Exploring the Universal Plug and Play Architecture

CUSTOM CONNECTIONS

Universal Plug and Play provides an easy framework for seamless integration of network devices. Learn how to build your own UPnP solution using the open source BRisa framework. **BY LEANDRO MELO DE SALES**

The arrival of consumer electronic devices such as PDAs, cell phones, and Internet tablets has placed new emphasis on techniques for connecting and sharing information [1]. The Universal Plug and Play (UPnP) standard [2] is a good candidate to provide pervasive services for a new generation of electronic devices.

The original concept of *Plug and Play* focused on dynamically attaching devices directly to a computer. A device driver controlled by the operating system detected the peripheral and made it available to the user via system calls. The new Universal Plug and Play standard offers a radically different approach: Devices operate on a network and are detected with the use of network protocols. The system calls used in the previous plug and play architecture are replaced by remote method invocation with the use of SOAP web services.

A UPnP network is a collection of interconnected computers, network appliances, and wireless devices that use UPnP to discover, advertise, and access network services. The goal is to provide an easy-to-use framework for creating tools to support the communication of network-based devices. UPnP achieves this goal by defining and publishing UPnP device control protocols built on open, Internet-based communication standards [3]. As you might expect, UPnP supports connections for devices such as cell phones or MP3 players, but UPnP also offers benefits for connecting to conventional peripherals such as printers, as well as wireless household electronic gadgets for controlling appliances, lights, doors, and curtains.

An example of a UPnP standard is the UPnP AV specification [4], released in the middle of 2002 and updated in 2006. UPnP AV supports UPnP-enabled devices that share multimedia content and services. The UPnP AV standard defines protocols for discovering, sharing, and playing multimedia content.

In this article, I will explain how to create a simple multimedia application based on the UPnP architecture. To build this simple solution, I will use the BRisa framework [5], which is an open source framework for developing custom UPnP applications.

The UPnP AV Specification

UPnP Audio and Video (AV) is a specification within the UPnP standard supervised by the Digital Living Network Alliance (DLNA) [6]. DLNA supports a vision of a wired and wireless interoperable network of personal computers, consumer electronics, and mobile devices sharing multimedia content and services in a seamless environment.

The UPnP AV specification defines a system of three UPnP devices:

- Media server – a component that shares multimedia items, such as audio (*.mp3*, *.wma*, *.ogg*), video (*.mpeg*, *.wmv*, *.fla*), image (*.gif*, *.jpg*, *.png*), and remote multimedia streaming, such as Internet radio stations and online photo albums.
- Media renderer – responsible for reproducing all multimedia data shared by any UPnP media server and providing mechanisms that allow remote devices to play, pause, stop, and seek a multimedia item.
- Multimedia control point – a small device, such as a cell phone or Internet tablet, that gives the user control over media servers and renderers.

Joining Up

Any UPnP-compatible device from any vendor can dynamically join a UPnP network through the following process:

- Step 0 – Addressing: devices obtain an IP address
- Step 1 – Discovery: devices advertise their services to control points on the network
- Step 2 – Description: Devices convey capabilities on request from control points, learning about the presence and capabilities of other devices
- Step 3 – Control: Devices provide services, and control points can send actions to these services
- Step 4 – Eventing: whenever requested, the control points register to remote devices' events
- Step 5 – Presentation: device may provide a presentation web page for a control point, allowing a user to control the remote device and view the device status

Regardless of the steps and states of a UPnP device, the device can leave the network automatically without leaving any unwanted state information behind.

Figure 1 illustrates the UPnP device discovery and service consumption process. When the control point is started, it sends a multicast message to the network asking for media servers and media renderers (Step 1). When the UPnP devices receive the multicast message sent by the UPnP control point, each device responds to the control point requester (Step 2).

