



# Curso Multimedia Home Platform 1.1.2

## MHP Graphics I. Devices & Scenes

Arquitectura gráfica

Configurando Devices

Configurando Scenes

## 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.

## Qué es HAVI en MHP

- HAVI: Specification of the Home Audio/Video Interoperability (HAVi) Architecture,
- En MHP 1.1.2 specs A0068r1:

HAVi 1.1 This comprises the following documents:

- HAVi v1.1 Chapter 8 15May 2001 (***Level 2 User Interface***)
- HAVi v1.1 Java L2 APIs, 15May2001 (*ídem al anterior*)
- HAVi v1.1 Chapter 7,15May2001 (***compatibilidades a nivel de bytecode, Havi IDL2Java...***)

[www.havi.org](http://www.havi.org)

Lo que nos interesa: **Chapter 8 – Level 2 User Interface**

## Por qué HAVI

- Estamos en entorno distinto al PC, para el cual se diseñó la java.awt, de manera que DVB, siguiendo su línea habitual, decidió adoptar un standard ya pensado para funcionar en “Consumer Devices” como la TV: HAVI, y más en concreto, el subconjunto definido como “Level 2 User Interface”
- Este API se diseñó para mostrar Streaming y Audio en “Consumer Devices”, pero lo que sobre todo nos interesa: también permite la construcción de aplicaciones gráficas en Java a partir de un subconjunto (eso sí, muy pequeño) de la AWT

## ¿ Contra qué nos “enfrentamos” ? Veamos algunas de las complejidades

- Entornos de muy bajo coste y potencia
- Pixel Aspect Ratios
  - Pixels cuadrados en PCs vs Pixels no-cuadrados en TV
  - Diferentes resoluciones con los mismos Aspect Ratios
  - A veces ocurre que el controlador de vídeo y el de gráficos no vierten sus contenidos hasta que se van a pintar en la pantalla, de manera que puede resultar muy complicado hacer que los pixels de los graphics “encajen” con los del vídeo
- Cambios de Aspect Ratio
  - Las aplicaciones deben verse correctamente en 4:3 y 16:9, pero es que además el aspect ratio puede variar dinámicamente, bien por la TV en sí o por elección del usuario. Una imagen diseñada para 4:3 puede verse “ensanchada” en 16:9

## ¿ Contra qué nos “enfrentamos” ? Veamos algunas de las complejidades (y 2)

- Transparency
  - Necesitamos transparencia para poder ver el vídeo detrás de los gráficos, y no podemos tener una plataforma Java 2, donde sí hay soporte para Transparency.
- Color Spaces
  - El sistema de colores RGB no mapea directamente con el sistema YUV de la TV
- No hay Window Manager
  - Modelo distinto para proporcionar recursos gráficos a las aplicaciones
  - Las aplicaciones han de coexistir de forma distinta
- No hay Mouse
  - Sólo se puede navegar con **el teclado del control remoto de la TV**
- La baja resolución de las TV y las restricciones para interactuar con las aplicaciones implica que el UI debe diseñarse de forma completamente diferente.

## ¿ En qué consisten las piezas ?

- Estos componentes se pueden dividir en 4 categorías:
  - **Clases para manipular y modelar** los diferentes elementos involucrados en el “**Display Stack**”: capas gráficas físicas del STB. (¿ Hardware ?)
  - Un conjunto de **componentes gráficos de usuario** para la TV: **Widget UI**
  - **Clases para gestionar la compartición de recursos** entre las aplicaciones (recordemos: entorno muy limitado)
  - **Extensiones para soportar los inputs** desde un control remoto.
- Es decir
  - Hemos de gestionar la naturaleza de los elementos físicos de presentación.
  - Hay que compartir la pantalla
  - Hay que controlar el input de un mando a distancia
  - Y finalmente hemos de crear componentes gráficos básicos con una nueva librería.

OK. Veámoslo.

## Display Stack

- El Stack gráfico MHP se compone de 3 capas

### Graphics

- MHP ofrece al usuario la capa de Graphics donde este puede “pintar” sus aplicaciones.
- Una aplicación **podrá presentar video dentro de un componente gráfico especial...**

### Video

- En la capa de Vídeo se muestra la señal de video (☺)
- Una aplicación puede controlar donde presentarla y con qué tamaño usando **JMF controls...**
- Puede cubrir toda la pantalla con lo que no se verá el background

### Background

- En la capa de Background se mostrará un color o una imagen...o un videodrip!
- Una aplicación puede definir qué presentar en la capa de Background...

## Display Stack

- La composición de las capas se hace de atrás hacia delante: Background, Video y por último Graphics





## Display Stack ¿ Puede haber más de una instancia de cada capa ?

- En realidad de **HBackgroundDevice** sólo puede haber 1.
- De **HVideoDevice** podría haber varias si existiera soporte de “Picture in Picture” (PIP), o dicho de otro modo, **si el STB tuviera más de un decodificador MPEG.**
- De **HGraphicsDevice** podría haber más de una instancia, si se soportase más de una capa de Graphics, de nuevo tiene que ver sobre todo con las posibilidades gráficas del STB (no es lo normal).



**Lo normal es que exista sólo una instancia/capa de cada tipo. Programar contemplando más posibilidades es complicarse la vida**

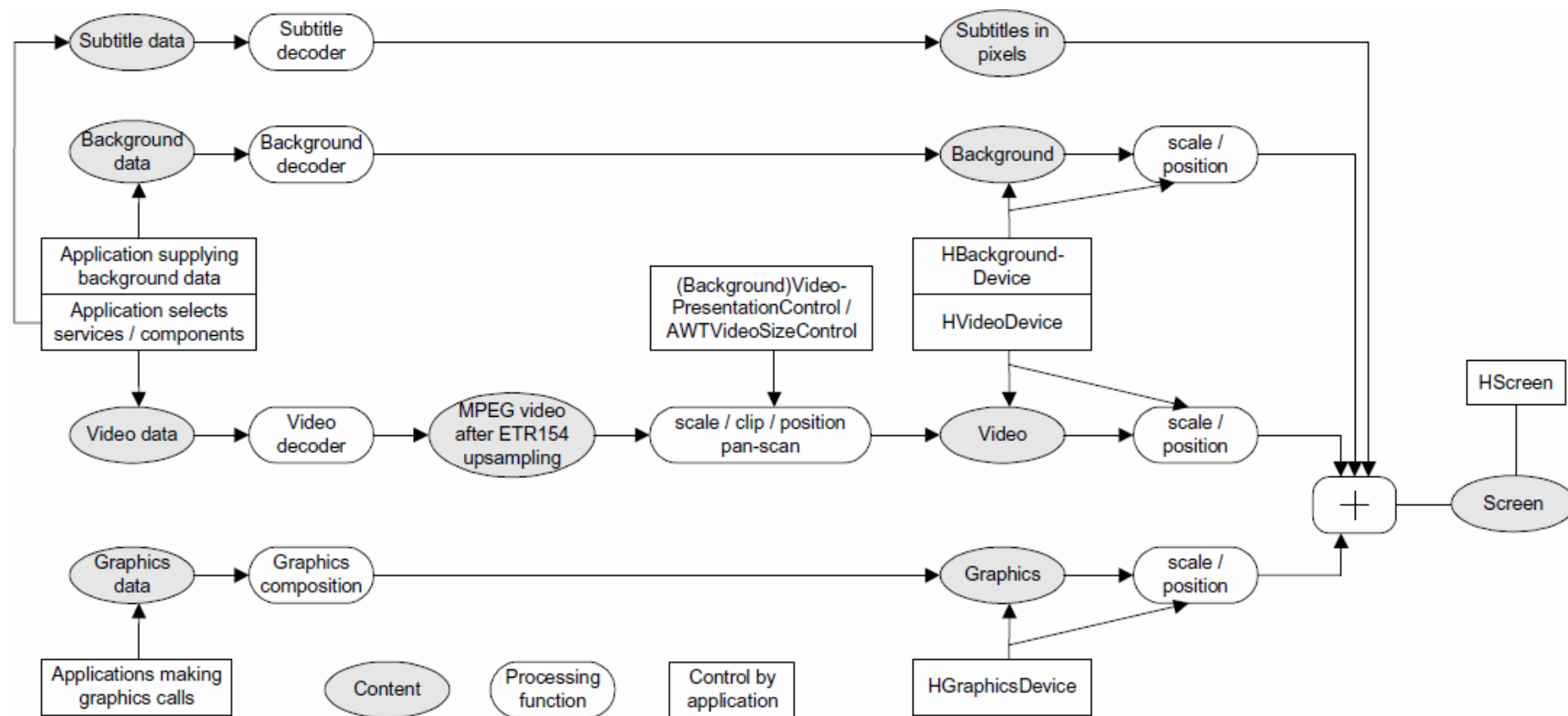
## Display Stack

- MHP permite que los terminales ejecuten más de una aplicación a la vez compartiendo la pantalla teniendo cada una una sub-área del screen en la que pueda pintar.
- MHP permite incluso que las áreas de las aplicaciones se solapen (raro), no obstante hay que decir que si esto sucede las aplicaciones se “cliparán”, es decir, si la que está delante es transparente **NO se verá la que está detrás, sino la capa de vídeo...**

Veamos en detalle la arquitectura y cómo podemos configurar sus piezas

## Display Stack. Arquitectura MHP

- Es importante tener claro qué es lo que ocurre dentro del STB desde el punto de vista de bloques funcionales para saber qué puedo y qué no puedo hacer y porqué



## Display Stack

- Como podemos observar hay **cuatro fuentes** de contenido que se funden en un único screen al final, lo que llamaremos la salida de vídeo final: **subtítulos, background, video y Graphics.**
- También observamos que la aplicación tiene capacidades “teóricas” para modificar la posición y tamaño de los distintos tipos de contenido. Decimos teóricas porque siempre será el STB el que permita o no hacer esas transformaciones.
- **OJO:** los (**scale/position**) son elementos tipo “**Processing Function**” al igual que los **Video Decoders y Background Decoders**, por lo tanto **NO SON ELEMENTOS SW de MHP sino que los ofrecerá (o NO) el STB y en grados.** Por eso veremos que cuando queramos configurar la salida del contenido mediante los Devices atacando a los “**Processing Function**” habrá cosas que podremos hacer y muchas que no.

