



Curso Multimedia Home Platform 1.1.2

Gestión de Recursos "Caros"

Reserva y uso de Recursos

Curso Multimedia Home Platform 1.1.2

Copyright 2008 © Enrique Pérez Gil

Licensed under the ***Creative Commons Attribution-Non-Commercial-No Derivative Works 3.0 Unported License***. You may not use this file except in compliance with the License. You may obtain a copy of the License at:

<http://creativecommons.org/licenses/by-nc-nd/3.0/legalcode>

This is a human-readable summary of the License applied:

(<http://creativecommons.org/licenses/by-nc-nd/3.0/>)

You are free to Share, to copy, distribute and transmit the work **Under the following conditions:**

- **Attribution.** You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
- **Noncommercial.** You may not use this work for commercial purposes.
- **No Derivative Works.** You may not alter, transform, or build upon this work.

For any reuse or distribution, you must make clear to others the license terms of this work. Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.

Introducción

- El contexto en el que nos encontramos es limitado y en él conviven varias aplicaciones que han de relacionarse de forma “amigable” para acceder de manera organizada a los recursos caros del sistema.
- Estos son por ejemplo: Modems, Decodificadores MPEG, Screens, Filtros Hardware de SI (los streams)...
- En general los recursos son los **Dispositivos Hardware cuya capacidad de servicio se limita a muy pocos clientes simultáneamente** (cuando no solo a uno!)
- La especificación usada para este propósito en MHP proviene de DAVIC y se encuentra en **org.davic.resources.* DAVIC 1.4.1p9**
- **IMPORTANTE:** es un API de Notificación de uso de Recursos, donde bien a una aplicación se le indica que otra necesita acceso al recurso, o bien el middleware indica que se ha retirado el acceso al mismo. **NO es un Manager de Recursos.**

El API

- Más que un API único usado por los diferentes contextos en los que se necesita, es un **framework** que sirve de base para la implementación de la gestión de notificación de recursos en dichos contextos.
- Existen tres elementos fundamentales:
 - **ResourceServer**: El que se encarga de procesar las peticiones
 - **ResourceClient**: Es el objeto a través del cual el gestor de recursos se va a comunicar con nosotros para decirnos, por ejemplo, que necesita que liberemos el recurso. **Este es el Interface que hemos de implementar. (el Listener vaya!)**
 - **ResourceProxy**: Un objeto donde se almacenan los parámetros de configuración del servicio al que queremos acceso y que además también ofrece apoyo a la gestión de la seguridad. **Es a través de él como se accede al recurso real(proxy), NO se hace directamente**, es decir, nunca obtendremos una referencia directa del objeto que da el servicio, será un **wrapper**.
- Aquel API que desee utilizar este API deberá implementar al menos el **ResourceServer**.

Veamos el API.

org.davic.resources.ResourceServer.

- **Nosotros no accedemos a este API. Es el API final el que debe hacerlo.**
- Se usa para ser notificado de cambios en la situación de los recursos.

```
public interface ResourceServer{  
    public void addResourceStatusEventListener (ResourceStatusListener listener);  
    public void removeResourceStatusEventListener(ResourceStatusListener listener);  
}
```

```
public interface ResourceStatusListener extends java.util.EventListener{  
    public void statusChanged(ResourceStatusEvent event);  
}
```

org.davic.resources.ResourceClient.

- Este es el interface que debemos implementar!!
- Cuando solicitemos la reserva de un recurso (siguiente slide) **normalmente pero NO siempre** se pasará un objeto del tipo **ResourceClient**, y como respuesta obtendremos (o en realidad estaremos trabajando ya con él si la reserva es llamando a un método del objeto que quieres reservar, por ejemplo `HGraphicsDevice.reserveDevice()`) uno que implementa el interface **AppProxy**

```
public interface ResourceProxy{  
    public abstract ResourceClient getClient();  
}
```

- Como vemos este interface simplemente devuelve el usuario actual del recurso caro. Si es que aún lo tiene. Es para uso del framework
- A través del **ResourceClient** pasado el gestor de recursos afectado nos notificará cuando lo considere oportuno que debemos liberar el recurso o bien que ya no lo tenemos!!

Métodos que se usan en MHP para reservar recursos

- Como se observa cada API lo define “a su manera”, pero el resultado es el mismo

Clase	Método
org.havi.ui.HScreenDevice	reserveDevice(ResourceClient client)
org.havi.ui.HVideoDevice	reserveDevice(ResourceClient client)
org.haviui.HGraphicsDevice	reserveDevice(ResourceClient client)
org.havi.ui.HBackgroundDevice	reserveDevice(ResourceClient client)
org.havi.ui.HEmulatedGraphicsDevice	reserveDevice(ResourceClient client)
org.davic.mpeg.sections.SectionFilterGroup	attach(org.davic.mpeg.TransportStream stream, ResourceClient client, Object requestData)
org.dvb.net.rc.ConnectionRCInterface	reserve(ResourceClient c, Object requestData)
org.dvb.event.EventManager	addExclusiveAccessToAWTEvent(ResourceClient client, UserEventRepository userEvents)
org.dvb.event.EventManager	addUserEventListener(UserEventListener listener, ResourceClient client, UserEventRepository userEvents)
org.davic.net.tuning.NetworkInterfaceController	reserve(NetworkInterface ni, Object requestData)
org.davic.net.tuning.NetworkInterfaceController	reserveFor(org.davic.net.Locator locator, Object requestData)
org.davic.net.ca.DescramblerProxy	Constructor: DescramblerProxy(ResourceClient c) startDescrambling()

org.davic.resources.ResourceClient

¿ qué nos puede decir el Resource Manager ?

