

Programación en Abanq (III). Módulos

Para terminar con la serie de artículos sobre el uso avanzado de Abanq, vamos a profundizar un poco más en el funcionamiento del sistema. Concretamente nos centraremos en el sistema de 'scripting' y los mecanismos que nos ofrece para poder implementar cualquier funcionalidad de manera interpretada mediante el lenguaje QSA.

Este capítulo es algo más teórico que los anteriores, pero aun así terminaremos con un ejemplo práctico que nos mostrará como crear un pequeño módulo para generar gráficos estadísticos a partir de los datos obtenidos de las consultas utilizadas para los informes.

Arquitectura por capas

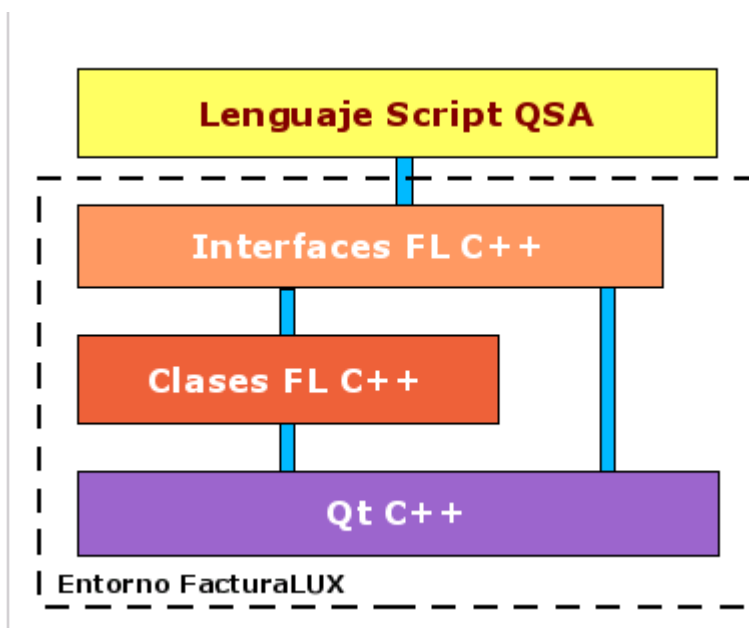
La arquitectura de Abanq ha sido cuidadosamente diseñada mediante capas perfectamente definidas. Esta estructura permite, en su capa mas externa, trabajar con un lenguaje de script (QSA) que oculta gran parte de los detalles internos y permite al desarrollador abstraerse de los mismos y centrarse solamente en la solución.

Las capas inferiores ofrecen mediante su interfaz (API) su funcionalidad a las capas superiores. La capa de más bajo nivel (el magnífico conjunto de herramientas Qt), es la que finalmente ofrece las características multiplataforma de Abanq ya que es capaz de lidiar con distintos sistemas operativos.

En el entorno de Abanq se utilizan dos tipos de lenguajes:

Lenguaje QSA. Es el utilizado en los scripts de los módulos, basado en ECMAScript (y por tanto muy parecido a JavaScript). No necesita ser compilado.

Lenguaje C++. Es el utilizado para crear el núcleo de Abanq (esto es, las aplicación base). Se utiliza el Qt, una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario. Utiliza el lenguaje C++ pero permite usar también C, Python y Perl, además cuenta con soporte para acceder a bases de datos mediante SQL, XML y API para el manejo de ficheros.



El funcionamiento básico es el siguiente; los objetos definidos en Qt (núcleo de la aplicación) pueden ser accedidos desde QSA (scripts), pero no directamente por razones de seguridad. Este acceso se realiza mediante la creación de unas clases intermedias (Interfaces FL C++) que actúan como interfaz de las clases de Abanq creadas en Qt (Interfaces FL C++), a su vez basadas en C++ (Qt C++).

Desde el código QSA se pueden crear objetos de las clases interfaz para acceder a los correspondientes objetos Qt. Por tanto, sólo se podrá acceder a aquellas clases (y, dentro de éstas, a los métodos) de Qt que hayan sido explícitamente definidos en el interfaz. El interfaz actúa como conexión entre QSA y Qt.

A su vez, QSA dispone de las funciones propias de cualquier lenguaje de programación.

Uso de QSA en Abanq

Variables. Las variables se declaran usando la palabra clave var:

```
var a;
```

En esta declaración la variable a esta indefinida, para crear una variable definida procederemos de la siguiente manera:

```
var a = "hola";
```

Ámbito de las variables. Cuando declaramos una variable con la palabra clave var su ámbito se reduce al bloque en el que ha sido declarada. Es una variable local. Sin embargo, si ponemos el nombre de la variable con su asignación sin la palabra clave var, se declara automáticamente como global.

Constantes. Las constantes se definen mediante la palabra clave const

```
const x = "bueno";
```

```
const y = 42;
```

Las constantes deben ser asignadas a la vez que se definen, ya que su valor no puede ser modificado más adelante. Si se definen dentro de un bloque su ámbito será local. Si se definen fuera serán de ámbito global.

