A arquitetura Universal Plug and Play

Rede Plug and Play

O Universal Plug and Play facilita a integração transparente de dispositivos de rede. Aprenda a usá-lo com o framework BRisa UPnP.

por Leandro Melo de Sales e Thiago Melo de Sales

Piotr Ciuchta – www.sxc.hu

A popularização dos dispositivos eletrônicos portáteis tais como PDAs, celulares e *Internet tablets* tem proporcionado o desenvolvimento de mecanismos para conectar e compartilhar informações [1]. O padrão de conectividade *Universal Plug and Play* (UPnP) [2] é considerado uma boa opção para prover serviços ubíquos por meio da utilização de dispositivos móveis, fazendo uso de tecnologias já consolidadas da Internet, tais como o HTTP, o SOAP e o XML.

O conceito original de *Plug and Play* (PnP) era focado na conectividade dinâmica de dispositivos ao computador. Um driver de dispositivo controlado pelo sistema operacional detecta o periférico e o torna disponível para o usuário através das chamadas de sistema. O padrão UPnP especifica uma abordagem bastante diferente, porém, muito semelhante: os dispositivos se conectam à rede e são detectados pelos sistemas com o uso de protocolos de rede, em lugar dos drivers de dispositivos. As chamadas ao sistema usadas no Plug and Play são substituidas por chamadas remotas de métodos com o uso de *web services* baseados no protocolo SOAP.

Uma rede UPnP é um conjunto de computadores interconectados, aparelhos de rede e dispositivos que utilizam o padrão UPnP para descobrir, anunciar e acessar serviços de rede. O objetivo é prover um *framework* para criar ferramentas que ofereçam suporte à comunicação entre dispositivos de rede. O UPnP alcança este objetivo por meio da definição e publicação de protocolos que controlam os dispositivos UPnP. Estes protocolos são considerados padrões na Internet [3], como HTTP, UDP e TCP. Como se pode esperar, o UPnP suporta conexões entre dispositivos como celulares e MP3 players, mas o padrão de conectividade UPnP também especifica perfis para conectar periféricos convencionais como impressoras e câmeras de segurança, assim como aparelhos domésticos como luzes, portas, sensores de temperatura, cortinas etc.

Um exemplo de perfil UPnP é a especificação *UPnP A/V* (Áudio e Vídeo) [4], lançada em meados de 2002 e atualizada recentemente em 2008. O UPnP A/V foi especificado para suportar dispositivos UPnP que compartilham conteúdos multimídia e serviços, tais como a reprodução deste conteúdo. Este perfil define protocolos para descobrir, compartilhar e reproduzir os conteúdos multimídia disponíveis na rede, desde que o usuário os compartilhe.

Neste artigo, explicaremos como criar uma aplicação multimídia simples baseada no padrão UPnP. Para desenvolver esta solução, usaremos o framework *BRisa UPnP* [5]. Trata-se de uma plataforma de código aberto desenvolvida em Python que permite o desenvolvimento de aplicações UPnP personalizadas.

UPnP A/V

O UPnP para áudio e vídeo é uma especificação dentro do padrão do UPnP supervisionado pela Digital Living Network Alliance (DLNA) [6]. O DLNA oferece suporte à interoperabilidade entre redes de computadores cabeadas e as redes sem-fio, dispositivos eletrônicos e dispositivos móveis que compartilham conteúdos multimídia e disponibilizam serviços de forma transparente (no ponto de vista de não ser obrigatório qualquer tipo de configuração no dispositivo que se conecta à rede).

A especificação UPnP para áudio e vídeo define um modelo de comunicação entre três dispositivos UPnP:

- *Media server* – um dispositivo na rede que compartilha itens multimídia, tais como áudio (`.mp3`, `.wma`, `.ogg`), vídeo (`.mpeg`, `.wmv`, `.ogg`), imagens (`.jpg`, `.gif`, `.png`) e até mesmo conteúdos multimídia disponíveis na Internet, como os vídeos do Youtube, as imagens do FlickR e as estações de rádios online, como a SHOUTcast;

- *Media renderer* – responsável por reproduzir todos os conteúdos multimídia disponibilizados pelos media servers. Ele provê serviços que permitem a dispositivos remotos enviar requisições para reproduzir, parar e controlar o volume de um conteúdo multimídia;
- *Control point multimídia* – geralmente é um dispositivo portátil, como um telefone celular ou um Internet tablet, que permite aos usuários navegarem no conteúdo de um servidor de mídia (*media server*) e controlar um renderizador de mídia (*media renderer*).

