Control remoto del sintetizador de frecuencia RACAL-DANA 3101

L. Barbas, R. Bolaño, P. de Vicente

Informe Técnico IT-OAN 2007-24

Histórico de revisiones

Versión	Fecha	Autor	Cambio
1.0	03-12-2007	Laura Barbas	Creación del Informe técnico

Índice

1 Introducción

En el Centro Astronómico de Yebes se ha desarrollado un receptor de holografía para el radiotelescopio de 40m. El receptor de holografía se utiliza para observar la radiobaliza de un satélite de comunicaciones. El satélite se mantiene en una órbita geoestacionaria, pero realiza un movimiento periódico de aproximadamente un día de periodo. Durante ese movimiento la velocidad del satélite respecto al observador es variable y por tanto la frecuencia recibida se desplaza por efecto Doppler. Para que la señal se mantenga en el centro de la FI el oscilador local del receptor de holografía debe modificar su frecuencia constantemente. En este caso la frecuencia que varía es la del segundo oscilador local, un sintetizador de frecuencia controlable remotamente.

Los sintetizadores permiten la generación precisa de señales armónicas de muy alta calidad y bajo ruido, donde la amplitud y la frecuencia son seleccionables por el usuario, en muchos casos remotamente.

El receptor de holografía utiliza un sintetizador modelo 3101 de RACAL-DANA de muy bajo ruido de fase y que genera señales en el rango de frecuencia de 10kHz hasta 1,3GHz. El sintetizador RACAL-DANA se puede controlar y monitorizar remotamente utilizando un puerto GPIB.

En este informe se describe el sintetizador de frecuencia RACAL-DANA 3101 y el desarrollo del componente ACS y el cliente java para su control remoto.

2 Control de instrumentación por GPIB

GPIB (General Purpose Instrument Bus) es un bus paralelo utilizado para interconectar equipos en un entorno de instrumentación automatizado y para realizar un control remoto sobre ellos. El estándar que sigue GPIB se describe en la especificación IEEE 488. Las características de nivel físico, mecánico y eléctrico y las características funcionales básicas del bus están definidas en la especificación IEEE-488.1. La especificación IEEE-488.2 describe el nivel operativo, es decir, el protocolo básico dentro del que se encuadra el intercambio de información, los datos y las instrucciones básicas de control.

Los equipos que disponen de un puerto GPIB, para su control constan de software interno de control que interpreta los mensajes que recibe por el bus GPIB e interacciona con el firmware propio del equipo. Para poder realizar el control remoto de un instrumento a través del bus es necesario disponer de un ordenador con una tarjeta GPIB y el driver GPIB adecuadamente instalados. En la fig.1. se puede ver un esquema de un entorno de instrumentación controlado mediante GPIB.

Los equipos conectados al bus GPIB pueden estar configurados en uno de los siguientes modos de funcionamiento:

- ***Controller***: con capacidad de establecer quién envía o recibe los datos y el modo de operación del bus. En un bus sólo puede existir un equipo con capacidad de poder tomar control del bus en todo momento. Este equipo tiene exclusividad en el control de las líneas IFC (Interface Clear) y REN (Remote Enable) del bus.
- ***Talker***: con capacidad de enviar datos a otros equipos.
- ***Listener***: con capacidad de recibir datos de otros equipos.
- ***Idler***: sin ninguna capacidad respecto del bus.

Normalmente un ordenador actúa como *controller*.

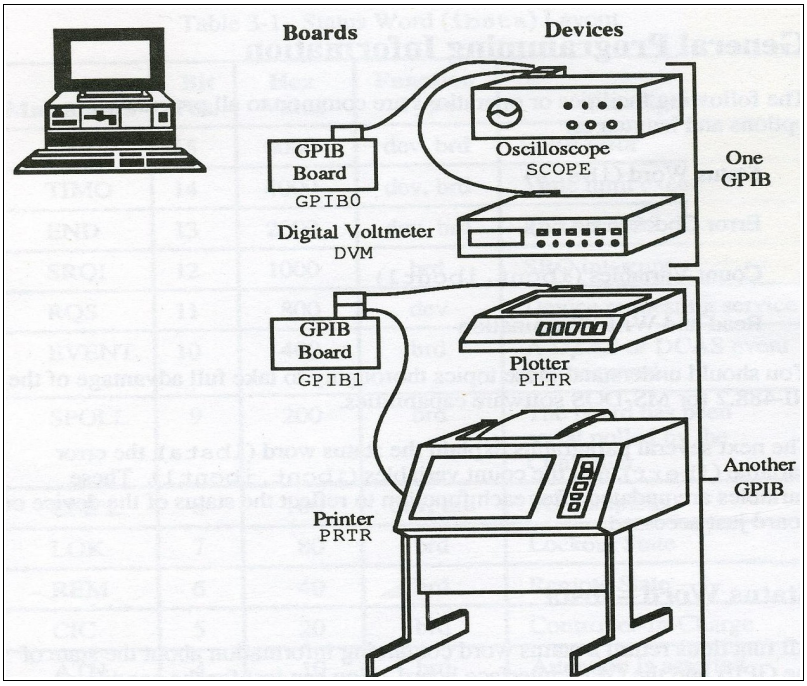

Fig.1. Entorno de instrumentación controlado mediante GPIB

El software de control de GPIB se basa en dos tipos de comandos: las rutinas y las funciones.

A. Las rutinas utilizan los procedimientos y protocolos de control descritos en la especificación IEEE-488.2. Es necesario indicar en sus parámetros el descriptor de la tarjeta GPIB, así como el descriptor del dispositivo. Algunas de las rutinas más importantes son [1]:

- `SendIFC(board%)`
- `Send(board%,address%,data$,eotmode%)`
- `Receive(board%,address%,data$,termination%)`

B. Las funciones se pueden clasificar en dos grupos dependiendo de los parámetros pasados a la función:

I. Las funciones de alto nivel o funciones de dispositivo son fáciles de utilizar y ejecutan automáticamente secuencias de comandos que manejan las operaciones de gestión del bus GPIB requeridas para realizar tareas como leer y escribir en el dispositivo (`ibwrt, ibrd`). Es necesario indicar en uno de sus parámetros el descriptor del dispositivo.

