Revisión	Fecha	Comentario	Autor
0	12/07/2010		Ulises Bigliati

Índice

1. Introducción

El **ConnectPort X4,** fabricado por Digi, es un dispositivo que permite obtener la funcionalidad de un gateway con la capacidad de comunicar dispositivos remotos miembros de redes ZigBee con redes TCP/IP, o directamente con una PC (puerto serie mediante).

Tal como su nombre lo indica, el X4 presenta cuatro Interfaces de comunicación (ZigBee, Ethernet, RS232, USB) cuyo funcionamiento puede ser íntegramente administrado mediante la interfaz web del equipo.

La funcionalidad de gateway se consigue utilizando el lenguaje de programación Python desde un entorno de desarrollo basado en Eclipse, pues el X4 posee, embebido en su sistema, el intérprete y máquina virtual necesarios para ejecutar programas escritos en dicho lenguaje.

De este modo, el desarrollador obtiene la flexibilidad necesaria para adecuar el funcionamiento del hardware de comunicaciones a sus necesidades y administrar los diferentes flujos de datos y eventos provenientes de los dispositivos remotos de una red ZigBee que converge en el X4 y que luego se distribuye sobre TCP/IP (o eventualmente mediante el puerto serie).

2. Objetivos de esta nota

Mediante este trabajo nos proponemos comprobar la flexibilidad de adaptación del *ConnectPort X4 ZB* a las necesidades del usuario en el marco de la integración de redes ZigBee con redes TCP/IP. Intentaremos verificar su facilidad de uso y capacidades para proveer soluciones completas con un mínimo esfuerzo de desarrollo.

Nos ocuparemos fundamentalmente del *ConnectPort X4 ZB* en sí mismo en interacción con módulos XBee ZB (módulos ZigBee)[1] valiéndonos de las herramientas de desarrollo que el fabricante provee gratuitamente desde su sitio web.

Vamos a generar la integración de un dispositivo XBee remoto sobre TCP/IP para obtener una aplicación en la cual suceda simultáneamente todo lo siguiente:

1. Que el XBee actúe como cliente de un servidor TCP (datalogging en formato CSV).
2. Que exista una página web utilizando AJAX para monitoreo y control de I/O del XBee.
3. Que el XBee actúe como servidor para proporcionar datos vía XML-RPC (XML vía HTTP request's)

3. Fuera de alcance

Queda fuera de alcance de esta nota todo aspecto referido a la configuración y detalles de utilización de módulos ZigBee como así también todo aspecto de configuración avanzada del ConnecCore X4, siendo remitido el lector a los respectivos manuales provistos por el fabricante, a los cuales haremos referencia bajo el siguiente subtítulo.

4. Herramientas y documentación básicas

La documentación y herramientas de desarrollo pueden encontrarse íntegramente en la sección de documentación del kit de desarrollo[2] del *ConnectPort X4 ZB*: http://www.digi.com/idigi_prof_zb
Allí encontraremos gran cantidad de documentación, pero por el momento será suficiente con lo siguiente:

4.1. ConnectPort™ X Family User's Guide.

Es el manual de usuario para configuración del gateway:

[1] Módulos 802.15.4, DigiMesh, y otras variantes de módulos no están actualmente soportados por el entorno de desarrollo.
[2] El kit de desarrollo basado en el ConnectPort X4 ZB es denominado "iDigi™ Professional Development Kit ZB". Además del "X4", incluye dos módulos XBee en sendas placas de evaluación y demás accesorios para facilitar las pruebas.

4.2. Digi Python Develop Environment
Es el entorno de desarrollo para programar la funcionalidad del gateway.

4.3. Device Discovery Application
Adicionalmente podemos descargar una utilidad de Digi para descubrir dispositivos dentro de una LAN cuando no se conoce su dirección IP, sin embargo esta aplicación esta también incluída como herramienta del entorno de desarrollo.

4.4. Product Manual: XBee / XBee-PRO ZB OEM RF Modules
Desde la misma página web también existe un enlace para descargar el manual de los módulos XBee, aunque esto queda fuera del alcance de esta nota.

4.5. Hoja de datos ConnectPort X family
La hoja de datos del producto puede encontrarse en la sección :
products -> Drop in Networking -> Gateways -> ConnectPort® X Gateways del sitio web de Digi:
http://www.digi.com/products/wirelessdropinnetworking/connectportxgateways.jsp#docs

4.6. Manual del hardware en línea
Cabe destacar que dentro de la interfaz web del equipo también se puede encontrar un completo manual en línea.

4.7. Manual del entorno de desarrollo en línea
También puede ser de gran ayuda (de hecho es recomendable consultarlo) el manual en línea presente en el entorno de desarrollo. (`Help -> Help Contents`)

4.8. Placa de evaluación *XBib development board*
Por último, cabe mencionar que para las pruebas utilizaremos una placa de evaluación para XBee que viene incluida en el kit de desarrollo del X4.

5. Características del hardware

El *ConnectPort X4* dispone de las siguientes interfaces:

Puerto serie:
RS-232 DB-9 hasta 230 Kbps con todas las señales disponibles: TXD, RXD, RTS, CTS, DTR, DSR y DCD.
Se puede configurar para ser utilizado en diferentes modos, entre los que se destacan:
Industrial Automation Protocol, mediante el cual es posible acceder a una red modbus y monitorear/controlar RTU's.
RealPort, para realizar una conexión serial transparente desde un host remoto.
Otros perfiles de funcionamiento del puerto serie pueden ser: *Console Management, TCP Sockets, UDP Sockets, Serial Bridge, Local Configuration, Modem Emulation, DialServ, Custom.*

Puerto USB:
USB tipo A (host).
Este puerto está destinado a la conexión de una cámara Digi WatchPort V2 USB. Luego es posible obtener el streaming de video mediante TCP en un servidor a especificar por parte del usuario.
También se puede obtener una instantánea accediendo a la siguiente URL:
http://device-ip/FS/dev/camera/0.

Ethernet:
RJ-45 10/100Base-T.
Es obviamente el puerto que brinda la conectividad TCP/IP para todos los servicios que ofrece el equipo. Cabe destacar que también se trata del puerto de conexión y debug para el entorno de desarrollo, es decir, que las aplicaciones escritas en Python desde el IDE (Integrated Development Environment) se envían sobre Ethernet y se interpretan y ejecutan directamente en el X4.

ZigBee:

Coordinador XBee Pro ZB en modo API.
Es posible acceder al módulo coordinador de la red ZigBee que estamos administrando, configurarlo y a través de aquel, también podemos acceder a los módulos remotos asociados, sean routers o end devices. También es posible la actualización del firmware, tanto del módulo coordinador como de los módulos remotos.

6. Configuración básicas

Ya sabemos que el dispositivo que estamos analizando es básicamente un gateway entre redes ZigBee y redes IP, por lo tanto, las configuraciones básicas que debemos realizar son las relacionadas con estas dos interfaces de red.

6.1. Configuración IP

Para comenzar a utilizar el equipo bastará conectarlo mediante su interfaz Ethernet a una red TCP/IP donde exista un servidor DHCP. A continuación podemos utilizar la utilidad *Device Discovery* referida en el *punto 4* y con ella descubriremos la direción IP que ha adquirido el X4 tal como puede verse a la derecha.

