El proyecto KDE

Antonio Larrosa Jiménez

Octubre de 1999

Resumen

En esta ponencia trataré de explicar los objetivos con los que se fundó KDE. Hablaré del estado actual y del camino que seguirá en un futuro el proyecto. Además, comentaré la facilidad con la que el programador puede desarrollar nuevas aplicaciones usando las mas avanzadas tecnologías opensources.

Por último, explicaré la compleja organización de KDE, uno de los mas grandes proyectos en el terreno del mundo opensource.

1 Introducción

Hace ya algunos años que existen sistemas UNIX . La mayoría de ellos son sistemas comerciales, hechos generalmente por la misma empresa que se dedica a fabricar el hardware sobre el que se ejecutan los programas (tal es el caso de SGI, HP, Digital, SUN, etc.).

Pero recientemente, viene siendo mas y mas habitual oir hablar de cierto sistema operativo llamado Linux en cualquier sitio (prensa, radio, tv ...) . Qué duda cabe de que el advenimiento de las mejoras en los interfaces de usuario ha tenido mucho que ver en esto.

La interfaz de usuario es lo que éste ve, y por lo tanto, cuando el usuario no tiene experiencia con un programa (y no sabe como funciona), es lo que le hace decidir si usar ésta u otra aplicación, que hace el trabajo de forma mas sencilla e intuitiva.

2 Objetivos de KDE

KDE se fundó para poner orden en ese tiempo en que cada aplicación de UNIX funcionaba y tenían un aspecto distintos al resto, lo que hacía que el usuario tuviera una curva de aprendizaje muy *resbaladiza*.

Los principales objetivos de KDE son :

- Proveer a los sistemas UNIX de un entorno abierto, amigable, potente, seguro y estable.
- Desarrollar librerías abiertas que faciliten y potencien la labor creativa del programador.
- Describir estándares para hacer aplicaciones mas intuitivas, homogéneas y fáciles de usar para el usuario inexperto a la vez que potentes y productivas para el usuario avanzado.
- Crear un modelo de componentes o aplicaciones que puedan ser usadas dentro de otros programas usando la nueva tecnología **Kanossa**

El primero de los puntos de arriba es muy importante, ya que el usuario de cualquier sistema espera todas esas características de su ordenador.

El segundo de los puntos es igualmente importante, porque si un programador quiere hacer un programa, y puede reutilizar código que ya haya escrito otra persona, está aprovechando su tiempo para dedicarlo exclusivamente a escribir nuevo código para su programa y a depurar el programa para que sea más estable. En la sección x**** **??** daré ejemplos mas extensos de programación bajo KDE.

El tercer punto hace referencia a los estándares que deben seguir los programas que sean escritos usando las librerías del proyecto para que sean considerados como programas comformes a los estándares de KDE. Dichos estándares especifican cosas como la posición de los elementos dentro de la estructura del menú, la colocación de los botones en un diálogo, la preferencia por hacer todos los programas transparentes a la red, de forma que el usuario pueda abrir un fichero local de la misma manera que abre un fichero que está en otra parte del mundo (usando ftp, http u otros), etc.

Para más información sobre los estándares de usuario, recomiendo leer las páginas de ***** en http://****

En el último punto, se habla de una nueva tecnología llamada Kanossa. Torben Weis, creador de la tecnología KOM/Openparts, que lleva usandose desde hace más de un año en proyectos como la suite KOffice, pensó en rediseñar todas estas especificaciones y librerías . Fruto de este rediseño , aparece **Kanossa**, un sistema para hacer aplicaciones y documentos que pueden empotrarse (embeeded) en otras rapidamente y de forma sencilla para el programador.

Gracias a Kanossa, un programa puede exportar su interfaz de usuario, de forma que cuando otra aplicación lo necesita, puede utilizarla, añadiendo entradas en los menús, y botones en las barras de herramientas, por ejemplo.

Actualmente, muchos de estos objetivos se cumplen ampliamente, otros, en cambio, no se cumplen del todo, y es ahí donde KDE 2.0 va a poner todo su esfuerzo.

3 Nuevas tecnologías en KDE

KDE 2.0 va a romper esquemas en el mundo de los entornos de escritorio, introduciendo nuevas tecnologías que hacen mucho mas efectivo el uso del ordenador.

Una de estos avances es el ya mencionado Kanossa. Kanossa es una librería abierta que provee de todas las funcionabilidades necesarias para meter unas aplicaciones dentro de otras de forma que las interfaces de usuario de ambos programas se mezclan de forma controlada para que actuen como si fueran uno sólo.

