JCAAC (2025), **2**, 75–86

JCAAC

SECTION: PROGRAMACIÓN DE DISPOSITIVOS ASTRONÓMICOS

Diseño, construcción y programación de dispositivos astronómicos: una introducción

Sergio Alonso[1] and Javier Flores[2]

[1]Dpto. de Lenguajes y Sistemas Informáticos, Universidad de Granada, Sociedad Astronómica Granadina, España.
E-mail: zerjioi@ugr.es.
[2]Observatorio Astronómico de Calar Alto, Sociedad Astronómica Granadina, España. E-mail: jflores@caha.es.

Palabras Clave: dispositivos astronómicos, *astronomical devices*, *hardware*, *software*, controlador, *driver*, INDI, INDIGO, panel de *flats*, *flats panel*

Resumen
En este primer artículo de la serie dedicada al diseño, construcción y programación de dispositivos astronómicos hacemos una introducción a como los sistemas operativos actuales son capaces de gestionar los dispositivos que usualmente conectamos a nuestros ordenadores. En el caso particular de la astronomía, por la especificidad de los dispositivos que manejamos (que pueden no ser de interés para el usuario general de la informática) además existen varias plataformas de más alto nivel (ASCOM, INDI, INDIGO) que permiten tanto el desarrollo de nuevo instrumental como el control del mismo, incluso de manera distribuida. Hacemos un repaso de las características principales de dos de estas plataformas (INDI e INDIGO) y presentaremos de manera breve el dispositivo (un panel de *flats*/*dark*) que usaremos a modo de ejemplo en los siguientes artículos de la serie.

Abstract
In this first article of the series dedicated to the design, construction and programming of astronomical devices we make an introduction to how current operating systems are able to manage the devices that we usually connect to our computers. In the particular case of astronomy, due to the specificity of the devices we handle (which may not be of interest to the general computer user) there are also several higher level platforms (ASCOM, INDI, INDIGO) that allow both the development of new instruments and their control, even in a distributed environment. We review the main features of two of these platforms (INDI and INDIGO) and briefly introduce the device (a *flats*/*dark* panel) that we will use as an example in the following articles of the series.

Introducción a la Sección

Presentamos una serie de cuatro artículos que pretenden facilitar al lector la tarea del diseño, construcción y programación de un dispositivo astronómico que pueda resultarle de utilidad en sus observaciones. En el desarrollo de la serie de artículos se utilizarán diversas tecnologías libres (y en muchos casos gratuitas o de bajo coste) y lenguajes de programación (como Python, Arduino, INDIGO, diseño paramétrico...) que permitan abordar las distintas tareas necesarias para conseguir un dispositivo controlable remotamente y completamente funcional. Los cuatro artículos se organizarán de la siguiente manera:

- Este primer artículo expone algunos de los conceptos necesarios para poder desarrollar nuestro dispositivo y todo el software que permitirá su control usando estándares y tecnologías actuales.

Además se presentará de manera sucinta un dispositivo astronómico que servirá a modo de ejemplo para el desarrollo durante toda la serie.

- El segundo artículo se centrará en el diseño del *hardware*, *firmware* y el protocolo de comunicación con el dispositivo. Se hará uso del software FreeCAD para el diseño de las piezas físicas del dispositivo (que podrán ser impresas en 3D), así como la plataforma Arduino (junto a su lenguaje de programación) para el control de los componentes electrónicos necesarios para el dispositivo.
- El tercer artículo abordará la programación de un *driver* INDIGO que permita el control del dispositivo desarrollado haciendo efectiva la comunicación entre el dispositivo basado en Arduino y el ordenador al que se conectará. Una vez finalizado el *driver* del dispositivo, estaremos en condiciones de poder controlarlo mediante cualquier *software* que implemente el protocolo INDI, como pueden ser KStars, Ekos, AstroDMx Capture, Cartes du Ciel, PixInsight, Stellarium...
- El cuarto y último artículo de la serie planteará la programación de un programa cliente que haga uso de nuestro dispositivo por si los programas de control de dispositivos astronómicos habituales no nos permitieran realizar algún tipo de tarea particular con el mismo (o no lo hagan de manera satisfactoria).

1. Introducción: *hardware*, abstracciones y dispositivos astronómicos

Desde el mismo comienzo de la informática, cuando los primeros diseños sobre hipotéticas máquinas electrónicas capaces de procesar datos y resolver problemas con los mismos empezaban a aparecer, se hacía patente que dichos proto-ordenadores necesitarían de dispositivos de entrada y salida, es decir, mecanismos que les permitieran introducir y extraer la información (los datos y los programas) de los mismos.

Por ejemplo, el mismo John von Neumann en el primer borrador de un informe sobre el EDVAC[1] (uno de los primeros ordenadores electrónicos) donde se describen las bases sobre las que se construirían los primeros ordenadores electrónicos conceptualmente similares a los que manejamos actualmente (arquitectura von Neumann[2]), se mencionan específicamente los dispositivos de entrada y salida.

Evidentemente los dispositivos de entrada y salida han evolucionado exponencialmente junto con el resto de componentes de los ordenadores: desde interruptores básicos y tarjetas perforadas como mecanismos de entrada y luces e impresoras básicas como mecanismos de salida, hasta sensores de alta sensibilidad y resolución (por ejemplo, las cámaras CCD) como dispositivos de entrada y pantallas de alta resolución o incluso impresoras 3D como dispositivos de salida actuales.