At this point, the UPnP control point knows about all of the connected UPnP devices. The user, operating through the interface provided on the control point, browses and selects any multimedia item listed by the UPnP media server (Step 3). After selecting a multimedia item, the user chooses a media renderer that will process (play) the multimedia item (Step 4). After selecting a media server and a media renderer, the user simply clicks on the Play button of the UPnP control point, and the media renderer starts to play the multimedia stream sent by the media server.

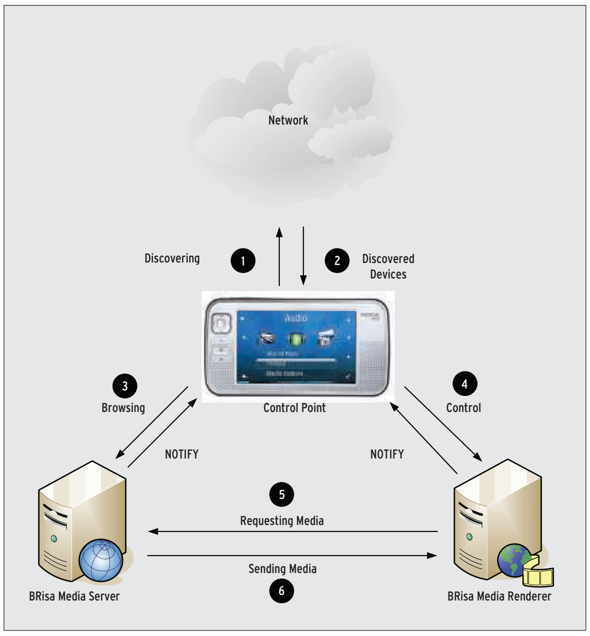

Figure 1: Basic communication scheme for UPnP audio/video devices.

BRisa UPnP Framework

BRisa is an open source framework written in Python that supports the development of UPnP devices and services.

The BRisa framework consists of the following:

- core UPnP classes that allow the development of new devices and services,
- the control point API,
- three end-user applications: the BRisa configuration tool, the media server, and the media renderer.

The BRisa media server and media renderer are developed on top of the BRisa UPnP framework, providing a reference implementation for the UPnP media server and media renderer specifications. These sample UPnP devices demonstrate

BRisa Dependencies

Before setting up BRisa, install the following Python packages:

- Python 2.5+ – *http://python.org/*
- GStreamer 0.10 – *http://gstreamer.freedesktop.org/download*
- GStreamer Plugins 0.10 – *http://gstreamer.freedesktop.org/download*
- Python GStreamer – *http://gstreamer.freedesktop.org/modules/gst-python.html*
- Python pyinotify – *http://pyinotify.sourceforge.net/*
- Python Mutagen – *http://pypi.python.org/pypi/mutagen/1.12*

Listing 1: BRisa Media Server Startup Message

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

the power and flexibility of the BRisa UPnP framework; you can modify and extend these examples to offer new network services based on other UPnP specifications.

The UPnP Control Point API included with the BRisa framework lets developers consume UPnP services for discovering devices, browsing for multimedia content, sending jobs to printers, turning home lights on or off, retrieving temperature information from temperature monitors, and controlling the motion of window curtains. As an example of what you can do with the BRisa framework, I

