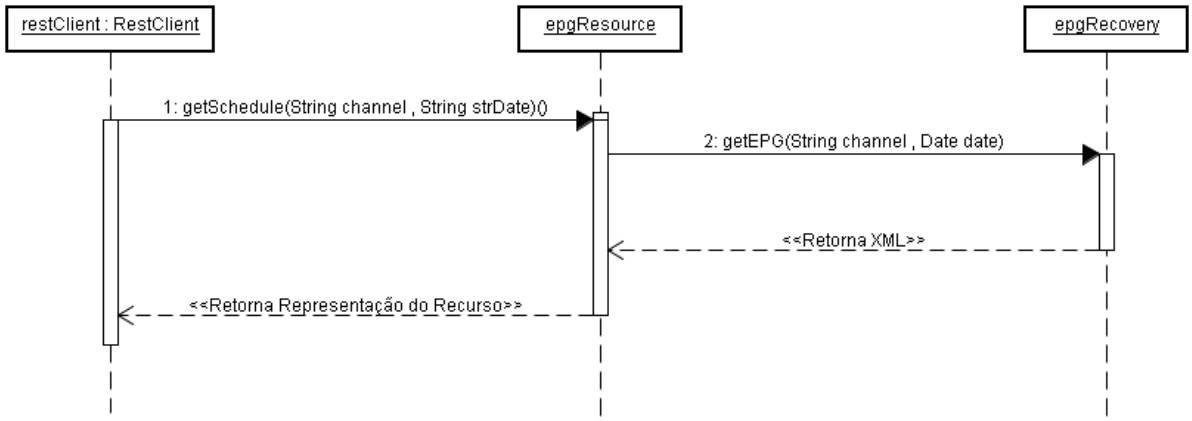


**Figura 4.7** – Diagrama de sequência

A classes *ChannelRecovery* e *EPGRecovery* utilizam os mecanismos de consulta disponibilizados pelo DB2 9 *PureXML*. Uma consulta XQUERY usada para retornar a programação, no horário corrente, de todos os canais é exemplificada na Figura 4.8.

```
xquery
for $c in db2-fn:xmlcolumn("EPG.PROGRAMME.XML")/tv/programme
for $w in db2-fn:xmlcolumn("EPG.CHANNEL.XML")/channel
let $anoStart := substring($c/@start , 1,4)
let $mesStart := substring($c/@start , 5,2)
let $diaStart := substring($c/@start , 7,2)
let $horaStart := substring($c/@start , 9,2)
let $minutoStart := substring($c/@start , 11,2)
let $segundoStart := substring($c/@start , 13,2)
let $dataStart := fn:adjust-dateTime-to-timezone(xs:dateTime(
fn:concat($anoStart,'-',$mesStart,'-',$diaStart,'T',$horaStart,':',$minutoStart,':',$segundoStart)), xdt:dayTimeDuration("-PT3H"))
let $anoStop := substring($c/@stop , 1,4)
let $mesStop := substring($c/@stop , 5,2)
let $diaStop := substring($c/@stop , 7,2)
let $horaStop := substring($c/@stop , 9,2)
let $minutoStop := substring($c/@stop , 11,2)
let $segundoStop := substring($c/@stop , 13,2)
let $dataStop := fn:adjust-dateTime-to-timezone(xs:dateTime(
fn:concat($anoStop,'-',$mesStop,'-',$diaStop,'T',$horaStop,':',$minutoStop,':',$segundoStop)), xdt:dayTimeDuration("-PT3H"))
let $current := fn:adjust-dateTime-to-timezone(fn:current-dateTime())
where $c/@channel = $w/@id and $dataStart <= $current and $dataStop >= $current
return <tv>{$w}{$c}</tv>
```

**Figura 4.8** – Exemplo de consulta XQuery

### 4.5.3 Componente EPGdroid

Visto que o componente EPGREST foi desenvolvido como um serviço web *RESTful*, uma infinidade de aplicações clientes, dos mais variados tipos, podem ser desenvolvidas a fim de consumir os recursos disponibilizados. Neste trabalho optou-se pela construção de um aplicativo Android, denominado EPGdroid.

#### 4.5.3.1 Aplicação

O EPGdroid fornece um conjunto de interfaces para que o usuário possa ter acesso aos serviços do componete EPGREST. Estas interfaces são construidas usando os elementos básicos da plataforma Android descritos no Capítulo 3, entre outros. A Figura 4.9, exibe um diagrama de classes que relaciona estes componentes.

