Trabajo Fin de Master
Master Universitario en Automática, Robótica y Telemática (R.D.1393/07)

Sistema OPC para automatización mediante redes de estado.

Autor: Antonio Tamairón Pérez
Tutor: Iván Maza Alcañiz

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Master
Master Universitario en Automática, Robótica y Telemática.

Sistema OPC para automatización mediante redes de estado.

Autor:
Antonio Tamairón Pérez

Tutor:
Iván Maza Alcañiz
Profesor titular

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Master: Sistema OPC para automatización mediante redes de estado.

Autor: Antonio Tamairón Pérez

Tutor: Iván Maza Alcañiz

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mi madre, a mi padre,

Indistintamente…

"La tristeza es un don del cielo, el pesimismo es una enfermedad del espíritu" (Amado Nervo)

Resumen

Existen numerosas soluciones software basadas en redes de estado, que permiten modelar sistemas definidos en términos de causa-evento, para entender de manera sencilla como estos funcionan y evolucionan (entendiendo el concepto de proceso como un conjunto de fases sucesivas de un fenómento o hecho complejo). Si estos procesos los trasladamos al campo de la automatización, se puede ver que la mayoría del software existente, de este tipo, no está orientado a dar soluciones en este aspecto, y al final carecen de las herramientas necesarias para implementar y simular un automatismo del tipo que sea, y aun mas, conseguir que este finalmente interactue con el entorno real (con el conjunto de sensores y actuadores). Por otro lado, el interfaz de usuario que poseen, deja mucho que desear al no estar orientado para el y no siendo nada adecuado para el trabajo de implementación del sistema. Todo esto concluye en tener herramientas que convierten una tarea supuestamente más sencilla, en una tediosa tarea de implementación con una ingrata experiencia para el usuario y con una acusada falta de soluciones para conectar con el mundo real.

Partiendo entonces de esta problemática, se planteó la idea de crear un software de desarrollo que subsanara dichas deficiencias, y de alguna manera sacase partido a la idea de implementar automatismos con la sencilla acción de unir elementos entre sí, con alguna que otra configuración rápida y además se pudiera visualizar el proceso de manera gráfica y su evolución, cuando este estuviese siendo ejecutado/procesado. Por otro lado, aportar un valor añadido a dicha herramienta y permitir que pudiera interactuar con entornos de control reales, aumentando su utilidad.

Este documento pretende describir dicho proyecto antes brevemente definido, sobre este software de implementación mediante interfaz gráfico, que permite automatizar procesos, ya sean de entorno industrial, domótico u otros entornos que puedan ser automatizados a alto nivel (partiendo de que los sensores y actuadores ya están direccionados en un controlador y es posible interactuar con ellos mediante un servidor OPC) construyendo estructuras de elementos estado-transición interconectadas, y haciendo uso del estándar OPC. El objetivo final es obtener un conjunto de herramientas contendidas en un interfaz gráfico muy orientado al usuario, que permita formar estructuras complejas de elementos activos interconectados (elementos que pueden realizar acciones, cálculos, operaciones de E/S OPC, etc.), permitiendo de manera sencilla automatizar procesos y poder interpretarlos (ver como evoluciona en tiempo de ejecución, gracias a un *runtimer*(1)) y tener la posibilidad de hacer que interactue con el mundo real, gracias al estandar OPC.

Es OPCSM VAI (OPCSM – Virtual Automation Interpreter) el software que reúne dichas condiciones, producto del desarrollo de este proyecto y que será descrito con mas detalle en este documento.

(1) Motor de ejecución basado en eventos de un temporizador con base de tiempo T, que reproduce en cada flanco las acciones de aquellos elementos del sistema seleccionados para ser procesados.

OPCSM VAI posee un interfaz gráfico amigable, proporcionando al usuario un conjunto de herramientas con capacidad de implementar sistemas basados en diagramas de estado de manera gráfica, con el objetivo de automatizar procesos para aprender sobre ellos (sirviendo pues como herramienta didáctica) y/o como solución a problemas de automatización, evitando la dificil fase de programación, mediante comandos, que se suele hacer en el lado del controlador (en este caso OPCSM VAI se convierte en un controlador de alto-nivel pudiendo estar como *middle-ware* entre SCADA y PLCs).

Por lo tanto, OPCSM VAI permite dar solución a sistemas reales, incluso servir de base a posibles líneas futuras dirigidas a este campo. La característica de permitir dar solución a problemas de implementación de sistemas automáticos, se consigue gracias a la integración del estándar OPC en la librería de elementos y en la conectividad que se lleva a cabo con servidores OPC, al definir un proyecto OPCSM nuevo.

La capacidad de poder enviar y recibir datos de una red en la que trabajan PLCs (y a su vez sensores y actuadores) permite que OPCSM VAI deje de ser una herramienta puramente didáctica y salte al conjunto de aplicaciones para automatizar procesos reales, aunque aun algunos aspectos de esta no le permitan estar en el grupo de aplicaciones profesionales (por ejemplo, al no poseer la capacidad de implementar sistemas de tiempo real) OPCSM VAI no deja de ser, una realidad de una idea que bien pudiera poseer bastantes vertientes futuras para dar soluciones mas profesionales y de bajo coste, adecuando cada vez mas el software a estas, extendiéndolo a otras plataformas, creando dispositivos hardware OPCSM, etc.

A lo largo del documento, se navegará entre varios capítulos, que de manera resumida, darán una visión de la complejidad interna de la herramienta, su arquitectura y versatilidad además de contener una guía de uso para presentar el entorno, y como sacarle partido al conjunto de herramientas que posee, elementos de los que dispone, etc.

En definitiva, el lector podrá observar finalmente que, OPCSM VAI parte de una filosofía de implementación de automatismos muy intuitiva y simple, apoyándose en el uso de sistemas causa-evento, mediante estructuras complejas de elementos activos interconectados, consiguiendo finalmente que esta simple idea pueda dar soluciones muy competentes en el campo de la automatización, sirviendo como elemento intermediario entre SCADAs y PLCs y pudiendo sustituir en parte la difícil y compleja fase de programación del controlador mediante comandos.

Índice

1 INTRODUCCIÓN

Si buscas resultados distintos, no hagas siempre lo mismo.

- Albert Einstein -

Los sistemas con estándar OPC (*OLE* for process control*) han permitido en el campo del control y la automatización abrir un amplio abanico de posibilidades en la comunicación entre sistemas para procesos industriales, liberándolos de la problemática existente a la hora de interaccionar en tiempo real con todos ellos y entre sí (Dada la diversidad de fabricantes, tecnologías y protocolos existentes) y permitiendo una fácil integración a alto nivel, evitando la necesidad de conocer en profundidad las características concretas de dichos sistemas, en cuanto al estándar de intercambio de datos que los rige, así pues OPC nace para solventar dicho problema. En 1996 un conjunto de empresas del sector industrial, integradores y desarrolladores se unen para desarrollar el estándar OPC, a grandes rasgos especifica la adaptación para el intercambio de datos en tiempo real entre dispositivos de control de diferentes fabricantes de una planta industrial, mediante un interfaz intermediario entre el área de control (PLC) y la supervisión (HMI/SCADA).

OPC posee una arquitectura de comunicación cliente-servidor y servidor-servidor, para llevar a cabo la labor de interfaz en el intercambio de datos (lectura y/o escritura) entre elementos OPC. Básicamente el principal objetivo es conseguir una adaptación fluida entre controladores electrónicos y autómatas o comúnmente denominados por sus siglas en inglés, PLC (*Programmable Logic Control*) y aplicaciones de control/supervisión HMI/SCADA (Entre otras aplicaciones de gestión, bases de datos, etc.) sea cual fuere su tecnología, protocolo, etc. El intercambio de información en un proceso industrial está a nivel de planta, ya que cada controlador hace su labor en su lazo de control y no se hace quizás perceptible que, el problema se acusa cuando se necesita interactuar con todos ellos, siendo estos de distinto fabricante mediante un entorno HMI/SCADA por ejemplo y permitir al operador entender la planta como una única unidad funcional, siendo transparente en cuanto a su arquitectura. La solución en este aspecto será insertar un elemento servidor OPC intermediario que adapte la información entre los extremos del sistema industrial.

El servidor al fin y al cabo lo puede solucionar porque posee un amplio conjunto de drivers (Interfaces entre sistema operativo y dispositivo) necesarios para poder entenderse con cada controlador y/o PLC por separado, entonces se encarga de enrutar los datos al driver correspondiente, previo formateo de la información.

El servidor además permite que el hardware que trabaja en un proceso de control sea actualizado sin necesidad de cambiar nada o mínimamente el entorno HMI/SCADA y las aplicaciones satélites colgadas de la planta, minimizando así el impacto de la sala de control/supervisión; puesto que con tan solo agregar nuevos drivers al servidor OPC es posible solventar el problema.

* OLE : *Object Linking and Embedding*

Los sistemas OPC han tenido una gran acogida dada la ventaja que supone, según se ha visto anteriormente y hoy día todos los fabricantes, especialmente los grandes fabricantes, poseen servidores/clientes OPC para poder intercambiar información con otros controladores del proceso, incluso en sus propios sistemas.

El proyecto descrito a lo largo de este documento quiere dar a conocer un software de desarrollo para automatización de procesos bien de carácter industrial o automatismos de otra índole, como domótica por ejemplo, mediante sistemas construidos con redes de estados (elementos activos, estados y transiciones, interconectados entre sí, que permiten implementar sistemas definidos en términos de causa-evento) pero con un valor añadido, proporcionar las herramientas necesarias para hacer que el software interactúe directamente con aplicaciones OPC, consiguiendo entonces una herramienta muy simple en su uso y a la vez potente en las soluciones que puede aportar para estos entornos, además de poder ser ampliada y mejorada.

El software denominado OPCSM VAI (*OPC StateMatic - Visual Automation Interpreter*) es un potente y sencillo entorno de desarrollo mediante interfaz gráfico, que permite crear automatismos e interpretarlos mediante un motor de ejecución temporizado o *runtimer* (pero por ahora no de tiempo real), sobre plantas, sistemas domóticos o entornos hardware de bajo coste como Arduino, para ser usados como solución docente o como solución software en el campo de la automatización. Esta herramienta podría dar solución rápidamente de control sobre PC, en entornos de procesos automáticos reales, entre otras soluciones como proyectos DIY (*Do I Yourself*) mediante hardware de bajo coste, etc.