II. Las funciones de bajo nivel o funciones de tarjeta realizan operaciones básicas GPIB, y por tanto ofrecen la flexibilidad necesaria para resolver las necesidades de nuestras aplicaciones. Estas funciones acceden directamente a la tarjeta de interfaz de GPIB y requieren como parámetro el descriptor de la tarjeta. Algunas de las funciones que se han utilizado para el sintetizador de frecuencia son:
   - `ibsre(ud, v)`
   - `ibloc(ud)`

Una misma función puede emplearse de dos modos diferentes. Por ejemplo, la función `ibwrt` se puede usar:

- Como función de dispositivo: `ibwrt(device%,data$).`

El dispositivo es direccionado como *listener* y la tarjeta es direccionada como *talk*.

- Como función de tarjeta: `ibwrt(board%,data$).`
  Este comando intenta escribir a un dispositivo conectado a la tarjeta de descriptor "`board%`". El dispositivo habrá sido direccionado con otro comando previo.

El software de control de GPIB ofrece además información general, la palabra de estado `ibsta` que contiene información del estado del bus GPIB y de la tarjeta de interfaz GPIB y la variable de error `iberr` que indica cualquier error ocurrido mediante una serie de códigos previamente establecidos.

Se puede obtener más información acerca del software de control de GPIB en [1].

3 Descripción del sintetizador RACAL-DANA

El sintetizador de frecuencia RACAL-DANA dispone de una interfaz IEEE-488-GPIB que permite controlar todas las funciones del sintetizador remotamente a través del bus GPIB[2].

En la fig.2. podemos observar parte del panel trasero del RACAL-DANA en el que hay dos zonas destacables, el conector GPIB al que se conecta el bus GPIB y la interfaz de direccionamiento. Los cinco primeros conmutadores de esta interfaz permiten seleccionar una de las 31 direcciones posibles para el equipo (el conmutador superior corresponde al bit menos significativo). Cuando el conmutador está en el lado derecho representa un 1 lógico. Mediante el sexto conmutador se indica si el equipo está o no en modo sólo escucha.

El RACAL-DANA permite cuatro modos de funcionamiento a través de GPIB y dispone de los indicadores correspondientes en el panel frontal:

- **REMOTE**: indica que el *controller* del bus GPIB ha establecido el equipo en el modo de operación remoto.
- **LISTEN**: indica que el *controller* del bus GPIB ha programado el equipo como "listener" o que está establecido a "1"el conmutador de sólo escucha.
- **TALK**: indica que el *controller* del bus GPIB ha programado el equipo como "talker".
- **SRQ**: indica que el equipo está transmitiendo una interrupción al *controller* del bus GPIB.

Para realizar el control remoto del sintetizador de frecuencia RACAL-DANA se construye un sistema sobre el bus GPIB que dispone de dos tipos de dispositivos, el ordenador y el equipo que se desea controlar remotamente (RACAL-DANA). El sistema está configurado con un único *controller* (el ordenador), por tanto las transferencias de datos posibles son desde el *controller* al equipo en modo comando y datos, y del equipo al *controller* sólo en modo datos.

- Cuando el *controller* (ordenador) ejecuta un comando "talk" con la dirección del RACAL-DANA éste pasa a activarse como *talker*, y a partir de este instante tiene capacidad para enviar datos por el bus.
- Cuando el *controller* (ordenador) ejecuta un comando "listen" con la dirección del RACAL-DANA éste pasa a activarse como *listener*, y a partir de este momento recibe y lee cada uno de los datos que le envíen por el bus.

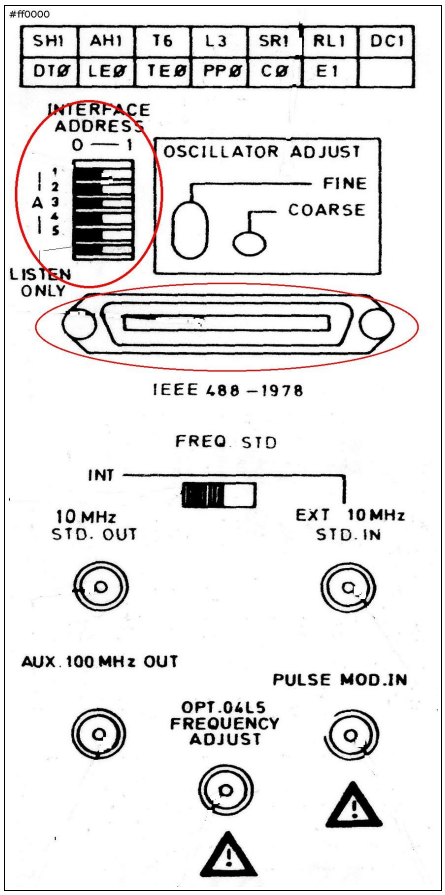

Fig. 2. Panel trasero del sintetizador

El sintetizador de frecuencia RACAL-DANA se distribuye desde fábrica con la dirección decimal 19 y es la que hemos utilizado. Si el sintetizador está en modo sólo escucha, la dirección indicada mediante los conmutadores del panel trasero es irrelevante y el equipo sólo acepta los comandos que le llegan a través del bus GPIB, pero no puede ser direccionado para enviar datos. En nuestro caso la configuración del equipo debe permitir tanto enviar como recibir datos por lo que el sexto conmutador ha de estar a la izquierda. La interfaz de direccionamiento queda configurada de la siguiente manera:

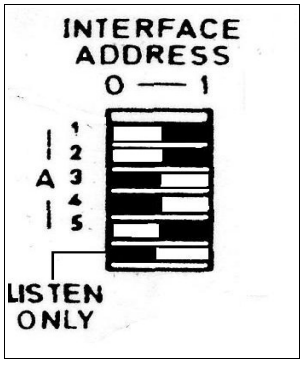

Fig. 3. Dirección seleccionada

El software de control de GPIB permite desarrollar programas en distintos lenguajes de programación, en nuestro caso hemos seleccionado el lenguaje C++ para desarrollar el programa de control remoto del sintetizador.