QUEDAOS CON LA FOTO DE CÓMO VA EL TEMA!

## Display Stack

- La salida de video final, es decir, **donde se van a volcar los 4 contenidos**, se representa por lo que llamamos **org.havi.ui.HScreen**, que es la suma del output de
  - Uno o más graphics decoders
  - Uno o más video decoders
  - Un background decoder
- La representación de estos decoders son lo que denominamos **Devices**: **org.havi.ui.HBackgroundDevice**, **org.havi.ui.HVideoDevice** y **org.havi.ui.HGraphicsDevice**
- **Como hemos visto en el gráfico de la arquitectura estos atacan al contenido ya procesado y a los “Processing Function” de Scale/Position justo antes de fundirse en la HScreen.**
- Cada uno de estos Devices se podrá configurar para generar una salida que nos interese, lo que ocurre es que las posibilidades son muy restrictivas. En primer lugar porque depende de las capacidades HW del deco **y en segundo porque se han de compartir entre las diferentes aplicaciones en ejecución.**

- Dado que existen 3 capas que se pintan en una final, veamos en qué sistemas de coordenadas lo hacen.

## Coordenadas Normalizadas: HScreen

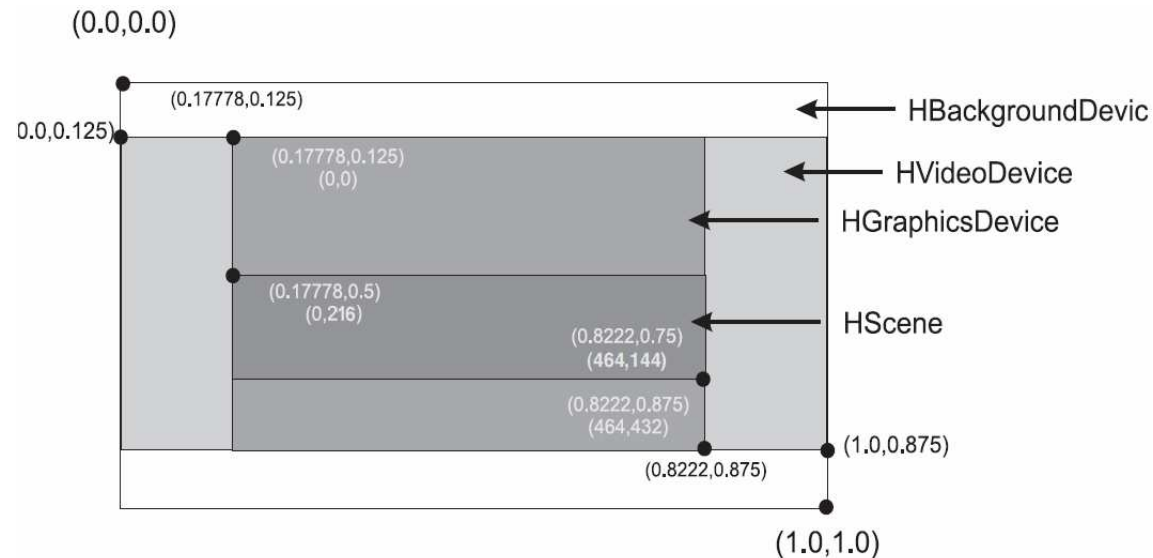
- **El esquema de coordenadas Normalizadas** ofrece la posibilidad de establecer posiciones y tamaños con respecto al **Video de Salida del terminal MHP sin tener en cuenta** para nada en qué resolución (w,h) se está tratando esta salida (hablaremos más adelante de la resolución)
- El esquema se representa como (0,0): esquina superior izquierda y (1,1): esquina inferior derecha.
- El componente que usa este esquema de coordenadas y que como ya hemos dicho representa la salida final de video que ve el usuario es **org.havi.ui.HScreen**
- En teoría cada una de las capas las podríamos posicionar/dimensionar con respecto a la salida de Video final usando este esquema de coordenadas sin embargo en la práctica hay importantes restricciones.
- **Por defecto todas las capas son full-screen: ocupan lo mismo que la salida final de video (luego lo vemos)**

## Hablando de Graphics. Coordenadas en Pixels y ¿ Normalizadas ?

- Una vez que he colocado/dimensionado mi “lienzo” `HGraphicsDevice` en coordenadas normalizadas, si es que he querido modificarlo, recordemos que por defecto su tamaño coincide con el full-screen, estoy en condiciones de crear las `RootWindow` de las aplicaciones en relación a este contexto gráfico.
- Sin embargo en el contexto de Gráficos MHP **no existen `RootWindows`**, sino **`org.havi.ui.HScenes`: paneles que se colocan en el “lienzo” `HGraphicsDevice`.**
- A pesar de que las **`HScene`** se pintan en el **`HGraphicsDevice`**, se puede definir su localización y tamaño usando coordenadas **normalizadas**. ¿ Por qué ? Es útil porque es muy normal que la capa de Graphics sea full-screen, es decir, coincida con el video de salida final, así por un lado tenemos libertad total respecto a la colocación de nuestra “`RootWindow`”, o mejor dicho `HScene`, y por otro tenemos “estabilidad” pues nos inhibimos de las complejidades de los “píxeles” y las resoluciones.
- Una vez que he situado el **`HScene`** en la capa gráfica los componentes que se añaden a la `HScene` usarán **coordenadas en pixels**.

## Coordenadas

- Ejemplo de configuración de las capas



MHP 1.1.2 A068r1

- El HBackgroundDevice **es full-screen**: (0,0)-(1,1)
- El HVideoDevice es (0,0.125)-(1,0.875)
- El HGraphicsDevice es (0.17778,0.125)-(0.8222,0.875) y tiene una resolución en píxeles de 464x432
- La HScene está definida en píxeles con respecto a su “capa gráfica” como: (0,216)-(464, 144)
- La HScene está definida en normalizadas con respecto a la screen como: (0.17778,0.5)-(0.8222,0.75)
- **En realidad las posibilidades de configuración son limitadas, existiendo pocas combinaciones posibles**

## Resolución

- Hemos visto que las señales de Background, Video y Graphics se fundirán en una salida de Video final, pero ¿ **con qué resolución** ? Veamos algunos conceptos de TV Digital.
  - **Standard-definition television, SDTV**: se refiere a sistemas de transmisión de TV que cumplen con los estándares mínimos, esto es, no son “enhanced” ni “HD”. Normalmente se aplica en el contexto de la TV digital cuando esta se transmite con una **resolución** similar a la analógica.
  - **Digital SDTV** con un **aspect ratio** de 4:3 tiene la misma apariencia que una TV analógica (NTSC, PAL, PAL2, SECAM) quitando las desventajas de esta última : ghosting (imagen doble), nieve, ruido...
  - Standards que pueden hacer broadcast de digital SDTV son **DVB(Europa)**, **ATSC(USA)** y **ISDB (Japan)**. Los dos últimos se diseñaron para HD pero se usan más las posibilidades de multi-streaming de audio/video que la alta definición (más opciones de contenido que poco contenido con gran resolución).
  - **Es decir, la señal digital de video que se transmite lo hace a una determinada resolución**. En las tablas siguientes se describen las referidas para SDTV.

## Resolución

- Resoluciones de SDTV con sus Aspect Ratios

Resolución (w,h)	Nombre	Pixel aspect ratio (W:H)		Usado en...
		4:3	16:9	
720×576	<b>576i</b>	16:15	64:45	D1/DV PAL
704×576	<b>576p</b>	12:11	16:11	EDTV PAL
720×480	<b>480i</b>	8:9	32:27	DV NTSC
720×486	<b>480i</b>	8:9	32:27	D1 NTSC (ITU-R 601)
704×480	<b>480p</b>	10:11	40:33	EDTV NTSC

## Resolución

- **La resolución de recepción del vídeo puede ser la misma que tendrán las capas de Graphics y de Background** por defecto. Hablaremos más adelante al respecto en el apartado **“Requerimientos”**
- Puede haber limitaciones HW en cuanto a las posibilidades de presentación en, por ejemplo, HD, en cuyo caso la resolución se verá restringida para todos.
- Lo importante es que siempre será posible en el sistema conocer las configuraciones iniciales, y analizar las diferentes posibilidades de configuración, de manera que podremos definir nuestro UI como convenga.