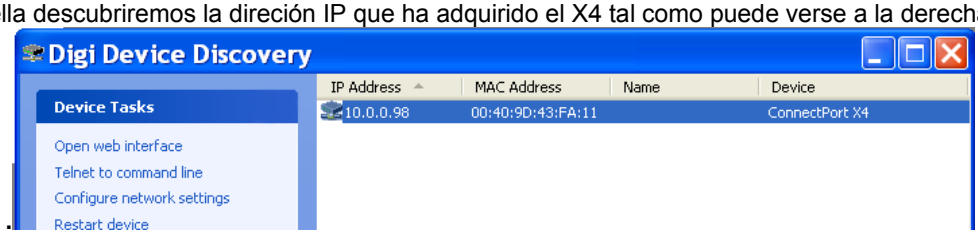


Fig. 1

Es recomendable especificar una dirección IP fija para poder acceder rápidamente en sucesivas operaciones, para ello, desde la interfaz web ir a "Configuration->Network" y establecer la dirección IP y la máscara de subred. Si requerimos acceso al exterior de la LAN será necesario indicar el default gateway.

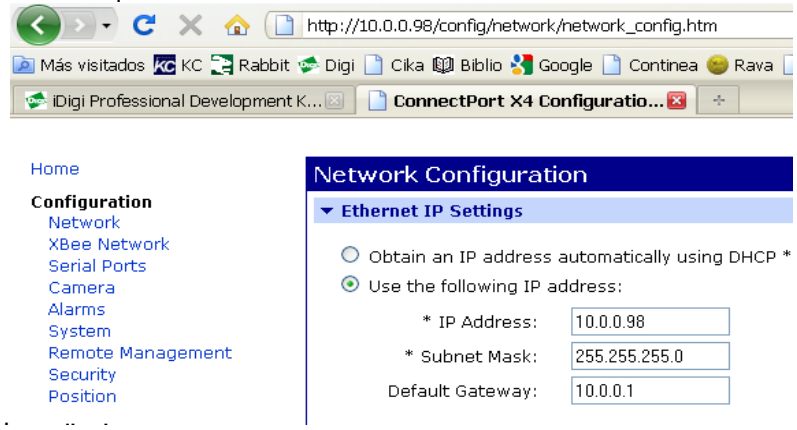


Fig. 2

6.2. Configuración ZigBee

Por el lado de la red ZigBee, el X4 actúa como el nodo Coordinador y como tal es responsable de establecer el canal de operación y el PAN ID de la red ZigBee. Nos dirigimos entonces a *Configuration->XBee Network* para acceder a las configuraciones. Allí veremos el nodo *Coordinador* que está integrado. Daremos una recorrida por los parámetros más importantes a fines de proporcionar una idea básica para la configuración inicial de la red Zigbee con la finalidad de ponernos a trabajar rápidamente en la definición de la funcionalid del gateway que es la parte central de esta nota.

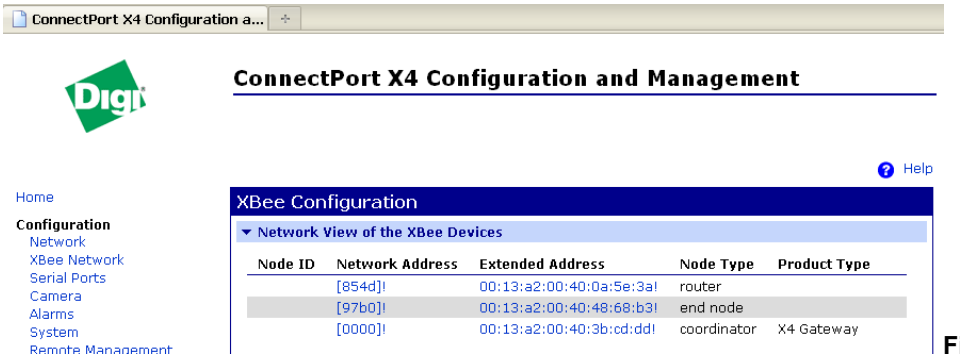


Fig. 3

6.2.1. Parámetros de red básicos

PAN ID (**ID**)

El PAN (Personal Area Network) ID, es el identificador de la red ZigBee. Si se establece en 0, se seleccionará un PAN ID aleatorio. Los Routers o End devices buscarán asociarse a una red que tenga su mismo PAN ID, salvo que esté configurado en 0, en cuyo caso se asociará a cualquier PAN ID que encuentre.

Node Identifier (**NI**)

Un string identificador del nodo, se devuelve dentro de los datos de respuesta del comando ND (Node Discover).

Discover Timeout (**NT**)

Define el tiempo que un nodo destinará en el descubrimiento de otros nodos cuando un Node Join o un Node Discover está en curso.

Scan Channels (**SC**)

Es el mapa de bits del listado de canales a escanear. El Coordinador elige uno de esos canales al iniciar la red. Si se trata de un Router o End device, el mapa de bits es la lista de canales que se escanearán para buscar Coordinadores o Routers.

Scan Duration (**SD**)

Sets the scan duration exponent of the Active and Energy Scans (on each channel) that are used to determine an acceptable channel and Pan ID for startup of the Coordinator.

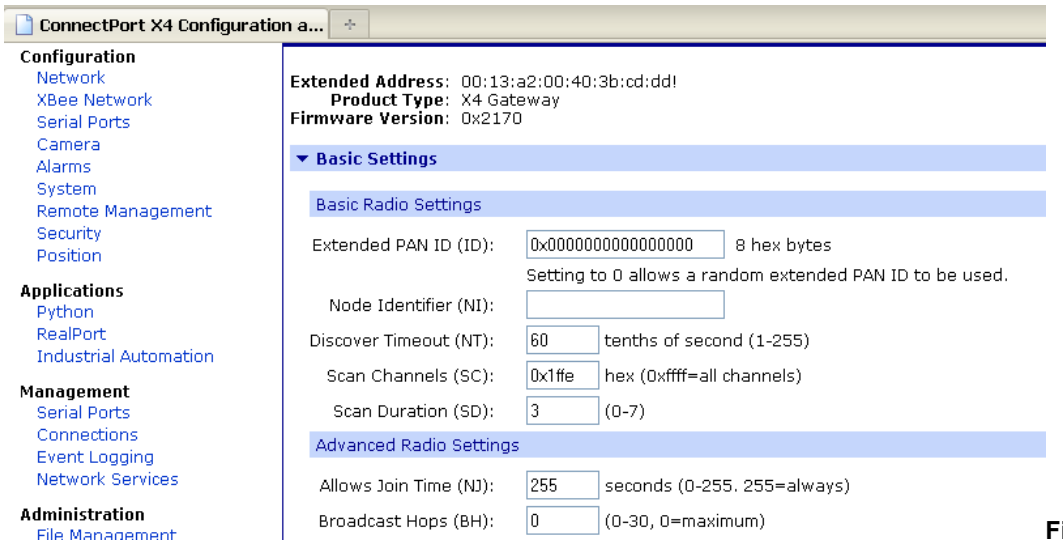


Fig. 4

En nuestro caso, utilizamos la configuración por defecto y tenemos una red XBee conformada de la siguiente manera:

Router: [854d]! 00:13:a2:00:40:0a:5e:3a!

Coordinador [0000]! 00:13:a2:00:40:3b:cd:dd! (X4)

Para mayores datos sobre como operar redes XBee ver documento referido en **4.4.**

7. El entorno de desarrollo

Si se ha descargado el instalador (según el punto 4.2) tras su ejecución se habrá instalado en la PC de desarrollo el ***Digi ESP for Python Development*** junto con el lenguaje de programación **Python 2.4.3.**

Al desarrollar una aplicación y ejecutarla desde el IDE, los scripts y las libraries necesarias se enviarán a través de Ethernet, directo al file system del X4, desde donde se leerá, interpreta y ejecutará el programa y a su vez se podrá realizar el debug.