Listing 2: Command-Line Control Point

```
import sys
from brisa.control_point.control_point_av import ControlPointAV
from brisa.main import ThreadManager

class CommandLineControlPointAV(ControlPointAV):

    def __init__(self):
        ControlPointAV.__init__(self)
        self.subscribe('new_device_event', self.on_new_device)
        self.subscribe('remove_device_event', self.on_remove_device)
        self.devices_found = []

    def on_new_device(self, dev):
        self.devices_found.append(dev)

    def on_remove_device(self, udn):
        for dev in self.devices:
            if dev.udn == udn:
                self.devices_found.remove(dev)
                break

    def cmd_list_devices(self):
        n = 0
        for dev in self.devices_found:
            print 'device %d:' % n
            print '\tudn:', dev.udn
            print '\tfriendly_name:', dev.friendly_name
            print '\tservices:', dev.services
            print '\ttype:', dev.device_type
            if dev.devices:
                print '\tchild devices:'
                for child_dev in dev.devices:
                    print '\t\tudn:', child_dev.udn
                    print '\t\tfriendly_name:', child_dev.friendly_name
                    print '\t\tservices:', dev.services
                    print '\t\ttype:', child_dev.device_type
            print
            n += 1

    def cmd_set_server(self, id):
        self.current_server = self.devices_found[id]

    def cmd_set_render(self, id):
        self.current_renderer = self.devices_found[id]

    def cmd_browse(self, id):
        result = self.browse(id, 'BrowseDirectChildren', '*', 0, 10)['Result']
        for d in result:
            print "%s %s %s" % (d.id, d.title, d.upnp_class)

    def run(self):
        exit = False
        try:
            while not exit:
                c = str(raw_input('>>> '))

                if c == 'start_search':
                    self.start_search(600, "upnp:rootdevice")
                    print 'search started!'
                elif c == 'stop_search':
                    self.stop_search()
                    print 'search stopped!'
                elif c == 'list':
                    self.cmd_list_devices()
                elif c.startswith('browse'):
                    self.cmd_browse(c.split(' ')[1])
                elif c.startswith('set_server'):
                    self.cmd_set_server(int(c.split(' ')[1]))
                elif c.startswith('set_render'):
                    self.cmd_set_render(int(c.split(' ')[1]))
                elif c.startswith('play'):
                    self.play(c.split(' ')[1])
                elif c == 'exit':
                    exit = True
                elif c == 'help':
                    print 'commands: start_search, stop_search, list, ' \
                          'browse, set_server, set_render, play, exit, help'
                c = ''
        except KeyboardInterrupt, k:
            print 'quiting'

        ThreadManager().stop_all()

def main():
    print "BRisa ControlPointAV example\n"
    cmdline = CommandLineControlPointAV()
    cmdline.run()
    sys.exit(0)

if __name__ == "__main__":
    main()

```

will describe a BRisa UPnP AV scenario that shows how to create a BRisa control point. I will also demonstrate to customize the BRisa media server by creating a media server plugin.

For more information on programming with the BRisa UPnP framework, visit the documentation section of the BRisa project website [7].

Setting Up the Environment

Consult the BRisa end-user documentation [8] for details on how to install the BRisa framework through your favorite Linux distribution (such as Debian, Fedora, Gentoo, or Ubuntu).

Regardless of which Linux package manager you use, the BRisa project provides a portable mechanism for installing the core infrastructure in any Python-enabled system. To set up the BRisa framework, you'll need to install a few additional Python packages. See the box titled "BRisa Dependencies."

Install and Configure

To install the BRisa UPnP framework on your computer:

1. Download the latest BRisa framework package [7].
2. Decompress the BRisa framework package as root:

```
tar zxvf brisa-{version}.tar.gz
```

3. Go to the BRisa source code directory:

```
cd brisa-{version}
```

4. Run the BRisa installation:

```
python setup.py install
```

It is also a good idea to install the BRisa configuration tool, which is available at the BRisa website. The configuration tool will save you from having to edit configuration text files by hand.

After configuring the BRisa media server, you can start it by simply executing the command *brisa-media-server* on a terminal, regardless of the current directory you are in. Similarly, you can start the BRisa media renderer with the command *brisa-media-renderer*. The configuration tool allows users to change their media server default parameters and the installed plugins. (Plugins provide additional functionality not found in the default media server. For example, the Filesystem plugin lets you set up filesystem directories in which the BRisa media server will share your audio, video, and image files.)

If the BRisa media server starts correctly, you'll see the message shown in Listing 1. If you don't see this message, check the *brisa.log* file to see if you can detect the problem. If the problem persists, contact the BRisa team [9] or try the troubleshooting section of the BRisa documentation [10].