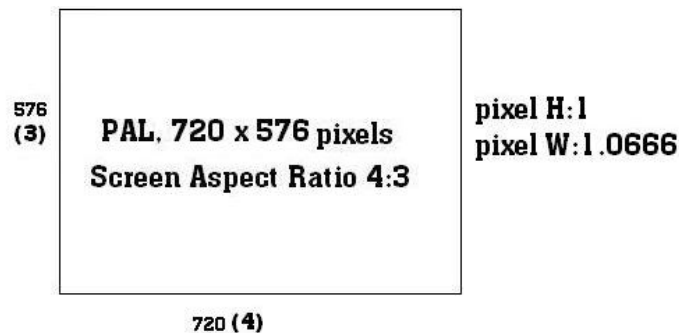
## Aspect Ratio & Pixel Aspect Ratio

- Como hemos visto además de llegar el video codificado a una determinada resolución también lo hace en una determinada configuración de visión: 4:3, 16:9...
- Este dato es lo que se denomina **Aspect Ratio**, que viene a significar algo así como: en el caso de 16:9, que si dividimos la base de la TV en 16 partes, el alto mide 9 de esas partes.
- Dependiendo de la **resolución** de la señal y del **aspect ratio** de la imagen tendremos un valor que se llama **Pixel Aspect Ratio: relación del ancho del pixel con su alto**.
- Estos 3 parámetros están relacionados: **resolución, image aspect ratio y pixel aspect ratio**, de manera que sus combinaciones no son “libres”

Pixel Aspect Ratio	Aspect Ratio	
	4:3 Display	16:9 Display
Resolución para full screen HGraphicsDevice		
640 x 480	1:1	4:3
720 x 480	8:9	32:27
720 x 576	16:15	64:45
768 x 576	1:1	48:36
854 x 480	3:4	1:1
1024 x 576	36:48	1:1

## Aspect Ratio & Pixel Aspect Ratio

- Los cambios de Aspect Ratio para el Texto puede ser un problema. Como se ve en el gráfico la diferencia de 4:3 a 16:9 en sus diferentes Pixel Aspect Ratios es importante.
- Una posible solución es usar **14:9** siempre, así el texto tendrá siempre el mismo aspecto.
- Veamos abajo en detalle de donde salen los números del Pixel Aspect Ratio.



**720 pixels en 4 partes y 576 en 3**

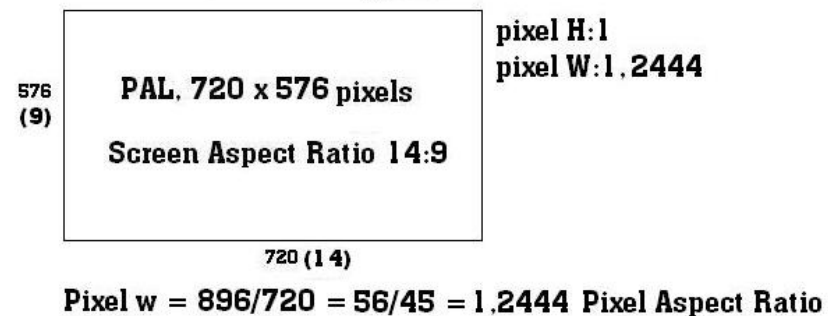
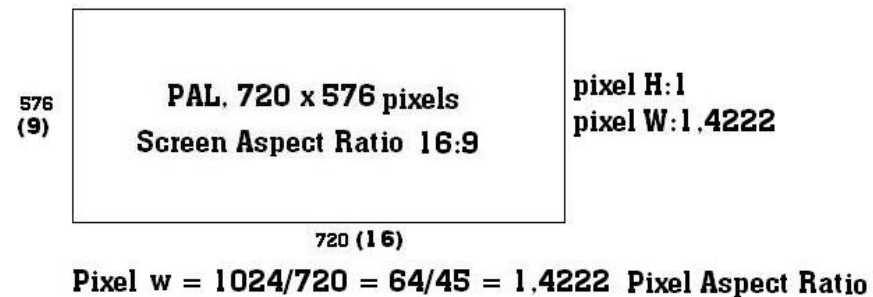
Si en 3 partes hay 576 en 4 habría 768  
si midieran lo mismo de alto que de ancho  
pero en 4 hay 720 , es decir,

hemos metido 768 "alturas" en 720

"anchos", con el lo que cada ancho

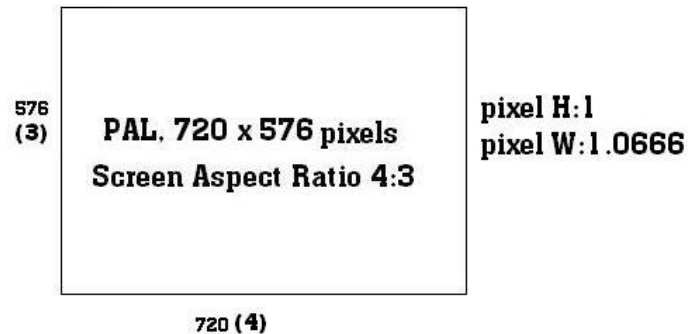
mide:  $768/720$  "alturas", o lo que

es lo mismo=  $48/45 = 1,06666$  Pixel Aspect Ratio



## Aspect Ratio & Pixel Aspect Ratio

- Los Receptores MHP **están obligados** a ofrecer **HGraphicsDevices** que soporten un Screen Aspect Ratio de **14:9**. En 13.3.7 dice "All HGraphicsDevices must be HEmulatedGraphicsDevices capable of emulating a14:9...".
- Esta "solución" tiene sus inconvenientes si quieres por ejemplo alinear video y gráficos, pero si tu UI es sobre todo texto puede ir bastante bien.



720 pixels en 4 partes y 576 en 3

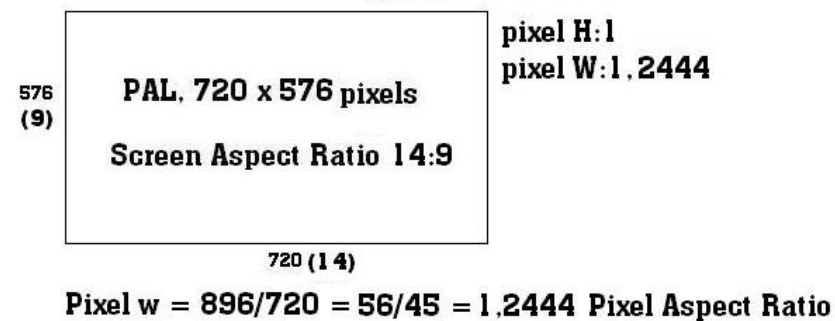
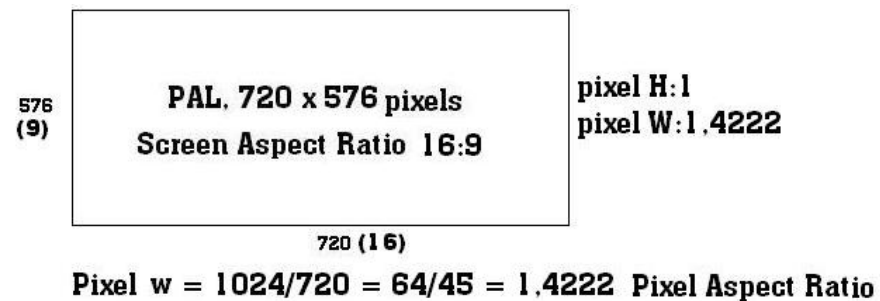
Si en 3 partes hay 576 en 4 habría 768  
si midieran lo mismo de alto que de ancho  
pero en 4 hay 720 , es decir,

hemos metido 768 "alturas" en 720

"anchos", con el lo que cada ancho

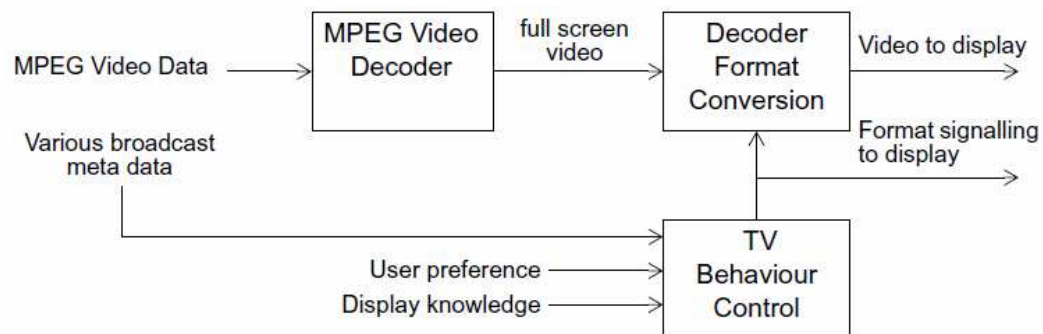
mide:  $768/720$  "alturas", o lo que

es lo mismo =  $48/45 = 1,06666$  Pixel Aspect Ratio



## Hablando de Video

- Como vemos en la figura la señal que llega sale del Decoder como **Full Screen**:
  - “In this version of the MHP specification, the video decoding process complies with clause 5.1.4 “Luminance resolution” of TR 101 154 [9] and applies up sampling for a defined set of luminance resolutions so that the decoded pictures **are displayed at full-screen size**. “.
- Full-screen significa que se podrá mostrar en el total de la salida de video, para lo cual el decoder hará el “upsampling” que fuera oportuno para conseguir los “tamaños” requeridos. Ved 5.1.4, ETSI TR 102 154 V1.1.1)
- En la siguiente slide vemos esta información.