De hecho, la amplia variedad de dispositivos de entrada y salida (de cualquier fabricante o incluso de fabricación artesanal) ha provocado que la gestión de todos los dispositivos que podamos llegar a conectar en nuestros ordenadores no sea una tarea trivial. Cuando los primeros ordenadores comienzan a ser accesibles por el público general, si comprábamos algún dispositivo nuevo (pongamos, por ejemplo, una nueva impresora que acabara de salir al mercado) era posible que los programas de ordenador (un *software* de edición de textos, por continuar con el ejemplo) que utilizáramos no fueran capaces de manejar dicho nuevo *hardware* por la sencilla razón de que dichos programas habrían sido creados antes de que el dispositivo en cuestión existiera.

Para solucionar este tipo de casuísticas es por lo que los sistemas operativos modernos[3] tienen la gestión del hardware de los distintos dispositivos que se pueden conectar a nuestros ordenadores como una de sus funciones principales. Esta labor la llevan a cabo mediante la inclusión de la denominada Capa de Abstracción *Hardware* (*Hardware Abstraction Layer, H.A.L.*)[4]. Dicha capa consiste en un mecanismo que permite definir abstracciones de *hardware* de tal manera que los programadores de aplicaciones ya no interactúan directamente con los dispositivos conectados al ordenador, sino que programarán contra una interfaz *software* que hará de "puente" entre el dispositivo físico y la aplicación propiamente dicha.

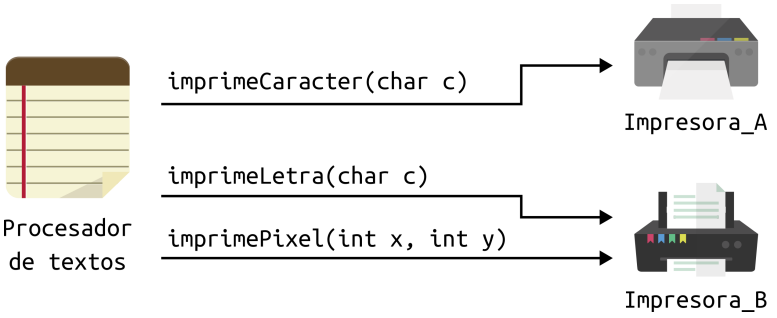

Figura 1. *Esquema simple donde un programa procesador de textos interacciona con dos impresoras: debe conocer el funcionamiento de cada una de ellas. Iconos por SVG Repo.*

Por clarificar la utilidad de utilizar un mecanismo de abstracción de *hardware*, continuaremos con el ejemplo de impresoras mencionado anteriormente: Suponemos que existe una impresora simple (`Impresora_A`) que podemos hacer funcionar invocando una función llamada `imprimeCaracter(char c)`. Dicha función simplemente imprimirá el carácter `c` que le hayamos pasado como parámetro. Supongamos que aparece en el mercado una nueva impresora más avanzada (`Impresora_B`). Al ser de un fabricante distinto, incluye las siguientes funciones que podemos invocar desde nuestro programa:

- `imprimeLetra(char c)`, completamente análoga a la función de la otra impresora (pero distinta, puesto que cada fabricante construye sus dispositivos siguiendo patrones distintos).
- `imprimePixel(int x, int y)`, que imprimiría un punto negro en unas coordenadas `x` e `y` del papel.

Para que un programa de ordenador pudiera manejar ambas impresoras, el programador de la aplicación tiene que conocer el funcionamiento de ambas impresoras (es decir, conocer qué funciones puede usar con la impresora concreta que tiene que utilizar). Esto hace muy compleja la labor del programador, puesto que es imposible estar al tanto de todas las impresoras existentes y su funcionamiento (Fig. 1). En este momento aparece el mecanismo de abstracción de *hardware*: El diseñador del sistema operativo piensa en la problemática que existe con la cantidad de impresoras distintas que existen en el mercado y define una abstracción llamada `Impresora`. Dicha abstracción no es una impresora concreta, ni se refiere a una marca o modelo concreto. Simplemente es la definición genérica de las instrucciones que debe llevar a cabo un dispositivo para que el sistema operativo la considere una impresora. En nuestro caso dicha abstracción podría consistir en la definición de las siguientes funciones:

- `imprimeCadena(String s)`: Esta función debería imprimir una secuencia de caracteres en la impresora.
- `puedeImprimirGraficos(): boolean`: Esta función devolvería `true` si la impresora es capaz de imprimir gráficos y `false` en caso contrario.
- `imprimeGrafico(boolean[][] pixeles)`: esta función imprimiría una gráfico a partir de la información en el array de `pixeles` que se le pasa por parámetro.