Las ventajas que obtenemos trabajando sobre esta plataforma, es que como desarrolladores, podemos obviar los detalles de las tramas API de los módulos XBee observándolos simplemente bajo la forma de diferentes abstracciones, como por ejemplo, un conjunto de simples elementos XML representando el valor de sus diferentes I/O.
Por otra parte, la implementación de aplicaciones utilizando un lenguaje interpretado como Python implica obtener ventajas en cuanto a portabilidad, esto último permite diseñar diversas aplicaciones que pueden ejecutarse tanto en PC como en el X4 y compatibles, es decir en cualquier plataforma que disponga de un intérprete Python y de las libraries necesarias (obviamente salvando los detalles de referenciación a hardware especializado). La facilidad de uso es otro factor que normalmente se asocia al lenguaje en cuestión.
Debemos reconocer que las ventajas citadas arriba juegan en perjuicio de la eficiencia. Lo cual significa que habrá inevitables pérdidas de rendimiento (aunque quizá imperceptibles para el usuario) que son producto de la intermediación del *interprete de Python* que procesa el *código fuente* para generar el *bytecode* para su *máquina virtual*, la cual finalmente lo ejecuta traduciéndolo al *código máquina* específico de la plataforma. Sin embargo, este aspecto debería ser compensado al menos en parte gracias a la potencia de procesamiento del hardware subyacente.

7.1. Configuraciones previas

Tras la instalación, y luego de iniciar por primera vez el entorno de desarrollo, (y luego periódicamente) es recomendable ejecutar la utilidad *Package Manager* que nos permite obtener cualquier tipo de actualización disponible para este IDE.

Asumiendo que nuestro X4 está debidamente conectado, vamos al menú principal, y en *Device Options -> Device Manager* presionamos el botón "*Device Discovery*" (a la izq. y arriba) y tras unos instantes se desplegará una ventana con el listado de los dispositivos descubiertos. Seleccionamos nuestro *ConnectPort X4* y presionamos "*Create configuration*". Veremos como el equipo se añade al árbol de la izquierda de la ventana y a su derecha, al seleccionarlo en el árbol, se muestra la ficha con sus configuraciones:
- Tipo de dispositivo (*ConnectPort X4*) y - Modo de conexión (local área network)
- Dirección IP y - Puerto Telnet
Son las importantes (al menos en el marco de este trabajo)

8. iDigi Dia

El concepto fundamental de la plataforma que estamos describiendo gira en torno a un framework llamado *iDigi Dia* (***Device Integration Application***). La idea de este framework es permitirle al usuario trabajar con abstracciones para generar verdaderas soluciones para aplicaciones donde se requiere la obtención de datos y/o el control y/o el monitoreo de diversos dispositivos remotos.

La solución de software es creada conceptualmente por el usuario dentro de este framework, y luego es compilada para materializarse en código fuente Python que se ejecuta sobre la familia de gateway's de Digi. El framework ofrece una solución rápida para la construcción de arquitecturas ya testeadas que resulta ventajoso cuando se compara con el intento de diseñar, implementar, probar y mantener los componentes de una aplicación distribuida desde el principio.

8.1. Arquitectura

Cómo se logra lo anticipado líneas arriba? El framework realiza la abstracción de tres subsistemas primarios:

- Device drivers
- Channels
- Presentations

- El núcleo conceptual del framework *Dia* es la base de datos de canales (***Channel Database***).
- Cada dispositivo remoto se encapsula en un ***Device Driver*** responsable de abstraer aquel dispositivo del mundo real en una serie de magnitudes conocidas como *propiedades* cuyos valores se exponen bajo la forma de diferentes tipos y número de **canales** de diversa naturaleza (existen definidos una gran variedad de ellos, por ejemplo, para diferentes dispositivos basados en XBee, módulos Rabbit, GPS, etc.).
- Estos **Device Driver's** proporcionan las magnitudes que toman del mundo real bajo la forma de **samples** para esos **canales** de la **base de datos de canales.**
- Las abstracciones llamadas ***Presentations*** acceden aquella **base de datos de canales** para generar la comunicación con el usuario o para mover los datos fuera del gateway y lo hacen por polling o por notificación de eventos. Ejemplos de presentaciones pueden ser: command-line interface, web interface, XML-RPC, Modbus, o Web Services.
- Otras abstracciones llamadas ***Loggers*** se pueden usar para volcar el flujo de datos de los canales en archivos dentro del file system.
- Otros objetos del sistema son los **Services** y consisten en tareas no encuadradas en los concepto de *device* o *presentation*. Los Servicios no crean canales ni transmiten información, en cambio realizan tareas como podría ser la implementación de un watchdog que periódicamente evalúe la cantidad de memoria libre y si esta cantidad fuera menor que cierto umbral especificado, entonces reiniciaría el sistema.

La siguiente es una vista conceptual de la arquitectura descripta:

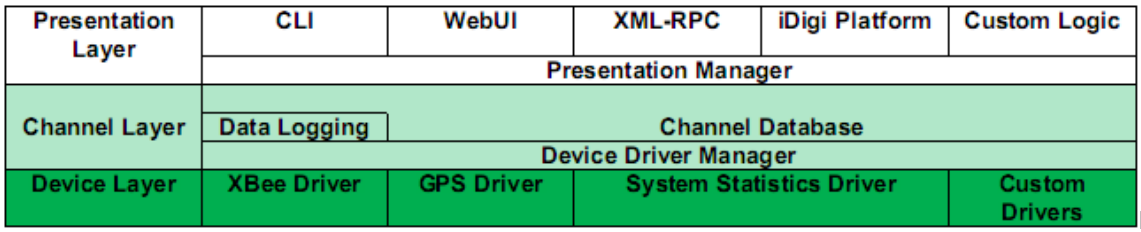


Fig. 5

Por el momento quizá se de utilidad dar una idea gráfica de como se administrará todo esto desde el IDE:

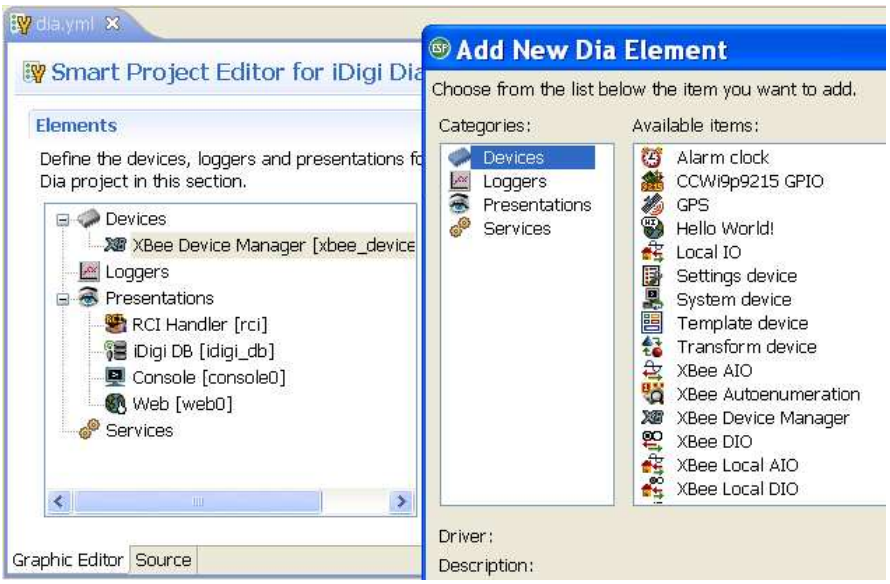


Fig. 6

8.1.1. Descripción adicional de los objetos del framework Dia

8.1.1.1. Canales

Cada **canal** es un elemento discreto y accesible en la **base de datos de canales** y puede ser accedido por las **presentaciones** acorde a un set de permisos. El comportamiento del canal en el sistema es modificado por ciertas opciones que pueden setearse sobre el canal. Todos estos parámetros están establecidos por el driver del dispositivo según lo exija la naturaleza del canal que se esté definiendo.

