Capítulo 11
IMPLEMENTACIÓN

11.1 INTRODUCCIÓN

En los capítulos anteriores hemos alcanzado los siguientes objetivos:

- Estudio teórico del protocolo de comunicación X-10.
- Estudio teórico de la red UPnP y la interactuación de los dispositivos en ella.

- **OBJETIVO FINAL**

A continuación veremos cómo hemos puesto en práctica toda esta información para que nuestro sistema X-10 se muestre como un dispositivo UPnP en la red, describiendo sus funciones y capacidades e interactuando con cualquier otro dispositivo.

En otras palabras, estudiaremos paso por paso el proceso de diseño, desde el entorno inicial de trabajo hasta la implementación final del programa.

11.2 ENTORNO DE DESARROLLO

11.2.1 .NET

Microsoft.NET es el conjunto de nuevas tecnologías en las que Microsoft ha estado trabajando durante los últimos años con el objetivo de obtener una plataforma sencilla y potente para distribuir el software en forma de servicios que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de

componentes con los que hayan sido desarrollados. Ésta es la llamada **plataforma .NET**, y a los servicios antes comentados se les denomina **servicios Web**.

.NET podría considerarse una respuesta de Microsoft al creciente mercado de los negocios en entornos Web, como competencia a la plataforma Java de Sun Microsystems.

A largo plazo Microsoft pretende reemplazar el API Win32 o Windows API con la plataforma .NET. Esto debido a que el API Win32 o Windows API fue desarrollada sobre la marcha, careciendo de documentación detallada, uniformidad y cohesión entre sus distintos componentes, provocando múltiples problemas en el desarrollo de aplicaciones para el sistema operativo Windows. La plataforma .NET pretende solventar la mayoría de estos problemas proveyendo un conjunto único y expandible con facilidad, de bloques interconectados, diseñados de forma uniforme y bien documentados, que permitan a los desarrolladores tener a mano todo lo que necesitan para producir aplicaciones sólidas.

Con esta plataforma Microsoft incursiona de lleno en el campo de los Servicios Web y establece el XML como norma en el transporte de información en sus productos y lo promociona como tal en los sistemas desarrollados utilizando sus herramientas.

.NET intenta ofrecer una manera rápida y económica pero a la vez segura y robusta de desarrollar aplicaciones -o como la misma plataforma las denomina, soluciones- permitiendo a su vez una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

11.2.2 .NET FRAMEWORK

El "framework" o marco de trabajo, constituye la base de la plataforma .NET y denota la infraestructura sobre la cual se reúnen un conjunto de lenguajes, herramientas y servicios que simplifican el desarrollo de aplicaciones en entorno de ejecución distribuido.

Bajo el nombre .NET Framework o Marco de trabajo .NET se encuentran reunidas una serie de normas impulsadas por varias compañías además de Microsoft (como Hewlett-Packard , Intel, IBM, Fujitsu Software, Plum Hall, la Universidad de Monash e ISE).

Microsoft publicó el denominado kit de desarrollo de software conocido como .NET Framework SDK para crear aplicaciones para la plataforma .NET, tanto servicios Web como aplicaciones tradicionales (aplicaciones de consola, aplicaciones de ventanas, servicios de Windows NT, etc.), que incluye las herramientas necesarias tanto para su desarrollo como para su distribución y ejecución. Asimismo, Visual Studio.NET, que permite hacer todo la anterior desde una interfaz visual basada en ventanas. Ambas herramientas pueden descargarse gratuitamente desde *http://www.msdn.microsoft.com/net*, aunque la última sólo está disponible para subscriptores MSDN Universal (los no subscriptores pueden pedirlo desde dicha dirección y se les enviará gratis por correo ordinario).

Los principales componentes del marco de trabajo son:

- El conjunto de lenguajes de programación
- La Biblioteca de Clases Base o BCL
- El Entorno Común de Ejecución para Lenguajes o CLR por sus siglas en inglés.

11.2.3 LENGUAJE C#

C# (leído en inglés "C Sharp" y en español "C Almohadilla") es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el **lenguaje nativo de .NET**

La sintaxis y estructuración de C# es muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic.

Un lenguaje que hubiese sido ideal utilizar para estos menesteres es Java, pero debido a problemas con la empresa creadora del mismo -

Sun-, Microsoft ha tenido que desarrollar un nuevo lenguaje que añadiese a las ya probadas virtudes de Java las modificaciones que Microsoft tenía pensado añadirle para mejorarlo aún más y hacerlo un lenguaje orientado al desarrollo de componentes.

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. El hecho de ser relativamente reciente no implica que sea inmaduro, pues Microsoft ha escrito la mayor parte de la BCL usándolo, por lo que su compilador es el más depurado y optimizado de los incluidos en el *.NET Framework SDK.*

Por todo lo anterior, y con el factor añadido de que las herramientas de Intel proporcionan el código en C#, dicho lenguaje ha sido elegido para la programación de nuestro dispositivo.

11.3 ACCESO AL PUERTO SERIE

La comunicación con el CM11 se hace a través del puerto serie, por tanto es de vital importancia programar eficientemente tanto la transmisión como la recepción por dicho puerto.