Developing a UPnP Control Point

After you have installed the BRisa UPnP framework, you can use the framework to develop a simple command-line UPnP control point. The control point API lets you create a control point class and inherit from either the *brisa.control_point.control_point.ControlPoint* or the *brisa.control_point.control_point_av.ControlPointAV* class, depending on your purpose. The control point example shown in Listing 2 searches UPnP devices, browses for multimedia items in a given media server, and asks for a media renderer to play an audio file shared by the media server.

In line 5 of Listing 2, the class *CommandLineControlPointAV* inherits from the BRisa class *brisa.control_point.control_point_av.ControlPointAV*. All available multimedia methods specified by the UPnP AV specification and provided by the class *brisa.control_point.control_point_av.ControlPointAV* are thus also used by the *CommandLineControlPointAV* class. This example creates the *run()* class method that reads continuously from the command line using the Python built-in *raw_input()* function (line 55).

Listing 3: Using the Command-Line Control Point

```
# python control_point_av.py
BRisa ControlPoint example
>>> start_search
search started!

>>> list_devices
(A list of devices found is printed here. Look for the type field for each device to determine what you will set as
media server and what you will set as media renderer.)


>>> set_server 0
>>> set_render 1

Browse the root folder of the media server
>>> browse 0
     1   Music        object.container
     3   Pictures     object.container
    12   Playlists    object.container
     2   Video        object.container
(Exploring the folder 2 - Video)

>>> browse 2
     5   Video Broadcast object.container

(Exploring the folder 5 - Video Broadcast)
>>> browse 5
     youtube:7   YouTube   object.container

(Exploring the items of the YouTube plug-in folder)
>>> browse youtube:7
youtube:hDiLH7jmVsU   Around the World   object.item.videoItem.videoBroadcast
youtube:B5X5cZ62FGg   Californication   object.item.videoItem.videoBroadcast
youtube:DF45X3mJsW   Easily   object.item.videoItem.videoBroadcast


(Play the video 'Around the World' in the media renderer)
>>> play youtube:hDiLH7jmVsU
>>> exit
```

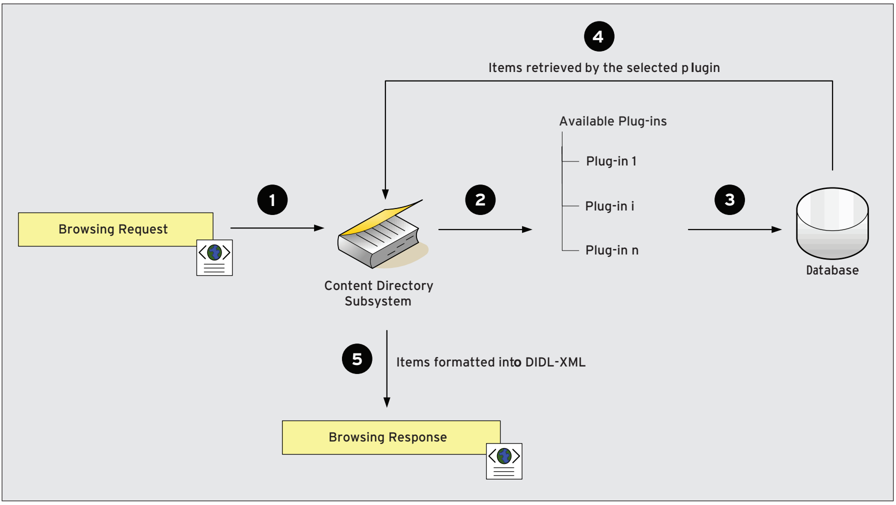
4
Items retrieved by the selected plugin
Available Plug-ins
Plug-in 1
Plug-in i
Plug-in n
1
2
3
Browsing Request
Content Directory Subsystem
Database
5
Items formatted into DIDL-XML
Browsing Response

Figure 2: The BRisa Content Directory subsystem manages information on available data.

Depending on the command line typed by the user, the application takes a specific action (lines 57 to 76). The possible actions are associated with the user commands *start_search*, *stop_search*, *list_devices*, *set_server*, *set_render*, *browse*, *play*, *exit*, and *help*.