A **figura 1** ilustra o processo de descoberta e acesso aos serviços UPnP de áudio e vídeo. Quando o ponto de controle é conectado à rede, ele transmite uma mensagem *multicast* à rede para procurar os servidores e os renderizadores de mídia (passo 1). Todos os dispositivos UPnP recebem a mensagem multicast enviada pelo ponto de controle e cada um destes responde a essa requisição (passo 2). Neste momento, o ponto de controle passa a ter conhecimento de todos os dispositivos UPnP na rede. Caso novos dispositivos entrem na rede, estes notificarão o ponto de controle assim que se conectarem à rede.

Através do ponto de controle, o usuário navega pelo conteúdo de um servidor de mídia e seleciona um conteúdo multimídia a ser reproduzido (passo 3). Após selecionar este item, o usuário escolhe em qual renderizador de mídia deseja reproduzir o item selecionado (passo 4). Em seguida, o usuário requisita a reprodução da mídia clicando no botão reproduzir disponível no ponto de controle. Neste momento, o renderizador de mídia começa a reproduzir o item selecionado, recebendo a partir do dispositivo servidor de mídia o fluxo de dados correspondente ao conteúdo multimídia selecionado anteriormente pelo usuário.

BRisa UPnP

O BRisa é um framework de código aberto desenvolvido em Python para oferecer suporte à construção de dispositivos e serviços UPnP.

O BRisa consiste nos seguintes elementos:

- as classes principais inerentes ao padrão UPnP, que permitem o desenvolvimento de novos dispositivos e serviços;
- a API de ponto de controle, que permite o desenvolvimento de pontos de controle personalizados;
- três aplicativos de usuário: *BRisa Configuration Tool*, *BRisa Media Server* e *BRisa Media Renderer*.

Tanto o dispositivo BRisa Media Server quanto o BRisa Media Renderer estão sendo desenvolvidos utilizando o framework BRisa, o que permite mostrar a flexibilidade do framework para o desenvolvimento de aplicações UPnP, além de disponibilizar uma implementação de exemplo da especificação UPnP A/V. A partir destes exemplos, pode-se modificá-los e estendê-los para a construção de novos serviços UPnP.

Para possibilitar o desenvolvimento de pontos de controle, o framework BRisa disponibiliza uma API que tem como objetivo abstrair do desenvolvedor as tarefas inerentes à descoberta de dispositivos e serviços UPnP, podendo englobar desde serviços relacionados à automação residencial – tais como a navegação por conteúdos multimídia e a obtenção de dados do ambiente, como a temperatura e a luminosidade – até o controle de serviços de rede, tais como o mapeamento de portas em

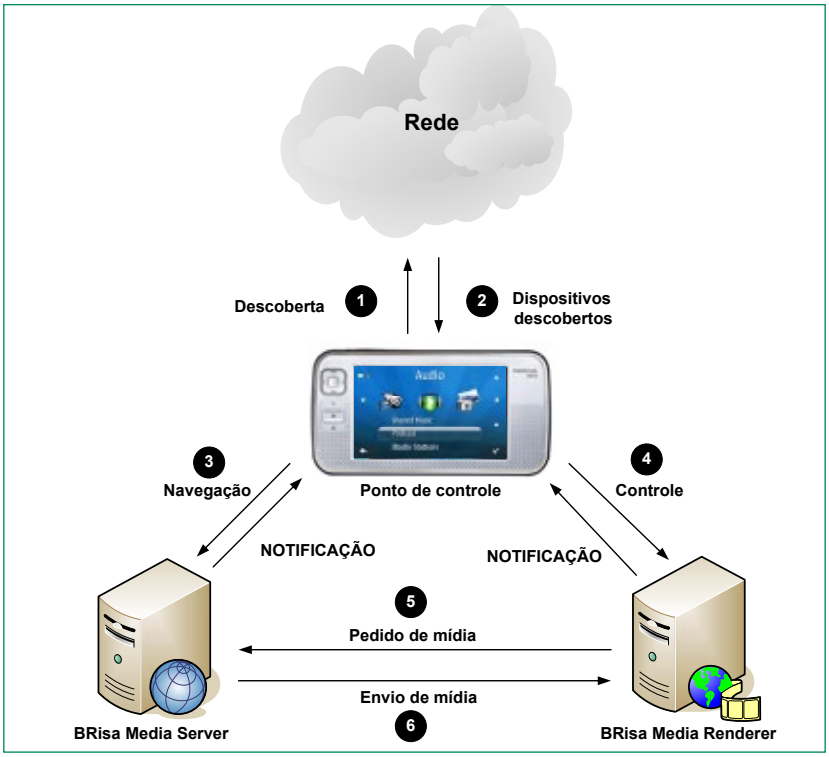

Figura 1 Esquema básico de comunicação entre dispositivos UPnP A/V.