## Hablando de Video

- **Full-Screen.**

5.1.4, TR 102 154 V1.1.1

- La señal de video llega codificada con unas resoluciones y Aspect Ratio determinadas. **Esta señal hay que procesarla para poder mostrarla en RESOLUCIÓN Full-Screen**

Table 4: Resolutions for Full-screen Display from contribution IRD

Coded Picture		Displayed Picture Horizontal upsampling	
Luminance resolution (horizontal x vertical)	Aspect Ratio	4:3 Monitors	16:9 Monitors
720 x 576	4:3	× 1	× 3/4 (see note 1)
	16:9	× 4/3 (see note 2)	× 1
	2,21:1	× 5/3 (see note 3)	× 5/4 (see note 4)
544 x 576	4:3	× 4/3	× 1 (see note 1)
	16:9	× 16/9 (see note 2)	× 4/3
	2,21:1	× 20/9 (see note 3)	× 5/3 (see note 4)
480 x 576	4:3	× 3/2	× 9/8 (see note 1)
	16:9	× 2 (see note 2)	× 3/2
	2,21:1	× 5/2 (see note 3)	× 15/8 (see note 4)
352 x 576	4:3	× 2	× 3/2 (see note 1)
	16:9	× 8/3 (see note 2)	× 2
	2,21:1	× 10/3 (see note 3)	× 5/2 (see note 4)
352 x 288	4:3	× 2	× 3/2 (see note 1)
	16:9	× 8/3 (see note 2)	× 2
	2,21:1	× 10/3 (see note 3)	× 5/2 (see note 4)
		(and vertical upsampling × 2)	(and vertical upsampling × 2)

NOTE 1: Upsampling of 4:3 pictures for display on a 16:9 monitor is optional in the contribution IRD, as 16:9 monitors can be switched to operate in 4:3 mode.

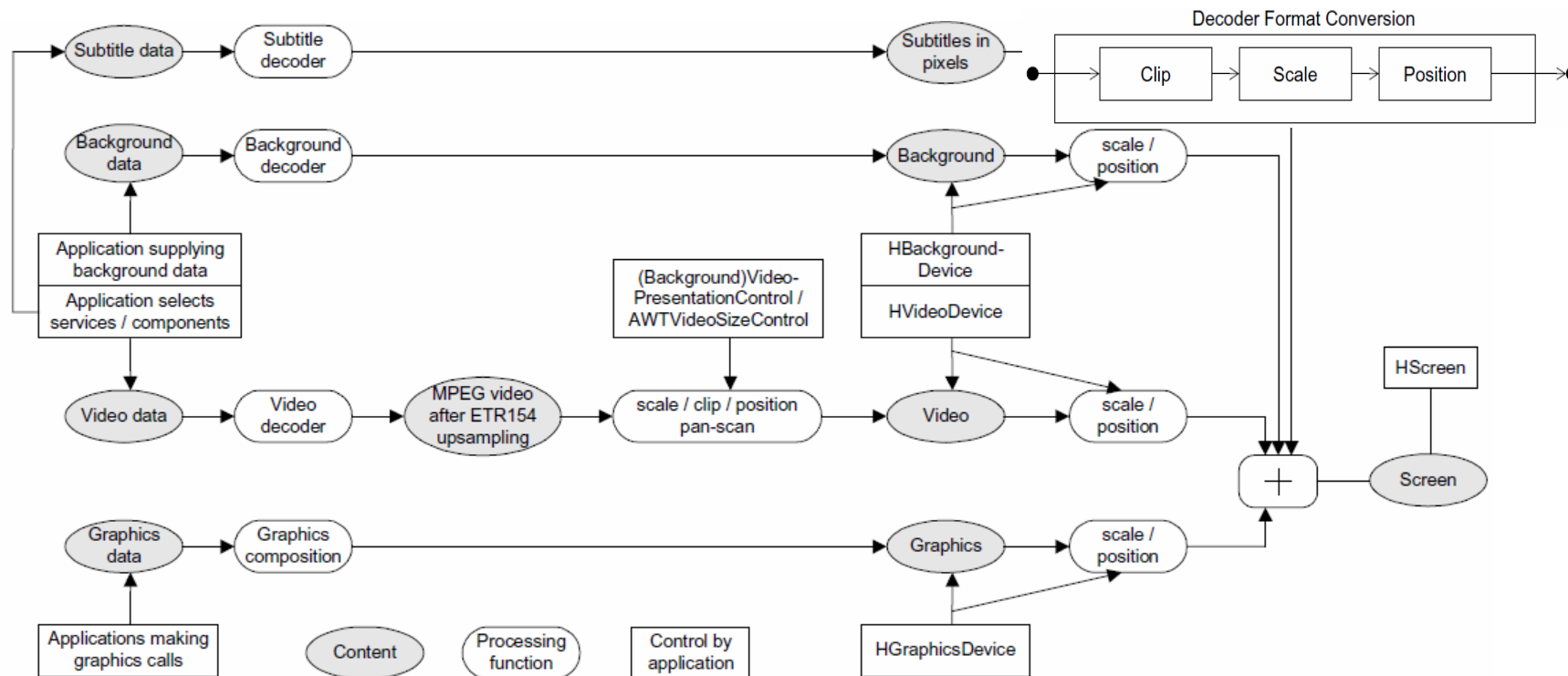
NOTE 2: The upsampling with this value is applied to the pixels of the 16:9 picture to be displayed on a 4:3 monitor.

NOTE 3: The upsampling with this value is applied to the pixels of the 2,21:1 picture to be displayed on a 4:3 monitor. Upsampling from 2,21:1 pictures for display on a 4:3 monitor is optional in the contribution IRD.

NOTE 4: The upsampling with this value is applied to the pixels of the 2,21:1 picture to be displayed on a 16:9 monitor. Upsampling from 2,21:1 pictures for display on a 16:9 monitor is optional in the contribution IRD.

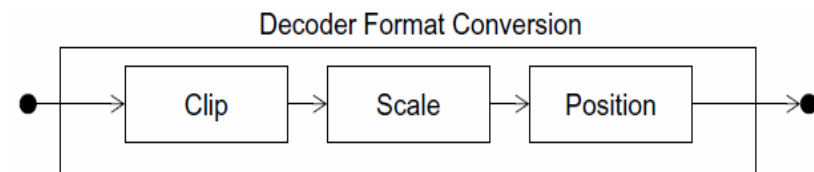
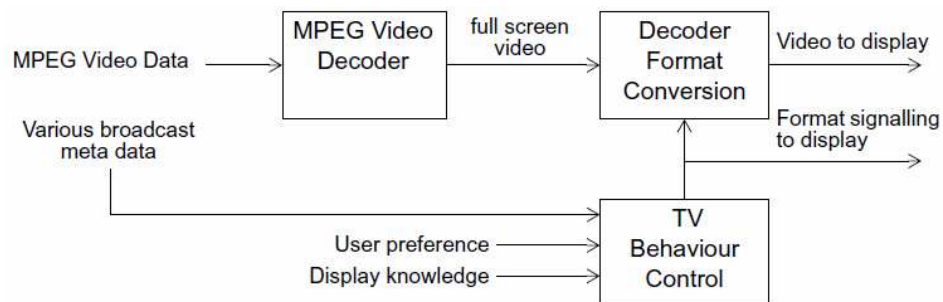
## Hablando de Video

- Una vez que se dispone de la señal en resolución FULL SCREEN, es en el “Decoder Format Conversion” donde se procesa la misma para la salida final teniendo en cuenta el aspect ratio, pan/scan, además del conocimiento que se tenga del formato del Display (4:3, 16:9) y su resolución, y por supuesto de las preferencias del usuario, para generar la señal que va al Display Final (recordemos la foto de la arquitectura)



## Hablando de Video

- El “Decoder Format Conversion” ejecuta en orden las acciones siguientes sobre el full-screen video: clipping, scaling y positioning.
- **Sin embargo este módulo no es “intocable”**: una aplicación podrá averiguar qué posibilidades existen en cada paso del “Decoder Format..” a través de **org.dvb.media.VideoPresentationControl** e incluso podrá establecer estos parámetros de forma atómica usando **org.dvb.media.VideoTransformation**. Por supuesto también podrá saber qué transformaciones se están aplicando en este momento.... (veremos todo esto en su capítulo correspondiente...)



## Hablando de Graphics

- **MHP 1.1.2 A.7.7.1:** “The **java.awt.Toolkit.getScreenSize** method shall be equivalent to the pixel resolution of the current configuration of the default screen device returned by **HScreen.getDefaultHGraphicsDevice**”,  
  
es decir, nos ofrece las dimensiones del Screen originales!!!
- .... y las coordenadas provistas por **java.awt.Component.getLocationOnScreen()** están en ese esquema de coordenadas.