Permisos: Los permisos pueden ser *get*, *set* o *refresh* indicando las operaciones que las *presentaciones* pueden efectuar sobre el canal en cuestión.

Opciones: Las opciones pueden ser: *auto timestamp***:** en cuyo caso un canal añadirá un timestamp a cualquier objeto *sample* que sea publicado por dicho canal. O *do not log***:** un canal con esta opción no permite que un *logger* obtenga sus *samples*.

8.1.1.2. Samples

Un *sample* representa un valor asociado a un canal en la base de datos de canales. Estos consisten en tres elementos: *value*, *unit*, *timestamp*.

8.1.1.3. Base de Datos de Canales

La base de datos de canales es la abstracción alrededor de la cual está construido el framework. Primero, los devices driver publican sus muestras de datos para los canales en la base de datos, luego, las presentaciones consumen las muestras de esa base de datos de canales. Con esto se intenta proveer una vista natural del movimiento de datos en el sistema.

8.1.2. Archivos del sistema

8.1.2.1 Archivo de configuración *dia.pyr*

Es el archivo que define y configura el sistema que estamos desarrollando (enumera los drivers, presentaciones, etc. y sus configuraciones respectivas). Normalmente será generado a partir del editor gráfico.

8.1.2.2. El archivo *dia.py*

El archivo *dia.py* es el archivo de script principal de la aplicación, se trata del ejecutable que inicia la aplicación *Dia* tomando las referencias presentes en el archivo de configuración *dia.pyr*.

8.1.2.2. El archivo *dia.zip*

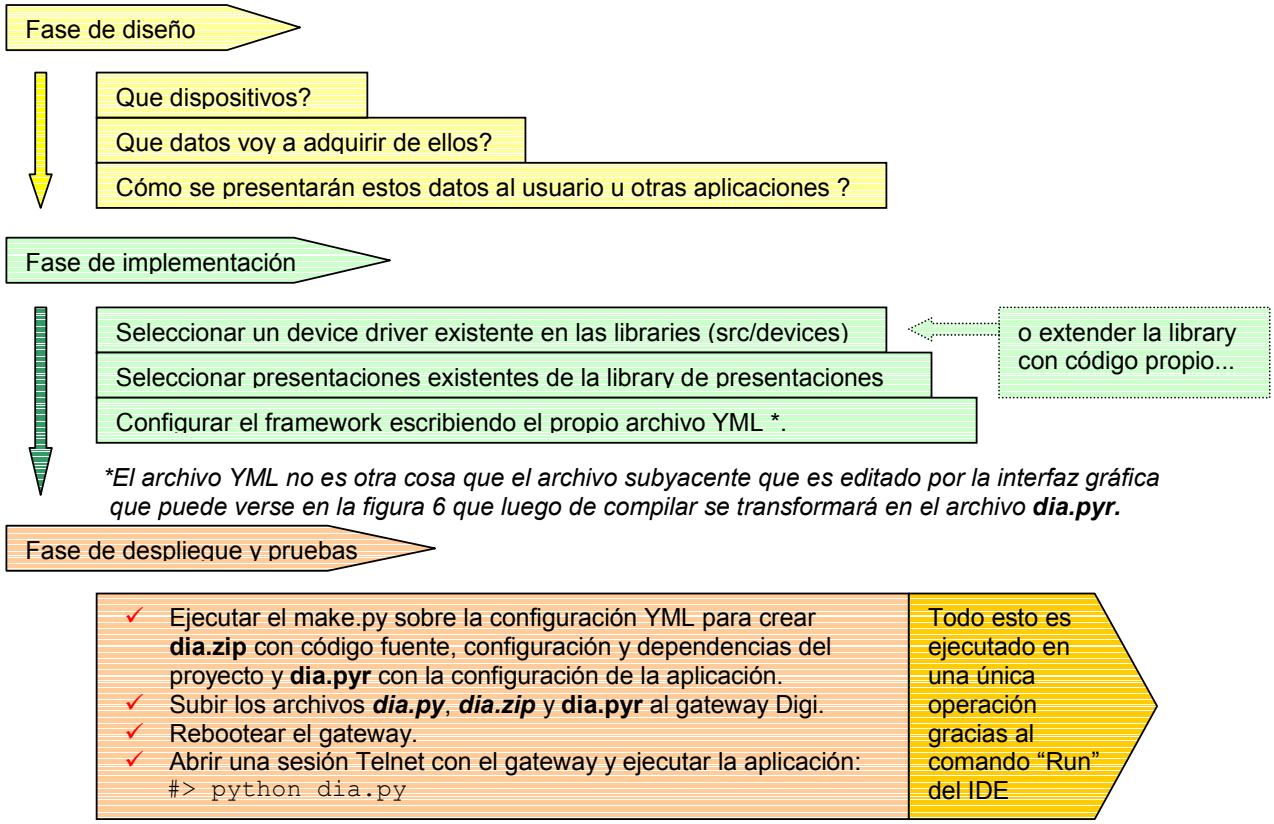
Para que el script corra sobre el gateway, el contenido del directorio *bin* del proyecto debe estar presente el file system del X4. El framework, al compilar el proyecto genera este archivo (dia.zip) con todas las dependencias necesarias para que la aplicación se ejecute en el X4.

8.1.2.2. El archivo *python.zip*

Este es un archivo del sistema que viene incluido de fábrica y contiene elementos básicos de operación del ambiente python como los necesarios para el manejo de threading, garbage collector, etc. Típicamente un file system del X4 conteniendo una aplicación de usuario luce como sigue:

python.zip
dia.py
dia.zip
dia.pyr

9. Ciclo de trabajo



A continuación vamos a tratar de poner en práctica los conceptos de este diagrama para no quedarnos en los papeles.

10. Aplicación de prueba

Finalmente vamos a generar una aplicación para interactuar con el módulo de diversas formas. Haremos el *mapeo* de los procedimientos que describiremos a continuación con el ciclo de trabajo diagramado en [9].

Fase de diseño

Al comenzar estaremos sitiados conforme al ciclo de trabajo en la fase de diseño. Por lo tanto, tenemos que respondernos algunas preguntas básicas:

Que dispositivos?

Asumimos que además del X4, tenemos conectado un módulo ZigBee por algún lado y que este estará asociado al Coordinador presente en el interior del gateway. Para esta demostración contamos con un módulo XBee remoto del tipo *ZB-Router–AT* montado sobre una placa de evaluación de Digi: la *XBib development board* que viene incluida en el kit de desarrollo del X4 y nos permite una rápida puesta en marcha del módulo XBee. Por el otro extremo, estará nuestra computadora de desarrollo, desde la cual consumiremos los datos generados en el XBee y facilitados por el X4.

Que datos voy a adquirir de ellos?

La placa *XBib* no hace mucho más que facilitar la alimentación del módulo XBee, ofrecer un puerto RS232, y exponer algunos I/O digitales. De esta forma tendremos disponibles 4 switches (entradas digitales) y 3 LEDs (salidas digitales), lo cual será más que suficiente para nuestras modestas ambiciones.

Cómo se presentarán estos datos al usuario u otras aplicaciones ?

Vamos a inventarnos la necesidad de presentación de los datos, es decir, de los estados de las líneas de I/O del dispositivo remoto en tres formas diferentes:

1. Datalogging en formato CSV vía TCP.
2. Monitoreo vía web.
3. Adquisición vía XML-RPC

Que, como puede verse obedecen a tres métodos diferentes de operación:

1. Operación cliente-servidor desde la perspectiva del XBee.
2. Visualización y eventual operación manual de las líneas de I/O por parte de un usuario
3. Operación cliente-servidor desde la perspectiva de un programa consumidor de los datos del XBee.