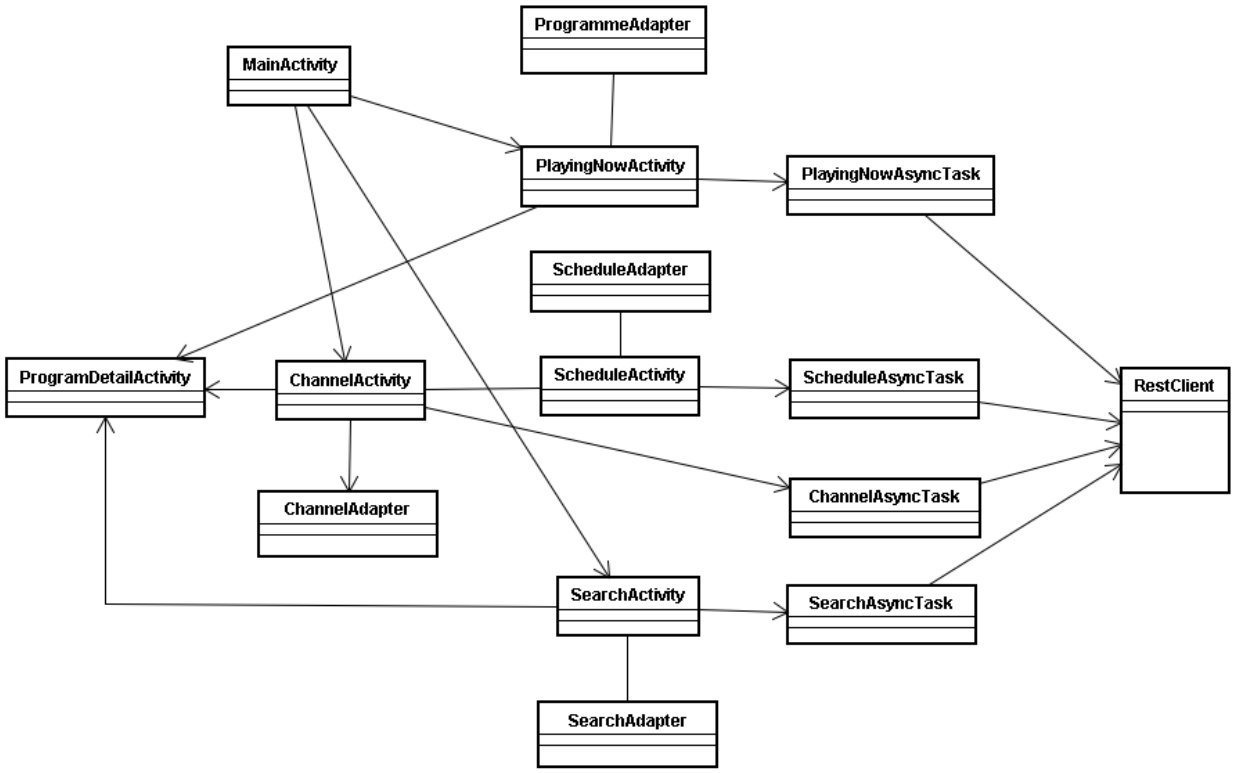


**Figura 4.9** – Diagrama de classes da aplicação Android

A classe *MainActivity* é responsável pela navegação e acesso às funcionalidades providas pelo aplicativo. Ela utiliza o recurso *TabHost* da API Android para organizar o conteúdo da aplicação de uma forma intuitiva e prática, através do uso de abas. A partir de *MainActivity* três novas atividades podem ser acessadas: *ChannelActivity*, que exibe a lista dos canais disponíveis, *PlayingNowActivity*, que exibe a programação de cada canal no momento atual, e *SearchActivity*, que permite ao usuário efetuar uma busca na programação.

Por se tratarem de listas, para cada uma das três atividades comentadas acima existe uma classe *Adapter* associada. Um objeto *adapter* atua como uma ponte entre uma lista e seus elementos, sendo responsável pela exibição e acesso aos itens de dados. Neste trabalho foram criadas classes que herdam de *BaseAdapter*, de modo a exibir listas compostas por objetos complexos, como os programas de um EPG.

Outros elementos importantes do diagrama da Figura 4.9 são as classes que herdam de *AsyncTask*: *PlayingNowAsyncTask*, *ScheduleAsyncTask*, *ChannelAsyncTask* e *SearchAsyncTask*.