La principal característica de Kanossa es la impresionante velocidad con que realiza todas las operaciones. Gracias a estar totalmente libre de usar Corba y al tremendo esfuerzo de Torben Weis, además los programas no sólo se ejecutan mas rápido, sino que compilan mas rápido . El código es mucho mas legible también ya que usa un fichero xml para definir la interfaz de usuario (los menús y botones de la barra de tareas por ejemplo), haciendo mucho mas fácil la tarea "rutinaria" del programador para que pierda su tiempo en cosas útiles (o eso se espera :-)). Además, para poner un ejemplo de la facilidad que Kanossa proporciona, se puede decir que sólo añadiendo 5 lineas de código a un programa ya existente, se le puso un navegador entero empotrado dentro de una ventana.

DCOP (Desktop COmmunication Protocol) es el protocolo que usa KDE 2.0 para la comunicación entre procesos. KDE 1.0 usaba X Atoms por medio de kwmcom para que distintas aplicaciones pudieran comunicarse, pero esto tenía ciertos problemas, como el tamaño limitado de los mensajes.

La nueva solución, DCOP, opera sobre sockets (tanto UDS como TCP/IP) y está construido sobre el protocolo Inter Client Exchange (ICE) que es una parte estandar de X11R6 y posteriores.

Como ejemplo de la facilidad con la que un programa se puede comunicar con otro, es bastante conveniente enseñar un poco de código fuente :

```
QByteArray parametros, respuesta;
QDataStream stream(parametros, IO_WriteOnly);
parametros << 5;
if (!cliente->call("AppId", "ciertoObjeto", "QString ciertaFuncion(int)",
                   parametros, respuesta))
  kdebug(KDEBUG_ERROR, 0, "Error usando DCOP.");
else {
  QDataStream stream2(respuesta, IO_ReadOnly);
  QString resultado;
  stream2 >> resultado;
  print("el resultado es: %s",resultado.latin1());
```

Y el mismo código es bastante auto explicativo. Simplemente, **cualquier** objeto que se quiera pasar a otra aplicación, se redirige a un stream y después se manda el stream por medio de una simple llamada a una función. Una nota importante a decir es que todas las clases de Qt (así como muchas de KDE) soportan streams de este modo.

Otra de las nuevas aportaciones que KDE 2.0 hará al mundo Unix es un entorno de escritorio totalmente compatible con Unicode y por tanto capaz de mostrar caracteres latinos, orientales, arábigos, etc. De esta forma, cualquier usuario de cualquier parte del mundo podrá tener un sistema Linux (o UNIX en general) completamente regionalizado.

Una de las nuevas características de KDE 2.0 será el soporte de temas "reales", no sólo se pueden cambiar los colores y bordes de las ventanas, ni tampoco se restringe a permitir cambiar los pixmaps de cada elemento de la pantalla (como botones, o menús) . Los nuevos temas de KDE 2.0 permiten (además de todo eso) controlar el comportamiento de los objetos así como la forma de éstos (pudiendo por ejemplo poner los botones de una barra de desplazamiento en cualquier parte de esta dependiendo del tema seleccionado).

La forma de crear nuevos temas es realmente sencilla pues existe un diseñador de temas que ayuda a los no programadores a hacer sus propios temas . Además, los programadores que deséen programar un tema (que cambie algo más que la apariencia de otro ya existente) , sólo tienen que derivar de una clase e implementar sus propios algoritmos para todas las funcionalidades que los temas ofrecen.

Otro de los puntos a destacar dentro de KDE es la implantación de DOM (Document Object Model) tanto en la reescrita librería khtml (la encargada de visualizar un documento html) como en la suite KOffice.

Para terminar esta sección, habría que comentar que todas estas nuevas tecnologías están usandose ya en la suite KOffice[1]. KOffice se compone de un procesador de textos, KWord; un programa para hacer presentaciones, KPresenter; una hoja de cálculo, KSpread; una base de datos, Katabase; un programa de diseño vectorial, KIllustrator; un programa de retoque fotográfico, KImageShop; un editor de ecuaciones, KFormula, etc.

Algunos de estos programas se encuentran ya en una fase en la que son totalmente funcionales, (como es el caso de KWord y KPresenter), mientras que otros son todavía versiones muy tempranas ya que se han empezado a desarrollar mas tarde.

Haciendo uso de Kanossa, se puede incluir un documento de un programa dentro de un documento de otro, de forma transparente para el usuario, que se encuentra con todas las opciones del programa "empotrado" dentro de la misma estructura demenús del programa original. Así se consigue por ejemplo, tener una hoja de cálculo hecha con KSpread dentro de un documento de KWord, un documento de KWord dentro de una presentación, etc.

