Aplicaciones móviles en maemo

Aplicaciones Móviles
Curso de Adaptación
Grado en Ingeniería de Sistemas Audiovisuales

Celeste Campo - Carlos García Rubio
celeste, cgr@it.uc3m.es

Índice

- Introducción.
- Entorno de desarrollo.
- Aplicaciones GUI: Hildon.
- Integrando la aplicación en el entorno.
- PyMaemo.
- Referencias.

Introducción

maemo™

- Maemo: plataforma software desarrollada por Nokia, para *Internet Tablets* y *smartphones*, basada en Debian GNU/Linux.
  - GUI Hildon, basado en GTK.
  - Manejador de ventanas Matchbox.
- Incluye:
  - Sistema operativo.
  - SDK.
- Terminales:
  - Nokia 770
  - N800
  - N810
  - N900




Introducción

- Versiones:

	Version	SDK	Dispositivos
OS2005	1.1	-	770
OS2006	2.0	Mistral	770
	2.1	Scirocco	770
	2.2	Gregale	770
OS2007	3.0	Bora	770*, N800
	3.1		
	3.2		
OS2008	4.0	Chinook	770*, N800, N810
	4.1	Diablo	770*, N800, N810
Maemo 5	5.0	Fremantle	N900
MeeGo	-	Harmattan	

Entorno de desarrollo

- Maemo SDK:
  - Es el entorno de desarrollo para maemo.
  - Se puede instalar en ordenadores Linux.
    - También se puede descargar como máquina virtual VMWare de http://maemovmware.garage.maemo.org
  - El SDK crea un entorno maemo en el ordenador Linux, denominado *scratchbox*.
    - Incluye además herramientas como GDB, valgrind, ltrace, strace, htop, oprofile, time, etc.
    - Permite realizar compilación cruzada para el dispositivo.
  - Lenguaje de programación:
    - C/C++.
    - Hay también buen soporte de Python (pymaemo).

Entorno de desarrollo

- Probando el *scratchbox*: `helloworld.c`
  - Editamos el fichero:

```
#include <stdio.h>

int main(int argc, char** argv) {
  printf("Hello world\n");
  return 0;
}
```

  - Lo compilamos:

```
[sbox-DIABLO_X86: ~] > gcc -Wall -g helloworld.c -o helloworld
```

    - -Wall para que imprima todos los avisos de sintaxis.
    - -g para que añada símbolos de depuración al fichero de salida compilado
  - Lo ejecutamos:

```
[sbox-DIABLO_X86: ~] > ./helloworld
Hello world
```


Entorno de desarrollo

- Probando el *scratchbox*: un "*Hello World*" con interfaz gráfico (`gtk_helloworld-1.c`):

```
#include <stdlib.h>
#include <gtk/gtk.h>

int main(int argc, char** argv) {
  GtkWindow* window;
  GtkLabel* label;
  gtk_init(&argc, &argv);
  window = g_object_new(GTK_TYPE_WINDOW,"border-width", 12,
    "title", "Hello GTK+", NULL);
  label = g_object_new(GTK_TYPE_LABEL, "label",
    "Hello World!", NULL);
  gtk_container_add(GTK_CONTAINER(window), GTK_WIDGET(label));
  gtk_widget_show_all(GTK_WIDGET(window));
  g_print("main: calling gtk_main\n");
  gtk_main();
  g_print("main: returned from gtk_main and exiting with
success\n");
  return EXIT_SUCCESS;
}
```

Entorno de desarrollo

- Probando el *scratchbox*: un "*Hello World*" con interfaz gráfico (`gtk_helloworld-1.c`):
  - Compilamos:

```
[sbox-DIABLO_X86: ~] > gcc -Wall -g gtk_helloworld-1.c -o gtk_helloworld-1
gtk_helloworld-1.c:15:21: gtk/gtk.h: No such file or directory
gtk_helloworld-1.c: In function `main':
gtk_helloworld-1.c:20: error: `GtkWindow' undeclared (first use in this function)
gtk_helloworld-1.c:20: error: (Each undeclared identifier is reported only once
gtk_helloworld-1.c:20: error: for each function it appears in.)
gtk_helloworld-1.c:20: error: `window' undeclared (first use in this function)
gtk_helloworld-1.c:21: error: `GtkLabel' undeclared (first use in this function)
gtk_helloworld-1.c:21: error: `label' undeclared (first use in this function)
gtk_helloworld-1.c:24: warning: implicit declaration of function `gtk_init'
gtk_helloworld-1.c:28: warning: implicit declaration of function `g_object_new'
gtk_helloworld-1.c:28: error: `GTK_TYPE_WINDOW' undeclared (first use in this function)
gtk_helloworld-1.c:34: error: `GTK_TYPE_LABEL' undeclared (first use in this function)
gtk_helloworld-1.c:39: warning: implicit declaration of function `gtk_container_add'
gtk_helloworld-1.c:39: warning: implicit declaration of function `GTK_CONTAINER'
gtk_helloworld-1.c:39: warning: implicit declaration of function `GTK_WIDGET'
gtk_helloworld-1.c:42: warning: implicit declaration of function `gtk_widget_show_all'
gtk_helloworld-1.c:45: warning: implicit declaration of function `g_print'
gtk_helloworld-1.c:46: warning: implicit declaration of function `gtk_main'
```

