Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Desarrollo de un Servidor OPC UA para una tarjeta de adquisición de datos USB-1408FS-Plus

Autor:

Aponte Rengifo, Oscar Emilio

Tutor:

Zamarreño Cosme, Jesús María
Ingeniería de Sistemas y Automática

Valladolid, julio 2018.

Resumen

En este proyecto se presenta el desarrollo de un servidor OPC UA para una tarjeta de adquisición de datos USB-1408FS-Plus de la empresa *Measurement Computing.*

Para el desarrollo se ha utilizado una librería de código abierto en C denominada open62541 que implementa el estándar OPC UA así como la librería Universal Library de *Measurement Computing* para el acceso a la funcionalidad de la tarjeta.

El servidor desarrollado permite que cualquier cliente OPC UA pueda tener acceso estándar a la funcionalidad de la tarjeta: lectura/escritura tanto en canales analógicos como digitales.

El desarrollo será utilizado en el laboratorio de control de procesos del departamento de Ingeniería de sistemas y Automática para ampliar las posibilidades de conexión a las plantas piloto.

Palabras clave

Servidor OPC UA

Tarjeta de adquisición de datos

Programación en C

Agradecimientos

A mi madre,

Que gracias a su fuerza y coraje se ha logrado el objetivo.

A mi familia por creer en mí y que me han animado a seguir adelante y no darme por vencido.

Quiero agradecerle a mi tutor Jesús María Zamarreño Cosme, que me ha cedido su tiempo y conocimiento para que la realización de este trabajo fin de grado haya sido posible.

También deseo manifestar especial agradecimiento a Patricia por su apoyo incondicional durante los últimos años de la titulación.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

1.1. Introducción

Los procesos industriales pueden ser de distinta naturaleza, pero en general es común en ellos la necesidad del control de algunas magnitudes, por lo tanto el control de procesos consiste en el control de las variantes inherentes al mismo, una de las formas de acceder a estas variables es a través de tarjetas de adquisición de datos.

El estándar OPC ofrece un interfaz común para la comunicación entre componentes de software permitiendo que estos interactúen y compartan datos. El servidor OPC es la fuente de datos (como un dispositivo hardware a nivel de planta) y cualquier aplicación basada en OPC puede acceder a dicho servidor para leer/escribir cualquier variable que ofrezca el servidor.

OPC UA (Arquitectura Unificada) ha logrado extender el éxito del protocolo de comunicación OPC, para la adquisición de datos, el modelado de la información y la comunicación entre planta y aplicaciones de una forma fiable y segura.

El estándar OPC (en su versión OPC DA) está implementado en las plantas piloto del laboratorio de control automático del departamento de Ingeniería de Sistemas y Automática;

1.2. Objetivos

El objeto de este proyecto es el desarrollo e implementación de un servidor OPC UA para una tarjeta de adquisición de datos USB-1408FS-Plus de *Measurement Computing* [1].

Un Cliente OPC UA a través del Servidor OPC UA permitirá acceder a los datos adquiridos por la tarjeta 1408FS-Plus, así como escribir datos en la tarjeta.

Actualmente multitud de plantas piloto del laboratorio de control automático ubicado en el departamento de Ingeniería de Sistemas y Automática cuentan con servidores OPC DA, antiguo estándar de la Fundación OPC englobado bajo lo que se denomina OPC Clásico. OPC UA es una arquitectura orientada a servicios independientes de la plataforma que integra toda la funcionalidad de las especificaciones OPC *Classic* existentes y es compatible con OPC *Classic*.

En el laboratorio la comunicación entre la instrumentación asociada a las plantas piloto y los ordenadores se realiza mediante tarjetas de adquisición de datos.

Como muestra del entorno de trabajo, se muestra a modo de ejemplo una de las plantas del laboratorio (figura 1), que consta de 4 transmisores (nivel, caudal y 2 de temperatura) y 2 actuadores (válvula y resistencia calefactora). Estas señales (4 entradas analógicas y 2 salidas analógicas) se conectan a una tarjeta de adquisición de datos USB-1408FS-Plus para su manejo por parte del ordenador. Un servidor OPC UA que represente a la tarjeta de adquisición de datos permitirá estandarizar el acceso a la misma.

La implementación del OPC DA de las plantas piloto del laboratorio se realizó a través del TFG "Desarrollo de un servidor OPC para la tarjeta de adquisición de datos externa PMD-1208FS" [2], el cual ha servido como base para el desarrollo de este proyecto.

Figura 1 Planta piloto - laboratorio de control de procesos del departamento de Ingeniería de sistemas y Automática – UVA

Figura 2 Tarjeta de adquisición de datos USB-1408FS-Plus en planta piloto

1.3. Estructura de la memoria

La memoria se divide en 8 capítulos. Además del presente capítulo de introducción, consta de:

- Capítulo 2 : OPC UA

Este capítulo introduce el estándar OPC, desde el OPC Clásico hasta el actual OPC UA, mostrando todas las características y ventajas de OPC UA respecto a OPC Clásico. También se exponen los conceptos y el modelo de información de OPC UA necesarios para comprender la arquitectura de un servidor OPC UA.

Este capítulo también incluye una descripción de la librería Open62541, la cual se empleará para el desarrollo del servidor OPC UA.

- Capítulo 3: Tarjeta de adquisición de datos USB-1408FS-Plus

El cuarto capítulo detalla las características y aspectos importantes a considerar de la tarjeta USB-1408FS-Pluspara el desarrollo del servidor OPC UA. Además presenta las características de la *Universal Library*, que será empleada para acceder a las funcionalidades de la tarjeta de adquisición de datos.

- Capítulo 4: Desarrollo del software

Este capítulo abarca desde el análisis de los requisitos del software, la implementación de las funciones de la UL para el acceso a las funcionalidades de la tarjeta, el diseño y desarrollo del servidor basado en la modelo de información de OPC UA empleando para ello la librería open62541.

- Capítulo 5: Instalación de la tarjeta y del software del servidor

En este capítulo se detalla la instalación de una tarjeta USB-1408FS-Plus empleando *Instacal*, así como la instalación del servidor OPC UA.

- Capítulo 6: Implementación y validación

La implementación va desde el análisis del servidor en el cliente OPC UA hasta la comprobación del correcto funcionamiento del software para cada uno de los requisitos.

- Capítulo 7: Conclusiones y líneas futuras

En este apartado se detallan las conclusiones que se han podido extraer de la realización del proyecto, así como las posibles mejoras y ampliaciones que podrían realizarse en un futuro.

- Capítulo 8: Referencias

Por último, se incorpora una lista de los recursos utilizados en el desarrollo del trabajo para cada uno de las partes que lo componen.

2. OPC UA

2.1. OPC *Foundation*

El uso de sistemas de automatización basados en PC aumentó rápidamente en la automatización industrial al inicio de los 90; especialmente *PCs* basados en Windows para visualización y control. Uno de los mayores esfuerzos en el desarrollo de software estandarizado se ha dirigido al acceso a datos en un número incontable de dispositivos, redes, protocolos e interfaces.

Existía un problema similar para las aplicaciones software y el acceso a impresoras en los días del viejo MS-DOS: cada aplicación debía escribir su propio driver de impresora para todas las impresoras que fuera a soportar.

Windows resolvió el problema de drivers de impresora incorporando el soporte a impresoras en el propio sistema operativo. De forma que las aplicaciones sólo debían incorporar la interfaz al driver de impresora del sistema operativo. Y los fabricantes de impresoras ofrecían los drivers de impresora.

Como los fabricantes de Interfaces-Hombre-Máquina *(Human-Machine-Intefaces – HMI*) y software de Supervisión y Control (*Supervisory Control and Data Acquisition – SCADA*) tenían problemas similares, en 1995 se creó un grupo de trabajo formado por las compañías *Fischer-Rosemount*, *Rockwell Software*, *Opto 22*, *Intellution* e *Intuitive Technology*. El objetivo de este grupo de trabajo fue el de definir un estándar *Plug & Play* para drivers de dispositivos proporcionando un acceso estandarizado a datos de automatización en sistemas basados en Windows.

El resultado fue la especificación OPC *Data Access* (OPC-DA) en agosto de 1996. Esta especificación está soportada por una organización sin ánimo de lucro OPC *Foundation* y casi todos los proveedores de sistemas para automatización industrial son miembros de ella. La OPC *Foundation* era capaz de definir y adoptar estándares mucho más rápido que otras organizaciones utilizando APIs basados en tecnologías Microsoft Windows como (*Component Object Model* – COM) y (*Distributed* COM – DCOM).

OPC es el estándar de interoperabilidad para el intercambio seguro y confiable de datos en el espacio de automatización industrial y en otras industrias. Es independiente de la plataforma y garantiza el flujo continuo de información entre los dispositivos de múltiples proveedores. El estándar OPC es una serie de especificaciones desarrolladas por vendedores de la industria, usuarios finales y desarrolladores de software. Estas especificaciones definen la interfaz entre los clientes y los servidores, así como los servidores y servidores, incluido el acceso a datos en tiempo real, la supervisión de alarmas y eventos, el acceso a datos históricos y otras aplicaciones. [3]

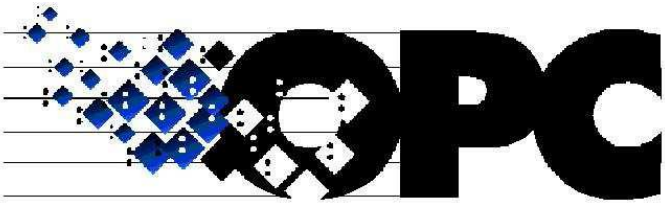


Figura 3 Logotipo de OPC - https://opcfoundation.org/

2.2. De OPC *Classic* a OPC-UA

Las especificaciones OPC *Classic* están basadas en tecnología *Microsoft Windows* usando COM/DCOM (*Distributed Component Object Model*) para el intercambio de dato entre componentes de software. Las especificaciones proporcionan las siguientes definiciones: para acceder a datos de proceso, alarmas y datos históricos [4].

- OPC *Data Access* (OPC DA)
  Define el intercambio de datos, incluidos los valores, el tiempo y la información de calidad [5].

- OPC *Alarms & Events* (OPC AE)
  Define el intercambio de información de mensajes de alarma y tipo de evento, así como estados variables y gestión de estado [6].

- OPC *Historical Data Access* (OPC HDA)
  Define métodos de consulta y análisis que pueden aplicarse a datos históricos con marca de tiempo [7].

Las especificaciones OPC *Classic* han servido bien a la comunidad OPC. Sin embargo, a medida que evolucionó la tecnología, también lo hizo la necesidad de especificaciones OPC.

En 2008, la OPC *Foundation* lanzó OPC *Unified Architecture* (OPC UA), una arquitectura orientada a servicios independiente de plataforma que integra toda la funcionalidad de las especificaciones OPC *Classic* existentes y es compatible con OPC *Classic*. Varios factores influyeron en la decisión de crear OPC UA:

- *Microsoft* ha quitado énfasis a COM (Modelo de Objetos Componentes) y DCOM (COM Distribuido) a favor de SOA multiplataforma (Arquitectura Orientada a Servicios).
- Los proveedores de OPC quieren un solo conjunto de servicios para exponer los modelos de datos de OPC, tales como acceso a datos, alarmas y eventos, acceso a datos históricos, etc.
- Para seguir siendo competitivos, los proveedores de OPC deben implementar OPC en sistemas que no sean de Microsoft, incluidos los dispositivos integrados.
- Otras organizaciones colaboradoras necesitan una forma confiable y eficiente de transportar datos estructurados de alto nivel. Los usuarios requieren la capacidad de acceder a los servidores OPC a través de firewalls de manera segura.
  [8]

2.3. OPC UA

OPC *Unified Architecture* [9] es una arquitectura orientada a servicios independiente de la plataforma que integra toda la funcionalidad de las especificaciones OPC *Classic* individuales en un marco extensible.

Este enfoque de varias capas cumple los objetivos de la especificación de diseño original de:

- Equivalencia funcional: todas las especificaciones COM de OPC *Classic* se asignan a UA.
- Independencia de la plataforma: desde un microcontrolador integrado a una infraestructura basada en la nube.
- Seguro: cifrado, autenticación y auditoría.
- Extensible: capacidad de agregar nuevas funciones sin afectar a las aplicaciones existentes.
- Modelado integral de información: para definir información compleja.

Equivalencia Funcional

Sobre la base del éxito de OPC *Classic*, OPC UA fue diseñado para mejorar y superar las capacidades de las especificaciones OPC Classic. OPC UA es funcionalmente equivalente a OPC Classic, pero capaz de mucho más:

- *Discovery*: La disponibilidad de servidores OPC en computadoras y / o redes locales
- *Address space*: todos los datos se representan de forma jerárquica (por ejemplo, archivos y carpetas), lo que permite que los clientes de OPC descubran y utilicen estructuras simples y complejas.
- *On-demand*: leer y escribir datos / información en función de permisos de acceso
- *Subscriptions*: monitorizar datos / información e informe por excepción cuando los valores cambian según los criterios del cliente.
- *Events*: notificar información importante según los criterios del cliente.
- *Methods*: los clientes pueden ejecutar programas, etc. según los métodos definidos en el servidor.

Plataforma independiente

Dada la amplia gama de plataformas de hardware y sistemas operativos disponibles, la independencia de la plataforma es esencial. OPC UA funciona en cualquiera de los siguientes y más:

- Plataformas de hardware: hardware tradicional de PC, servidores basados en la nube, PLC, microcontroladores (ARM, etc.)
- Sistemas operativos: Microsoft Windows, Apple OSX, Android o cualquier distribución de Linux, etc.

OPC UA proporciona la infraestructura necesaria para la interoperabilidad en toda la empresa, de máquina a máquina, de máquina a empresa y todo lo que se encuentre en el medio.

Seguridad

Una de las consideraciones más importantes al elegir una tecnología es la seguridad. OPC UA es amigable con el *firewall* al abordar las preocupaciones de seguridad al proporcionar un conjunto de controles:

- Transporte: se definen numerosos protocolos que proporcionan opciones tales como el transporte OPC-binario ultrarrápido o el SOAP-HTTPS más universalmente compatible.

- Cifrado de sesión: los mensajes se transmiten de forma segura a niveles de cifrado de 128 o 256 bits.
- Firma de mensajes: los mensajes se reciben exactamente como se enviaron.
- Paquetes secuenciados: la exposición a los ataques de reproducción de mensajes se elimina con la secuencia.
- Autenticación: cada cliente y servidor UA se identifica a través de certificados *OpenSSL* que proporcionan control sobre qué aplicaciones y sistemas pueden conectarse entre sí.
- Control de usuario: las aplicaciones pueden requerir que los usuarios se autentiquen (credenciales de inicio de sesión, certificado, etc.) y pueden restringir y mejorar aún más sus capacidades con derechos de acceso y "vistas" de espacio de direcciones.
- Auditoría: las actividades por usuario y / o sistema se registran proporcionando un seguimiento de auditoría de acceso.

Extensible

La arquitectura multicapa de OPC UA proporciona un marco "a prueba del futuro". Las tecnologías y metodologías innovadoras, como los nuevos protocolos de transporte, algoritmos de seguridad, estándares de codificación o servicios de aplicaciones, se pueden incorporar a OPC UA a la vez que se mantiene la compatibilidad con versiones anteriores para productos existentes. Los productos UA construidos hoy trabajarán con los productos del mañana.

Modelado de información

El marco de modelado de información OPC UA convierte los datos en información. Con capacidades completas orientadas a objetos, incluso las estructuras más complejas de niveles múltiples se pueden modelar y extender. Los tipos de datos y las estructuras se definen en los perfiles. Por ejemplo, las especificaciones de OPC *Classic* existentes se modelaron en perfiles UA que también pueden ser extendidos por otras organizaciones.

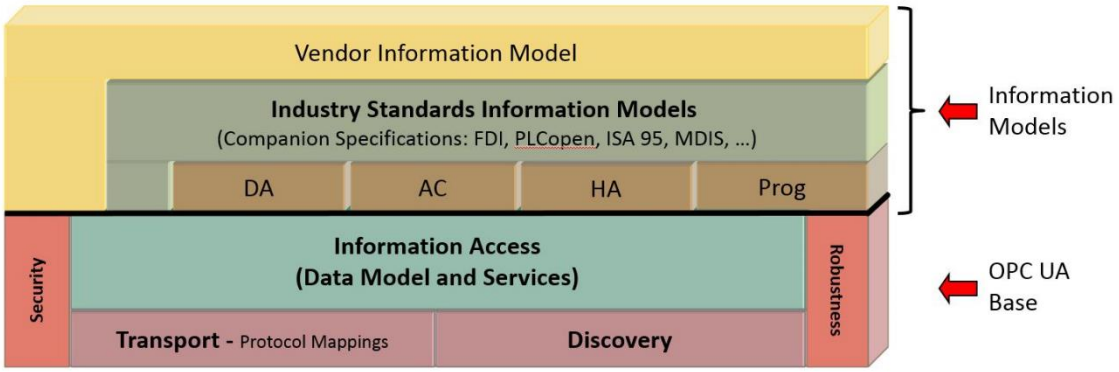


Figura 4 Modelo de información de OPC UA - https://opcfoundation.org/about/opc-technologies/opc-ua/

Especificaciones OPC UA

Las especificaciones OPC UA se dividen en varios documentos:

- OPC Foundation. Part 1: OPC UA Specification: Part 1 – Concepts.
- OPC Foundation. Part 2: OPC UA Specification: Part 2 – Security Model.
- OPC Foundation. Part 3: OPC UA Specification: Part 3 – Address Space Model.
- OPC Foundation. Part 4: OPC UA Specification: Part 4 – Services.
- OPC Foundation. Part 5: OPC UA Specification: Part 5 – Information Model.
- OPC Foundation. Part 6: OPC UA Specification: Part 6 – Mappings.
- OPC Foundation. Part 7: OPC UA Specification: Part 7 – Profiles.
- OPC Foundation. Part 8: OPC UA Specification: Part 8 – Data Access.
- OPC Foundation. Part 9: OPC UA Specification: Part 9 – Alarms and Conditions.
- OPC Foundation. Part 10: OPC UA Specification: Part 10 – Programs.
- OPC Foundation. Part 11: OPC UA Specification: Part 11 – Historical Access.
- OPC Foundation. Part 12: OPC UA Specification: Part 12 – Discovery and Global.
- OPC Foundation. Part 13: OPC UA Specification: Part 13 – Aggregates.
- OPC Foundation. Part 14: OPC UA Specification: Part 14 – PubSub.

2.4. Conceptos OPC UA

A continuación se muestran los conceptos OPC UA [10] considerados para este proyecto.

Arquitectura Cliente – Servidor

La arquitectura de sistemas OPC UA modela los clientes y servidores como socios que interactúan. Cada sistema puede contener múltiples Clientes y Servidores. Cada Cliente puede interactuar simultáneamente con uno o más Servidores, y cada servidor puede interactuar concurrentemente con uno o más Clientes.

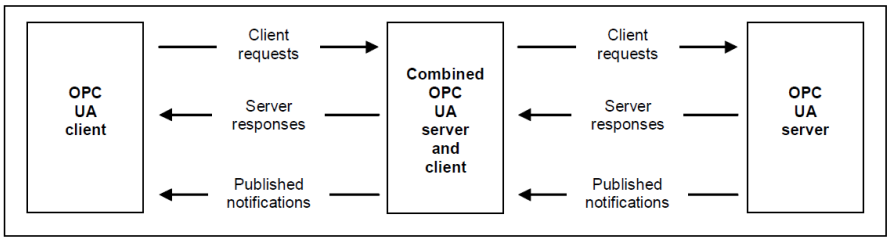


Figura 3 OPC UA System Architecture –OPC UA Specification: Part 1 Concepts http://www.opcfoundation.org/UA/Part1/

Cliente OPC UA

La arquitectura del cliente OPC UA modela el punto final del cliente de las interacciones cliente / servidor. La figura ilustra los principales elementos de un Cliente típico y cómo se relacionan entre sí.