Listagem 1: Mensagem de início do BRisa Media Server

```
01 BRisa Media Server version 0.7 started. Please refer to /home
➥/leandro/.brisa/brisa.log for more information.
02 Listen address: http://192.168.1.55:16672
```

roteadores e pontos de acesso. Como exemplo, explicaremos como a criação de um ponto de controle para o BRisa UPnP A/V pode ser desenvolvido. Além disso, mostraremos como desenvolver novos plugins para o BRisa Media Server, permitindo extensibilidade para novos tipos de fontes que disponibilizam conteúdos multimídia.

Para mais informações sobre como desenvolver aplicações baseadas no

Listagem 2: Ponto de controle na linha de comando

```
01 import sys
02 from brisa.control_point.control_point_av
➥ import ControlPointAV
03 from brisa.main import ThreadManager
04
05 class CommandLineControlPointAV(ControlPointAV):
06
07   def __init__(self):
08     ControlPointAV.__init__(self)
09     self.subscribe('new_device_event', self.
➥on_new_device)
10     self.subscribe('remove_device_event', self.
➥on_remove_device)
11     self.devices_found = []
12
13   def on_new_device(self, dev):
14     self.devices_found.append(dev)
15
16   def on_remove_device(self, udn):
17     for dev in self.devices:
18       if dev.udn == udn:
19         self.devices_found.remove(dev)
20         break
21
22   def cmd_list_devices(self):
23     n = 0
24     for dev in self.devices_found:
25       print 'device %d:' % n
26       print '\tudn:', dev.udn
27       print '\tfriendly_name:', dev.friendly_name
28       print '\tservices:', dev.services
29       print '\ttype:', dev.device_type
30       if dev.devices:
31         print '\tchild devices:'
32         for child_dev in dev.devices:
33           print '\t\tudn:', child_dev.udn
34           print '\t\tfriendly_name:', child_dev.
➥friendly_name
35           print '\t\tservices:', dev.services
36           print '\t\ttype:', child_dev.device_type
37       print
38       n += 1
39
40     def cmd_set_server(self, id):
41       self.current_server = self.devices_found[id]
42
43     def cmd_set_render(self, id):
44       self.current_renderer = self.devices_found[id]
45
46     def cmd_browse(self, id):
47       result = self.browse(id, 'BrowseDirect
➥Children', '*', 0, 10)['Result']
48       for d in result:
49       print "%s %s %s" % (d.id, d.title, d.upnp
➥_class)
50
51   def run(self):
52     exit = False
53     try:
54       while not exit:
55         c = str(raw_input('>>> '))
56
57         if c == 'start_search':
58           self.start_search(600,
➥ "upnp:rootdevice")
59           print 'search started!'
60         elif c == 'stop_search':
61           self.stop_search()
62           print 'search stopped!'
63         elif c == 'list':
64           self.cmd_list_devices()
65         elif c.startswith('browse'):
66           self.cmd_browse(c.split(' ')[1])
67         elif c.startswith('set_server'):
68           self.cmd_set_server(int(c.split(' ')
➥[1]))
69         elif c.startswith('set_render'):
70           self.cmd_set_render(int(c.split(' ')
➥[1]))
71         elif c.startswith('play'):
72           self.play(c.split(' ')[1])
73         elif c == 'exit':
74           exit = True
75         elif c == 'help':
76           print 'commands: start_search, stop_
➥search, list, ' \
77             'browse, set_server, set_render, play,
➥ exit, help'
78         c = ''
79     except KeyboardInterrupt, k:
80       print 'quiting'
81
82     ThreadManager().stop_all()
83
84 def main():
85   print "BRisa ControlPointAV example\n"
86   cmdline = CommandLineControlPointAV()
87   cmdline.run()
88   sys.exit(0)
89
90 if __name__ == "__main__":
91   main()
92
```

padrão UPnP com o framework BRisa, visite a seção de documentação do projeto BRisa [7].

Configuração do ambiente

Para maiores informações sobre o processo de instalação do framework BRisa numa distribuição Linux (tais como Debian, Fedora, Gentoo, ou Ubuntu), visite a documentação do usuário disponível em [8]. Qualquer que seja o gerenciador de pacotes utilizado pela distribuição, o projeto BRisa oferece um mecanismo portável para a instalação da infraestrutura em qualquer sistema, sendo apenas necessário possuir um compilador Python e algumas bibliotecas de dependências: GStreamer, GStreamer-plugins, Python GStreamer, Python Inotify, Python Mutagen.