Realmente OPCSM VAI sigue un camino que no es nuevo, en el que existen multitud de simuladores e intérpretes de todo tipo basados en la filosofía de redes de estado (principalmente redes de Petri) pero de los cuales, pocos se acercan a la idea de ser una herramienta que permita dar solución a la programación de automatismos reales, sino únicamente adentrar al usuario en el funcionamiento de las redes de estado, con lo cual son deficientes en este aspecto, además no están bien orientadas al usuario, y sus interfaces gráficos son complicados de usar.

Por otro lado OPCSM VAI no puede competir con aplicaciones de desarrollo visual para automatización de procesos industriales, SCADA y/o HMI, como podrían ser LabView (de National Instruments), Intouch HMI (de Wonderware), SIMATIC (de Siemenes), Automation Studio, etc., al tratarse de sistemas muy complejos y potentes, cuyo objetivo en el campo de la automatización es otro, y aunque podrían dar solución mediante desarrollos parecidos para el mismo objetivo que lo haría OPCSM VAI gracias a OPC, su complejidad de uso no los hace aptos para llevar a cabo soluciones en las este último si es mas adecuado, pues reduce el tiempo de preparación necesario y aprendizaje, entre otras cosas.

OPCSM VAI hace uso del estándar OPC clásico (por cuestiones de coste), pero OPC ha evolucionado hasta un estándar denominado OPC UA (*Unified Architecture*) que es mas avanzado y novedoso, aunque para la cuestión que atañe, con la versión clásica la funcionalidad que se necesita en el proyecto, queda perfectamente cubierta.

1.1 Motivación del proyecto.

Cuando comencé la genial andadura por el mundo de la automatización y la robótica que este Master (semilla de este proyecto) ha estado construyendo en mi conocimiento durante estos años, siempre he tenido la idea de crear un sistema que permitiese implementar automatismos aprovechando la versatilidad de un entorno gráfico, y de la sencilla tarea de interconectar elementos para crear una red compleja que mediante código en el controlador se convertiría en cientos de líneas escritas sin a priori, una visión clara de un vistazo.

La motivación vino mas tarde promovida por una ingrata experiencia al trabajar con varias herramientas para desarrollar y simular procesos automáticos mediante redes de Petri, con una inadecuada interfaz hacia el usuario y con pocas herramientas de edición además de poca variedad de elementos para construir la estructura del sistema y lo más importante, carecer de la posibilidad de interacción con el mundo real. Fue en ese instante cuando la idea que había estado intentando hacer realidad y a la cual encontrar utilidad, se vio claramente materializada ¿Por qué no crear un entorno que mejorase esta idea de implementación mediante interconexión de elementos, con un buen interfaz gráfico que permita una buena experiencia de trabajo al usuario y que una vez creados los automatismos estos puedan ejecutarse e interactúen con SCADAs y PLCs? Pudiendo además,

mediante un controlador preparado con solamente el direccionamiento hacia los sensores y actuadores, realizar toda la fase de programación y ejecución desde este entorno. Aquí es entonces cuando nace OPCSM VAI.

Buscando por la red, podemos ver que existen una infinidad de aplicaciones para simular redes de Petri (ya que no se alejan de esa idea) pero la mayoría no se acercan a la posibilidad de servir como herramienta de ayuda en sistemas reales y mucho menos orientados al aprendizaje en el campo de la automatización. Quizás una de las herramientas más cercanas a esta filosofía podría ser Petri.NET, pero su interfaz gráfico no parece muy amigable y no parece estar orientado al usuario, y aunque posee herramientas de implementación el entorno no es tan intuitivo como cabría esperar. No tiene una jerarquía de desarrollo orientada a proyecto, y múltiples documentos para organizar la implementación, su filosofía está basada en usar bloques, que contienen subsistemas, con lo cual al final se pierde esa visión general. Por otro lado, un detalle muy importante es que la integración OPC no es tan directa, necesitando de un desarrollo más complejo mediante Python, para poder comunicarse con el servidor OPC, aunque si es cierto que esta última característica lo hace más versátil, pero al final se convierte en un entorno de programación perdiendo la idea inicial de crear sistemas para automatizar procesos industriales de manera sencilla y rápida.

1.2 Introducción al estándar OPC

OPC es un estándar para intercambio seguro y fiable de datos en el sector de la automatización industrial y otras industrias. Es una plataforma independiente que asegura un flujo de información entre múltiples fabricantes aun con sus distinciones en cuanto a tecnologías, protocolos, etc.

El estándar OPC es una serie de especificaciones desarrolladas por los fabricantes del sector industrial, los usuarios finales y los desarrolladores de software. Estas especificaciones definen el interfaz entre servidor/cliente así como entre servidor/servidor, incluyendo acceso en tiempo real a los datos, monitorizado de alarmas y eventos, acceso al histórico de datos y otras aplicaciones.

Cuando el estándar fue lanzado por primera vez en 1996, se propuso abstraer los protocolos específicos de PLCs, como Modbus, Profibus, etc en un interfaz estándar permitiendo a los sistemas HMI/SCADA tener un interfaz controlado por el hombre, que convertiría las solicitudes genéricas de L/E en solicitudes específicas que cada dispositivo entendiese y viceversa. Como resultado, surgió toda una industria artesanal de productos basados en esta idea, permitiendo al usuario final implementar sistemas de calidad que interactuaban perfectamente con OPC.

Inicialmente, el estándar OPC fue restringido al sistema operativo Windows, de hecho el acrónimo OPC nació de OLE para Control de Procesos (OLE for Process Control). Estas especificaciones, las cuales son conocidas como OPC Clásico, y han sido utilizadas con gran aceptación en múltiples industrias, incluidos el sector de la producción, fabricantes de automatismos, refinerías, gaseoductos, energías renovables y muchas otras.

Con la introducción de las arquitecturas orientadas a servicios en los sistemas de producción, se introducen cambios y se plantean nuevos retos en seguridad y modelado de los datos. La fundación OPC desarrolló las especificaciones OPC UA para cubrir estas necesidades a la misma vez proporcionar una tecnología de futuro, con una arquitectura de plataforma abierta que permite mejorar, escalar y extenderse con el paso del tiempo.

Estas son solo algunas de las razones por las que algunos miembros y organizaciones tecnológicas (Colaboradores) están actualizándose a OPC UA. En este proyecto trataremos solo el estándar OPC Clásico.

1.2.1 Estándar OPC Clásico.

El estándar OPC Clásico está especificado para plataforma Microsoft Windows haciendo uso de COM/DCOM (Distributed Component Object Model) para el intercambio de datos entre componentes del software. Las especificaciones proporcionan definiciones separadas para el proceso de acceso a los datos, alarmas e históricos, que OPC cubre con OPC DA, OPC AE y OPC HDA. A continuación se puede ver una breve descripción de estos. Este proyecto solo hace uso de OPC DA.

1.2.1.1 OPC Data Access (OPC DA)

Define el intercambio de datos incluyendo valores, momento (tiempo) y calidad de información. Actualmente se encuentra en la versión 3.0 de la especificación, y cubre los siguientes aspectos:

- Conceptos de la tecnología OPC Cliente/Servidor y definición de datos.
- Actividades generales incluyendo definición de un espacio de direcciones y de su explorador, lector, escritor y subscriptor para las notificaciones de aquellos datos modificados.
- Descripción detallada del interfaz, métodos, parámetros y comportamiento esperado.
- Descripción detallada de los tipos de datos y estructuras.
- Código de ejemplo (OPC Server) que contiene definición de interface y códigos de error.

1.2.1.2 OPC Alarms and Events (OPC AE)

Define el intercambio de alarma y mensajes de información de tipo eventos también estado de variables y gestión de estado. Esta especificación describe el comportamiento de un servidor OPC que puede monitorizar áreas y notificaciones de clientes referente a condiciones de alarma. No han evolucionado desde la versión 1.1. Las especificaciones cubren:

- Propósito de las alarmas y tecnología de eventos
- Resumen de la arquitectura general y metodología de alarmas, eventos, acuses de recibo, consultas y áreas.
- Descripción detallada del interfaz, métodos, parámetros y comportamiento esperado.
- Descripción detallada de los tipos de datos y estructuras.
- Código de ejemplo (OPC Server) que contiene definición de interface y códigos de error.

1.2.1.3 OPC Historical Data Access (OPC HDA)

Define los métodos de consulta y analíticos que pueden aplicarse al historial y la trazabilidad de los datos. La especificación describe el comportamiento del servidor OPC referente al almacenaje de la información en una especie de BBDD, y como los clientes pueden recuperar dicha información usando OPC. Las especificaciones en su versión 1.2, cubren:

- Visión general y propósito de la tecnología del histórico de datos.
- Visión general de la arquitectura y metodología.
- Descripción detallada del interfaz, métodos, parámetros y comportamientos esperados.
- Descripción detallada de las funciones agregadas con ejemplos de consulta y resultados de cada uno.
- Descripción detallada de los tipos de datos y estructuras.
- Código de ejemplo de servidor OPC conteniendo el interfaz y el control de errores.

1.3 Servidor OPC

Un servidor OPC es una aplicación de software (driver) que cumple con una o más especificaciones definidas por la OPC Foundation. El Servidor OPC hace de interfaz comunicando por un lado con una o más fuentes de datos utilizando sus protocolo nativos (típicamente PLCs, DCSs, básculas, Módulos I/O, controladores, etc.) y por el otro lado con Clientes OPC (típicamente SCADAs, HMIs, generadores de informes, generadores de

gráficos, aplicaciones de cálculos, etc.).

En una arquitectura Cliente OPC/ Servidor OPC, el Servidor OPC es el esclavo mientras que el Cliente OPC es el maestro. Las comunicaciones entre el Cliente OPC y el Servidor OPC son bidireccionales, lo que significa que los Clientes pueden leer y escribir en los dispositivos a través del Servidor OPC.