Es importante recalcar que la definición abstracta de `Impresora` no ha implementado el como se imprime con ninguna impresora particular. De hecho, solo con esa definición el sistema operativo (ni ninguna aplicación instalada en el mismo) aún no es capaz de imprimir en ninguna impresora concreta puesto que el diseñador del sistema ni de las aplicaciones no tienen por qué conocer como funcionan las impresoras fabricadas por atrás empresas. De hecho, son dichos fabricantes de *hardware* los que

tienen que proporcionar los controladores (comúnmente denominados *drivers*) que serán los encargados de que sus dispositivos sean utilizables a través del sistema operativo. En ese sentido, un *driver* no es ni más ni menos que un "pequeño" programa de ordenador hace de intermediario ("traduce") entre las instrucciones abstractas definidas en la abstracción `Impresora` a las instrucciones concretas que necesita el dispositivo concreto. Por tanto, el fabricante de la `Impresora_A` deberá construir un *driver* cuyas funciones serían:

- `imprimeCadena(String s)`: Esta función hará uso de la función `imprimeCaracter(char c)` de manera iterativa para cada uno de los caracteres de la cadena `s` para imprimir la cadena completa.
- `puedeImprimirGraficos(): boolean`: En este caso devolverá `false`, puesto que la impresora no es capaz de imprimir gráficos.
- `imprimeGrafico(boolean[][] pixeles)`: esta función no haría nada, puesto que la impresora no es capaz de imprimir gráficos.

En el caso del segundo fabricante, deberá proporcionar un driver para la `Impresora_B` de la siguiente manera (usando las funciones propias de dicha impresora):

- `imprimeCadena(String s)`: Esta función hará uso de la función `imprimeLetra(char c)` de manera iterativa para cada uno de los caracteres de la cadena `s` para imprimir la cadena completa.
- `puedeImprimirGraficos(): boolean`: En este caso devolverá `true`, puesto que la impresora si puede imprimir gráficos.
- `imprimeGrafico(boolean[][] pixeles)`: Esta función imprimirá la imagen llamando iterativamente a la función `imprimePixel(int x, int y)` de la impresora para cada pixel definido en `pixeles`.

De esta manera, el programador de una aplicación (por ejemplo un procesador de textos) ya no tiene que conocer como funciona cada una de las impresoras para usarlas. Solo debe conocer la abstracción `Impresora` y manejar las tres funciones que se definen en dicha interfaz, puesto que el sistema operativo se encargará de pasar las llamadas a las funciones de `Impresora` al driver de la impresora que corresponda. Y lo que es más importante, cuando aparezca una nueva impresora en el mercado, podrá ser usada por los programas existentes (sin tener que modificarlos) puesto que dicha impresora implementará exactamente las mismas funciones definidas en la abstracción `Impresora`. Este esquema, más complejo pero mucho más flexible, puede verse esquematizado en la Fig. 2.

Es importante enfatizar que es obligación de los fabricantes de *hardware* proveer los *drivers* para su dispositivo para cada sistema operativo en los que quieran que funcione su instrumento. En ningún caso es tarea del sistema operativo o de una tercera parte el programar dicho *driver* aunque en ocasiones los propios sistemas operativos o comunidades de programación ajenas a los fabricantes hayan desarrollado *drivers* para muchos dispositivos populares, en algunos casos llevando a cabo una labor de ingeniería inversa importante.

La mayoría de los dispositivos actuales que conectamos a nuestros ordenadores tienen algún tipo de interfaz de comunicación para conectarlo. En la actualidad se ha impuesto el interfaz USB (Bus Serie Universal), aunque siguen existiendo otros puertos en uso, como el puerto serie, paralelo, *firewire*, RJ45 (puerto de red), etc. A través de este tipo de interfaces los dispositivos pueden recibir comandos y responder a los mismos usando algún tipo de protocolo de comunicaciones usualmente definido por el fabricante.

En el caso particular de la astronomía, donde los usuarios manejan una gran cantidad de dispositivos muy específicos y no de uso común fuera de este ámbito (como por ejemplo ruedas porta-filtros, enfocadores, monturas, calentadores...), no cabe esperar que los diseñadores de los sistemas operativos

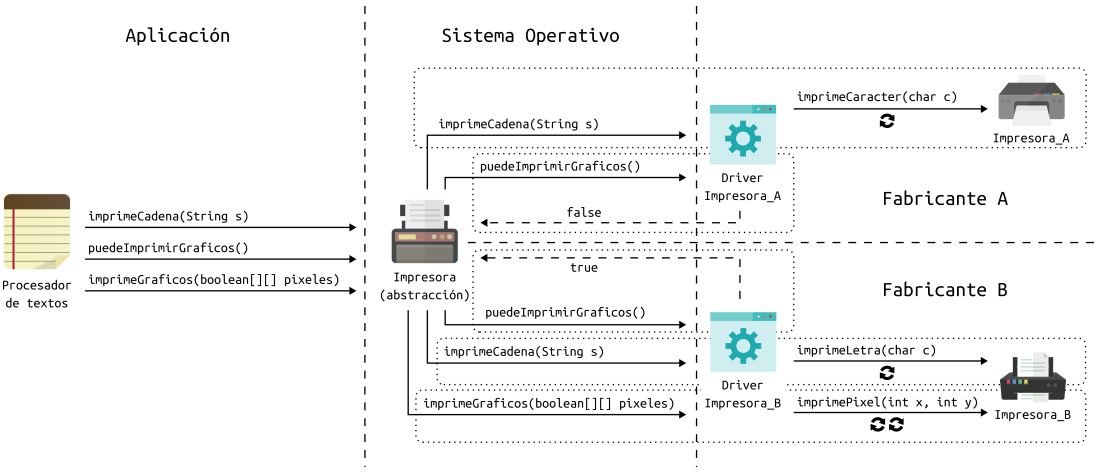

Figura 2. *Esquema de aplicación de una H.A.L.: el sistema operativo provee una abstracción impresora y la aplicación por tanto no tiene que conocer los detalles de implementación de cada impresora, que es tarea específica de los drivers. Iconos por SVG Repo.*