Instalação e configuração

Para instalar o framework BRisa, siga os seguintes passos:

1. Faça o download da última versão do framework em [7];
2. Descompacte o pacote do framework como root: `tar zxvf brisa-1.5.2.tar.gz`;
3. No diretório do código-fonte do BRisa, execute o seguinte comando: `python setup.py install`.

É recomendável realizar a instalação da ferramenta de configuração do BRisa, o qual está disponível no site do projeto. Esta ferramenta abstrairá todo o processo de configuração do framework. Após configurar o BRisa Media Server, ele pode ser iniciado por meio de qualquer terminal com o comando `brisa-media-server`. Da mesma forma, é possível executar o BRisa Media Renderer com o comando `brisa-media-renderer`. A ferramenta de configuração permite que os usuários alterem as configurações padrão do servidor de mídias e dos plugins instalados. Os plugins permitem adicionar funcionalidades extras não encontradas na versão padrão do framework BRisa. Por exemplo, o plugin de sistemas de arquivos permite configurar diretórios no sistema de arquivos nos quais o BRisa Media Server irá compartilhar seus conteúdos multimídia.

Caso o BRisa Media Server inicie normalmente, ele mostrará uma mensagem similar àquela descrita na **listagem 1**. Caso contrário, verifique no arquivo `brisa.log` se houve algum problema durante o processo de instalação. Se o problema persistir, entre em contato com a equipe de desenvolvimento do BRisa [9].

Desenvolvimento de ponto de controle

O framework BRisa fornece uma API que disponibiliza as classes `brisa.control_point.control_point.ControlPoint` e `brisa.control_point.control_point_av.Control-PointAV` para a construção de pontos de controle UPnP. O exemplo descrito na **listagem 2** apresenta um ponto de controle baseado em linha de comando que tem como objetivo a descoberta de

Listagem 3: Uso do ponto de controle na linha de comando

```
01 # python control_point_av.py
02 BRisa ControlPoint example
03 >>> start_search
04 search started!
05
06 >>> list_devices
07 (A list of devices found is printed here. Look for the type field
➥ for each device to determine what you will set as
08 media server and what you will set as media renderer.)
09
10
11 >>> set_server 0
12 >>> set_render 1
13
14 Browse the root folder of the media server
15 >>> browse 0
16 1 Music object.container
17 3 Pictures object.container
18 12 Playlists object.container
19 2 Video object.container
20 (Exploring the folder 2 - Video)
21
22 >>> browse 2
23 5 Video Broadcast object.container
24
25 (Exploring the folder 5 - Video Broadcast)
26 >>> browse 5
27 youtube:7 YouTube object.container
28
29 (Exploring the items of the YouTube plug-in folder)
30 >>> browse youtube:7
31 youtube:hDiLH7jmVsU Around the World object.item.videoItem.
➥videoBroadcast
32 youtube:B5X5cZ62FGg Californication object.item.videoItem.
➥videoBroadcast
33 youtube:DF45X3mJsW Easily object.item.videoItem.videoBroadcast
34
35
36 (Play the video 'Around the World' in the media renderer)
37 >>> play youtube:hDiLH7jmVsU
38 >>> exit
```

dispositivos, a navegação por conteúdos multimídia de um dado servidor de mídias e a seleção de um renderizador de mídia para a reprodução do conteúdo multimídia desejado.

Na **linha 5 da listagem 2**, pode-se observar que a classe `CommandLineControlPointAV` estende a classe `brisa.control_point.control_point_av.ControlPointAV`, disponibilizando assim para esta aplicação todos os serviços inerentes à especificação UPnP A/V, tais como *play* e *pause*. Neste exemplo, o método `run()` lê continuamente o nome do comando que o usuário deseja executar (por meio da função Python `raw_input()`). As possíveis ações estão associadas a comandos de usuário, como *start_search*, *stop_search*, *list_devices*, *set_server*, *set_render*, *browse*, *play*, *exit*, e *help*. Dependendo do comando que o usuário deseja invocar, a aplicação seleciona a ação específica associada a esse comando (**linhas entre 57 e 76**).

Os comandos *start_search* e *stop_search* apenas delegam aos respectivos métodos da super-classe (`brisa.control_point.control_point_av.ControlPointAV`) o início e o término da descoberta de dispositivos UPnP. Já o comando `list_devices` invoca o método `CommandLineControlPointAV.cmd_list_devices()` (**linha 22**) para imprimir a lista de todos os dispositivos UPnP descobertos desde a primeira invocação do serviço `start_search()`. Finalmente, os comandos `set_server` e `set_render` especificam o servidor de mídias e o dispositivo de reprodução, respectivamente.

