第四周实习报告

5080309684 仇卓

任务概述

1. 修改之前关于 UPnP 设备部分相关代码，完成局域网内 UPnP 设备的发现，信息获取以及设备提供的 Action 的调用。
2. 测试 Prosyst 公司提供的 device management 相关 Bundle。

技术介绍

UPnP

通用即插即用（UPnP）是由通用即插即用论坛（UPnP™ Forum）提出的一套网络协议。该协议的目标是使家庭网络（数据共享、通信和娱乐）和公司网络中的各种设备能够相互无缝连接，并简化相关网络的实现。UPnP 通过定义和发布基于开放、因特网通讯网协议标准的 UPnP 设备控制协议来实现这一目标。UPnP 这个词是从随插即用（Plug-and-play）派生而来的。随插即用是一种热拔插技术。可以将 Plug computer 中的 Bundle 或整个 Plug computer 虚拟为一个 UPnP 设备，提供一些获取自己状态以及执行一些命令的接口供其他 UPnP 设备使用。

Device Manager Bundle

Prosyst 公司提供的 SDK 中有一个 devicem.jar 的 bundle，可以用于设备管理。需要被发现的设备只需实现 org.osgi.service.device.Device 接口即可虚拟为一个 osgi 设备，管理端需要实现 org.osgi.service.device.Device 接口从而虚拟为一个 osgi 设备驱动，若发现合适的设备或有新的合适设备连接时会触发驱动工作。

实现要点

UPnP 设备的发现以及对 UPnP 设备 Action 的调用

UPnP 设备的模拟部分代码与上周相同。

UPnP 设备的发现以及设备 Action 的调用：

```
package upnptest;

import java.util.Dictionary;
import java.util.Enumeration;
import java.util.Hashtable;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
```

```
import org.osgi.framework.Filter;
import org.osgi.framework.ServiceEvent;
import org.osgi.framework.ServiceListener;
import org.osgi.framework.ServiceReference;
import org.osgi.framework.ServiceRegistration;
import org.osgi.service.upnp.UPnPDevice;
import org.osgi.service.upnp.UPnPEventListener;
import org.osgi.service.upnp.UPnPService;

public class Activator implements BundleActivator, UPnPEventListener, ServiceListener {

    private static BundleContext context;
    private ServiceRegistration srr;
    // 用于过滤查找到的设备
    private static String filter = "(|(" + UPnPDevice.UDN + "=*)("
            + UPnPService.ID + "=*))";

    static BundleContext getContext() {
        return context;
    }

    public void start(BundleContext bundleContext) throws Exception {
        Activator.context = bundleContext;
        // 查找 UPnP 设备
        ServiceReference[] dvs = bundleContext.getServiceReferences(
                UPnPDevice.class.getName(),
                "(ObjectClass=" + UPnPDevice.class.getName() + ")");

        if (dvs == null) {
            System.out.println("No UPnP device found");
        } else {
            for (int i = 0; i < dvs.length; i++) {
                System.out.println("Device: " + dvs[i].getProperty(UPnPDevice.UDN));

                // 红色部分代码是对 Action 的调用
                // 获取设备对象(UPnPDevice)
                UPnPDevice dev = (UPnPDevice) context.getService(dvs[i]);
                // 获取设备的所有服务（每个服务可能包含多个 Action）
                UPnPService[] services = dev.getServices();
                // 获取需要的 Action（本例中只有一个 service，其中有一个 Action）
                UPnPAction action = services[0].getActions()[0];
                Hashtable prop = new Hashtable();
                // 设置 Action 调用参数（字典类，key 是参数名称，value 是参数值）
                prop.put(action.getInputArgumentNames()[0], "test");
```

```
                // 传入参数调用 Action（本例没有返回值，若有返回值则为
                // 一个字典对象，key 和 value 分别对应返回值类型和数值）
                action.invoke(prop);
            }
        }

        // 添加监听器，监听设备的连接和断开事件
        bundleContext.addServiceListener(this,
                "(ObjectClass=" + UPnPDevice.class.getName() + ")");

        Hashtable tmp = new Hashtable();
        Filter fi = null;
        try {
            fi = bundleContext.createFilter(filter);
        } catch (Exception e) {
            e.printStackTrace();
        }
        tmp.put(UPNP_FILTER, fi);
        srr = bundleContext.registerService(
                UPnPEventListener.class.getName(), this, null);
    }

    public void stop(BundleContext bundleContext) throws Exception {
        Activator.context = null;
        srr.unregister();
    }

    // 有设备发生变化时被调用
    public void notifyUPnPEvent(String deviceId, String serviceId, Dictionary events) {
        Enumeration en = events.keys();
        System.out.println();
        System.out.println("Event!");
        System.out.println("UDN: " + deviceId);
        System.out.println("ServiceID: " + serviceId);

        while (en.hasMoreElements()) {
            String ssvName = (String) en.nextElement();
            Object value = events.get(ssvName);
            System.out.println("Variable: " + ssvName + " = " + value);
        }
    }

    // 有 service 改变如注册，取消注册等时被调用
    public void serviceChanged(ServiceEvent event) {
```

```
            System.out.println(event);

            if (event.getType() == ServiceEvent.UNREGISTERING) {
                System.out.println(
                        "UNREGISTERING: " +
                        event.getServiceReference().getProperty(UPnPDevice.UDN));
            }
            else if (event.getType() == ServiceEvent.REGISTERED) {
                System.out.println(
                        "REGISTERED: " +
                        event.getServiceReference().getProperty(UPnPDevice.UDN));
            }
            else if (event.getType() == ServiceEvent.MODIFIED) {
                System.out.println(
                        "MODIFIED: " +
                        event.getServiceReference().getProperty(UPnPDevice.UDN));
            }

        }

}
```