Una de las principales funciones de los servidores OPC es la traducción de datos/mapping, es decir la función de un Servidor OPC es el traducir datos nativos de la fuente de datos en un formato OPC que sea compatible con una o más especificaciones OPC mencionadas anteriormente (ejemplo: OPC DA para datos en tiempo real).

Las especificaciones de la OPC Foundation solo definen la porción OPC de las comunicaciones del Servidor OPC, así que la eficiencia y calidad de traducción del protocolo nativo a OPC y de OPC al protocolo nativo dependen enteramente de la implementación del desarrollador del Servidor OPC.

Otra característica de los servidores OPC es que, comunican nativamente con las fuentes de datos, por ejemplo: dispositivos, controladores y aplicaciones. Las especificaciones de la OPC Foundation no especifican como el Servidor OPC se debe comunicar con la fuente de datos porque hay una gran variedad de fuentes de datos disponibles en el mercado. Cada PLC, DCS, controlador, etc. tiene su propio protocolo de comunicación o API que a su vez permiten la utilización cualquier cantidad de conexiones físicas (serial RS485 o RS232, Ethernet, wireless, redes propietarias, etc.)

Dos ejemplos comunes de cómo se comunican los Servidores OPC con la Fuente de Datos son:

- A través de una interfaz de programación de aplicaciones (API) para un driver personalizado escrito específicamente para la Fuente de Datos.
- A través de un protocolo que puede o no ser propietario, o basado en un estándar abierto (por ejemplo utilizando el protocolo Modbus. (MatrikonOPC Server para Modbus)

Entonces todo esto lo resuelve también el desarrollador, es la parte de adaptación formada por el protocolo de comunicaciones, adaptación de datos, etc, pero no queda definida por el estándar, obviamente por la diversidad.

Uno de los servidores OPC mas completos que se ha podido probar, bajo experiencia propia es ServerEx del desarrollador Kepware. ServerEx posee una multitud de drivers y protocolos, para poder mantener una fluidez de datos entres casi cualquier controlador, PLC, etc. existente.

1.3.1 Direcciones OPC. TAG.

Sin entrar en mayor detalle en los servidores OPC, sí que conviene entender el formato para acceder a los datos en el servidor. Estos están direccionados según una ruta que es simplemente un punto de dato y según la especificación OPC del tipo de servidor y del desarrollador del servidor se tendrá un direccionamiento concreto. Para el caso de Servidores OPC DA (Que es justamente lo que OPCSM VAI utiliza), el TAG identifica un acceso a una variable concreta, que puede ser un registro en un PLC, o un dispositivo concreto, o un canal en un controlador. El TAG es la última parte de la ruta. La ruta está formada por varias secciones (2 o 3, según el desarrollador del servidor), por ejemplo, en un servidor de Kepware la dirección está formado por tres partes "A"."B"."C", donde

A) Es el canal "Channel_1"

B) Es el dispositivo "Omron_PC2"

C) La variable o registro del PLC, por ejemplo Re1 que puede ser, R0.0, D0001, etc.

Entonces la ruta quedaría "Channel_1.Omron_PC2.Re1"

Conviene entender este direccionamiento ya que OPCSM VAI, al conectar a un servidor, extrae todos sus TAGs para poder ser asignados mediante elementos OPC, InOPC o OutOPC y realizar lecturas/escrituras.

2 ARQUITECTURA SOFTWARE

No se equivoquen, la sencillez solo se logra a través del trabajo duro.

- Clarice Lispector -

El proyecto constituye un entorno de desarrollo visual de sistemas basados en diagramas de estado con un motor de ejecución (que interpreta los diagrama de estados de manera sincronizada procesando elementos que poseen un testigo) y un interfaz OPC, que permite dar vida a los sistemas implementados.

El objetivo principal de dicho proyecto, además del valor añadido dado por OPC, es conseguir un entorno de desarrollo versátil, que permitiese facilitar al usuario la implementación de sus proyectos de automatización, facilitando el desarrollo de los distintos diagramas de estado, la organización de los elementos y gestión del proyecto. Esta idea, conlleva dar al usuario una serie de herramientas de edición visual, mediante objetos que ayude en todas las tareas de creación del sistema, además de utilidades y ayudas en la ejecución/depuración del desarrollo final.

El primer paso antes de nada era saber en que línea debía ser resuelto el proyecto, con que tecnología de desarrollo, y la respuesta a todas estas premisas clamaban solucionar el problema mediante tecnología software orientada a objetos, ya que los propios requisitos dejaban ver que la única manera de mantener cierta organización era construir el software de manera modular, y que entre dichos módulos (objetos) existiese desde el más bajo al más alto en la jerarquía una comunicación. Por lo tanto el proyecto era adecuado para será ser implementado mediante objetos.

Así pues, se hace necesario tener varios objetos base, elementos, con sus propiedades, métodos y funciones para realizar acciones entre ellos y que permitan en conjunto formar un entorno completo, estructurado y jerárquico. El producto final sería, un entorno para agilizar las tareas de edición y simulación de cara al usuario y poder ir extendiendo poco a poco, en cuanto a su funcionalidad, etc.

Para conseguir todo esto, hay detrás una compleja programación, llevada a cabo en plataforma *Microsoft Windows 7*, bajo el entorno de desarrollo *Microsoft Visual Studio .NET 2015* y lenguaje C#, que da vida a OPCSM VAI. El proyecto está compilado con una versión 4.6 del *framework .NET*

En los distintos apartados que se podrán ver a continuación, se describirá con mayor detalle aspectos relevantes del desarrollo que darán una idea al lector, de la complejidad que supone realizar un entorno con estas características, para facilitar las tareas de desarrollo al usuario y conseguir una satisfactoria experiencia final cuando trabaja con OPCSM VAI.

2.1 Arquitectura de la aplicación

Toda la aplicación está basada en programación orientada a objetos. Desde este instante se debe tener en cuenta este aspecto para entender cómo funciona todo. Para comenzar, se muestra un diagrama simple que permite proporcionar al lector un concepto general de la implementación de OPCSM VAI.

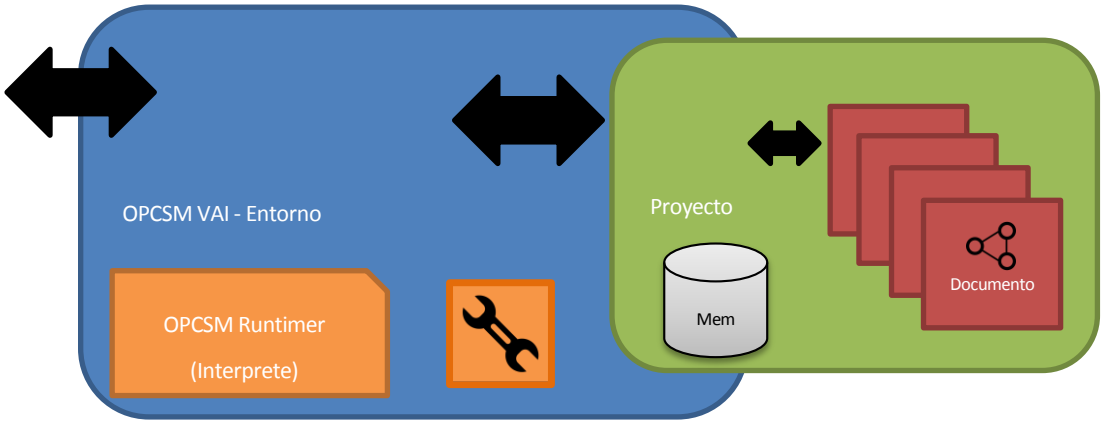


Fig. 2.1-1 – Arquitectura del entorno OPCSM VAI

En la *figura 2.1-1*, se puede interpretar que el entorno de edición de OPCSM VAI necesita de otros objetos (módulo Proyecto, módulo documento, objetos elementos) para que el usuario pueda crear un "Proyecto OPCSM" (Para el automatizado de procesos mediante diagrama de estados y estándar OPC).

El "entorno" interactúa a través del "proyecto" con los distintos "documentos" y estos a su vez con los "elementos" que contiene, los cuales forman finalmente un sistema de diagramas de estados, para automatizar un proceso.

Cada objeto (Proyecto, documento, elemento) posee una serie de métodos y propiedades necesarias a la hora de realizar sobre ellos acciones o que necesiten interacción entre ellos (Ya que hay una comunicación constante entre dichos objetos). Gracias a todo esto, es posible un "entorno" de desarrollo que proporcione al usuario toda la funcionalidad disponible de estos elementos.

2.2 Módulos

2.2.1 Objeto proyecto

El objeto proyecto, (Librería cProjectOSM) es el objeto de más alto nivel en la jerarquía de objetos que posee el programa. Es el contenedor de todos los documentos que el usuario generará, para automatizar un proceso concreto.

El objeto proyecto posee información pertinente acerca del proyecto real que se lleva a cabo, datos necesarios para su desarrollo y simulación, así como punteros a los distintos documentos creados. Posee una serie de métodos y propiedades, al igual que el resto de objetos, cuyo objetivo es permitir que una vez creada una instancia de este objeto, se pueda interactuar con él.

Por sí solo, el objeto proyecto no posee entorno gráfico para el usuario, ya que esa parte se ha dejado absolutamente dedicada a los objetos documento, con lo cual se debe entender a este objeto como un almacén organizado de objetos documento. El objeto proyecto proporciona además a los documentos un elemento muy importante a la hora de realizar una ejecución del diagrama de estado, un especio de memoria compartido para todos los documentos del proyecto. Este espacio de memoria, se genera para que todos los elementos que posee un documento tengan donde almacenar sus datos, y puedan acceder a los datos de otros elementos, dando igual en que documento estén. Es una solución sencilla para permitir que los elementos puedan interactuar a la hora de ejecutar un diagrama de estados.

La parte de comunicación OPC se ha centralizado aquí también, así como las propiedades de la simulación. Con lo cual un proyecto posee información del servidor OPC al que conectarse, datos del tiempo base de reloj para la simulación, etc.