4 Introducción a la programación bajo KDE/Qt

En esta sección voy a explicar un poco como se programa en KDE/Qt, cosa que es realmente sencilla. Cualquiera que quiera programar usando dichas librerías no tendrá excusa después de haber leido este apartado :-)

Como base fundamental, sólo se requiere conocimientos de C++. Quien lo desée, también puede programar para KDE usando los *bindings* que hay para Perl, Python, C, u otros, pero estos temas se salen del ámbito de esta introducción.

Ejemplo 1. Hola Mundo en Qt

main.cpp :

```
------------------
#include <qapplication.h>
#include <qpushbutton.h>

int main( int argc, char **argv )
{
    QApplication *a=new QApplication( argc, argv );

    QPushButton *hola=new QPushButton( "Hola mundo!", 0 );
    hola->resize( 100, 30 );

    QObject::connect( hola, SIGNAL(clicked()), a, SLOT(quit()) );

    a->setMainWidget( hola );
    hola->show();

    int r = a->exec();

    delete a;

    return r;
}
----------------
```

Este es un programa mínimal, que hace uso sólo de Qt. Veamos qué pasa linea a linea.

```
QApplication *a=new QApplication( argc, argv );
```

Crea una aplicación Qt, este objeto controla el flujo de control de la aplicación, las señales/slots de Qt, etc.

```
QPushButton *hola=new QPushButton( "Hola mundo!", 0 );
hola->resize( 100, 30 );
```

Esta linea crea un botón de los que se pueden pulsar (QPushButton) con texto "Hola mundo!" y como ventana padre, le ponemos 0 porque realmente no tiene. Además, le cambiamos el tamaño, para que el usuario lo vea bien.

[1] http://koffice.kde.org

```
    QObject::connect( hola, SIGNAL(clicked()), a, SLOT(quit()) );
```

Esta linea hace uso de uno de las características fundamentales de Qt, las señales y los slots (ranuras). Los objetos en Qt (y KDE) emiten una señal cuando cambian su estado. Además, también hay ranuras, que són funciones a las que se pueden conectar las señales, para que sean llamadas automáticamente cuando otro objeto cambia su estado.

Dicho esto, será mas facil comprender el significado de esa línea. Con ese código se conecta la señal que el botón emite cuando se pulsa (clicked()) al slot de la *a* que cierra la aplicación.

```
    a->setMainWidget( hola );
    hola->show();
```

Selecciona el botón *hola* como widget principal de la aplicación, y lo muestra en la pantalla.

```
    int r = a->exec();
```

Comienza la ejecución del bucle de eventos de Qt, de forma que el usuario ya puede interaccionar con la aplicación.

Una vez que la aplicación termina (porque el usuario haya pulsado en el botón o por cualquier otro motivo), la ejecución sigue a partir de esta linea.

```
    delete a;

    return r;
```

Destruye la aplicación y termina el programa.

Facil, ¿no? :-)

Ejemplo 2. Hola Mundo en KDE

main.cpp :

```
------------------
nclude <kapp.h>
#include <klocale.h>
#include <qpushbutton.h>

int main( int argc, char **argv )
{
    KApplication *a=new KApplication( argc, argv );

    QPushButton *hola=new QPushButton( i18n("Hola"), 0 );
    hola->resize( 100, 30 );

    QObject::connect( hola, SIGNAL(clicked()), a, SLOT(quit()) );

    a->setMainWidget( hola );
    hola->show();

    int r = a->exec();

    delete a;

    return r;
}
----------------
```

Este programa es el equivalente al anterior, pero ahora sí que usamos KDE. El programa se parece mucho, pero realmente tiene mucha mas funcionalidad añadida gratuitamente.

Nos fijamos en el primer cambio:

```
    KApplication *a=new KApplication( argc, argv );
```

Usamos un objeto KApplication en vez de QApplication. La diferencia es que KApplication hace gestiones desarrolladas específicamente por KDE para usar mas comodamente las kdelibs (clases como KConfig, KLocale, etc.)

```
    QPushButton *hola=new QPushButton( i18n("Hola"), 0 );
```

Aquí, en vez de poner como texto la palabra "Hola", ponemos el texto que devuelva i18n("Hola"). Simplemente con eso, conseguimos tener nuestro programa traducido a cualquier idioma, la función i18n mira en qué idioma tenemos configurado nuestro escritorio y automaticamente traduce el texto "Hola" al idioma correcto (por supuesto, esto necesita de traductores para cada idioma, pero con un programa así, todos van a querer traducirlo :-))