O *AsyncTask* é um mecanismo apropriado para dar feedback ao usuário durante a execução de alguma operação que demande tempo. Como as requisições ao serviço web são feitas através da Internet, é aconselhavél a criação de uma nova linha de processamento diferente da atividade. Essa preocupação permite que a atividade execute as tarefas relacionadas à exibição da interface gráfica sem ser prejudicada pelo processamento da requisição HTTP.

A escolha de *AsyncTask* em detrimento de Services se deve ao fato de que *AsyncTasks* podem atualizar a tela de interação com o usuário sem a necessidade de artifícios como *Handlers*. Estes modificam a interface do usuário através de mensagens trocadas entre a *Thread* ou o *Service* que executa o processamento e a activity responsável pela exibição da tela.

Por fim, todo processamento relacionado ao serviço web foi encapsulado na classe *RestClient*. Esta classe é responsável por preparar e enviar as requisições HTTP e receber os resultados das solicitações. A negociação do conteúdo da resposta HTTP é feita através do campo accept enviado dentro do cabeçalho da requisição, neste caso, o formato escolhido foi text/xml.

Para que os dados pudessem ser utilizados de forma livre pelo EPGdroid, foi aplicado um processo de desserialização do conteúdo XML para objetos Java em memória. Este processo foi automatizado pela biblioteca *SimpleXML*.

*SimpleXML* é um framework de alto desempenho para configuração e serialização de XML para Java. Seu objetivo é permitir o rápido desenvolvimento de sistemas que necessitem realizar configuração e comunicação de dados XML. A biblioteca dispõe de recursos para serialização e desserialização de objetos completos, incluindo as referências para objetos aninhados (Jersey, 2012).

Neste contexto, foram desenvolvidas classes a partir dos dados disponíveis no formato XMLTV, como programas, categorias, canais, títulos, etc. Estas classes foram anotadas com as anotações *SimpleXML* a fim de automatizar o processo de desserialização. A Figura 4.10 exibe um diagrama de classes que expressa o relacionamento entre elas.

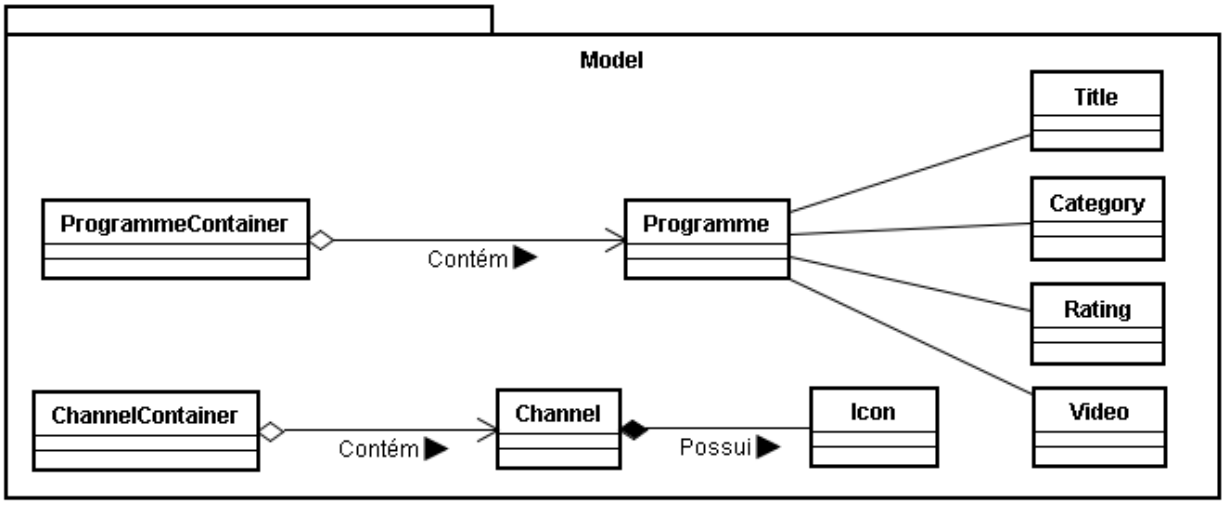


**Figura 4.10** – Diagrama de classes das entidades

#### 4.5.3.2 Interface com Usuário

A primeira tela da aplicação exibe a lista dos programas de TV que estão sendo exibidos no momento em todos os canais. A partir dessa tela o usuário pode navegar na programação atual de todos os canais, podendo escolher um programa qualquer para visualizar os detalhes do mesmo, conforme a Figura 4.11. Ainda nesta tela, existem três abas na parte inferior que permitem a navegação para outras funcionalidades do aplicativo. As opções são, além da programação atual, a programação por canal e data e uma tela para a busca de programas.