对 Prosyst 公司提供的 device management 的 Bundle 进行测试：

Device 接口：

```
package device;

import org.osgi.service.device.Device;

public interface TunerDevice extends Device {

    public void setState(int state);

    public int getState();

}
```

Device 实现：

```
package device.impl;

import java.util.Hashtable;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceRegistration;
```

```
import org.osgi.service.device.Device;

import device.TunerDevice;

public class TunerDeviceImpl implements TunerDevice, BundleActivator {

	ServiceRegistration sReg = null;
	private int state = -1;

	public TunerDeviceImpl() {
		state = 0;
	}

	public void noDriverFound() {
		state = -1;
	}

	public void start(BundleContext context) throws Exception {
		Hashtable deviceProps = new Hashtable();
		deviceProps.put(org.osgi.service.device.Constants.DEVICE_CATEGORY, "tuner");
		deviceProps.put(org.osgi.service.device.Constants.DEVICE_SERIAL, "AB12");
		deviceProps.put(org.osgi.framework.Constants.SERVICE_PID, "my.device.tuner");
		sReg = context.registerService(
				new String[] {
					Device.class.getName(),
					TunerDevice.class.getName()
				},
				this,
				deviceProps);
	}

	public void stop(BundleContext context) throws Exception {
		if (sReg != null) {
			sReg.unregister();
		}
	}

	public void setState(int state) {
		dump("State is set to " + state);
		this.state = state;
	}

	public int getState() {
		dump("State = " + state);
```

```
        return state;
    }

    private void dump(String msg) {
        System.out.println("[TUNER DEVICE] " + msg);
    }

}
```

Driver 的实现：

```
package drivertest;

import java.util.Hashtable;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;
import org.osgi.framework.ServiceRegistration;
import org.osgi.service.device.Constants;
import org.osgi.service.device.Device;
import org.osgi.service.device.Driver;

import device.TunerDevice;

public class TunerDriver implements BundleActivator, Driver {

    private static BundleContext context;
    static final String TUNER_DEVICE_CATEGORY = "tuner";
    static final String TUNER_DRIVER_ID = "my.driver.tuner";
    private ServiceRegistration sReg = null;

    static BundleContext getContext() {
        return context;
    }

    /*
     * (non-Javadoc)
     * @see org.osgi.framework.BundleActivator#start(org.osgi.framework.BundleContext)
     */
    public void start(BundleContext bundleContext) throws Exception {
        TunerDriver.context = bundleContext;
        Hashtable props = new Hashtable();
        props.put(Constants.DRIVER_ID, TUNER_DRIVER_ID);
        sReg = context.registerService(Driver.class.getName(), this, props);
```

```
    }

    /*
     * (non-Javadoc)
     * @see org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
     */
    public void stop(BundleContext bundleContext) throws Exception {
        TunerDriver.context = null;
        if (sReg != null) {
            sReg.unregister();
        }
    }

    public int match(ServiceReference reference) throws Exception {
        if (reference != null) {
            String deviceCategory = (String) reference.getProperty(
                    Constants.DEVICE_CATEGORY);
            if (deviceCategory.equals(TUNER_DEVICE_CATEGORY)) {
                return 1;
            }
        }
        return Device.MATCH_NONE;
    }

    public String attach(ServiceReference reference) throws Exception {
        if (reference != null) {
            TunerDevice device = (TunerDevice) context.getService(reference);
            dump("Initial State = " + device.getState());
            device.setState(5);
        }
        return null;
    }

    private void dump(String msg) {
        System.out.println("[MY TUNER DRIVER] " + msg);
    }

}
```

注意事项和问题

一个 UPnPDevice 设备可以提供多个 Service 来完成不同的功能，而每个 Service 又可以包含多个具体的 Action 供其他设备调用。要调用一个 UPnP 设备的 Action，首先需要获得这个 UPnPDevice 对象。通过将找到的 ServiceReference 强制转型为 UPnPDevice 就可以得到该对象，然后调用 getService()或 getServices()来获取该 UPnPDevice 的 service，再通过 getAction()或 getActions()方法得到具体的 Action，设置好参数后调用该 Action 的 invoke()方法，将参数传递进去。需要注意参数的格式，invoke 的参数只有一个，是一个 Dictionary 对象(可以用 Hashtable 等具体化)，其中 key 是 Action 需要的具体参数名称，value 是对应参数值。若有返回值，格式同输入参数相同，是一个 Dictionary 对象，key 是名称，value 是数值。

对 Prosyst 提供的设备管理相关部分的研究及实验发现，该功能似乎只能在同一个 OSGi Framework 上运行，即其他 framework 中的设备这个 framework 的 driver 是找不到的。资料上说可以发现多种低级通信协议的设备（如 USB,X-10 等），但是没有测试。并且这种设备管理局限性较大，必须使用 prosyst 公司提供的 OSGi Framework 才可以使用，其他设备无法管理，因此决定之后的实验中仍然继续使用 UPnP 设备的管理方式。

参考资料

http://zh.wikipedia.org/wiki/Upnp
UPnP

http://dz.prosyst.com/pdoc/mBS_SH_SDK/runtime/um/upnp/developer/osgi_upnp/osgi_upnp_export.html
UPnP 设备的模拟

http://dz.prosyst.com/pdoc/mBS_SH_SDK/runtime/um/upnp/demos/upnpdemo/upnpdemo.html
UPnP 设备的发现

http://dz.prosyst.com/pdoc/mBS_PE/um/framework/bundles/osgi/devicem/devicem.html
Device Manager Bundle