Los comandos GPIB más importantes utilizados para el control remoto del RACAL-DANA son [2]:

PARÁMETRO	COMANDO	OPERACIÓN
Frequency	FQ<valor><unidades>	Permite establecer la frecuencia de la señal RF.
Amplitude	AP<valor><unidades>	Permite establecer la frecuencia de la señal RF.
Initialization	IP	Pone el sintetizador en su estado inicial, estableciéndose ciertos valores predeterminados. Por ejemplo: frecuencia = 100 MHz, Amplitud = 0 dBm, salida RF = ON, etc.).
Instrument status	IS	Provoca que el sintetizador envíe una cadena de 27 bytes de la que, entre otras cosas, se pueden extraer los códigos de error que hayan podido ocurrir.
Instrument data	ID	Hace que el sintetizador envíe una cadena de 163 caracteres ASCII de la que se puede obtener la configuración del sintetizador, es decir, frecuencia, amplitud, modo de funcionamiento GPIB, etc.

4 Componente ACS para el sintetizador

4.1 Desarrollo del componente

Todo el desarrollo del componente se ha realizado en un directorio denominado `RacalDana`.

El diseño del componente requiere identificar qué parámetros del sintetizador se van a definir como propiedades y qué funciones se van a definir como métodos, para establecer así la interfaz del sintetizador. El archivo `IDL` que contiene la declaración de estas propiedades y métodos, se ha denominado `oanRacalDana3101.idl` y se encuentra en el subdirectorio `idl`. En este subdirectorio también se encuentra el archivo `racalDanaError.xml` que define las excepciones lanzadas por el componente en caso de error.

El generador de código `acsGenerator` utiliza el archivo IDL para generar las plantillas en C++ a partir de las que se desarrolla el código del componente. El generador, en su forma más sencilla, se usa así:

```
> cd acsGenerator/
> acsGenerator --workdir  ~/aries21/RacalDana/
~/aries21/RacalDana/idl/oanRacalDana3101.idl
```

El generador de código crea los directorios *include, src* y *config*. Los dos primeros contienen numerosos archivos de cabecera e implementación que permiten aislar la implementación del código del interfaz del componente. Esta característica que es muy útil si se desea rehacer la interfaz (modificar el IDL) se hace a costa de multiplicar el número de ficheros. Pensamos que una vez que se ha utilizado el interfaz es mucho más claro mantener un sólo archivo para la implementación y la cabecera. De acuerdo con esta filosofía la estructura de directorios queda así:

```
|-- config
|   `-- CDB
|       `-- schemas
|           `-- RACALDANA3101.xsd
|-- idl
|   |-- ACSErrTypeCommon.idl
|   |-- ACSErrTypeCommon.xml
|   |-- oanRacalDana3101.idl
|   |-- racalDanaError.idl
|   `-- racalDanaError.xml
|-- include
|   |-- DevIOAmp.h
|   |-- DevIOData.h
|   |-- DevIOFreq.h
|   |-- DevIOStat.h
|   |-- oanExport.h
|   |-- oanRacalDana3101Impl.h
|   `-- rd3101.h
|-- src
|   |-- Makefile
|   |-- alma
|   |   `-- oanRacalDana3101
|   |       |-- CBdoubleRDAmp.java
|   |       |-- CBdoubleRDFreq.java
|   |       |-- RacalDanaClient.java
|   |       |-- RacalDanaGUI.java
|   |       `-- images
|   |           |-- down.png
|   |           `-- up.png
|   |-- oanRacalDana3101Impl.cpp
|   |-- rd3101.cpp
```

En el subdirectorio `src` se ha creado una clase C++ que implementa la funcionalidad del sintetizador (`rd3101.cpp`) mediante GPIB. El fichero de cabecera correspondiente se sitúa en el subdirectorio `include` (`rd3101.h`). Los `DevIOs` se desarrollarán también en este último subdirectorio.

Por último es necesario modificar el archivo `Makefile` para que utilice los archivos adecuados y las excepciones. El `Makefile` queda de la siguiente manera:

1) Es necesario incluir la biblioteca de GPIB:

```
# additional include and library search paths
USER_INC = -I$(QTDIR)/include -I/usr/local/include/sys
USER_LIB = -lACE \
     ...
    -L/usr/local/lib \
    -lgpibapi
```

2) Se indican los ficheros de cabecera. Sólo hay que añadir aquellos que sean necesarios para otras aplicaciones o componentes.

```
#
# Includes (.h) files (public only)
# ---------------------------------
INCLUDES = oanRacalDana3101Impl.h
```

3) Se introduce el nombre asociado al esquema del componente:

```
#
# Configuration Database Files
# ----------------------------
CDB_SCHEMAS = RACALDANA3101
```

4) Se introduce el nombre del fichero IDL:

```
#
# IDL Files and flags
#
IDL_FILES = oanRacalDana3101
```

4.2 Descripción del componente. Clase C++

Los ficheros que contienen el código se han denominado `rd3101.h` y `rd3101.cpp`, en ellos están, respectivamente, las declaraciones y definiciones de las variables y funciones utilizadas para el control remoto del RACAL-DANA.