The *start_search* and *stop_search* user commands just delegate a simple call to the respective methods available in the *brisa.control_point.control_point_av.ControlPointAV* class. When the user types the command *list_devices* in the prompt, the method *CommandLineControlPointAV.cmd_list_devices()* (line 22) is invoked. This method prints the list of the UPnP devices discovered since the first time the *start_search* command was run by the user.

BRisa's UPnP framework provides four important bits of information about the discovered devices: the device's UDN, which is an exclusive identifier for the UPnP device in the network; the device type, which tells whether the device is a media server, a media renderer, or some other UPnP device; a list of services provided by the device; and the Friendly Name, a short, more convenient name for the device.

The *set_server* and *set_render* commands specify the media server and media renderer, respectively. While you are using the control point infrastructure of the BRisa UPnP Framework, the BRisa Control Point API can notify your application about any new UPnP device that connects to the network.

In Listing 2, lines 9 and 10, I have used the *brisa.control_point.control_point_av.ControlPointAV.subscribe()* method to tell the BRisa Control Point API to notify the *CommandLineControlPointAV* class when a new UPnP device has been discovered and also to notify this same class when a UPnP device has left the network. The notification for both events is performed by the BRisa Control Point API by calling the subscribed callback functions *CommandLineControlPointAV.on_new_device()* and *CommandLineControlPointAV.on_remove_device()*. Note that because of Python's inheritance mechanism, the class *CommandLineControlPointAV* inherits the *subscribe()* superclass method.

The *browse* command allows the application to browse a specific item in the media server, and the *play* command sends a play request to the media renderer. The class method *CommandLineControlPointAV.cmd_browse()* is invoked when the command *browse* is typed. The *cmd_browse()* command makes use of the *browse()* superclass method to retrieve information on the item from the current media server.

A similar process occurs with the *play()* superclass method, which makes use of a media renderer previously selected by the user to play the specified multimedia item. Listing 3 shows how to use this UPnP control point.

Developing a BRisa Plugin

The BRisa framework comes with a ready-made media server. Rather than create a new one from scratch, you can adapt the BRisa media server with the use of plugins. Next, I will show you how to develop a simple YouTube plugin for the BRisa media server. A full implementation of a YouTube plugin is provided by the BRisa Project.

The BRisa media server plugin architecture (Listing 4) allows the addition of new plugins based on the abstract class called *Plugin*. Once you have created your own classes that inherit from the *brisa.services.cds.plugin.Plugin* class, your class becomes a BRisa media server plugin, and you should provide the implementation for the abstract methods *load()*, *unload()*, *browse()*, *search()*. The most important methods are the last two, which make the plugin capable of responding to remote browsing or searching for multimedia items.

The BRisa UPnP framework provides a configuration API that allows retrieval of values from a configuration *.ini*–style file. The default configuration file for all parameters used by the BRisa UPnP Framework and its applications is stored in the file *~/.brisa/brisa.conf*. Therefore, if you need to store and retrieve configuration parameters, you should use the BRisa configuration API.

BRisa's Content Directory subsystem detects and loads the plugin. The Content Directory also delegates the browsing and searching requests when the BRisa media server receives a request from a remote control point to retrieve the multimedia items shared by any installed plugin.

To deploy your plugin after you finish implementing it, simply create a folder

Listing 4: BRisa Plugin Structure

```
from brisa.services.cds.plugin
 import Plugin

class MyOwnBRisaPlugin(Plugin):

    def __init__(self):
         Plugin.__init__(self)

    def load(self):
         pass

    def unload(self):
         pass

     def browser(self):
         pass

    def search(self):
         pass
```

named *my_plugin* under the directory *$PYTHON_DIR/site-packages/brisa/services/cds/plugins* and save your plugin class in a file called *implementation.py* directory. The BRisa media server will automatically load and export the content shared by your plugin through the BRisa Content Directory subsystem. This mechanism is illustrated in Figure 2.

