Control remoto del sistema de enfoque para el telescopio óptico del RT de 40m

P. de Vicente

Informe Técnico IT-OAN 2006-5

Índice

1. Introducción

Se ha adquirido un sistema de focalización para enfocar remotamente el telescopio óptico que se pretende instalar en el subreflector del radiotelescopio del 40m. Este sistema permitirá mover el foco desde la sala de control sin necesidad de estar junto al telescopio. La elección se hizo buscando un modelo que fuese controlable remotamente a través de un puerto serie mediante el envío de comandos simples. La elección recayó sobre el modelo TCF-S de Optec. Las siglas TCF corresponden a «Temperature Compensating Focuser». Este dispositivo permite mantener automáticamente el foco del telescopio compensando las dilataciones y contracciones debidas a la temperatura.

En el informe se describe el sistema de enfoque y el componente ACS desarrollado para su control y monitorización.

2. Descripción del equipo

El sistema de enfoque está compuesto por:

- el módulo central, que dispone de un conector para la sonda de temperatura y un conector DB9 que alimenta el motor del sistema y a través del que se envía la tensión que mueve el motor.
- la caja de control, compuesta por un visor, un interruptor, un selector de modo, y dos botones que permiten mover el motor manualmente en ambas direcciones.
- un cable de alimentación que se conecta a la caja de control.
- una sonda de temperatura PT100
- un cable DB9 - RJ12

La figura 1 muestra el bloque central y la caja de control invertida para poder observar los conectores de los cables. En la caja de control, y según se ve en la fotografía, el conector de la izquierda se emplea para la alimentación, el central para comunicarse con el puerto serie de un PC y el de la derecha para comunicarse con el bloque central.

La sonda de temperatura está unida a un cable en cuyo extremo existe un conector PS2 macho. Este conector se inserta en un conector mini-DIN de 6 pines en el bloque principal del TCF-S. El bloque principal del sistema de enfoque recibe las señales para el motor, y envía los datos de la sonda de temperatura, a través de un cable con conector RS232 en el lado del bloque y con un conector RJ45 de 8 patillas en el extremo de la caja de control. Finalmente, la caja de control se puede controlar remotamente a través de un cable con una terminación RJ12 en un extremo (el de la caja) y RS232 DB9 en el otro. La figura 2 muestra el detalle del cableado entre cada conector. En el CAY se añadió al cable RJ12 el conector DB9 de acuerdo con las instrucciones del dibujo.

Figura 1: *Sistema de enfoque: bloque central (a la izquierda), cable con la sonda de temperatura y caja de control (a la derecha)*

3. Funcionamiento

La comunicación entre el PC y el TCF-S se debe hacer a 19200 baudios, con 8 bits de datos, 1 bit de parada y sin paridad. Para el control de este equipo se emplea un conversor Lantronix de 4 puertos serie. Se ha conectado el cable serie a uno de los puertos para que el equipo esté disponible a través de la red local.

El sistema TCF-S admite varios modos de operación:

- modo manual. En este modo el movimiento del motor se comanda a través de los botones blancos «IN» y «OUT» en la caja de control.
- modo remoto. En este modo el movimiento del motor se comanda remotamente empleando órdenes ASCII desde un ordenador.
- modos de aprendizaje A y B. En estos modos el sistema ajusta su posición en función de la temperatura. Para ello dispone de una correspondencia que asocia cambios de temperatura con cambios en la posición, en una u otra dirección.
- modo durmiente. En este modo el sistema entra en modo de bajo consumo, aunque es capaz de recibir órdenes remotas para activarse de nuevo.

Para que el sistema responda a órdenes remotas es necesario situar el conmutador de la caja de control en la posición PC MANUAL. Las otras posibles posiciones son AUTO A y AUTO B que permiten situar el sistema de enfoque en los modos de operación automáticos en los que se corrige la posición en función de la temperatura. También es necesario que el conmutador PROGRAM esté en la posición RUN.