## Requerimientos (MHP 1.1.2 Annex G)

- El aspect ratio por defecto del Graphics Device será **14:9** (ojo que alguna otra aplicación puede haberlo cambiado)
- El STB implementará al menos un HGraphicsDevice que será **full-screen** (0,0,1,1)
- El STB implementará al menos un HBackgroundDevice. Estos son siempre **full-screen**.
- El STB implementará al menos un HVideoDevice que siempre se podrá configurar como **full-screen**.
- El STB implementará al menos un HScreen que soportará al menos un video, graphics y background device como los definidos arriba.
- **Todo apunta a que por defecto los Devices tienen la misma resolución que la señal de vídeo full-screen de entrada (asumiendo las limitaciones del STB claro)**

## **Requerimientos por defecto respecto a Graphics (MHP 1.1.2 Annex G)**

- Los STB MHP de mercados de 625 líneas/50Hz soportarán simultáneamente cualquier combinación de HVideoConfiguration, HBackgroundConfiguration y HGraphicsConfiguration/HEmulatedGraphicsConfiguration descrita en la Tabla **G.1** siempre que tengan un **aspect ratio de salida común**.
- Los STB MHP de mercados de 525 líneas/60Hz soportarán simultáneamente cualquier combinación de HVideoConfiguration, HBackgroundConfiguration y HGraphicsConfiguration/HEmulatedGraphicsConfiguration descrita en la Tabla **G.2** siempre que tengan un **aspect ratio de salida común**.
- Los STB MHP que soporten HD soportarán adicionalmente de forma simultánea cualquier combinación de HVideoConfiguration, HBackgroundConfiguration y HGraphicsConfiguration descritas en las tablas **G.3** y **G.4**.

Echadle un ojo al Anexo G de las MHP 1.1.2 specs A0068r1

## Más requerimientos

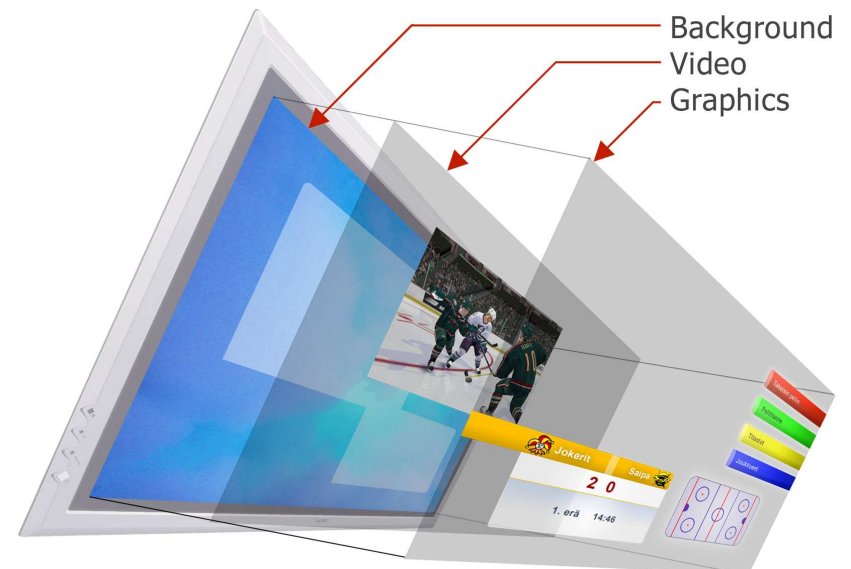
- **MHP establece un mínimo de resolución de 720 x 576 pixels y 256 colores para el Graphics layer**, pero puede ser mayor dependiendo del receptor
- La plataforma MHP garantiza las siguientes resoluciones para las capas:
  - Background: 720x576
  - Video: 720x576
  - Graphics: 720x576
- La capa Gráfica sólo está obligada a soportar pixels no-cuadrados. O dicho de otra forma: no está obligada a soportar pixels cuadrados.
- Opcionalmente la capa Gráfica podría soportar píxeles cuadrados en resoluciones de 768x576 y 4:3 Displays y en resoluciones de 1024x576 en 16:9 Displays.
- Los STBs están obligados a soportar un EmulatedGraphicsDevice con un aspect ratio de 14:9 para poder ofrecer un output razonable en Displays de 4:3 o 16:9

## ¿ Cómo se gestionan las diferentes capas ?

Veamos las clases de Havi que modelan las piezas involucradas en el Display

### HScreen: org.havi.ui.HScreen

- Representa un **Display físico**, es la clase que muestra las 3 capas unidas.
- Si un STB tuviera la capacidad de generar dos señales de TV de dos canales distintos que fuesen a dos TV diferentes entonces habría 2 instancias de HScreen, cada una encargada de una señal.



## HScreen: org.havi.ui.HScreen

- Una HScreen contiene al menos una instancia de cada tipo siguiente que representa cada una de las capas (todas heredan de org.havi.ui.HScreenDevice)

org.havi.ui.**HBackgroundDevice**

org.havi.ui.**HVideoDevice**

org.havi.ui.**HGraphicsDevice**

- Y una interesante: org.havi.ui.**HEmulatedGraphicsDevice**: es una capa lógica que representa una emulación de una configuración gráfica determinada.

## ¿ cómo defino la configuración de las diferentes capas?

- Lo que se hace es plantearle al sistema nuestras necesidades y él nos responderá con la instancia configurada de la forma más cercana posible a nuestras aspiraciones.
- Esto significa que la arquitectura interna del STB asume esos parámetros y se comporta de acuerdo a ellos.

## Para entendernos

- Estamos solicitando que elementos compartidos con otras aplicaciones (y con la nuestra propia) se configuren de determinada forma!!! Estas configuraciones han de ser compatibles para todos
- **NO es “MI” capa de video, ni “MI” capa de Graphics, ni “MI” capa de “Background”, es “La” capa de vídeo, “La” capa de Graphics y “La” capa de “Background”**

- ¿ Qué parámetros contienen esas configuraciones, donde están definidos y a qué Devices puedo aplicarlos ?

Parameter	Background	Graphics	Video
HScreenConfigTemplate.PIXEL_ASPECT_RATIO	OK	OK	OK
HScreenConfigTemplate.PIXEL_RESOLUTION	OK	OK	OK
HScreenConfigTemplate.INTERLACED_DISPLAY	OK	OK	OK
HScreenConfigTemplate.FLICKER_FILTERING	OK	OK	OK
HScreenConfigTemplate.VIDEO_GRAPHICS_PIXEL_ALIGNED		OK	OK
HScreenConfigTemplate.SCREEN_RECTANGLE	OK	OK	OK
HScreenConfigTemplate.ZERO_BACKGROUND_IMPACT	OK	OK	OK
HScreenConfigTemplate.ZERO_GRAPHICS_IMPACT	OK	OK	OK
HScreenConfigTemplate.ZERO_VIDEO_IMPACT	OK	OK	OK
HBackgroundConfigTemplate.CHANGEABLE_SINGLE_COLOR	OK		
HBackgroundConfigTemplate.STILL_IMAGE	OK		
HGraphicsConfigTemplate.IMAGE_SCALING_SUPPORT		OK	
HGraphicsConfigTemplate.MATTE_SUPPORT		OK	
HGraphicsConfigTemplate.VIDEO_MIXING		OK	
HVideoConfigTemplate.GRAPHICS_MIXING			OK

- Los parámetros de configuración se solicitan indicando un nivel de exigencia en su cumplimiento:
  - org.havi.ui.HScreenConfigTemplate.**REQUIRED** = 1: ha de tenerse, es imprescindible
  - org.havi.ui.HScreenConfigTemplate.**PREFERRED** = 2; nos interesaría tenerlo pero no es imprescindible
  - org.havi.ui.HScreenConfigTemplate.**DONT\_CARE** = 3; no nos importa lo que tenga
  - org.havi.ui.HScreenConfigTemplate.**PREFERRED\_NOT** = 4; no nos gustaría tenerlo pero no es imprescindible que no se dé
  - org.havi.ui.HScreenConfigTemplate.**REQUIRED\_NOT** = 5; no ha de darse, es imprescindible que NO se de
- Si el sistema no consigue cumplir con las **REQUIRED** y **REQUIRED\_NOT** devolverá null -> NO es posible configurar el Device con los requerimientos **pasados...rebaja tus pretensiones!!**

- El sistema, una vez tenga cumplidas las anteriores exigencias intentará satisfacer al máximo las restantes: que se den las más posibles de PREFERRED y las menos posibles de PREFERRED\_NOT

Veamos qué significan estos parámetros

## Descripción de los parámetros. Integración de Gráficos y Vídeo

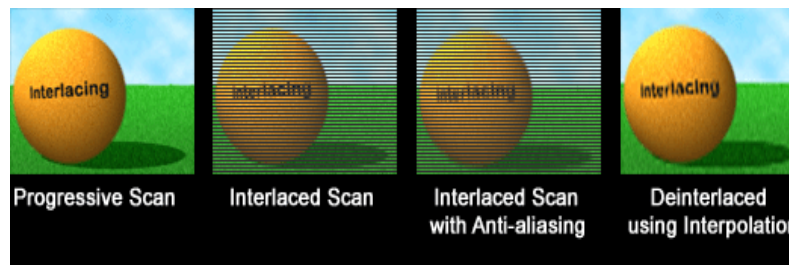
- Como veremos más adelante, HAVI nos permite establecer los requerimientos (Templates) de Graphics, Video y Background, y a partir del método `HScreen.getCoherentScreenConfigurations(Template[])` obtener configuraciones coherentes con esos requisitos.
- Además es posible establecer restricciones entre **vídeo/graphics** que pueden usarse para aplicaciones que ya tienen Video mostrándose y este ha de cuadrar con una configuración de Graphics, o bien para otras que ya estén funcionando con la capa de Graphics y esta debe cuadrar con una capa de Video.
- `HGraphicsConfigTemplate.VIDEO_MIXING [G]`
  - La capa de graphics va a soportar algún tipo de Transparencia de manera que se podrá ver el vídeo de la capa de detrás y no hay ningún tipo de requerimiento en cuanto al alineamiento de pixels de ambas capas
  - Se puede utilizar como parámetro una configuración de Video: `org.havi.ui.HVideoConfiguration`