As shown in Figure 2, the plugin can use any kind of storage mechanism to store its contents. When a browsing action arrives from the network (step 1), the BRisa Content Directory subsystem redirects the browsing to the correct plugin (steps 2 and 3).

The plugin properly handles the browse request and returns the multimedia items to the BRisa Content Directory subsystem (step 4). The Content Directory subsystem gets the returned list of multimedia items and formats them into an XML-specific standard known as DIDL (Digital Item Declaration Language) (step 5). DIDL is used to represent complex digital objects [11].

Finally, the BRisa media server gets DIDL-XML content, wraps it up into a SOAP message, and sends it back to the remote control point that formats the output in the device screen. On the basis of this extensible architecture, the BRisa media server offers the deployment of third-party plugins that can share multimedia data from a specific source. For more about plugin development, see Section 10 of the BRisa developer documentation [10].

Listing 4 presents a simple plugin class stub. Note that the class *MyOwnBRisaPlugin* inherits from the class *brisa.services.cds.plugin.Plugin*. As an example of a real plugin for the BRisa media server, Listing 5 shows the most important part of the YouTube BRisa media server plugin. In Listing 5, a third-party module is used to retrieve YouTube items from the YouTube website. The *youtube_api* and *youtube_dl* modules are imported in lines 1 and 2 and are used to retrieve video information and download the *.fla* file. Around lines 30 and 33, the class *YoutubePlugin* inherits the BRisa plugin class (as in Listing 4). A BRisa media server plugin has three important attributes: an *id* that uniquely identifies a plugin, the *name* that is a short description of the plugin, and a *usage* that indicates to the BRisa media server whether it should load the plugin automatically.

Listing 5: BRisa YouTube Plugin

```
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('ngR1Q8w0OEk')
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()
```

The class method *_register_plugin()*, line 39, is used to register a plugin folder (formally known as a UPnP media server container) in the browse tree of the BRisa media server. Note that in Listing 3, the browse command is executed three times: browsing folder *0* (root folder), then browsing the *video* folder (*id* 2), and then the *videoBroadcast* folder (*id* 5), where the YouTube folder plugin is registered.

Line 40 (Listing 5) registers the YouTube plugin folder under the folder *VideoBroadcast*. Note that the YouTube plugin uses the BRisa *RootPlugin* to register its folder. The *RootPlugin* is a special Content Directory subsystem plugin that creates the default containers of the BRisa media server (Audio, Video, and Pictures), as shown in Figure 3.

Each item of the default containers is statically identified by an *id*. The id for *VideoBroadcast* is 5, which corresponds to the third parameter passed to the method *add_container()* provided by the *RootPlugin*. From line 41 to 50, the plugin method *load()* loads all the user's uploaded videos and stores them in the list of videos represented by the *videos* attribute of the plugin. In line 45, the plugin makes use of the BRisa Configuration API. Finally, the method *browse* is called by the Content Directory subsystem when the user sends a browse request to *youtube:7* and returns the list of videos loaded by the plugin (Listing 3).

To identify the YouTube folder Content Directory, the subsystem combines the plugin attributes *id* and *name*, which is why the result produced for the YouTube registered folder is *youtube:7*. This same idea relates to the YouTube shared items, such as *youtube:hDiLH7jmVsU*, used in the example shown in Listing 3. In this manner, when the Content Directory Subsystem receives a requests for the tuple *youtube:hDiLH7jmVsU*, it splits in two parts at the ":" character, which permits it to identify the plugin and the requested item specified by the control point.

After you finish implementing your plugin, you must create the directory *my_youtube_plugin* under the directory *$PYTHON_DIR/site-packages/brisa/services/cds/plugins*, put your plugin source code in the file *implementation.py*, and save it under the directory you created.

The BRisa media server will load your plugin automatically, and you can now use the UPnP command-line Control Point example or create a more elaborate control point to browse your shared items using the Canola Media Player. Figure 3 shows a list of videos shared by the YouTube plugin.