O framework BRisa fornece quatro informações importantes sobre os dispositivos descobertos: o UDN do dispositivo, que identifica exclusivamente este dispositivo numa rede UPnP; o tipo do dispositivo, que determina se o dispositivo é um servidor de mídias, um reprodutor de mídias ou qualquer outro dispositivo UPnP; a lista de serviços disponibilizados pelo dispositivo em questão e, finalmente, o nome do dispositivo descoberto.

A API do framework BRisa para a construção de pontos de controle é responsável por notificar a sua aplicação sobre novos dispositivos que são descobertos na rede. Desta forma, as **linhas 9 e 10 da listagem 2** descrevem a maneira (pelo método `brisa.control_point.control_point_av.ControlPointAV.subscribe()`) como pontos de controle são registrados em mensagens de anúncios de entrada e saída de dispositivos na rede, respectivamente. Ambas as notificações de eventos são realizadas pela API através de *callbacks* para os métodos `CommandLineControlPointAV.on_new_device()` e `CommandLine-ControlPointAV.on_remove_device()`. Observe que, em virtude do mecanismo de herança da linguagem Python, a classe `CommandLineControlPointAV` herda o método `subscribe()` da super-classe `brisa.control_point.control_point_av.ControlPointAV`.

Os comandos `browse()` e `play()` permitem, respectivamente, que o ponto de controle navegue por conteúdos multimídia de um servidor de mídia e inicialize a reprodução de um conteúdo multimídia em um dispositivo de reprodução desejado. No primeiro caso, o método `CommandLineControlPointAV.cmd_browse()` é responsável por invocar o método `browse()` da super-classe para recuperar informações do conteúdo multimídia do servidor de mídia selecionado. De forma similar, o método `play()` da super-classe é invocado para inicializar a reprodução do conteúdo multimídia no dispositivo de reprodução selecionado pelo usuário. A **listagem 3** descreve uma maneira para utilizar o ponto de controle UPnP discutido nesta seção.

Plugin do Media Server

O framework BRisa já disponibiliza uma implementação de um servidor de mídia (BRisa Media Server) com suporte às funcionalidades básicas definidas na especificação UPnP A/V. A implementação é baseada no mecanismo de plugins, o que permite a adição de novas funcionalidades ao servidor de mídia. A seguir, mostraremos os passos para a criação de um plugin para o BRisa Media Server com o objetivo de recuperar conteúdos multimídia do YouTube.

A arquitetura de plugins do BRisa Media Server permite a adição de novos plugins com base na classe abstrata `brisa.services.cds.plugin.Plugin`, que disponibiliza os métodos abstratos `load()`, `unload()`, `browse()` e `search()`. Os dois últimos métodos são os mais importantes, pois permitem que o plugin responda a requisições remotas de navegação e a buscas por conteúdos multimídia, respectivamente.

O framework BRisa disponibiliza uma API de configuração que permite a recuperação de informações

Listagem 4: Estrutura de plugins do BRisa

```
01 from brisa.services.cds.plugin import Plugin
02
03 class MyOwnBRisaPlugin(Plugin):
04
05 def __init__(self):
06   Plugin.__init__(self)
07
08 def load(self):
09   pass
10
11 def unload(self):
12   pass
13
14 def browser(self):
15   pass
16
17 def search(self):
18   pass
```

a partir de arquivos de configuração do tipo `.ini`. O arquivo de configuração padrão utilizado pelo framework BRisa e pelas implementações fica armazenado no arquivo `~/.brisa/brisa.conf`. Entretanto, se for preciso armazenar e recuperar parâmetros de configuração, pode-se utilizar a API de configuração do projeto BRisa.

O subsistema de diretório de conteúdo do BRisa é responsável por detectar e carregar o plugin. O diretório de conteúdo também invoca os processos de navegação (*browsing*) e busca (*searching*) quando o BRisa Media Server recebe alguma requisição de um ponto de controle para a recuperação de conteúdos multimídia compartilhado por algum plugin. Como exemplo, a **listagem 4** apresenta uma classe de plugin (`MyOwnBRisaPlugin`), que estende a super-classe `brisa.services.cds.plugin.Plugin`.

Para instalar um novo plugin, é necessário criar uma nova pasta chamada `my_plugin/` no diretório `$PYTHON_DIR/site-packages/brisa/services/cds/plugins`, salvando o arquivo que contém a classe python no plugin chamado `implementation.py`. A **figura 2** ilustra o processo de utilização do novo plugin, no qual o BRisa Media Server automaticamente carrega o plugin e exporta o conteúdo multimídia compartilhado pelo plugin através do subsistema do diretório de conteúdo do BRisa Media Server.