desarrollen abstracciones específicas para cada uno de estas tipologías de dispositivos. Es por ello que han aparecido varias plataformas (la mayoría de ellas libres[10]) que, a un nivel más alto (por encima del sistema operativo), ofrecen características similares a las de una H.A.L. pero centradas específicamente en dispositivos de interés para la comunidad astronómica. Una de esas plataformas es ASCOM [5] que, basada principalmente en el ecosistema Windows, lleva desde 1998 creando una serie de interfaces y abstracciones para poder controlar dispositivos astronómicos. En la actualidad sigue siendo muy utilizado y la mayoría de fabricantes de *hardware* astronómico suelen proporcionar un *driver* ASCOM para sus dispositivos. Sin embargo, la plataforma ASCOM ha estado siempre muy centrada en el control de dispositivos conectados directamente al ordenador donde el usuario ejecuta su *software* de control. Sin embargo, con la evolución rapidísima de las redes de comunicaciones y las facilidades y velocidad que estas nos ofrecen, la astronomía moderna se está transformando y permitiendo de manera cada vez más sencilla controlar instrumental astronómico en remoto (por ejemplo en zonas geográficas menos contaminadas) e incluso de manera distribuida, es decir, manejando de manera simultanea equipamiento que puede encontrarse en diversas localizaciones (para sincronizar observatorios, o simplemente para manejar gran cantidad de instrumental que no podría funcionar conectado a un único ordenador.

Para poder llevar a cabo este tipo de control remoto de instrumental han aparecido varias iniciativas que pretenden facilitar dicha tarea. INDI e INDIGO son algunas de ellas, que describiremos en las siguientes secciones, puesto que serán sobre las que en los siguientes artículos de la serie se desarrollará el código de control de nuestro dispositivo de ejemplo. Pero antes de presentar sendas iniciativas, es de justicia señalar que en el año 2018 los desarrolladores de la plataforma ASCOM, viendo el interés creciente en el control remoto de observatorios (que usualmente los usuarios de ASCOM realizaban mediante algún tipo de aplicación de control remoto de escritorio) empezaron a desarrollar ASCOM Alpaca[6], una iniciativa que permitiría controlar dispositivos remotamente (usando un protocolo similar a `REST` que usa `http` y `json` para comunicarse por la red) e incluso bajo sistemas operativos diferentes a Windows. Sin embargo, dicha iniciativa, pese a que es funcional, todavía no está tan extendida como la plataforma ASCOM original.

2. INDI: una aproximación distribuida al control de dispositivos astronómicos

INDI[7] (*Instrument Neutral Distributed Interface*) es un sistema de control distribuido de instrumental, pensado originalmente para astronomía, pero que en realidad puede usarse para cualquier tipo de dispositivos. Nace en el año 2003 con la publicación de la primera versión del protocolo INDI[8], que está basado en `XML` y, por tanto, es un protocolo fácil de leer y escribir por cualquier lenguaje de programación moderno (existen numerosas bibliotecas que permiten leer y escribir dicho protocolo).

Además del protocolo en sí, con el tiempo se han ido desarrollando diversos paquetes de software que implementan dicho protocolo, como pueden ser diversos clientes (Ekos, Kstars, Cartes du Ciel, Stellarium...) pero, sobretodo la biblioteca INDILib[9] que es una implementación de referencia, basada en el lenguaje de programación `C` y disponible para sistemas operativos de tipo UNIX (como por ejemplo GNU/Linux) que no solo implementa el protocolo de comunicación INDI, sino que también proporciona algunas herramientas fundamentales como son un Servidor INDI y la mayoría de los *drivers* INDI disponibles para el instrumental astronómico. Es interesante reseñar que existen implementaciones del protocolo INDI que permiten la creación de *drivers*, clientes y servidores en otros lenguajes como Java[11] o Python[12].

En lo que sigue de esta sección haremos una descripción general del sistema INDI, sin prestar especial atención al protocolo de comunicación en sí, puesto que dicho protocolo no es necesario conocerlo para poder usar e incluso programar nuestros propios clientes o *drivers* INDI.

2.1. Arquitectura de la plataforma INDI

INDI nace asumiendo desde el principio que el control de los distintos dispositivos de los que disponemos se hará de manera distribuida. Es decir, los dispositivos que queramos controlar no tienen por qué estar conectados en el propio ordenador desde donde deseamos controlarlos. De hecho, la propia arquitectura permite de manera simple controlar dispositivos que pueden estar conectados a varios ordenadores en diversas localizaciones, siempre y cuando haya un enlace de red a los mismos. Incluso se pueden controlar y monitorizar los distintos dispositivos desde varios ordenadores diferentes.