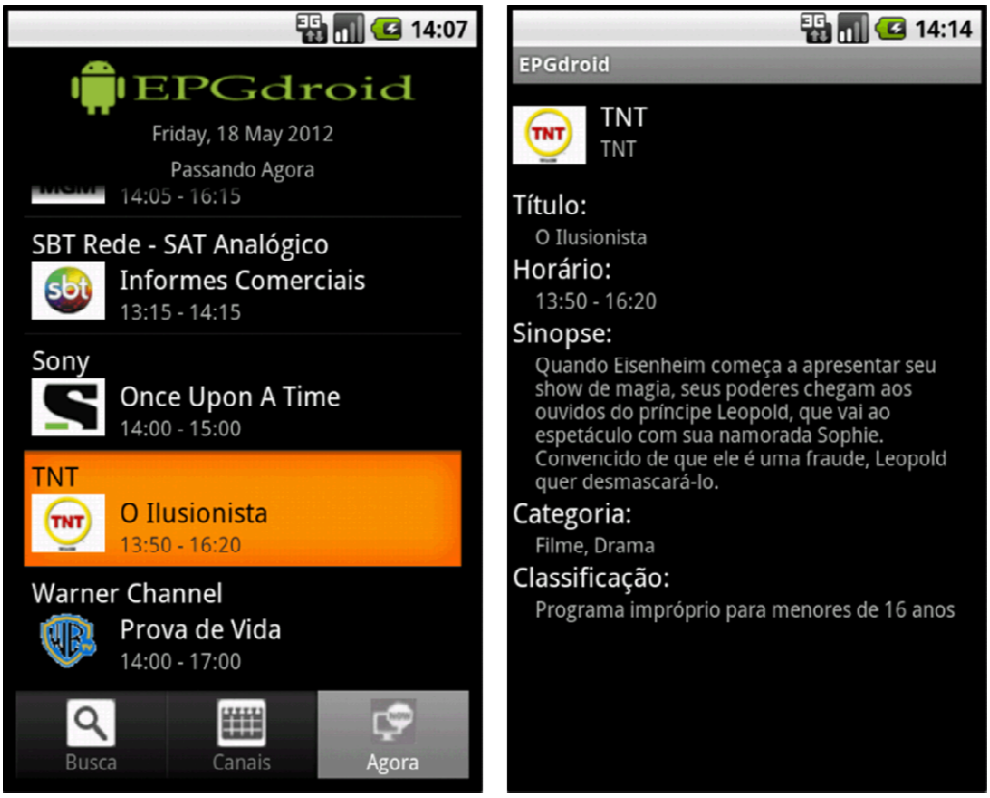


**Figura 4.11** – (A) Lista de programas passando agora e (B) Detalhes de um programa escolhido pelo usuário

Para acompanhar a programação de um canal específico, o usuário deve selecionar a aba Canais, o que exibirá uma tela contendo a lista de todos os canais disponíveis. Através desta lista o usuário pode escolher o canal de sua preferência. Uma vez selecionado o canal, o aplicativo mostra toda a programação do mesmo para a data atual, com opção de navegação para datas anteriores e posteriores. A programação é exibida como uma lista de programas ordenados por hora de início, sendo que o usuário pode escolher um programa específico para visualizar suas informações detalhadas. A Figura 4.12 e 4.13 exibem as telas envolvidas nessa funcionalidade.