En las comunicaciones seriales usando la norma RS232 se debe tener en cuenta de que la recepción de un dato ocurre de manera asíncrona, es decir, puede ocurrir en cualquier momento. El hardware de la computadora se encarga de interrumpir cualquier proceso que esté en ejecución cuando un nuevo dato llega al puerto. Es posible que el sistema operativo no pueda atender al nuevo dato de entrada, por lo que lo almacena en un buffer que contiene la secuencia de datos que van llegando. En el anexo final podemos encontrar una información más detallada de todas las características y funcionalidades del puerto serie.

Windows se encarga de la gestión de los puertos y nosotros podemos acceder a ellos de varias formas. Inicialmente el código fue desarrollado en C++ en 'Borland C++ Builder 6', usando el API de Windows para comunicarnos con el puerto serie. Más tarde, al trabajar con UPnP y hacernos eco de la existencia de las herramientas de Intel, que proporcionan el código en C#, se decidió optar por programar todo en este nuevo lenguaje. De este modo el entorno de desarrollo elegido fue Microsoft Visual Studio 2005, que viene dotado de un namespace específico dedicado al puerto serie.

Veamos a continuación cómo se accede al puerto serie desde estas dos posibilidades:

11.3.1 PROGRAMACIÓN MEDIANTE LAS FUNCIONES DE LA API DE WINDOWS

La Interfaz de Programación de Aplicaciones, cuyo acrónimo en inglés es API (Application Programming Interface).

Se trata del conjunto de llamadas al sistema que ofrecen acceso a los servicios del sistema desde los procesos y representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software. De esta forma, los programadores se benefician de las ventajas de la API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio.

Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, permitiendo que una aplicación corra bajo el sistema operativo Windows.

Los puertos serie, por tratarse de dispositivos incluidos como parte de los PC desde sus comienzos, están muy bien integrados en el API de Windows, por lo tanto, tenemos un amplio repertorio de funciones para manejarlos.

11.3.2 PROGRAMACIÓN MEDIANTE EL CONTROL DE COMUNICACIONES DE MICROSOFT VISUAL STUDIO

Aunque el puerto serie ha desaparecido en la mayoría de los ordenadores actuales, muchos todavía lo conservan, y gracias al adaptador serie-USB cualquier ordenador puede tener uno.

Así que la integración del puerto serie en .NET Framework 2.0 fue un deber para revivir este puerto y hacer posible su funcionamiento con todas las tecnologías del Framework.

Hemos de decir que en Visual Studio también sería posible trabajar con el puerto serie usando la API de Windows, ya que la API de Windows contiene miles de funciones, estructuras y constantes que podemos utilizar en cualquier proyecto. Sin embargo, esas funciones están escritas en lenguaje C, y por tanto deben ser declaradas antes de poder usarlas.

Por tanto, la implementación de una API en C# es un trabajo arduo y tedioso, ya que antes de poder implementarla debemos saber cómo se implementan las estructuras en c#, la conversión de tipos, safe/unsafe code, managed/unmanaged code y muchos más. Por ello

usaremos el control de comunicaciones específico de Visual Studio para el puerto serie.

System.IO.Ports namespace

Los programadores del Framework 2.0 desarrollaron un namespace exclusivamente destinado al control del puerto serie, el namespace System.IO.Ports, dentro del cual se encuentra la clase serialport, que permite dirigir y controlar un puerto serie. Esta clase posee todas las propiedades, métodos y eventos permitiendo configurar y organizar la transferencia de datos por el puerto serie.

De este modo se facilita enormemente su implementación. Lo que es más, su manejo es similar a la lectura o escritura en un archivo, de esta manera se aumenta el periodo de vida de este puerto.

11.4 HERRAMIENTAS INTEL. DISEÑO DEL DISPOSITIVO UPNP

El diseño de nuestro dispositivo UPnP consiste en 3 pasos básicos. El primero consiste en crear el archivo de descripción del servicio. Este archivo de descripción contiene toda la información del servicio, incluyendo las acciones y las variables de estado disponibles. Usando esta descripción, seremos capaces de crear el UPnP stack para el dispositivo, que supone el paso dos. El diseño del stack conforma una de las partes más tediosas. Consiste en registrar el sevicio, anunciar el servicio y procesar las suscripciones y las peticiones del servicio. La parte final consiste en implementar las acciones del servicio.

- **DESCRIPCIÓN DEL SERVICIO**

En el directorio de herramientas de Intel (Intel Tools for UPnP Technology), seleccionamos **'Service Author'**. Esta aplicación genera la descripción XML de los servicios que nos gustaría ofrecer.