Fase de implementación

Pasamos ahora a la fase de implementación de lo descripto más arriba y para empezar, asumiendo que el IDE está correctamente instalado y en ejecución, vamos a `File->new->iDigi Dia Project` y a continuación completamos con el nombre del proyecto y seleccionamos el intérprete de Python 2.4.3. Acto seguido seleccionamos el dispositivo destino de la aplicación y ya estamos en condiciones de comenzar. Obtendremos una vista del IDE como la que sigue:

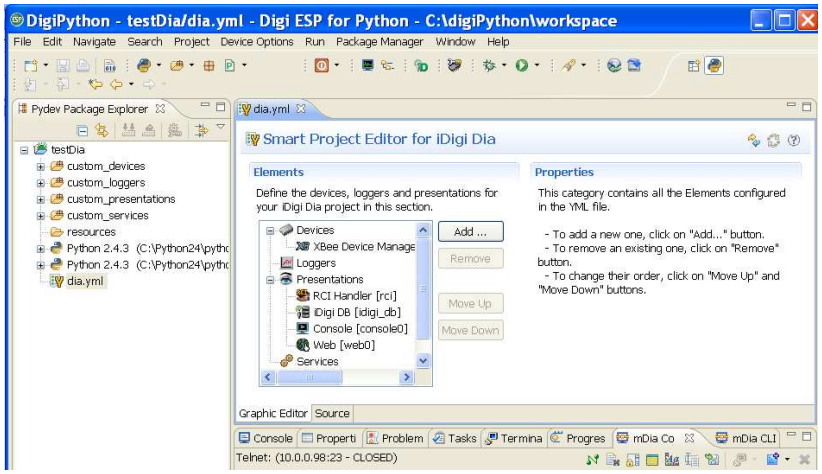


Fig. 7

Allí se puede apreciar el editor gráfico del archivo de configuración del proyecto *Dia (dia.yml)* nótese en el árbol de la ventana central la presencia de los diversos objetos Devices, Loggers, Presentations, Services que hemos descripto brevemente en sección [8.1 Arquitectura].
Podemos ver la lista de *Loggers* y *Services* vacías, la de *Devices* conteniendo un objeto llamado *XBee Device Manager* y la de *Presentations* con varias de ellas presentes. Para comenzar con nuestro proyecto de forma ordenada eliminaremos todas las *presentaciones* de la lista dejándola por lo pronto vacía.
Nuestro ciclo de trabajo nos indica que a continuación debemos seleccionar el *Device Driver* según las especificaciones que definiéramos en la fase de diseño.

Seleccionar un device driver existente en las libraries (src/devices)

10.1. El device driver

Podemos ver en la fig. 8 que en la lista de *Devices* existe en forma predeterminada un dispositivo llamado *XBee Device Manager.* Este es un dispositivo virtual utilizado para realizar las configuraciones de todos los XBee declarados en el archivo YML.

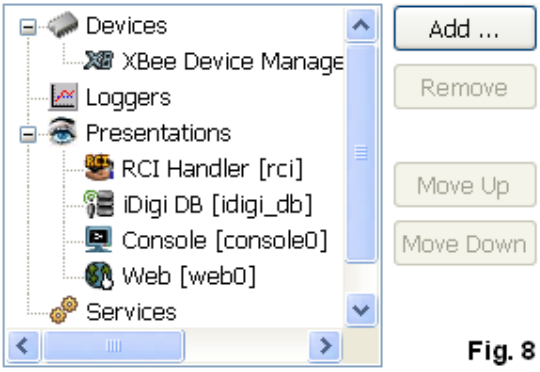


Fig. 8

Todos los dispositivos XBee declarados requieren como uno de sus parámetros de configuración el nombre de un *XBee Device Manager* que esté presente en el proyecto. Y por el momento solo nos quedará presente el *XBee Device Manager* cuyas configuraciones dejaremos como están, salvo que quisiéramos cambiarle el nombre.
Ahora bien, nuestro módulo XBee remoto habíamos dicho que es un *ZB-Router–AT* montado sobre una placa de evaluación de Digi XBib development board por lo tanto, buscaremos y agregamos este driver que ya estará presente entre los dispositivos provistos por el framework. A continuación lo veremos en el árbol de los objetos del proyecto (figura 9).

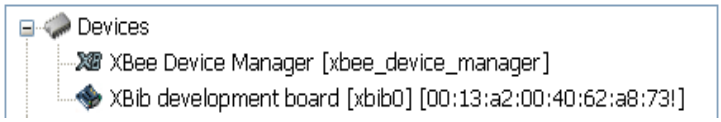


Fig. 9

Si tuviéramos mas dispositivos remotos en nuestro sistema agregaríamos los drivers que fueran necesarios, uno por cada uno de ellos, pues véase que cada driver posee un único número de MAC.

Configurar el framework escribiendo el propio archivo YML

Este punto de la fase de implementación se corresponde con prácticamente cualquier aspecto de edición del archivo ***dia.yml***, es decir, que sería una tarea iterativa a realizar por cada objeto que agreguemos en nuestro diseño. En nuestro caso lo estamos haciendo mediante el editor gráfico, pero podemos alternar indistintamente entre la vista de código y la vista gráfica en cualquier momento.
Entonces, para continuar tenemos que completar las configuraciones de los drivers que agreguemos (y en general de cualquier otro objeto que forme parte del sistema). En nuestro caso tenemos un solo driver que presenta como propiedades, además del propio **nombre**, el nombre de su *Device Manager* y la **dirección** MAC del dispositivo remoto que representa, que al menos en esta demo tenemos que conocer y la podemos extraer por ejemplo, ingresando a la interfaz web del X4 en la lista de dispositivos de la red ZigBee.

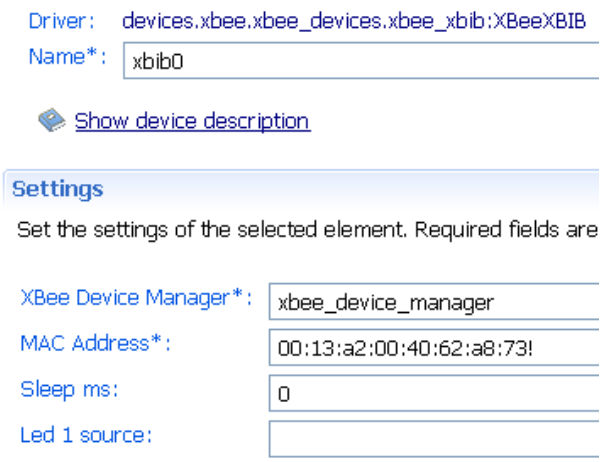


Fig. 10

Si fuera necesario podríamos establecer los parámetros de ahorro de energía (tiempos de sleep y awake) pero en este caso lo dejamos en cero (no dormir) ya que se trata de un router.
El resto de los parámetros lo dejamos en blanco, pero podrían usarse para indicar la fuente de la cual se tomaría el valor para establecer el estado del canal en cuestión.
A continuación (Fig. 11) puede observarse la correspondencia en código de la edición gráfica que acabamos de realizar:

Fig. 11

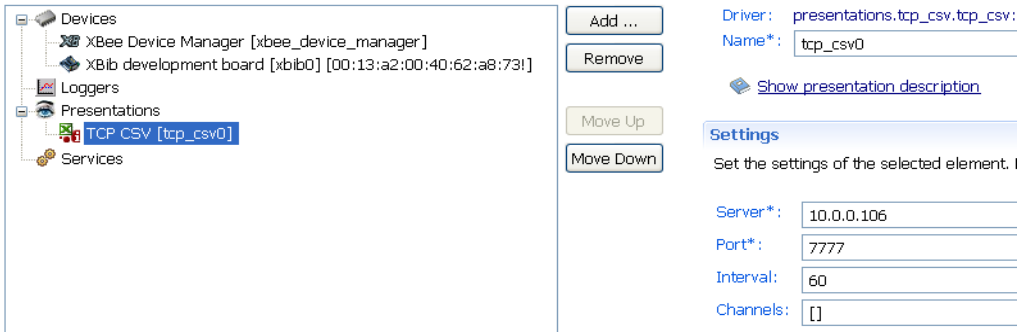
Seleccionar presentaciones existentes de la library de presentaciones

10.2. Presentaciones

Seguimos adelante y ahora nos paramos sobre *Presentations* allí presionando el botón *Add* tendremos la oportunidad de seleccionar dentro de varias opciones nuestros esquemas de presentación de datos según lo hemos definido en la fase de diseño, a saber:

10.2.1. Presentación 1

Elegimos entonces la presentación TCP_CSV. Sobre el árbol del proyecto lo veremos de esta forma:



A su derecha vemos la propiedades de este objeto y debemos completar la dirección IP de un servidor TCP[3] y el puerto donde estará escuchando. En ese servidor vamos a recibiremos según el tiempo indicado, la cadena de caracteres en formato de valores separados por comas conteniendo las muestras provenientes de nuestro módulo remoto. En nuestro caso iniciamos un simple servidor TCP para escuchar todo lo que el X4 tenga para transmitir. Indicamos la dirección IP de la máquina donde esté ejecutándose el servidor y el puerto TCP que es 7777 en este caso.

10.2.2. Presentación 2

Ahora, otras vez posicionados sobre *Presentations* agregamos una presentación del tipo *Web.* Y como puede verse en su página de propiedades, indicamos: nombre de la página y el puerto de acceso HTTP a dicho recurso, si fuera necesario. Con "*Use default server*" en `True`, se utiliza el puerto HTTP por default y se accede a la página de la siguiente forma:

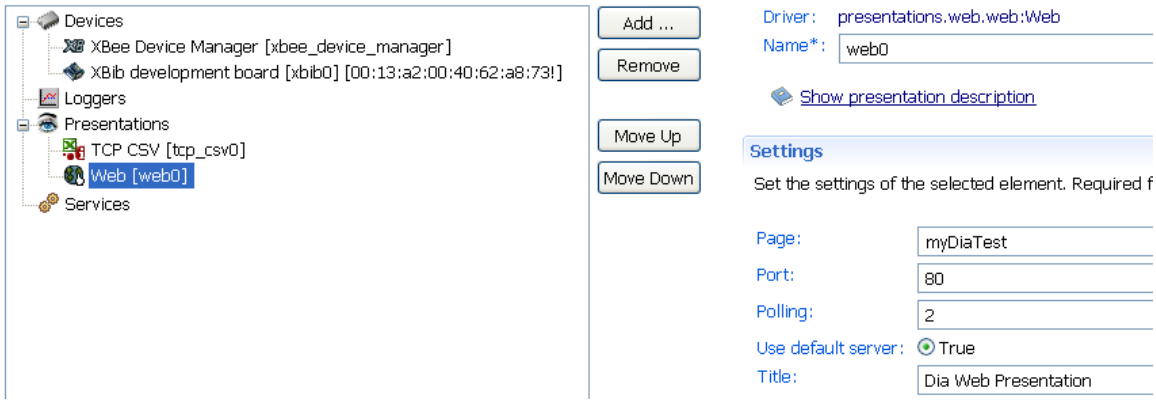
`http://[X4_IP_ADDRESS]/`**`myDiaTest`**

de lo contrario, habrá que indicar el número de puerto en la configuración y luego para acceder:

`http://[X4_IP_Adress]:[Port]/`**`myDiaTest`**.

Es importante destacar que el nombre de la página es *case sensitive.*

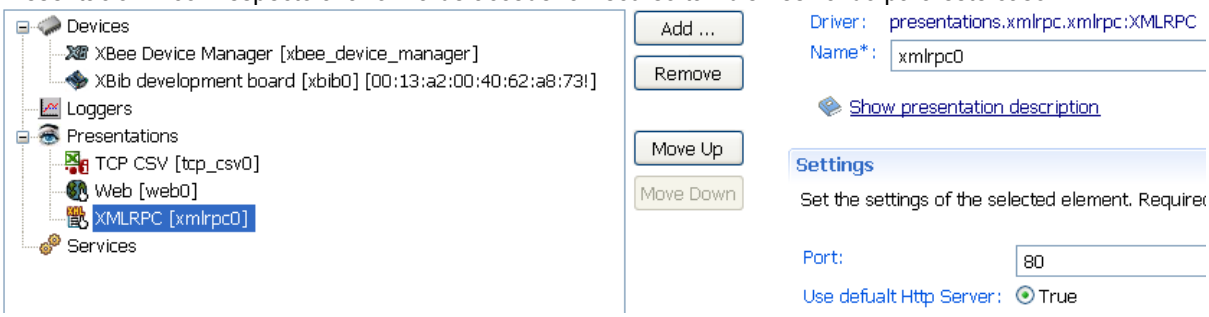
Si así lo deseamos, podemos establecer el intervalo de actualización automática (*Polling*) de los datos presentados en la página en un valor mayor que cero. Esta funcionalidad el *framework* la provee mediante AJAX en forma totalmente transparente al usuario.



[3] Junto con este trabajo se provee del rudimentario servidor TCP que fue utilizado para las pruebas, este ejecutable forma parte de un Kit de Digi y por simplicidad y comodidad lo utilizamos sin modificaciones.

10.2.3. Presentación 3

Por último, también decretamos la necesidad de la presentación XML-RPC[4]. Que ofrece un servicio de transporte y presentación para la interfaz de comandos RCI, de modo que podemos ejecutar acciones de consulta y modificación sobre los canales utilizando XML sobre HTTP. Por el momento adelantamos que XML-RPC es un protocolo que usa XML para codificar las llamadas a procedimientos remotos utilizando HTTP POST como mecanismo de transporte. El servidor XML-RPC se localiza en `http://[X4_IP_ADDRESS]/RPC2`
Solo nos resta indicar el puerto de operación, o poner en `True` "*Use default server*". Lo dicho en la Presentación 2 con respecto a la forma de acceder al recurso también es válido para este caso.



Fase de despliegue y pruebas

10.3. Ejecución y pruebas

En la fase de despliegue y pruebas es cuando ya tenemos la aplicación completa, y solo nos resta probarla y depurarla, de modo que asumiremos que tenemos tanto al Gateway X4 como al módulo XBee remoto en perfecto funcionamiento y ambos en línea.

10.3.1. Ejecución

A continuación compilamos y ejecutamos la aplicación directamente con el comando:

```
Run->Run As -> iDigi Dia Project
```

El entorno de desarrollo dará los siguientes pasos sin necesidad de intervención del usuario. Las siguientes acciones pueden monitorearse desde las distintas consolas del IDE:

1) "Compilación" del proyecto, esto es fundamentalmente para:
   a) Determinar e incluir las dependencias dentro del archivo `dia.zip`
   b) Transformar las configuraciones desde el archivo `dia.yml` generando el archivo de configuración de la aplicación para el *runtime*: `dia.pyr`.
2) Apertura de una sesión de línea de comandos (CLI) en un puerto TCP[5] predefinido para:
   a) Envía los archivos generados al file system del X4.
   b) Resetea el X4.
3) Una vez en línea nuevamente, apertura de una sesión Telnet y ejecución del programa.