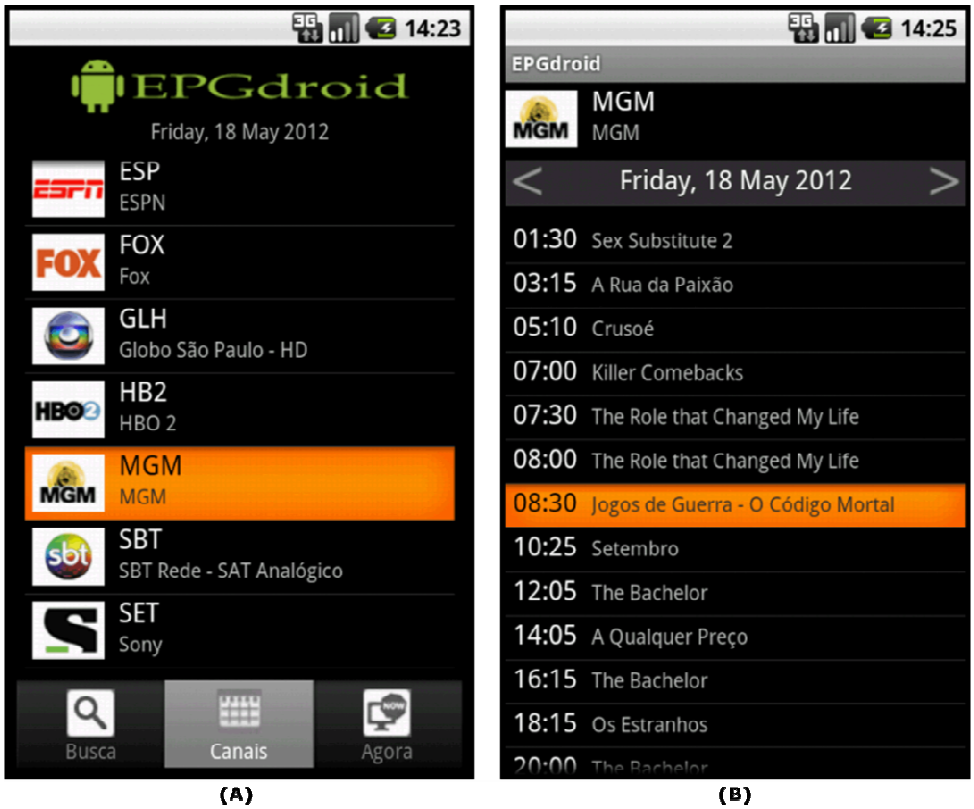


**Figura 4.12** – (A) Lista de canais e (B) Programação de um canal específico

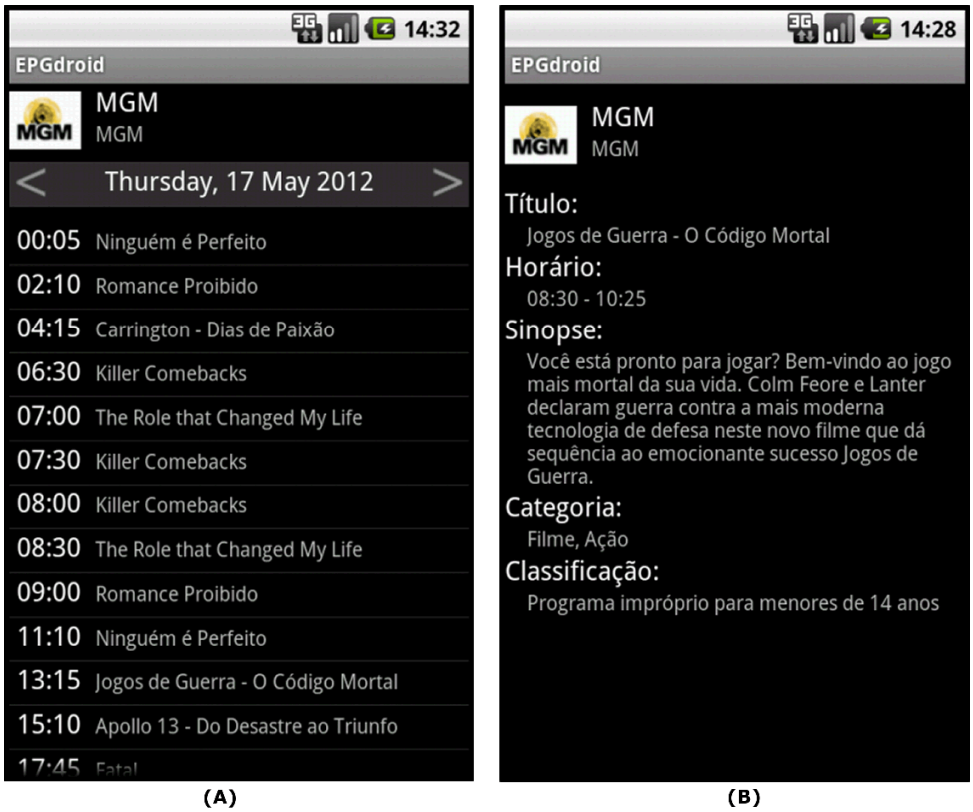


**Figura 4.13** – (A) Programação de um canal específico e (B) Detalhes de um programa escolhido pelo usuário

A última aba permite ao usuário ter acesso à tela de busca. Esta tela solicita a entrada de uma ou mais palavras que serão usadas para pesquisar programas dentro da programação de todos os canais. Após a digitação dos termos para a pesquisa, o usuário deve clicar no botão em forma de seta, localizado ao lado do campo de texto. O sistema exibe então uma lista com todos os progra mas que corresponderam ao critério de busca, permitindo ao usuário selecionar um deles para visualizar os detalhes (Figura 4.14).