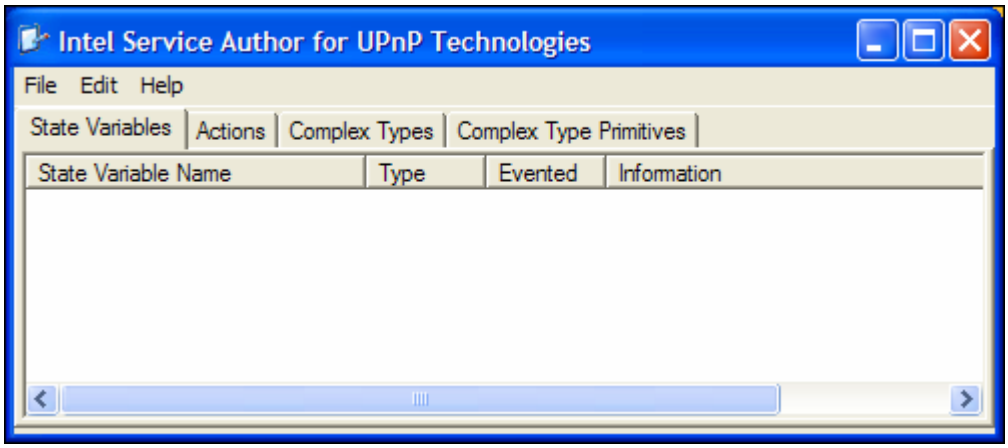

Figura 58: Service Author

En nuestro caso, crearemos 3 variables de estado y una acción para el dispositivo. De este modo, haciendo click con el botón derecho en el área de las variables de estado, seleccionamos 'Add new state variable'.

Añadiremos 3 variables de estado:

- **HouseCode**, tipo string. Evented ON. Valor por defecto 'A'.
- **UnitCode**, tipo byte. Evented ON. Dependiendo de la versión de 'Service Author' aparecerán distintos tipos de datos a seleccionar. Si no aparece la opción byte, seleccionaremos int, ya que el código de dispositivo será siempre un número de 1-16. Le daremos por defecto el valor '1'.
- **Order**, tipo string. Evented ON. Valor por defecto 'ON'.
- **Macro,** tipo string. Evented OFF

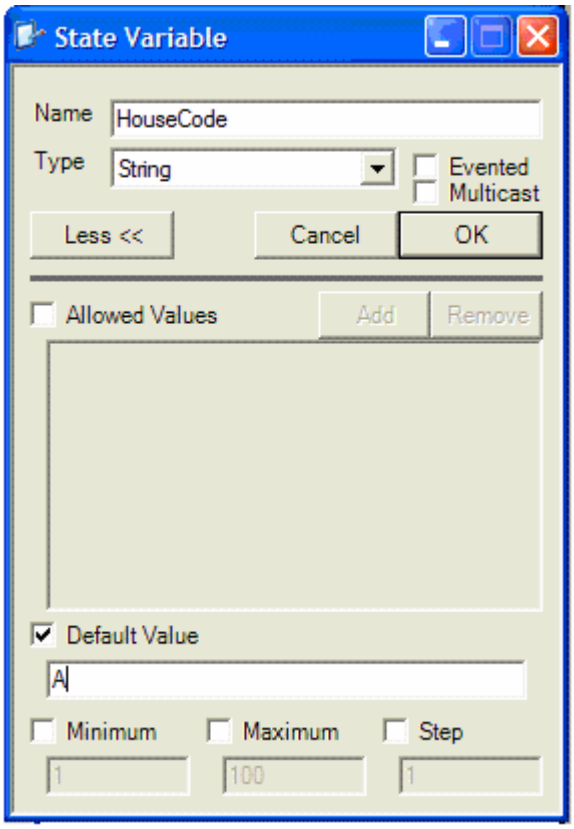

Figura 59: Selección de las variables de estado

A continuación diseñamos las funciones que va a ofrecer el dispositivo:

- **Command.** Para el envío de los comandos a la red eléctrica.