Entorno de desarrollo

- Probando el *scratchbox*: un "*Hello World*" con interfaz gráfico (`gtk_helloworld-1.c`):
  - Hay que indicarle dónde puede encontrar el fichero de cabecera y las librerías para enlazar (*link*).
  - Nos podemos ayudar de la herramienta `pkg-config`.

```
[sbox-DIABLO_X86: ~] > pkg-config --list-all | sort
.. listing cut to include only relevant libraries ..
dbus-glib-1    dbus-glib - GLib integration for the free
desktop message bus
gconf-2.0      gconf - GNOME Config System.
gdk-2.0        GDK - GIMP Drawing Kit (x11 target)
gdk-pixbuf-2.0 GdkPixbuf - Image loading and scaling
glib-2.0       GLib - C Utility Library
gnome-vfs-2.0  gnome-vfs - The GNOME virtual file-system
libraries
gtk+-2.0       GTK+ - GIMP Tool Kit (x11 target)
hildon-1       hildon - Hildon widgets library
hildon-fm-2    hildon-fm - Hildon file management widgets
pango          Pango - Internationalized text handling
x11            X11 - X Library  return EXIT_SUCCESS;
```

Entorno de desarrollo

- Probando el *scratchbox*: un "*Hello World*" con interfaz gráfico (`gtk_helloworld-1.c`):
  - Para saber qué directorios necesitamos incluir al compilar para que busque cabeceras:

```
[sbox-DIABLO_X86: ~] > pkg-config --cflags gtk+-2.0
-I/usr/include/gtk-2.0 -I/usr/lib/gtk-2.0/include
-I/usr/include/atk-1.0
-I/usr/include/cairo -I/usr/include/pango-1.0
-I/usr/include/glib-2.0
-I/usr/lib/glib-2.0/include -I/usr/include/freetype2
-I/usr/include/libpng12
```

  - Y las librerías para enlazar:

```
[sbox-DIABLO_X86: ~] > pkg-config --libs gtk+-2.0
-lgtk-x11-2.0 -lgdk-x11-2.0 -latk-1.0 -lgdk_pixbuf-2.0 -lm
-lpangocairo-1.0 -lpango-1.0 -lcairo -lgobject-2.0 -lgmodule-2
-ldl -lglib-2.0
```

Entorno de desarrollo

- Probando el *scratchbox*: un "*Hello World*" con interfaz gráfico (`gtk_helloworld-1.c`):
  - Se puede hacer:

```
[sbox-DIABLO_X86: ~] > gcc -Wall -g gtk_helloworld-1.c  \
 `pkg-config --cflags gtk+-2.0` -o gtk_helloworld-1  \
 `pkg-config --libs gtk+-2.0
```

  - Para ejecutarlo:

```
[sbox-DIABLO_X86: ~] > ./gtk_helloworld-1
gtk_helloworld-1[4759]: GLIB WARNING ** Gtk - cannot open display:
```

  - Hay que arrancar un servidor X virtual para el emulador (Xephyr). Para ello, en otro terminal (ventana) hacemos:

```
user@system:~$ Xephyr :2 -host-cursor -screen 800x480x16 -dpi \
96 -ac -extension Composite
```

Entorno de desarrollo

- Probando el *scratchbox*: un "*Hello World*" con interfaz gráfico (`gtk_helloworld-1.c`):
  - De nuevo el en *scratchbox* definimos la variable DISPLAY:

```
[sbox-DIABLO_X86: ~] > export DISPLAY=:2
```

  - Arrancamos el *Application Framework* (AF):

```
[sbox-DIABLO_X86: ~] > af-sb-init.sh start
```

- Probando el *scratchbox*: un "*Hello World*" con interfaz gráfico (`gtk_helloworld-1.c`):
  - Ejecutamos la aplicación:

```
[sbox-DIABLO_X86: ~] > ./gtk_helloworld-1
```

Entorno de desarrollo

- Probando el *scratchbox*: un "*Hello World*" con interfaz gráfico (`gtk_helloworld-1.c`):
  - Mejor:

```
[sbox-DIABLO_X86: ~] > run-standalone.sh ./gtk_helloworld-1
```

    - Pone ciertas variables de entorno relacionados con el tema, para que el color de fondo y el tamaño de letra sea el que corresponde al tema actual.