Funciones y Llamadas. Las funciones deben ser declaradas con la palabra clave function seguida del nombre de la función:

```
function nomFuncion()
{
    .....
    .....
    return x;
}
```

la palabra nomFuncion es el nombre de la función, el paréntesis indica los parámetros que se le pasan a la función (en este caso no los hay). Todo el cuerpo de la función se encuentra encasillado por llaves ({}). La palabra clave return devuelve el resultado de la ejecución de la función (en este caso x)

Para que esta función se ejecute debemos realizar una llamada. El formato de la llamada depende del lugar desde el que se efectúe:

* Dentro del script

```
var y = nomFuncion();
```

El resultado de llamar a la función será almacenado por la variable y.

* Dentro del módulo principal nombre_modulo_principal.nombre_funcion();

Supongamos que nuestra función se encuentra en el módulo principal flfactppal, entonces la llamada a la función será:

```
flfactppal.nomFuncion();
```

* Dentro del script asociado al formulario maestro

Imaginemos que en nuestro fichero .xml tenemos la siguiente acción:

```
<action>
  <name>se_persona</name>
  <table>se_persona</table>
  <form>se_master</form>
  <formrecord>se_persona</formrecord>
  <scriptform>se_persona</scriptform>
</action>
```

Para llamar a una función que se encuentra en el script asociado a la etiqueta <scriptform>:

```
form_nombre_accionXML.nomFuncion();
```

En este caso la llamada sería la siguiente: formse_persona.crearfuncion();

* Dentro del script asociado al formulario edición

```
<action>
  <name>se_telefono</name>
  <table>se_telefono</table>
  <form>master</form>
  <formrecord>se_telefono</formrecord>
  <scriptformrecord></scriptformrecord>
</action>
```

En este caso la función se encuentra en el script correspondiente a la etiqueta <scriptformrecord> que pertenece al formulario edición. La llamada sería:

```
formrecord_nombre_accionXML.nomFuncion();
```

En este ejemplo la llamada se realizaría de la siguiente forma:

```
formRecordse_telefono.nomFuncion();
```

Principales objetos

Veremos algunos de los objetos y métodos más habituales en el acceso a datos desde los scripts. Estos objetos (y las clases de las que derivan) pertenecen al interfaz entre el núcleo de Abanq y los scripts.

Método cursor() sobre formularios. Los formularios nos permiten crear cursores. Un cursor sobre un formulario da acceso a todos los objetos que forman parte del mismo. La forma de declarar un cursor sobre un formulario es la siguiente:

```
var cursorF = this.cursor();
```

Así creamos un cursor sobre el formulario actual con el nombre cursorF.

Algunos de los métodos más utilizados sobre estos objetos son valueBuffer(nombreCampo) para leer el contenido de un campo y setValueBuffer(nombreCampo, valor) para establecer el valor de un campo.

Método child() sobre formularios. Todos los componentes del formulario se consideran hijos de este. Podemos almacenar el contenido de cada uno de ellos en variables de la forma siguiente:

```
var nombre = this.child("fdbNombre");
```

La variable nombre contiene el elemento fdbNombre del formulario que a su vez se refiere a un campo de la tabla asociada a dicho formulario. La variable nombre será un objeto del mismo tipo que el componente del que procede, frecuentemente un FLFieldDB. En función del tipo de objeto de que se trate, dispondremos de un conjunto de métodos y propiedades para acceder a él.

Existe la posibilidad de habilitar o deshabilitar un componente del formulario mediante el uso de:

- setDisabled() cuando se trata de un objeto tipo botón
- setTabEnabled() cuando se trata de un objeto tipo tab
- enabled cuando se trata de un objeto tipo campo.

Ejemplos:

```
this.child("nombreboton").setDisabled("true ó false");
```

```
this.child("nombretab").setTabEnabled(numeropagina, true ó false);
```

```
this.child("nombrecampo").enabled = true ó false;
```

Cursor sobre Tablas. Se utiliza para poder trabajar directamente con los datos de una tabla sin pasar por un formulario. Son objetos de tipo FLSqlCursor:

```
var cursorTab = new FLSqlCursor("nombreTabla");
```

Donde nombreTabla es el nombre de la tabla en base de datos.

Métodos más comunes:

* setModeAccess: Define el modo de acceso del cursor sobre la tabla. Existen cuatro modo de acceso:

- o 1. Insert. Inserción de datos.
- o 2. Edit. Edición de datos.
- o 3. Del. Borrado de datos.
- o 4. Browse. Acceso de sólo lectura.

El modo de acceso es una propiedad del objeto formulario; se utilizará así:

```
cursorTab.setModeAccess (cursorTab.Insert);
```

* setValueBuffer: Para insertar un valor en un campo de la tabla.

```
cursorTab.setValueBuffer(nombrecampo,valor_a_insertar);
```

* valueBuffer: Para leer un valor de un campo de la tabla. Para ello hay que posicionar el cursor sobre el registro de la tabla:

```
var codart = 155; var cursorArt = new FLSqlcursor("articulos"); cursorArt.Select("codarticulo = " + codart);
cursorArt.first(); var idarticulo = cursorArt.valueBuffer("id");
```

En este ejemplo se crea una variable que contiene el valor 155, la cual es utilizada como criterio de búsqueda. La función select() establece este criterio. La función first() posiciona el cursor sobre el registro deseado. Una vez posicionado el cursor se obtiene el valor mediante valueBuffer()