Por otro lado, el proyecto también posee información sobre los datos del proyecto, como título, dirección en disco de los ficheros que lo componen, etc.

2.2.2 Objeto Documento

2.2.2.1 Descripción breve del objeto documento.

El objeto documento, (Librería cDocumentOSM) es un objeto compuesto por dos partes, una parte visual (formulario) y una instancia de código no visual. Es el segundo nivel dentro de la jerarquía de elementos de un proyecto OPCSM, es quizás el más importante de todos, ya que posee toda la funcionalidad de edición para el usuario, pero no podría funcionar solo sin estar contenido en un proyecto, ya que se quedaría aislado del resto de documentos.

Al crear una instancia del objeto documento, se crea una hoja de edición en entorno gráfico, sobre la cual se pueden ejecutar acciones con el ratón y el teclado. De cara al usuario se muestra una especie de hoja tipo plano, con un cajetín predefinido que es posible cumplimentar mediante acciones sobre el documento. El tamaño de la hoja, el espaciado de la cuadrícula, entre otras propiedades son configurables, así pues este objeto es bastante moldeable para ajustarse a la necesidad del proyecto que se esté implementando.

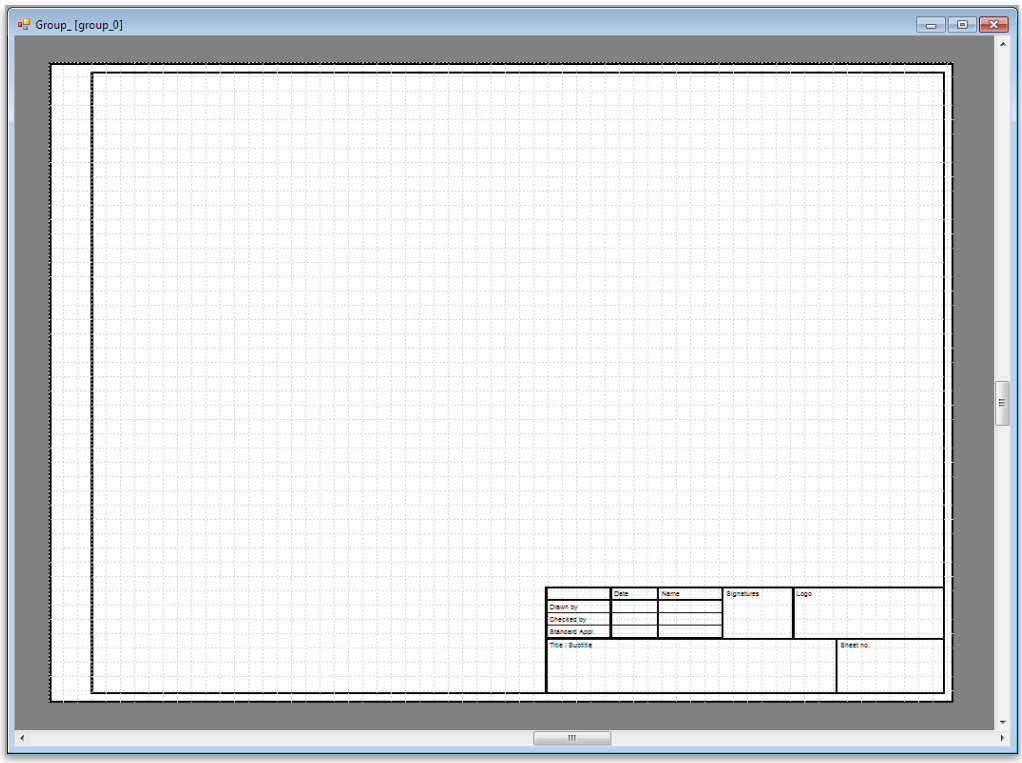


Fig. 2.2-1 – Objeto documento.

Por otro lado posee métodos para ayudar al usuario en la edición de los diagramas de estado, permite mover, copiar, cortar, borrar, seleccionar, hacer zoom, imprimir, etc. acciones que se pueden llevar a cabo referidas a los elementos contenidos en el documento.

La otra parte inseparable del objeto documento, contiene el motor de métodos y propiedades del "documento", entre otras funciones necesarias. Esta parte es un contenedor de elementos que siguen al documento en la jerarquía de la estructura del software, que son los elementos que forman los diagramas de estado. Al igual que el proyecto contenía documentos, estos contienen elementos.

El sistema gráfico no es más que un lienzo vacío en el que se van agregando elementos dibujados. La técnica de invalidado de área gráfica y el redibujado con doble-buffer, dan vida a este entorno. Ejemplo, si se desea mover un elemento pasando por encima de otros, no es necesario borrar estos cada vez y volverlos a redibujar, tan solo se vuelven a redibujar todos los elementos del área visible cada vez que algún elemento sea modificado e interfiera en el resto, de este modo se genera un efecto muy nítido y suave de movimiento. Para solicitar un redibujado de los elementos en pantalla, existe la acción de invalidar, indicando que el área ya no es adecuado y hay que redibujar todo el contenido del documento visible, en el área gráfica.

2.2.2.2 **Serialización de elementos.**

La serialización es el proceso de convertir un objeto en una secuencia de bytes para conservarlo en memoria, una base de datos o un archivo. Su propósito principal es guardar el estado de un objeto para poder crearlo de nuevo cuando se necesita. El proceso inverso se denomina deserialización. El objeto se serializa en una secuencia que, además de los datos, contiene información sobre el tipo de objeto, como la versión, referencia cultural y nombre de ensamblado. Esa secuencia se puede almacenar en una base de datos, un archivo o en memoria. (Fig. 2.2-2)

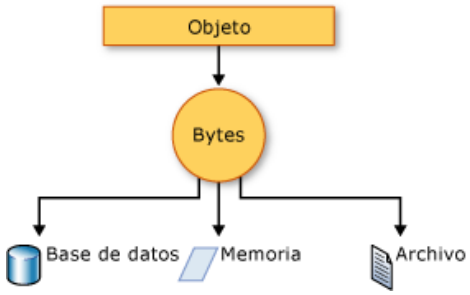


Fig. 2.2-2 – Esquema de una serialización

La serialización permite al desarrollador guardar el estado de un objeto y volver a crearlo cuando es necesario, y proporcionar almacenamiento de objetos e intercambio de datos. A través de la serialización, un desarrollador puede realizar acciones como enviar un objeto a una aplicación remota por medio de un servicio de red, pasar un objeto a disco directamente sin más, etc.

Todos los elementos contenidos en el documento están serializados, esto quiere decir que a la hora de realizar acciones tan simples desde el punto de vista del usuario, como copiar, cortar, pegar o guardar, y muy complejas desde el punto de vista de la programación, gracias a la serialización, todos los objetos que lo son, están preparados para ser gestionados en memoria fácilmente ya que al fin y al cabo, dichas acciones se basan en ayudarse de un espacio de memoria para poder realizar el duplicado y recuperación, o la salvaguarda de datos en disco, etc.

A la hora de serializar los objetos para este proyecto, se codifican en XML. Esto permite además obtener ficheros en disco fácilmente editables desde fuera del entorno de desarrollo OPCSM VAI, si en algún caso se necesita corregir alguna propiedad puntual de algún elemento, ya sea del propio proyecto o documento.

2.2.2.3 **Acciones y propiedades del objeto documento.**

Antes se ha comentado que el documento posee una serie de métodos, funciones y propiedades que pueden ser usadas para que el usuario pueda realizar la edición de un proyecto OPCSM. Este interfaz, formado por un conjunto de acciones que se llevan a cabo sobre el documento, existen gracias a un complicado motor de decisiones implementado en la parte visual del objeto documento. Este motor funciona en base a eventos sobre el ratón y el teclado que son interpretados dependiendo de unos estados, que van cambiando dependiendo de

las acciones de edición. De esta manera se puede ir deduciendo que es lo que el usuario quiere hacer en cada momento y facilitarle el trabajo, así pues es la aplicación la que se adapta al usuario y no al revés (Como puede ocurrir en otros entornos gráficos de desarrollo, donde la tarea de diseño se hace muy complicada para el usuario)

Para entenderlo, véase el ejemplo a continuación, teniendo en cuenta el siguiente entorno:

"Se desea mover un elemento contenido en el documento desde una posición a otra de la pantalla, teniendo en cuenta que también es posible realizar acción de selección de grupo de elementos en el documento."

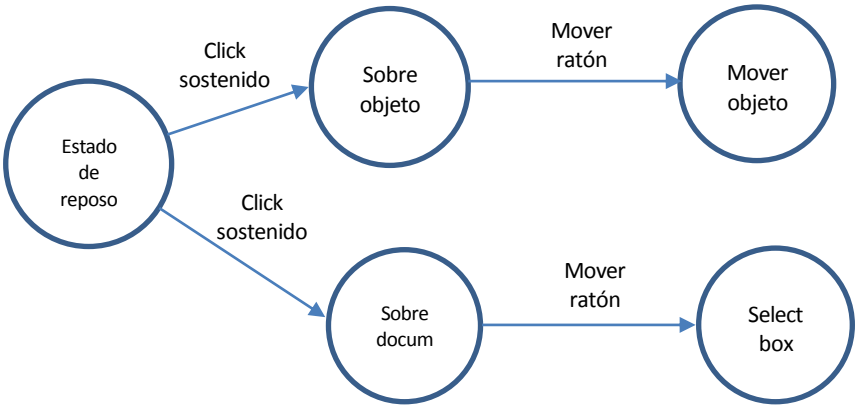


Fig. 2.2-3 – Ejemplo del motor de decisiones.

Para el ejemplo propuesto el motor de decisiones del objeto documento necesita conocer en qué momento de la edición se encuentra el usuario, es decir que estado está activo cada vez; Donde está el ratón, que botón se ha pulsado, sobre que elemento se ha pulsado, si se mueve con un botón pulsado, etc. Se puede entender el motor de decisiones como un autómata con distinto estados excitados por los eventos de teclado y ratón. (Ver fig. 2.2-3)

A continuación se hará una descripción de las acciones más relevantes del objeto documento, para ver las herramientas que se ponen a disposición del usuario en la tarea de implementación.