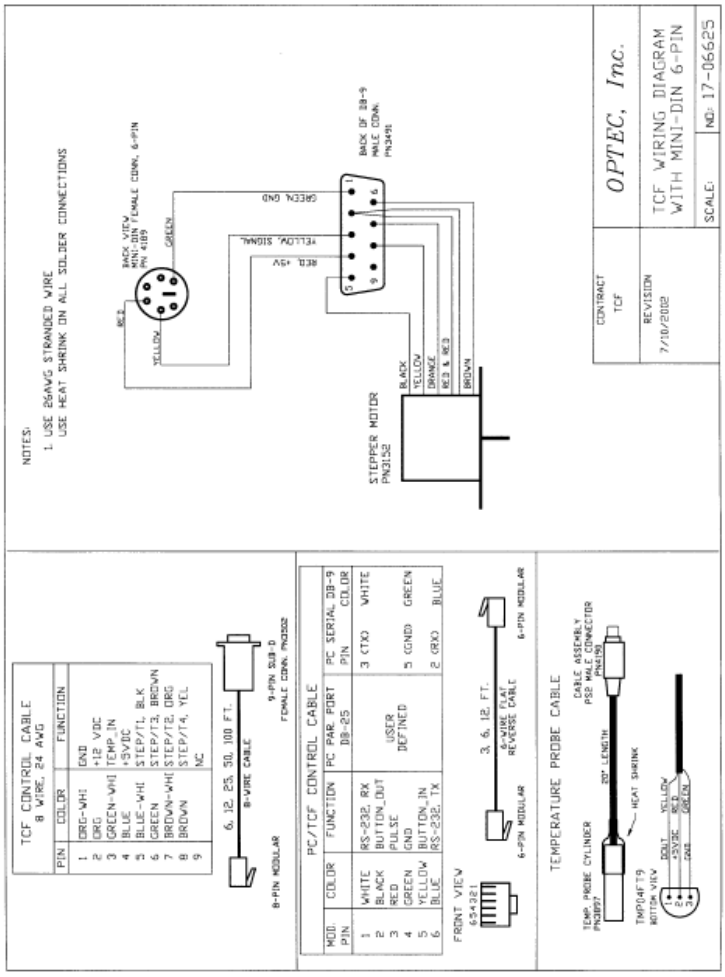

Figura 2: *Cableado del equipo.*

El modo de operación remota se activa al enviar a través del puerto serie el comando FMMODE. A partir de este momento el sistema de enfoque es controlable a través de los comandos que se relacionan en la tabla 1. Tras cada comando recibido el sistema de enfoque responde con una cadena de caracteres que permite verificar al usuario que la orden se ha recibido y se ha ejecutado o que se ha producido un error. En la tabla 1 se muestra la respuesta que envía el equipo TCF-S.

Comando	Respuesta	Descripción breve
FMMODE		Modo remoto
FAMODE	A	Modo automático A
FBMODE	B	Modo automático B
FInnnn	*	Mover nnnn pasos hacia dentro
FOnnnn	*	Mover nnnn pasos hacia fuera
FPOSRO	P=nnnn	Leer posicion
FTMPRO	T= $\pm$ nn.n	Leer temperatura
FCENTR	CENTER	Centrar en 3500
FSLEEP	ZZZ	Modo durmiente
FWAKUP	WAKE	Despertar del modo durmiente
FREADA	A=0nnn	Leer pendiente A
FREADB	B=0nnn	Leer pendiente B
FLAnnn	DONE	Comandar pendiente A
FLBnnn	DONE	Comandar pendiente B
FFMODE	END	Modo manual

Cuadro 1: *Comandos aceptados por el TCF-S y respuestas a dichos comandos. Todos las respuestas a los comandos acaban con un retorno de carro y una nueva línea*

El sistema de enfoque se puede mover en pasos de 0.002032 mm cantidad que corresponde aproximadamente a un 1/3 de la longitud de onda del visible. El número máximo de pasos es 7000. La posición 7000 se corresponde con la máxima separación entre la cámara CCD y el telescopio, y 0 la mínima separación. El valor en uso se muestra en el visor de la caja de control.

La temperatura de la sonda se ofrece en grados centígrados y las pendientes A y B indican el número de pasos por grado centígrado que se ha de mover el sistema para mantener el foco. Estos valores sólo se pueden obtener empleando el modo remoto.

4. Proceso de desarrollo del componente ACS

El proceso que se ha seguido en el desarrollo del componente es el mismo que con el resto de los componentes ACS para el sistema de control del radiotelescopio de 40m.

Se comenzó creando un directorio denominado `focuser` con el subdirectorio `idl` y dos archivos: `TCFSfocuser.idl` y `focuserError.xml`. En el primero archivo se describe el interfaz del sistema de enfoque con sus propiedades y métodos y en el segundo archivo se definen las posibles excepciones que puede lanzar el componente en caso de error. A continuación se

utilizó el generador de código `acsGenerator` para generar los ficheros de la implementación, las cabeceras y los DevIOs. Para poder generar las clases que heredan de DevIO el generador de código precisa un archivo auxiliar en el que figuren los nombre de las propiedades y su tipo. En nuestro caso el archivo `devionames` contenía las siguientes líneas:

```
operationMode, DevIOoperationMode, TCFSfocuser::opMode
position, DevIOposition, CORBA::Double
temperature, DevIOtemperature, CORBA::Double
slopeA, DevIOslopeA, CORBA::Long
slopeB, DevIOslopeB, CORBA::Long
```

Para lanzar el generador de código se introdujeron las siguientes ordenes en el intérprete del sistema operativo:

```
cd aries21/focuser
acsGenerator --deviofile devionames idl/TCFSfocuser.idl
```

El generador de código crea los directorios `include`, `src` y `config`. A continuación en el subdirectorio `src` se desarrolló una clase C++ (`TCFS`) que implementa las funcionalidades del sistema de enfoque. La clase no es C++ pura porque los métodos pueden lanzar excepciones definidas en el archivo `focuserError.xml` y emplean la macro `ACS_LOG` para depurar el código.

El generador de código implementa el componente en numerosos archivos, uno por método, pero se reunificaron en uno sólo para evitar la dispersión de código. Los DevIOs se implementaron en el fichero de cabecera donde además se declaran, eliminando de este modo los archivos `.cpp` creados por el generador de código.

Se reformó el archivo `Makefile` para tener en cuenta todas las modificaciones anteriores, es decir, la ausencia de los archivos `.cpp` correspondientes a los DevIOs, la biblioteca para la clase C++ pura, y el uso de excepciones. Más abajo se muestra un extracto de la parte más importante del archivo `Makefile`:

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

#
# Libraries (public and local)
# ----------------------------
LIBRARIES = tcfs TCFSfocuser

tcfs_OBJECTS = tcfs
tcfs_LIBS = focuserError socketError sockscpp

#
# <brief description of TCFSfocuser library>
TCFSfocuser_OBJECTS = TCFSfocuserImpl
```

```
TCFSfocuser_LDFLAGS =
TCFSfocuser_LIBS    = tcfs TCFSfocuserStubs focuserError

#............
#
# Configuration Database Files
# ----------------------------
CDB_SCHEMAS = focuser