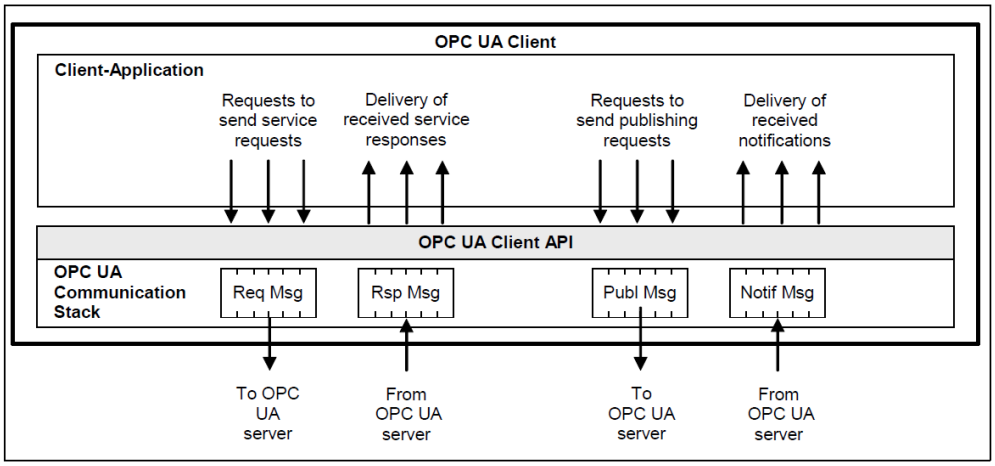


Figura 5 OPC UA Client Architecture –OPC UA Specification: Part 1 Concepts http://www.opcfoundation.org/UA/Part1/

La aplicación de cliente es el código que implementa la función del cliente. Utiliza la API de cliente para enviar y recibir solicitudes de servicio OPC UA y respuestas al servidor.

La "API cliente" es una interfaz interna que aísla el código de la aplicación cliente de una pila de comunicación OPC UA. La pila de comunicación OPC UA convierte las llamadas de API de cliente en mensajes y las envía a través de la entidad de comunicaciones subyacente al servidor a petición de la aplicación de cliente. La pila de comunicación OPC UA también recibe respuesta y mensajes de notificación de la entidad de comunicaciones subyacente y los entrega a la aplicación del cliente a través de la API del cliente.

Servidor OPC UA

La arquitectura del servidor OPC UA modela el punto final del servidor de las interacciones cliente / servidor. La Figura ilustra los principales elementos del Servidor y cómo se relacionan entre sí.

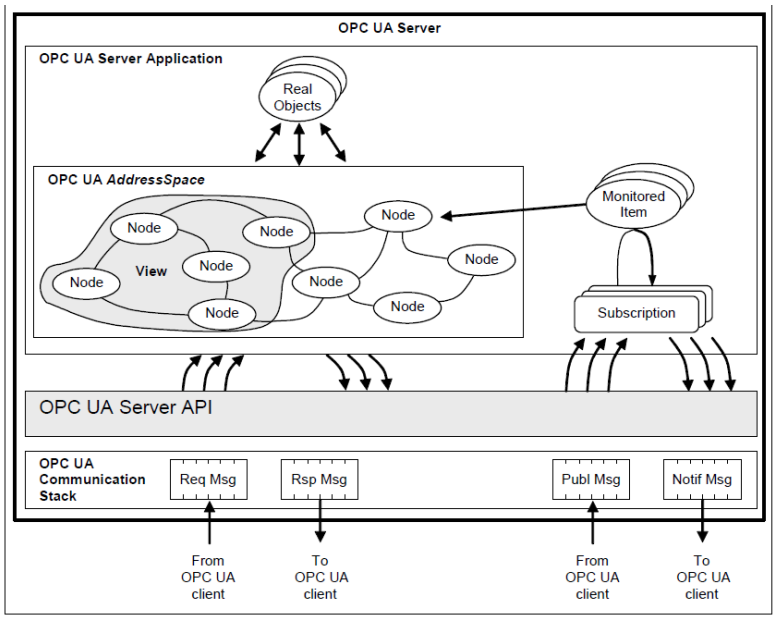


Figura 6 OPC UA Server Architecture –OPC UA Specification: Part 1 Concepts http://www.opcfoundation.org/UA/Part1/

Los objetos reales son objetos físicos o de software a los que puede acceder la aplicación Servidor o que mantiene internamente.

La aplicación de servidor es el código que implementa la función del servidor. Utiliza la API del servidor para enviar y recibir mensajes OPC UA de los clientes. Tenga

en cuenta que la "API del servidor" es una interfaz interna que aísla el código de la aplicación del servidor de una pila de comunicación OPC UA.

AddressSpace OPC UA

AddressSpace (espacio de direcciones) se modela como un conjunto de Nodos accesibles por los Clientes que utilizan los Servicios OPC UA (interfaces y métodos). Los nodos en *AddressSpace* se utilizan para representar objetos reales, sus definiciones y sus referencias entre sí.

Los servidores son libres de organizar sus Nodos dentro del *AddressSpace* como lo deseen. El uso de las referencias entre nodos permite a los servidores organizar el espacio de direcciones en jerarquías, una red de nodos de malla completa o cualquier mezcla posible.

Las vistas se usan para restringir los Nodos que el Servidor hace visibles al Cliente, restringiendo así el tamaño del Espacio de Direcciones para las solicitudes de Servicio enviadas por el Cliente. La vista por defecto es todo el *AddressSpace*. Los servidores pueden definir opcionalmente otras vistas. Las vistas ocultan algunos de los Nodos o Referencias en el *AddressSpace*.

OPC UA *AddressSpace* admite modelos de información. Este soporte se proporciona a través de:

a) *Node References* que permiten que los objetos en la *AddressSpace* se relacionen entre sí.

b) *Nodos ObjectType* que proporcionan información semántica para objetos reales (definiciones de tipo).

c) Nodos ObjectType para admitir subclases de definiciones de tipo.

d) *Data type* expuestas en el *AddressSpace* que permiten el uso de tipos de datos específicos de la industria.

e) Estándares complementarios de OPC UA que permiten definir cómo se representarán sus modelos de información específicos en el *Server AddressSpace*.

Entidades de suscripción

MonitoredItems son entidades en el servidor creados por el cliente que supervisa los nodos *AddressSpace* y sus contrapartes del mundo real. Cuando detectan un cambio de datos o un evento / alarma, generan una Notificación que se transfiere al Cliente mediante una Suscripción.

Una Suscripción es un punto final en el Servidor que publica Notificaciones a los Clientes. Los clientes controlan la velocidad de publicación al enviar mensajes de publicación.

2.5. Modelo de información OPC UA

El objetivo principal del *addressSpace* OPC UA es proporcionar una forma estándar para que los servidores representen objetos para los clientes. El modelo de objetos OPC UA ha sido diseñado para cumplir este objetivo. Define Objetos en términos de Variables y Métodos. También permite expresar las relaciones con otros objetos.

Además, los objetos pueden escribirse a máquina, es decir, OPC UA proporciona una forma de definir y exponer tipos de objetos (clases con variables miembro y métodos miembros) e instancias de objetos.

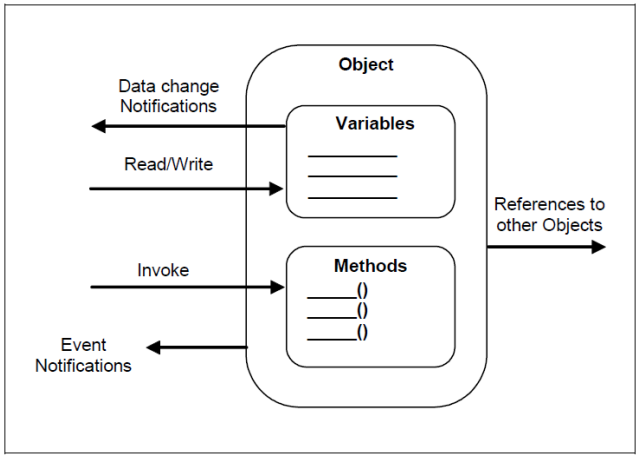


Figura 7 OPC UA model Object - Part 5: OPC UA Specification: Part 5 Information Model

http://www.opcfoundation.org/UA/Part5/

Los servicios UA se usan para acceder a los objetos y sus componentes, como leer o escribir un valor variable, llamar a un método o recibir eventos del objeto. El servicio de exploración se puede utilizar para explorar las relaciones entre los objetos y sus componentes.

Los elementos de este modelo están representados en el *addressSpace* como nodos. Cada nodo se asigna a una clase de nodo, por ejemplo, objeto, variable y método, y representa un elemento diferente del modelo de objetos.

El conjunto de objetos y la información relacionada que el servidor OPC UA pone a disposición de los clientes es su *addressSpace*.

Los objetos y sus componentes se representan en el espacio de direcciones como un conjunto de nodos descritos por atributos e interconectados por referencias.

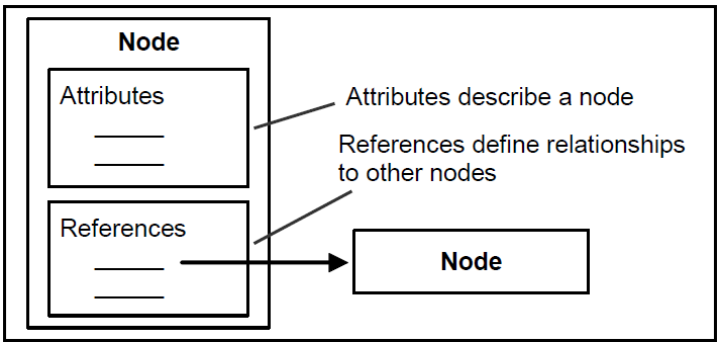


Figura 8 AddressSpace Node Model - Part 5: OPC UA Specification: Part 5 Information Model

http://www.opcfoundation.org/UA/Part5/

Clases de nodo

Las clases de nodo definen atributos y referencias para diferentes nodos. OPC UA define ocho clases de nodos. Cada nodo en el espacio de direcciones es una instancia de una de estas clases de nodos. Los clientes y servidores no pueden definir clases de nodo adicionales ni ampliar las definiciones de estas clases de nodo.

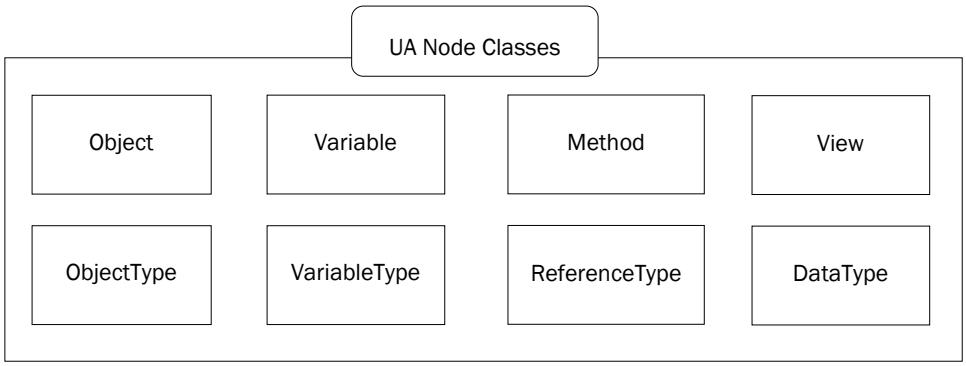


Figura 9 Clases de Nodo

Atributos

Los atributos son elementos de datos que describen nodos. Los clientes pueden acceder a los valores de los atributos mediante los servicios de Lectura, Escritura, Consulta y Suscripción.

Los atributos son componentes elementales de las clases de nodo. Las definiciones de atributos forman parte de las definiciones de clase de nodo y, por lo tanto, son conocidas por los clientes y no son directamente visibles en el espacio de direcciones.

Cada definición de atributo consiste en un identificador de atributo, un nombre, una descripción, un tipo de datos y un indicador obligatorio / opcional. Los clientes o servidores no pueden extender el conjunto de atributos definidos para cada clase de nodo. Cuando se crea una instancia de un nodo en el espacio de direcciones, se deben proporcionar los valores de los Atributos de clase de nodo obligatorios.

Referencias

Las referencias se utilizan para relacionar nodos entre sí. Se puede acceder a ellos utilizando los servicios de navegación y consulta.

Al igual que los atributos, se definen como componentes fundamentales de los nodos. A diferencia de los atributos, las referencias se definen como instancias de nodos *ReferenceType*. Los nodos *ReferenceType* son visibles en el espacio de direcciones y se definen con la clase de nodo *ReferenceType*.

El nodo que contiene la referencia se denomina nodo de origen y el nodo al que se hace referencia se denomina nodo de destino. La combinación del nodo de origen, el *ReferenceType* y el nodo de destino se utilizan en los servicios de OPC UA para identificar de forma exclusiva las referencias.

El nodo objetivo de una referencia puede estar en el mismo espacio de direcciones o en el espacio de direcciones de otro servidor OPC UA. Los nodos de destino ubicados en otros servidores se identifican en los servicios OPC UA utilizando una combinación del nombre del servidor remoto y el identificador asignado al Nodo por el servidor remoto.

Variables

Las variables se utilizan para representar valores. Se definen dos tipos de variables, *properties* y *dataVariables*. Difieren en el tipo de datos que representan y si pueden contener otras variables.

Las propiedades difieren de los atributos en que caracterizan lo que representa el nodo, como un dispositivo o una orden de compra.

DataVariables representa el contenido de un objeto. Por ejemplo, un archivo *Object* puede ser definido que contiene una secuencia de bytes. La secuencia de

bytes se puede definir como una DataVariable que es una matriz de bytes. Las propiedades se pueden usar para exponer el tiempo de creación y el propietario del archivo Objeto. [11] [12]

2.6. Protocolo OPC UA

Este apartado ofrece una descripción general del protocolo binario OPC UA [13], ya que es el que se implementó en open62541. El protocolo binario basado en TCP es el más común en la capa de transporte de OPC UA.

La comunicación en OPC UA se comprende mejor empezando por los siguientes principios.

- Solicitud/Respuesta

Toda la comunicación se basa en el modelo Solicitud/Respuesta. Solo los clientes pueden enviar una solicitud a un servidor. Y los servidores solo pueden responder a una solicitud.

- Respuestas asíncronas

Un servidor no tiene que responder inmediatamente a las solicitudes y las respuestas se pueden enviar en un orden diferente. Además, las suscripciones (también conocidas como notificaciones push) se implementan a través de solicitudes especiales donde la respuesta se retrasa hasta que se genera una notificación.

Establecimiento de conexión

Una conexión cliente-servidor en OPC UA consta de tres niveles: conexión sin formato, canal seguro y sesión

- Conexión sin procesar

La conexión sin procesar se crea al abrir una conexión TCP al nombre de host y puerto correspondiente y a un saludo inicial HEL/ACK. El protocolo de enlace establece los ajustes básicos de la conexión, como la longitud máxima del mensaje.

- Canal seguro

SecureChannels se crean en la parte superior de la conexión TCP sin procesar. Se establece un SecureChannel con una solicitud OpenSecureChannel y un par de mensajes de respuesta.

Con la firma de mensajes o el cifrado habilitados, los mensajes de OpenSecureChannel se encriptan usando un algoritmo de encriptación asimétrica

(criptografía de clave pública). Como parte de los mensajes de OpenSecureChannel, el cliente y el servidor establecen un secreto común sobre un canal inicialmente no seguro. Para los mensajes subsiguientes, el secreto común se utiliza para el cifrado simétrico, que tiene la ventaja de ser mucho más rápido.

Diferentes políticas de seguridad - definidas en la parte 7 del estándar OPC UA - especifican los algoritmos para el cifrado asimétrico y simétrico, longitudes de clave de cifrado, funciones hash para la firma de mensajes, y más.

- Sesión

Las sesiones se crean sobre un SecureChannel. Esto garantiza que los usuarios puedan autenticarse sin enviar sus credenciales, como nombre de usuario y contraseña, en texto sin formato. Los mecanismos de autenticación actualmente definidos son el inicio de sesión anónimo, el nombre de usuario / contraseña, los certificados Kerberos y x509. Este último requiere que el mensaje de solicitud vaya acompañado de una firma para demostrar que el remitente posee la clave privada con la que se creó el certificado.

Se requieren dos intercambios de mensajes para establecer una sesión: CreateSession y ActivateSession. El servicio ActivateSession se puede usar para cambiar una sesión existente a un SecureChannel diferente. Esto es importante, por ejemplo, cuando la conexión se averió y la sesión existente se reutiliza con un nuevo SecureChannel.

Estructura del mensaje

- Encabezado del mensaje

El encabezado del mensaje contiene información básica, como la longitud del mensaje, así como la información necesaria para relacionar los mensajes a un canal seguro y cada solicitud a la respuesta correspondiente. "Chunking" se refiere a la división y el reensamblado de mensajes que son más largos que el tamaño máximo de paquete de red.

- Cuerpo del mensaje

Todos los servicios OPC UA tienen una firma en forma de una estructura de datos de solicitud y respuesta. Estos se definen de acuerdo con el sistema de tipo de protocolo OPC UA. El cuerpo del mensaje comienza con el identificador del siguiente tipo de datos. Luego, sigue la carga principal del mensaje.

2.7. Open62541 v0.3.

open62541 [14] es una implementación de open *source* y gratuita de OPC UA escrita en el subconjunto común de los lenguajes C99 y C ++ 98. La biblioteca se puede utilizar con todos los principales compiladores y proporciona las herramientas necesarias para implementar clientes y servidores OPC UA, o para integrar comunicaciones basadas en OPC UA en aplicaciones existentes. La biblioteca open62541 es independiente de la plataforma. Toda la funcionalidad específica de la plataforma se implementa mediante complementos intercambiables. Las implementaciones de complementos se proporcionan para los principales sistemas operativos.

Características

open62541 implementa la pila de protocolo binario OPC UA, así como un SDK de cliente y servidor. Actualmente es compatible con el perfil del servidor del dispositivo micro incorporado más algunas características adicionales. Los binarios del servidor pueden tener un tamaño muy inferior a 100 kb, según el modelo de información que contenga.

Stack de comunicación

- Protocolo binario OPC UA.
- *Chunking* (división de grandes mensajes).
- Capa de red intercambiable (complemento) para utilizar API de redes personalizadas (por ejemplo, en objetivos integrados).
- Comunicación encriptada.
- Solicitudes de servicio asíncrono en el cliente.

Modelo de información

- Soporte para todo tipo de nodos OPC UA.
- Soporte para agregar y eliminar nodos y referencias también en tiempo de ejecución.
- Soporte para herencia y creación de instancias de tipos de objetos y variables.
- Control de acceso para nodos individuales.

Suscripciones

- Soporte para suscripciones/elemento supervisados para notificaciones de cambio de datos.
- Consumo de recursos muy bajo para cada valor supervisado.

3. Tarjeta de adquisición de datos: DAQ-USB-1408FS-Plus

3.1. Introducción

La tarjetas de adquisición de datos son dispositivos cuya función es obtener muestra de una variable física (voltaje, temperatura, nivel se sonido, etc.), es decir, toman una señal de un sistema sensor-transmisor (sistema analógico) y después la adecuan para transformarla en un dato que pueda ser reconocido y registrado por un sistema digital con el fin de que la pueda leer una computadora y realizar una tarea en específico mediante un software específico [15].

Un PC con software programable controla la operación del dispositivo DAQ y es usada para procesar, visualizar y almacenar datos de medida. Diferentes tipos de PCs son usados en diferentes tipos de aplicaciones. Un PC de escritorio se puede utilizar en un laboratorio por su poder de procesamiento, una laptop se puede utilizar por su portabilidad o una PC industrial se puede utilizar en una planta de producción por su robustez.

El hardware DAQ actúa como la interfaz entre un PC y señales del mundo exterior. Funciona principalmente como un dispositivo que digitaliza señales analógicas entrantes para que un PC pueda interpretarlas. Los tres componentes clave de un dispositivo DAQ usado para medir una señal son el circuito de acondicionamiento de señales, convertidor analógico-digital (ADC) y un bus de PC. Varios dispositivos DAQ incluyen otras funciones para automatizar sistemas de medidas y procesos. Por ejemplo, los convertidores digitales-analógicos (DAC) envían señales analógicas, las líneas de E/S digital reciben y envían señales digitales y los contadores/temporizadores cuentan y generan pulsos digitales [16].

Acondicionamiento de Señales

Las señales de los sensores o del mundo exterior pueden ser ruidosas o demasiado peligrosas para medirse directamente. El circuito de acondicionamiento de señales manipula una señal de tal forma que es apropiado para entrada a un ADC. Este circuito puede incluir amplificación, atenuación, filtrado y aislamiento. Algunos dispositivos DAQ incluyen acondicionamiento de señales integrado diseñado para medir tipos específicos de sensores.

Convertidor Analógico Digital (ADC)

Las señales analógicas de los sensores deben ser convertidas en digitales antes de ser manipuladas por el equipo digital como un PC. Un ADC es un chip que proporciona una representación digital de una señal analógica en un instante de tiempo. En la práctica, las señales analógicas varían continuamente con el tiempo y un ADC realiza "muestras" periódicas de la señal a una razón predefinida. Estas muestras son transferidas a un PC a través de un bus, donde la señal original es reconstruida desde las muestras en software.