- Acción CREAR

  Es una acción esencial en la edición de un diagrama, permite añadir al documento elementos activos (Que se verán más adelante) para ir conformando el contenido del proyecto OPCSM. Esta acción, genera una instancia del elemento seleccionado, le da unas propiedades y lo almacena en el contenedor del documento, además le asigna un espacio de memoria. Una vez esto, el objeto aparecerá físicamente en el entorno gráfico del documento.

- Acción ELIMINAR

  Es otra de las acciones básicas, al contrario que la acción crear, esta elimina un elemento seleccionado del entorno gráfico, y posteriormente del contenedor de elementos del documento. A su vez, si este elemento está enlazado con otros, sus enlaces también son eliminados para no dejar residuos.

- Acción SELECCIONAR

  Permite decirle al interfaz de desarrollo que elemento o elementos se desean marcar para editar. Cuando se marca un elemento con el ratón y este es detectado como tal, se selecciona. También ocurre una selección, si se arrastra el ratón sobre el documento y se capturan aquellos que están dentro del recuadro formado, entonces estos serán seleccionados como un grupo. Esto se denomina SELECCIÓN MULTIPLE.

- Acción MOVER

  La acción mover desplaza un elemento de una posición (Xi, Yi) hacia otra posición (Xf, Yf) distinta. También es posible mover grupos de elementos con esta acción.

- Acción COPIAR/CORTAR/PEGAR

  Esta acción hace uso de la característica comentada anteriormente de serialización. Gracias a esta codificación de los elementos, es posible enviarlos a memoria y recuperarlos, con lo cual es posible duplicar elementos, al copiar un elemento en memoria y generarlo todas las veces que se desee en el documento. De manera similar funciona la acción "cortar".

- Acción CONECTAR

  Esto permite enlazar dos o más elementos, indicando al documento que existe una unión entre ellos, y una jerarquía padre-hijo. De este modo se puede formar un árbol de elementos, esencial para el motor de ejecución que recorrerá este árbol completamente ejecutando los elementos que posean un testigo.

  El testigo se va desplazando por cada uno de ellos siguiendo las flechas de conexión.

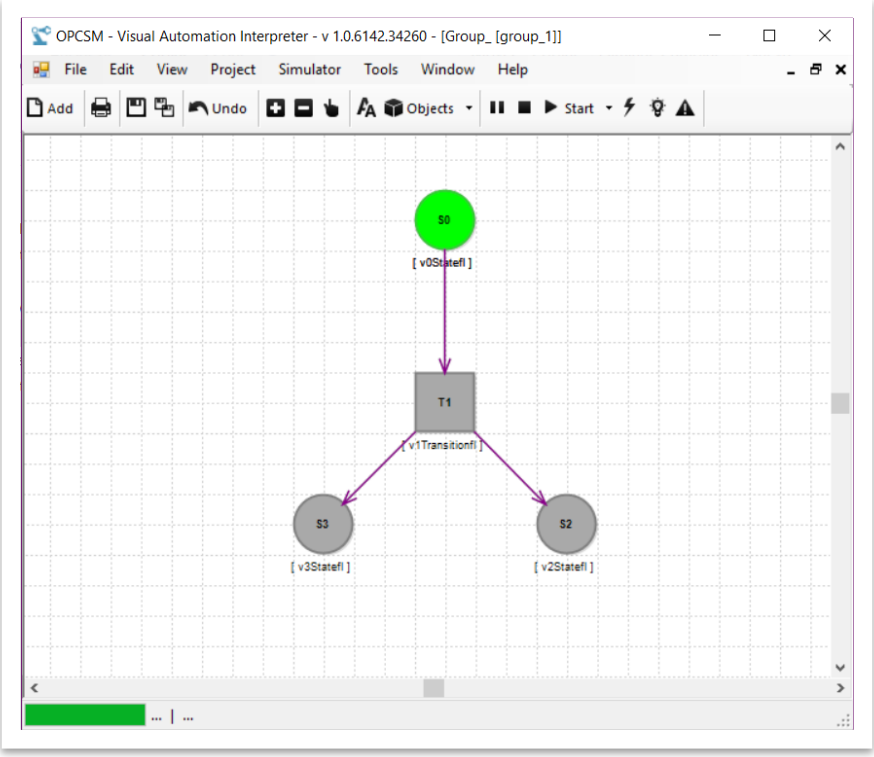


Fig. 2.2-4 – Detalle de una conexión de elementos

- Acción MODIFICAR CONEXIÓN

  Si se hace selección sobre una conexión, es posible editarla modificando a que elementos conecta, moviendo uno de sus extremos cada vez.

- Acción DESPLAZAR DOCUMENTO

  Sobre el documento también se pueden realizar acciones, entre ellas está el desplazamiento en la ventana del "papel". Es posible desplazarlo en horizontal o vertical, bien mediante unas barras de desplazamiento lateral o haciendo uso de la herramienta mano, que permite desplazamiento del "papel" mediante arrastre con el ratón.

- Acción ZOOM

  Esta acción permite tener una visión más general de un área del documento, o ver con más detalle algunos elementos focalizando una parte concreta del área visible. Haciendo en cada caso más

pequeños los elementos contenidos en el documento o más grandes, se crea la ilusión de zoom.

- Acción IMPRESIÓN

  Esta acción permite, gracias a la técnica de redibujado, re-direccionar todos los elementos contenidos en un documento, hacia la impresora, obteniendo un documento impreso en papel.

- Acción GUARDAR/CARGAR

  Esta acción permite almacenar todo el proyecto en disco, gracias al método de serializado de objetos. De igual manera es posible recuperar todo el proyecto mas tarde.

2.2.3 Elementos del documento

En este apartado se van a describir los distintos elementos existentes en la versión actual del software, disponibles para poder crear un proyecto OPCSM. Los elementos tienen un conjunto de propiedades comunes que los definen, como son posición en el documento, tipo de elemento, forma, colores, nombre, etc. Cada elemento posee además propiedades concretas que caracterizan a un elemento respecto a los demás. Por otro lado, los elementos tienen asociados unos métodos que son lanzados por el documento para dibujar el elemento, detectar que ha sido tocado por el ratón, etc. y que están definidos dentro de cada elemento. Además al convertirse en elementos activos cuando se ha iniciado una ejecución, cada elemento tiene definida una función como es contador, resolutor de expresiones matemáticas, elementos de control de flujo, elementos de escritura/lectura OPC, etc.

Los elementos están divididos en dos elementos esenciales, transiciones y estados. Las transiciones son elementos de control, mientras que los estados son elementos de entrada/salida/ejecución.

Todos los elementos estados y transiciones poseen una variable interna definida como "MyVariable", esto permite intercambiar los resultados de las operaciones una vez ejecutadas, con el resto de elementos. Esta variable pertenece al espacio de memoria compartido del proyecto.

De este modo por ejemplo, un contador es un elemento estado, que cada vez que tiene el testigo y es ejecutado incrementa en +1 su variable interna.

Los elementos deben ir siempre enlazados estado – transición – estado, es decir, no es posible unir dos elementos del mismo tipo base, por cada E/S hay un elemento de control.

2.2.3.1 Propiedades comunes de los elementos

Todos los elementos poseen una serie de propiedades comunes que serán descritas en este apartado. En la tabla siguiente se pueden ver tales propiedades divididas en secciones. Estas propiedades pertenecen a la librería común de todos los objetos, cObjectOSM.

Sección	**Propiedad**	**Valores / Campos**	**Descripción**
Main	Name	Cadena de caracteres y números	Nombre del elemento en el proyecto
	PID Label	Cadena de caracteres y números	Nombre P&ID
Labels	View name	True | False	Indica si se muestra o no el nombre en el documento
	View type id	True | False	Indica si se muestra o no el ID del elemento en el documento (El ID se genera de manera automática y no se puede modificar.

Others	Size	▪ Width ▪ Heigh	Ancho y alto del elemento.
My Variable	Initial value	Valor numérico	Valor inicial cargado en la variable interna
	Name	Cadena de caracteres y números	Nombre de la variable interna.
	Value	Valor numérico	Valor actual de la variable interna (Solo lectura)

2.2.3.2 Estado

Librería: cStateOSM | Tipo base: Estado

Es un elemento muy simple, su función es poner a "1" su variable interna mientras tiene el testigo.

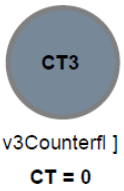
Cuando pierde el testigo, el valor de su variable interna cambia a "0".

2.2.3.3 Transición

Librería: cTransitionOSM | Tipo base: Transición

Es un elemento de paso. Su condición siempre es TRUE, es decir que no condiciona el paso del testigo entre dos elementos de tipo estado.

2.2.3.4 Contador



Librería: cCounterOSM | Tipo base: Estado

Es un elemento cuya función es contar, en incremento o decremento, en 1 unidad, cuando recibe el testigo y por cada flanco de reloj. El resultado del contador es actualizado en cada flanco en su variable interna.

Propiedad	**Valores / Campos**	**Descripción**
CounterLoad		Valor inicial del contador
CounterType	counterUp counterDown	Indica el modo de funcionamiento de contador, en incremento (counterUp) o decremeneto (counterDown)

2.2.3.5 Expresion

Librería: cExpressionOSM | Tipo base: Estado

Elemento para calcular expresiones matemáticas. Este objeto está implementado en base a una librería matemática, denominada "Mathos.Parser", *MathParser.dll*

El elemento "Expression" usa una serie de operadores matemáticos y variables. Se ha de definir las variables para que el elemento la reconozca en la expresión. Para editar la expresión se hace uso del editor de código.