Funciones predefinidas

Las funciones predefinidas son invocadas automáticamente al producirse determinados eventos. Estas funciones pueden ser redefinidas según las necesidades del programador. Salvo que se indique lo contrario, estas funciones se deben ubicar en el script correspondiente al formulario en el que se produce el evento. En este apartado haremos alusión a las más utilizadas:

init()

- Momento de ejecución: Apertura del formulario asociado con el script que la contenga.
- Parámetros: Ninguno.
- Valor de retorno: Ninguno
- Uso: Acciones de inicialización del formulario. Ejemplo: En el caso en el que al abrir el formulario deseemos que aparezcan algunos campos deshabilitados y conectar algún botón a una función:

```
function init()
{
  form.child("fdbCliente").setDisabled(true);
  var pbnGenerarFactura = this.child("pbnGenerarFactura");
  connect(pbnGenerarFactura, "clicked()", this, "generarFactura");
}
```

Hemos deshabilitado el componente fdbCliente y hemos conectado la pulsación de un botón llamado pbnGenerarFactura con la ejecución de la función generarFactura()

validateForm()

- Momento de ejecución: Pulsación del botón Aceptar o Aceptar y continuar del formulario asociado con el script que la contenga.
- Parámetros: Ninguno
- Valor de retorno: Valor booleano que indica si la validación ha sido correcta o no.
- Uso: Se suele utilizar para hacer comprobaciones sobre los datos introducidos en el formulario.

Ejemplo: Se comprueba antes de validar el formulario que el contenido de uno de los campos no sea vacío.

```
function validateForm()
{
  var cursor = form.cursor();
  if (cursor.valueBuffer("campo") == ""){
    var util = new FLUtil();
    MessageBox.warning(util.translate("scripts", "El campo no tiene datos"),
      MessageBox.Ok, MessageBox.NoButton, MessageBox.NoButton);
    return false;
  }
}
```

```
} else  
    return true;  
}
```

acceptedForm()

- Momento de ejecución: Una vez validado el formulario asociado con el script que la contenga, es decir, una vez que la función validateForm (si existe) ha devuelto true.
- Parámetros: Ninguno.
- Valor de retorno: Ninguno.
- Uso: Se utiliza para incluir una serie de acciones a realizar antes de hacer cambios en la base de datos una vez validado el formulario.

calculatedField()

- Momento de ejecución: Grabación del formulario
- Parámetros: Nombre del campo cuyo valor se desea calcular.
- Valor de retorno: Valor calculado del campo que se le pasa a la función como parámetro
- Uso: Calcula el valor de campos definidos en la base de datos como campos calculados (<calculated>true</calculated>).

calculateCounter()

- Momento de ejecución: Apertura del formulario asociado con el script que la contenga.
- Parámetros: Nombre del campo.
- Valor de retorno: Devuelve el valor del campo contador que se pasa como parámetro a la función
- Uso: esta función se utiliza para calcular los campos que en la base de datos se han definido como de tipo contador (true).

beforeCommit_nombreTabla()

- Momento de ejecución: Se ejecuta antes de producirse un commit en la tabla sufijo del nombre de la función.
- Parámetros: Cursor sobre la tabla especificada que contiene el registro a modificar.
- Valor de retorno: Si las acciones y validaciones realizadas son correctas la función devuelve true. En caso contrario devuelve false. Si la función devuelve false, la función commitBuffer() asociada al cursor devolverá false.
- Uso: Se utiliza para incluir acciones y validaciones previas a la modificación de un registro en la tabla especificada.
- Script de ubicación: En el script principal del módulo. Ejemplo: para módulo principal de facturación en el script flfactppal.qs

afterCommit_nombreTabla()

- Momento de ejecución: Se ejecuta después de producirse un commit en la tabla en la tabla sufijo del nombre de la función.
- Parámetros: Cursor sobre la tabla especificada que contiene el registro a modificar.
- Valor de retorno: Si las acciones y validaciones realizadas son correctas la función devuelve true. En caso contrario devuelve false. Si la función devuelve false, la función commitBuffer() asociada al cursor devolverá false.
- Uso: Se utiliza para incluir acciones y validaciones inmediatamente posteriores a la modificación de un registro en la tabla especificada.
- Script de ubicación: En el script principal del módulo.

recordDelBeforenombreTabla()

- Momento de ejecución: Se ejecuta antes de producirse un borrado en la tabla sufijo del nombre de la función desde el formulario asociado con el script que la contenga.
- Parámetros: Ninguno.
- Valor de retorno: Ninguno.
- Uso: Se utiliza para incluir acciones inmediatamente anteriores al borrado de un registro en la tabla especificada.
- Script de ubicación: En el script principal del módulo.

recordDelAfternombreTabla()

- Momento de ejecución: Se ejecuta después de producirse un borrado en la tabla sufijo del nombre de

la función desde el formulario asociado con el script que la contenga.