Entorno de desarrollo

- Automatizaremos la construcción de nuestro ejecutable usando *Makefiles*.
  - Hay una muy buena introducción en la sección 4 del "*Maemo Diablo 4.1 release documentation*".
- Otra opción es usar un IDE como ESbox.
  - *ESbox* es un producto Eclipse para desarrollar aplicaciones en la plataforma maemo.
  - Está integrado con *scratchbox*.
  - Soporta C/C++, Python y los SDKs maemo 4.x y 5.x.
  - Viene instalado en la máquina virtual de http://maemovmware.garage.maemo.org

Aplicaciones GUI: GTK+ y Hildon

- Hildon es el interfaz gráfico de usuario que usa Maemo:
  - Diseñado para dispositivos pequeños.
  - Basado en GNOME.
  - Usa un manejador de ventanas más ligero (*Matchbox*).
- La mayor parte de los objetos gráficos (*widgets*) y métodos de GTK+ (conjunto de bibliotecas para desarrollar interfaces gráficas de usuario para GNOME y otros) funcionan en Hildon.
  - Ver "*GTK+ Reference Manual*".
- Aunque a la hora de portar una aplicación, tendremos que hacer algunos cambios para adaptarnos al tamaño de la pantalla.

Aplicaciones GUI: GTK+

- Antes de ver Hildon veamos cómo es un programa GTK+.
- El programa GTK+ más sencillo:

```
#include <gtk/gtk.h>

int main (int argc, char *argv[])
{
    GtkWidget *window;

    // arranca la librería y establece parámetros por defecto
    // pasamos parámetros de línea de comando a librería
    gtk_init (&argc, &argv);

    // creamos y muestramos una ventana
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_widget_show (window);

    // comienza el proceso del bucle principal GTK
    gtk_main ();

    // Para salir del bucle deberíamos llamar a la siguiente función:
    // void gtk_main_quit ();  si no habrá que hacer ^C

    return 0;
}
```

Aplicaciones GUI: GTK+

- Un Hola mundo hecho con GTK+ (`gtk_helloworld-1.c`):

```
#include <stdlib.h>
#include <gtk/gtk.h>

int main(int argc, char** argv) {
  GtkWindow* window;
  GtkLabel* label;
  gtk_init(&argc, &argv);
  window = g_object_new(GTK_TYPE_WINDOW,"border-width", 12,
    "title", "Hello GTK+", NULL);
  label = g_object_new(GTK_TYPE_LABEL, "label",
    "Hello World!", NULL);
  gtk_container_add(GTK_CONTAINER(window), GTK_WIDGET(label));
  gtk_widget_show_all(GTK_WIDGET(window));
  g_print("main: calling gtk_main\n");
  gtk_main();
  g_print("main: returned from gtk_main and exiting with
success\n");
  return EXIT_SUCCESS;
}
```

Aplicaciones GUI: GTK+

- Tipos de *widgets* en GTK+:
  - Button (Normal Button, Toggle Button, Check Button, Radio Button).
  - Range (Scrollbar, Scale, …).
  - Miscellaneous (Labels, Arrows, Progress Bars, Dialogs, Rulers, Statusbars, Text Entries, Spin Buttons, Combo Box, Calendar, Color Selection, File Selections, …).
  - Menu.
  - Container (para *widgets* que no tienen una ventana X asociada).
- Los *widgets* se pueden agrupar en cajas con gtk_hbox_new(), gtk_vbox_new(), gtk_box_pack_start(), gtk_box_pack_end()

Aplicaciones GUI: Hildon

- Modos de pantalla:
  - Hay cuatro. Las aplicaciones pueden usar cualquiera de ellos y cambiar de uno a otro dinámicamente.

  1. Vista normal:
     - Task Navigator area: 80 pixels ancho.
     - Statusbar/titlebar area: 720x60 pixels.
     - Application area: 696x396 pixels.
  2. Vista normal con barra de tareas:
     - Como la anterior, pero con una barra de tareas en la parte de abajo de 360 pixels.
  3. Vista a pantalla completa:
     - Toda la pantalla (800x480 pixels) está reservada para la aplicación.
  4. Vista a pantalla completa con barra de tareas:
     - Pantalla completa con barra de tareas de 422 pixels.