Como podemos observar na **figura 2**, o plugin desenvolvido pode utilizar qualquer mecanismo de persistência para armazenamento dos conteúdos multimídia. Desta forma, quando uma requisição de navegação é recebida pelo BRisa Media Server (etapa 1), o subsistema de diretório de conteúdo do BRisa redireciona a requisição para o plugin que se adequa àquela determinada requisição (etapas 2 e 3).

Por sua vez, o plugin recupera os conteúdos multimídia desejados e os retorna ao subsistema de diretório de conteúdo (etapa 4), que por sua

Listagem 5: Plugin do YouTube para BRisa

```
01 from youtube_api import YouTubeClient
02 from youtube_dl import get_real_video_url
03 from brisa.services.cds.plugin import Plugin
04 from brisa.utils import properties
05 from brisa.upnp.didl.didl_lite import
➥ VideoBroadcast
06 from brisa import config
07
08 youtube_video_url = config.get_
➥parameter('youtube', 'videourl')
09
10 class YouTubeItem(VideoBroadcast):
11
12   protocolInfo = 'http-get:*:video/flv:*'
13
14   def __init__(self, id, parent_container_id,
➥ namespace, title, description,
15        duration, author, rating):
16     VideoBroadcast.__init__(self, id,
17                 parent_container_id=parent_
➥container_id,
18                 namespace=namespace,
19                 author=author,
20                 rating=rating,
21                 duration=duration,
22                 title=title,
23                 name=title,
24                 description=description)
25                 self._uri = get_real_video_
➥url("%s%s" % (youtube_video_url, id))
26
27   def _gen_uri(self):
28     return self._uri
29
30 class YouTubePlugin(Plugin):
31   id = "7"
32   name = 'youtube'
33   usage = config.get_parameter_bool('youtube',
➥ 'usage')
34   videos = {}
35
36   def __init__(self, *args, **kwargs):
37     Plugin.__init__(self, *args, **kwargs)
38
39   def _register_plugin(self):
40     self.ytcontainer = self.plugin_manager.
➥root_plugin.add_container("YouTube", self.id,
➥ "5", self)
41
42   def load(self):
43     self._register_plugin()
44     yt = YouTubeClient('ngR1Q8wOOEk')
45     username = config.get_parameter('youtube',
➥ 'username')
46     for video in yt.list_by_user(username):
47       video_info = yt.get_details(video['id'])
48       self.add_item(video['id'], self.
➥ytcontainer.id,
49           video_info['title'], video_
➥info['description'],
50           video_info['length_seconds'], date,
51           video_info['author'], video_
➥info['rating_avg'])
52
53   def add_item(self, video_id, parent_id, title,
➥ description, duration, date, author, rating):
54     item = YouTubeItem(video_id, parent_id,
➥ self.name, title, description,
55         duration, date, author, rating)
56         self.videos[video_id] = item
57
58   def browse(self, id, browse_flag, filter,
➥ starting_index, requested_count, sort_criteria):
59     if browse_flag == 'BrowseMetadata' and id !=
➥ self.id:
60       return [self.videos[id]]
61     else:
62       return self.videos.values()
```