- Parámetros: Ninguno.
- Valor de retorno: Ninguno.
- Uso: Se utiliza para incluir acciones inmediatamente posteriores al borrado de un registro en la tabla especificada.
- Script de ubicación: En el script principal del módulo.

main()

- Momento de ejecución: Cuando el usuario active una acción conectada a un slot `execMainScript`, se ejecutará la función `main` contenida en el script definido en la etiqueta de dicha acción.
- Parámetros: Ninguno.
- Valor de retorno: Ninguno.
- Uso: Esta función se utiliza cuando una de las opciones de la ventana principal no se encuentra conectada con la apertura de un formulario, sino que únicamente deseamos que se ejecute el código incluido en la función `main`.

Interfaz de Scripts

El interfaz es el puente entre los scripts en lenguaje QSA y el núcleo de Abanq programado en C++ a través de las librerías Qt. Abanq trabaja con los objetos C++ que, en principio, no son accesibles desde QSA.

Para permitir el acceso a ciertos objetos desde los scripts se ha creado un conjunto de clases intermedias que sí son accesibles desde QSA y que permiten trabajar con clases de C++. Este conjunto de clases es lo que llamamos interfaz.

Por ejemplo, sabemos que podemos acceder a un cursor de una tabla para manipular sus datos desde un script. Si queremos obtener el nombre del cliente cuyo código es el 100 de la tabla clientes:

```
var cursorCliente = new FLSqlCursor("clientes");
var codCliente = 100;
cursorCliente.select("codcliente = "+ codCliente);
cursorCliente.first();
nombreCliente = cursorCliente.valueBuffer("nombre");
```

Hemos usado un objeto de la clase `FLSqlCursor`. Esta clase es accesible desde el script porque en el interfaz se ha definido la clase `FLSqlCursorInterface`, y los métodos `select` y `first` se encuentran también en el interfaz.

Importante. Muchas de las clases del interfaz se han definido con el formato `nombre_clase + Interface`. Desde los scripts accederemos a ellas sólo por el `nombre_clase`. Como hemos visto antes, la clase del interfaz es `FLSqlCursorInterface`, pero nosotros usamos `FLSqlCursor` en el script.

Un módulo de gráficos

Para practicar un poco más todo lo aprendido vamos a desarrollar un pequeño módulo de gráficos. Esto nos permitirá ver la estructura general de un módulo y la potencia de QSA que permite fácilmente aprovechar la funcionalidad de Abanq e interactuar con el sistema operativo.

Antes de comenzar sería aconsejable tener a mano los artículos sobre Abanq de los números anteriores de Septiembre y Octubre, ya que obviaremos todo lo que ya se ha explicado en ellos. También sería buena idea disponer de la documentación que se puede encontrar en la página del proyecto (<http://Abanq.org/documentacion>).

Nos basaremos en el módulo de informes de facturación.

Descargamos dicho módulo desde <http://prdownloads.sourceforge.net/Abanq/fact-informes-1.7.tar.gz>, lo guardamos en nuestro disco y comenzamos a trabajar.

Preparando el módulo

Vamos a descomprimir el módulo de informes y haremos una copia del mismo en un nuevo directorio llamado gráficos:

Ahora debemos trabajar sobre ese directorio graficos. Hemos elegido como identificador del módulo el código flfactgraf. Por lo tanto lo primero que hacemos es renombrar los siguientes ficheros

El fichero flfactgraf.mod es el que describe el módulo con la información necesaria para que la aplicación base de Abanq pueda cargarlo correctamente, debemos editarlo para que contenga exactamente esto:

```
<!DOCTYPE MODULE>
<MODULE>
  <name>flfactgraf</name>
  <alias>Gráficos</alias>
  <area>F</area>
  <areaname>Area de Facturación</areaname>
  <version>1.7</version>
  <icon>flfactgraf.xpm</icon>
  <flversion>1.7</flversion>
  <dependencias>
    <dependency>flfactinfo</dependency>
  </dependencias>
  <description>
    Gráficos
  </description>
</MODULE>
```

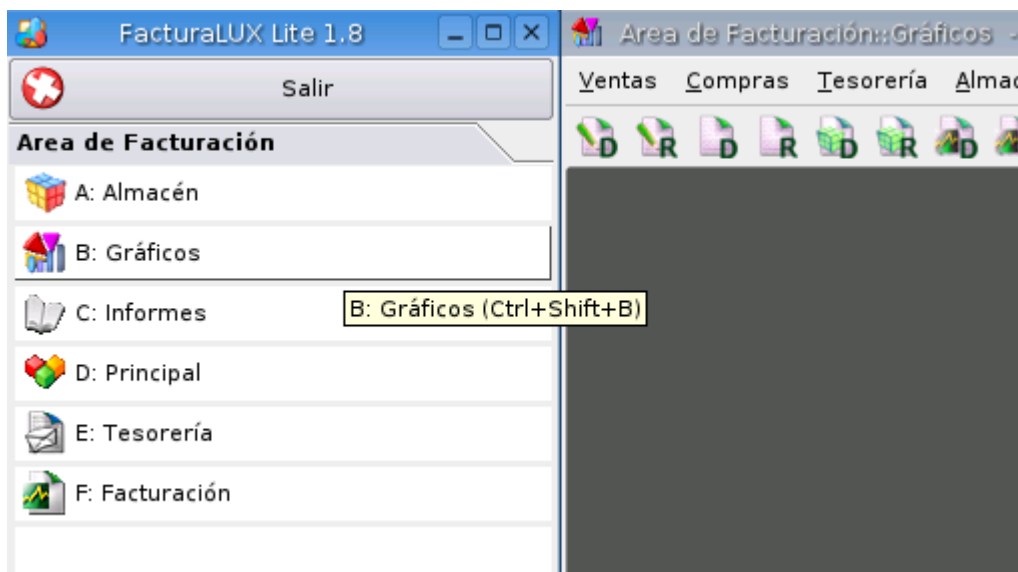
El contenido de este fichero es muy claro y se explica por sí sólo, hay que destacar que la propiedad <name> es la más importante porque marca el identificador del módulo, si olvidamos asignar esta propiedad correctamente las cosas no funcionarán correctamente.