Bus del PC

Los dispositivos DAQ se conectan a un PC a través de una ranura o puerto. El bus del PC sirve como la interfaz de comunicación entre el dispositivo DAQ y el PC para pasar instrucciones y datos medidos. Los dispositivos DAQ se ofrecen en los buses de PC más comunes, incluyendo USB, PCI, PCI Express y Ethernet. Recientemente, los dispositivos DAQ han llegado a estar disponibles para 802.11 Wi-Fi para comunicación inalámbrica. Hay varios tipos de buses y cada uno de ellos ofrece diferentes ventajas para diferentes tipos de aplicaciones.

3.2. Tarjeta de adquisición de datos USB-1408FS-PLUS de *Measurement Computing*

Este apartado incluye la descripción de la estructura y funcionamiento de la tarjeta USB-1408FS-Plus [17] de *Measurement Computing* sobre la cual se implementará el servidor OPC UA.

Figura 10 Tarjeta de adquisición de datos USB-1408FS-Plus

La tarjeta obtiene su alimentación de funcionamiento de 5 voltios al ser conectada al puerto USB del ordenador. Posee un led de color verde que al estar encendido indica que la tarjeta es alimentada, mientras que si el encendido del led es intermitente indica que datos están siendo transferidos.

El USB-1408FS-Plus es compatible con puertos USB 1.1 y USB 2.0. La velocidad del dispositivo puede estar limitada cuando se utiliza un puerto USB 1.1 debido a la diferencia en las tasas de transferencia en las versiones USB 1.1 del protocolo (baja velocidad y velocidad completa).

Terminales

- Ocho conexiones de entrada analógica (CH0 IN a CH7 IN, CH0 IN HI/LO a CH3 IN HI/LO)
- Dos conexiones de salida analógica (D/A OUT 0 a D/A OUT 1)
- 16 conexiones de E/S digitales (PortA0 a Port A7, y Port B0 a Port B7)
- Entrada de activación externa (TRIG_IN)
- Entrada de contador externo (CTR)

- Terminal bidireccional para reloj externo o sincronización de unidades múltiples (SYNC)
- Potencia de salida (+ VO)
- Tierra analógica (AGND) y tierra (GND)

Se deben usar cables de 16 AWG (Calibración de alambre estadounidense) a 30AWG para la conexión a los terminales de la tarjeta.

Las funciones del dispositivo se ilustran en el diagrama de bloques que se muestra aquí.

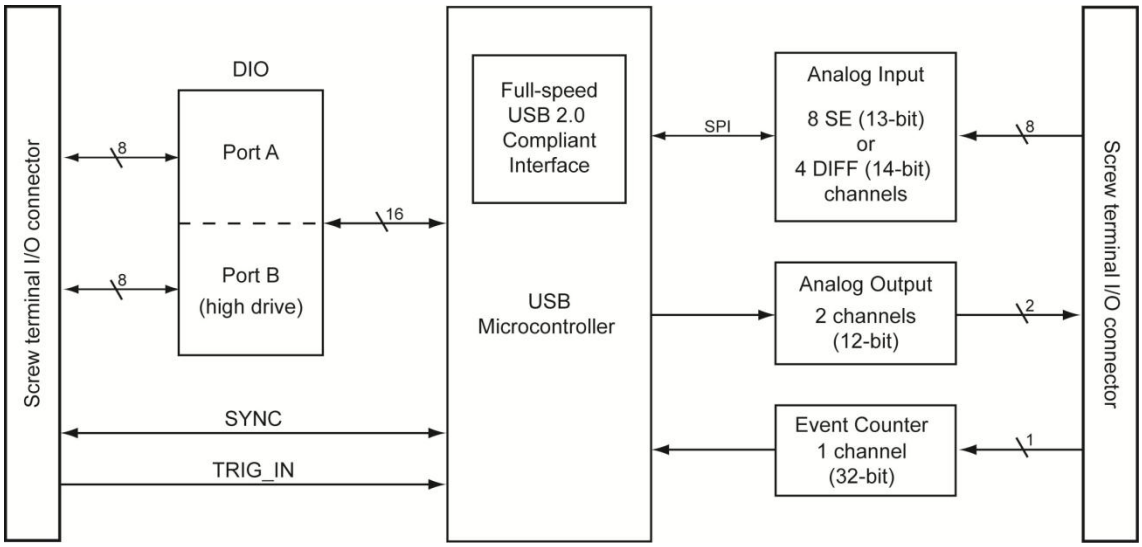


Figura 11 USB-1408FS-PLUS Diagrama de bloques funcional

3.2.1. Entradas analógicas

La tarjeta permite realizar hasta ocho conexiones de entrada analógica a los terminales que contiene los pines 1 a 20 (CH0 IN a CH7 IN).

Se puede configurar los canales de entrada analógica como ocho canales de un solo terminal o cuatro canales diferenciales. Cuando se configura para el modo diferencial, cada entrada analógica tiene una resolución de 14 bits. Cuando se configura para el modo de terminación única, cada entrada analógica tiene una resolución de 13 bits, debido a las restricciones impuestas por el convertidor A / D. La configuración se realiza a través de la aplicación “InstaCal” proporcionada por el fabricante.

Configuración en modo común

Cuando se configura para el modo de entrada en modo común, la señal de entrada se hace referencia a la tierra de la señal (GND) y se envía a través de dos cables:

- Se conecta el cable que transporta la señal que se va a medir a CH # IN.
- Se conecta el segundo cable a AGND.

El rango de entrada para el modo común es ± 10 V. El pin-out del modo común se muestra en la siguiente figura.

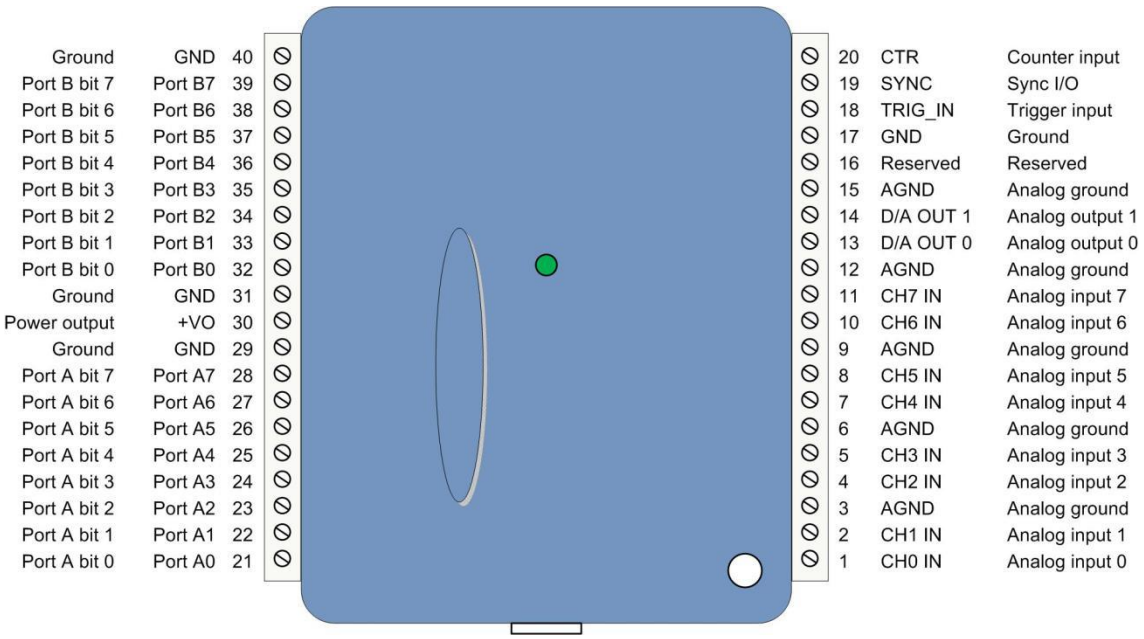


Figura 12 USB-1408FS-Plus en modo común

Configuración en modo diferencial

Cuando se configura para el modo de entrada diferencial, la señal de entrada se mide con respecto a la entrada baja y se entrega a través de tres cables:

- Se conecta el cable que transporta la señal que se va a medir a CH # IN HI
- Se conecta el cable que transporta la señal de referencia a CH # IN LO
- Se conecta el tercer cable a GND.

El pin-out del modo diferencial se muestra en la Figura 13.

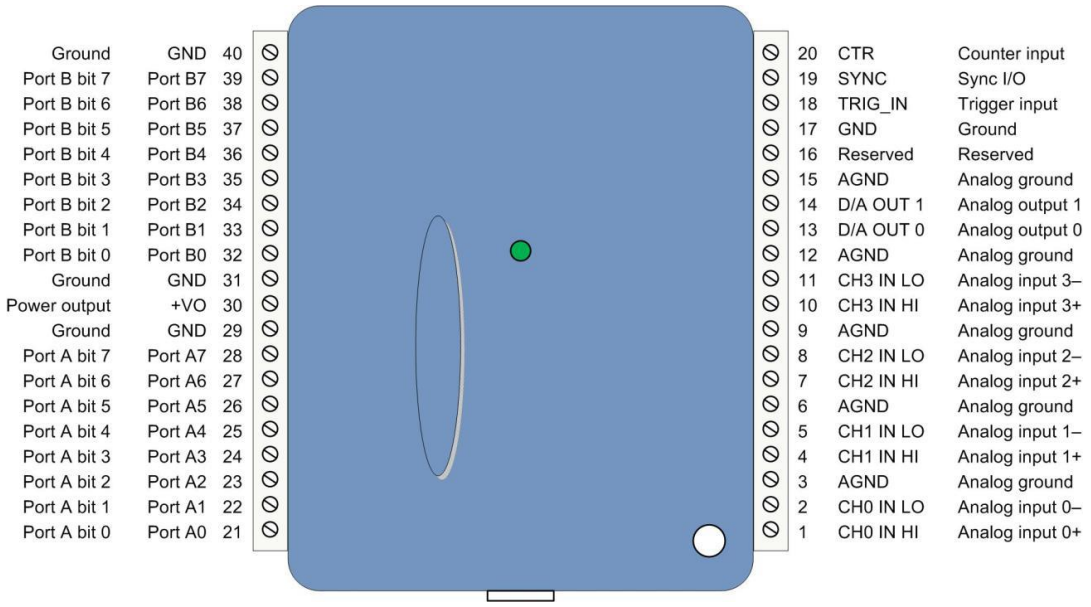


Figura 13 USB-1408FS-Plus en modo diferencial

Nota: Para realizar una medición de un solo extremo utilizando canales diferenciales, conecte la señal a CH # IN HI y conecte a tierra la entrada asociada CH # IN LO.

Un amplificador de ganancia programable de precisión (PGA) de bajo ruido está disponible en canales diferenciales para proporcionar ganancias de hasta 20 y un rango dinámico de hasta 14 bits. Los rangos de voltaje de entrada del modo diferencial son ± 20 V, ± 10 V, ± 5 V, ± 4 V, ± 2,5 V, ± 2,0 V, ± 1,25 V y ± 1,0 V.

En el modo diferencial, se deben cumplir los dos requisitos siguientes para la operación lineal:

- Cualquier entrada analógica debe permanecer en el rango de -10 V a + 20 V con respecto a tierra en todo momento.
- La tensión diferencial máxima en cualquier par de entrada analógica debe permanecer dentro del rango de tensión seleccionado.

La entrada [voltaje + señal en modo común] del canal diferencial debe estar en el rango de -10 V a +20 V para obtener un resultado útil.

3.2.2. Salidas analógicas

Se pueden conectar hasta dos conexiones de salida analógica a D/A OUT 0 y D/A OUT 1. Cada canal se puede estimar a velocidades de hasta 50,000 actualizaciones por segundo. El rango de salida es de 0 V a 5 V.

3.2.3. Entradas y Salidas digitales

El dispositivo tiene 16 canales DIO que están configurados como dos puertos de 8 bits: puerto A y puerto B. Puede conectar hasta ocho líneas DIO al puerto A0 al puerto A7 y hasta a ocho líneas DIO al puerto B0 al puerto B7. Puede configurar cada puerto para entrada o salida. Los puertos digitales se configuran para la entrada cuando el dispositivo está encendido o restablecido.

Cuando se configura para la entrada, los terminales de E/S digitales pueden detectar el estado de cualquier nivel de TTL.

3.3. Universal library

La *Universal Library (UL)* para Windows incluye bibliotecas de programación y componentes para desarrollar aplicaciones de 32 y 64 bits con Visual Studio® y otros lenguajes de programación. UL admite Ethernet, USB, WLS, WEB y la mayoría de los productos de adquisición de datos PCI y PCIe (DAQ) de MCC (*Measurement Computing*), y simplifica su configuración y funcionamiento [18].

Parte de este paquete incluye *instaCal* que sirve para instalar, calibrar y testear las tarjetas conectadas al ordenador.

Una vez conectada la USB-1408FS-Plus *InstaCal* la detecta y asigna los recursos necesarios para el almacenamiento de los datos. *InstaCal* se debe usar para instalar o desinstalar nuevas tarjetas, además para cambiar sus configuraciones.

4. Desarrollo del software

Tras analizar la tarjeta USB-1408FS-Plus, las funcionalidades a las que se puede acceder a través de la UL, los conceptos y modelo de información de OPC UA, y así como la librería open62541, El siguiente paso al análisis general es el descrito en este capítulo: Los objetivos del software, los requisitos del software, descripción de los casos de uso, las funciones de la librería UL que utilizará el programa para acceder a las funcionalidades del sistema, el modelo del servidor basado en el modelo de información de OPC UA, los casos de uso explicados a través de diagramas de secuencia, las funciones empleadas de la librería open62541 para el desarrollo del servidor, y por último, las funciones desarrolladas en el server.c que engloban el modelo de información junto con las funciones de la librería open62541 , todo esto empleando lenguaje C.

4.1. Objetivos del software

El Servidor OPC UA hace de interfaz comunicando con la tarjeta de adquisición de datos 1408FS-Plus de "*Measurement Computing*" y por otro lado con cualquier Cliente OPC UA. La comunicación entre el cliente y el servidor es bidireccional, lo que significa que los clientes pueden leer y escribir en la tarjeta a través del servidor OPC UA.

4.2. Requisitos del software

Los requisitos del software comprenden las funcionalidades y los servicios que debe proporcionar.

- La aplicación debe permitir leer la configuración de la tarjeta (Común/diferencial).
- La aplicación debe permitir leer los datos de todos los canales analógicos de entrada de la tarjeta.
- La aplicación debe permitir mostrar los datos de los canales analógicos de entradas en formato binario, voltios.
- La aplicación debe permitir fijar y consultar el rango de los canales analógicos de entrada.
- La aplicación debe permitir escribir sobre los canales analógicos de salida de la tarjeta
- La aplicación debe permitir escribir sobre los canales analógicos de salida en diferentes formatos (binario, volts).
- La aplicación debe permitir escribir y leer sobre los puertos digitales.

- La aplicación debe permitir configurar los puertos digitales tanto en modo salida como en modo entrada.

4.3. Casos de uso

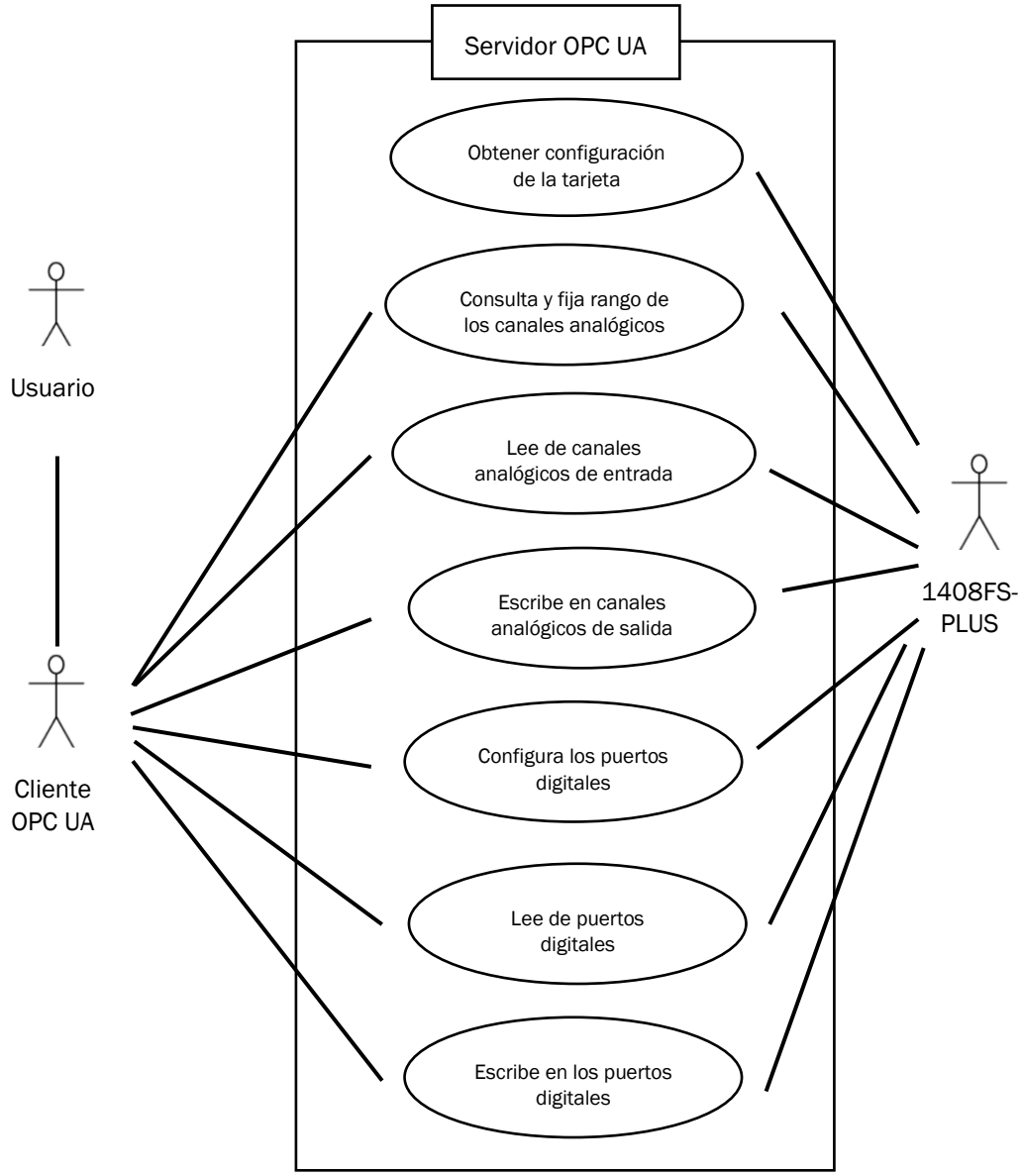
Actores:

Usuario: Aquel que inicializa la aplicación y usa el interfaz para la consulta de datos.

Cliente OPC UA: Se conecta al servidor OPC UA, y se suscribe a los objetos sobre los cuales desea leer o escribir.

Tarjeta de adquisición de datos: El servidor accede a sus funcionalidades (configuración/lectura/escritura) de los puertos.

Diagrama de casos de uso [19] [20]**:**



Flujo de eventos de los casos de uso:

- Obtener configuración de la tarjeta:
  1. El usuario ejecuta el servidor
  2. El servidor accede a las funcionalidades de la tarjeta: número de canales analógicos de entrada.

- Consulta y fija rango de los canales analógicos de entrada
  1. El usuario lee el valor del rango.
  2. El usuario modifica si lo desea el rango del canal analógico.
  3. El servidor fija el nuevo rango en la tarjeta.

- Lee de canales analógicos de entrada:
  1. El servidor lee de la tarjeta el valor del canal analógico de entrada.
  2. El usuario a través del cliente lee del valor leído por el servidor.

- Escribe en canales analógicos de salida
  1. El usuario a través de cliente escribe en el canal analógico de salida
  2. El servidor escribe sobre la tarjeta el valor escrito por el usuario

- Configura puertos digitales.
  1. El usuario a través del cliente escribe sobre la configuración del puerto
  2. El servidor asigna la configuración en la tarjeta

- Escribe en puerto digital
  1. El usuario a través de cliente escribe en el puerto digital
  2. El servidor escribe sobre la tarjeta el valor escrito por el usuario

- Lee de puerto digital
  1. El servidor lee de la tarjeta el valor del canal analógico de entrada.
  2. El usuario a través del cliente lee del valor leído por el servidor.