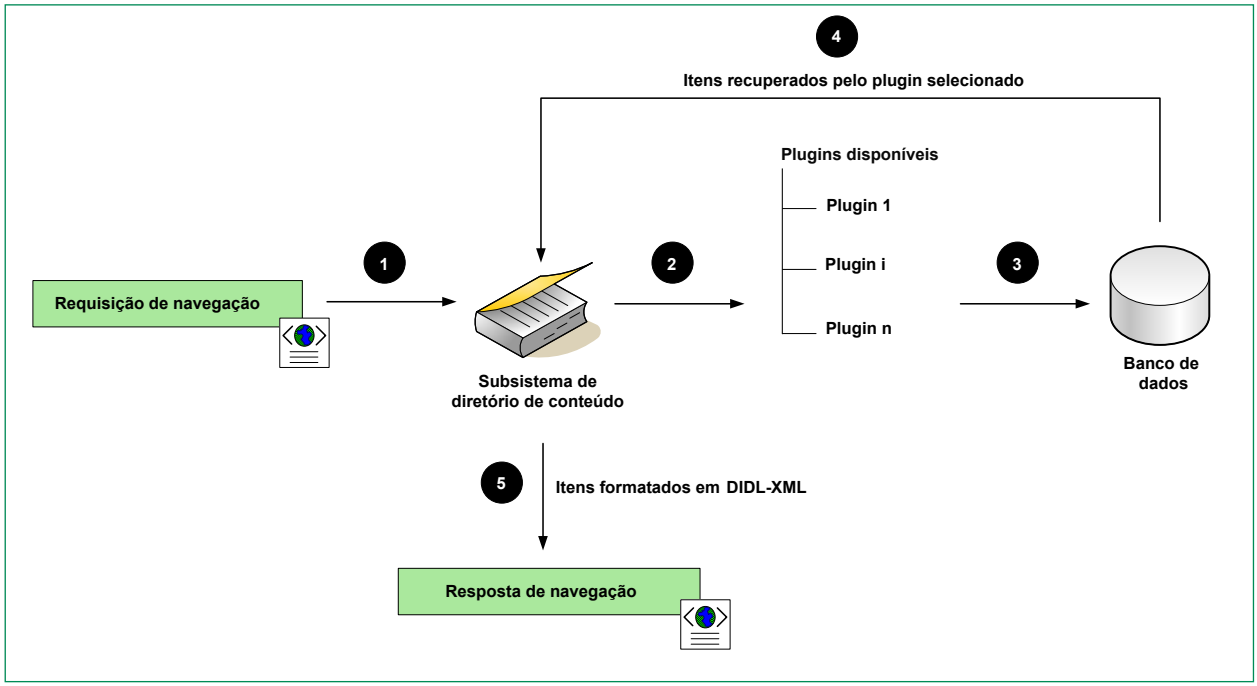

Figura 2 O subsistema de diretório de conteúdo do BRisa gerencia informações sobre os dados disponíveis.

vez formata os conteúdos para um modelo padrão conhecido por DIDL (*Digital Item Declaration Language*) (etapa 5). Esse padrão é utilizado para representar conteúdos digitais, tais como áudio, vídeo e imagem [10].

Finalmente, ao receber o conteúdo multimídia formatado em DIDL, o BRisa Media Server envia o conteúdo multimídia por uma mensagem SOAP para o ponto de controle. Por ser baseado numa arquitetura extensível, o BRisa Media Server possibilita a adição de plugins que possuem como objetivo o compartilhamento de conteúdo multimídia de uma fonte específica. Para mais informações sobre o desenvolvimento de plugins para o projeto BRisa, acesse a seção 10 da documentação de desenvolvimento do BRisa [11].

Como exemplo, a **listagem 5** descreve o plugin para recuperar conteúdos multimídia do YouTube. Para recuperar os vídeos no formato `.fla` e as respectivas informações associadas a eles, usa-se os módulos `youtube_api` e `youtube_dl`, como se pode observar nas **linhas 1 e 2**. Entre as **linhas 30 e 33**, cria-se o novo plugin (`YouTubePlugin`), definindo o identificador (`id`), o nome, que pode ser uma breve descrição do plugin, e o parâmetro `usage`, que determina se o BRisa Media Server deve ou não carregar automaticamente o plugin em questão.

O método de classe `_register_plugin()`, definido na **linha 39**, é utilizado para registrar a pasta do novo plugin (formalmente conhecido como contêiner do servidor de mídias UPnP) na árvore de navegação do BRisa Media Server. Observe na **listagem 3** que o comando `browse` é executado três vezes: `browse 0` para navegar na pasta principal (`root/`), `browse 2` para navegar na pasta `video/` e, finalmente, `browse 5` para navegar na pasta `videoBroadcast/`, onde está registrado o plugin de pastas do YouTube.

Continuando na **listagem 5**, na **linha 40** o plugin do YouTube é registrado e colocado dentro do diretório `VideoBroadcast/` do diretório de conteúdo do BRisa Media Server. Observe que o plugin do YouTube faz uso da classe `RootPlugin`, fornecida pelo BRisa Media Server para registrar a pasta. O plugin *RootPlugin* é um plugin especial que cria os diretórios padrão do BRisa Media Server, tais como os diretórios da raiz: `audio/`, `vídeo/` e `images/`.

Cada item do contêiner padrão é identificado por um *id*. O *id* do contêiner `VideoBroadcast` é 5, que corresponde ao terceiro parâmetro definido no método `add_container()` disponibilizado pelo plugin `RootPlugin`. Nas **linhas 41 a 50**, o método `load()` do plugin carrega todos os vídeos do usuário e os armazena na lista de vídeos representada pelo atributo de vídeo do plugin. Na **linha 45**, o plugin faz uso da API de configuração do BRisa (BRisa Configuration API). Finalmente, o método `browse` é invocado pelo subsistema de diretório de conteúdo (quando o usuário envia uma requisição de navegação para `youtube:7`), o qual retorna a lista de vídeos carregado pelo plugin (**listagem 3**).