Seguidamente a estos pasos obtendremos en la terminal Telnet los mensajes que reportan las diferentes acciones que está ejecutando nuestro programa:

[4] Para obtener información sobre el estándar consultar http://www.xmlrpc.com/.
[5] Puede verse esta configuración en `Project->Properties->Run/Rebug Settings`, y `edit` luego de seleccionar el nombre del proyecto.

```
#> python dpdsrv.py

XBeeDeviceManager(xbee_device_manager): retrieving node list
XBeeDeviceManager(xbee_device_manager): node '00:13:a2:00:40:62:a8:73!' moved to CONFIGURE state.
XBeeDeviceManagerConfigurator: worker assigned to '00:13:a2:00:40:62:a8:73!'
Starting Presentation Manager...
XBeeConfigBlockDDO.apply_config: trying 'SM' = '0x0' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'P2' = '0x0' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'PR' = '0x1fff' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'DL' = '\x40\x3b\xcd\xdd' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'P1' = '0x0' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'DH' = '\x00\x13\xa2\x00' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'IR' = '0x0' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'IC' = '0xf' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'D4' = '0x0' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'D2' = '0x3' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'D3' = '0x3' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'D0' = '0x3' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockDDO.apply_config: trying SET 'D1' = '0x3' to '[00:13:a2:00:40:62:a8:73]!'
XBeeConfigBlockFinalWrite: trying 'WR' to '[00:13:a2:00:40:62:a8:73]!'
XBeeDeviceManager(xbee_device_manager): configuration done for node '00:13:a2:00:40:62:a8:73!' promoting to RUNNING state.
XBeeXBIB(xbib0): running indication
XBeeXBIB(xbib0): sample indication
Web(web0): using web page myDiaTest and using digiweb
Starting Services Manager...
Core services started.
```

10.3.2. Pruebas

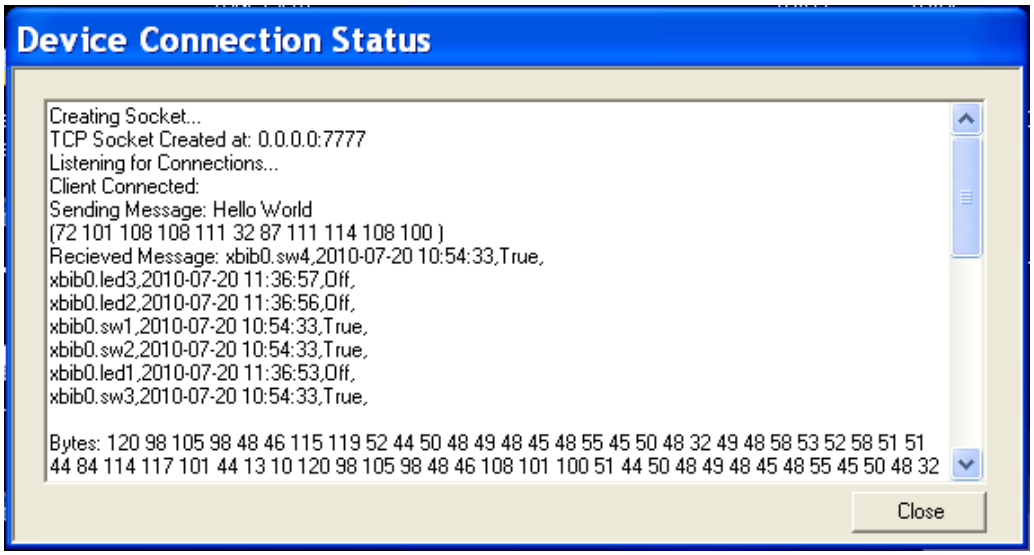
En estas instancias comprobaremos el funcionamiento del sistema desde la perspectiva de las presentaciones pautadas en el inicio del ciclo de trabajo.
Para esto tenemos que utilizar, de ser necesario, algunas herramientas auxiliares ajenas al framework.
A saber: el rudimentario servidor TCP [nota al pie nº 3] para comprobar el funcionamiento de la presentación 1, un browser para comprobar la presentación 2, y el envío de un script con el requerimiento XML-RPC para la presentación 3.

10.3.2.1 Prueba 1

Debemos ejecutar el programita *TCPServer.exe* de lo contrario habremos notado quizá el siguiente mensaje de error en la consola Telnet al ejecutar:

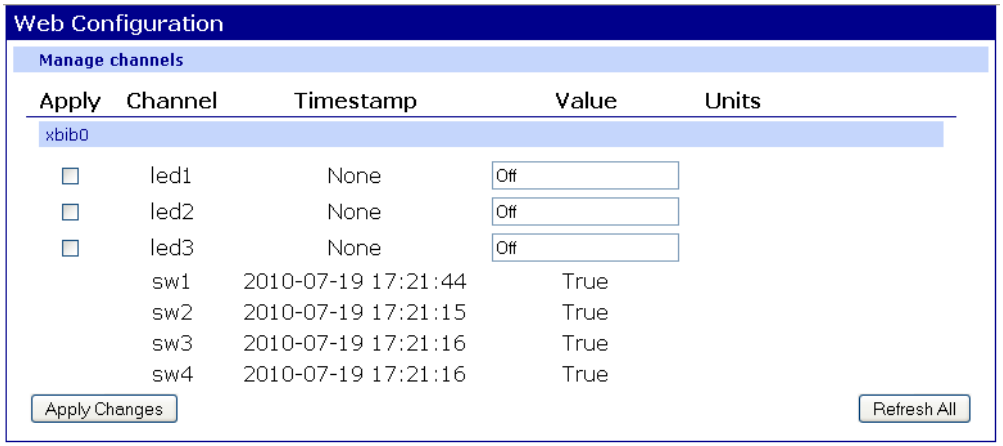
```
TCPCSV(tcp_csv0): error connecting to 10.0.0.106:7777: (111, 'Connection refused')
```

Si ejecutamos entonces el programita servidor y tenemos la paciencia de esperar como máximo 60 segundos (así lo configuramos en la *presentación 1*) veremos como llegan los datos esperados de nuestro equipo remoto:



10.3.2.2 Prueba 2

Para la prueba 2, es decir la prueba de la segunda presentación que hemos definido, necesitamos sencillamente un browser, lo apuntamos al recurso previamente definido (http://10.0.0.98/myDiaTest) y como podemos ver a continuación, obtenemos la pantalla de control y monitoreo que buscábamos:



10.3.2.3 Prueba 3

Para finalizar con las pruebas nos resta comprobar el funcionamiento de la interfaz XML-RPC, para esto, necesitamos enviar al servidor RPC un requerimiento HTTP-POST con el siguiente contenido:

```
POST /RPC2 HTTP/1.0
Host: 10.0.0.106
Content-Type: text/xml
Content-length: 150

<?xml version="1.0"?>
      <methodCall>
            <methodName>channel_dump</methodName>
            <params>
                  <param>
                        <value></value>
                  </param>
            </params>
      </methodCall>[ENTER]
[ENTER]
```

Cómo se envía semejante cosa? muy simple: abrimos una conexión TCP contra la dirección IP del X4 en el puerto 80, por ejemplo, con un `Telnet <IP_ADDRESS_X4> 80` y acto seguido copiamos en la consola el código de arriba. Como respuesta obtendremos inmediatamente un formato similar con la lista de canales y sus correspondientes valores actuales como se ejemplifica a continuación:

```
<?xml version='1.0'?>
<methodResponse>
<params>
  <param>
    <value>
        <struct>
          <member>
            <name>xbib0.sw4</name>
              <value>
                <struct>
                 <member>
                   <name>timestamp</name>
                   <value>
                     <double>1279623273.0</double>
                   </value>
                 </member>
                 <member>
                   <name>value</name>
                   <value>
                     <boolean>1</boolean>
                   </value>
                 </member>
                 <member>
                   <name>unit</name>
                   <value>
                     <string></string>
                   </value>
                 </member>
                </struct>
              </value>
          </member>
           ...
           ...
           [y así sucesivamente hasta finalizar con toda la lista de canales]
           ...
           ...
  </param>
</params>
</methodResponse>
```