Como ya sabemos el fichero flfactgraf.xml es el que describe las acciones, de momento no necesita ninguna modificación.

El fichero flfactgraf.xpm es el icono que se mostrará asociado al módulo, no es necesario, pero podemos sustituirlo por otro más acorde al módulo que estamos construyendo.

Y por último el fichero flfactinfo.ui es la ventana principal del módulo y que de momento tampoco vamos a modificar.

Con esto lo único que hemos conseguido es realizar una copia exacta del módulo de informes, si lo cargamos y probamos vemos que hace exactamente lo mismo que el original, ahora debemos pasar a modificarlo para que el funcionamiento sea el que explicamos en el siguiente apartado.



Ventana principal del módulo de gráficos

Funcionamiento del módulo

La solución que vamos a adoptar se basa en la idea de reutilizar la gran mayoría del módulo de informes y centrarnos solamente en crear una nueva capa de presentación para los resultados. Este sencillo enfoque nos permite ahorrar mucho trabajo y nos demuestra la facilidad de integración y reutilización entre los distintos módulos.

En líneas generales el módulo de informes de facturación funciona realizando consultas parametrizadas a la base de datos según cada tipo de documento (presupuestos, pedidos, albaranes, facturas, etc..) y a nivel de resumen o detalle. Estas consultas generan unos resultados o conjuntos de registros los cuales son procesados y presentados como los típicos informes (listados o documentos) listos para imprimir.

En nuestro módulo de gráficos podemos aprovechar todo lo que ya está implementado en el módulo de informes sobre la solicitud de parámetros al usuario y ejecución de consultas, y sólo modificar la última parte de presentación, que serán gráficos resultantes del cruce de los distintos resultados obtenidos.

Para la presentación, en vez de crear nuestra herramienta de generación de gráficos, vamos a utilizar gnuplot, que se encuentra 'de serie' en la mayoría de las distribuciones de Linux, y que podemos instalar fácilmente con las herramientas facilitadas para tal fin, o en su defecto descargándolo desde <http://www.gnuplot.info/>

Como vemos el grado de reutilización es muy alto, y el ahorro de tiempo es significativo. Esto demuestra, una vez más, que la filosofía de desarrollo del software libre es mucho más eficaz que cualquiera otra conocida, ya que permite unos grados de integración y reutilización inimaginables en otros modelos.

En definitiva, el módulo de informes y de gráficos compartirán la mayoría de los formularios, consultas y tablas. De hecho van a trabajar sobre las mismas tablas de tal forma que los parámetros que definamos para los informes se pueden utilizar para gráficos y viceversa.

Manos a la obra

Vamos a quitar todo lo que sobra del módulo de gráficos y que reutilizaremos del de informes:

- Eliminamos el directorio tables.
- Eliminamos el directorio queries.
- Eliminamos el directorio reports.
- Eliminamos el directorio translations.
- Entramos en el directorio forms y eliminamos todos los formularios (*.ui) excepto flfactgraf.ui
- Entramos en el directorio scripts y eliminamos todos los scripts (*.qs).

Para continuar y saber que debemos incluir o modificar hacemos un pequeño análisis de los scripts del módulo de informes, y observamos que todos son los asociados a los formularios maestros de cada uno de los tipos de documentos; i_masterfacturascli.qs, i_masterpedidoscli.qs, i_masteralbaranescli.qs, i_masterfacturasprov.qs, etc.. Y todos tiene un contenido muy similar, casi idéntico, que podemos resumir en una conexión del botón de imprimir a una función que recoge los parámetros de la consulta según el registro actual e invocan finalmente a una función global del objeto flfactinfo llamada lanzarInforme definida en flfactinfo.qs.

Todo indica que lo único que debemos hacer es crear una nueva función del objeto flfactgraf en el script flfactgraf.qs y que llamaremos lanzarGrafico, y posteriormente crear los scripts para los formularios maestros que nombraremos como; g_masterfacturascli.qs, g_masterpedidoscli.qs, g_masteralbaranescli.qs, g_masterfacturasprov.qs, etc... Estos formularios tendrán un contenido muy similar a sus homólogos del módulo de informes, la única diferencia significativa radica en que llamarán a la función lanzarGrafico en lugar de lanzarInforme.