Aplicaciones GUI: Hildon

- En Hildon tenemos los siguientes leementos:
  - HildonProgram.
  - HildonWindow.
- HildonProgram:
  - Es el objeto gráfico que constituye la base de cualquier aplicación Hildon.
  - Hereda de GObject (de GTK+).
  - Se crea con:

```
HildonProgram *hildon_program_get_instance (void)
```

  - Se le añade una ventana con:

```
void hildon_program_add_window (HildonProgram *self,
                                HildonWindow *window)
```

Aplicaciones GUI: Hildon

- HildonWindow:
  - Representa una ventana de la aplicación.
  - Hereda de GtkWindow (de GTK+).
  - Se crea con:

```
GtkWidget *hildon_window_new (void)
```

  - Puede haber varias ventanas en la aplicación.
- Ejemplo aplicación Hildon (siguiente transparencia):

Aplicaciones GUI: Hildon

```
#include <hildon/hildon-program.h>
#include <gtk/gtkmain.h>
#include <gtk/gtklabel.h>

int main(int argc, char *argv[])
{
    /* Create needed variables */
    HildonProgram *program;
    HildonWindow *window;
    /* Initialize the GTK. */
    gtk_init(&argc, &argv);
    /* Create the Hildon program and setup the title */
    program = HILDON_PROGRAM(hildon_program_get_instance());
    g_set_application_name("App Title");
    /* Create HildonWindow and set it to HildonProgram */
    window = HILDON_WINDOW(hildon_window_new());
    hildon_program_add_window(program, window);
    /* Add example label to window */
    gtk_container_add(GTK_CONTAINER(window),
                      GTK_WIDGET(gtk_label_new("HildonProgram Example")));
    /* Begin the main application */
    gtk_widget_show_all(GTK_WIDGET(window));
    /* Connect signal to X in the upper corner */
    g_signal_connect(G_OBJECT(window), "delete_event",
    G_CALLBACK(gtk_main_quit), NULL);
    gtk_main();
    /* Exit */
    return 0;
}
```

Aplicaciones GUI: Hildon

- Ejemplo aplicación Hildon:
  - Compilamos con:

```
$ gcc -o example_hildonprogram example_hildonprogram.c `pkg-config
gtk+-2.0 hildon-1 --cflags --libs` -Wall
```

  - Lanzamos Xephyr, y desde un *scratchbox* lo lanzamos con:

```
$ run-standalone.sh ./example_hildonprogram
```

Aplicaciones GUI: Hildon

- Señales en GTK+:
  - En el ejemplo anterior hacíamos:

```
g_signal_connect(G_OBJECT(window), "delete_event",
                 G_CALLBACK(gtk_main_quit), NULL);
```

  - Los elementos gráficos en GTK pueden lanzar señales.
  - Las funciones para conectar las señales a un manejador son:

```
gulong g_signal_connect (gpointer *object, const gchar *name,
                         GCallback func, gpointer func_data);
```

  - Dónde los parámetros son:
    - object : el *widget* que lanza la señal.
    - name: nombre de la señal del *widget* a la que queremos conectar un manejador. Hay muchos eventos que podemos tratar.
      - button_press_event, button_release_event, drag_begin_event…
    - func: nombre de la función del manejador.
    - func_data: Parámetro de la función del manejador.

Aplicaciones GUI: Hildon

- Menús:
  - Pueden ser comunes a todas las ventanas de la aplicación:

```
void hildon_program_set_common_menu (HildonProgram *self,
                                     GtkMenu *menu)
```

  - O específicos de una ventana:

```
void hildon_window_set_menu (HildonWindow *self,
                             GtkMenu *menu)
```

- Barras de herramientas:
  - Idem. Común a todas las ventanas de la aplicación:

```
void hildon_program_set_common_toolbar (HildonProgram *self,
                                        GtkToolbar *toolbar)
```

  - Específicos de cada ventana:

```
void hildon_window_add_toolbar (HildonWindow *self,
                                GtkToolbar *toolbar)
```

Aplicaciones GUI: Hildon

- Menús y barras de herramientas:
  - En maemo se usan las funciones de GTK+ para crear menús y barras de herramientas.
  - Se puede hacer de dos maneras:
    - A mano (usando GtkMenu y GtkToolbar)
    - Usando GtkUIManager, que crea los menús a partir de ficheros XML.
      - Ver ejemplo de fichero XML en siguiente transparencia.

Aplicaciones GUI: Hildon

- Fichero XML de definición de interfaz con GtkUIManager:

```
<ui>
  <menubar>
    <menu name="FileMenu" action="FileMenuAction">
      <menuitem name="New" action="New2Action" />
      <placeholder name="FileMenuAdditions" />
    </menu>
    <menu name="JustifyMenu" action="JustifyMenuAction">
      <menuitem name="Left" action="justify-left"/>
      <menuitem name="Centre" action="justify-center"/>
      <menuitem name="Right" action="justify-right"/>
      <menuitem name="Fill" action="justify-fill"/>
    </menu>
  </menubar>
  <toolbar action="toolbar1">
    <placeholder name="JustifyToolItems">
      <separator/>
      <toolitem name="Left" action="justify-left"/>
      <toolitem name="Centre" action="justify-center"/>
      <toolitem name="Right" action="justify-right"/>
      <toolitem name="Fill" action="justify-fill"/>
      <separator/>
    </placeholder>
  </toolbar>
</ui>
```