Nótese que en el requerimiento POST se remarcaron las siguientes líneas:

a) **`POST /RPC2 HTTP/1.0`**
Se destaca /RPC2 que es el recurso solicitado, el servidor web embebido en el X4 interpreta un llamado XML-RCP cuando le solicitan este recurso.

b) **`Content-length: 150`**
Se destacó por la importancia de que este número debe indicar con exactitud el número de caracteres que siguen luego del espacio en blanco hasta el último caracter inclusive, se deben contabilizar espacios en blanco, `<CR>` y `<LF>`, etc. de lo contrario no funcionará, es decir no obtendremos una respuesta del servidor.

c) **`<methodName>channel_dump</methodName>`**
El encerrado entre las etiquetas **`<methodName>`** es el método remoto invocado, y puede ser alguno de los siguientes[6]:

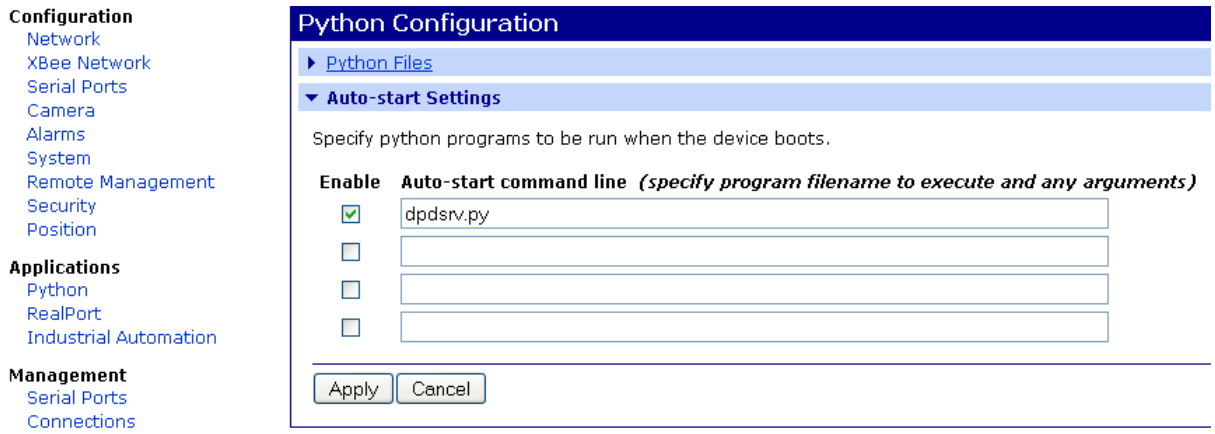
```
device_instance_list()
channel_list()
channel_get(channel_name)
channel_set(channel_name, timestamp, value, unit, autotimestamp = True)
channel_info(channel_name)
channel_refresh(channel_name)
```

[6] Para mayor información consultar la ayuda en línea del entorno de desarrollo:
`iDigi Dia Documentation->iDigi Dia 1.3.8->User Documentation->Presentations`

```
channel_dump()
logger_list()
logger_set(logger_name)
logger_dump()
logger_next()
logger_prev()
logger_rewind()
logger_seek(self, offset, whence='set', record_number=0)
```

10.4 Puesta en marcha

Una vez que todo resultó según lo planeado, probablemente necesitemos que nuestro proyecto quede corriendo en el X4 todo el tiempo, para esto tenemos que indicar en la configuración de este último que tal programa se auto-ejecute al inicio.
Esto es, accediendo con el browser a la interfaz de configuración del X4 en `Python->Auto-start Settings`. Allí se ingresa el nombre del programa (seguido por los parámetros que fueran necesarios, si existieran). Luego se habilita el auto-inicio del programa chequeando la casilla de verificación, se confirman los cambios con `"Apply"` y eso es todo.



11. Conclusiones

El *ConnectPort X4* junto con su plataforma de desarrollo están concebidos para integrar redes ZigBee con redes TCP/IP creando soluciones implementadas en Python sin necesidad de conocer en profundidad ni los detalles del hardware subyacente ni los detalles del lenguaje o de la API de programación.

Conociendo los parámetros básicos de operación de la red ZigBee y de la red IP, y siguiendo un esquema bien definido para el Ciclo de Trabajo **[9]**, es posible construir sistemas totalmente funcionales partiendo de la capa de red, y llegando hasta una capa de alto nivel, como lo es la de presentación de los datos. Esto permite acceder inmediatamente a interfaces de monitoreo y control y a la articulación directa y estandarizada con otros subsistemas para el procesamiento directo de datos obtenidos de la red ZigBee sin escribir ni una sola línea de código.

Si el usuario puede encuadrar y dirigir sus necesidades dentro del ciclo de trabajo presentado, entonces este producto muy probablemente sea el apropiado para sus necesidades. El framework incluye un listado de drivers que proporcionan abstracciones para diversos dispositivos ZigBee (y otros) de Digi bien definidos[7]. Conforme a esto último, si el hardware a integrar se corresponde, está presente, o puede adaptarse a los drivers preexistentes entonces el ciclo de trabajo será extremadamente rápido y "limpio". Por el contrario, si ninguno de los drivers se adapta exactamente a las necesidades del hardware subyacente, y/o el escenario de integración no se ajusta al ciclo de trabajo propuesto, entonces el usuario

[7] Como por ejemplo: *XBee WatchPort, XBee WallRouter, XBee Adapters* (*IO, AIO, Serial*), *XBib development board* (usada en las demos) y otros como GPS, Rabbit, etc.

se verá obligado a diseñar y desarrollar sus propios drivers, presentaciones y quizá otros objetos, para lo cual deberá familiarizarse mucho más con el lenguaje y con las particularidades del framework, en consecuencia, la ventaja de la extrema de rapidez y simplicidad de integración pueden comenzar a diluirse.

Otro enfoque a tener en cuenta es el de la eficiencia, ya que como se ha mencionado en **[7]** el lenguaje de implementación es interpretado, lo cual implica siempre la existencia de un intermediario (como mínimo). En este caso, el *intérprete de Python*, genera el *bytecode* para la *máquina virtual*, que a su vez ejecuta el código *de máquina* correspondiente para cada bytecode.
En aplicaciones donde la eficiencia sea crítica, sin dudas el usuario preferirá conectar un módulo *ZigBee Coordinador* al puerto serie de la plataforma de procesamiento de su preferencia, interpretar las tramas API de los módulo XBee y construir desde allí hacia arriba. Claro que en este caso, la contrapartida está en prescindir de las bondades de la generación de código de alto nivel bajo la forma de presentaciones listas para usar.

Apéndice

Knows bugs en iDigi Dia 1.3.8

Al agregar una prentación del tipo TCPCSV el código generado automáticamente por el editor gráfico será similar el siguiente:

```
presentations:
  - name: tcp_csv0
    driver: presentations.tcp_csv.tcp_csv:TCPCSV
    settings:
        server: "10.0.0.106"
        port: 7777
```

Luego al compilar y ejecutar el proyecto existirán errores porque la referencia al driver (`driver:`) es incorrecta, debiendo ser en realidad: **`presentations.tcpcsv.tcpcsv:TCPCSV`** es decir, sin guiones bajos.