Operadores	**+ , - , * , / , % , ( ) ,** ^ **, : , < , > , = ,** & **, |**
Comandos	cos(n), cosh(n), arcos(n), sin(n), sinh(n), arcsin(h), tan(n), tanh(n), arctan(n), sqrt(n), rem(n), root(n), pow(n), exp(n), abs(n), log(n), round(n), truncate(n), floor(n), ceiling(n), sign(n)
Meta-comandos	define{…} constant{…} function{…}
Comentarios	// texto …

Expresión (1)

Ejemplo de expresión matemática sencilla, multiplicar por dos el valor del contador "v2Counterfl", el resultado se almacenará en la variable interna del elemento expresión (No es necesario definirla si no se usa como valor de una expresión)

```
//
//Expression testing. Code sample.
//

define{v2Counterfl}
function
{
    v2Counterfl * 2
}
```

Expresion (2)

La expresión a continuación, realiza dos cálculos matemáticos en una misma pasada de reloj, y además el resultado de la variable interna de "v0Expressionfl" pasa a "v10Variablefl". Obsérvese que para las expresiones primeras, "v0Expressionfl + 2" y "v0Expressionfl * 2" al no existir asignación definida, se interpreta como una asignación directa a la variable interna del elemento expresión, en este caso, "v0Expressionfl", lo mismo ocurre con la expresión "cos(v10Variablefl)" que una vez calculada es enviado su resultado a "v0Expressionfl" al no haber asignación definida.

```
//
//Expression testing. Code sample 2.
//
//multi-line expression

define{v0Expressionfl, v10Variablefl}
function
{
    v0Expressionfl + 2;
    v0Expressionfl * 2;
    v10Variablefl = v0Expressionfl;
    cos(v10Variablefl);
}
```

Expresion (3)

Uso de variables auxiliares, para continuar realizando cálculos intermedios. Una vez pasado el resultado de "v0Expressionfl" a "v10Variablefl" es posible calcular por separado el coseno de "v10Variablefl" afectando el resultado a "v10Variablefl"

```
//
//Expression testing. Code sample 3.
//
//Sample multi-line expression. Assignation defined.

define{v0Expressionfl, v10Variablefl}
function
{
    v0Expressionfl + 2;
    v0Expressionfl * 2;
    v10Variablefl = v0Expressionfl;
    v10Variablefl = cos(v10Variablefl);
}
```

2.2.3.6 **Variable**

Librería: cVariableOSM | Tipo base: Estado

Este elemento permite realizar asignación desde otras variables, valores definidos, etc. Tiene varios modos de funcionamiento que se pueden ver en la tabla siguiente.

Propiedad	**Valores / Campos**	**Descripción**
Variable mode	Assignation Variable WriteToMyVariable WriteToTarget DefineTarget ResetTargetValue	**Assignation**: Se asigna a "Variable target" el valor de "Variable source" **Variable**: Indica que la variable se usa en código de una expresión como variable intermedia. **WriteToMyVariable**: Asigna a "MyVariable" el valor de "Variable target". **WriteToTarget**: Asigna a "Variable target" el valor que contiene "Variable source" **DefineTarget**: Asigna un valor constante definido por "Variable value" a "Variable target" **ResetTargetValue**: Se asigna el valor de "initial value" [My Variable] a "Variable target"
Variable source		Define una variable que se usará en los modos anteriormente descritos
Variable target		Define una variable que se usará en los modos anteriormente descritos
Variable value		Define un valor que se usará en los modos anteriormente descritos

2.2.3.7 Condición

Librería: cConditionOSM | Tipo base: Transición

Es un elemento control de flujo que permite el paso del testigo en base expresiones lógicas con resultado verdadero, "1". A diferencia de "Expression", el elemento "Condition" no permite evaluar varias líneas a la vez, solamente una expresión lógica formada por operadores y variables.

Operadores	<, >, =, <=, >=, &, | , not()
Meta-comandos	define{…} constant{…} function{…}
Comentarios	// texto …

Ejemplo de expresión lógica del elemento "Condition" escrita con el editor de código.

```
//
//Logical expression

define{v2Counterfl, v3Counterfl}
function
{
    (v2Counterfl > 2) | (v3Counterfl < 3)
}
```

2.2.3.8 Temporizador

Librería: cTimerOSM | Tipo base: Transición

Es un elemento de control de flujo que permite el paso del testigo cuando ha cumplido la temporización programada. La base de tiempo viene dada por el tiempo de reloj del motor de ejecución (Se observa que el elemento indica el tiempo total que temporiza cuando se ha configurado. TI = 10 (5s))

2.2.3.9 Parada

Librería: cStopOSM | Tipo base: Estado

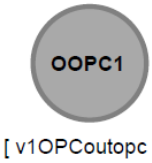
Este elemento no deja avanzar el testigo en la rama en la que está ubicado.

2.2.3.10 Finalización

Librería: cEndOSM | Tipo base: Estado

Este elemento finaliza la ejecución del sistema parando el motor de ejecución.

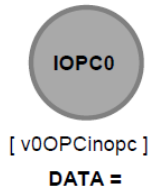
2.2.3.11 **Salida OPC**



Librería: cOPCoutOSM | Tipo base: Estado

Este elemento permite escribir el valor de la "Variable source" en la ruta del servidor OPC definido por "OPC tag". Enviar un valor desde el sistema al servidor OPC.

2.2.3.12 **Entrada OPC**



Librería: cOPCinOSM | Tipo base: Estado

Este elemento permite escribir el valor obtenido de la ruta del servidor OPC definido por "OPC tag", y almacenarlo en la variable interna. Entrada de un valor desde un servidor OPC hacia el sistema.

2.2.3.13 **Objetos interactivos.**

Estos objetos tienen un funcionamiento asíncrono, ya que pueden ser actuados en cualquier instante, aunque el valor que envían al ser actuados, está sincronizado con el reloj de ejecución. Uno de los elementos básicos es el botón, que se describe a continuación.

Librería: cButtonOSM | Tipo base: Estado

Este elemento envía un "1" a su variable interna, cuando es actuado por el usuario y mientras está activo. En el momento que no está actuado, está almacenando un "0". Si se aplica un doble-click al botón se produce el anclado del botón a "1" sin necesidad de seguir pulsándolo.

2.2.3.14 **Objetos de indicación.**

Permiten representar el valor de "Variable source", mediante objeto gráfico.

Librería: cDisplayOSM | Tipo: Interactivo

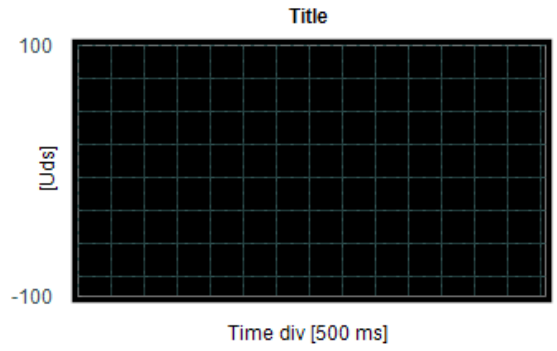
Este elemento posee varias propiedades descritas a continuación, que permiten personalizar la representación del dato que muestra.

Propiedad	**Valores / Campos**	**Descripción**
Mode	Binary Multivalue Numerical	**Binary**: Representa "ON" con "1" y "OFF" con "0" **Multivalue**: (No definido aun) **Numerical**: Representa valores numéricos mas una unidad a la derecha del número. Ejemplo: 120 Kg

Num. Type	Decimal Integer Engineer Binary8 Binary16 Binary32 Binary64 Binary128 Binary256 Hexadecimal Time String	Tipos para representar un valor en modo “Numerical”. NOTA: El tipo “Hexadecimal” no está aun definido.
Unit		Cadena de caracteres que representa una unidad de medida definida por el usuario.
Value		Valor absoluto representado en el indicador
Font size		Tamaño de la fuente del indicador

2.2.3.15 **Objeto gráfico**

Permite representar valores de la variable *source* en una gráfica a lo largo del tiempo.



Librería: cGraphicsOSM | Tipo: Interactivo

Este elemento posee varias propiedades descritas a continuación, que permiten personalizar el objeto gráfico.

Sección	**Propiedad**	**Descripción**
Graphic	Autoscale	Actualiza la escala del eje Y de manera automática
	Main title	Título superior del gráfico
	Max value scale	Valor máximo del eje Y, cuando autoscale = false
	Min value scale	Valor mínimo del eje Y, cuando autoscale = false
	Y-axis unit	Unidad para mostrar en el eje Y

2.2.3.16 **Objetos de etiquetado.**

Estos objetos no realizan ninguna función durante la ejecución, son elementos únicamente de etiquetado, para poner texto en el sistema. No necesitan mayor mención en este documento.

Text_label_0

Librería: cTextOSM | Tipo: Interactivo

Este elemento posee varias propiedades descritas a continuación, que permiten personalizar la etiqueta.

Sección	**Propiedad**	**Descripción**
Color	Background color	Color para el fondo de la etiqueta
	Text color	Color para el texto de la etiqueta
Design	Show background	Indica si se visualiza el color de fondo o no
	Show border	Indica si se visualiza el borde de la caja
Font	Font name	Fuente elegida
	Font size	Tamaño de la fuente
Text	Align	Tipo de alineado del texto en la caja
	Text	Texto a mostrar en la etiqueta

2.2.4 Sistema de intercambio de datos

OPCSM VAI hace uso de un sistema de memoria compartida entre todos los documentos, esta memoria es un contenedor de variables tipo cadena, que apuntan a cada elemento activo del proyecto (Elementos como estados, transiciones, expresiones, contadores, etc). Gracias a esta memoria compartida, todos los elementos pueden leer y escribir de/en cualquier elemento de cualquier documento, de manera muy sencilla.

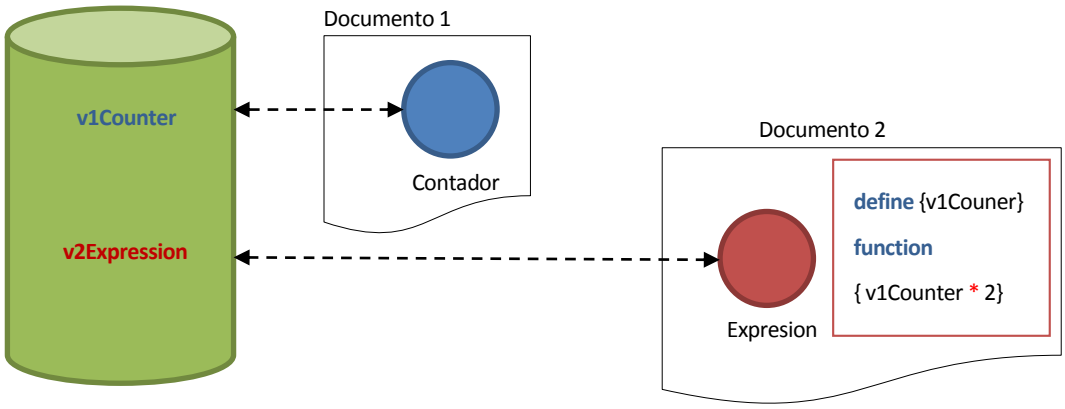


Fig. 2.2-5 – Esquema del sistema de memoria compartida.