Integrando la aplicación en el entorno

- Incluir la aplicación en el menú:
  - Para hacer que la aplicación sea visible al navegador de tareas de maemo, tenemos que crear un fichero `.desktop` que contiene la información necesaria para mostrar la aplicación en el menú.

```
[Desktop Entry]
Encoding=UTF-8
Version=1.0
Type=Application
Name=Example libOSSO
Exec=/usr/bin/example_libosso
X-Osso-Service=org.maemo.example_libosso
Icon=qgn_list_gene_default_app
MimeType=application/x-example;
```

  - Cuando la aplicación se instala, debe crearse un enlace a este fichero en el directorio `/etc/others-menu`.

```
[sbox-DIABLO_X86: /etc/others-menu] > ln -s \
 /usr/share/applications/hildon/example_libosso.desktop \
 /etc/others-menu/0010_example_libosso.desktop
```


Integrando la aplicación en el entorno

- Incluir la aplicación en el menú:
  - Campos del fichero `.desktop`:
    - Encoding: codificación del fichero, debería ser UTF-8.
    - Version: versión del fichero .desktop.
    - Type: tipo de entrada, debería ser Application.
    - Name: nombre de la aplicación que se mostrará en el menú del dispositivo.
    - Exec: aplicación binaria que se debe lanzar
    - X-Osso-Service: nombre del servicio D-BUS de la aplicación.
      - Lo explicaremos después.
    - Icon: nombre del icono que representa a la aplicación.
      - La aplicación puede instalar sus propios iconos, y además hay otros genéricos disponibles en la plataforma.
      - Formato PNG (no se debe poner la extensión en este campo, debe ser .png).
    - MimeType: tipo(s) MIME soportados por la aplicación.

Integrando la aplicación en el entorno

- D-BUS:
  - D-BUS es un mecanismo de comunicación entre procesos usado en GNOME y Hildon.
  - Permite enviar mensajes de unas aplicaciones a otras.
  - Hay un demonio que funciona como bus de mensajes, al cual pueden conectarse varias aplicaciones.
  - El demonio encamina mensajes desde una aplicación a cero, una o más aplicaciones (puede hacer difusión).
  - Una característica interesante de D-BUS es que si llega un mensaje a una aplicación que no está ejecutándose, D-BUS arrancará la aplicación automáticamente.
  - Además, sólo puede ejecutarse a la vez una instancia de cada servicio D-BUS, lo que garantiza que cada aplicación se está ejecutando sólo una vez.

Integrando la aplicación en el entorno

- Fichero de servicios D-BUS:
  - Tiene un formato más sencillo que el .desktop:

```
[D-BUS Service]
Name=org.maemo.example_libosso
Exec=/usr/bin/example_libosso
```

  - Campos:
    - Name: nombre del servicio D-BUS. Debe ser único. Para evitar colisiones se suele construir con la URL del proveedor de la aplicación (al revés), seguido del nombre de la aplicación.
    - Exec: camino completo del binario que debe ejecutar D-BUS cuando le llegue un mensaje.

Integrando la aplicación en el entorno

- Librería LibOSSO:
  - Es una librería básica, que contiene funciones que deben ejecutar todas las aplicaciones maemo, y otras que son útiles.
  - Inicialización:
    - Toda aplicación que se arranque desde el menú debe inicializarse, o puede tener un comportamiento inesperado (típicamente cerrarse a los pocos segundos).
    - Consiste en llamar a la función `osso_initialize()`.
      - Parámetros:
        - nombre D-BUS.
        - Versión.
        - Tercer parámetro TRUE si lo ha lanzado el D-BUS.
        - Bucle principal de contexto Glib (normalmente NULL).
      - Se debe que guardar el resultado (`osso_context_t`) para uso posterior.
        - Al cerrar la aplicación, hay que llamar a `osso_deinitialize(osso_context_t *osso)`

Integrando la aplicación en el entorno

- Librería LibOSSO:
  - Inicialización.

```
#define OSSO_EXAMPLE_NAME    "example_libosso"
#define OSSO_EXAMPLE_SERVICE "org.maemo."OSSO_EXAMPLE_NAME
/* ... */
    osso_context_t *osso_context;
    /* ... */
    /* Initialize maemo application */
    osso_context = osso_initialize(OSSO_EXAMPLE_SERVICE, "0.0.1", TRUE, NULL);
    /* Check that initialization was ok */
    if (osso_context == NULL) {
        return OSSO_ERROR;
    }
    /* ... */
    /* Deinitialize OSSO */
    osso_deinitialize(osso_context);
```