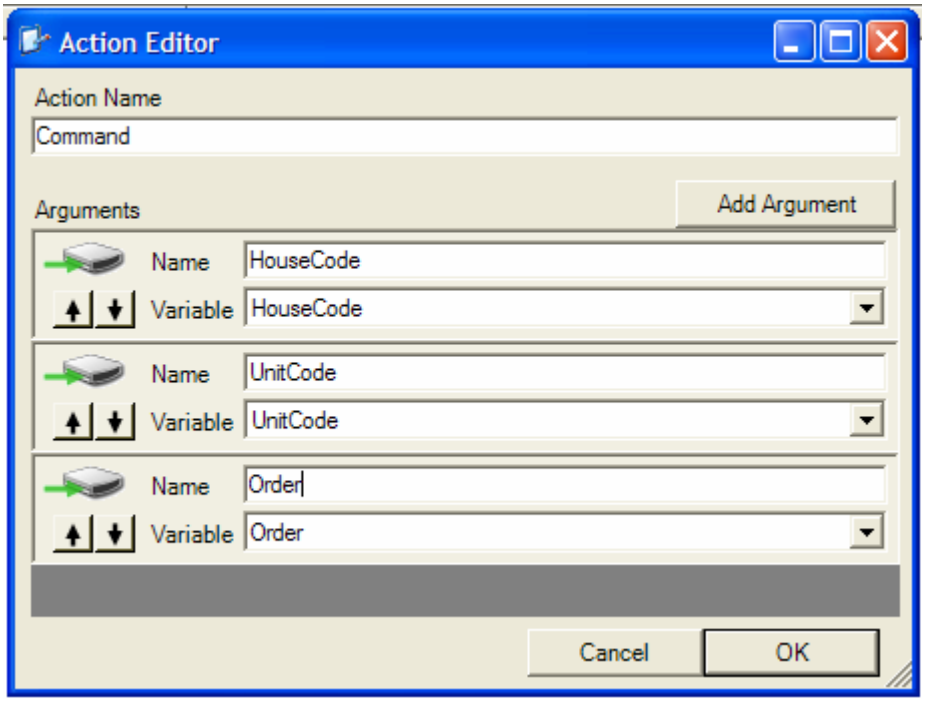

Figura 60: Selección de las acciones

Ahora hemos de cambiar la dirección de los argumentos, ya que todo son de salida. Esto se hace situando el ratón en la flecha verde, y seleccionando OUTPUT:

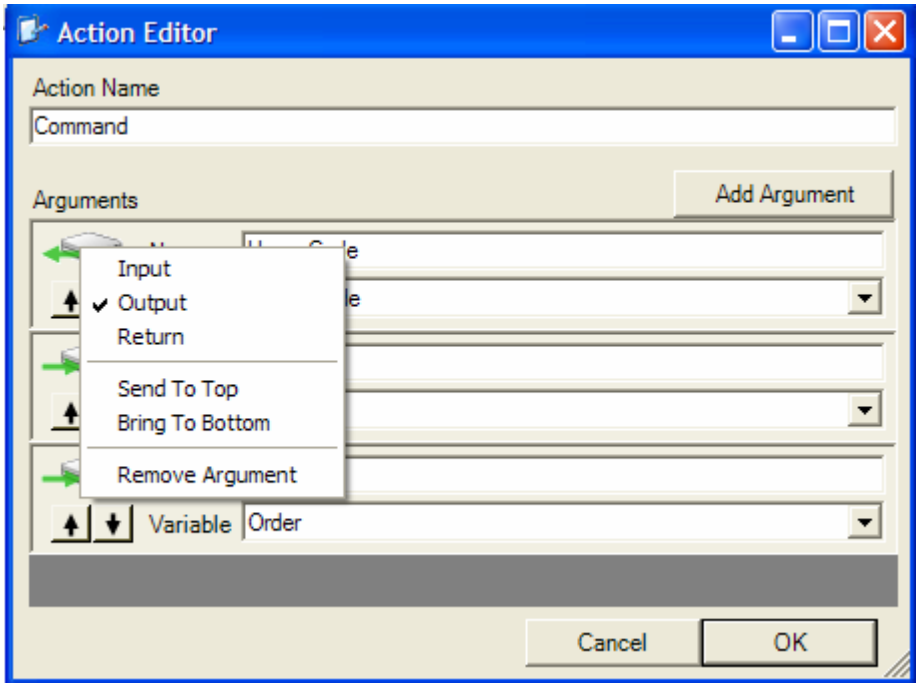

Figura 61: Dirección de los argumentos

- **Macro.** Enviamos una macro programada para enviarla a la Eeprom del CM11.

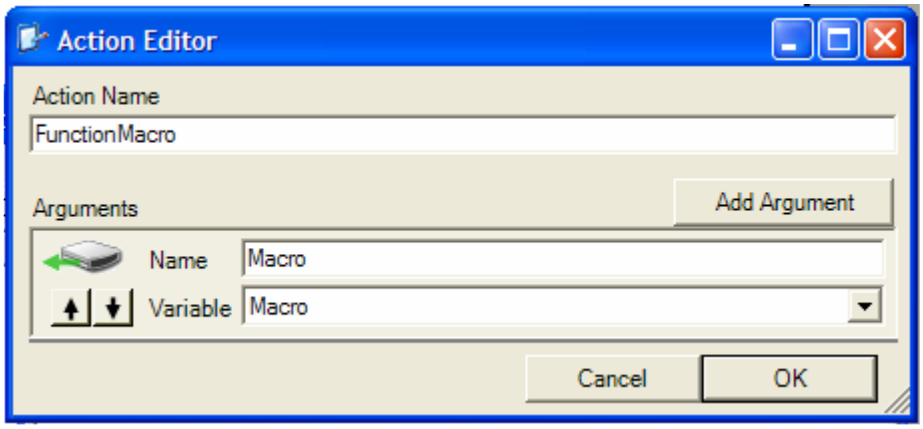

Figura 62: Función macro

De este modo, tenemos ya creada la descripción de nuestro dispositivo:

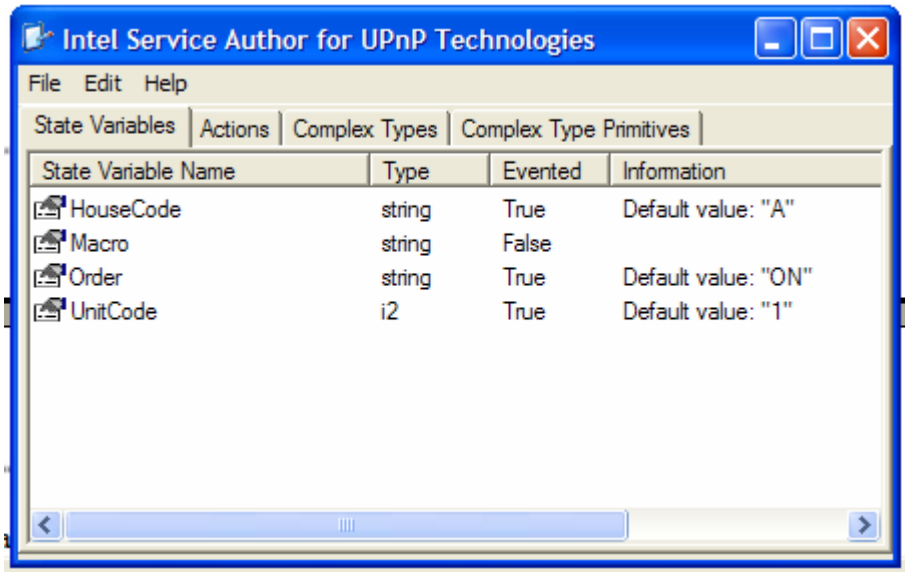

Figura 63: Argumentos del dispositivo

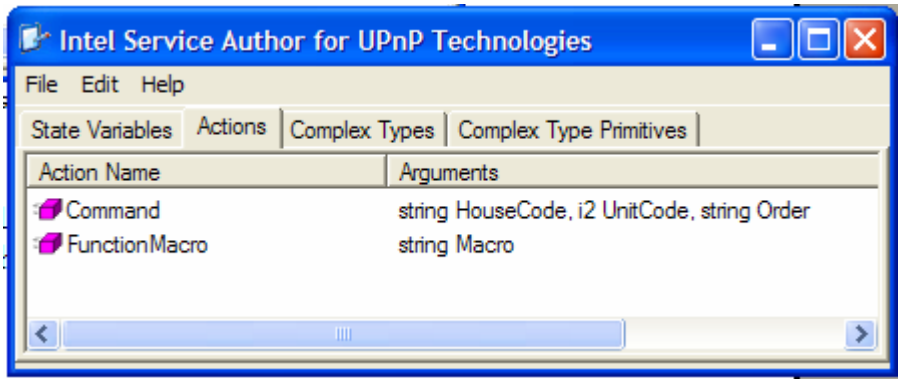

Figura 64: Funciones del dispositivo

Guardamos el archivo como 'CM11Service.xml' en el directorio que deseemos. Si ahora abrimos el archivo observamos las acciones y las variables de estado creadas:

```
<?xml version="1.0" encoding="utf-8" ?>
- <scpd xmlns="urn:schemas-upnp-org:service-1-0">
  - <specVersion>
      <major>1</major>
      <minor>0</minor>
    </specVersion>
  - <actionList>
    - <action>
        <name>Command</name>
      - <argumentList>
        - <argument>
            <name>HouseCode</name>
            <direction>out</direction>
            <relatedStateVariable>HouseCode</relatedStateVariable>
          </argument>
        - <argument>
            <name>UnitCode</name>
            <direction>out</direction>
            <relatedStateVariable>UnitCode</relatedStateVariable>
          </argument>
        - <argument>
            <name>Order</name>
            <direction>out</direction>
            <relatedStateVariable>Order</relatedStateVariable>
          </argument>
        </argumentList>
      </action>
    </actionList>
  - <serviceStateTable>
    - <stateVariable sendEvents="yes">
        <name>HouseCode</name>
        <dataType>string</dataType>
        <defaultValue>A</defaultValue>
      </stateVariable>
    - <stateVariable sendEvents="yes">
```

Figura 65: Fragmento del archivo XML

- **CREACIÓN DEL DEVICE STACK**

En el directorio 'Intel Digital Home Device Code Wizard' seleccionamos **'Device Builder'**. Escribir la descripción del servicio a mano sería un trabajo tedioso, pero gracias al 'Device Builder' generamos el UPnP device stack code a partir del archivo XML anterior.