La parte inicial del código permite establecer el modo de funcionamiento del sintetizador de frecuencia, que ha de ser un modo de funcionamiento remoto:

```
 ...

SendIFC(m_boardID);
if (ibsta & ERR)
  throw racalDanaError::ErrorSendCommandExImpl(__FILE__,__LINE__, "RD3101::RD3101");

if(ibsre(m_boardID,1) & ERR)
```

```
  throw racalDanaError::CouldntEnableRemoteExImpl(__FILE__,__LINE__,"RD3101::RD3101");

if(ibloc(m_boardID) & ERR)
   throw racalDanaError::CouldntEnableLocalExImpl(__FILE__,__LINE__,"RD3101::RD3101");

 ...
```

En primer lugar se envía a la tarjeta de interfaz GPIB el comando `SendIFC(m_boardID)`que permite inicializar las funciones de todos los dispositivos conectado a dicha tarjeta (en este caso únicamente el RACAL-DANA). En segundo lugar se utiliza la función `ibsre(m_boardID,1)` que permite poner todos los dispositivos conectados al bus en modo remoto (el parámetro "1" indica la activación de la línea REN del bus). Los dispositivos no estarán realmente en modo remoto hasta la siguiente vez que sean direccionados por el *controller* del bus. El *controller* es el que tiene la exclusividad en el control de las líneas REN e IFC del bus (ver pág. 1). Finalmente se utiliza la función `ibloc(m_boardID)` para poner la tarjeta de nuevo en modo local.

Una vez que se tiene configurados adecuadamente el bus, la tarjeta y el sintetizador de frecuencia, se puede enviar u obtener datos del sintetizador de frecuencia RACAL-DANA mediante las rutinas GPIB `Send` y `Receive`.

El sintetizador de frecuencia dispone de comandos para establecer los valores de frecuencia, amplitud, etc. a través de GPIB. Así mismo dispone de una serie de comandos que nos permiten obtener su configuración y su estado. Estos comandos se introducen como parámetro en las rutinas `Send` o `Receive`. Por ejemplo:

```
Send(dir_boardID,dir_devID,comando,strlen(comando),NLend);
```

En los ficheros del código desarrollado es necesario indicar la biblioteca en la que están definidas las rutinas, funciones y variables propias de GPIB, mediante la directiva:

```
#include <ugpib.h>
```

4.3 Excepciones

Para la definición y tratamiento de excepciones se utiliza el sistema de error del ACS. Este sistema de error queda completamente definido en el documento *"ACS Error System"* que podemos encontrar en las páginas de documentación de ACS [5].

Las excepciones son situaciones anómalas que tienen lugar durante la ejecución de un programa debido, por ejemplo, a la ocurrencia de algún tipo de error. El sistema de errores de ACS precisa que los errores se definan en un fichero xml con el código de cada error, su nombre y descripción.

```
<?xml version="1.0" encoding="UTF-8"?>
<Type xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="ACSError.xsd" name="racalDanaError"
       type="2001" _prefix="alma">
 <Code name="ActionOK" shortDescription="Action performed" description="Action
      performed"/>
<ErrorCode name="ErrorSendCommand" shortDescription="Error Send Command"
                                          description="Error Send Command"/>
<ErrorCode name="ErrorReceiveCommand" shortDescription="Error Receive Command"
                                             description="Error Receive Command"/>
<ErrorCode name="CouldntEnableRemote" shortDescription="Component couldn't be enabled
          remote" description="CouldntEnabledLocal"/>
<ErrorCode name="CouldntEnableLocal" shortDescription="Component couldn't be enabled
          local" description="Component couldn't be enabled local"/>
 </Type>
```

Todos los errores que se pueden producir en el componente del RACAL-DANA pertenecen al tipo `racalDanaError`, que está formado por los siguientes códigos de error:

- Error al enviar un comando al sintetizador: Error Send
- Error al recibir datos del sintetizador: Error Receive
- Error al activar control remoto: Error Enable Remote
- Error al activar control local: Error Enable Local

Otro error que se puede producir en el sintetizador de frecuencia es que los datos introducidos para la frecuencia o la amplitud excedan los límites permitidos. En este caso se ha utilizado un código de error definido en ACS, en lugar de definir uno nuevo. El código de error es:

- Fuera de rango: Out of Bounds

y está incluido en el tipo de error `ACSErrTypeCommon`.

El `Makefile` genera automáticamente el código necesario para el manejo de errores y excepciones tanto para el IDL como para todos los lenguajes de programación soportados por ACS. Para ello, en la etiqueta `ACSERRDEF`, hay que añadir los nombres de los ficheros `xml` que contienen los tipos de error.

```
#
# ACS Error definition
#
ACSERRDEF = racalDanaError ACSErrTypeCommon
```

Tras ejecutar el `Makefile` se crearán dos nuevos ficheros `ACSErrTypeCommon.idl` y `racalDanaError.idl`. Ambos se incluirán en el `idl` del componente (`oanRacalDana3101.idl`) para utilizar las excepciones:

```
#include <acserr.idl>
#include <racalDanaError.idl>
...
void reset() raises(racalDanaError::ErrorSendCommandEx);
```

Además para cada tipo de error se genera un fichero de cabecera de C++, con el nombre del tipo de error, en el que quedan declaradas todas las excepciones. En este caso `ACSErrTypeCommon.h` y `racalDanaError.h`. Estos .h se deben incluir en aquellos ficheros que lanzan/capturan excepciones. En particular será necesario incluirlos en `rd3101.h`:

```
#include <ACSErrTypeCommon.h>
#include <racalDanaError.h>
...
double Amplitude() throw (racalDanaError::racalDanaErrorExImpl &);
char *InstData() throw (racalDanaError::ErrorSendCommandExImpl &,
                        racalDanaError::ErrorReceiveCommandExImpl &);
void Initialize() throw (racalDanaError::ErrorSendCommandExImpl &);
```

Todas la excepciones están definidas en los correspondientes ficheros .cpp: `ACSErrTypeCommon.cpp` y `racalDanaError.cpp`. Para cada tipo de error se genera una clase base con el mismo nombre del tipo de error seguido del sufijo "`ExImpl`". La clase base en este caso se llama `racalDanaErrorExImpl`. Además por cada código de error también se genera una clase, siguiendo la misma nomenclatura, que hereda de la clase base anterior: `ErrorSendCommandExImpl`, `ErrorReceiveCommandExImpl`, `CouldntEnableRemoteExImpl`, etc.

Todas las excepciones generadas se utilizan de manera local, es decir, se lanzan y se capturan en el componente mediante los bloques `try/catch`, según el método convencional de tratamiento de excepciones en C++.