Primero veamos como quedaría el script flfactgraf.qs que copiaremos en el directorio scripts y que hemos construido siguiendo como ejemplo flfactinfo.qs pero creando la presentación con gnuplot:

```
function lanzarGrafico( cursor:FLSqlCursor, nombreInforme:String, orderBy:String, groupBy:String )
{
  var q:FLSqlQuery = prepararConsultaInforme(cursor, nombreInforme, orderBy);
  var campos:Array = q.select().split( " " );
  var util:FLUtil = new FLUtil();
  var coorX:String = Input.getItem( "Coordenada X", campos, campos[0], false, "Coordenada X" );
  coorX = coorX.replace( " ", "" );
  var compX:Array = coorX.split( " " );
  var tipoX:Number = util.fieldType( compX[1], compX[0] );
}
```

```

var esFechaX:Boolean = false;
if ( tipoX == 26 )
    esFechaX = true;
var coorY:String = Input.getItem( "Coordenada Y", campos, campos[0], false, "Coordenada Y" );
coorY = coorY.replace( " ", "" );
var compY:Array = coorY.split( "." );
var tipoY:Number = util.fieldType( compY[1], compY[0] );
var esFechaY:Boolean = false;
if ( tipoY == 26 )
    esFechaY = true;
if ( q.exec() ) {
    if ( q.size() > 0 ) {
        var stdin:String, datos:String;
        var fechaInicioX:String, fechaFinX:String, fechaInicioY:String, fechaFinY:String;
        if ( esFechaX || esFechaY ) {
            q.first();
            if ( esFechaX )
                fechaInicioX = util.dateAMDtoDMA( q.value( coorX ) );
            else
                fechaInicioX = q.value( coorX );
            if ( esFechaY )
                fechaInicioY = util.dateAMDtoDMA( q.value( coorY ) );
            else
                fechaInicioY = q.value( coorY );
            datos = fechaInicioX + " " + fechaInicioY + "\n";
            q.last();
            if ( esFechaX )
                fechaFinX = util.dateAMDtoDMA( q.value( coorX ) );
            if ( esFechaY )
                fechaFinY = util.dateAMDtoDMA( q.value( coorY ) );
            q.first();
        }
        var tempX:String, tempY:String;
        while ( q.next() ) {
            if ( esFechaX )
                tempX = util.dateAMDtoDMA( q.value( coorX ) );
            else
                tempX = q.value( coorX );
            if ( esFechaY )
                tempY = util.dateAMDtoDMA( q.value( coorY ) );
            else
                tempY = q.value( coorY );
            datos += tempX + " " + tempY + "\n";
        }
        stdin =
        if ( esFechaX ) {
            stdin += "\nset xdata time";
            stdin += "\nset timefmt \"%d-%m-%Y\"";
            stdin += "\nset xrange [\"" + fechaInicioX + "\":\"" + fechaFinX + "\"]";
            stdin += "\nset format x \"%d-%m\"";
        }
        if ( esFechaY ) {
            stdin += "\nset ydata time";
            stdin += "\nset timefmt \"%d-%m-%Y\"";
            stdin += "\nset yrange [\"" + fechaInicioY + "\":\"" + fechaFinY + "\"]";
            stdin += "\nset format y \"%d-%m\"";
        }
        stdin += "\nset xlabel \"" + coorX + "\"";
        stdin += "\nset ylabel \"" + coorY + "\"";
    }
}

```

```

        stdin += "\nset grid";
        stdin += "\nplot '-' using 1:2 t " with line smoot bezier, '-' using 1:2 t " with impulses\n";
        stdin += datos;
        stdin += "\ne\n"
        stdin += datos;
        Process.execute( ["gnuplot", "-persist"], stdin);
    }
}
}
function prepararConsultaInforme( cursor:FLSqlCursor,  nombreInforme:String,  orderBy:String,
groupBy:String ):FLSqlQuery
{
    var q:FLSqlQuery = new FLSqlQuery(nombreInforme);
    var util:FLUtil = new FLUtil();
    var fieldList:String = util.nombreCampos(cursor.table());
    var cuenta:Number = parseFloat(fieldList[0]);
    var signo:String;
    var fN:String;
    var valor:String;
    var primerCriterio:Boolean = false;
    var where:String = "";
    for (var i:Number = 1; i <= cuenta; i++) {
        if (cursor.isNull(fieldList[i]))
            continue;
        signo = obtenerSigno(fieldList[i]);
        if (signo != "") {
            fN = fieldName(fieldList[i]);
            valor = cursor.valueBuffer(fieldList[i]);
            if (valor == util.translate("scripts", "Si"))
                valor = 1;
            if (valor == util.translate("scripts", "No"))
                valor = 0;
            if (valor == util.translate("scripts", "Todos"))
                valor = "";
            if (!valor.toString().isEmpty()) {
                if (primerCriterio == true)
                    where += "AND ";
                where += fN + " " + signo + " " + valor + " ";
                primerCriterio = true;
            }
        }
    }
    q.setWhere(where);
    if (q.exec() == false) {
        MessageBox.critical(util.translate("scripts", "Falló la consulta"),
            MessageBox.Ok, MessageBox.NoButton, MessageBox.NoButton);
        return q;
    } else {
        if (q.first() == false) {
            MessageBox.warning(util.translate("scripts",
                "No hay registros que cumplan los criterios de búsqueda establecidos"),
                MessageBox.Ok, MessageBox.NoButton, MessageBox.NoButton);
            return q;
        }
    }
    if (orderBy)
        q.setOrderBy(orderBy);
    return q;
}