4.4. Diseño del software

Tras analizar la arquitectura del protocolo OPC UA, las características de la tarjeta 1408FS-Plus, las librerías open62541.org y *Universal library* se procede de la siguiente manera al diseño del software:

- Creación de Tarjeta.h y Tarjeta.c que serán las encargadas del acceso a la tarjeta 1408FS-Plus empleando las funciones de la librería *Universal library*.
- Creación del modelo del servidor siguiendo el modelo de información de OPC UA empleando la librería open62541.org
- Descripción de las funciones empleadas del open62541 para el desarrollo del servidor.
- Creación de server.c que unifica todo lo anterior.

4.4.1. Tarjeta 1408FS-PLUS

Contienen las variables y las funciones utilizadas para el acceso a la tarjeta de adquisición de datos [21].

Universal Library

La siguiente tabla muestra las funciones de la Universal Library que se utilizarán para acceder a la tarjeta 1408FS-Plus.

Ámbito	Función	Prototipo	Descripción
Configuración de la tarjeta	cbGetConfig	int cbGetConfig(int InfoType, int BoardNum, int DevNum, int ConfigItem, int *ConfigVal)	Devuelve una opción de configuración configurada actualmente para una placa. De forma predeterminada, la configuración se carga desde el archivo cb.cfg creado por InstaCal. Si se llama primero a cbIgnoreInstaCal (), los valores de configuración serán los valores predeterminados para la placa en uso.
Configuración de la tarjeta	cbSetConfig	int cbSetConfig(int InfoType, int BoardNum, int DevNum, int ConfigItem, int ConfigVal)	Cambia las opciones de configuración de la placa en el tiempo de ejecución. De forma predeterminada, la configuración se carga desde el archivo cb.cfg creado por InstaCal. Si se llama primero a cbIgnoreInstaCal (), los valores de configuración serán los valores predeterminados para la placa en uso. En cualquier caso, puede cambiar la configuración actual dentro de un programa en ejecución usando esta función.

Entradas Analógicas	cbAIn	int cbAIn(int BoardNum, int Channel, int Range, unsigned short *DataValue);	Lee un canal de entrada A / D de la placa especificada y devuelve un valor entero sin signo de 16 bits. Si la placa A / D especificada tiene ganancia programable, entonces establece la ganancia en el rango especificado. El valor A / D sin formato se convierte a un valor A / D y se devuelve a DataValue.
Salidas Analógicas	cbAOut	int cbAOut(int BoardNum, int Channel, int Range, unsigned short DataValue)	Establece el valor de un canal D / A.
Puertos Digitales	cbDConfigPort	int cbDConfigPort(int BoardNum, int PortType, int Direction)	Configura un puerto digital como entrada o salida.
Puertos Digitales	cbDBitOut	int cbDBitOut(int BoardNum, int PortType, int BitNum, unsigned short BitValue)	Establece el estado de un único bit de salida digital. Esta función trata todos los puertos DIO de un tipo particular en una placa como un solo puerto grande. Le permite establecer el estado de cualquier bit individual dentro de este gran puerto. La mayoría de los puertos configurables requieren configuración antes de escribir.
Puertos Digitales	cbDBitIn	int cbDBitIn(int BoardNum, int PortType, int BitNum, unsigned short *BitValue)	Lee el estado de un solo bit de entrada digital. Esta función trata todos los puertos DIO de un tipo particular en una placa como un solo puerto. Le permite leer el estado de cualquier bit individual dentro de este puerto.
Puertos Digitales	cbDIn	int cbDIn(int BoardNum, int PortType, unsigned short *DataValue)	Lee un puerto de entrada digital. Tenga en cuenta que para algunos tipos de puertos, como los puertos 8255, si el puerto está configurado para DIGITALOUT, esta función proporcionará la lectura del último valor de salida. Consulte el tema "Hardware de entrada / salida digital" para obtener más información sobre el uso de las funciones de E / S digitales de UL.

Puertos Digitales	cbDOut	int cbDOut (int BoardNum, int PortType, unsigned short DataValue)	Escribe un byte en un puerto de salida digital.
Conversión de unidades	cbToEngUnits	int cbToEngUnits(int BoardNum, int Range, unsigned short DataVal, float *EngUnits)	Convierte un valor de recuento de enteros en un valor equivalente de voltaje de precisión simple (o actual). Esta función se usa típicamente para obtener un valor de voltaje de datos recibidos de un A / D con funciones tales como cbAIn ().
Conversión de unidades	cbFromEngUnits	int cbFromEngUnits(int BoardNum, int Range, float EngUnits, unsigned short *DataVal)	Convierte un solo valor de voltaje de precisión (o corriente) en unidades de ingeniería en un valor de conteo entero. Esta función se usa típicamente para obtener un valor de datos de un valor de voltaje para salida a un D / A con funciones tales como cbAOut ().

Tabla 1 Universal Library - Funciones empleadas en este servidor OPC UA

A continuación se muestran las funciones y variables encargadas de las acciones correspondientes a la tarjeta 1408FS-Plus.

Configuración de la tarjeta

- *int configuracion_modo()*
  Retorna el número de canales analógicos de entrada: modo común = 8 y modo diferencial = 4.

Entradas analógicas

- *typedef struct{*
  float value;
  int Range;
  }EntradaAnalogica

  Almacena los atributos de las entradas analógicas.
  value : el valor de la entrada analógica en ese instante.
  Range: el rango de la entrada analógica.
- *EntradaAnalogica In[8]*
  Array de entradas analógicas.
- *void init_entradas()*
  Inicializa las entradas analógicas, asigna los rangos (BIP5VOLTS) a las entradas analógicas.
- *int get_tarjeta_en_uso()*
  Retorna el número asignado a la tarjeta en uso.
- *float get_In_volts(int wChannel)*
  Retorna el valor en voltios del canal analógico wChannel.
- *unsigned int get_In_bin(int wChannel)*
  Retorna el valor binario del canal analógico wChannel.
- *bool selecciona_Range(int channel,float rang)*
  Asigna a rang el formato correspondiente (wConfCode).
- *bool setRange(int wChannel, int wConfCode)*
  Asigna el rango wConfCode al canal analógico wChannel.

Salidas analógicas

- *int Out[2]*
  Último valor escrito en las salidas analógicas.
- *int MaximoNumeroAnalogioOut*
  Máxim número que se puede enviar al canal analógico de salida. Es determinado por la resolución del conversor D/A del canal.
- *void init_salidas()*
  Inicializa los canales analógicos de salida con valor 0.
- *bool set_Out(int wChannel,int wOut)*
  Asigna el valor wOut al canal de salida wChannel.
- *bool set_Out_Volts(int wChannel, float voltios)*

Asigna el valor voltios al canal de salida wChannel.

- *int get_Out(int wChannel)*
  Retorna el valor del canal analógico de salida wChannel.
- *get_Out_Volts(int wChannel)*
  Retorna el valor en voltios del canal analógico de salida wChannel.

Puertos digitales

- *int modoA:*
  Número que almacena el modo de configuración del puerto digital A de la tarjeta en uso (entrada/salida).
- *int modoB:*
  Ídem del anterior referido al puerto digital B.

- *int init_digitales();*
  Inicializa los puertos digitales configurandolos en modo entrada.
- *void set_digmode_AIN();*
  Configura el puerto digital A en modo entrada
- *void set_digmode_AOUT();*
  Configura el puerto digital A en modo salida
- *void set_digmode_BIN();*
  Configura el puerto digital B en modo entrada
- *void set_digmode_BOUT();*
  Configura el puerto digital B en modo salida
- *int get_digmode_a();*
  Devuelve la configuración del puerto digital A
- *int get_digmode_b();*
  Devuelve la configuración del puerto digital B
- *bool set_bit_Do_A(int bit,int value);*
  Estable el bit del puerto digital A a 0 o 1.
- *bool set_bit_Do_B(int bit,int value);*
  Establece el bit del puerto digital B a 0 o 1.
- *int get_bit_Di_A(unsigned int bit);*
  Devuelve el valor del bit del puerto digital A.
- *int get_bit_Di_B(unsigned int bit);*
  Devuelve el valor del bit del puerto digital B.
- *int get_Di();*
  Devuelve el valor puerto digital A.

4.4.2. Servidor OPC UA

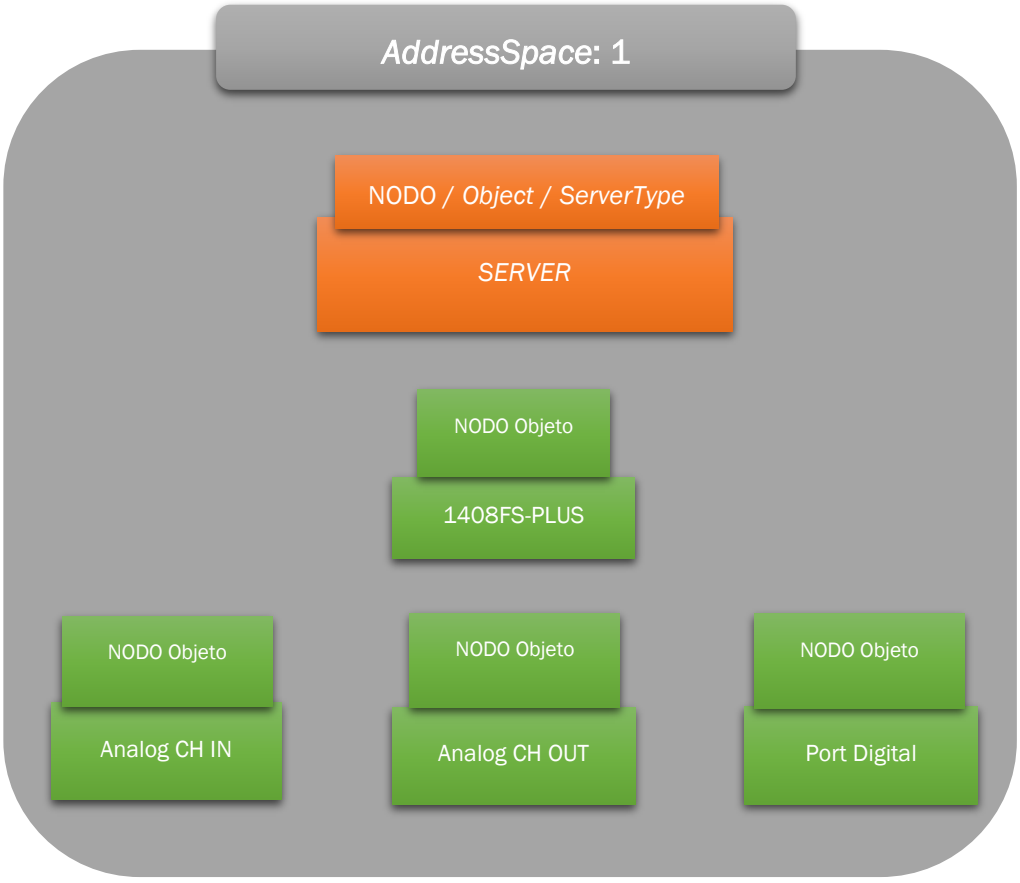
Este apartado muestra la estructura del servidor según el modelo de información de OPC UA empleando la librería Open62541.org. Describiendo el espacio de direcciones, los nodos objetos y los nodos variables; específicos para este proyecto.

AddressSpace

El AddressSpace proporciona al servidor un estándar para representar objetos a los clientes.

Los servicios UA son utilizados para acceder a estos objetos y sus componentes, además los elementos de este modelo son representados en el espacio de direcciones como Nodos.

La siguiente imagen muestra el espacio de direcciones que contiene los objetos server, 1408FS-Plus, Analog CH IN, Analog CH OUT y PORT Digital.



Nodos Objetos

Los objetos se utilizan para representar sistemas, componentes del sistema, objetos del mundo real y objetos de software. Los objetos son instancias de un tipo de objeto y pueden contener variables, métodos y otros objetos. Estos objetos son representados como nodos que son descritos por atributos e interconectados por referencias.

ObjectType proporciona definiciones para objetos. *ObjectTypes* se definen utilizando *ObjectType NodeClass*,. Las referencias *HasComponent* identifican las *DataVariables*, los métodos y los objetos contenidos en el *ObjectType*.

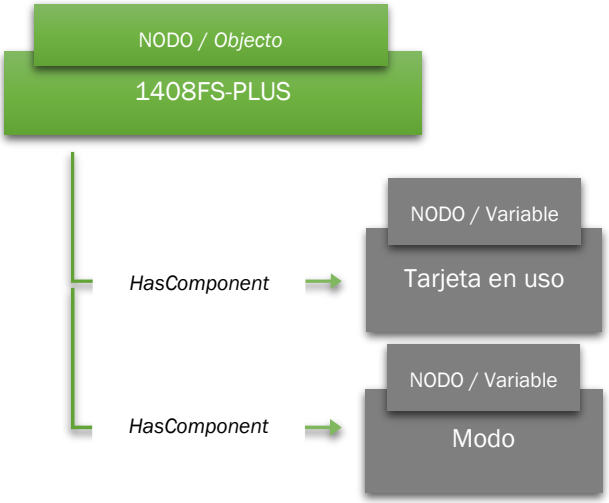
ServerType: El objeto del servidor es, como casi todos los objetos UA de OPC, una instancia de una definición de tipo. En su caso, su definición de tipo es el *ServerType* que es en sí mismo, un subtipo del *BaseObjectType*. Este *ObjectType* define las capacidades admitidas por el servidor OPC UA.

Los nodos Servidor y Tarjeta 1408FS-Plus son nodos de clase *Object*, Los nodos "Tarjeta en uso" y "modo" son nodos de clase Variable, además son componentes del nodo Object 1408FS-Plus.

- Servidor

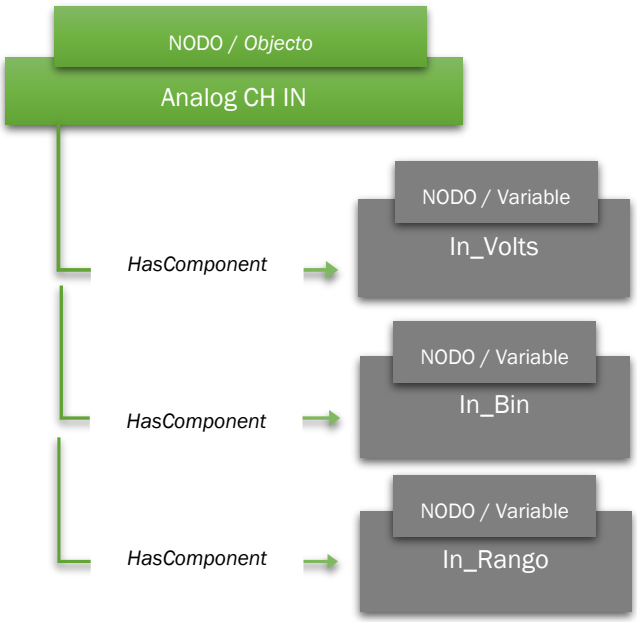
- Tarjeta 1408FS –PLUS

El objeto Tarjeta 1408FS-Plus representa a la tarjeta en uso, se usa para contener las variables necesarias para obtener la información y configuración de la tarjeta.



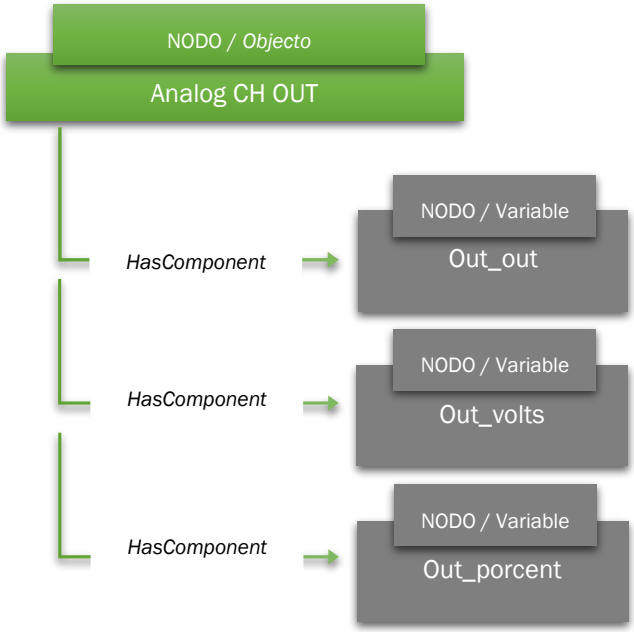
- Entradas analógicas

El objeto Analog CH IN representa las entradas analógicas de la tarjeta, tiene como componentes a los nodos de clase Variable In_Volts, In_Bin e In_Rango.



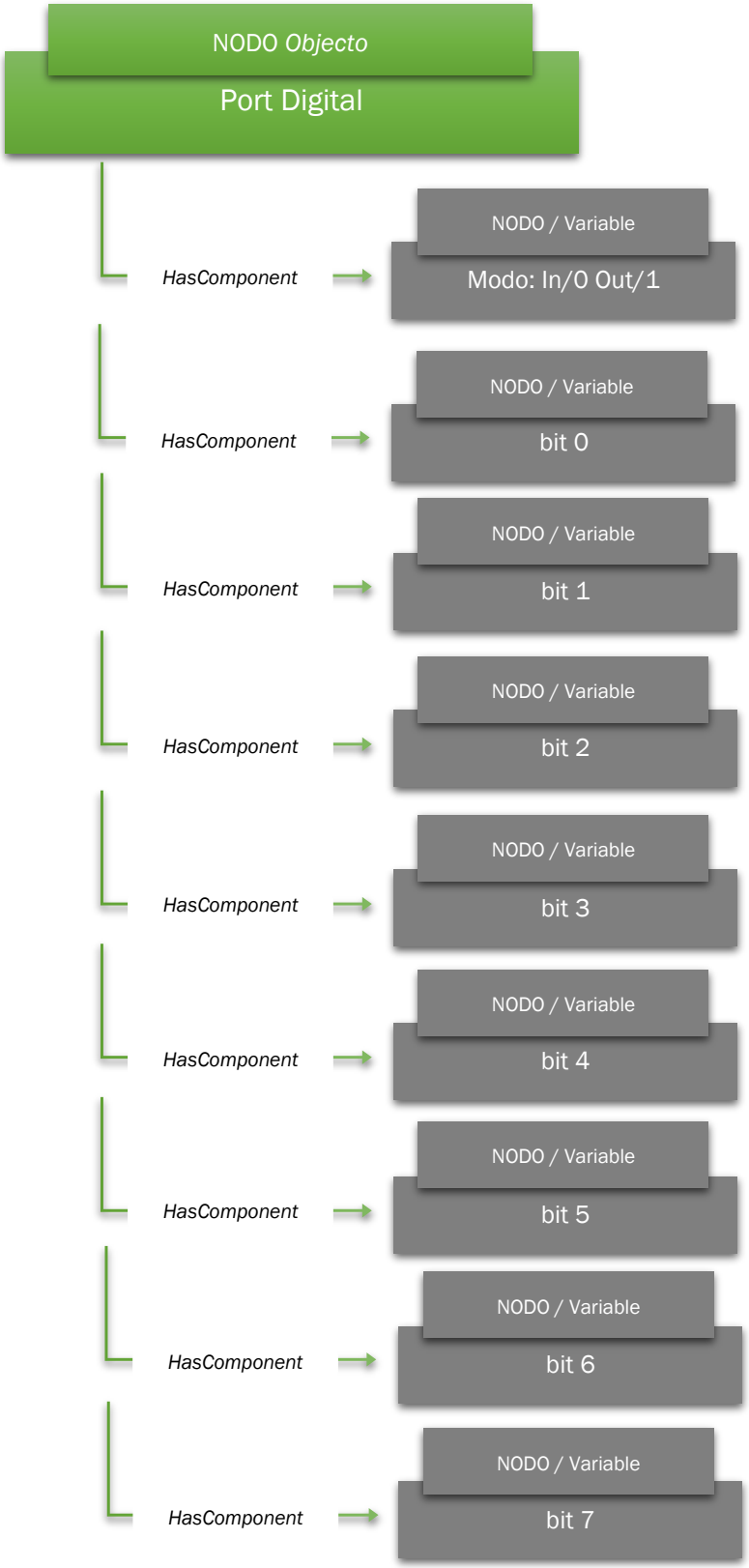
- Salidas analógicas

El objeto Analog CH OUT representa las salidas analógicas de la tarjeta tiene como componentes a los nodos de clase Variable Out_out, Out_Volts y Out_porcent.



- Puerto digital

El objeto “Port Digital” representa a los puertos digitales de la tarjeta, este nodo contiene las variables Modo y bit [0…7].