Para identificar a pasta de diretório de conteúdo do YouTube, o subsistema combina os atributos `id` e `name` pertencentes ao plugin. No caso do plugin do Youtube, o resultado dessa

combinação é `youtube:7`. Esta mesma ideia também está relacionada aos items compartilhados do YouTube, tais como `youtube:hDiLH7jmVsU`, utilizado no exemplo descrito na **listagem 3**. Desta forma, quando o subsistema de diretório de conteúdo recebe uma requisição para a dupla `youtube:hDiLH7jmVsU`, o subsistema a divide em duas partes, com base no caractere `:`, permitindo identificar o plugin e o item desejados, especificados pelo ponto de controle.

Após o fim da implementação do plugin, deve-se criar o diretório `my_youtube_plugin/` dentro da árvore de diretório `$PYTHON_DIR/site-packages/brisa/services/cds/plugins/`, colocar o código-fonte do novo plugin no arquivo `implementation.py` e salvá-lo no diretório recém-criado. O BRisa Media Server carregará seu plugin automaticamente. Por meio de um ponto de controle baseado em linha de comando, é possível testar o plugin para navegar nos conteúdos multimídia compartilhados.

Conclusão

Este artigo apresentou os conceitos básicos do padrão de conectividade UPnP. Neste padrão é definida uma arquitetura de rede aberta e distribuída para conectividade *ad hoc* entre diferentes dispositivos com diferentes interfaces de comunicação, proporcionando serviços heterogêneos de forma transparente aos usuários. Um exemplo para o padrão UPnP é a especificação UPnP A/V, que propõe uma abordagem para que pontos de controle possibilitem a navegação por conteúdos multimídia armazenados em um servidor de mídia e os reproduzam em um dispositivo denominado renderizador de mídias.

Neste contexto, foi apresentado neste artigo o framework BRisa, que objetiva oferecer uma arquitetura extensível para o desenvolvimento de aplicações UPnP. Com efeito, o projeto BRisa tem ganhado a atenção da comunidade científica, porém, assim como outros projetos de código aberto, o BRisa ainda se encontra em fase de desenvolvimento. Atualmente, novos plugins estão sendo desenvolvidos, integrando outros serviços de compartilhamento de conteúdos multimídia tais como Yahoo Music, Facebook, Orkut e PicasaWeb. Esses plugins proporcionarão uma forma centralizada e transparente para os usuários compartilharem conteúdos multimídia por meio de serviços UPnP baseados no BRisa Media Server. ■

Mais informações

[1] D. Saha e A. Mukherjee, "Pervasive Computing: A Paradigm for the 21st Century". IEEE Computer Society Press, páginas 25-31.

[2] Fórum UPnP: http://www.upnp.org/

[3] P. Louridas, "SOAP and Web Services". IEEE Software, vol. 23, nº 6, páginas 62-67

[4] Especificações UPnPA/V1.0: http://www.upnp.org/standardizeddcps/mediaserver.asp

[5] L. Sales et al., "BRisa UPnP A/V Framework". IEEE International Conference on Consumer Electronics, páginas 1-2.

[6] J. Kim et al., "Implementation of the DLNA Proxy System for Sharing Home Media Contents", IEEE Transactions on Consumer Electronics, vol. 53, nº 1

[7] Site oficial do BRisa: http://brisa.garage.maemo.org

[8] Documentação do BRisa para usuários: http://brisa.garage.maemo.org/documentation-enduser.htm

[9] Canal IRC do BRisa: irc.freenode.com, canal brisa

[10] S. Hashemipour e M. Ali, "MPEG-21 & DIDL: Dawn of a New Multimedia Eve". IEEE International Symposium on Consumer Electronics, páginas 91-95.

[11] Documentação do BRisa para desenvolvedores: http://brisa.garage.maemo.org/documentation-developer.htm

Sobre o autor

Leandro Melo de Sales usa Linux desde 1997 e começou o projeto UPnP BRisa no final de 2006. Ele chefia o grupo de desenvolvedores do framework UPnP BRisa e seus aplicativos básicos, e trabalha no Laboratório de Sistemas Embarcados e Computação Pervasiva da Universidade Federal de Campina Grande, PB, apoiado pelo Instituto Nokia de Tecnologia.

Gostou do artigo?

Queremos ouvir sua opinião. Fale conosco em cartas@linuxmagazine.com.br

Este artigo no nosso site: http://lnm.com.br/article/3048