```

```

function sqlConsultaInforme( cursor:FLSqlCursor, nombreInforme:String, orderBy:String, groupBy:String )
:String
{
    var q:FLSqlQuery = prepararConsultaInforme(cursor, nombreInforme, orderBy);
    return q.sql();
}
function obtenerSigno(s:String):String
{
    if (s.toString().charAt(1) == "_") {
        switch(s.toString().charAt(0)) {
            case "d": {
                return ">=";
            }
            case "h": {
                return "<=";
            }
            case "i": {
                return "=";
            }
        }
    }
    return
}
function fieldName(s:String):String
{
    var fN:String = "";
    var c:String;
    for (var i:Number = 2; (s.toString().charAt(i)); i++) {
        c = s.toString().charAt(i);
        if (c == "_")
            if (s.toString().charAt(i + 1) == "_") {
                fN += "_";
                i++;
            } else
                fN += "."
            else
                fN += s.toString().charAt(i);
    }
    return fN;
}

```

Si observamos el código, a parte de las funciones auxiliares que son las mismas, se ha dividido la función original lanzarInforme, en las funciones lanzarGrafico y preparaConsultaInforme, para diferenciar más claramente lo que es la consulta de la presentación de los resultados de la misma.

En la parte final de lanzarGrafico vemos un pequeño mecanismo que solicita al usuario que campo de la consulta debe tomarse como coordenada X y cual como coordenada Y, llamando finalmente al comando gnuplot con los parámetros y datos adecuados.

Para invocar a gnuplot se ha utilizado el objeto Process de QSA, muy útil y que nos ofrece total flexibilidad para interactuar con distintos comandos y herramientas del sistema operativo.

Finalmente debemos crear y habilitar los scripts para los formularios maestros siguiendo como ejemplo los del módulo de informes, por cuestiones de espacio y como el proceso es muy parecido para cada uno de ellos, solo veremos los pasos para el formulario de facturas de clientes, dejando el tratamiento para el resto como ejercicio para el lector inquieto.

Los pasos a seguir serían:

Crear el script g_masterfacturascli.qs en el directorio scripts y con el contenido:

```

function init()
{

```

```

    connect (this.child("toolButtonPrint"), "clicked()", this, "lanzar()");
}
function lanzar()
{
    var cursor:FLSqlCursor = this.cursor();
    var seleccion:String = cursor.valueBuffer("id");
    if (!seleccion)
        return;
    var nombreInforme:String = cursor.action();
    nombreInforme = nombreInforme.replace( "g_", "i_" );
    var orderBy:String = "";
    var o:String = "";
    for (var i:Number = 1; i < 3; i++) {
        o = formi_albaranescli.iface.pub_obtenerOrden(i, cursor, "facturascli");
        if (o) {
            if (orderBy == "")
                orderBy = o;
            else
                orderBy += ", " + o;
        }
    }
    flfactgraf.lanzarGrafico( cursor, nombreInforme, orderBy );
}

```

Modificar flfactinfo.xml y hacemos las modificaciones correspondientes para utilizar nuestra nueva versión del script, hay que hacer estas sustituciones:

```

<action>
    <name>i_facturascli</name>
    <alias>QT_TRANSLATE_NOOP("MetaData","Facturas de clientes")</alias>
    <description>QT_TRANSLATE_NOOP("MetaData","Informe que contiene una página
por cada factura. Se muestran los datos de las líneas de la factura, el total
de la factura y los totales de IVA, recargo de equivalencia e IRPF ")</description>
    <table>i_facturascli</table>
    <form>i_master</form>
    <formrecord>i_facturascli</formrecord>
    <scriptform>i_masterfacturascli</scriptform>
</action>

```

lo sustituimos por;

```

<action>
    <name>g_facturascli</name>
    <alias>QT_TRANSLATE_NOOP("MetaData","Facturas de clientes")</alias>
    <description>QT_TRANSLATE_NOOP("MetaData","Informe que contiene una página
por cada factura. Se muestran los datos de las líneas de la factura, el total
de la factura y los totales de IVA, recargo de equivalencia e IRPF ")</description>
    <table>i_facturascli</table>
    <form>i_master</form>
    <formrecord>i_facturascli</formrecord>
    <scriptform>g_masterfacturascli</scriptform>
</action>

```

y despues;

```

<action>
    <name>i_resfacturascli</name>
    <alias>QT_TRANSLATE_NOOP("MetaData","Resumen de facturas de clientes")</alias>
    <description>QT_TRANSLATE_NOOP("MetaData","Cada línea del informe contiene
los datos de una factura. Al final del informe se añade la línea de totales")</description>
    <table>i_facturascli</table>
    <form>i_master</form>

```

```

</formrecord>i_facturascli</formrecord>
</scriptform>i_masterfacturascli</scriptform>
</action>

```

lo sustituimos por;