Las excepciones lanzadas en el componente también pueden ser capturadas en el cliente, de modo que éste detecte si se ha producido un error. Este es el caso del método `reset()` definido en el IDL. Para poder lanzar una excepción remotamente hay que especificar el método previamente en el IDL:

```
void reset() raises(racalDanaError::ErrorSendCommandEx);
```

En la clase de implementación del componente `RacalDana3101` está definida la función `reset()` que realiza una llamada a la función `Initialize()` de la clase RD3101. La función `Initialize()` permite reinicializar el sintetizador de frecuencia por medio de GPIB. Si al enviar el comando correspondiente por el bus GPIB se produce un error, esta función lanzará una excepción.

```
void RD3101::Initialize() throw (racalDanaError::ErrorSendCommandExImpl &)
{
        //Initializes the instrument

        //Addresses the device (m_devID) as listener, then writes data (IP) onto the bus
        Send(m_boardID,m_devID,(void *)"IP",2L,NLend);
        if(ibsta & ERR)
          throw racalDanaError::ErrorSendCommandExImpl(__FILE__,
                                                       __LINE__,"RD3101::Initialize");
}
```

El método `reset()` si captura una excepción del tipo `ErrorSendCommandExImpl` de manera local, la lanza remotamente utilizando la función `get<NombreRemotoExcepción>`:

```
void RacalDana3101::reset() throw(CORBA::SystemException,
racalDanaError::ErrorSendCommandEx)
{
        char msg[100];
        try{
                ACS_SHORT_LOG ((LM_INFO,"Initializing the component..."));
                m_racalDana_p->Initialize();
        }
        catch(racalDanaError::ErrorSendCommandExImpl &excep) {
          ...

                ACS_SHORT_LOG ((LM_ERROR,msg));
           throw racalDanaError::ErrorSendCommandExImpl(excep, __FILE__, __LINE__,
                                "RacalDana3101::initialize").getErrorSendCommandEx();
        }
}
```

Para que el cliente pueda capturar las excepciones es necesario incluir los paquetes en los que se definen:

```
import alma.ACSErr.racalDanaError;
import alma.racalDanaError.ErrorSendCommandEx;
```

y posteriormente implementar el bloque `try/catch` con el nombre de la excepción que puede ser capturada:

```
public void init()
  {
    try
    {
      rd.reset();
    }
    catch(ErrorSendCommandEx e)
    {
            ...
    }
```

4.4 IDL

Como se ha comentado anterirormente el archivo IDL contiene la declaración de las propiedades y los métodos del componente. Este archivo se encuentra en el directorio `idl` y se ha denominado `oanRacalDana3101.idl`.

4.4.1 Propiedades

El estado del sintetizador se describe básicamente por la frecuencia y amplitud de la señal que genera. En el RACAL-DANA la frecuencia puede tomar valores de 10 kHz hasta 1300 Mhz con una resolución de 0.5 Hz. La amplitud puede tomar valores de +19dBm hasta 0 dBm (2 V – 224 mV) con una resolución de 0.1 dBm (3 dígitos en voltios). La frecuencia y la amplitud se considerarán propiedades reales (variables tipo `double`) de lectura y escritura. Los rangos de valores quedarán especificados mediante características como veremos más adelante.

- `ACS::RWdouble frequency;`
- `ACS::RWdouble amplitude;`

Cuando el control y monitorización se hacen a través del GPIB, el sintetizador de frecuencia genera datos de salida con el estado de las operaciones y con la configuración del instrumento. Estos datos los presenta en dos cadenas de bytes en las que cada byte representa una información determinada. Las cadenas se denominan “instrument-status” e “instrument-data” respectivamente, y también se consideran propiedades del sintetizador, porque ofrecen información de su estado. Como el usuario no puede modificar su valor se corresponden con propiedades de sólo lectura.

- `ACS::ROstring instrumentStatus;`
- `ACS::ROstring instrumentData;`

4.4.2 Métodos

El sintetizador dispone de un botón en el panel frontal denominado “Initialize” y una instrucción para control por GPIB denominada “Initialization” que permite reiniciarlo en cualquier instante. Se ha defino el siguiente método:

```
void reset() raises(racalDanaError::ErrorSendCommandEx);
```

El código completo del IDL se muestra a continuación:

```
#include <baci.idl>
#include <acserr.idl>
#include <racalDanaError.idl>
#pragma prefix "alma"

module oanRacalDana3101
{
        interface RacalDana3101: ACS::CharacteristicComponent          // @device
```

```
        {
                readonly attribute ACS::RWdouble frequency;        // @Properties
                readonly attribute ACS::RWdouble amplitude;
                readonly attribute ACS::ROstring instrumentStatus;
                readonly attribute ACS::ROstring instrumentData;
                void reset() raises(racalDanaError::ErrorSendCommandEx);
        };
};
```

El módulo `module oanRacalDana3101` indica, según las especificaciones de la ESO, que es una parte de código que reúne una funcionalidad específica (igual que un paquete JAVA o un espacio de nombres en C++).

La interfaz `interface RacalDana3101` es una colección de métodos y propiedades específicas para un objeto (como una clase en C++) . Un módulo puede contener varias interfaces con algunas características comunes.

Para todas las propiedades se han utilizado tipos de datos derivados definidos por ACS en `baci.idl`. En el caso de la frecuencia y amplitud se ha utilizado `RWdouble` ya que se tiene que permitir tanto lectura como escritura y ambas toman valores reales. Por el contrario las propiedades `instrumentStatus` e `instrumentData` únicamente son de lectura y se corresponden con una cadena de caracteres (bytes), por esta razón son del tipo `ROstring`. Por último el método `reset()` no admite parámetros ni devuelve ningún valor, únicamente lanza una excepción en caso de error.

4.5 Implementación del componente

Para implementar el componente se utilizan los ficheros `rd3101.cpp`, `oanRacalDana3101Impl.cpp` y los DevIOs.