Nodos Variables

Variable *Value Callback*

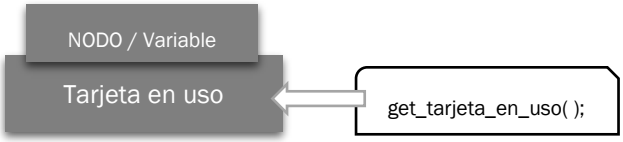
Cuando un valor cambia continuamente, como la hora del sistema, actualizar el valor en un bucle cerrado requeriría una gran cantidad de recursos. Las devoluciones de valor (*Value Callback*) permiten sincronizar un valor variable con una representación externa. Ellos asocian devoluciones de llamada a la variable que se ejecutan antes de cada lectura (*onRead*) y después de cada operación de escritura (*onWrite*) [22].

onRead: Función que es llamada cada vez que la función vinculada a esta cambia su valor de devolución.

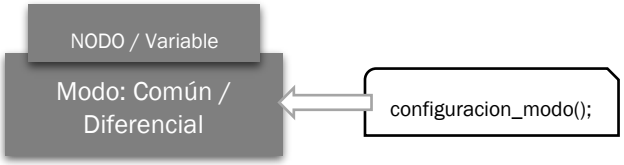
onWrite: Función que es llamada cada vez que se escribe en un nodo variable.

- Tarjeta 1408FS-Plus

- Contiene el número asociado con la tarjeta cuando se instaló con *Instacal* o se creó con cbCreateDaqDevice, puede tomar una valor entre 0 y 99. Este valor es necesario para acceder a las funciones de configuración en la Universal Library.

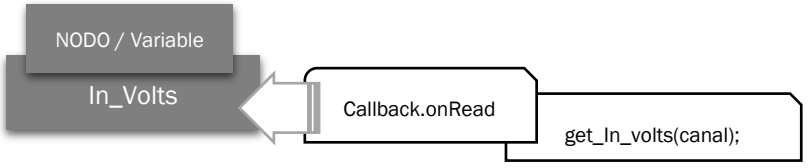


- Contiene la configuración de la tarjeta, ya se Común=8 entradas analógicas) o diferencial =4 entradas analógicas.

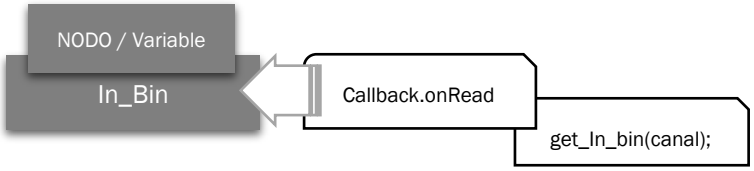


- Entradas analógicas

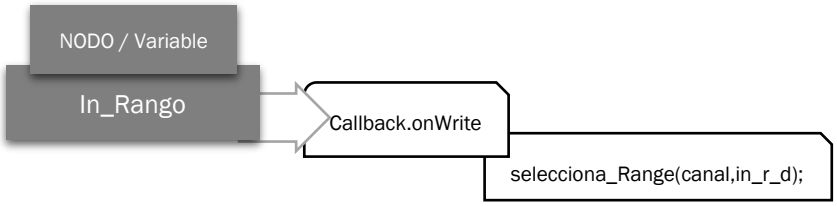
- Contiene el valor en voltios de la entrada analógica.



- Contiene el valor en binario de la entrada analógica.

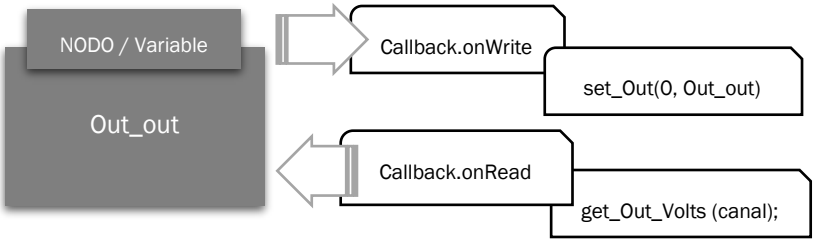


- Contiene el valor del rango de la entrada analógica. El usuario podrá escribir en esta variable.

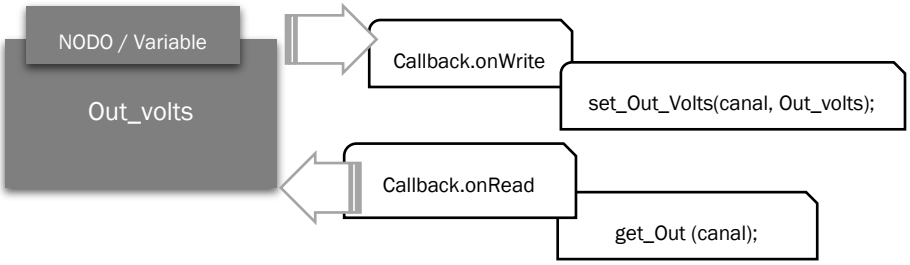


  - Salidas analógicas

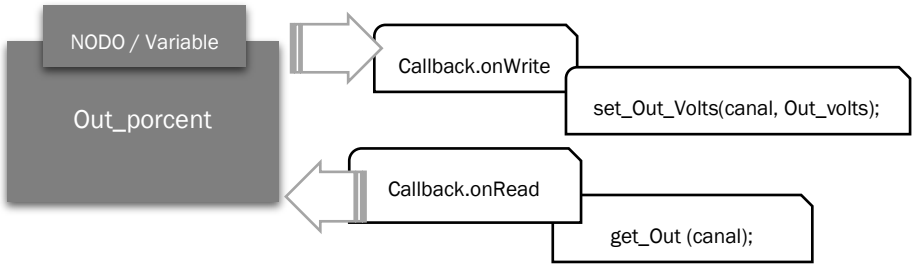
- El usuario escribirá el valor en binario de la salida analógica en Out_out. Al escribir en esta variable se actualizar el valor del nodo Out_volts y Out_porcent a través de *onRead*,



- El usuario podrá escribir el valor en voltios de la salida analógica en Out_Volts. Al escribir en esta variable se actualizar el valor del nodo Out_out y Out_porcent a través de *onRead*,

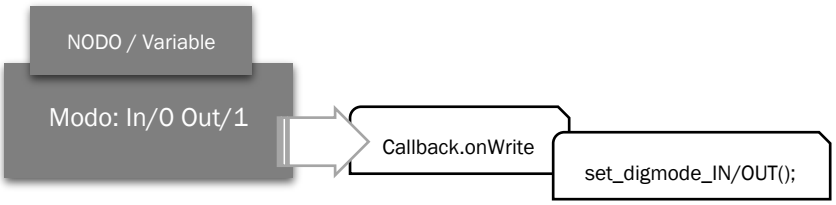


- El usuario podrá escribir el valor en porcentaje de la salida analógica en Out_porcent. Al escribir en esta variable se actualizar el valor del nodo Out_out y Out_volts a través de *onRead*,

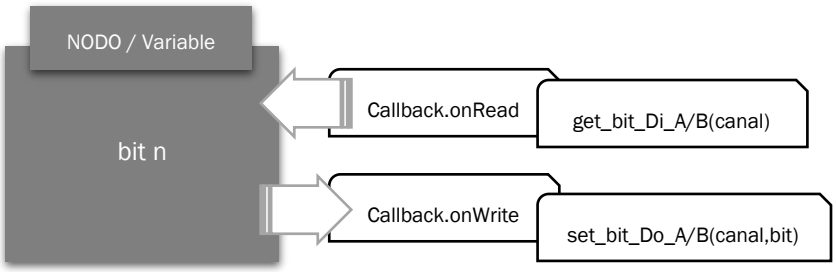


  - Puertos digitales

- Según el valor escrito en esta variable los puertos digitales funcionará en modo entrada o salida.



- Variable que representa cada bit del puerto digital. El usuario podrá escribir en caso de que el puerto se encuentre en modo salida, o leer si se encuentra en modo entrada.



4.4.3. Diagramas de secuencia

Los diagramas de secuencia se utilizan para representar la interacción entre los objetos de un sistema, en este caso los objetos del sistema son representados por el Cliente OPC UA, el Servidor OPC UA y la tarjeta USB-1408fs-Plus. [23] [24] [25]

- Inicialización del Servidor

La inicialización de la tarjeta consiste en asignar el rango a los canales de entrada, que por defecto será BIP5VOLTS (5 voltios bipolares). Se configura los puertos digitales en modo entrada, y además es necesario conocer el modo en el que este configuradas las entradas analógicas (común/diferencial).

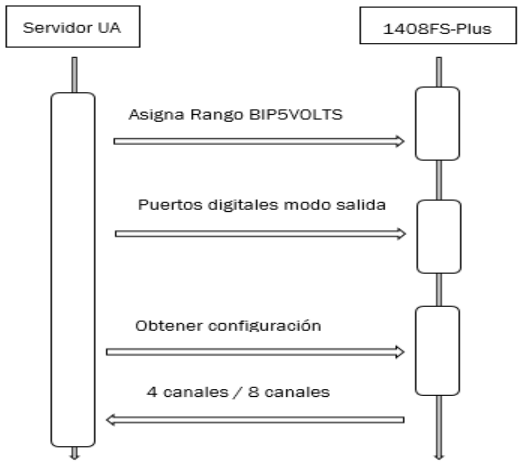


Figura 14 Diagrama de secuencias - Inicialización del servidor

- Obtención de los valores de las entradas analógicas

Según la configuración de las entradas analógicas el servidor creará 4 u 8 canales analógicos de entrada. A partir de ese momento el cliente podrá suscribirse a cada una de las variables de cada uno de los canales analógicos, con lo que el servidor empleando el *onRead* notificará al cliente ante cada cambio que se realice en el valor de la variable a la cual se ha suscrito.

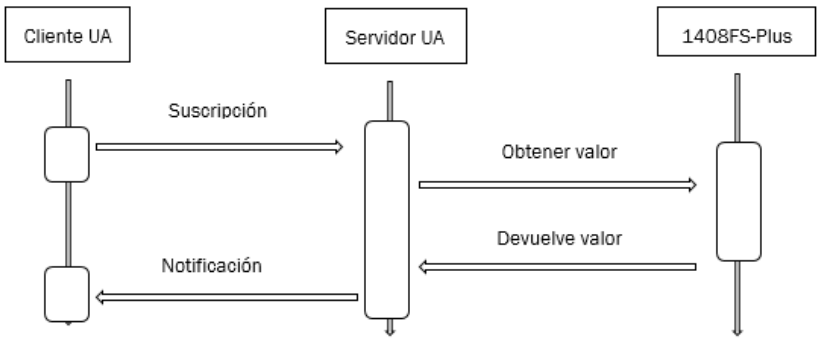


Figura 15 Diagrama de secuencias - Entradas analógicas

- Asignación de los valores a los canales analógicos de salida

Al suscribirse el cliente a una variable, puede escribir en el nodo variable, posteriormente el servidor actualiza el valor en la tarjeta empleando el onWrite. Ya que onWrite es una función que se llama cada vez que se escribe en el nodo vinculado a ella. Por lo tanto, cada vez que el usuario escriba en la variable, el servidor llamara a onWrite que contiene la función de la librería Tarjeta.h que permite escribir en el canal determinado.

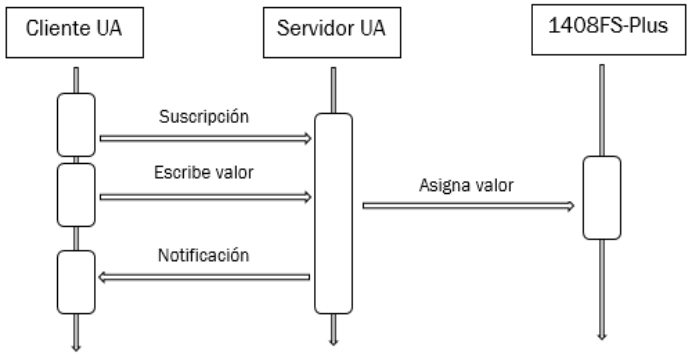


Figura 16 Diagrama de secuencias - Salidas analógicas

En este caso al escribir sobre una variable también es necesario actualizar el valor de la otra, por ejemplo; al escribir un valor en la variable Out_volts, es necesario que el servidor actualice el valor de la variable Out_out y Out_porcent. Todo el proceso es el siguiente:

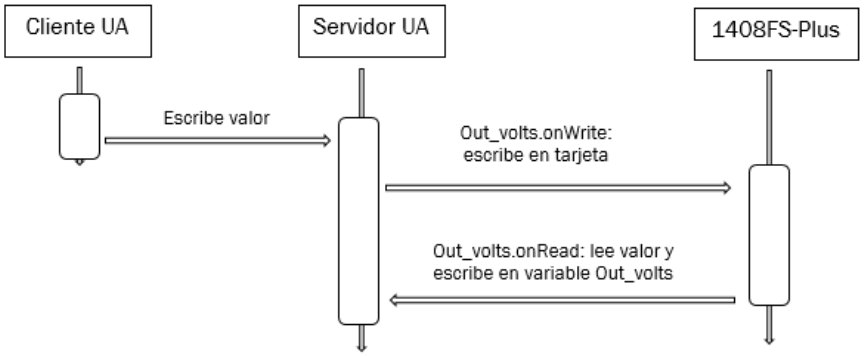


Figura 17 Diagrama de secuencias - Salidas analógicas actualización de nodos variables

- El cliente escribe en Out_volts
- onWrite de Out_volts actualiza el valor en la tarjeta
- el servidor llama al onRead de Out_volts que detecta un cambio en el valor escrito en la tarjeta, y este, a su vez al detectar el cambio en el valor escribe sobre la variable Out_out y Out_porcent.

El modelo de información de OPC UA no ha permitido escribir directamente sobre un nodo empleando un onWrite de otro nodo, razón por la cual se ha tenido que utilizar el onRead.

- Puertos digitales

La configuración inicial de los puertos digitales es en modo Salida, al suscribirse a la variable "modo" se podrá modificar esta configuración, escribiendo 1 si se desea modo salida o 0 si se desea que el funcionamiento sea en modo entrada.

El cliente se podrá suscribir a cualquier bit o a todos los bits de cualquiera de los dos puertos.

Si se encuentra en modo salida el servidor empleará el onWrite de la variable de determinado bit, si se encuentra en modo entrada empleará el onRead.

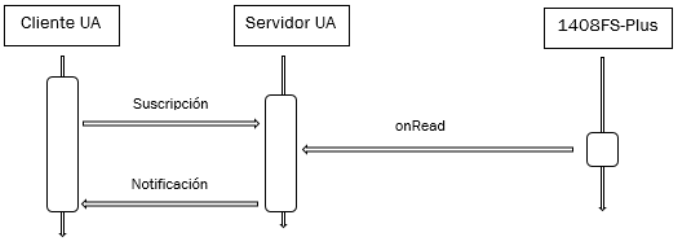


Figura 18 Diagrama de secuencias - Puerto digital modo entrada

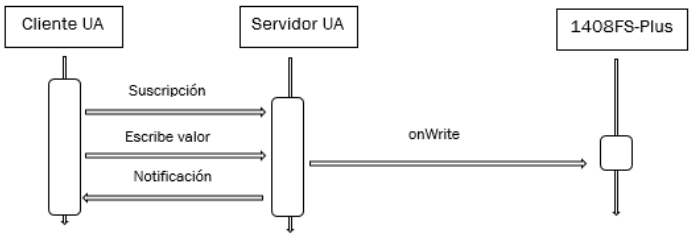


Figura 19 Diagrama de secuencias - Puerto digital modo salida

4.4.4. Open62541.org

A continuación se muestran las **estructuras** y **tipos de datos** de la librería open62541 usadas para la creación de este servidor OPC UA y sus componentes [26]…[34].

Ámbito	Estructura/Tipo de Dato	Descripción
ServerType	UA_ServerConfig	Contiene la configuración del servidor OPC UA
ServerType	UA_Server	Servidor UA
ServerType	UA_StatusCode	Un valor numérico que se utiliza para informar el resultado de una operación realizada por un servidor OPC UA.
Node	UA_NodeId	Un identificador para un nodo en el espacio de direcciones de un servidor OPC UA
Node	UA_ObjectAttributes	Los atributos para un nodo objeto
Node	UA_VariableAttributes	Los atributos para un nodo variable
DataType	UA_UInt16	Un valor entero entre 0 y 65535
DataType	UA_Int32	Un valor entero entre -2147483648 y 2147483647
DataType	UA_Float	Un valor de coma flotante de precisión simple IEEE (32 bit)
DataType	UA_Variant	Las *variants* pueden contener valores de cualquier tipo junto con una descripción del contenido. Las variantes pueden contener un valor escalar o una matriz.
Callback	UA_ValueCallback	Los valeCallback permiten sincronizar un valor variable con una representación externa.

Tabla 2 Open62541 - Estructuras y tipos de datos empleados en este servidor OPC UA

A continuación se muestran las **funciones** de la librería open62541 usadas para la creación del servidor y sus componentes.

Ámbito	Funciones	Descripción
ServerType	UA_ServerConfig_new_minimal	Asigna la configuración del servidor OPC UA.
ServerType	UA_Server_new	Crear un nuevo servidor OPC UA.
ServerType	UA_Server_run	Inicia un servidor OPC UA.
ServerType	UA_Server_Delete	Elimina un servidor OPC UA.
ServerType	UA_ServerConfig_delete	Elimina la configuración de un servidor OPC UA.
Node	UA_NODEID_STRING	Genera un NODEID de tipo string.
Node	UA_LOCALIZEDTEXT	Genera texto legible con un identificador opcional.
Node	UA_NODEID_NUMERIC	Genera un NODEID de tipo numérico.
Node	UA_QUALIFIEDNAME	Un nombre calificado por un espacio de nombres.
Node	UA_VariableAttributes_default	Genera atributos por defecto de un nodo variable.
Node	UA_Server_addObjectNode	Añade un nodo objeto.
Node	UA_Server_addVariableNode	Añade un nodo variable
Node	UA_Variant_setScalar	Asigna un escalar a un tipo de dato *variant.*
Node	UA_Server_readValue	Lee del valor del atributo de un nodo.
Node	UA_Server_writeValue	Escribe en el valor del atributo de un nodo.
Callback	onRead	Es llamado antes de leer del valor del atributo de un nodo.
Callback	onWrite	Es llamado después de leer del valor del atributo de un nodo.

Tabla 3 Open62541 - Funciones empleadas en este servidor OPC UA

4.4.5. Server.c

En este apartado se describen las funciones creadas para el server.c, agrupadas según los *objects* del servidor.

USB-1408FS-PLUS

- *static void define_tarjeta (UA_Server *server, char* tarjeta)*
  Añade el objeto "1408FS-Plus" al *Addresspace* del servidor.

- *static void add_config_modo(UA_Server *server,char *tarjeta)*
  Añade el objeto/variable "Modo común/diferencial" al *Addresspace* del servidor.

- *int read_config_modo(UA_Server *server)*
  Devuelve el valor del objeto/variable "Modo común/diferencial".

Analog CH IN [0...7]

- *static void Define_object_entradas_analogicas(UA_Server *server, char * id_analogic_in)*
  Añade un objeto "ANALOG CH IN n" al *Addresspace* del servidor.

- *static void add_In_Volts_Variable (UA_Server * server,char * id_analogic_in ,char* id_volts)*
  Añade el objeto/variable "In_Volts" al *Addresspace* del servidor.

- *void read_in_volts(UA_Server *server,char *id_nodo,int canal)*
  Escribe en el objeto/variable el valor en voltios del canal de entrada.

- *static void beforeRead_In_Volts_CH_0 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onRead del objeto/variable "In_Volts".

- *static void addValueCallbackTo_in_volts(UA_Server *server, char *id_node,int canal)*
  Añade el ValueCallback del objeto/variable "In_Volts".

- *static void add_In_Bin_Variable (UA_Server * server,char * id_analogic_in ,char* id_bin)*
  Añade el objeto/variable "In_Bin" al *Addresspace* del servidor.

- *void read_in_bin(UA_Server *server,char *id_nodo,int canal)*
  Escribe en el objeto/variable el valor en binario del canal de entrada "In_Bin".

- *static void beforeRead_In_Bin_CH_[0…7] (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onRead del objeto/variable "In_Bin".

- *static void addValueCallbackTo_in_bin(UA_Server *server,char *id_nodo,int canal)*
  Añade el ValueCallback del objeto/variable "In_Bin".

- *static void add_In_Rango_Variable (UA_Server * server,char * id_analogic_in ,char* id_rango)*
  Añade el objeto/variable "In_rango" al *Addresspace* del servidor.