#
# IDL Files and flags
#
IDL_FILES = TCFSfocuser
IDL_TAO_FLAGS =
USER_IDL =
#
ACSERRDEF = focuserError
```

Se podría considerar que el archivo `Makefile` indica dos tipos de instrucciones:

- las necesarias para generar las bibliotecas de funciones
- las necesarias para instalar en el directorio `$INTROOT` las bibliotecas de funciones, los archivos IDL, los archivos XML y los ficheros de cabecera.

Las entradas precedidas por la etiqueta: `INCLUDES` indican los archivos de cabecera que se van a instalar en `$INTROOT/include` y que por tanto son públicos y pueden ser incluidos por otros componentes en caso necesario. Las entradas correspondientes a la etiqueta `LIBRARIES` indican las bibliotecas que se van a generar y que se van a instalar en `$INTROOT/lib`. A continuación para cada biblioteca se escriben las etiquetas con el nombre de cada biblioteca seguida de `_OBJECTS` y de `_LIBS` que indican los objetos y las librerías de los que dependen y que por tanto son necesarios para construirse.

Durante el proceso de compilación se genera la biblioteca «TCFSfocuser» del componente que carga el contenedor y la documentación en HTML empleando «doxygen». Si se mantienen actualizados los comentarios en el código la documentación, que se regenera automáticamente, permanece actualizada.

5. Descripción del componente ACS para el sistema de enfoque

Se ha desarrollado una clase C++ que implementa toda la funcionalidad del sistema de enfoque. Dado que este sistema se controla a través de una conexión RS232, conectada a un conversor Lantronix serie-ethernet, en el constructor de la clase se pasa como parámetro la dirección IP del conversor Lantronix y el puerto en que se encuentra el sistema de enfoque. No existe ningún método que permita variar estos valores sobre la marcha porque tal posibilidad

no tiene sentido. Es necesario destruir el objeto y volver a crearlo con nuevos parámetros para realizar esta modificación. El constructor llama a un método llamado `resetLan()` que limpia el puerto al que se encuentra conectado el sistema de focalización para evitar interferencias en las órdenes y en las lecturas a través de dicho puerto. La limpieza se consigue conectandose al puerto 23 del Lantronix y enviando la orden `logout port 2`. Para poder ejecutar esta orden es necesario entrar como usuario con todos los privilegios.

Existen funciones para fijar el sistema en cada uno de los posibles modos: remoteMode, manualMode, AutomaticAMode, automaticBMode, manualMode y sleepMode y una variable privada que almacena el estado actual del sistema. No es posible pasar de un estado a otro arbitrariamente. En la mayor parte de los casos el tránsito de un modo a otro requiere pasar por el estado remoto antes. Si el estado de partida no es el adecuado se genera una excepción indicando que el modo de destino es incorrecto. Obviamente se puede pasar al modo remoto desde cualquier otro estado.

La información procedente de la caja de control a través del puerto serie incluye los caracteres de fín de línea y retorno de carro. Para poner de manifiesto con claridad y explícitamente esta situación, en el código, en todas las lecturas se escribe el número de caracteres más los dos caracteres del final de cada cadena, en lugar de la suma completa. Por ejemplo:

```
client->Read(buf,7+2,0);
```

Se han escrito funciones para leer la posición, temperatura y las pendientes para los modos A y B automáticos. Existe una variable privada para cada una de estas propiedades. La posición se indica en pasos, la temperatura en grados centígrados y las pendientes en pasos/grado centígrado. Como funciones auxiliares se han escrito dos funciones privadas que interpretan las cadenas generadas por el sistema de enfoque y las transforman en números enteros o reales según el caso. También existen las correspondientes funciones para fijar la pendiente en los modos A y B. La posición relativa del foco se puede modificar empleando dos funciones denominadas `stepIn` y `stepOut`, que admiten como argumento un entero que indica el número de pasos que se desea desplazar el foco. Como ya se ha mencionado anteriormente un paso equivale a 0.002032 mm. Además se han creado dos funciones que mueven el sistema a la posición central (3500) y que permiten fijar el foco en una posición absoluta.

Cuando el sistema se coloca en modo automático A o B, la caja de control informa continuamente de la temperatura del sensor y de la posición del foco enviando dos cadenas. La función `continuousRead()` lee en dos fases esta información: 8 caracteres para la posición, y 9 para la temperatura y guarda los valores en las variables privadas correspondientes.

Cuando se envía un comando `FMMODE` y el sistema está en cualquiera de los modos automáticos se pasa a remoto deteniendo, aunque no inmediatamente, el envío de esa información. La información del modo A y B se detiene una vez que las cadenas completas se han completado y a continuación se devuelve una cadena de confirmación como respuesta al comando de paso a modo remoto. Para evitar que al leer el mensaje de confirmación el sistema genere una excepción se ha escrito una función privada denominada `flushBuffer()` que lee carácter a carácter el puerto descartando todo lo leído hasta que no quede nada en el buffer. Este flujo de caracteres que se descarta corresponde al contenido del buffer como respuesta a los modos automáticos y al mensaje de confirmación de paso a modo remoto.

A continuación se incluye una relación de las funciones de la clase `Mintron` y de sus variables privadas:

Funciones públicas:

`void TCFS(const char *`**tcpiaddress**`, unsigned int` **port**`) throw (focuserError::cannotConstructExImpl &, focuserError::cannotConnectExImpl &, focuserError::cannotInitializeExImpl &)`

Constructor de la clase: reinicia el dispositivo Lantronix y abre un conexión TCP con la dirección y el puerto que se pasan como parámetro.

tcpiaddress *Dirección del dispositivo Lantronix.*

port *Puerto [1,4] al que está conectado la cámara.*

`void reset() throw (focuserError::readErrorExImpl &, focuserError::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &, focuserError::paramOutOfLimitsExImpl &)`

Pasa el sistema a modo remoto y situa al sistema con las variables predefinidas.

`void remoteMode() throw (focuserError::readErrorExImpl &, focuserError::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &, focuserError::paramOutOfLimitsExImpl &)`

Pasa el sistema a modo remoto desde el que es posible acceder a cualquier otro modo y controlar y monitorizar todo el sistema.

`void manualMode() throw (focuserError::readErrorExImpl &, focuserError::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &, focuserError::wrongModeExImpl &)`

Pasa el sistema a modo manual permitiendo el uso de los botones en la cara frontal de la caja de control.

`void autoAMode() throw (focuserError::readErrorExImpl &, focuserError::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &, focuserError::wrongModeExImpl &)`

Pasa el sistema a modo automático A desde el que el sistema varía la posición del foco de acuerdo con la temperatura ambiente según la pendiente almacenada en el registro A.

`void autoBMode() throw (focuserError::readErrorExImpl &, focuserError::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &, focuserError::wrongModeExImpl &)`

Pasa el sistema a modo automático desde el que el sistema varía la posición del foco de acuerdo con la temperatura ambiente según la pendiente almacenada en el registro B.

`void goSleep() throw (focuserError::readErrorExImpl &, focuserError::write-ErrorExImpl &, focuserError::wrongAcknowledgementExImpl &, focuser-Error::wrongModeExImpl &)`

Pasa el sistema a modo durmiente, en el que el consumo de energía es menor.

`void goWakeUp() throw (focuserError::readErrorExImpl &, focuserError::write-ErrorExImpl &, focuserError::wrongAcknowledgementExImpl &, focuser-Error::wrongModeExImpl &)`

Pasa el sistema a modo remoto cuando el estado de partida es el modo durmiente.

`void moveIn(long` **steps**`) throw (focuserError::readErrorExImpl &, focuser-Error::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &, focuserError::paramOutOfLimitsExImpl &)`

Mueve el sistema de enfoque disminuyendo la distancia entre la cámara y el telescopio.

zoom Número de pasos a mover. 1 paso = 0.00008 pulgadas = 0.002032 mm.

`void moveOut(long` **steps**`) throw (focuserError::readErrorExImpl &, focuser-Error::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &, focuserError::paramOutOfLimitsExImpl &)`

Mueve el sistema de enfoque aumentando la distancia entre la cámara y el telescopio.

zoom Número de pasos a mover. 1 paso = 0.00008 pulgadas = 0.002032 mm.

`void move(long` **steps**`) throw (focuserError::readErrorExImpl &, focuser-Error::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &, focuserError::paramOutOfLimitsExImpl &)`

Mueve el sistema de enfoque en posición absoluta. El cero se produce cuando la cámara y el telescopio están lo más próximo que el sistema permite. Los valores positivos aumentan la distancia entre el telescopio y la cámara. El centro del movimiento está en 3500 y la posición máxima 7000.

steps Número de pasos a mover. 1 paso = 0.00008 pulgadas = 0.002032 mm.

`void moveToCenter() throw (focuserError::readErrorExImpl &, focuser-Error::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &)`

Mueve el sistema al valor medio del rango.

`void position() throw (focuserError::readErrorExImpl &, focuserError::write-ErrorExImpl &, focuserError::wrongAcknowledgementExImpl &)`

Devuelve la posición del sistema de enfoque. Unidades: pasos.

`void temperature() throw (focuserError::readErrorExImpl &, focuser-Error::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &)`