Los ficheros de cabecera (.h) y los DevIOs se sitúan en el directorio `include`. A partir del IDL se obtiene el patrón del fichero `oanRacalDana3101Impl.h` que contiene la declaración de cada una de las propiedades y métodos, pero es necesario añadir código para implementar la funcionalidad deseada. Para cada una de las propiedades utilizadas es necesario añadir, utilizando la directiva `#include`, el tipo derivado correspondiente (`baciRWdouble.h`, `baciROstring.h`).

Los DevIOs se asocian a las propiedades y permiten establecer u obtener los valores de cada propiedad. La funcionalidad del DevIO se define e implementa en un fichero .h. Todos los DevIOs definidos heredan de la clase `DevIO<T>` lo que permite reimplementar sus métodos `read` y `write`, que son los encargados de obtener o establecer valores en las propiedades, respectivamente. Mediante el `template <T>` se indica el tipo del DevIO.

Para el sintetizador de frecuencia RACAL-DANA se han definido cuatro DevIOs, uno por cada propiedad:

- Frecuencia, propiedad tipo double de lectura y escritura:
  - Fichero `DevIOFreq.h`
  - Clase `class DevIOFreq : public DevIO<CORBA::Double>`
  - Reimplementa los métodos `read` y `write`
- Amplitud, propiedad tipo double de lectura y escritura:
  - Fichero `DevIOAmp.h`

- Clase `class DevIOAmp : public DevIO<CORBA::Double>`
- Reimplementa los métodos `read` y `write`

• Estado, propiedad tipo string de sólo lectura:
- Fichero `DevIOStat.h`
- Clase `class DevIOStat : public DevIO<ACE_CString>`
- Reimplementa el método `read`

• Datos, propiedad tipo string de sólo lectura:
- Fichero `DevIOData.h`
- Clase `class DevIOData : public DevIO<ACE_CString>`
- Reimplementa el método `read`

Los DevIOs toman como referencia un objeto de la clase `RD3101` de modo que tienen acceso a todas sus funciones. Por tanto los métodos `read` y `write` de cada propiedad llaman a las funciones definidas en el fichero `rd3101.cpp`, que se encargan del control remoto GPIB.

El método `initialize()` de la clase `RacalDana3101Impl` instancia los DevIOs y crea las propiedades. Las propiedades obtienen su valor predeterminado tras consultar la base de datos (`RACAL_1.xml`) empleando el método `write()` de los DevIOs.

La implementación de los métodos definidos en el IDL también se realiza en esta clase `RacalDana3101Impl`. En el método `initialize()` se crea un objeto de la clase `RD3101` para tener acceso a todas sus funciones públicas. Cada método realiza una llamada a la función de la clase C++ `RD3101` correspondiente.

4.6 Archivo CDB

El directorio `config` contiene un subdirectorio denominado CDB donde está definido el esquema XML del componente. El esquema RACALDANA3101.xsd describe la composición del componente y se utiliza para validar cada una de las instancias XML del componente. (Puede haber más de una instancia de un mismo componente).

```
.
`-- CDB
    `-- schemas
        `-- RACALDANA3101.xsd
```

El esquema del componente RACAL-DANA no define las características que se asocian al componente o a sus propiedades, pero sí que incluye, mediante la directiva `import`, características comunes a algunos tipos definidos por ACS como `Rwdouble` o `RWstring`.

La siguiente estructura de directorios muestra la distribución de la base de datos.

```
.
`-- CDB
    |
    `-- alma
        `-- RACAL_1
            `-- RACAL_1.xml
```

Las características asociadas a cada una de las propiedades del sintetizador de frecuencia son particulares para cada instancia XML del componente, por lo que para cada una de ellas se definen en un fichero xml. En nuestro caso la instancia que hemos utilizado se denomina “RACAL_1” y el fichero que contiene las características de cada una de las propiedades del sintetizador de frecuencia es `RACAL_1.xml`. En este fichero se especifican, por ejemplo, los valores máximos y mínimos de frecuencia y de amplitud.

Cada instancia del componente se almacena en la base de datos, en particular en el fichero `Components.xml` que se encuentra dentro del directorio CDB/MACI:

```
<?xml version="1.0" encoding="utf-8"?>
<Components  xmlns="urn:schemas-cosylab-com:Components:1.0"
       xmlns:cdb="urn:schemas-cosylab-com:CDB:1.0"
       xmlns:baci="urn:schemas-cosylab-com:BACI:1.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

       <_ Name="RACAL_1"
          Code="oanRacalDana3101"
          Type="IDL:alma/oanRacalDana3101/RacalDana3101:1.0"
          Container="OCholo"/>

</Components>
```

En este archivo se indica el contenedor en el que se ha de ejecutar la instancia del componente. El contenedor se denomina, en este caso, `OCholo.`

5 Cliente Java para el sintetizador

Para desarrollar el cliente se ha seleccionado el lenguaje de programación JAVA, pues una de sus principales ventajas es ser independiente de la plataforma de desarrollo, lo que permitirá la portabilidad del cliente. Desde el punto de vista de las interfaces gráficas en JAVA existen, dentro de su biblioteca, clases gráficas como `awt` y `swing`, que permiten crear objetos gráficos comunes altamente configurables y con una arquitectura independiente de la plataforma.

Se ha desarrollado un cliente Java para el control remoto del sintetizador RACAL-DANA. El cliente implementa una interfaz gráfica que permite interactuar con el sintetizador de un modo sencillo.

Se ha creado la siguiente estructura de directorios dentro del directorio `src`:

```
|-- alma
|   `-- oanRacalDana3101
|       |-- CBdoubleRDAmp.java
|       |-- CBdoubleRDFreq.java
|       |-- RacalDanaClient.java
|       |-- RacalDanaGUI.java
|       `-- images
|           |-- down.png
|           `-- up.png
```

Todas las clases definidas formarán parte del mismo paquete JAVA, denominado `package alma.oanRacalDana3101` siguiendo el formato de la estructura de directorios.

La compilación de los ficheros y la generación del paquete `RDPanel.jar` que incluye todas clases se consigue añadiendo las siguientes líneas en el archivo `Makefile`:

```
#
# Jarfiles and their directories
#
JARFILES= RDPanel
RDPanel_DIRS= alma
```

El directorio `alma` debe estar en el mismo directorio que el archivo `Makefile`.