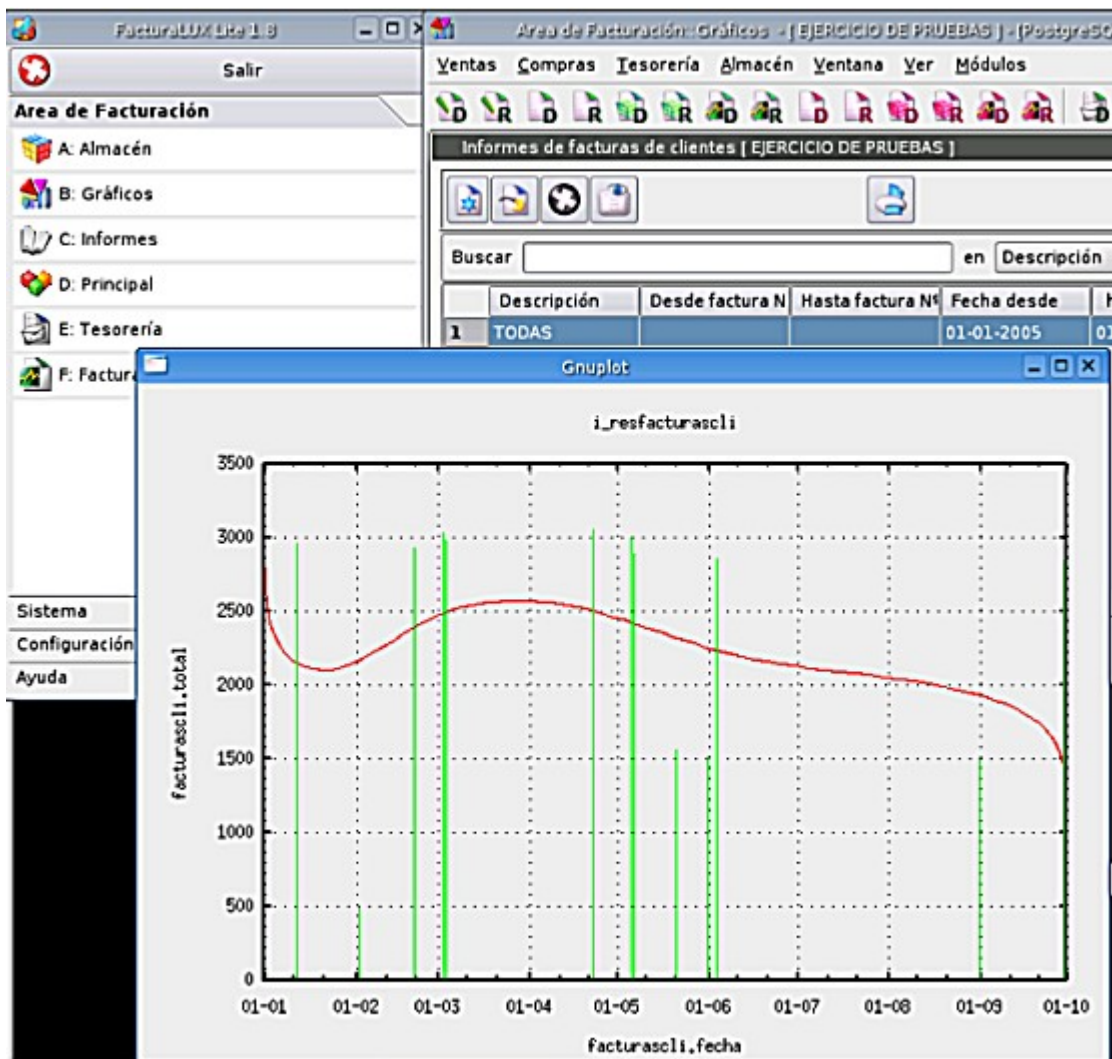
```

<action>
  <name>g_resfacturascli</name>
  <alias>QT_TRANSLATE_NOOP("MetaData","Resumen de facturas de clientes")</alias>
  <description>QT_TRANSLATE_NOOP("MetaData","Cada línea del informe contiene los datos
de una factura. Al final del informe se añade la línea de totales")</description>
  <table>i_facturascli</table>
  <form>i_master</form>
  <formrecord>i_facturascli</formrecord>
  </scriptform>g_masterfacturascli</scriptform>
</action>

```

Por último editamos con QtDesigner flfactgraf.ui y cambiamos los nombres de las acciones i_facturascli por g_facturascli, y i_resfacturascli por g_resfacturascli.

Con esto ya hemos acabado, si volvemos a cargar el módulo, creamos como ejemplo un listado resumen de la facturación anual y hacemos clic en el botón con el icono de la impresora obtendremos un resultado parecido a este, si seleccionamos como coordenada X el campo fecha y como coordenada Y el campo total:



Ventana principal del módulo de gráficos

Conclusión

A lo largo de esta serie de artículos hemos visto las capacidades reales de Abanq y que lo hacen tan atractivo para las empresas. Su carácter abierto y dinámico permite disponer de una herramienta que se puede adaptar sin límites y ajustándose a las necesidades presentes y futuras que cualquier tipo de empresa.

En cualquier implantación de un ERP ya no es solamente importante que tenga la funcionalidad que necesitamos ahora, también tiene mucho importancia, y cada vez más, que se pueda garantizar la funcionalidad que se necesitará mañana. Y preferiblemente con libertad de proveedor y huyendo de la cautividad.

En este contexto tan dinámico y cambiante que nos toca vivir, existen muy pocas soluciones que ofrezcan o puedan ofrecer lo que Abanq ya ofrece.

Actualizado el 16/02/2007