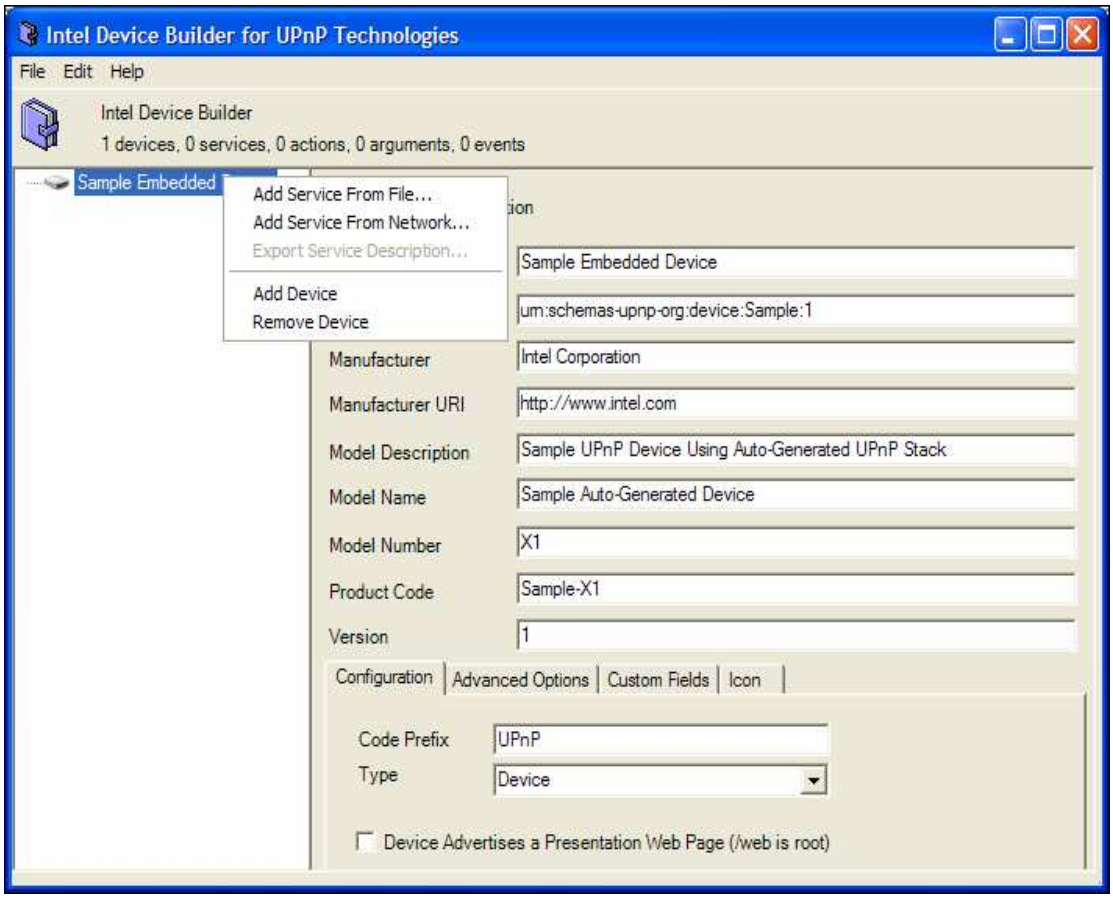

Figura 66: Intel Device Builder

Haciendo click sobre 'Add Service From File' seleccionamos el archivo 'tutorial device.xml'. Cambiamos ahora algunos parámetros:

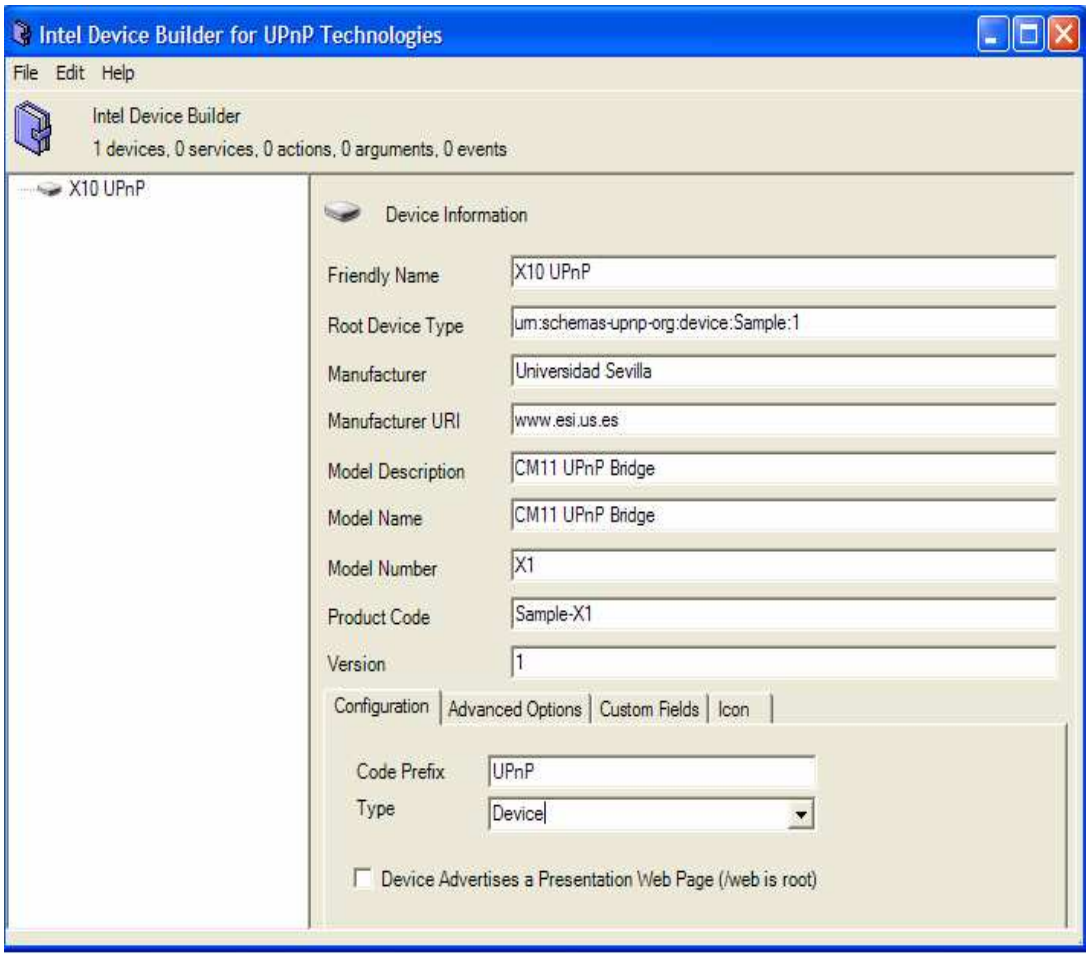

Figura 67: Parámetros del dispositivo

El tipo de servicio es un parámetro muy importante ya que es el parámetro mediante el que el punto de control UPnP buscará al dispositivo.