Integrando la aplicación en el entorno

- Librería LibOSSO:
  - Registrar la función de *callback* del D-BUS (1/2):

```
#include <hildon/hildon-program.h>
#include <hildon/hildon-banner.h>
#include <gtk/gtk.h>
#include <libosso.h>
#define OSSO_EXAMPLE_NAME    "example_libosso"
#define OSSO_EXAMPLE_SERVICE "org.maemo."OSSO_EXAMPLE_NAME
#define OSSO_EXAMPLE_OBJECT  "/org/maemo/"OSSO_EXAMPLE_NAME
#define OSSO_EXAMPLE_IFACE   "org.maemo."OSSO_EXAMPLE_NAME

/* ... */

/* Callback for normal D-BUS messages */
gint dbus_req_handler(const gchar * interface, const gchar * method,
                      GArray * arguments, gpointer data,
                      osso_rpc_t * retval)
{
    /* ... */
}
```

Integrando la aplicación en el entorno

- Librería LibOSSO:
  - Registrar la función de *callback* del D-BUS (2/2):

```
int main(int argc, char *argv[])
{
    osso_return_t result;

    /* ... */
    /* Add handler for session bus D-BUS messages */
    result = osso_rpc_set_cb_f(appdata->osso_context,
                               OSSO_EXAMPLE_SERVICE,
                               OSSO_EXAMPLE_OBJECT,
                               OSSO_EXAMPLE_IFACE,
                               dbus_req_handler, appdata);

    if (result != OSSO_OK) {
        g_print("Error setting D-BUS callback (%d)\n", result);
        return OSSO_ERROR;
    }
    /* ... */

    /* Deinitialize OSSO */
    osso_deinitialize(osso_context);
}
```


Integrando la aplicación en el entorno

- Librería LibOSSO:
  - Enviar un mensaje a un servicio usando D-BUS:

```
/* ... */
#define OSSO_EXAMPLE_NAME    "example_libosso"
#define OSSO_EXAMPLE_SERVICE "org.maemo."OSSO_EXAMPLE_NAME
#define OSSO_EXAMPLE_OBJECT  "/org/maemo/"OSSO_EXAMPLE_NAME
#define OSSO_EXAMPLE_IFACE   "org.maemo."OSSO_EXAMPLE_NAME
#define OSSO_EXAMPLE_MESSAGE "HelloWorld"
/* ... */
ret = osso_rpc_run(osso_context,
                   OSSO_EXAMPLE_SERVICE,
                   OSSO_EXAMPLE_OBJECT,
                   OSSO_EXAMPLE_IFACE,
                   OSSO_EXAMPLE_MESSAGE, &retval, DBUS_TYPE_INVALID);
/* ... */}
```

Integrando la aplicación en el entorno

- Librería LibOSSO:
  - Estado hardware:
    - Las aplicaciones maemo pueden conectarse a D-BUS para recibir mensajes como “batería baja” o “apagando (*shutdown*)” para por ejemplo preguntar al usuario si quiere salvar los ficheros o reaccionar de alguna manera.
    - Registraría la función de *callback* con:

```
/* ... */

/* Add handler for hardware D-BUS messages */
result = osso_hw_set_event_cb( appdata->osso_context,
    NULL, hw_event_handler, (gpointer) appdata );

if (result != OSSO_OK)
{
    g_print("Error setting HW state callback (%d)\n", result);
    return OSSO_ERROR;
}

/* ... */
```


Integrando la aplicación en el entorno

- Librería LibOSSO:
  - Estado hardware: Ejemplo de función de *callback* :

```
void hw_event_handler(osso_hw_state_t *state, gpointer data)
{
    AppData *appdata;
    appdata = (AppData *) data;
    if (state->shutdown_ind) {
        hildon_banner_show_information(GTK_WIDGET(appdata->window), NULL,
                                       "Shutdown event!");
    }
    if (state->memory_low_ind) {
        hildon_banner_show_information(GTK_WIDGET(appdata->window), NULL,
                                       "Memory low event!");
    }
    if (state->save_unsaved_data_ind) {
        hildon_banner_show_information(GTK_WIDGET(appdata->window), NULL,
                                       "Must save unsaved data event!");
    }
    if (state->system_inactivity_ind) {
        hildon_banner_show_information(GTK_WIDGET(appdata->window), NULL,
                                       "Minimize application inactivity event!");
    }
}
```


Integrando la aplicación en el entorno

- Librería LibOSSO:
  - Hay también un mensaje para cuando el sistema quiere cerrar nuestra aplicación por la razón que sea.
  - Nos permite realizar las acciones necesarias (p.ej. salvar datos) antes de que nos cierre.

```
int main( int argc, char* argv[] )
{
/* ... */
    /* Add handler for Exit D-BUS messages */
    result = osso_application_set_exit_cb(appdata->osso_context,
                                          exit_event_handler,
                                          (gpointer) appdata);
    if (result != OSSO_OK) {
        g_print("Error setting exit callback (%d)\n", result);
        return OSSO_ERROR;
    }
/* ... */
}
```