Para conseguirlo, INDI distingue tres actores software distintos:

- **Cliente INDI:** Es el programa de ordenador que maneja el usuario que quiere controlar los dispositivos. Usualmente tendrá algún tipo de interfaz de usuario (por ejemplo una ventana gráfica o un interfaz en línea de comandos).
- ***Driver* INDI:** Es un programa de ordenador que se encarga del controlador de un dispositivo concreto. Dicho programa cumple con la función clásica de los *drivers* de dispositivo en los sistemas operativos: usando una serie de abstracciones permite controlar un dispositivo concreto.
- **Servidor INDI:** Es el programa de ordenador que permite la comunicación (hace de intermediario) entre los clientes y los *drivers*. El servidor estará instalado en el ordenador (o los ordenadores) al que estén conectados los dispositivos. Además se encarga de ejecutar los *drivers* y es el que permite múltiples conexiones de clientes para controlar los dispositivos.

En la Fig. 3 se muestra un esquema simple de conexión entre clientes, servidores y *drivers* INDI. En este caso 2 clientes distintos (uno en un ordenador portátil y otro en un dispositivo móvil) quieren supervisar y controlar tres dispositivos astronómicos distintos: Una cámara, una montura y una cúpula. Es interesante resaltar que los clientes INDI pueden tener naturalezas muy distintas: pueden ser una aplicación web, una aplicación "de escritorio" (para casi cualquier sistema operativo), una App de un dispositivo móvil, etc. Por motivos de organización del observatorio a controlar, la cámara está conectada a un ordenador (con un interfaz USB rápido), mientras que la montura y la cúpula estarán conectadas a otro ordenador distinto. En sendos ordenadores se instarán y lanzaran dos instancias del

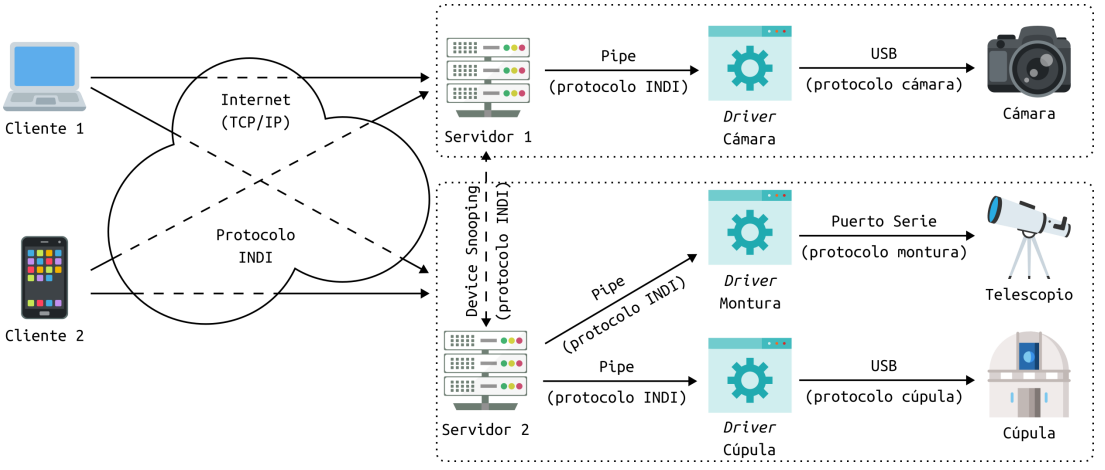

Figura 3. *Iconos por SVG Repo.*

servidor INDI. Cada una de ellas ejecutará y se comunicará con los *drivers* INDI para los dispositivos conectados. Dicha conexión entre los servidores y los *drivers* se lleva a cabo a través de *pipes* [14] (tuberías) estándar de UNIX. En este ejemplo asumimos que los clientes (tanto el portátil como el dispositivo móvil) no se encuentran ni siquiera en la misma red local que los servidores y que se conectan a los mismos a través de Internet usando *sockets* (TCP/IP) estándar. En la figura también viene indicada una conexión entre los dos servidores INDI que permite el *device snooping*, es decir, que un dispositivo pueda obtener información de otro distinto de manera directa, sin tener que pasar a través de ningún cliente. Por ejemplo, una cámara podría obtener información directamente de la montura para poder rellenar la información de las cabeceras del fichero FITS que va a devolver. Pese a que las conexiones entre los distintos componentes de la arquitectura INDI pueden ser de naturaleza distinta (sockets, pipes...), la comunicación entre los mismos siempre se hace usando el protocolo INDI, lo que facilita la programación de los distintos elementos.

2.2. Los dispositivos y drivers INDI

Una vez conocida la arquitectura general que propone INDI debemos conocer como INDI crea las abstracciones de hardware para poder construir los drivers de un dispositivo concreto. Para INDI un dispositivo es una colección de lo que llama *propiedades* (*property*). Una propiedad es un atributo del dispositivo. Existen 5 tipos distintos de propiedades que se pueden definir en INDI:

- **Cadenas de texto (*string*):** Representan una propiedad textual. Por ejemplo, puede ser el nombre del dispositivo, la versión del mismo, la ruta del puerto donde debe conectar el ordenador para comunicarse con el dispositivo...
- **Números:** Representan cualquier característica que pueda definirse mediante un número. En principio los números en INDI están definidos en coma flotante, aunque se permite especificar cual es el mínimo, máximo y paso de la propiedad. Por ejemplo, podríamos definir una propiedad numérica que pudiera tomar como valor mínimo `-10.5`, máximo `14.7` y el paso entre un valor y otro fuera `0.1`. Dada la importancia del sistema de numeración sexagesimal en astronomía, los números pueden expresarse también en grados (u horas), minutos y segundos.
- **Luces:** Representan características que pueden representarse por cualquiera de las siguientes etiquetas:

  - `OK`: La propiedad está encendida, activada. Por ejemplo, hemos activado un interuptor de un dispositivo. Normalmente se identifica este estado con el color verde ●.
  - `BUSY`: La propiedad está ocupada. Por ejemplo, está moviéndose un dispositivo y todavía no está listo para funcionar. Normalmente se identifica este estado con el color amarillo ●.
  - `ALERT`: La propiedad indica que ha ocurrido algo erróneo o algún tipo de problema o inconsistencia. Por ejemplo, se ha activado algún fin de carrera del dispositivo al que no debería haberse llegado. Normalmente se identifica con el color rojo ●.
  - `IDLE`: Indica un estado "no inicializado" o "inactivo". Normalmente se identifica con el color gris ●.

- **Opciones (*Switch*)**: Representan características que solo pueden tomar algunos valores prefijados. Por ejemplo, una propiedad de este tipo podría ser `Sensibilidad ISO` y que pudiera tener uno de los siguientes valores: `100`, `200`, `400`, `800`, `H1`, `H2`. Además las propiedades de tipo *switch* tienen una regla asociada que determina cuantos valores pueden o deben seleccionarse de entre sus opciones: *una de muchos* (debe seleccionarse una y solo una de las opciones), *como mucho una* (puede seleccionarse una opción o ninguna) y *cualquiera de muchas* (se puede seleccionar ninguna, una o más de una de las opciones).
- **Datos binarios (*BLOB*)**: Cualquier tipo de dato binario que no pueda codificarse fácilmente con las propiedades anteriores. El ejemplo típico son los datos asociados a los píxeles captados por una cámara (es decir, la imagen en sí captada por la cámara).

Por afinar un poco más, hay que mencionar que en realidad una propiedad es una colección de uno o más valores (*values*) de alguna de las 5 naturalezas presentadas anteriormente: no tiene por qué referirse a un único valor. Esto es especialmente interesante para, por ejemplo, los datos que deben estar agrupados de manera natural. Por ejemplo, para una montura puede existir una propiedad numérica llamada `EQUATORIAL_EOD_COORD` que representa la ascensión recta (AR, un número) y declinación (dec, otro número) a la que está apuntando en ese momento.

Además, las propiedades pueden definirse como "de solo lectura" (*read only*), es decir, que el cliente no puede pedir que cambien los valores de la misma (típico de propiedades que representan el estado de un sensor), "de lectura y escritura" (*read/write*), que los clientes si pueden pedir su modificación y de "solo escritura" *write only* (algo bastante más inusual). Por último mencionaremos que las propiedades pueden cada una de ellas tener un estado propio ● `OK`, ● `BUSY`, ● `ALERT` o ● `IDLE` que determina si el valor de la misma es válido, está cambiando, tiene algún problema o aún no está inicializado.

Cuando un cliente INDI conecta con un servidor INDI, este último le manda una lista con todas las propiedades que tienen disponibles cada uno de los *drivers* que el gestiona. De esa manera, totalmente dinámica, el cliente averigua cuales son las capacidades de cada uno de los dispositivos a los que puede conectar. Este dinamismo, en el que a priori el cliente no conoce cuales son las características de los dispositivos que maneja permite el desarrollo de clientes que van a ser capaces de interactuar perfectamente con futuros dispositivos que aún pueden no haberse desarrollado sin tener que actualizarse explícitamente.

Los clientes pueden pedir a los dispositivos que cambien el valor de sus propiedades, lo cual puede desencadenar que el dispositivo ejecute alguna acción concreta. Por ejemplo, si el dispositivo que estamos controlando es una cámara y tiene una propiedad llamada `CCD_EXPOSURE` que expresa el tiempo de exposición que queremos para una toma (en segundos), el cliente puede pedirle al *driver* que cambie dicho valor a `10.5` si queremos una toma de dicha duración. En ese momento el *driver* actualizará el valor de dicha propiedad y se lo notificará al cliente y comenzará la nueva exposición. Durante la exposición el *driver* puede mandar mensajes al cliente especificando que el tiempo de exposición cada vez es más pequeño (y además puede haber señalizado que dicha propiedad está ocupada (● `BUSY`) (va

quedando menos tiempo para terminar la toma). Una vez finalizada la toma, la propiedad volverá al valor `0`, estado `OK` y la propiedad de tipo *BLOB* que contiene los datos RAW de la imagen también se actualizará con la imagen recién capturada.

Para conseguir que los clientes puedan interpretar y presentar un interfaz de usuario adecuado al usuario (no es lo mismo manejar una cámara que una montura, que una rueda porta-filtros), existen una serie de propiedades estándar que deben definirse cuando desarrollamos un *driver* de una tipología concreta de dispositivo. Por ejemplo, si queremos desarrollar un *driver* para una rueda porta-filtros, debemos implementar dos propiedades: `FILTER_SLOT` (numérica), que determina cual es el filtro seleccionado en cada momento y `FILTER_NAME` (cadena de caracteres) que determina el nombre del filtro en cada hueco de la rueda.