Ejemplo 3. Un ejemplo mas real

p3.h :

```
------------------
#include <ktmainwindow.h>
#include <qpopupmenu.h>
#include <klocale.h>
#include <kmenubar.h>

class MiVentana : public KTMainWindow
{
 Q_OBJECT
public:

MiVentana ( char * titulo );

public slots:
void archivo_open();
void archivo_save();

};
----------------
```

p3.cpp :

```
------------------
#include "p3.h"
#include <kfiledialog.h>
#include <kapp.h>
#include <klocale.h>

MiVentana::MiVentana ( char * titulo ) : KTMainWindow (titulo)
{
    QPopupMenu *m_archivo = new QPopupMenu;
    m_archivo->insertItem( i18n("&Open"), this, SLOT(archivo_open()) );
    m_archivo->insertItem( i18n("&Save"), this, SLOT(archivo_save()) );
    m_archivo->insertItem( i18n("&Quit"), kapp, SLOT(quit()) );
    KMenuBar *menu = new KMenuBar(this);
    menu->insertItem(i18n("&File"),m_archivo);
    setMenu(menu);

}

void MiVentana::archivo_open()
{
    QString filename=KFileDialog::getOpenFileName(QString::null,"*",this);
}

void MiVentana::archivo_save()
{
    QString filename=KFileDialog::getSaveFileName(QString::null,"*",this);
}
----------------
```

main.cpp :

```
------------------
#include <kapp.h>
#include "p3.h"

int main( int argc, char **argv )
{
    KApplication *a=new KApplication( argc, argv );

    MiVentana *miventana=new MiVentana( "Un ejemplo" );
    miventana->resize( 300, 200 );

    a->setMainWidget( miventana );
    miventana->show();

    int r = a->exec();

    delete a;

    return r;
}
----------------
```

Bien, este programa es un poco mas complejo, pero no mucho mas. Veamos paso por paso. En la función main, en vez de crear un botón ahora creamos un objeto de la clase MiVentana, que hereda de KTMainWindow (la clase de la que deben heredar todas las ventanas principales de una aplicación).

Notamos que en la declaración de la clase MiVentana, hemos escrito Q_OBJECT, esto es una macro definida por Qt, que hace posible el definir señales y slots en las declaraciones de las clases.

Además, definimos dos slots, fichero_open(), y fichero_save(), para esto, usamos *public slots:*.

```
QPopupMenu *m_archivo = new QPopupMenu;
m_archivo->insertItem( i18n("&Open"), this, SLOT(archivo_open()) );
m_archivo->insertItem( i18n("&Save"), this, SLOT(archivo_save()) );
m_archivo->insertItem( i18n("&Quit"), kapp, SLOT(quit()) );
```

En el constructor de la ventana, definimos un QPopupMenu, y le añadimos tres elementos que serán traducidos oportunamente. Además, los conectamos a este (this) objeto, a los slots que hemos definido, y el tercer elemento del menú, lo conectamos a la aplicación, para que la termine.

```
KMenuBar *menu = new KMenuBar(this);
menu->insertItem(i18n("&File"),m_archivo);
setMenu(menu);
```

Y aquí definimos el menú real, con el popup menú recien estrenado. Y lo activamos.

Además, con:

```
QString filename=KFileDialog::getOpenFileName(QString::null,"*",this);
```

Tenemos un diálogo de abrir fichero de KDE, que sigue todos los estándares. Y esto nos da el nombre del fichero a abrir en la cadena *filename*.

Si quisieramos, podríamos usar KFileDialog::getOpenFileURL, que permite al usuario introducir cualquier url (e incluso, navegar usando ftp desde el mismo diálogo). Esto nos obligaría a poner soporte de URLs en nuestro programa, que no es dificil usando la librería KIO, pero que queda fuera de esta introducción.

Quien quiera mas información para programar en KDE, puede visitar developer.kde.org[2] donde hay extensa documentación. Además, la ayuda y los tutoriales de Qt son *extremadamente* útiles, extensos, y detallados. Puedes también hacer cualquier pregunta al respecto en las listas de KDE para desarrolladores, mira aqui[3] para ver cómo suscribirte.

[2] http://developer.kde.org

[3] http://www.kde.org/contact.html

5 Conclusiones

Hace poco que se celebró la KDE-II, donde los desarrolladores de KDE hemos precisado cuales van a ser los pasos que KDE va a dar en el futuro. De todas formas, por la naturaleza opensource del proyecto, el avance es contínuo, y lo dicho en estas lineas puede ser reemplazado por nuevas formas mas rápidas y sencillas en cualquier momento.