- *void after_write_in_rango(UA_Server *server, char *id_node , int canal)*
  Actualiza el rango.

- *static void afterWrite_In_rango_CH_0 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onWrite del objeto/variable "In_rango".

- *static void addValueCallbackTo_in_rango(UA_Server *server,char *id_nodo,int canal)*
  Añade el ValueCallback del objeto/variable "In_rango".

Analog CH OUT [0…1]

- *static void Define_object__salidas_analogicas(UA_Server *server, char* id_analogic_out)*
  Añade el objeto "Analog CH OUT n" al *Addresspace* del servidor.

Out_out

- *static void add_Out_out_Variable (UA_Server * server,char * id_OUT ,char*id_Out_out)*
  Añade el objeto/variable "Out out" al *Addresspace* del servidor.

- *void write_Out_out(UA_Server *server,char *id_node_out,int canal)*
  Escribe en el objeto/variable "Out_out" el valor en binario del canal analógico de salida.

- *static void beforeRead_Out_out_CH_0 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onRead del objeto/variable "Out_out" del canal de salida 0.

- *static void beforeRead_Out_out_CH_1 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onRead del objeto/variable "Out_out" del canal de salida 1.

- *static void afterWrite_Out_out_CH_0 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onWrite del objeto/variable "Out_out" del canal de salida 0.

- *static void afterWrite_Out_out_CH_1 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onWrite del objeto/variable "Out_out" del canal de salida 0.

- *static void addValueCallbackTo_Out_out(UA_Server *server,char *id_nodo, int canal)*
  Añade el Value Callback del objeto/variable "Out_out".

Out_volts

- *static void add_Out_volts_Variable (UA_Server *server, char * id_OUT ,char* id_Out_volts)*
  Añade el objeto/variable "Out volts" al *Addresspace* del servidor.

- *void write_Out_volts(UA_Server *server,char *id_node_volts,int canal)*

Escribe en el objeto/variable “Out_volts” el valor en volts del canal analógico de salida.

- *static void beforeRead_Out_volts_CH_0 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onRead del objeto/variable “Out_volts” del canal analógico de salida 0.

- *static void beforeRead_Out_volts_CH_1 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onRead del objeto/variable “Out_volts” del canal analógico de salida 1.

- *static void afterWrite_Out_volts_CH_0 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onWrite del objeto/variable “Out_volts” del canal de salida 0.

- *static void afterWrite_Out_volts_CH_1 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onWrite del objeto/variable “Out_volts” del canal de salida 1.

- *static void addValueCallbackTo_Out_volts(UA_Server *server,char *id_nodo, int canal)*
  Añade el Value Callback del objeto/variable “Out_volts”.

Out_porcent

- *static void add_Out_porcent_Variable (UA_Server *server, char * id_OUT ,char* id_Out_ porcent)*
  Añade el objeto/variable “Out porcent” al *Addresspace* del servidor.

- *void write_Out_ porcent (UA_Server *server,char *id_node_ porcent,int canal)*
  Escribe en el objeto/variable “Out_porcent” el valor en volts del canal analógico de salida.

- *static void beforeRead_Out_porcent_CH_0 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onRead del objeto/variable "Out_porcent" del canal analógico de salida 0.

- *static void beforeRead_Out_porcent_CH_1 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onRead del objeto/variable "Out_porcent" del canal analógico de salida 1.

- *static void afterWrite_Out_porcent_CH_0 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onWrite del objeto/variable "Out_porcent" del canal de salida 0.

- *static void afterWrite_Out_porcent_CH_1 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onWrite del objeto/variable "Out_porcent" del canal de salida 1.

- *static void addValueCallbackTo_Out_porcent(UA_Server *server,char *id_nodo, int canal)*
  Añade el Value Callback del objeto/variable "Out_porcent".

Port Digital A

- *static void add_dig_bit(UA_Server *server,char *Port_a,char * dig_bit_id)*
  Añade el objeto/variable "bit" al *Addresspace* del servidor.

- *static void Define_Port_dig_a (UA_Server *server, char* id_Port)*
  Añade el objeto "Port Digital A" al *Addresspace* del servidor.

- *static void add_Port_a_config(UA_Server *server,char *Port_a)*

Añade el objeto/variable "Modo: In/0 Out/1" componente del objeto "Port Digital A" al *Addresspace* del servidor.

- *static void afterWrite_config_a (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onWrite del objeto/variable "Modo: In/0 Out/1" componente del objeto "Port Digital A".

- *static void addValueCallbackTo_config_a_Variable (UA_Server * server)*
  Añade el Value Callback del objeto/variable "Modo: In/0 Out/1" componente del objeto "Port Digital A".

A bit

- *static void beforeRead_dig_bit_A_0 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añadel el onRead del objeto/variable "bit" componente del objeto "Port Digital A".

- *static void afterWrite_dig_bit_A_0 (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añadel el onWrite del objeto/variable "bit" componente del objeto "Port Digital A".

- *static void addValueCallbackTo_dig_bit_A (UA_Server * server,char * id_node_bit_a,int canal)*
  Añadel el Value Callback del objeto/variable "bit" componente del objeto "Port Digital A".

Port Digital B

- *static void add_Port_b_config(UA_Server *server,char *Port_a)*
  Añade el objeto/variable "Modo: In/0 Out/1" componente del objeto "Port Digital B" al *Addresspace* del servidor.

- *static void afterWrite_config_b (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añade el onWrite del objeto/variable "Modo: In/0 Out/1" componente del objeto "Port Digital B".

- *static void addValueCallbackTo_config_b_Variable (UA_Server * server)*
  Añade el Value Callback del objeto/variable "Modo: In/0 Out/1" componente del objeto "Port Digital B".

B bit

- *static void beforeRead_dig_bit_B_[0...7] (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añadel el onRead del objeto/variable "bit" componente del objeto "Port Digital B".

- *static void afterWrite_dig_bit_B_[0...7] (UA_Server * server ,const UA_NodeId * sessionId , void * sessionContext ,const UA_NodeId * nodeid , void * nodeContext ,const UA_NumericRange * range , const UA_DataValue * data)*
  Añadel el onWrite del objeto/variable "bit" componente del objeto "Port Digital B".

- *static void addValueCallbackTo_dig_bit_B (UA_Server * server,char * id_node_bit_b, int canal)*
  Añadel el Value Callback del objeto/variable "bit" componente del objeto "Port Digital B"

Server

El servidor se inicia con UA_Server_run. Internamente, el servidor utiliza tiempos de espera para programar tareas regulares. Entre los tiempos de espera, el servidor escucha en la capa de red los mensajes entrantes.

Para saber cuándo el servidor deja de funcionar se crea una variable global en ejecución:

- UA_Boolean running = true;

Además, se registra el método *stopHandler* que capta la señal (interrupción) que recibe el programa cuando los sistemas operativos intentan cerrarlo.

- static void stopHandler(int sig)

Esto sucede, por ejemplo, cuando presiona ctrl-c en un programa de terminal. El manejador de señal luego establece que la variable *running* se ejecute en falso y el servidor se apaga una vez que retoma el control.

- UA_ServerConfig_new_minimal()

Esta función devuelve la configuración del servidor, y tiene como argumentos el puerto de escucha y un certificado (opcional). Este servidor está configurado para escuchar en el puerto **1408**.

- *int main(void)*

Esta función administra el ciclo de vida del servidor: crea un servidor, ejecuta el servidor y elimina el servidor.

Otras funciones

Estas funciones son llamadas dentro de la función main (void), y estas funciones contienen a las funciones encargadas de añadir los nodos, vistas en el apartado anterior a ***server***.

- void Diferencial_mode(UA_Server *server)
  Añade 4 canales analógicos de entrada junto con sus variables, 2 canales analógicos de salida junto con sus variables.
- void Comun_mode(UA_Server *server)
  Añade otros 4 canales analógicos de entrada junto con sus variables.
- static void add_Ports_digitales(UA_Server *server)
  Añade los dos puertos digitales, A y B.

La asignación del nombre del servidor se ha realizado a través de la definición que viene por defecto en la librería open62541.org:

*#define APPLICATION_NAME "**open62541-based OPC UA Application**"*

Esta asignación por defecto se ha modificado para que el servidor sea:

*#define APPLICATION_NAME "**Servidor OPC UA para USB-1408FS-Plus**"*

5. Instalación de la tarjeta y del servidor

5.1 Instalación de la tarjeta

Previa a la instalación de la tarjeta se debe de disponer de la aplicación *InstaCal* para poder instalar y configurar las tarjetas de adquisición de datos conectadas al equipo. *InstaCal* forma parte del software Universal Library.

El primer paso es conectar la tarjeta al puerto USB, el led verde se encenderá indicando que es alimentada. Al enviar o recibir datos el led se encenderá intermitentemente.

Luego de abrir *instaCal* el siguiente paso es detectar la tarjeta.

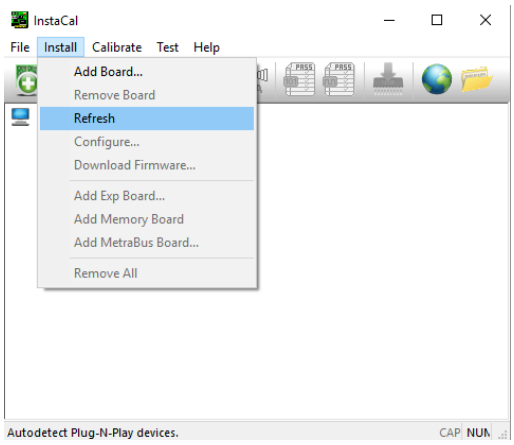


Figura 20 Instalación de la tarjeta – Instacal – Detecta tarjeta

Una vez detectada la tarjeta, en el caso en el que haya mas de una se deberá elegir la USB-1408FS-Plus.

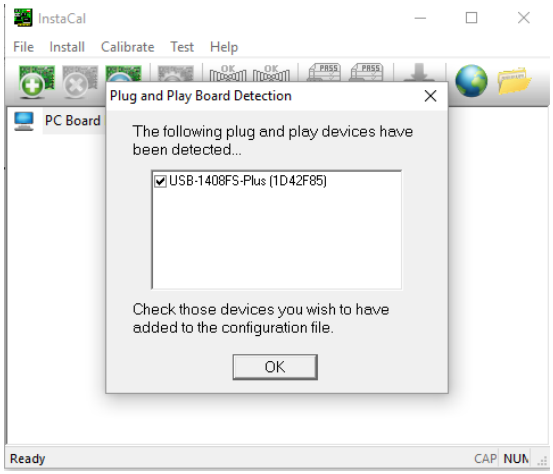


Figura 21 Instalación de la tarjeta – Instacal – Selecciona tarjeta

InstaCal asignará por defecto "0" para la tarjeta seleccionada.

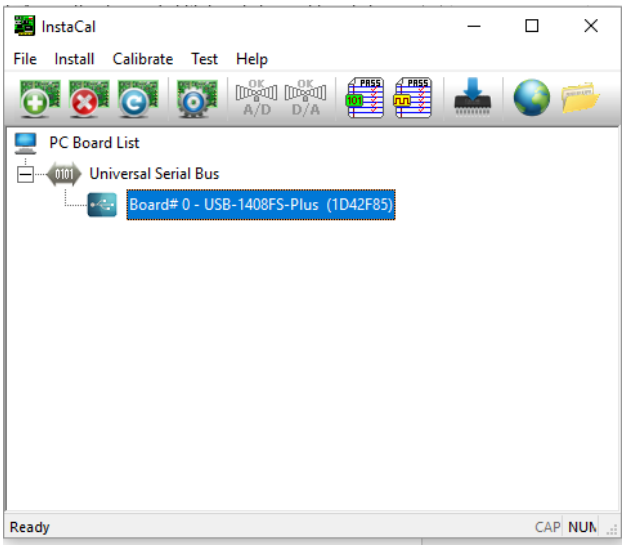


Figura 22 Instalación de la tarjeta – *Instacal* – Número de tarjeta

Al hacer clic derecho sobre la tarjeta se podrá observar las siguientes opciones:

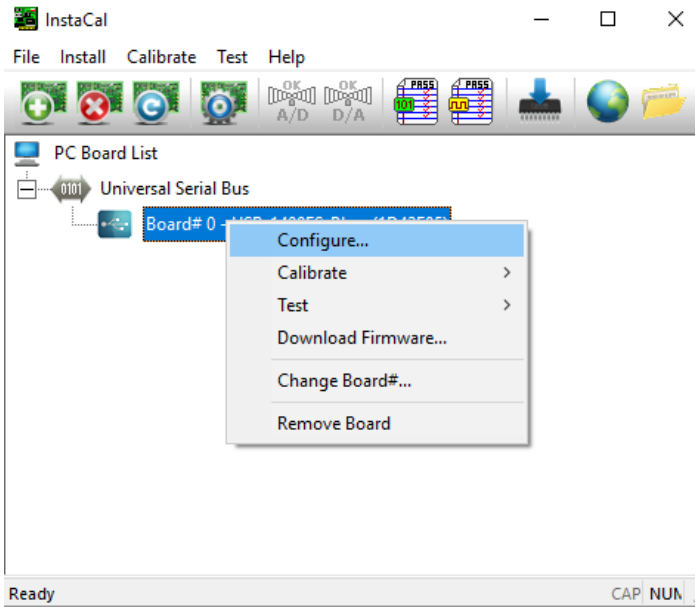


Figura 23 Instalación de la tarjeta – *Instacal* - Configuración

Si seleccionamos configuración se puede observar que se podrá elegir entre modo diferencial y modo común.

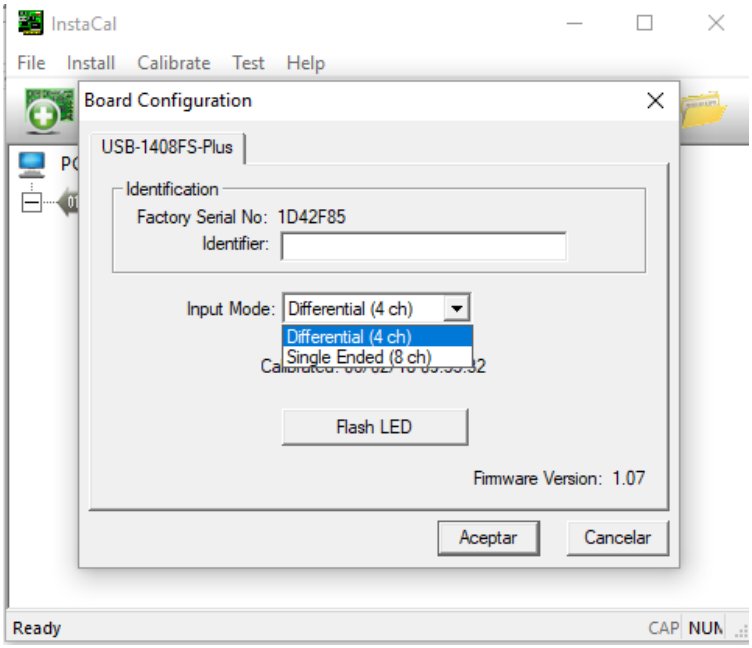


Figura 24 Instalación de la tarjeta – Instacal – Modo diferencial/Común

InstaCal te permitirá cambiar el número (número de referencia en *instaCal*) de la tarjeta instalada.

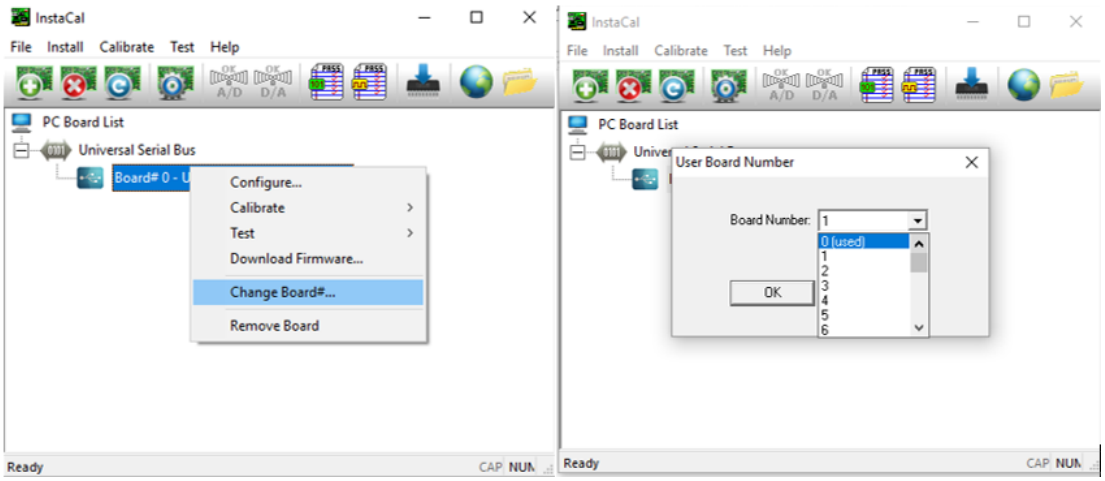


Figura 25 Instalación de la tarjeta – Instacal – Cambia número de tarjeta

5.2. Instalación del servidor

Para la creación del instalador se ha empleado el software NSIS [35] que es un sistema de código abierto profesional para crear instaladores de Windows.

La instalación de la aplicación está automatizada, al ejecutar el fichero de instalación se inicia el proceso. Además se incluye el desinstalador de la aplicación.

Figura 26 Instalador de la aplicación

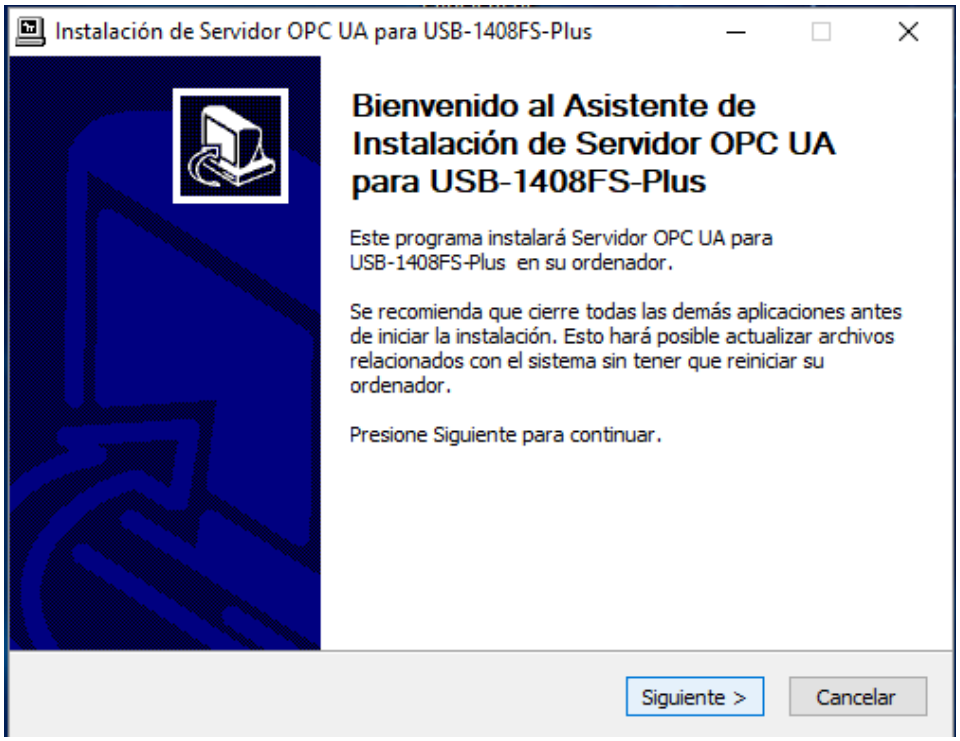


Figura 27 Ejecución del fichero de instalación

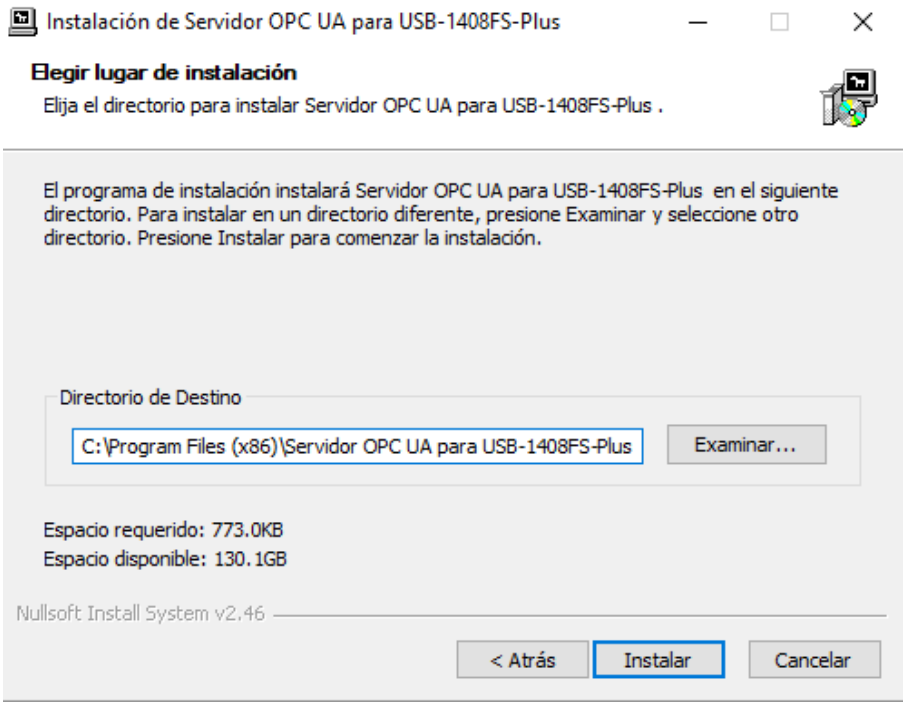


Figura 28 Instalador - Directorio de destino

En el directorio de destino se abra creado el siguiente ejecutable:

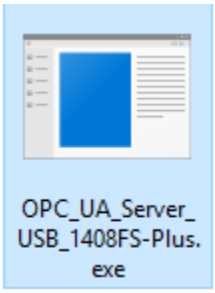


Figura 29 Ejecutable del servidor

La siguiente imagen muestra al servidor en ejecución y en escucha en el puerto 1408.