3. INDIGO: Evolucionando INDI

INDIGO [15] aparece en 2016 como una escisión del proyecto INDI. Este nuevo proyecto presenta algunas novedades y diferencias que pueden suponer una ventaja frente al uso o desarrollo de aplicaciones bajo INDI. Entre otras cabe citar:

- **Cambios en la licencia:** INDILib (la implementación de referencia de INDI) tiene una licencia de código abierto. Concretamente utiliza la licencia GNU LGPL v2.1 [16]. Dicha licencia ofrece muchas libertades al público que quiera modificarla, usarla (incluso en proyectos remunerados), distribuirla, etc. Sin embargo hay un aspecto de dicha licencia que no gustaba a los desarrolladores principales de INDIGO. La licencia GNU LGPL exige que si la usas, el código fuente del *software* que se distribuya que haya utilizado la biblioteca debe también liberarse. Este aspecto viral de la licencia (que obliga a mantener el software que se desarrolle libre y accesible para el resto de la comunidad) "impide" en cierta manera obtener una remuneración por el *software* desarrollado según los esquemas típicos (vender la aplicación), ya que el software libre permite a cualquiera compilar el código fuente y obtener una copia funcional del software sin ningún coste ni restricción. Por tanto, la licencia de INDIGO no exige la publicación del código fuente del software que se haya desarrollado usándola (en caso de que sea un software que se distribuya).
- **Mejoras de rendimiento:** Dado que la comunicación entre los clientes, servidores y *drivers* INDI se realiza usando usando el protocolo INDI y mediante el uso de *sockets* de red o *pipes* entre procesos cuando varios de dichos componentes se encuentran en la misma máquina (lo cual es común en configuraciones simples donde no hace falta un control remoto de dispositivos) INDI es ineficiente y, de hecho, poco útil para aplicaciones que exijan una alta velocidad (como por ejemplo *lucky-imaging* o fotografía planetaria). Por ello la implementación base de INDIGO permite que la comunicación entre los componentes pueda hacerse directamente en la memoria del ordenador lo cual incrementa la eficacia en varios ordenes de magnitud. Pese a la aparente ventaja no hay que perder de vista que dicho aumento de eficiencia NO es posible en un esquema distribuido.
- **Mejoras en la transmisión de los datos binarios:** Los datos binarios (como las imágenes capturadas por las cámaras) se pueden transmitir de manera más eficiente a los clientes usando un servidor `http` que incorpora el servidor INDIGO. Además se desacopla el envío de dichos datos del resto de información (resto de propiedades) del dispositivo.
- **Incorporación de `JSON` al protocolo de comunicación:** Con INDIGO se puede optar por mandar la información tanto en `XML` (como hace INDI) como en `JSON` [17], un formato más actual de codificación de la información. Esto puede facilitar o hacer más cómoda la creación de componentes de INDIGO en otros lenguajes de programación como Javascript.

- **Racionalización de los nombres:** Los nombres de las propiedades estándar de INDI y de los *drivers* se han ajustado para que sean más coherentes entre sí y facilitar su utilización. En [18] se puede ver la lista de propiedades estándar para cada tipo de dispositivo astronómico existente.
- **Retrocompatibilidad:** Pese a la cantidad de cambios introducidos, INDIGO mantiene una alta compatibilidad con INDI. Es decir, es posible usar un cliente INDI para conectar con un servidor INDIGO o a la inversa. Eso implica que, por ejemplo, si aparece un nuevo *driver* INDI para un dispositivo que aún no tiene su contrapartida programada en INDIGO, es perfectamente posible usarlo en esta nueva plataforma.

Por algunas de las razones mencionadas los autores hemos elegido esta plataforma para el desarrollo de esta serie de artículos sobre la construcción y programación de dispositivos astronómicos. En la siguiente sección finalizaremos con una breve descripción del dispositivo que, a modo de ejemplo, vamos a usar para esta tarea.

4. Descripción general del dispositivo astronómico a desarrollar

Desde la aparición de INDI, poco a poco se han ido desarrollando los *drivers* para el control de muchos dispositivos comerciales, ya sean desarrollados por las propias compañías para dar soporte a esta plataforma o por desarrolladores independientes que necesitan un *driver* de control específico para un cierto dispositivo y así poder controlarlo desde su cliente.

Debido a la libertad de desarrollo de *drivers* que ofrecen tanto INDI como INDIGO, han empezado a surgir proyectos basados en microcontroladores para poder interactuar con dispositivos usados comúnmente en astronomía (motores o servomotores, *encoders*, finales de carrera, etc). Uno ejemplo bastante conocido es el de la motorización del control de foco en un telescopio con el proyecto conocido como `myFocuserPro2` [13]. Éste proyecto está basado en la platafroma Arduino para construir la electrónica y *firmware* del dispositivo. Además del propio dispositivo los autores han desarrollado *drivers* tanto para la plataforma INDI como para ASCOM.

Para algunas tareas típicas en astronomía puede hacer falta el control de pequeños dispositivos como motores, servomotores, u otros actuadores que pueden ser difíciles (o caras) de obtener como soluciones comerciales (por lo particular que puede ser nuestro equipamiento astronómico). Es por ello que el movimiento "hazlo to mismo" (Do It Yourself, DIY) está cobrando bastante importancia y mucha gente desarrolla sus propios controladores de monturas, enfocadores, ruedas porta-filtros, rotadores, control de cúpulas, etc., aunque en realidad el uso de microcontroladores puede tener otras muchas aplicaciones en el ámbito de la astronomía.