Conclusion

In this article, I presented the basic concepts of UPnP, a very flexible standard that lets computers, peripherals, appliances, and electronic devices automatically connect and share services. By defining and publishing UPnP device and service descriptions through the BRisa UPnP framework, developers have an open source, but powerful, mechanism to simplify the implementation of devices and services.

In this article, I focused on the UPnP Audio and Video specification, but the BRisa UPnP framework also provides extensible resources to implement other kinds of UPnP services, such as tools for controlling home automation devices.

The BRisa UPnP Project is getting attention from many developers, but, like other open source tools, BRisa is a work in progress. We are currently developing BRisa plugins for some well-known online services, including Yahoo Music, Facebook, Orkut, and PicasaWeb. These plugins will let users centralize all their Internet-based multimedia content in one convenient and flexible UPnP service provided by the BRisa media server. ■

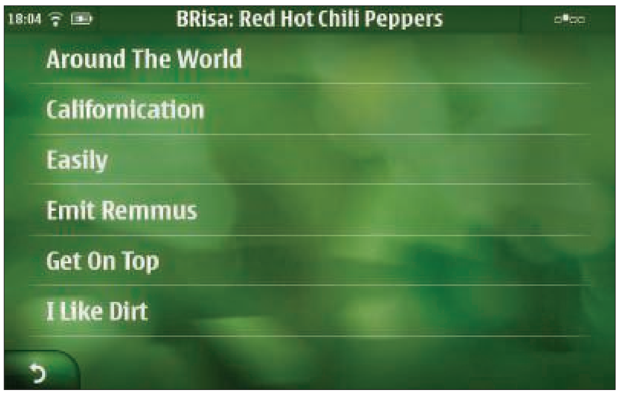

Figure 3: Using the Canola control point to browse the YouTube plugin folders.

INFO

[1] "Pervasive Computing: A Paradigm for the 21st Century" by D. Saha and A. Mukherjee, *Computer*, IEEE Computer Society Press, March 2003, pp. 25-31

[2] UPnP Forum: *http://www.upnp.org/*

[3] "SOAP and Web Services" by P. Louridas, *IEEE Software*, vol. 23, no. 6, December 2006, pp. 62-67

[4] UPnP AV 1.0 specifications: *http://www.upnp.org/standardizeddcps/mediaserver.asp*

[5] "BRisa UPnP A/V Framework" by L. Sales, et al., IEEE International Conference on Consumer Electronics, January 2008, pp. 1-2

[6] "Implementation of the DLNA Proxy System for Sharing Home Media Contents" by J. Kim, et al., *IEEE Transactions on Consumer Electronics*, vol. 53, no. 1, February 2007

[7] BRisa official website: *http://brisa.garage.maemo.org*

[8] BRisa End User Documentation: *http://brisa.garage.maemo.org/documentation-enduser.htm*

[9] IRC Freenode: *irc.freenode.com*, channel *brisa*

[10] BRisa Developer Documentation: *http://brisa.garage.maemo.org/documentation-developer.htm*

[11] "MPEG-21 & DIDL: Dawn of a New Multimedia Eve" by S. Hashemipour and M. Ali, IEEE International Symposium on Consumer Electronics, September 2004, pp. 91-95

THE AUTHOR

Leandro Melo de Sales has enjoyed Linux since 1997 and started the UPnP BRisa project in the end of 2006. He manages the developer group for the BRisa UPnP Framework and its base applications and works at the Embedded Systems and Pervasive Computing Lab/UFCG, which is supported by Nokia Institute of Technology, Brazil. Thanks to the other authors who contributed to this article: Angelo Perkusich, Hyggo Almeida, André Dieb, José Luis, Thiago de Sales, Danilo Freire, and Adrian Livio from Embedded Systems and Pervasive Computing Lab/UFCG; Renato Chencarek and André Magalhães from Nokia Institute of Technology; and Marcello de Sales at the Computer Science department of San Francisco State University (SFSU).