Guardamos el archivo como 'CM11Service.upnpsg'. La extensión 'upnpsg' significa 'Stack Generation File'.

Para el último paso, necesitamos generar el device stack. En el menú Archivo, seleccionamos 'Export Device Stack'.

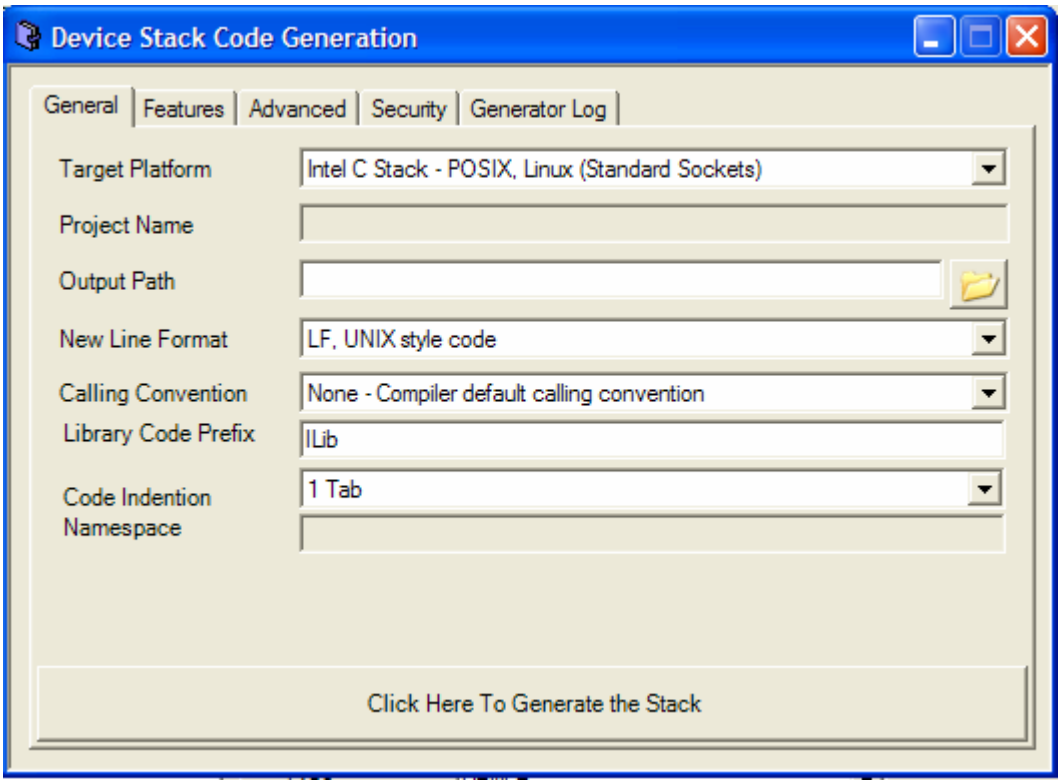

Figura 68: Generación de device code stack

Y en la ventana de 'Device Stack Code Generation' seleccionamos las siguientes opciones:

- En 'Target Platform' seleccionamos 'Intel.NET Framework Stack (C#)'.
- 'Output Path' debe ser nuestro directorio de salida.

El resto de los parámetros pueden quedarse igual. Haciendo click para generar el stack, en nuestro directorio debe aparecer un nuevo proyecto en C#, nuestro dispositivo UPnP. Podemos entonces cerrar Device Builder, la tarea ha finalizado.

Efectivamente, vemos que en el directorio seleccionado hay varios archivos en C#, que podemos abrir con nuestro programa Microsoft Visual Studio.

Ahora deberemos unir este código, que nos ofrece un dispositivo UPnP con nuestro código correspondiente a la comunicación con el CM11 por el puerto serie. De este modo podremos interactuar en una red UPnP como cualquier otro dispositivo.

11.5 PROBLEMAS EN EL DISEÑO

A continuación hacemos referencia de forma resumida a algunos problemas encontrados durante el proceso de diseño de la aplicación, tanto en la parte asociada a X-10 como a UPnP.

11.5.1 PROBLEMAS ASOCIADOS A LAS COMUNICACIONES X-10

El diseño de las comunicaciones del protocolo X-10 es uno de los apartados vitales del desarrollo, ya que controlar la interfaz PC-CM11 supone controlar los dispositivos del hogar automatizado. El CM11 es el eje de las comunicaciones X-10, por lo que el estudio del protocolo X-10 –desarrollado en los capítulos anteriores- para a continuación diseñar la aplicación real supone uno de los aspectos principales del proyecto.