En esta serie de artículos (y a modo de ejemplo) nos centraremos en crear un panel de *flats*/*darks* robótico (Fig. 4), que además va a poder usarse como tapadera para resguardar las ópticas mientras el equipo no esté en funcionamiento.
Nuestro panel de *flats* tendrá dos componentes electrónicos fundamentales que controlaremos con una electrónica basada en Arduino:

1. Un motor servo digital DS RDS3115MG. Permitirá abrir/cerrar la tapa que se posicionará en la entrada del telescopio.
2. Módulo de control PWM u optoacoplador ILD213T. Ésto nos permitirá controlar la luz del propio panel de *flats* independientemente del voltaje que requiera.

Debido a la gran variedad de telescopios existentes en el mercado, tendremos que adaptar nuestro panel de *flats* con piezas creadas ex profeso. Por este motivo diseñaremos las mismas mediante un software de modelado paramétrico (FreeCAD) y luego las imprimiremos con una impresora 3D.

Figura 4. *Panel automático de flats en telescopio Skywatcher Esprit 120 ED.*

En el siguiente artículo describiremos como podemos comenzar a construir nuestro dispositivo, partiendo desde los componentes que necesitamos, programar el *firmware* en el microcontrolador e implementar un protocolo de comunicación. En los siguientes artículos nos centraremos en la programación del driver oportuno para INDIGO y la creación de un cliente específico.

5. Conclusiones

En este primer artículo de la serie dedicada al diseño, construcción y programación de dispositivos astronómicos hemos hecho una introducción a como los sistemas operativos de nuestros ordenadores utilizan mecanismos de abstracción para ser capaces, mediante el uso de *drivers*, de controlar distintos dispositivos periféricos que conectemos a los mismos. En el caso particular de las astronomía, donde la especificidad de los dispositivos hace que los desarrolladores de sistemas operativos no incluyan soporte directo para una amplia variedad de dispositivos como pueden ser ruedas porta-filtros, enfocadores, monturas, etc, hace falta la utilización de plataformas software de más alto nivel como son ASCOM, INDI o INDIGO que permitan un control y programación de nuestro instrumental.

Hemos presentado con más detalle dos de dichas plataformas (INDI y INDIGO). Ambas están pensadas y diseñadas desde el principio teniendo en cuenta que una de las tendencias actuales en astronomía es el control remoto y distribuido del instrumental. Particularmente, en los futuros artículos de la serie haremos uso de INDIGO como plataforma de desarrollo de los *drivers* de nuestros dispositivos.

Por último, hemos realizado una somera descripción del dispositivo que, a modo de ejemplo, desarrollaremos en los siguientes artículos de la serie: un panel de *flats*/*darks* para nuestro telescopio.

Referencias

[1] von Neumann, John (1945), First Draft of a Report on the EDVAC. Disponible en https://web.archive.org/web/20130314123032/http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf

[2] Wikipedia - Arquiectura de von Neumann. Disponible en: https://es.wikipedia.org/wiki/Arquitectura_de_Von_Neumann
[3] Tanenbaum, Andrew (2023), Modern Operating Systems, Global Edition. Pearson Education Limited. ISBN: 978-1292459660.
[4] Wikipedia - Hardware abstraction. Disponible en: https://en.wikipedia.org/wiki/Hardware_abstraction
[5] ASCOM. Astronomy Common Object Model. Disponible en: https://ascom-standards.org/
[6] ASCOM Alpaca. Disponible en: https://ascom-standards.org/AlpacaDeveloper/Index.htm
[7] Wikipedia. Instrument Neutral Distributed Interface. Disponible en: https://en.wikipedia.org/wiki/Instrument_Neutral_Distributed_Interface
[8] Downey, Elwood Charles (2007), INDI: Instrument-Neutral Distributed Interface, Protocol Version 1.7, Document Version 1.3. Disponible en: https://clearskyinstitute.com/INDI/INDI-1.7-1.3.pdf
[9] INDI Lib. Open Astronomy Instrumentation. Disponible en: https://indilib.org/
[10] Free Software Foundation. ¿Qué es el software libre?. Disponible en: https://www.gnu.org/philosophy/free-sw.es.html
[11] INDIForJava. Disponible en: https://indiforjava.github.io/
[12] INDI Python Bindings. Disponible en: https://www.indilib.org/develop/indi-python-bindings.html
[13] Arduino ASCOM Focuser Pro2 DIY. Disponible en: https://sourceforge.net/projects/arduinoascomfocuserpro2diy/
[14] Wikipedia. Pipeline (Unix). Accesible en: https://en.wikipedia.org/wiki/Pipeline_(Unix)
[15] INDIGO Astronomy. Accesible en: https://www.indigo-astronomy.org/
[16] GNU Lesser General Public License, version 2.1. Accesible en: https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html
[17] Wikipedia. JSON. Accesible en: https://es.wikipedia.org/wiki/JSON
[18] INDIGO properties. Accesible en: https://github.com/indigo-astronomy/indigo/blob/master/indigo_docs/PROPERTIES.md