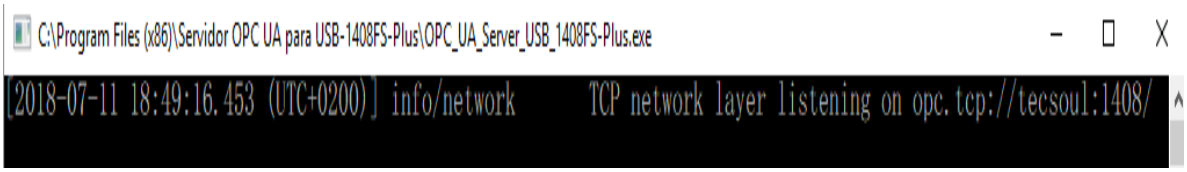


Figura 30 Servidor OPC UA en espera de conexiones

6. Implementación y validación

Las pruebas del funcionamiento de servidor se han realizado empleando el Client *dataFeed* OPC UA [36] ofrecido gratuitamente por la empresa Softing [37].

6.1. Cliente *dataFEED* OPC UA

El *dataFEED* OPC UA *Client* gratuito muestra las posibilidades del kit de desarrollo OPC UA de la empresa softing.

Sus principales características aparecen en la siguiente tabla.

OPC Specifications	OPC *Unified Architecture* V1.03
OPC UA Client	*Core Characteristics, Data Access, Complex Data, Base Eventing, Methods, Audit, Node Management, Historical Access, Alarms and Conditions, Redundancy*
OPC UA Transport	*OPC UA TCP transport; UA Binary Encoding* *SOAP/HTTP transport: UA Binary and XML encoding* *HTTPS transport; UA Binary and XML encoding*
OPC UA Security	*Security policy: Basic256Sha256/Basic256/Basic128Rsa15/None* *Authentication: anonymous/user name and password/user certificate* *Full compliance to OPC UA 1.03 specification*
Operating Systems	*Windows 7 (32 Bit and 64 Bit), Windows 8.1 (32 Bit and 64 Bit), Windows 10 (32 Bit and 64 Bit), Windows Server 2008 RS2, Windows Server 2012, Windows Server 2016*
Compliance/Certification	*Regularly tested at OPC Foundation interoperability workshops, test with OPC Foundation compliance test tool*

Tabla 4 Softing - Cliente OPC UA - Características

- Interfaz

La interfaz se puede dividir en 5 bloques:

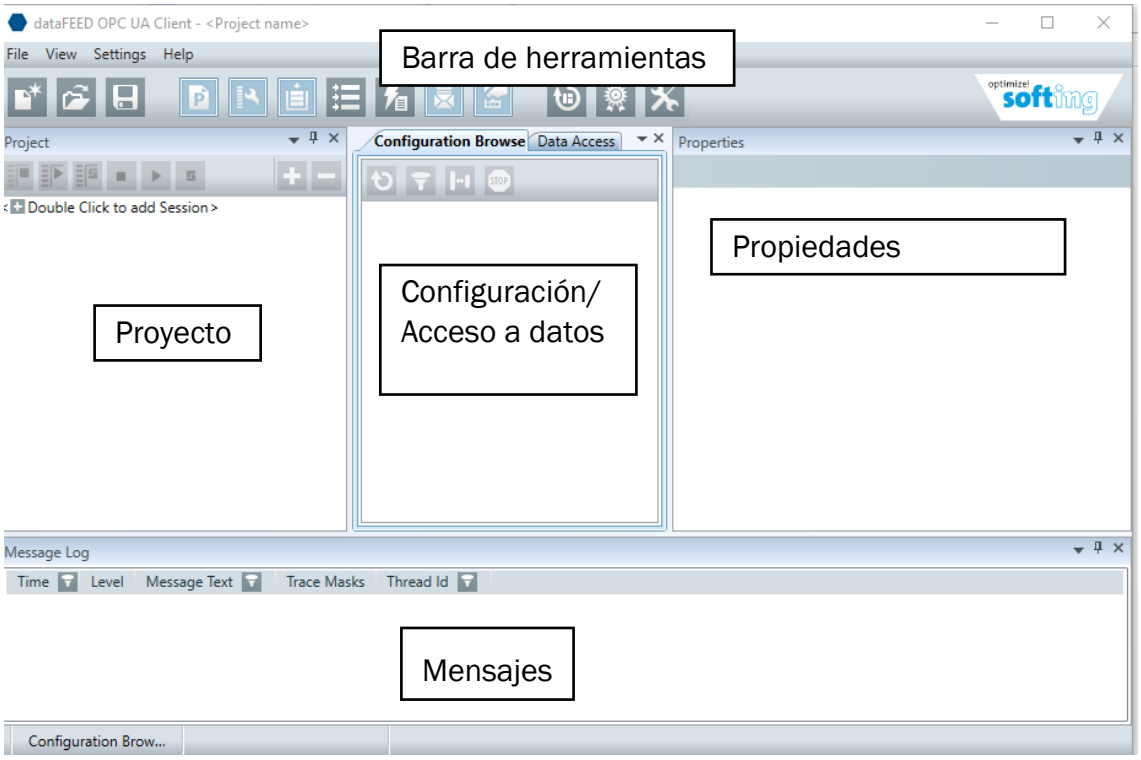


Figura 31 Cliente OPC UA – Interfaz –Bloques

En el bloque *Proyect* se añade la sesión seleccionando el servidor.

Luego de hacer doble *click* en "añadir servidor" que se muestra a continuación se debe escribir el puerto en el que el servidor escuchara, en este caso, el servidor está programado para escuchar en el puerto **1408**.

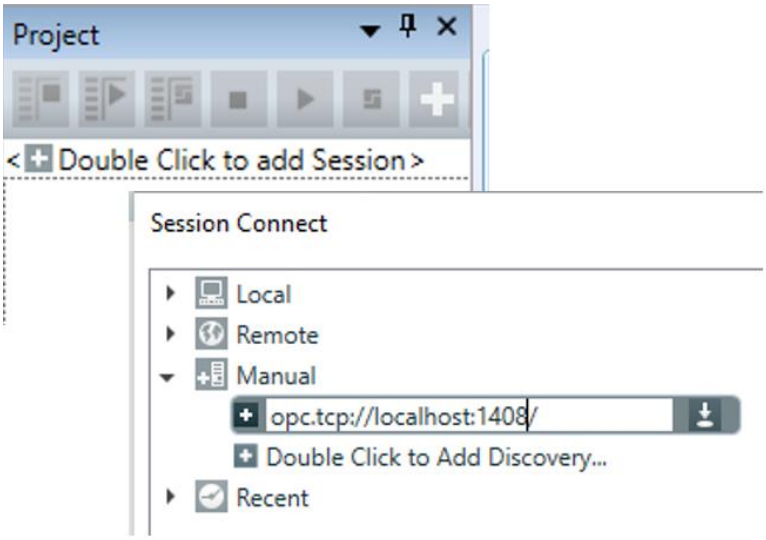


Figura 32 Cliente OPC UA – Interfaz –Añadir servidor - puerto 1408

El apartado “Session Properties” muestra las propiedades de la sesion, el siguiente paso es validar la conexión.

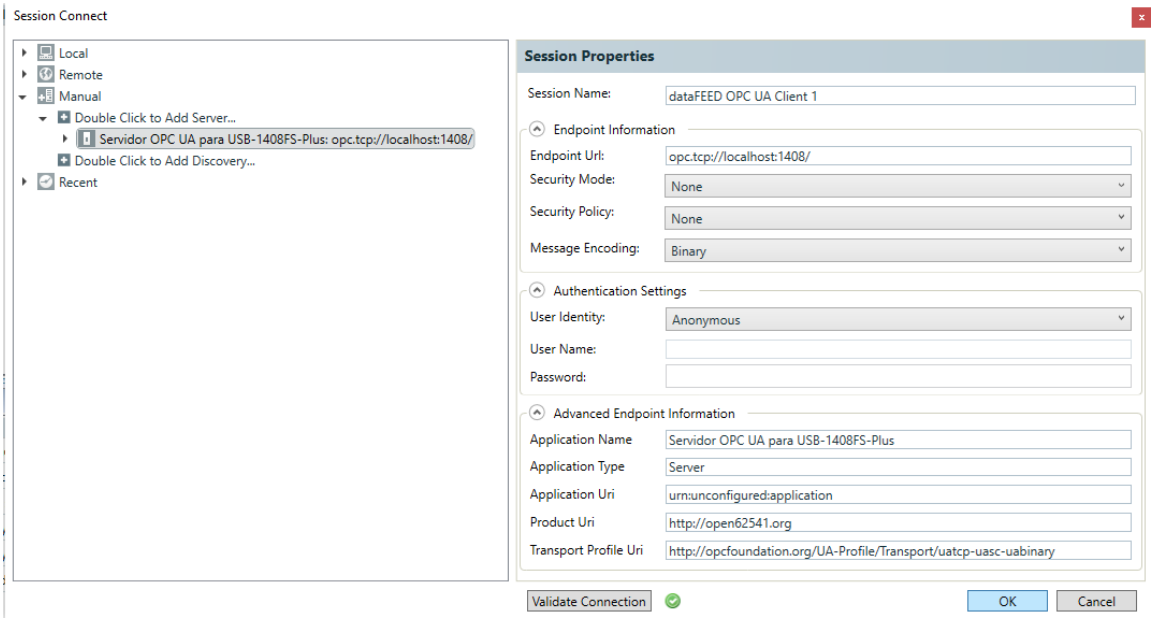


Figura 33 Cliente OPC UA – Interfaz –Validar conexión

Se observan los objetos del servidor, el bloque Properties muestra las propiedades de cada objeto seleccionado.

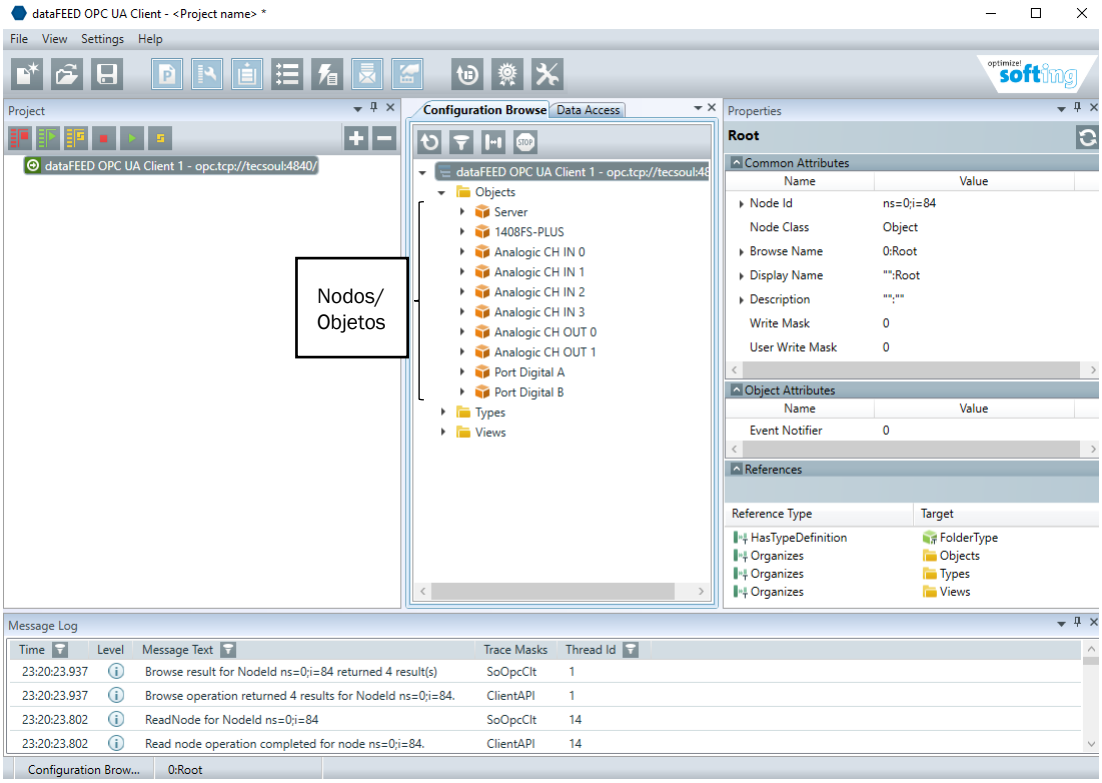


Figura 34 Cliente OPC UA – Interfaz –Nodos Objetos

A continuación se muestran cada uno de los objetos con sus variables:

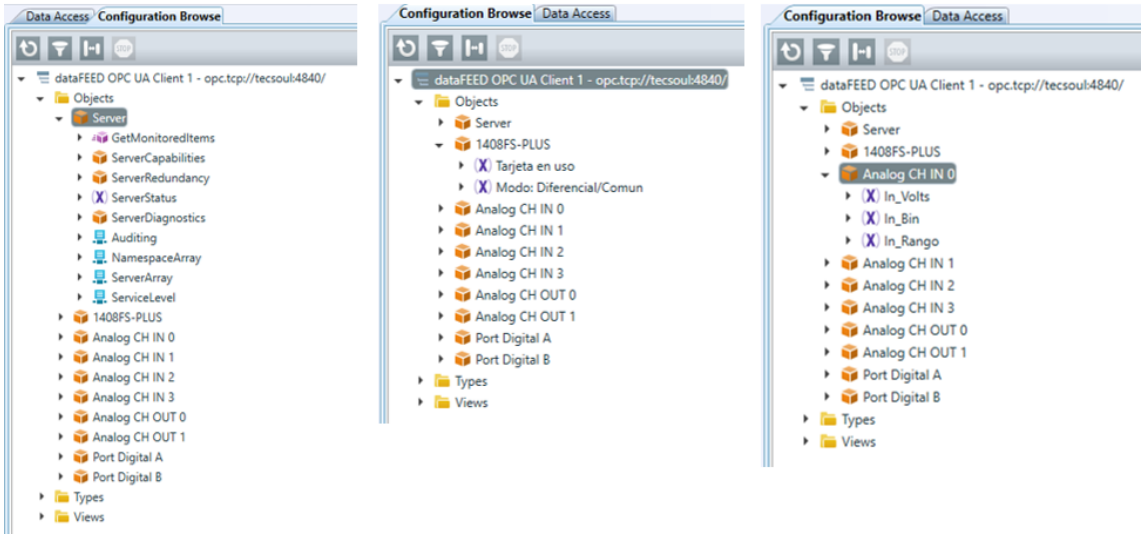


Figura 35 Cliente OPC UA – Interfaz –Nodos Variables de: 1408FS-Plus y Entradas analógicas

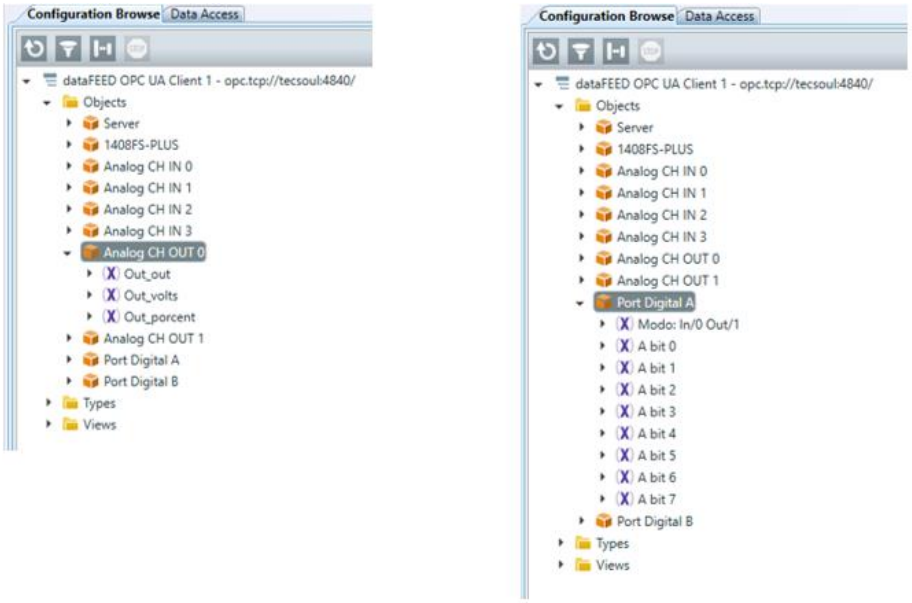


Figura 36 Cliente OPC UA – Interfaz –Nodos Variables de: Salidas analógicas y Puertos digitales

El bloque *Proyect* muestra los objetos a los que el cliente se ha suscrito (doble clic sobre el objeto).

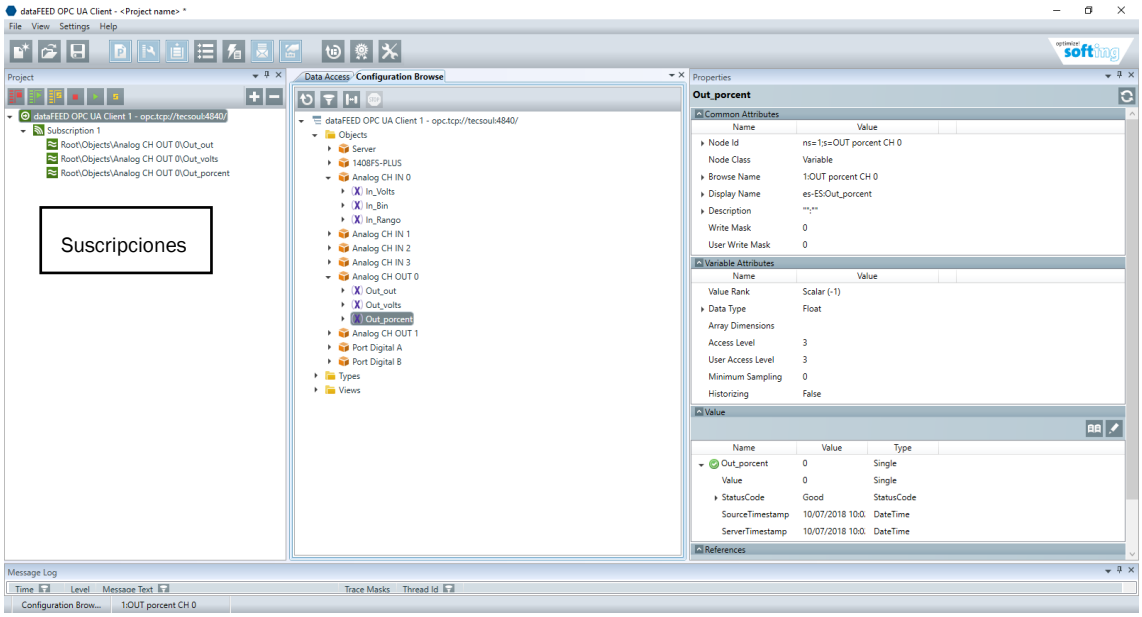


Figura 37 Cliente OPC UA – Interfaz –Suscripción

La pestaña *DataAccess* muestra el valor en tiempo real de los nodos a los que el cliente se ha suscrito.