Devuelve la temperatura medida por la sonda. Unidades: grados centígrados.

`void readASlope() throw (focuserError::readErrorExImpl &, focuser-Error::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &)`

Devuelve el valor de la pendiente en el modo automático A. Pasos por grados centígrado

`void readBSlope() throw (focuserError::readErrorExImpl &, focuser-Error::writeErrorExImpl &, focuserError::wrongAcknowledgementExImpl &)`

Devuelve el valor de la pendiente en el modo automático B. Pasos por grados centígrado

`void writeASlope(int` **value**`) throw (focuserError::readErrorExImpl &, focuserError::writeErrorExImpl &, focuserError::wrongAcknowledgement-ExImpl &, focuserError::paramOutOfLimitsExImpl &)`

Escribe el valor de la pendiente en el modo automático A.

value Pendiente equivalente al número de pasos a mover por grado centígrado

`void writeBSlope(int` **value**`) throw (focuserError::readErrorExImpl &, focuserError::writeErrorExImpl &, focuserError::wrongAcknowledgement-ExImpl &, focuserError::paramOutOfLimitsExImpl &)`

Escribe el valor de la pendiente en el modo automático B.

value Pendiente equivalente al número de pasos a mover por grado centígrado

`void continuousRead() throw (focuserError::readErrorExImpl &)`

Lee continuamente la posición y la temperatura que proporciona la caja de control cuando el sistema en un modo automático.

`void operationMode()`

Devuelve el modo de operación en el que se encuentra el sistema

`void flushBuffer()`

Lee el flujo de caracteres de un socket hasta que no queda ninguno en el «buffer».

`void resetLan (const char *`**tcpiaddress**`, unsigned int` **port**`) throw (focuserError::cannotConnectExImpl &, focuserError::writeErrorExImpl &)`

Reinicia el dispositivo Lantronix abriendo una sesión al puerto 23 y reiniciando el puerto.

tcpiaddress *Dirección del dispositivo Lantronix*

port *Puerto [1,4] que se reinicia.*

La figura 3 muestra un esquema de las clases empleadas.

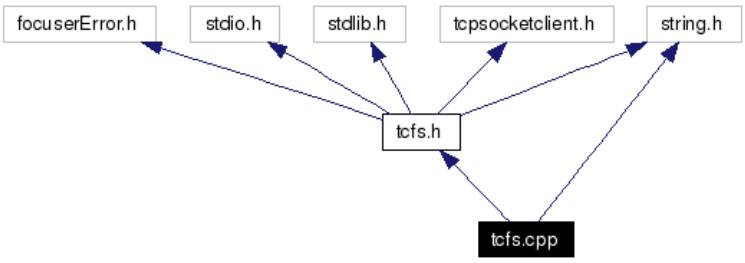

Figura 3: *Gráfico de dependencias del archivo tcfs.cpp*

5.1. Errores. Lanzamiento de excepciones

En el componente ACS del sistema de enfoque se han definido un conjunto de excepciones en el fichero `focuserError.xml` y se han identificado con el número 2006. En la lista que se relaciona a continuación se incluye el nombre identificativo y la descripción que figura en el archivo `focuserError.xml`:

focuserCorrect Description: «TCFS request sucessfully processed»

cannotConstruct Description: «TCFS constructor error»

cannotConnect Description: «Cannot connect to TCFS»

cannotInitialize Description: «Cannot talk and initialize the TCFS». Se produce cuando se produce un error en la conexión de los sockets.

writeError Description «Error sending data to TCFS». Esta excepción se genera a partir de una excepción en la escritura sobre un objeto de la clase *Tcpsockclient*

readError Description «Error receiving data from TCFS». Esta excepción se genera a partir de una excepción en la lectura de un objeto de la clase *Tcpsockclient*

paramOutOfLimits Description: «Parameters sent to TCFS out of limits»

wrongAcknowledgement Description: «Wrong acknowledgement from TCFS»

wrongMode Description: «Cannot read/write property from this mode. Change to remote mode»

Las excepciones se lanzan desde la clase C++ pura, desde las clase de los DevIOs y desde la clase que implementa el componente, de acuerdo con la política de ACS.

5.2. Definición del archivo IDL

En el archivo IDL se ha definido un módulo llamado *TCFSfocuser* y un interfaz denominado *focuser* que hereda de *ACS::CharacteristicComponent*. En el módulo se ha definido un ENUM que se utiliza para describir el estado del sistema de enfoque y a partir del cual se ha creado una propiedad ACS de sólo lectura. Se han creado 7 propiedades y 8 métodos síncronos:

- *reset()*, se utiliza para reiniciar el sistema de enfoque y fijar sus opciones a los valores predefinidos. También se emplea para pasar el sistema a modo remoto desde cualquiera de los otros modos.
- *stpeIn()* y *stepOut* sirven para mover paso a paso el foco en una y otra dirección.
- *manualMode()* pasa el sistema a modo manual,
- *automaticAMode()* y *automaticBMode()* pasan el sistema a los modos automáticos A y B
- *goSleep()* sirve para poner la cámara en modo durmiente
- *goWakeUp* sirve para sacarla del modo durmiente.

La cámara se controla a través de propiedades de lectura y escritura que en algunos casos son enums, enteros largos (long) y reales largos (double). Existen tres propiedades de sólo lectura: la dirección IP del dispositivo Lantronix, el puerto al que se conecta el equipo y el modo en el que se encuentra el dispositivo. Las dos primeras propiedades se leen durante la inicialización a partir del archivo XML en la base de datos.

Hubiera sido posible que la propiedad del estado del sistema fuese escribible y de este modo seleccionar el modo. Sin embargo se decidió emplear una propiedad de sólo lectura para informar del estado y métodos para cambiar dicho estado para evitar generar más DevIOs.

Las propiedades del interfaz se relacionan a continuación:

ACS::ROstring lantronixIPNumber: Número TCIP en forma de cadena del dispositivo Lantronix al que está conectada la cámara.

ACS::ROlong lantronixSerialPort: Puerto serie del dispositivo Lantronix al que está conectada la cámara.

ROopMode operationMode: Modo de operación en el que se encuentra el sistema de enfoque.

ACS::RWlong position: Posición del sistema de enfoque en pasos. Comandadble.

ACS::ROdouble temperature: Temperatura medida por la sonda.

ACS::RWlong slopeA: Número de pasos que se mueve el sistema de enfoque en el modo automático A. Comandadble.

ACS::RWlong slopeB: Número de pasos que se mueve el sistema de enfoque en el modo automático B. Comandadble.

5.3. Implementación del componente: servidor