La clase `RacalDanaClient` define el cliente y hereda de `ComponentClient`. En el fichero `RacalDanaClient.java` se importan los paquetes con las clases que permiten el acceso al componente del sintetizador de frecuencia, y por tanto a sus propiedades y métodos:

```
import alma.oanRacalDana3101.*;
import alma.acs.component.client.ComponentClient;
```

En el constructor se obtiene una referencia a la instancia (RACAL_1) del componente y se define una variable para cada una de las propiedades del mismo:

```
  private RacalDanaGUI rdGUI;
  private RacalDana3101 rd;

  private RWdouble freq;
  private RWdouble amp;
  private ROstring status;
  private ROstring data;
  private Monitordouble monFreq;
  private Monitordouble monAmp;

public RacalDanaClient(String managerLoc, String clientName) throws Exception
  {
    super(null, managerLoc, clientName);

    this.rd =
RacalDana3101Helper.narrow(this.getContainerServices().getComponent("RACAL_1"));
    this.rdGUI = new RacalDanaGUI(this);

    freq = rd.frequency();
    CBdoubleRDFreq cbRDFreq = new CBdoubleRDFreq(rdGUI);
    CBDescIn cbDescFreq = new CBDescIn();
    CBdouble cbFreq =
CBdoubleHelper.narrow(getContainerServices().activateOffShoot(cbRDFreq));
    monFreq = freq.create_monitor(cbFreq, cbDescFreq);
    monFreq.set_timer_trigger(100000000); // 10 secs.

    amp = rd.amplitude();
    CBdoubleRDAmp cbRDAmp = new CBdoubleRDAmp(rdGUI);
    CBDescIn cbDescAmp = new CBDescIn();
    CBdouble cbAmp =
CBdoubleHelper.narrow(getContainerServices().activateOffShoot(cbRDAmp));
    monAmp = amp.create_monitor(cbAmp, cbDescAmp);
    monAmp.set_timer_trigger(100000000); // 10 secs.

    data = rd.instrumentData();
    status = rd.instrumentStatus();

    rdGUI.show();
  }
```

La clase `RacalDanaClient` contiene el método principal `main`, y por tanto será la que permita la ejecución del cliente:

```
public static void main(String[] args) throws Exception
{
```

```
          RacalDanaClient rdClient = new RacalDanaClient
                                     "corbaloc::ariestest.oan.es:3000/Manager",
                                     "RacalDanaClient");

       }
```

Para las propiedades de frecuencia y amplitud se define un *monitor*. El monitor permite obtener los valores de frecuencia y amplitud del sintetizador periódicamente, de modo que se puede detectar si han sido modificados desde el panel frontal. La frecuencia y la amplitud se monitorizan cada 10 segundos.

Las clases que permiten monitorizar la frecuencia y la amplitud son similares y heredan de `CBdoublePOA`. A continuación se muestra el código correspondiente a la monitorización de amplitud:

```
package alma.oanRacalDana3101;

import alma.ACS.CBdoublePOA;
import alma.ACS.CBDescOut;
import alma.ACSErr.Completion;

public class CBdoubleRDAmp extends CBdoublePOA
{
  private RacalDanaGUI rdGUI;

  public CBdoubleRDAmp(RacalDanaGUI rdg)
  {
      rdGUI = rdg;
  }

  public void working(double value, Completion completion, CBDescOut desc)
  {
     rdGUI.setGUIAmplitude(value);
  }

  public void done(double value, Completion completion, CBDescOut desc)
  {
     rdGUI.setGUIAmplitude(value);
  }

  public boolean negotiate(long myLong, CBDescOut desc)
  {
    return true;
  }
}
```

El constructor de esta clase recibe como parámetro una referencia a la interfaz gráfica (`rdGUI`) para permitir establecer en la interfaz los valores obtenidos con la monitorización .

Para establecer y obtener valores en las propiedades del componente, y por tanto en el propio RACAL-DANA están las funciones `set_sync` y `get_sync`. La función `set_sync` sólo está disponible en aquellas propiedades que permiten escritura (como son la amplitud y la frecuencia).

La interfaz gráfica queda definida en la clase `RacalDanaGUI`. Se crea un objeto de esta clase y se hace una llamada a la función `show()` para permitir su visualización. La clase `RacalDanaGUI` recibe como parámetro el objeto `RacalDanaClient` para poder acceder a todas las propiedades del componente. Las variables definidas en `RacalDanaClient` para las propiedades son privadas, por lo que es necesario definir funciones públicas que contengan los métodos `set_sync` y `get_sync` de cada propiedad, y así pueden ser llamadas desde `RacalDanaGUI`.

```
...

 public void setFrequency(double val)
 {
   freq.set_sync(val);
 }
 public double getFrequency()
 {
    return freq.get_sync(ch);
 }
...
```

La clase `RacalDanaGUI` se encarga del aspecto de la interfaz gráfica, que es el que se muestra en la figura siguiente.