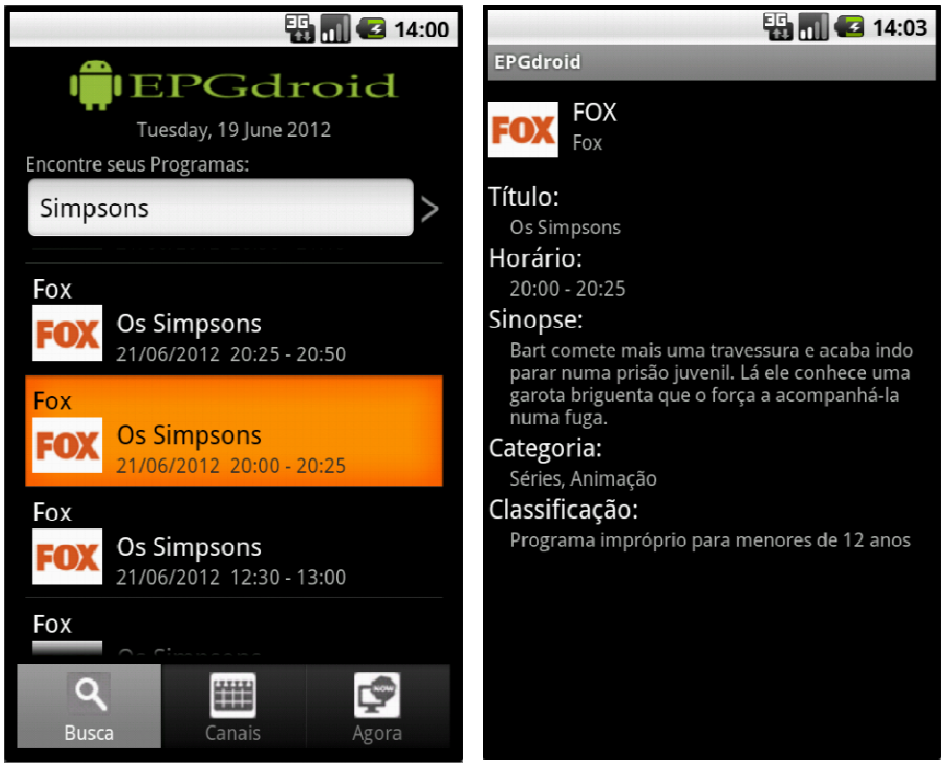


**Figura 4.14** – (A) Programas retornados pela pesquisa e (B) Detalhes de um programa escolhido pelo usuário

# 5 CONCLUSÃO

Neste trabalho foi apresentada a utilização do estilo arquitetural REST para o desenvolvimento de um guia eletrônico de programação capaz de fornecer as programações atualizadas das principais emissoras de TV brasileiras, de modo que as mesmas pudessem ser facilmente consumidas por aplicações heterogêneas.

Inicialmente documentou-se uma revisão bibliográfica a respeito da tecnologia de serviços web como solução para a interoperabilidade entre sistemas computacionais. Também foram incluídos tópicos relacionados às abordagens orientadas a serviços e a recursos que norteiam o desenvolvimento de aplicações baseadas em serviços web. Neste contexto, foi introduzido o estilo arquitetural REST, sobre o qual está fundamentado este trabalho.

Em seguida, foram abordados os tópicos essenciais relativos à plataforma Android, a qual foi utilizada para a implementação do cliente *RESTful* responsável por consumir os serviços e apresentar as informações de forma clara e interativa para os usuários finais.

Finalmente, foram descritas as características do guia eletrônico de programação desenvolvido, detalhando os componentes de sua arquitetura. O primeiro componente comentado foi o *consolidator*, responsável pela agregação das informações de programação disponíveis na Internet em um repositório padronizado e baseado em XML. O segundo componente discutido foi o EPGREST, que é constituído por recursos disponibilizados segundo as restrições arquiteturais REST. Por último, foi detalhado o desenvolvimento de um aplicativo Android responsável pela interface gráfica com o usuário, o componente EPGdroid.