La implementación se hace a través de los métodos de lectura y escritura de las clases de los DevIO y de los métodos síncronas. En el constructor de cada clase DevIO se pasa una referencia del objeto *TCFS*. De este modo todos los métodos públicos de esta clase están disponibles para al DevIO. Los métodos de lectura y escritura crean un *muttex* para impedir la modificación de las variables por varios usuarios simultáneamente, llaman al método necesario de la clase TCFS y actualizan el valor de la variable privada. El *muttex* está también implementado en los métodos que cambian de modo el sistema. La temperatura es una propiedad de sólo lectura y su valor se obtiene tras consultar el valor de la variable privada correspondiente de la clase C++ a través de la función `temperature()`. Esta función devuelve el valor almacenado de la temperatura obtenida si los modos automáticos están en marcha, genera una excepción si el modo no es remoto, y si es remoto envía una instrucción al dispositivo para obtener el valor. Se ha evitado realizar una consulta durante los modos automáticos porque para ello sería necesario detener dichos modos, pasar a remoto y luego retornar al modo automático original.

La posición se obtiene del mismo modo que la temperatura. Sin embargo las pendientes de los modos automáticos, se obtienen sólamente si el modo es remoto a través de las funciones públicas de la clase C++. Para cambiar estas propiedades se emplean los métodos públicos correspondientes de la clase C++.

No hemos querido obtener el valor de la temperatura y posición de variables privadas en los modos durmiente y manual porque estos valores en estos modos pueden cambiar con el tiempo, y el único modo real de obtenerlos es situar el equipo en modo remoto e interrogar al equipo. Por tanto si se trata de conocer la temperatura o la posición en estos modos la implementación genera una excepción indicando al usuario que el modo es incorrecto y que es necesario pasar a remoto para efectuar esta operación. Este no es el caso de los valores de las pendientes que sólo se pueden modificar explicitamente empleando los métodos públicos de la clase TCFS.

Las funciones síncronas del componente llaman a las funciones correspondientes de la clase C++ pura que permiten pasar el sistema de un modo a otro. Cuando el modo de destino es cualquiera de los dos modos automáticos el proceso es el siguiente:

1. Se comanda el sistema para que pase a modo automático.

2. Se inicia un hilo de ejecución que pone en marcha el método `continuousRead()` de la clase C++ pura que lee las cadenas que envía el sistema constantemente. El hilo principal queda a la espera de las peticiones de los clientes.

Cuando se pasa desde un modo automático al modo remoto el proceso es el siguiente:

1. Se detiene el hilo de ejecución
2. Se comanda el modo remoto. Es muy posible que esta orden genere un error, porque en el mensaje de respuesta se mezclan los caracteres del modo automático con los de confirmación. Téngase en cuenta que tras comandar el modo remoto, el sistema de enfoque envía tres cadenas: el resto del mensaje que hay en el buffer correspondiente a la información de temperatura y posición (dos cadenas) y la confirmación del paso a modo remoto.
3. Si se produce el error anterior se lee el «buffer» hasta que este queda limpio.
4. Se vuelve a comandar el modo remoto. Esta vez si se produce un error se genera una excepción porque tan sólo se acepta una confirmación correcta.

La figura 4 muestra un esquema de la interrelación entre la clase de la implementación (*cameraImpl*), las clases de los DevIOs y la clase *TCFS*:

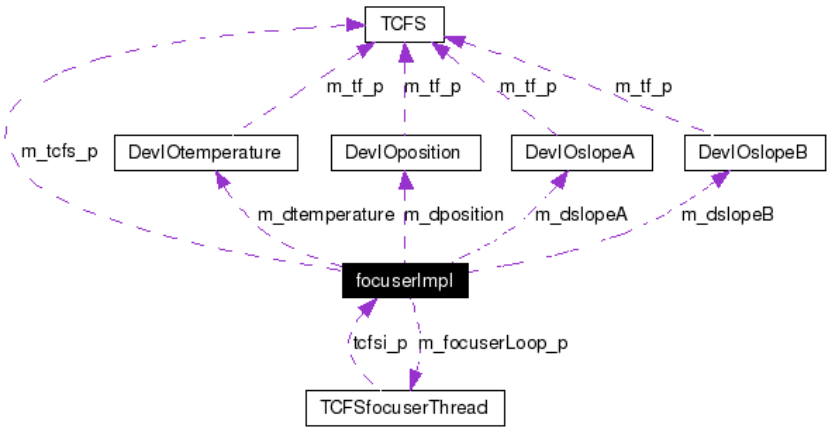

Figura 4: *Gráfico de relaciones entre la implementacion, los DevIOs y la clase TCFS*

La clase *focuserImpl* instancia los DevIOs en el método initialize() en lugar de hacerlo en el constructor, ya que, en aquel, habitualmente se reserva memoria o/y se abre una conexión con un socket. Cualquiera de las operaciones anteriores puede fallar y resulta más adecuado en ese caso que estas operaciones se hagan en la inicialización del componente según la política de ACS. Al crearse cada DevIO, se lee su valor predeterminado de la base de datos (del archivo XML) y se llama al método *write()* que escribe el valor de dicha variable.

Se ha incluido un método denominado `reset()` que pasa el sistema a modo remoto y que configura el sistema de enfoque a su posición inicial: 3500 y fija las pendientes de los modos automáticos A y B a 86. Estos valores se toman de la base de datos y por tanto si se cambian allí el método `reset()` emplea los nuevos valores predeterminados sin necesidad de cambiar el código.

5.4. Configuración inicial de la instancia: archivo de la base de datos

El componente del `focuser` puede crear varias instancias, aunque como sólo disponemos de un sólo sistema de enfoque sólo existira una instancia. Los valores predefinidos con los que se caracteriza el componente se guardan en el archivo `TCFS/TCFS.xml`, que reside en el directorio `OANCDB/CDB/alma`. Este directorio contiene la base de datos de todos los componentes del radiotelescopio de 40m y se puede navegar utilizando la aplicación cdbBrowser. La figura 5 muestra el contenido de los valores para la propiedad «temperature» de la instancia TCFS del componente `focuser`.

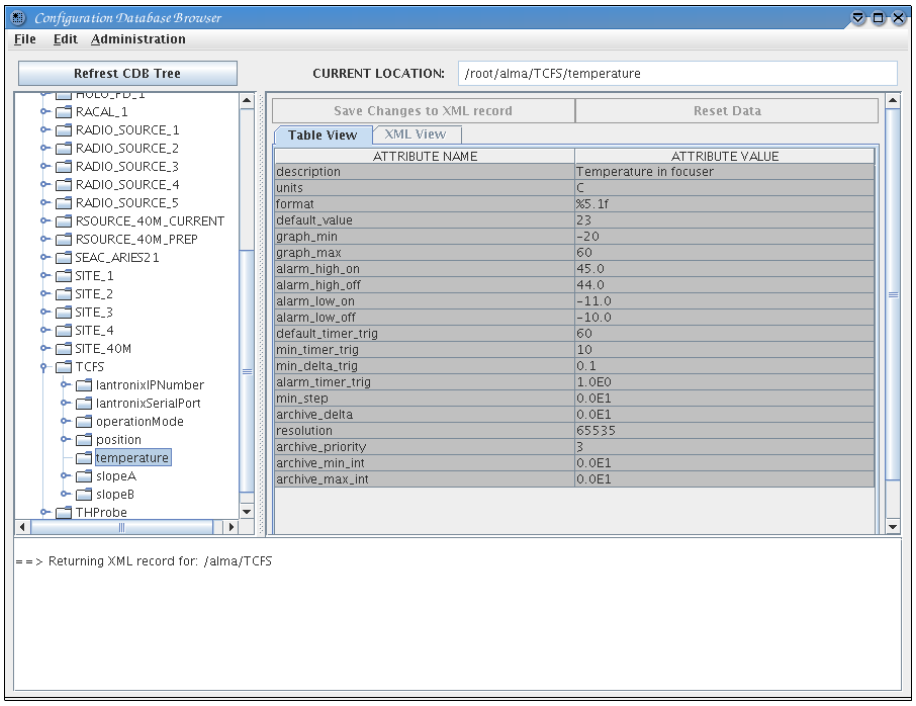

Figura 5: *Parámetros de la propiedad «temperature» de la instancia TCFS del focuser*

Los diferentes valores se pueden modificar desde la aplicación `cdbBrowser` pulsando dos veces sobre cada campo, y pulsando después sobre «Save Changes to XML Record». Finalmente es conveniente refrescar la base de datos pulsando sobre «Administration->Clear Cache».

Los principales campos a tener en cuenta en la temperatura son:

- `description`. Descripción de la propiedad
- `units`. Unidades, que en el caso de la temperatura son grados C.
- `format`. Formato de los datos
- `alarm_high_on` y `alarm_high_off`. Por encima de 45 grados se activa una alarma y se desactiva por debajo de 44.

- `alarm_high_on` y `alarm_high_off`. Por debajo de -11 grados se activa una alarma y se desactiva por encima de -10.
- `default_timer_trig, min_timer_trig, delta_timer_trig`. En el caso de utilizar un Monitor y no fijar el intervalo del cronómetro este vale 60 segundos y el mínimo aceptado es: 10 segundos. Por otra parte cambios de la temperatura superiores a 0.1 grado centígrado deberían activar el método `working`

5.5. Implementación del cliente en Python

Se ha escrito un cliente gráfico en Python para el control del sistema de enfoque. En primer lugar se empleó el diseñador gráfico de Qt que guarda el aspecto gráfico en un archivo XML que se denominó `focuser.ui`. A continuación se empleó el programa `pyuic` para generar una clase en Python (`focuserUI.py` que emplea los paquetes de Qt para Python. En el archivo `Makefile` se ha incluido una entrada para realizar automáticamente esta operación:

```
UItoPython: ../test/focuser.ui
        pyuic ../test/focuserUI.ui -o ../test/focuserUI.py
```

El principal problema con el cliente reside en el método de actualización de las propiedades de sólo lectura. Existen varias opciones para escribir un cliente:

1. Se puede utilizar un cliente que use «Monitors». Este caso desgraciadamente no funciona totalmente bien. A pesar de ello se describe aquí para describir las pruebas que se realizaron.

   Se utilizaron dos archivos:

   - `focuserClient.py` contiene dos clases: `focuserACS` y `Monitor`. El constructor de «focuserACs» crea un objeto de la clase `PySimpleClient`, otro del componente «TCFS» y uno por cada propiedad de este componente:

     ```
     class focuserACS:
        def __init__(self,argv):
             self.__sc = PySimpleClient()
             self.__focuser = self.__sc.getComponent("TCFS")

             self.__lantronixIPNumber = self.__focuser._get_lantronixIPNumber
             self.__lantronixSerialPort = self.__focuser._get_lantronixSerial
             self.__position = self.__focuser._get_position()
             self.__temperature = self.__focuser._get_temperature()
             self.__operationMode = self.__focuser._get_operationMode()
             self.__slopeA = self.__focuser._get_slopeA()
             self.__slopeB = self.__focuser._get_slopeB()
           .....
     ```

     Además en esta clase se definen métodos para fijar y leer las propiedades. Por ejemplo:

```
....
 def position(self):
    """@return position in steps. [0, 7000]
    """
    return self.__position.get_sync()[0]

def setPosition(self, value):
    """Sets the new position in steps. [0, 7000]
    """
    cl =  self.__position.set_sync( value )
  .....
```

La clase «Monitor» de ACS contiene las definiciones de dos métodos: `working` y `done` que es necesario reimplementar. A la clase «Monitor» se le pasa una referencia de la ventana (QApplication) para poder usar un método que refresque el panel LCD cada vez que la temperatura se actualiza. Cada vez que el cronómetro vence o la propiedad asociada modifica su valor se ejecuta el método `working`. El valor del cronómetro se fija explicitamente al crear el objeto Monitor o si no se fija se toma el valor predefinido en la base de datos en el archivo `TCFS.xml` en el campo `default_timer_trig`. En este caso el método `working` emite una señal de PYTHON denominada «tempSignal» con un valor. Esta señal se conecta con un slot en otra clase que explicaremos a continuación.

```
def working (self, value, completion, desc):
    self.window.emit(PYSIGNAL("tempSignal"),(value,))

def done (self, value, completion, desc):
    self.window.emit(PYSIGNAL("tempSignal"),(value,))
```

La creación del «Monitor» y el callback que este necesita se hace en la función `initMonitor()` de la clase `focuserACS`. En este método se indica el intervalo de disparo del cronómetro en centenas de nanosegundos:

```
def initMonitor(self, tempLCD, strVal):
    self.cbTemp = MyMonitor(tempLCD,strVal)
    self.cbTempS = self.cbTemp._this()
    self.desc1 = ACS.CBDescIn(0L, 0L, 0L)
    self.TempMon = self.__temperature.create_monitor(self.cbTempS, s
    self.TempMon.set_timer_trigger(600000000)
```

- El archivo `focuserGUI.py` contiene la clase `focuserGUI`. Esta clase hereda de `focuserUI` y por tanto tiene acceso a todos los «widgets» gráficos. En el constructor de la clase se crea un objeto de la clase `focuserACS` y de este modo se tiene acceso completo a todos sus métodos. También se crea el «Monitor»:

```
class focuserGUI(focuserUI):
    def __init__(self):
        focuserUI.__init__(self)
        self.__focuser = focuserACS("")
        ....
```

```
initMonitor()
```

Finalmente en esta clase se asocian los eventos gráficos, «signals» a los métodos de la clase `mintronACS` que se han considerado «slots» y se inicializan los valores de los «widgets» tras leer el estado del componente a través de los métodos de lectura de las propiedades definidos en `focuserACS`. Un caso especial de conexión «signal» - «slot» es el que asocia la señal de PYTHON con el «slot» que actualiza el widget LCD:

```
sWindow.connect(sWindow,PYSIGNAL("tempSignal"),sWindow.slotUpdateTemp)
```

donde:

```
def slotUpdateTemp(self, value):
    self.temperatureLCD.display("%4.1f" % (value) )
    #self.repaint()
```

Desafortunadamente si la ventana de la aplicación no tiene el foco el valor de la temperatura no se actualiza a no ser que se produzca un evento gráfico que la repinte. No entendemos este comportamiento, ya que el mecanismo de señales y ranuras debería ser adecuado para insertar eventos en el ciclo de vida de la ventana que se inicia al ejecutar:

```
app.exec_loop()
```

2. En este caso la actualización de la temperatura corre de cuenta de un cronómetro (QTimer) de QT:

```
internalTimer = QTimer( self )      # create internal timer
self.connect( internalTimer, SIGNAL("timeout()"), self.updateTemp )
internalTimer.start( 10000 ) # 10 seconds
```

donde el procedimiento que se llama al vencer el tiempo es como sigue:

```
def temperature(self):
    return self.__temperature.get_sync()[0]

def updateTemp(self):
    tempVal = self.temperature()
    self.temperatureLCD.display("%4.1f" % (tempVal))
```

En esta opción hemos agrupado todo el código, idéntico al caso anterior, en un sólo archivo: `focuserGUI2.py`. El constructor contiene las instrucciones para crear un objeto de la clase `PySimpleClient`, otro del componente «TCFS» y uno por cada propiedad de este componente. Además inicializa todos los widgets gráficos. En este archivo se agrupan todos los métodos contenidos en la clase `focuserACS` y `focuserGUI` del ejemplo anterior.

La figura 6 muestra el cliente gráfico definitivo, que externamente es idéntico en ambas casos. La posición se puede comandar desde el cuadro incremental (SpinBox) y desde el deslizador. Los diferentes modos se comandan desde la lista desplegable y los valores de las pendientes desde los recuadros incrementales correspondientes. Cuando el modo seleccionado no es remoto, todos los recuadros, excepto la lista desplegable que permite el paso al modo remoto, se deshabilitan.

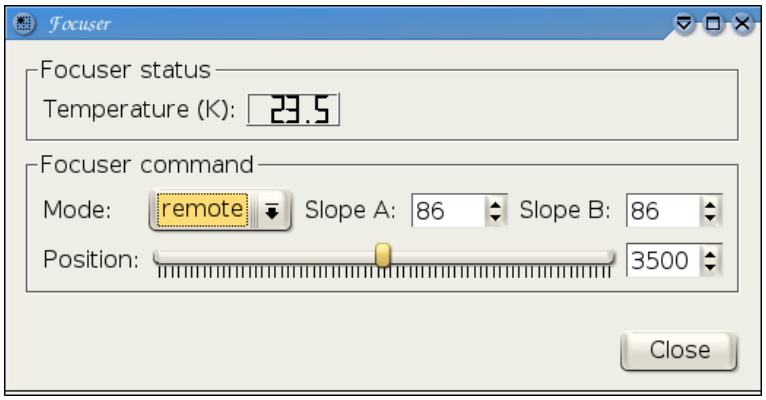

Figura 6: *Captura del cliente gráfico en Python*

Referencias

[1] “Temperature Compensating Focuser. Technical Manual”, Optec Inc. Revision 6. 2002