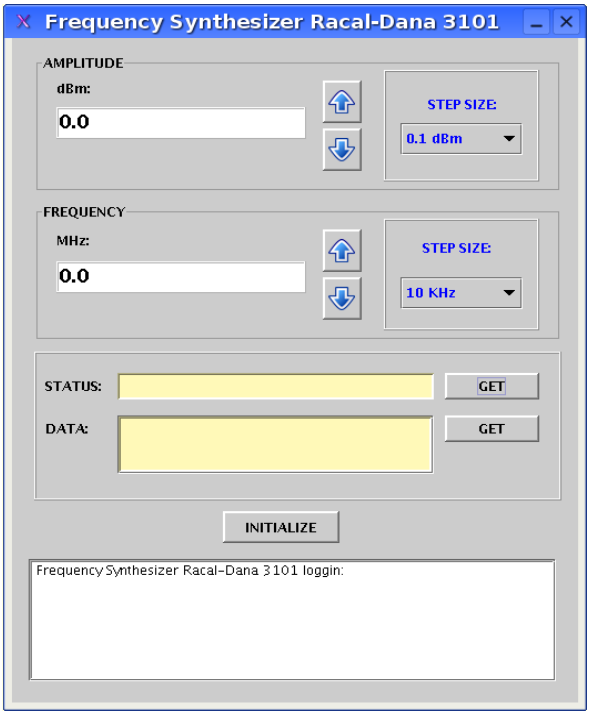

Fig. 4. Interfaz Gráfica del Racal-Dana

La clase `RacalDanaGUI` implementa la interface `java.awt.event.ActionListener`, que escucha los eventos producidos en la interfaz gráfica, por ejemplo la pulsación de sus botones. Al detectar el evento realiza una llamada al método `actionPerformed()`. Este método obtiene el tipo de evento producido y la fuente que lo ha producido, es decir, el botón que se ha pulsado, reaccionando en consecuencia [7]. Por ejemplo, a continuación se muestra el código correspondiente al botón "GET" que permite obtener la palabra de datos del sintetizador de frecuencia:

```
...
      dataLabel.setText("DATA:");
      dataLabel.setBounds(new Rectangle(10, 60, 90, 25));
      dataButton.setText("GET");
      dataButton.setBorder(BorderFactory.createBevelBorder(BevelBorder.RAISED));
      dataButton.setBounds(new Rectangle(370, 60, 85, 25));
      dataButton.addActionListener(new ActionListener()
```

```
        {
                public void actionPerformed(ActionEvent e)
                {
                  dataButton_actionPerformed(e);
                }
        });
...
        private void dataButton_actionPerformed(ActionEvent e)
        {
          this.dataTextPane.setText(rdClient.getData());
        }
...
```

Mediante el objeto `rdClient` de la clase `RacalDanaClient`, que se ha pasado como parámetro en el constructor, se obtiene el valor de la propiedad "instrumentData".

La clase `RacalDanaGUI` también implementa la interface `java.awt.event.KeyListener` que, procediendo de manera similar, escucha un evento producido en el teclado y realiza una llamada al método `keyPressed()`. Este método obtiene el tipo de evento producido, si corresponde a la pulsación de *<enter>* realiza las operaciones adecuadas. Por ejemplo, podemos analizar el código necesario para establecer un valor de frecuencia desde la interfaz por medio de la pulsación de la tecla *<enter>*:

```
...

        freqEditorPane.setBorder(BorderFactory.createEtchedBorder(EtchedBorder.RAISED));
        freqEditorPane.setBounds(new Rectangle(20, 55, 225, 30));
        freqEditorPane.setFont(new Font("Dialog", 1, 18));
        freqEditorPane.addKeyListener(new java.awt.event.KeyAdapter()
        {
                public void keyPressed(KeyEvent e)
                {
                  if(e.getKeyCode() == KeyEvent.VK_ENTER)
                     freqEditorPane_keyPressed(e);
                }
        });

...
        private void freqEditorPane_keyPressed(KeyEvent e)
        {
               Double freq = new Double(freqEditorPane.getText());
              fq = freq.doubleValue();
              if(fq < 0.01)
                fq = 0.01;
              if (fq > 1300)
                fq = 1300;

              rdClient.setFrequency(fq);
              fq = rdClient.getFrequency();
              freqEditorPane.setText(str.valueOf(fq));
        }
...
```

Tras comprobar que el dato introducido está en el rango adecuado, se establece el valor de la propiedad de frecuencia utilizando los métodos del objeto `rdClient`.

6 Ejecución del componente-cliente

La ejecución de manager, componente y cliente puede realizarse en una misma máquina o en máquinas distintas [3]. En el OAN el manager se ejecuta en un ordenador distinto al del contenedor y el cliente (cada ordenador ha de tener instalado el ACS). El punto de acceso central al ACS es el manager que mantiene toda la información del estado de los contenedores y sus componentes. Los contenedores y los clientes consultan la variable de entorno `MANAGER_REFERENCE` para saber en qué ordenador se está ejecutando el manager y en qué puerto atiende las peticiones.

La puesta en marcha del componente y el cliente se resume en los siguientes pasos:

1. Iniciamos el manager en un ordenador A (en el CAY *hera.oan.es*). El manager ha de tener acceso al CDB que en este caso está en el mismo ordenador.
   ```
   $> acsStart
   ```

2. Iniciamos en otro ordenador B el contenedor `oanHoloContainer` que contiene el componente del RACAL-DANA. En este ordenador ha de residir la estructura de directorios descrita a lo largo de este informe. La variable de entorno `MANAGER_REFERENCE` ha de tener el valor adecuado:
   ```
   $> export MANAGER_REFERENCE=corbaloc::hera.oan.es:3000/Manager
   $> acsStartContainer -cpp oanHoloContainer
   ```

3. Finalmente iniciamos el cliente en el mismo ordenador B, pero en otro terminal para ver los mensajes de ejecución del contenedor/componente y cliente de manera independiente. Para acceder al cliente es necesario indicar el nombre del paquete java que contiene las clases seguido del nombre de la clase que define el método `main()`.
   ```
   $> acsStartJava alma.oanRacalDana3101.RacalDanaClient
   ```

7 Referencias

[1] NATIONAL INSTRUMENTS. *NI-488.2 Software Reference Manual for MS-DOS*. Mayo1992.

[2] RACAL-DANA. *Operators Manual: 3101 Frequency Synthesizer*. Enero1998.

[3] VICENTE, P. y BOLAÑO, R. Informe Técnico 6. Observatorio Astronómico Nacional. 2004.

[4] Atacama Large Millimeter Array. http://www.eso.org/~gchiozzi/AlmaAcs/. *ACS C++ Component/Container Framework Tutorial*. 2003.

[5] Atacama Large Millimeter Array. http://www.eso.org/~gchiozzi/AlmaAcs/. *ACS Error System*. 2003.

[6] ALMA Homepage: http://www.mma.nrao.edu/

[7] JAVA Homepage: http://java.sun.com