## Descripción de los parámetros. Integración de Gráficos y Vídeo

- HVideoConfigTemplate.**GRAPHICS\_MIXING** [V]
  - Cuando aplicado a Video: asume que se van a mostrar gráficos en su Device y no hay ningún tipo de requerimiento en cuanto a alineación de Pixels
  - Se puede utilizar como parámetro una configuración de Graphics:  
org.havi.ui.HGraphicsConfiguration

## Descripción de los parámetros

- HScreenConfigTemplate.**PIXEL\_ASPECT\_RATIO [B,G,V]**
  - mediante un objeto Dimension se indica el Pixel Aspect Ratio deseado para el device. p.e. [width=64,height=45], ya lo hemos visto en la explicación de Pixel Aspect Ratio: “estoy metiendo 64 alturas en 45 espacios”.
- HScreenConfigTemplate.**INTERLACED\_DISPLAY [B,G,V]**
  - El Device habrá de soportar un Display Entrelazado
- HScreenConfigTemplate.**FLICKER\_FILTERING [B,G,V]**
  - El device soportará el flicker filtering para evitar el efecto en entrelazado



[http://en.wikipedia.org/wiki/Progressive\\_scan](http://en.wikipedia.org/wiki/Progressive_scan)

## Descripción de los parámetros

- HBackgroundConfigTemplate.**CHANGEABLE\_SINGLE\_COLOR [B]**
  - Se solicita un Background que soportará un color de fondo que se va a poder cambiar por las aplicaciones
- HBackgroundConfigTemplate.**STILL\_IMAGE [B]**
  - Si deseamos tener una imagen como Background. **Esta debe especificarse en el Template.** El Configuration object que se nos devolverá será del tipo **HStillImageBackgroundConfiguration** el cual ofrece dos métodos para establecer la imagen, la cual se define mediante la clase: HBackgroundImage (lo vemos después)
- HGraphicsConfigTemplate.**IMAGE\_SCALING\_SUPPORT [G]**
  - La capa de Graphics deberá soportar escalado rápido (hardware) de imágenes

## Descripción de los parámetros

- HScreenConfigTemplate.**SCREEN\_RECTANGLE [B,G,V]**
  - Se le está pidiendo al Device que se presente en el “cuadrado” definido por el área HScreenRectangle: coordenadas normalizadas referidas al Screen area
  - Si estas coordenadas son [0,0,1,1] pues sencillamente se presentará en todo el Screen (full screen, por defecto) pero si no es así, se producirá un escalado para ajustarse a esa “ventana”. Si el sistema no puede escalar no se obtendrá el Configuration deseado.
- HScreenConfigTemplate.**PIXEL\_RESOLUTION [B,G,H]**
  - Resolución del Device descrita con un objeto Dimension (WxH). Esta Resolución se entiende se corresponde con el área especificada en **SCREEN\_RECTANGLE**
- HGraphicsConfigTemplate.**MATTE\_SUPPORT [G]**
  - La capa de Graphics deberá tener soporte de Matte (ah! Pero es opcional ? Mal asunto....)

## Descripción de los parámetros

- HScreenConfigTemplate.**ZERO\_BACKGROUND\_IMPACT [B,G,V]**
  - Se desea que la configuración de la capa NO afecte en absoluto a las de background de las otras aplicaciones incluyendo la mía.
- HScreenConfigTemplate.**ZERO\_GRAPHICS\_IMPACT [B,G,V]**
  - Se desea que la configuración de la capa NO afecte en absoluto a la de Graphics de las otras aplicaciones incluyendo la mía.
- HScreenConfigTemplate.**ZERO\_VIDEO\_IMPACT [B,G,V]**
  - Se desea que la configuración de la capa NO afecte en absoluto a la de vídeo de las otras aplicaciones incluyendo la mía.

## Descripción de los parámetros

- HScreenConfigTemplate.**VIDEO\_GRAPHICS\_PIXEL\_ALIGNED [G,V]**
  - Se desea que la configuración de píxeles de VIDEO y GRAPHICS estén alineados con **píxeles del mismo tamaño**: así, la superposición de Graphics sobre Video será perfecta.
  - Ojo: la alineación de los orígenes de los dos sistemas de coordenadas no es un requerimiento
  - Puesto que la capa de video se mueve en tiempo real (pan & scan) la de Graphics ha de ser capaz de controlar dichos cambios para alinearse; si no puede no será posible la configuración.
  - El objeto a pasar como preference Object en el caso de Graphics será un HVideoConfiguration y viceversa.

- Antes de entrar en cómo es el proceso de configuración en detalle intentemos simplemente saber con qué configuración trabajaríamos sin “solicitar nada”

## Descubriendo la configuración

- Disponemos de tres tipos de clases para cada Device (hacemos el ejemplo con HGraphicsDevice, el resto es idéntico cambiando el nombre de la capa: HVideoDevice y HBackgroundDevice)
  - **HGraphicsDevice**: el Device en sí.
  - **HGraphicsConfiguration**: Objeto que contiene una configuración de Device. Es el que un Device va a usar para configurarse.
  - **HGraphicsConfigTemplate**: Objeto repositorio de parámetros que definen una Configuración. Se usa para solicitar configuraciones y para saber los parámetros de una determinada configuración.
    - En algún caso es posible establecer datos de configuración sin necesidad del Template, p.e.:  
`HBackgroundConfiguration.setColor`

## Descubriendo la configuración

- Para saber cómo es la configuración de un Device hemos de:
  - 1º Obtener el HScreen
  - 2º Obtener el H[Graphics]Device
  - 3º Obtener su H[Graphics]Configuration actual
  - 4º Obtener el H[Graphics]ConfigTemplate a partir de H[Graphics]Configuration

[Graphics] es sustituible por [Video] y [Background]

- Los objetos H[Graphics]Configuration pueden ofrecer cierta información que no está en el Template, p.e.: HGraphicsConfiguration.getFonts();

## Descubriendo la configuración

- **Paso 1.** Accediendo al HScreen y sus Devices. Con los métodos en negrita accedemos primero a los posibles HScreen (**static**) y después a las diferentes instancias de Devices de HScreen.

```
public class HScreen{
```

```
    public static HScreen[]           getHScreens()  
    public static HScreen           getDefaultHScreen()
```

```
    public HVideoDevice[]           getHVideoDevices()  
    public HGraphicsDevice[]        getHGraphicsDevices()
```

```
    public HVideoDevice             getDefaultHVideoDevice()  
    public HGraphicsDevice          getDefaultHGraphicsDevice()  
    public HBackgroundDevice        getDefaultHBackgroundDevice()
```

```
    public HScreenConfiguration[]    getCoherentScreenConfigurations(HScreenConfigTemplate[] hscta)  
    public boolean                   setCoherentScreenConfigurations(HScreenConfiguration[] hsca)  
    public HVideoConfiguration       getBestConfiguration(HVideoConfigTemplate[] hvcta)  
    public HGraphicsConfiguration    getBestConfiguration(HGraphicsConfigTemplate[] hgcta)  
    public HBackgroundConfiguration  getBestConfiguration(HBackgroundConfigTemplate[] hbcta)
```

```
}
```

## Descubriendo la configuración

- **Paso 2.** Obteniendo los Configuration. Para obtener el Configuration cada Device dispone de algunos métodos que nos dan H[Graphics]Configuration
  - El método que nos da la **configuración actual** es:

```
HGraphicsConfiguration hgdc = device.getCurrentConfiguration();
```
  - Un método que nos daría una **configuración “por defecto”**, que no significa que la tenga, sino que sería una genérica que se podría aplicar en la mayoría de las ocasiones es:

```
HGraphicsConfiguration hgdc = device.getDefaultConfiguration();
```
  - Un método que nos ofrece **TODAS las posibles configuraciones** que puede soportar es:

```
HGraphicsConfiguration[] hgdc = device.getConfigurations();
```

## Descubriendo la configuración

- **Paso 3.** Para obtener el template se llama al método getConfigTemplate del objeto Configuration, así:
  - HGraphicsConfigTemplate = graphicsConfiguration.getConfigTemplate()
  - HBackgroundConfigTemplate = backgroundConfiguration.getConfigTemplate()
  - HVideoConfigTemplate = videoconfiguration.getConfigTemplate()

Todos heredan de HScreenConfigTemplate

## Descubriendo la configuración

- **Paso 4.** Para obtener la información de los parámetros de configuración empleamos los siguientes 2 métodos disponibles en H[Graphics]ConfigTemplate:
  - Object getPreferenceObject(*PARAMETER*): obtenemos el valor del parámetro (si aplica)
  - int getPreferencePriority(*PARAMETER*); obtenemos su nivel de Exigencia

## **Ejercicios Bloque G1**

## 2 Ejemplos de Situación

### Caso 1

- Supongamos que una aplicación ha especificado (ZERO\_GRAPHICS\_IMPACT, REQUIRED) a su configuración **de Graphics (pixel ratio y resolución)**, lo que significa que no quiere que esta afecte **a ninguna configuración de Graphics de nadie** (incluida la suya).
- Si nuestra aplicación establece (VIDEO\_GRAPHICS\_PIXEL\_ALIGNED, REQUIRED) **a su configuración de Graphics** el STB sólo puede satisfacer ambas si cambia la configuración del Video para encajar con el Graphics Device, pues este no puede modificarse al existir una configuración de Graphics previa que requiere no ser afectada; si el Video no soporta la resolución y pixel ratio del Graphics no nos podrá dar una configuración.