Integrando la aplicación en el entorno

- Librería LibOSSO:
  - Mensaje de cierre. Ejemplo de función de *callback*.

```
/* Callback for exit D-BUS event */
void exit_event_handler(gboolean die_now, gpointer data)
{
    AppData *appdata;
    appdata = (AppData *) data;
    g_print("exit_event_handler called\n");
    /* Do whatever application needs to do before exiting */
    hildon_banner_show_information(GTK_WIDGET(appdata->window), NULL,
                                   "Exiting...");
}
```

Integrando la aplicación en el entorno

- Valores de configuración:
  - Maemo usa GConf para manter los datos de configuración de la aplicación.
- Ayuda:
  - Para la ayuda se usa un fichero XML

```
<?xml version="1.0" encoding="UTF-8"?>
<ossohelpsource>
  <folder>
    <title>Help Framework Example</title>
    <topic>
      <topictitle>Main Topic</topictitle>
      <context contextUID="osso_example_help" />
      <para>This is a help file with example content.</para>
    </topic>
  </folder>
</ossohelpsource>
```

Integrando la aplicación en el entorno

- Para añadir la ayuda en la aplicación:

```
#include <libosso.h>
#include <hildon/hildon-help.h

#define OSSO_HELP_TOPIC_EXAMPLE "osso_example_help"

/* handler for Help button */
void help_activated(GtkWidget *win, gchar *help_id)
{
    osso_return_t retval;
    if (!help_id) {
        return;
    }
    retval = hildon_help_show(
        appdata.osso_context, /* global osso_context */
        help_id,              /* topic id */
        HILDON_HELP_SHOW_DIALOG);
}

/* ... */
/* Add a Help button */
help_button = gtk_button_new_with_label("Help");
g_signal_connect(G_OBJECT(help_button), "clicked", G_CALLBACK(help_activated),
                 OSSO_HELP_TOPIC_EXAMPLE)
/* ... */
```


Otros componentes de maemo

- Sistema de ficheros GnomeVFS:
  - Permite acceder a ficheros usando un amplio abanico de protocolos de acceso, de forma transparente al usuario:
    - Sistema local de ficheros, HTTP, FTP, OBEX sobre Bluetooth…
    - Independiente de si es un fichero local o remoto.
  - Se usan las siguientes funciones:
    - `gnome_vfs_init()`: inicializa la librería GnomeVFS. Es necesario hacerlo una vez al inicio de nuestro programa.
    - `gnome_vfs_shutdown()`: libera recursos de la librería y la cierra.
    - `gnome_vfs_open()`: abre un fichero usando una URI (p.ej. file:///tmp/somefile.txt), y devuelve un manejador de fichero si la operación se ha realizado con éxito.
    - `gnome_vfs_get_file_info()`: obtiene información sobre el fichero (similar a fstat).
    - `gnome_vfs_read()`: para leer datos de un fichero abierto.
    - `gnome_vfs_write()`: para escribir datos en un fichero abierto.

Otros componentes de maemo

- Alarmas:
  - Permite manejar fácilmente eventos que tienen que ocurrir en un determinado instante de tiempo.
  - Cada evento de alarma se identifica con una *cookie* única (), que se obtiene cuando se añade a la cola de eventos.

```
typedef long cookie_t;
```

  - Para añadir una alarma a la cola:

```
cookie_t alarm_event_add(alarm_event_t *event);
```

  - Para borrarla:

```
int alarm_event_del(cookie_t event_cookie);
```

Otros componentes de maemo

- Alarmas:
  - Estructura de un evento de alarma:

```
typedef struct {
        time_t alarm_time;          // number of seconds since 00:00:00 January 1,
                                    // 1970
        uint32_t recurrence;        // Interval in minutes, over which the event
                                    // should be repeated.
        int32_t recurrence_count;   // Number of times the event should repeat.
                                    //-1 for infinite.
        uint32_t snooze;            // Number of minutes the event is postponed
                                    // when Snooze button is pressed
        char *title;                // Title shown in the alarm dialog
        char *message;              // Message shown in the alarm dialog
        char *sound;                // Path to the sound file to be played
        char *icon;                 // Path to the icon file to show
        char *dbus_interface;
        char *dbus_service;         // Service to be called on D-Bus action
        char *dbus_path;
        char *dbus_name;
        char *exec_name;            // Command to run if the dbus is not set.
        int32_t flags;              // Options for the event
        uint32_t snoozed;           // Number of times the event has been snoozed
    } alarm_event_t;
```