Al crear un proyecto se genera automáticamente esta memoria de intercambio y a medida que se van creando elementos en el documento se genera en ella un espacio asignado a cada uno de estos. A dicha asignación de memoria se denomina “variable interna” y que se puede encontrar en las propiedades del elemento como “My Variable”.

La memoria de intercambio es otro objeto, formado por propiedades y métodos. El objetivo es hacer sencillo la lectura y escritura de información en dicha memoria. Para ello entre otras funciones existen dos comandos, "write" y "read", a los cuales se le pasa como parámetro el nombre de la variable de la cual queremos leer o en la cual queremos escribir.

Librería: cMemOSM

Funcion	**Parámetros**	**Descripción**
Write (Nombre variable, Dato)	Nombre variable Dato	Escribe "Dato" en la memoria asignada a "Nombre variable"
Read (Nombre variable)	Nombre variable	Devuelve el dato leído de la memoria asignada a "Nombre variable"

2.2.5 Entorno de desarrollo

Todo el conjunto de objetos descritos en los apartados anteriores, por si solos no tienen utilidad, ya que deben estar integrados en un software para poder hacer uso de su funcionalidad y permitir que puedan interactuar en conjunto. Para ello se ha desarrollado OPCSM VAI, un entorno que contiene una serie de herramientas para facilitar la labor de implementación del proyecto OPCSM. Este es entonces el programa principal desde el punto de vista del usuario, para crear proyectos, añadir documentos al proyecto, añadir elementos a los documentos, preparar el sistema en cuanto a etiquetado, editarlo, ejecutarlo, depurarlo, etc. En los apartados siguientes se dará a conocer el entorno para familiarizarse con él y entender cómo funciona, aunque ya se ha hecho un análisis en mayor detalle en apartados anteriores de los módulos que conforman dicho software, en esta sección da una visión de todos los módulos conectados.

2.2.5.1 Área de edición

En el momento que se crea un proyecto, se habilita el área de edición y se añade un documento para comenzar a trabajar. En este instante proyecto, documento y elementos están enlazados para poder implementar los sistemas de estado. El área de edición es puramente un contenedor MDI (Multi-Document Interface) que permite ir agregando ventanas hijas a la ventana padre (Definida por el programa principal), esta forma de contener los documentos del proyecto permite tener organizado todo el conjunto de ventanas de una forma más adecuada.

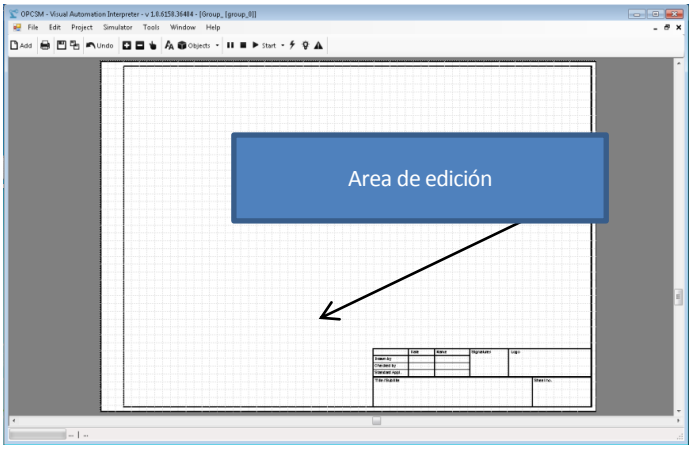


Fig. 2.2-6 – Área de edición – OPCSM VAI

En el área de edición (*figura 2.2-6*) se llevan a cabo acciones sobre los documentos. Toda la funcionalidad la proporciona el interfaz gráfico de dicho documento ya que el área de edición no es más que eso, un contenedor que agrupa todos los documentos del proyecto en curso.

El programa principal interactúa con el documento activo, el documento activo es aquel que el usuario tiene en el nivel superior de la vista, y para hacerlo activo es tan simple como hacer click en la ventana. Cuando es detectado se pueden realizar sobre acciones de edición, como interactuar con los elementos de su interior, agregar nuevos elementos, guardar el documento, imprimir, ver sus propiedades, etc. Todo este interfaz adaptado a las condiciones establecidas en cada momento, permite facilitar al usuario la tarea de edición, frente a otros programas para los que no se puede intuir el funcionamiento, y entonces eso hace complicar su uso.

Esta premisa ha sido muy tenida en cuenta en el desarrollo del software, porque la experiencia final del usuario es importante, tanto como la utilidad de la herramienta y lo versátil que pueda llegar a ser.

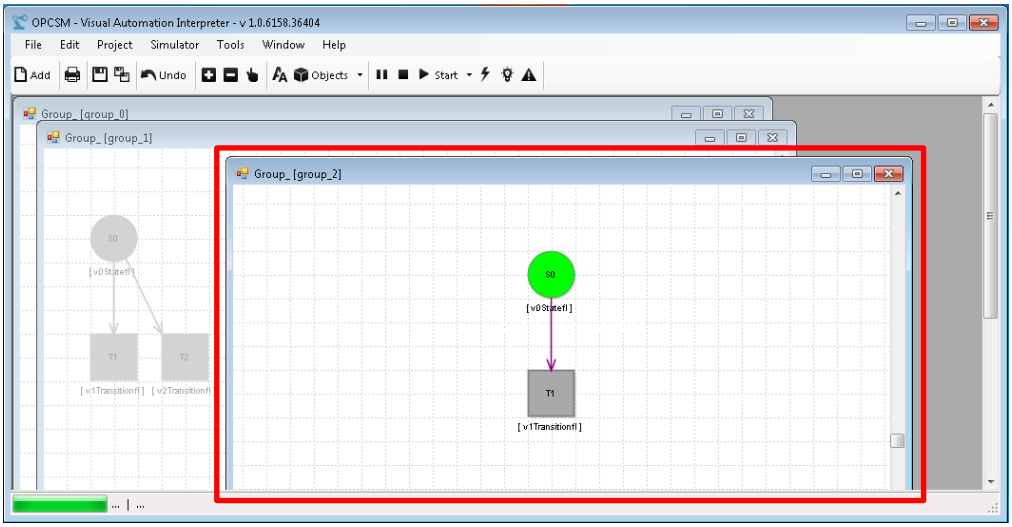


Fig. 2.2-7 – Representación de varios documentos y el documento activo.

En la *figura 2.2-7* se puede observar el documento activo marcado en rojo (La marca es una indicación de este documento, no aparece visible en el programa)

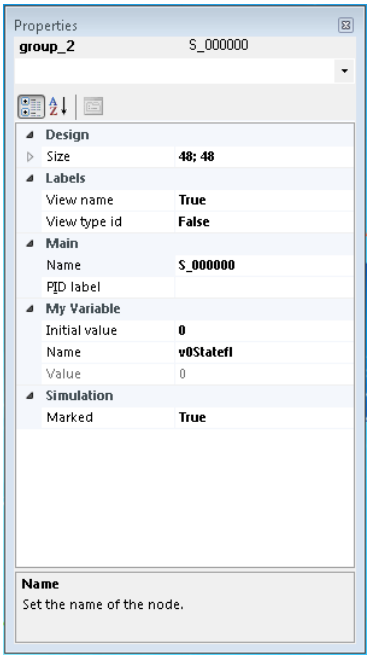
2.2.5.2 **Módulos extras**

El área de edición queda complementada con un conjunto de ventanas que añaden utilidad al conjunto ya disponible de herramientas de las que dispone el entorno para el desarrollo de los proyectos OPCSM. Estos elementos concretamente son:

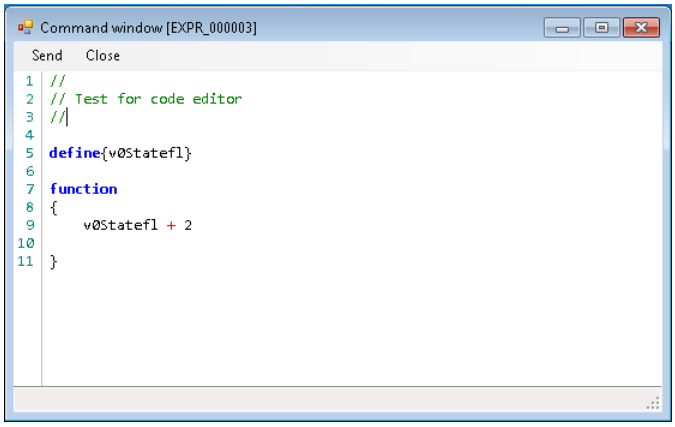
- La ventana de propiedades
- El editor de código
- La ventana de depuración
- La ventana de errores.

Entre ellas la ventana propiedades es concretamente un objeto propio del entorno .NET, multipropósito que permite directamente mostrar las propiedades que han sido definidas en un objeto a nivel de código, previo formateo de los datos que se desea mostrar, es posible obtener una sencilla herramienta con la cual caracterizar los elementos y de fácil uso, evitando tener que programar un objeto específicamente diseñado para tal labor. A continuación es posible ver en detalle dicha ventana, además de la ventana de edición de código, las cuales merecen una especial mención, pues quizás son las de mayor frecuencia de uso.

Ventana de propiedades – Muestra todas las propiedades de un elemento del documento o las propiedades comunes de un grupo de elementos. También muestra las propiedades del propio documento o del proyecto. De esta manera se puede editar el objeto fácilmente y cambiar sus propiedades, acercándose al estilo propio de un entorno de desarrollo software con el que quizás se esté más familiarizado.