### • Caso 2

- Si se solicita una configuración de Background con ZERO\_VIDEO\_IMPACT, entonces , si esta configuración varía en algo con respecto al video sólo se obtendrá un Template si el decoder usado para HBackgroundDevice es distinto del usado para video.

## Proceso de Configuración. Esquema General

- **Paso 1:** Defino mis pretensiones

```
HGraphicsConfigTemplate hgraphicsDeviceConfTemplate = new HGraphicsConfigTemplate();  
hgraphicsDeviceConfTemplate.setPreference(  
    HGraphicsConfigurationTemplate.ZERO_VIDEO_IMPACT,  
    HGraphicsConfigurationTemplate.REQUIRED);
```

## Proceso de Configuración. Esquema General

- **Paso 2:** Solicito al Device una posible configuración que satisfaga mis pretensiones al máximo: método **device.getBestConfiguration**

```
// Obtengo el HScreen
```

```
HScreen hs = HScreen.getDefaultHScreen();
```

```
// Obtengo el Device
```

```
HGraphicsDevice hgraphicsDevice = HScreen.getDefaultHGraphicsDevice();
```

```
// Le pido un objeto Configuration que satisfaga mis pretensiones
```

```
HGraphicsConfiguration hgraphicsConfiguration =  
    hgraphicsDevice.getBestConfiguration(hgraphicsDeviceConfTemplate);
```

- Lo que hace **getBestConfiguration(...)** es ofrecer un objeto **Configuration** con las propiedades tales que satisfagan los requerimientos establecidos. Si para establecer esa configuración es necesario cambiar la configuración de algún otro device y NO puede hacerlo, entonces devolverá **HConfigurationException**.
- **getBestConfiguration NO te garantiza que la Configuration que te ha dado podrá aplicarla**, te garantiza que ese objeto satisface tus necesidades especificadas teniendo en cuenta las configuraciones actuales en curso y las limitaciones del STB.

## Proceso de Configuración. Esquema General

- **Paso 3:** establezco la configuración al Device si la respuesta es distinta de null

```
if (hgraphicsConfiguration!=null){  
    // reservamos el recurso....ya lo veremos  
    hgraphicsDevice.reserveDevice((org.davic.resources.ResourceClient)this);  
    // establecemos la configuración!!!  
    hgraphicsDevice.setGraphicsConfiguration(hgraphicsConfiguration);  
    // liberamos el recurso  
    hgraphicsDevice.releaseDevice();  
}
```

- El método setGraphicsConfiguration puede lanzar una serie de Exceptions, por ejemplo SecurityException si el operador de red no permite la modificación del Device (propietario)
- Si no hemos reservado el Device o bien si el cambio en la configuración de este puede afectar a la configuración de otro que no hemos reservado, también lanzará una Exception, del tipo HPermissionDeniedException
- Igualmente si al final no es posible establecer la configuración deseada, se puede lanzar una HConfigurationException

## Estableciendo la Configuración usando HScreen

- Es posible establecer la configuración a través de los métodos siguientes de HScreen, los cuales te permiten manejar conjuntos (arrays) de Templates para obtener aquel que satisface de la mejor manera posible a todos ellos.

```
public HScreenConfiguration[] getCoherentScreenConfigurations(HScreenConfigTemplate[] hscta)
```

- Devuelve una Configuration por cada Template pasado como input. Todas las Devueltas son coherentes entre sí. El input deberá ser de un Template por Device (para no liarnos)

```
public boolean setCoherentScreenConfigurations(HScreenConfiguration[] hsca)
```

- Establece las Configuration pasadas a los Devices correspondientes siempre que entre las mismas exista coherencia.

```
public HVideoConfiguration getBestConfiguration(HVideoConfigTemplate[] hvcta)
```

```
public HGraphicsConfiguration getBestConfiguration(HGraphicsConfigTemplate[] hgcta)
```

```
public HBackgroundConfiguration getBestConfiguration(HBackgroundConfigTemplate[] hbcta)
```

- Ofrecen la mejor configuración a partir de un set de Templates todos del mismo tipo

## Reserva de Recursos

- Los Devices son recursos caros que **habremos de tener reservados cuando vayamos a realizar su configuración.**
- De esta forma evitamos que en mitad de nuestro proceso de establecimiento de propiedades alguien más nos modifique el contexto.
- **Ojo:** hay que liberarlos una vez efectuada la configuración!!!

## Cambiando de Espacios de Coordenadas

- Tenemos 3 espacios:
  - HScreen, Normalizadas ((0,1), (0,1)) para colocar HScenes en referencia al Screen sin preocuparnos por los pixels que mida.
  - HScene (pixel): para colocar componentes
  - HComponent (pixel): para pintar elementos dentro del componente.
- En las normalizadas, para colocar HScenes en relación a HScreen, se usan:
  - HScreenPoint
  - HScreenDimension
  - HScreenRectangle
- Se llaman coordenadas virtuales a las medidas en pixels con respecto a HScreen: de raro uso pero se ve de vez en cuando.

## ¿ Cómo transformar coords de un modelo Normalizado al resto y viceversa ?

- Coordenadas Normalizadas de un Component

HGraphicsConfiguration

```
public HScreenRectangle getComponentHScreenRectangle(Component component)
```

- Coordenadas Normalizadas de un “Rectangle en Pixels” referido a una HScene

HScene

```
public HScreenRectangle getPixelCoordinatesHScreenRectangle(Rectangle r)
```

## ¿ Cómo transformar coords. de un modelo Normalizado al resto y viceversa ?

- Coordenadas AWT en pixels relativas a un Container a partir de unas normalizadas HScreenRectangle, es decir, las HScreenRectangle corresponderían a un componente dentro del Container

HGraphicsConfiguration

```
public Rectangle getPixelCoordinatesHScreenRectangle(HScreenRectangle sr, Container cont)
```

- Es decir, que si SR son las coordenadas normalizadas que queremos para **compA**, lo siguiente es coherente:

```
// obtenemos las coordenadas en pixels dentro del contenedor correspondientes a las normalizadas  
Rectangle r = getPixelCoordinatesHScreenRectangle(SR, cont);  
// añademos el componente  
cont.add(compA);  
// le establecemos sus coordenadas en pixels.  
compA.setBounds(r.x, r.y, r.width, r.height);
```

## **Ejercicios Bloque G2**

## ¿ Cómo saber que una configuración de un Device ha cambiado ?

- Necesitamos Listeners de cambios de configuraciones!!! En general, una modificación de un HScreenDevice puede requerir que el UI se modifique, por ejemplo si la resolución o el pixel aspect ratio han cambiado
- **Los tenemos:** HScreenDevice dispone de los **2** métodos siguientes:

```
public void addScreenConfigurationListener(HScreenConfigurationListener hscl)
```

Notifica cuando la configuración del Device ha cambiado

```
public void addScreenConfigurationListener(HScreenConfigurationListener hscl,  
HScreenConfigTemplate hsct)
```

Notifica cuando la configuración del Device ha cambiado y **YA NO ES COMPATIBLE CON EL TEMPLATE**

```
public interface HScreenConfigurationListener extends java.util.EventListener {  
    public void report(HScreenConfigurationEvent gce);  
}
```

## ¿ **Cómo saber que una configuración de un Device ha cambiado ?**

- Así pues: la configuración se puede re-adaptar sólo cuando recibimos eventos de este tipo.
- Hemos de suscribirnos en cada Device.