El hecho de que el protocolo X-10, como ya hemos citado en varias ocasiones a lo largo de varios capítulos, no sea un protocolo estándar, ha sido un hecho de vital importancia a la hora de recabar información, ya que la suma de datos contradictorios encontrados por la red es cuantiosa.

No existe un organismo regulador que haya adoptado o regulado los cambios y adherencia de nuevos datos al protocolo, ya que en vez de ser un estándar, es sólo un **protocolo de facto**.

La última versión "oficial" del protocolo es del 25 de diciembre de 1996 y son numerosos los errores e imprecisiones contenidos en el mismo. Por ello a la hora de estudiar el protocolo X-10 nos hemos valido de aportaciones de usuarios en foros o en artículos que ampliaban la información del protocolo en sí mismo publicado por la compañía X10. La verdad es que cada usuario aporta su granito de arena, y en varias ocasiones he tenido que preguntar en primera persona dudas muy concretas que la información ofrecida por el protocolo ni siquiera mencionaba. Resulta muy interesante el haber podido contactar con personas muy importantes dentro del ámbito de la domótica y sobre todo de X-10 como son Charles Sullivan y Dave Houston. Desde aquí mi agradecimiento.

Por otra parte, y a pesar de la publicidad de las empresas distribuidoras, la verdad es que el CM11 es un elemento inestable, que muchas veces se bloquea sin explicación aparente. Y el usuario debe tener mucho cuidado al enviarle dos órdenes seguidas sin considerar cierto tiempo de espera entre ambas, siempre que no desee que el CM11 "muera" en el intento de procesarlas.

En las distintas fases de desarrollo de la interfaz de comunicaciones con el CM11 hemos encontrado también ciertos obstáculos a la hora del diseño, que son los siguientes:

- **TRAMA BÁSICA DEL PROTOCOLO**

El objetivo que perseguíamos en esta fase era ofrecer al usuario la posibilidad de enviar al CM11 la trama básica del protocolo, que consta de código de casa, de dispositivo y de función (*"On","Off","Dim"*…).

Uno de los principales problemas que encontramos a la hora de comunicarnos con el CM11 fue precisamente la velocidad de los datos transmitidos a la red. Antes de acceder al puerto serie con *Visual Studio*, lo hicimos directamente con un programa de monitorización del puerto serie, que no poseía ningún tipo de temporizador. Entonces mantener un diálogo con el CM11 era realmente complicado, ya que responder al CM11 una décima de segundo más tarde suponía la pérdida del diálogo con el mismo.

Otro problema al que tuvimos que hacer frente se producía cuando recibíamos un dato por el puerto serie. En ese momento se producía una interrupción en el programa, y la ejecución se trasladaba a otro hilo distinto del hilo del programa original. Entonces Visual Studio nos impedía acceder a cualquier función o formulario de nuestro programa. Por ello tuvimos que hacer uso de los denominados "*delegados*".

En esa misma línea, dado que la recepción de cualquier dato por el puerto serie causaba una interrupción en el programa, fue problemático el poder analizar qué órdenes eran recibidas cuando se hacía uso del mando a distancia RF, ya que en ese caso se recibían un número de bytes indeterminado.

Por otra parte, las funciones "Status", "Status On", "Status Request" no han podido ser probadas en la aplicación, ya que sólo son compatibles con módulos bidireccionales, de los que no poseíamos ninguno.

- **PROGRAMACIÓN DE LA MACRO**

Esta fase ha sido de gran complejidad, no sólo por la compleja distribución de los datos en la Eeprom, sino por la escasez de

información de dicha distribución en sí, ya que el significado de los bytes a enviar a la Eeprom no está detallado en el protocolo.

Por otra parte, cada vez que el usuario desee programar una nueva macro, todos los valores almacenados previamente en la Eeprom han de ser modificados; no sólo los punteros, sino tanto las órdenes como los iniciadores, debido al complejo sistema de almacenamiento de datos. Establecer correctamente la dirección a los punteros era vital para el correcto funcionamiento del programa, ya que un simple error de un byte podía llevar al CM11 a un bucle en la ejecución del que es muy difícil salir.

Para evitarnos severos dolores de cabeza intentando activar una macro con un iniciador programado por el usuario previamente, debemos hacerlo siempre desde un dispositivo distinto al CM11, ya que de otra manera no funcionaría. A pesar de la importancia de este hecho, es algo que no viene reflejado en el protocolo.

11.5.2 PROBLEMAS ASOCIADOS A UPNP

Gracias a las herramientas de Intel, el diseño del dispositivo UPnP se ha visto simplificado en gran medida. Sin embargo, el problema residía en integrar el código generado por las herramientas Intel con el código del dispositivo X-10, y asociar las funciones UPnP a las funciones X-10.

Por otra parte, hay que tener presente que no todos los cortafuegos son compatibles con UPnP, de modo que impide el funcionamiento correcto de nuestra aplicación, ya que no permite el acceso libre a la red.