Otros componentes de maemo

- Subsistema de conectividad maemo:
  - Permite gestionar la conexión del terminal a través de WiFi, WiMAX o de un teléfono vía Bluetooth.
- Información de localización:
  - Accede a la información de GPS.
    - Se incluye la librería:

```
#include <location/location-gps-device.h>
```

    - Hay que crear el objeto:

```
LocationGPSDevice *device;
device = g_object_new (LOCATION_TYPE_GPS_DEVICE, NULL);
```

    - Definimos la función de "*callback*" para cuando se obtiene una nueva localización:

```
g_signal_connect (device, "changed", G_CALLBACK
(location_changed), NULL);
```


Otros componentes de maemo

- Información de localización:
  - Accede a la información de GPS.
    - La función de "*callback*" tiene el siguiente aspecto:

```
static void
location_changed (LocationGPSDevice *device, gpointer userdata)
{
        g_print ("Latitude: %.2f\nLongitude: %.2f\nAltitude: %.2f\n",
                 device->fix->latitude, device->fix->longitude, device-
>fix->altitude);
}
```

    - Podemos acceder a la siguiente información del dispositivo GPS:
      - `device->online`
      - `device->status`
      - `device->fix`
      - `device->satellites_in_view`
      - `device->satellites_in_use`
      - `device->satellites`

PyMaemo

- Permite programar en maemo usando Python.
- Un programa sencillo:

```
import gtk
import hildon

def main():
    gtk.set_application_name("Simplest example")

    window = hildon.Window()
    window.show()

    gtk.main()

if __name__ == "__main__":
    main()
```

- Para ejecutarlo (hay que tener python instalado en el dispositivo):

```
run-standalone.sh python2.5 base.py
```


PyMaemo

- Hello World Python Hildon (1/2):

```
import gtk
import hildon

# This is a callback function. The data arguments are ignored in this example.
def hello(widget, data):
    print "Hello World!"

def main():
    # Get an instance of HildonProgram. It is an object used to represent an
    # application running in the Hildon framework.
    program = hildon.Program.get_instance()

    # create a new hildon window
    window = hildon.Window()

    # Registers a window as belonging to the program
    program.add_window(window)

    # When the window is given the "delete_event" signal (this is given by the
    # window manager, usually by the "close" option, or on the titlebar), we
    # ask it to call the delete_event () function as defined above. The data
    # passed to the callback function is None and is ignored in the callback
    # function.
    window.connect("delete_event", gtk.main_quit, None)
```

PyMaemo

- Hello World Python Hildon (2/2):

```
        button = hildon.Button(gtk.HILDON_SIZE_AUTO,
hildon.BUTTON_ARRANGEMENT_VERTICAL, "Hello world!")

        # When the button is given the "clicked" signal, we ask it to call the
        # hello () function as defined above. The data passed to the callback
        # function is None and is ignored in the callback function.
        button.connect("clicked", hello, None)

        # This packs the button into the window (a GTK+ container).
        window.add(button)

        # The final step is to display this newly created widget and all widgets it
        # contains.
        window.show_all()

        # All GTK+ applications must have a gtk_main(). Control ends here and waits
        # for an event to occur (like a key press or mouse event).
        gtk.main()

    if __name__ == "__main__":
        main()
```

Qt

- Qt es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario.
- Inicialmente desarrollada por la empresa noruega Trolltech (en aquel momento «Quasar Technologies») siguiendo un desarrollo basado en el código abierto.
- En 2008 la compró Nokia.
- Sigue siento distribuida bajo los términos de GNU Lesser General Public License (y otras), Qt es software libre y de código abierto.
- Qt se ha usado para desarrollar aplicaciones como KDE, Google Earth, etc.
- Nokia ha desarrollado recientemente Qt for Symbian y Qt for maemo, y pretende que con ello se pueda desarrollar fácilmente aplicaciones para el PC de sobremesa (Apple, Microsoft y Linux) y para móvil (Symbian y maemo).

Referencias

- "Maemo Diablo Reference Manual for maemo 4.1“, http://maemo.org/maemo_release_documentation/maemo4.1.x/
- “Maemo 5 Developer Guide”, http://wiki.maemo.org/Documentation/Maemo_5_Developer_Guide
- “GTK+ Reference Manual”, http://maemo.org/api_refs/4.1/gtk+2.0-2.10.12/libgtk2.0/
- “PyMaemo/UI Tutorial”, http://wiki.maemo.org/PyMaemo/UI_tutorial

- Otras referencias:
  - Qt Whitepaper, http://qt.nokia.com/files/pdf/qt-4.6-whitepaper
  - Qt Reference Documentation, http://doc.qt.nokia.com/4.6/index.html