## **Ejercicios Bloque G3**

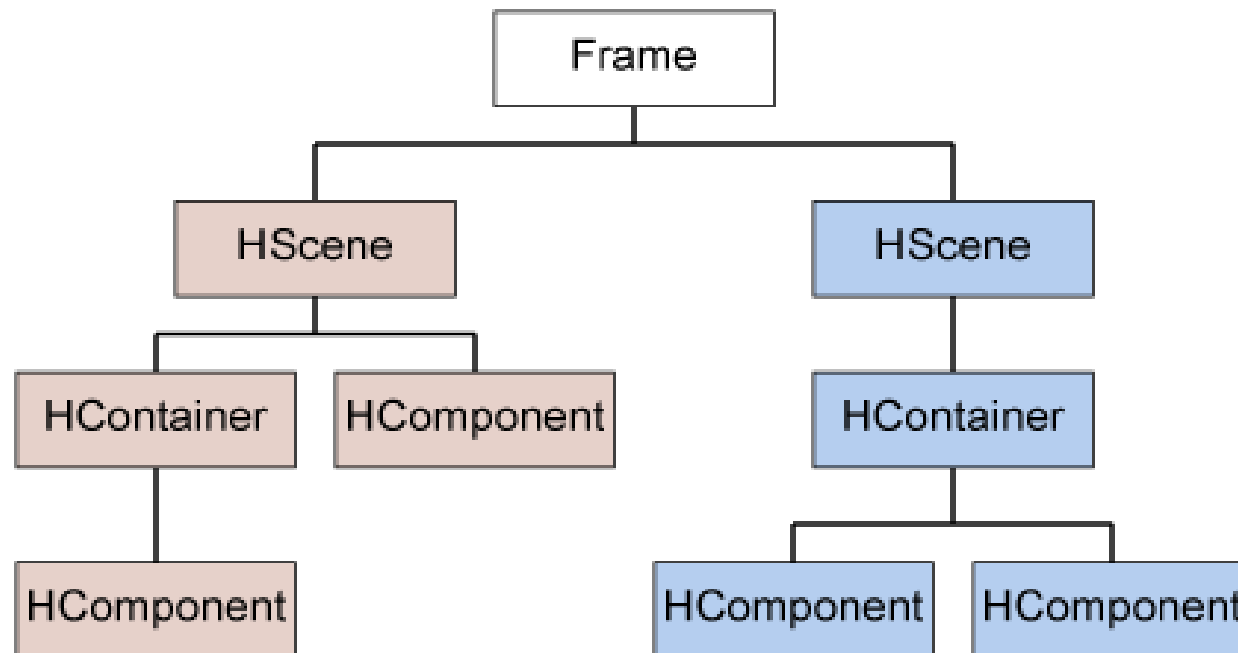
- Una vez que sabemos como configurar los devices de nuestra pantalla, veamos cómo crear componentes gráficos: nuestro UI sobre ella.
- Existen no obstante, “algunas” diferencias con respecto a un UI “clásico”:
  - No tenemos Frames
  - No existe un Sistema de Ventanas con el que podamos “comunicarnos”
  - Las aplicaciones habrán de compartir el área gráfica, y lo normal es que no puedan “solaparse”
  - Nuestro TOP-LEVEL Container se llama **HScene**

## HScene

- Es como un Container dentro del cual podremos añadir nuestros componentes.
- SOLO UNO por aplicación
- Como un HScene es un container de alto nivel está oculto por defecto.
- Cuando se destruya el Xlet, MUY IMPORTANTE: HScene.dispose()!!!
- Las HScene de las aplicaciones deben coexistir
- HAVI es capaz de manejar las aplicaciones simultáneas con: Ventanas, Tabbed Panes o Cuadrículas, o sencillamente permitiendo a únicamente 1 aplicación ser visible.

## Jerarquía Gráfica

- El padre de HScene NO es accesible



## ¿ Como se crean las HScenes ?

- Siguen el mismo esquema de Templates que los Devices, con parámetros, aunque disponemos de métodos que nos dan algunas por defecto.
- Elemento principal: **HSceneFactory**. Nos ofrecerá los métodos para obtener los mejores Templates y las HScenes (además de otros de interés). Veámoslo primero.

## HSceneFactory API

```
public static HSceneFactory getInstance()
```

- acceso a la Factory (singleton)

```
public HSceneTemplate getBestSceneTemplate(HSceneTemplate hst)
```

- Mejor Template para el Template deseado

```
public HScene getBestScene(HSceneTemplate hst)
```

- Mejor HScene para el template deseado(**cuidado: puede haber conflictos con HScreen**)

## HSceneFactory API

```
public HSceneTemplate resizeScene(HScene hs, HSceneTemplate hst)
```

- Queremos cambiar el tamaño/posición del HScene usando la info del template.

### Los 3 métodos siguientes nos ofrecen una HScene configurada por el sistema

```
public HScene getDefaultHScene(HScreen screen)
```

- Dame un HScene por defecto para mi HScreen (para cuando hay varias HScreen)

```
public HScene getDefaultHScene()
```

- Dame una HScene por defecto (= anterior cuando sólo hay una HScreen)

```
public HScene getFullScreenScene(HGraphicsDevice device)
```

- Queremos una HScene **full-screen** que ocupe todo el área de Graphics pasada.

```
public void dispose(HScene scene)
```

- Solicitamos y notificamos la destrucción de una HScene. Podría ocurrir que creásemos otra más adelante..

## Obteniendo HScenes con un Template

- ¿ Cuales son los parámetros del Template/HScene ? ¿ cómo se establecen ?
- Usamos **HSceneTemplate** para indicar nuestras preferencias, las cuales se establecen con un **nivel de exigencia**:

HSceneTemplate.**REQUIRED**: debe de otorgarse la propiedad obligatoriamente

HSceneTemplate.**PREFERRED**: *nos interesaría que se diera.*

HSceneTemplate.**UNNECESSARY**: *no es necesaria*

Veamos ahora qué parámetros podemos establecer

## Parámetros de HSceneTemplate

- HSceneTemplate.**GRAPHICS\_CONFIGURATION**
  - **Para cuando hay más de un HGraphicsDevice:** para que se construya con una HGraphicsConfiguration específica. Por defecto se contruyen con la Configuration actual del HGraphicsDevice.
- HSceneTemplate.**LARGEST\_PIXEL\_DIMENSION**
  - **No es un parámetro sino un valor** que debemos usar para establecer el parámetro **SCENE\_PIXEL\_DIMENSION** (a continuación) con el mayor tamaño posible

## Parámetros de HSceneTemplate

- HSceneTemplate.**SCENE\_PIXEL\_LOCATION**
  - Para establecer la **posición** en PIXELS. Con objeto java.awt.**Point**
- HSceneTemplate.**SCENE\_PIXEL\_DIMENSION**
  - Para establecer el tamaño deseado en PIXELS. Con objeto java.awt.**Dimension**.
- HSceneTemplate.**SCENE\_SCREEN\_LOCATION**
  - **Posición** del HScene en HScreen Normalized Dimensions!!! (0,0-1,1). Con **HScreenPoint**
- HSceneTemplate.**SCENE\_SCREEN\_DIMENSION**
  - **Tamaño** del HScene en HScreen Normalized Dimensions!!! (0,0-1,1). Con **HScreenDimension**

## Estableciendo los Parámetros de HSceneTemplate

- Para establecer los parámetros hemos de usar los métodos:

```
public void setPreference(int preference, Object object, int priority)
```

```
public Object getPreferenceObject(int preference)
```

```
public int getPreferencePriority(int preference)
```

## Me pueden cambiar la configuración de mi HScene!!

- Dado que una HScene puede ser movida, oculta, presentada, o cambiada de tamaño por el Program Manager, por ejemplo, para que aparezca al lado de otra, podemos suscribirnos al siguiente evento para enterarnos:

```
scene.addComponentListener(java.awt.ComponentListener)
```

```
ComponentListener{  
    public void componentResized(ComponentEvent e)  
    public void componentMoved(ComponentEvent e)  
    public void componentShown(ComponentEvent e)  
    public void componentHidden(ComponentEvent e)  
}
```

**OJO AL componentShown:** nos indica un momento en los que podemos estar seguros que la HScene se ha mostrado; es posible que sea buen momento para hacer visibles nuestros componentes gráficos....

## Recomendaciones/Varios respecto a HScenes

- Conviene no establecer las preferencias de LOCATION Y TAMAÑO de vuestra HScene como required, dado que de esa forma, si ya está presente y el middle necesita redimensionarla no podrá y la ocultará; y si no está presente puede que no os permita crearla
- Las HScenes se crean ocultas: hay que llamar a **setVisible(...)**
- El propósito de una HScene es la de establecer el área en la que se mostrarán los componentes realmente “visibles” de la aplicación.
- Se dice que una HScene tiene el foco sólo si uno de sus componentes hijos lo tiene
- Una HScene nos permite establecer su área como Transparente, que es lo típico, o bien podemos establecerle un fondo: color o imagen

**OverScan (muy resumidamente)**, <http://en.wikipedia.org/wiki/Overscan>

- Debido a que hace unos años en la emisión se incluía información por fuera de los bordes del contenido, los terminales de tipo CRT hacían/hacen (casi todos, aunque no se puede asegurar nada) lo que se llama OverScan, es decir, proyectan la imagen sobre-dimensionada para asegurarse de que no se vea dicha información.
- Por esto no podemos asegurar que :
  - una aplicación se vea correctamente en el punto 0,0
  - Y tampoco que el texto no se vea borroso debido a ese “estiramiento” en una determinada franja.
- Para evitar ambos problemas se establece lo que se denominan “**Zonas de Seguridad**”, las cuales, dependiendo del **Screen Ratio** pueden variar en cierta medida. **No existen ni porcentajes fijos ni acordados en la industria.** Las dos zonas son:
  - **Action Safe**: que la aplicación se vea
  - **Title Safe**: que los textos se vean correctamente y no borrosos.

**OverScan (muy resumidamente)**, <http://en.wikipedia.org/wiki/Overscan>

- Tabla con una recomendación de márgenes por cada lado:

Screen Ratio	Action Safe		Title Safe	
	Vertical	Horizontal	Vertical	Horizontal
<b>4:3</b>	3,5%	3,3%	5%	6,7%
<b>16:9</b>	3,5%	3,5%	5%	10%
<b>14:9 en 16:9</b>	3,5%	10%	5%	15%
<b>4:3 en 16:9</b>	3,5%	15%	5%	17,5%

## **Ejercicios Bloque G4**

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