Através deste trabalho pôde-se exemplificar como sistemas baseados em REST podem promover a dissociação entre as interfaces dos serviços e suas implementações, visto que os clientes precisam apenas descobrir essas interfaces e consumir os resultados disponibilizados sob o padrão XML, sem a necessidade de conhecer detalhes internos ao serviço.

O componente *consolidator* foi bem sucedido em seu papel de organizar as informações coletadas em provedores de conteúdo, preparando-as para serem utilizadas e disseminadas pelo EPGREST.

O módulo EPGREST logrou êxito em oferecer uma interface uniforme para acesso a informaçoes de programação de TV brasileiras, retornando seus resultados em um formato portável, baseado em XML e facilmente interpretado por praticamente todas as linguagens de programação modernas.

A mobilidade foi garantida através do EPGdroid, visto que o componente foi desenvolvido sobre a plataforma Android, uma das mais abrangentes e bem sucedidas plataformas para ambientes móveis da atualidade.

Dentre algumas melhorias para o aprimoramento do guia eletrônico de programação descrito neste documento e a título de trabalhos futuros pode-se citar:

- Efetuar configurações avançadas no banco de dados como meio para aumentar o desempenho das consultas;
- Sistema de recomendação de programação baseado no perfil do usuário;
- Estudos de usabilidade da interface do aplicativo desenvolvido para Android a fim de garantir uma experiência mais rica e prazerosa ao usuário.
- Desenvolver clientes em outras linguagens e plataformas para atestar o princípio da interoperabilidade provido por sistemas *RESTful*.

# REFERÊNCIAS


ANDROID DEVELOPERS. 2012. Disponível em: <http://developer.android.com/guide/topics/ui/index.html>. Acesso em: 01 maio 2012.

AUSTIN, D.; BARBIR, A.; FERRIS, C.; GANG, S. **W3c, Web Services Architecture Requirements.** 2002. Disponível em: <http://www.w3.org/TR/2002/WD-wsa-reqs-20020819#IDAIO2IB>. Acesso em: 30 abr. 2012.

BENZ, B.; DURANT, J. R. **XML Programming Bible.** 1 ed. Indianapolis: Wiley Publishing, 2003. 987 p.

BURKE, B. **RESTful Java with JAX-RS**. 1 ed. Sebastopol: O'Reilly, 2010. 311 p.

DALVIK VIRTUAL MACHINE. 2008. Disponível em: <http://www.dalvikvm.com>. Acesso em 08 maio 2012.

ENGLANDER, R. **Java and SOAP: building web services in java**. 1 ed. USA: O'Reilly. 2002. 276p.

FERREIRA, G. D. **Um middleware declarativo na plataforma AndroidTM para o Sistema Brasileiro de Televisão Digital (SBTVD).** Vitória, ES, 2010. 78 p. Dissertação (Mestrado em Informática). Universidade Federal do Espírito Santo.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. Irvine, California, 2000. 162 p. Dissertation. (Degree of Doctor of Philosophy in information and computer science). University of California.

FRANÇA, T. C. de; PIRES, P. F.; PIRMEZ, L.; DELICATO, F. C.; Farias. Web das Coisas: Conectando Dispositivos Físicos ao Mundo Digital. *In*: **XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos – SBRC**, 2011. Disponível em: <http://sbrc2011.facom.ufms.br/files/anais/files/mc/mc3.pdf>. Acesso em: 30 abr. 2012.

GARGENTA, M. Main_Building_Blocks. *In*: Gagenta, Marko. **Learning Android**. 1 ed. Sebastopol: O'Reilly. 2010. 268 p.

JERSEY PROJECT. 2012. Disponível em: <http://jersey.java.net/>. Acesso em: 12 maio 2012.

KRAFZIG, D.; BANKE K.; SLAMA, D. **Enterprise SOA: Service oriented architecture best pratices**. 1 ed. Indianápolis: Prentice Hall PTR, 2008. 408p.

MACIEL, D. R. **Implementação de um Disco Virtual Seguro Baseado em Web Services**. São Luis, Maranhão, 2007. 101 p. Monografia (Bacharelado em Ciência da Computação). Universidade Federal do Maranhão.