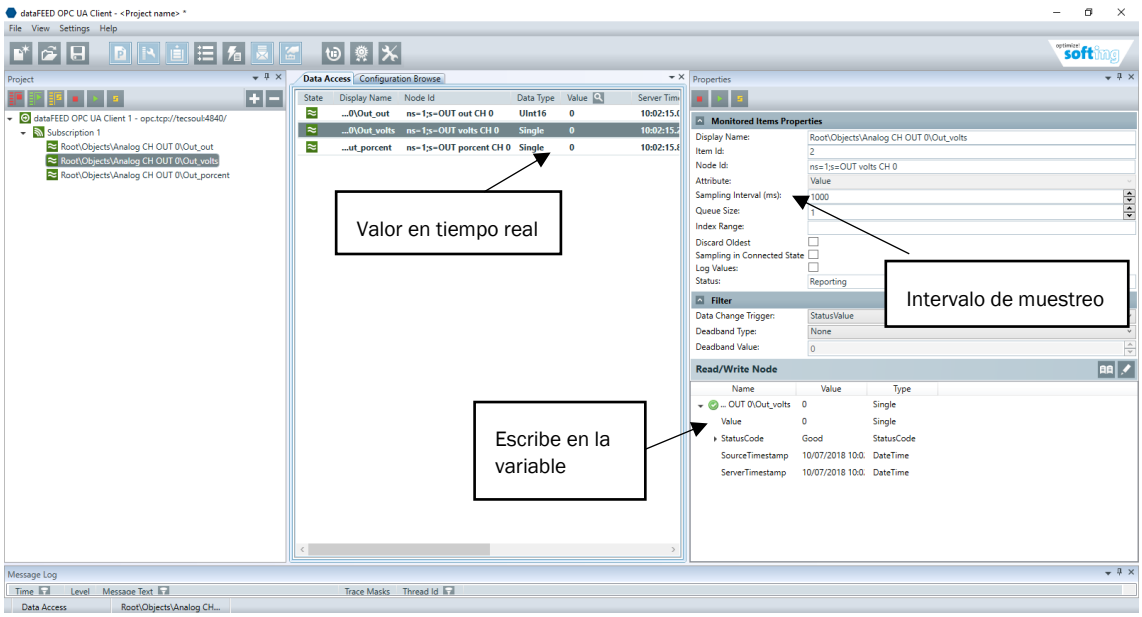


Figura 38 Cliente OPC UA – Interfaz –*DataAccess*

Ejemplo de ejecución con la variable Out_volts. Se observa que además el valor de Out_out y Out_porcent se actualiza.

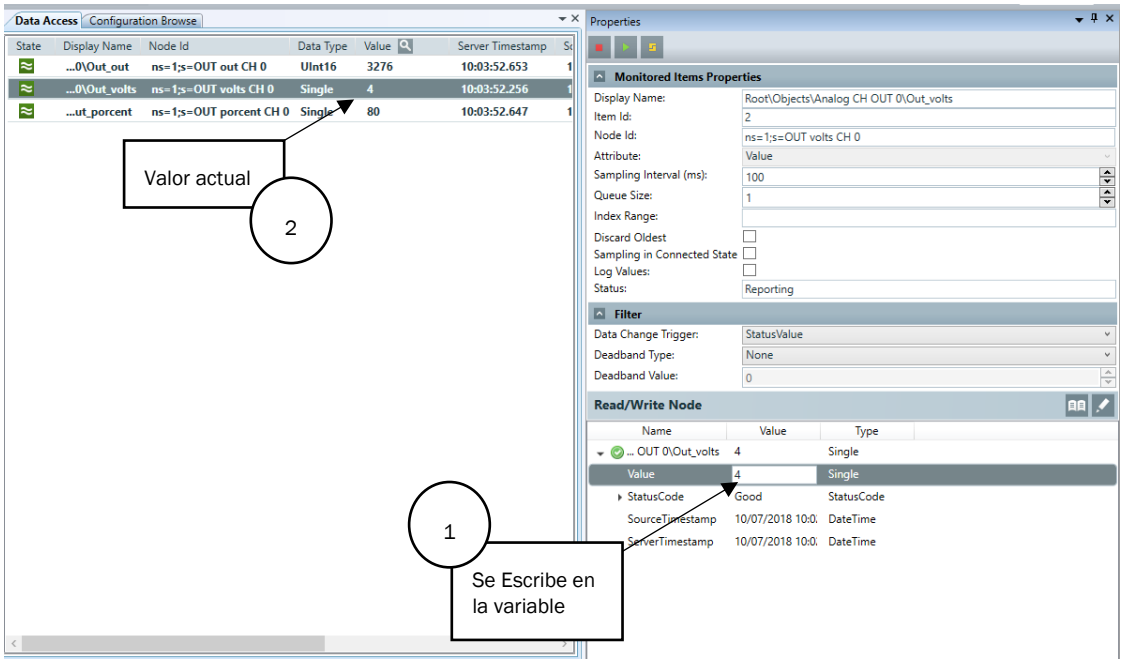


Figura 39 Cliente OPC UA - Interfaz - Escribe sobre variable

Se pueden agregar y/o eliminar suscripciones en tiempo de ejecución.

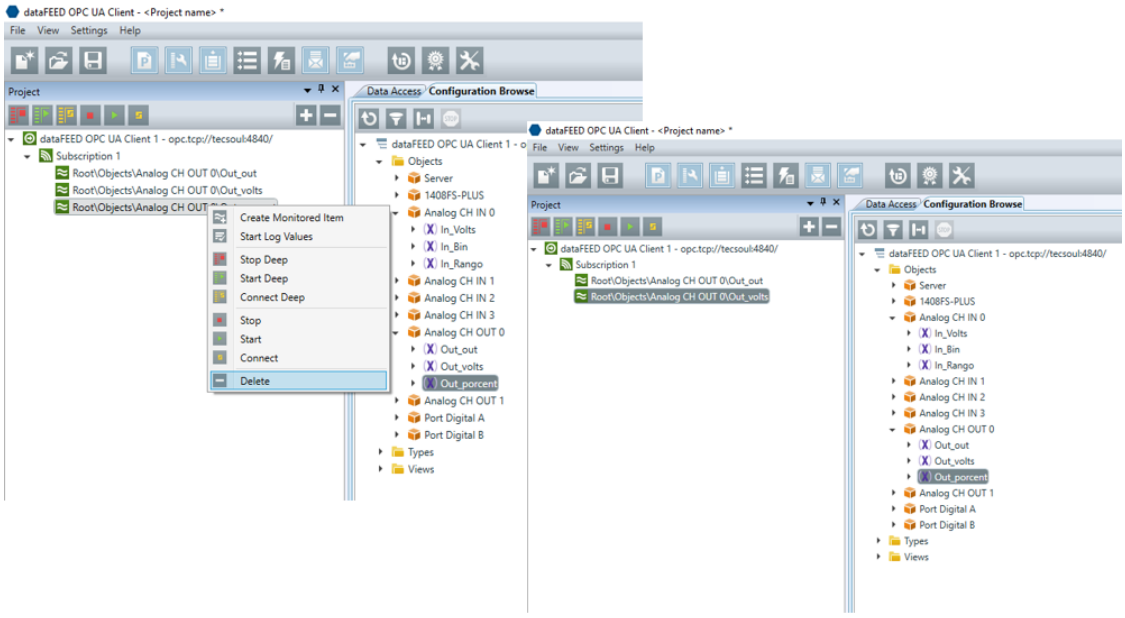


Figura 40 Cliente OPC UA - Interfaz - Eliminar suscripción

6.2. Validación

La verificación y validación del software es una disciplina técnica de la Ingeniería de Software, la cual tiene como propósito ayudar a las organizaciones de desarrollo a construir software de calidad durante el ciclo de vida del software. Determinar si los procesos y productos de software, satisfacen plenamente las necesidades de uso y del usuario según los requerimientos especificados.

La siguiente tabla muestra los objetos y variables que se crearon en el espacio de direcciones.

Nodo Objeto	Descripción	Nodo Variable	Descripción
1408FS-PLUS	Representa a la tarjeta USB-1408-FS-Plus	**Tarjeta en uso**	Número de referencia asignado en *InstaCal*.
		Modo: Diferencial/Comun	En modo diferencial= 4 En modo común = 8

Tabla 5 Servidor OPC UA: Tarjeta USB-1408FS-Plus

Analog CH IN 0	Representa al canal analógico de entrada 0.	**In volts CH 0**	Valor en voltios del canal.
		In bin CH 0	Valor en binario del canal.
		In rango CH 0	Rango del canal.
Analog CH IN 1	Representa al canal analógico de entrada 1.	**In volts CH 1**	Valor en voltios del canal.
		In bin CH 1	Valor en binario del canal.
		In rango CH 1	Rango del canal.
Analog CH IN 2	Representa al canal analógico de entrada 2.	**In volts CH 2**	Valor en voltios del canal.
		In bin CH 2	Valor en binario del canal.
		In rango CH 2	Rango del canal.
		In volts CH 3	Valor en voltios del canal.

Analog CH IN 3	Representa al canal analógico de entrada 3.	**In bin CH 3**	Valor en binario del canal.
		In rango CH 3	Rango del canal.
Analog CH IN 4	Representa al canal analógico de entrada 4.	**In volts CH 4**	Valor en voltios del canal.
		In bin CH 4	Valor en binario del canal.
		In rango CH 4	Rango del canal.
Analog CH IN 5	Representa al canal analógico de entrada 5.	**In volts CH 5**	Valor en voltios del canal.
		In bin CH 5	Valor en binario del canal.
		In rango CH 5	Rango del canal.
Analog CH IN 6	Representa al canal analógico de entrada 6.	**In volts CH 6**	Valor en voltios del canal.
		In bin CH 6	Valor en binario del canal.
		In rango CH 6	Rango del canal.
Analog CH IN7	Representa al canal analógico de entrada 7.	**In volts CH 7**	Valor en voltios del canal.
		In bin CH 7	Valor en binario del canal.
		In rango CH 7	Rango del canal.

Tabla 6 Servidor OPC UA: Canales analógicos de entrada

Analog CH OUT 0	Representa al canal analógico de salida 0.	**OUT out CH 0**	Valor en binario del canal.
		OUT volts CH 0	Valor en voltios del canal.
		OUT porcent CH 0	Valor en porcentaje del canal.
Analog CH OUT 1	Representa al canal analógico de salida 1.	**OUT out CH 1**	Valor en binario del canal.
		OUT volts CH 1	Valor en voltios del canal.
		OUT porcent CH 1	Valor en porcentaje del canal.

Tabla 7 Servidor OPC UA: Canales analógicos de salida

Port Digital A	Representa al puerto digital A	**A bit 0**	Valor en el bit 0 del puerto A
		A bit 1	Valor en el bit 1 del puerto A
		A bit 2	Valor en el bit 2 del puerto A
		A bit 3	Valor en el bit 3 del puerto A
		A bit 4	Valor en el bit 4 del puerto A
		A bit 5	Valor en el bit 5 del puerto A
		A bit 6	Valor en el bit 6 del puerto A
		A bit 7	Valor en el bit 7 del puerto A
Port Digital B	Representa al puerto digital B	**B bit 0**	Valor en el bit 0 del puerto B
		B bit 1	Valor en el bit 1 del puerto B
		B bit 2	Valor en el bit 2 del puerto B
		B bit 3	Valor en el bit 3 del puerto B
		B bit 4	Valor en el bit 4 del puerto B
		B bit 5	Valor en el bit 5 del puerto B
		B bit 6	Valor en el bit 6 del puerto B
		B bit 7	Valor en el bit 7 del puerto B

Tabla 8 Servidor OPC UA: Puertos digitales

A continuación se expondran pruebas realizadas al sistema para obtener una idea de los objetivos alcanzados.

Prueba	Válido
Obtener el id del tipo de tarjeta conectada al puerto USB.	✓
Obtener el modo de configutacion de la tarjeta.	✓
Fijar rango de las entradas analógicas en modo diferencial.	✓
Obtener el valor en voltios de las entradas analógicas.	✓
Obtener el valor en binario de las entradas analógicas.	✓
Fijar el valor en voltios de las salidas analógicas	✓
Fijar el valor en binario de las salidas analógicas	✓
Fijar los puertos digitales en modo entrada.	✓
Fijar los puertos digitales en modo salida.	✓
Fijar el valor del bit del puerto digital de salida	✓
Dos clientes OPC UA se suscriben a la mismas varaible.	✓

Tabla 9 Validación - validación de los objetivos

7. Conclusiones y líneas futuras

Para poner fin a la explicación del Trabajo de Fin de Grado realizado, este séptimo capítulo expone las conclusiones obtenidas a partir de la realización del proyecto, los objetivos alcanzados y las posibles líneas futuras que podrían dar lugar a mejoras o ampliación de las funcionalidades del servidor.

A través de la realización de este proyecto se ha logrado aprender y obtener experiencia en campos de la ingeniería como la programación al desarrollar el servidor, la electrónica al estudiar la tarjeta de adquisición de datos y a la investigación de material que en su mayoría se encontraba en ingles que ha añadido una dificultad extra al Trabajo de Fin de Grado. Por estas razones se considera que con el TFG se han cumplido las expectativas de aumentar la madurez en el desarrollo de proyectos de ingeniería y de poder aprovechar los conocimientos y capacidades adquiridas a los largo de la titulación.

El objetivo principal del TFG fue crear un Servidor OPC UA para una tarjeta de adquisición de datos USB-1408FS-Plus que para el cual se han ido logrando los siguientes hitos:

- ✓ Se realizó un estudio detallado de la tarjeta USB-1408FS-plus adquiriendo el conocimiento de las características y como debía ser el uso que se hiciese de ella, teniendo en cuenta todos los valores y tolerancias que posee la tarjeta. A demás, la Universal Library complementaba esta información para así poder pasar a la siguiente etapa que fue la parte de programación correspondiente al acceso a la tarjeta. En esta etapa del proyecto fue parte fundamental el TFG "Desarrollo de un servidor OPC para la tarjeta de adquisición de datos externa PMD-1208FS" debido a que se pudo reutilizar funciones ya creadas en ese proyecto.

- ✓ El análisis de los requisitos llevó el proyecto hacia el estudio del Desarrollo de Software, razón por la cual se añadieron al proyecto puntos cómo: los casos de uso, los diagramas de secuencia y la validación del software.

- ✓ Conocer desde lo que es el estándar OPC *Classic* hasta el OPC UA, parte fundamental de este proyecto fue comprender el modelo de información OPC UA para así poder implementarlo al servidor. Una vez comprendido el modelo el siguiente paso fue programar el servidor empleando la librería Open65412.org, esta fue la etapa más difícil debido a que debía de

desarrollar el servidor basándome en ejemplos muy básicos del uso de la librería, a pesar de ello el resultado cumple los requisitos.

- ✓ El último paso fue implementar en conjunto todo lo visto hasta el momento. Llegando al hito final que ha sido el servidor OPC UA.

Se espera que el servidor desarrollado pueda ser utilizado en el laboratorio de Control y Automática, y brindar todas las ventajas que proporciona OPC UA. Y además que sirva como base a otros estudiantes que deseen continuar la línea de investigación de estándares de comunicación.

Como líneas futuras, destaca la posibilidad de extender la funcionalidad del servidor OPC UA para representar otros modelos de tarjetas MCC, así como de ahondar en las ventajas de seguridad que proporciona OPC UA a través del uso de certificados electrónicos y contraseñas.

8. Referencias

[1] Mccdaq. USB1408FS-PLUS. Recuperado en marzo de 2018, de:

https://www.mccdaq.com/pdfs/manuals/USB-1408FS-Plus.pdf

[2] Marcos Pérez González. TFG Desarrollo de un servidor OPC para la tarjeta de adquisición de datos externa PMD-1208FS. Universidad de Valladolid, 2007.

[3] OPC Foundation. What is opc. Recuperado en marzo de 2018, de:

https://opcfoundation.org/about/what-is-opc/

[4] OPC Foundation. Classic. Recuperado en marzo de 2018, de:

https://opcfoundation.org/about/opc-technologies/opc-classic/

[5] OPC Foundation. Data Access Custom Interface Standard. Recuperado en marzo de 2018, de:

https://opcfoundation.org/about/opc-technologies/opc-classic/

[6] OPC Foundation. Alarms and Events Custom Interface Standard. Recuperado en marzo de 2018, de:

https://opcfoundation.org/about/opc-technologies/opc-classic/

[7] OPC Foundation. OPC Historical Data Access Specication. Recuperado en marzo de 2018, de:

https://opcfoundation.org/about/opc-technologies/opc-classic/

[8] OPC Foundation. Classic. Recuperado en marzo de 2018, de:

https://opcfoundation.org/about/opc-technologies/opc-classic/

[9] OPC Foundation. Unified Architecture. Recuperado en marzo de 2018, de:

https://opcfoundation.org/about/opc-technologies/opc-ua/

[10] OPC Foundation .Part 1: OPC UA Specification: Part 1 – Concepts. Recuperado en marzo de 2018, de:

http://www.opcfoundation.org/UA/Part1/

[11] OPC Foundation. Part 3: OPC UA Specification: Part 3 – Address Space Model. Recuperado en marzo de 2018, de:

http://www.opcfoundation.org/UA/Part3/

[12] OPC Foundation.Part 5: OPC UA Specification: Part 5 – Information Model. Recuperado en marzo de 2018, de:

http://www.opcfoundation.org/UA/Part5/

[13] Open62541.org. Protocol. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/protocol.html

[14] Open62541.org. Recuperado en abril de 2018, de:

https://open62541.org/

[15] JMIndustrial. Tarjetas de adquisición de datos. Recuperado en abril de 2018, de:

https://www.jmi.com.mx/tarjetas-de-adquisicion-de-datos

[16] National Instruments. Adquisición de datos – DAQ. Recuperado en abril de 2018, de:

https://www.ni.com/data-acquisition/what-is/esa/

[17] Mccdaq. USB1408FS-PLUS. Recuperado en marzo de 2018, de:

https://www.mccdaq.com/pdfs/manuals/USB-1408FS-Plus.pdf

[18] Universal Library. Recuperado en abril de 2018, de:

https://www.mccdaq.com/daq-software/universal-library.aspx

[19] Web de: Infor.uva.es – Casos de Uso. Recuperado en abril de 2018, de:

https://www.infor.uva.es/~chernan/Ingenieria/Teoria/Tema3D.pdf

[20] Web de: Infor.uva.es - El proceso Software. Recuperado en abril de 2018, de:

https://www.infor.uva.es/~mlaguna/is1/apuntes/7-Proceso.pdf

[21] Unversal Library. ULHelp.chw. Recuperado en mayo 2018, de:

https://www.mccdaq.com/Software-Downloads

[22] Open62541.org. Connecting a Variable with a Physical Process. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/tutorial_server_datasource.html

[23] Big learning. De casos de Uso a Diagramas de secuencia. Recuperado en junio de 2018, de:

https://www.youtube.com/watch?v=7vsPNUnhzTA

[24] UPV. Diagrama de casos de uso. Recuperado en junio de 2018, de:

https://www.youtube.com/watch?v=orvAkFFWo5o

[25] UPV. ¿Cómo construir un diagrama de secuencia? Recuperado en junio de 2018, de:

https://www.youtube.com/watch?v=Q1kH7XKxK5I

[26] Open62541.org. Information Modelling. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/nodestore.html

[27] Open62541.org. Working with Data Types. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/tutorial_datatypes.html

[28] Open62541.org. Building a Simple Server. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/tutorial_server_firststeps.html

[29] Open62541.org. Adding Variables to a Server. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/tutorial_server_variable.html

[30] Open62541.org. Connecting a Variable with a Physical Process. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/tutorial_server_datasource.html

[31] Open62541.org. Working with Variable Types. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/tutorial_server_variabletype.html

[32] Open62541.org. Working with Objects and Object Types. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/tutorial_server_object.html

[33] Open62541.org. Data Types. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/types.html

[34] Open62541.org. Server. Recuperado en abril de 2018, de:

https://open62541.org/doc/0.3/server.html

[35] NSIS. Recuperado en julio de 2018, de:

http://nsis.sourceforge.net/Main_Page

[36] Softing. OPC UA Demo Client dataFEED OPC / OPC UA SDKs. Recuperado en mayo de 2018, de:

https://data-intelligence.softing.com/products/datafeed-opcopc-ua-sdks/datafeed-opc-ua-client/

[37] Softing. Recuperado en mayo de 2018, de:

https://industrial.softing.com/en/startpage.html