public boolean **requestRelease**(ResourceProxy proxy, Object requestData);

- Se notifica que el recurso que representa el proxy ha sido solicitado por otra app.
- Si se decide liberarlo, se hace y se devuelve true, si no, false (nos quedamos con el por ahora)
- RequestData se puede usar por las aplicaciones para “dialogar” y decidir quien se queda con el recurso. Deberá ser un **java.rmi.Remote** para que ambas aplicaciones puedan comunicarse: **ver Cap Inter-Xlet Comm.**

public void **release**(ResourceProxy proxy);

- Se notifica que debe liberar el recurso **en este método.**

public void **notifyRelease**(ResourceProxy proxy);

- Se notifica que el proxy ha perdido el acceso al recurso. Ocurre por limitaciones HW o bien por estar Empleando demasiado tiempo en release...o sencillamente porque no se liberó en release.

org.davic.resources.ResourceProxy

- Decíamos que este Interface puede servir para guardar parámetros de conexión....etc.
- Lo que hace el API que vas a usar es definir una clase donde además de establecer los parámetros de configuración hace que sea el ResourceProxy en sí. Suele ser además una clase que es la que se usa como API para reservar el recurso.
 - Así podemos usar varias veces el mismo objeto.
 - Recordemos que el recurso real no lo usamos directamente, el ResourceProxy actúa como un Wrapper.

org.davic.resources.ResourceProxy

- Por ejemplo:

`org.davic.net.tuning.NetworkInterfaceController` **implements ResourceProxy**{

Constructor con el usuario.

`NetworkInterfaceController(ResourceClient rc)`

ResourceProxy implementación. No es para nosotros.

`public ResourceClient getClient()`

Para reservar. Incluso se puede pasar el `java.rmi.Remote`

`public synchronized void reserve(NetworkInterface ni, Object requestData)`

El usado en la reserva. Lo podemos usar de nuevo.

`public NetworkInterface getNetworkInterface()`

Liberamos el recurso

`public synchronized void release()`

org.davic.resources.ResourceProxy

- Otro ejemplo: Los GraphicsDevices que se reservan son Proxies!!!!

```
public class HScreenDevice implements org.davic.resources.ResourceProxy
```

Proceso de solicitud de recursos (cada API lo hará a “su manera”).

1. El ResourceClient, la app, solicita a un API un recurso (seguramente a través de un ResourceProxy). El API se lo comunica a su ResourceServer, pasándole una instancia de ResourceProxy
2. Si se ofrece acceso, el ResourceServer llama a un método privado del Proxy para decirle que OK y que puede usar “este recurso”
3. El ResourceClient llama a métodos Standard del Proxy para manipular el recurso (..). El proxy funciona como una indirección pasando las llamadas al Recurso.
4. Cuando el ResourceClient desea liberar el recurso se llama a un método en el ResourceServer que notifica esa liberación. Este método recibe el Proxy como cliente.
5. El ResourceServer llama a un método privado del Proxy para decirle que ya no es válido y para desasignar el recurso (o decirle que no le use). El Server actualiza sus tablas de recursos para marcarlo como free.

¿ y en función de qué se da un recurso a una app o a otra ?

1. Se debe tener permiso para acceder a un recurso, aunque esté libre. Las Aplicaciones NO Firmadas (ya lo veremos) tienen acceso a muy pocos recursos.
2. Cuando hay que quitar un recurso a alguna app se hace empezando por las de menor prioridad hacia las de mayor.
3. Si una aplicación solicita un recurso y este lo tiene una app de menor prioridad se le quitará. De igual forma a una app con menor prioridad que solicite un recurso se le denegará

- Como mejor se ve y asimila este API es gestionando recursos Caros. Cada API lo hace de una forma, pero el funcionamiento es el mismo.
- Lo importante es que seamos conscientes de que los recursos del STB han de compartirse, y de que hemos de gestionar este hecho de forma correcta para que nuestras aplicaciones (y las de otros) funcionen sin problemas.

ISO/IEC 13818-1	Part 1. Elementary Streams transport definition
ISO/IEC 13818-6	Part 6. Extensions for DSM-CC. Digital Storage Media Command and Control
ETSI EN 300 468	Digital Video Broadcasting (DVB);Specification for Service Information (SI) in DVB systems
ETSI EN 301 192	DVB specification for data broadcasting
ETSI TR 101 202	Implementation Guidelines for Data broadcasting
ETSI TR 101 162	Digital broadcasting systems for television, sound and data services; Allocation of Service Information (SI) codes for Digital Video Broadcasting (DVB) systems
ETSI TR 102 154	Implementation guidelines for the use of MPEG-2 Systems, Video and Audio in Contribution and Primary Dist
ETSI TR 101 211	Guidelines on implementation and usage of Service Information (SI)
ETSI TR 101 200	Digital Video Broadcasting (DVB); A guideline for the use of DVB specifications and standards
DAVIC	Digital Audio Visual Council. davic 1.4.1
HAVI	Specification of the Home Audio/Video Interoperability (HAVi) Architecture
Interactivetvweb	http://www.interactivetvweb.org/
Wikipedia DSMCC	http://en.wikipedia.org/wiki/DSM-CC
MHP 1.1.2	Multimedia Home Platform, A068r1 & tam668r23_11xdraft_20061115
MHP 1.1.3	Multimedia Home Platform, A068r3
CDC 1.1	Connected Device Configuration (CDC) 1.1 (JSR=218).
PBP 1.1	Personal Basis Profile 1.1 (JSR 217)
MHP.org	www.mhp.org
INTRO MHP 1.1.3	tam1032r1-mhp-iptv-presentation