MACK, R. S. **Sistema de recomendação baseado na localização e perfil utilizando a plataforma android.** Porto Alegre, Rio Grande do Sul, 2010. 56 p.

Monografia (Bacharel em Ciências da Computação). Universidade Federal do Rio Grande do Sul.

MAIA, P. P. C.; LEITE, J.; BATISTA, T. **MyPersonal-EPG: Um EPG Personalizável e com Suporte à Recomendações**. 2010. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/webmedia/2010/07_webmi_c.pdf>. Acesso em 08 maio 2012.

MARTINS, J. M.; POLLETO, A. S. R. de S. **DB2 pureXML: Entendendo e Aplicando**. Disponível em: dcon.com.br/jd.comment/juliano/Artigo_Juliano_v9.doc. Acesso em: 08 maio 2012.

MEIER, R. **Professional Android 2 Application Development (Wrox Programmer to programmer).** 2 ed. Indianapolis: Wiley Publishing. 2010. 580 p.

NEWCOMER, E.; LOMOW, G. **Understanding Web Services- XML, WSDL, SOAP and UDDI**. 1 ed. Indianapolis: Addison-Wesley Professional. 2002. 215 p.

NICOLA, M.; KUMAR-CHATTERJEE, P. **DB2 pureXML Cookbook: Master the Power of the IBM Hybrid Data Server**. Boston: Pearson Education. 2009. 787 p.

OLIVEIRA, F. N. B. de. **Aplicação Adaptativa de Guia Eletrônico utilizando o Ginga-NCL.** Rio de Janeiro, RJ, 2010. 104 p. Dissertação (Mestrado em Informática). Pontifícia Universidade Católica Do Rio De Janeiro - Puc-Rio**.**

OREILLY. 2012. Disponível em: <http://ofps.oreilly.com/titles/9781449390501/Main_Building_Blocks.html>. Acesso em: 08 jun. 2012.

PAPAZOGLOU, M. P.; GEORGAKOPOULOS, D. Service-Oriented Computing**. Communications of the ACM**, local, v. 46, n. 10, p. 25–28, out. 2003.

PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. RESTful Web Services vs. “Big” Web Services: Making the Right Architectural Decision. *In*: **Proceeding of the 17th international conference on World Wide Web**, **Beijing, China,** 2008. Disponível em: <http://portal.acm.org/citation.cfm?id=1367606>. Acesso em: 02 dez. 2010.

PAZ, I. M. **Um Estudo da Utilização do Padrão XML como Armazenamento de Dados nos Principais Bancos de Dados Comerciais**. Novo Hamburgo, Rio Grande do Sul, 2009. 83 p. Trabalho de Conclusão de Curso (Graduação em Ciências da Computação). Centro Universitário Feevale.

REVEL. 2012. Revista Eletrônica. Disponível em: <http://xmltv.revistaeletronica.com.br/wp>. Acesso em: 08 maio 2012.

RICHARDSON, L.; RUBY, S. **Restful web services: web services for the real world**. 1 ed. USA: O’Reilly. 2007. 440 p.

ROGERS, R.; LOMBARDO, J.; MEDNIEKS, Z.; MEIKE, B. **Android Application Development**. 1 ed. Sebastopol: O’Reilly. 2009. 336 p.

SILVESTRE, E.; POLÔNIA, P. V. **Uma Aplicação da Arquitetura Orientada a Recursos**. Santa Catarina, 2008. 271 p. Trabalho de Conclusão de Curso (Graduação em Ciências da Computação) - Universidade Federal de Santa Catarina.

SOUSA, T. N. de. **Desenvolvimento de uma Rede P2P para a Plataforma Android**. São Luis, Maranhão, 2011. 49 p. Monografia (Bacharelado em Ciência da Computação). Universidade Federal do Maranhão.

TEIXEIRA, M. A. M. Notas de aulas da disciplina Arquitetura Orientada a Serviços. Disponível em <http://www.deinf.ufma.br/~mario/disciplinas.html>. Acesso em: 15 jun. 2012.

TIDWELL, D.; SNELL, J.; KULCHENKO, P. **Programming Web Services with SOAP.** 1 ed. USA: O'Reilly. 2001. 216p.

WYKE, R. A.; WATT, A. **XML Schema Essentials.** 1 ed. Indianapolis: Wiley Computer Publishing, 2002. 386 p.

XMLTV. 2012. Disponível em: <wiki.xmltv.org/index.php/Main_Page>. Acesso em: 08 maio 2012.