Editor de código – El editor de código permite definir funciones matemáticas y expresiones lógicas para los elementos “expression” y “condition”.



2.2.5.3 **Motor de ejecución**

El motor de ejecución es una herramienta importante dentro del software, ya que permite ejecutar el sistema de estados que se ha creado. El funcionamiento de este motor está basado en un reloj, cuyo periodo queda configurado en las propiedades del proyecto. Por cada flanco de reloj se ejecuta una rutina que hace avanzar un testigo en base a unas premisas que dependen de la conexión entre nodos y de las condiciones de avance (Por ejemplo como el caso del elemento “condition” que dejará pasar el testigo si la expresión lógica resulta en “1”)

El motor de ejecución es un intérprete, ya que no compila a código nativo el sistema implementado antes de ejecutarlo, tan solo lo va interpretando sobre la marcha. El motor de ejecución no es un sistema de tiempo real, principalmente porque se ejecuta sobre una plataforma que no cumple las premisas de un sistema en tiempo real. En futuras mejoras se puede crear un motor sobre otra plataforma y permitir ejecutar el sistema con condiciones de tiempo real, esto por supuesto llevará entonces una fase de compilado, pero de este modo podrá ser ejecutado en un microcontrolador, sistema Linux/Unix, etc.

El motor de ejecución del intérprete de OPCSM VAI posee dos modos de funcionamiento:

- **Modo Normal** – Por cada flanco de reloj se disparan los nodos tipo estado (State, Expression, Counter, Variable, etc) que poseen el testigo y avanzarán si la expresión lógica de los elementos tipo transición "hijos" (Colgados de él) da como resultado "1"

- **Modo Petri** – Por cada flanco de reloj se disparan todas las transiciones y solo se deja pasar el testigo cuando la expresión lógica de la transición resulta en "1"

El motor de ejecución en modo Normal,

1. Se realiza una preparación previa del sistema de estados. Se hace una "foto" del estado de partida, que nodos poseen testigo, etc.

   Comienza la ejecución

   (Por cada flanco de reloj)

2. Se escanean todos los nodos del documento, y todos los documentos del proyecto, en busca de aquellos que poseen un testigo de ejecución (marcador).
3. Se lanza un trigger sobre dichos elementos (método que poseen todos los elementos activos)
4. Se espera el resultado de procesar el trigger y ver si es satisfactorio o por el contrario hay un error.
5. En el caso de ser satisfactorio, se localiza el/los nodos hijos enganchados al nodo que posee el testigo, y se avanza el testigo hasta ellos, a la espera del siguiente flanco de reloj.
6. Se para el proceso si hay algún elemento "END" en otro caso se continua otra vez por el punto (2)

Cuando se detiene el programa, se vuelve a la configuración inicial (Guardada por la "foto" realizada al principio de la ejecución)

Al no ser un sistema en tiempo real, no se asegura un tiempo de ejecución máximo hasta el próximo flanco de reloj, es decir, esto es lo que ocurre en la ejecución (Ver imagen abajo)

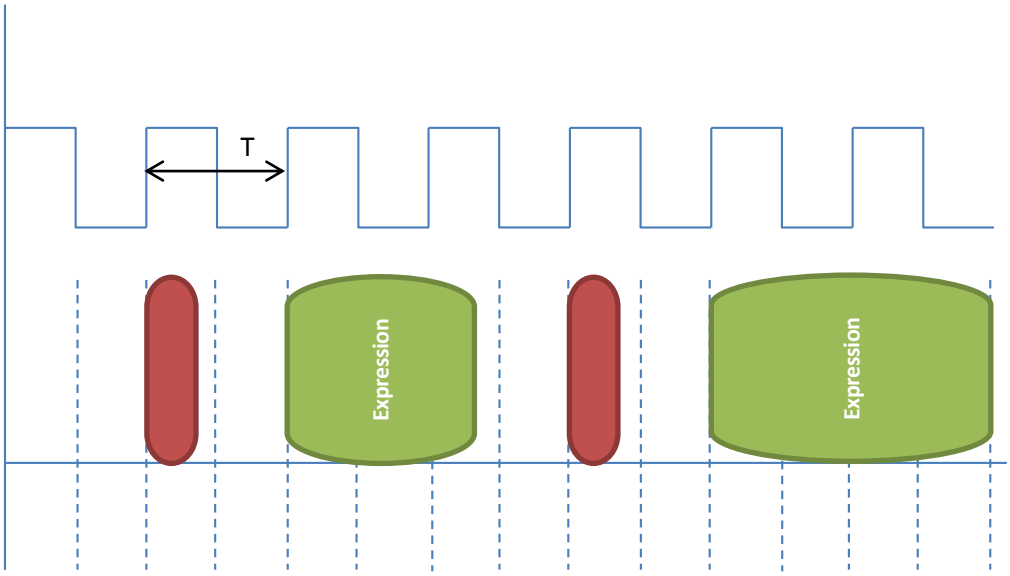


Fig. 2.2-8 – Representación en tiempo del disparo de un elemento STATE y un elemento EXPRESSION.

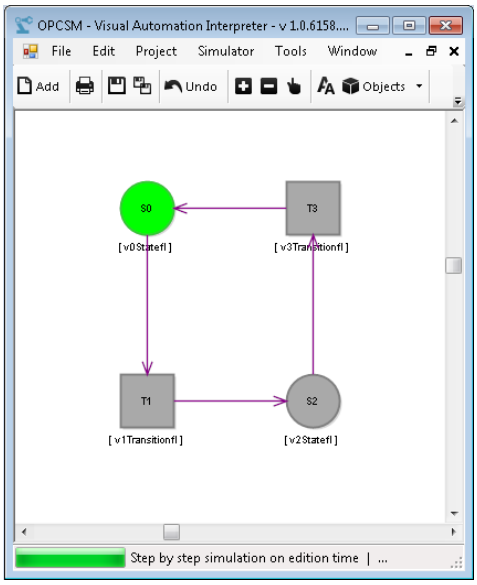
Se define en el diagrama de la *figura 2.2-8* en color burdeos el disparo y ejecución de un elemento "STATE" y en color verde el disparo y ejecución de un elemento "EXPRESSION" con una expresión matemática compleja. Se puede observar que la ejecución del elemento "STATE" se hace antes de llegar el siguiente flanco de reloj, pero vemos que el elemento "EXPRESSION" sin embargo usa mas de un flanco de reloj además depende de la complejidad de la expresión matemática definida y de los valores para los que calcularla, con lo que no es posible asegurar que la ejecución de la expresión se pueda hacer en un tiempo configurado fijo, es decir no se sabe con exactitud cuándo acabará la ejecución.

- Ejemplo de funcionamiento paso a paso del motor de ejecución,

Paso 1

El motor de ejecución es activado por un flanco de reloj.

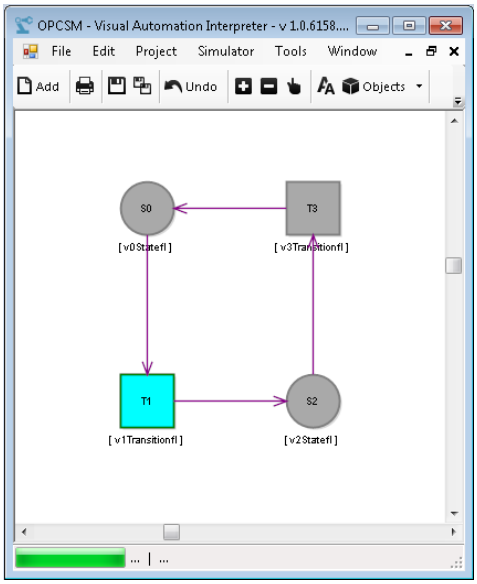
Toma una foto del estado actual del sistema. En este caso se almacena que S0 tiene inicialmente el testigo.



Paso 2

El motor de ejecución es activado por un flanco de reloj.

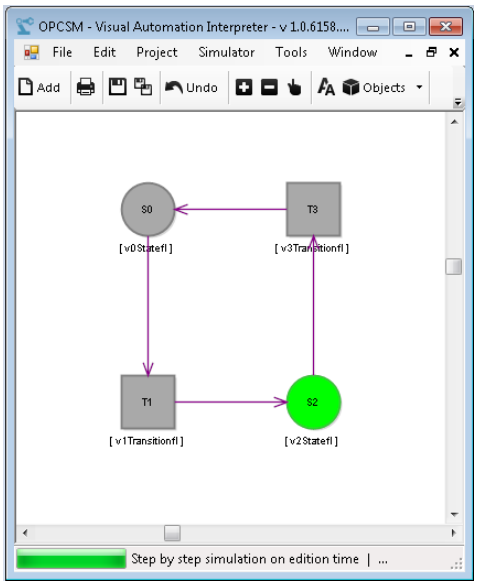
Se recorren todos los nodos, y se disparan (Lanza un comando trigger) los nodos que llevan el testigo. Una vez ejecutado se le quita el testigo y se pasa a la transición que cuelga del nodo. En este caso la transición a la que debe pasar es T1.



Paso 3

El motor de ejecución es activado por un flanco de reloj.

El motor de ejecución recorre todas las transiciones y dispara (Lanza un comando trigger) aquellas que llevan el testigo. Una vez ejecutado se le quita el testigo y se pasa al nodo que cuelga de la transición. En este caso es el nodo S2 el que adquiere el testigo



3 INTEGRACIÓN OPC

La integración del estándar OPC es el valor añadido que se le ha dado a este software frente a otros. El objetivo era conseguir que los sistemas diseñados, pudiesen interactuar con el mundo exterior (Autómatas - PLC) mediante un servidor OPC. Para ello, a la librería "cProyectOSM" (Objeto proyecto de OPCSM VAI) se le implementó todo lo necesario para conectar el entorno de trabajo a un servidor OPC y tener disponibles todos sus TAGs en todos los elementos OPC de cada documento. Además se desarrollan elementos de E/S para asignar dichos TAGs OPC al sistema de estados. Así pues, al interpretar el sistema de estados, es posible enviar y recibir datos desde la planta y automatizar procesos reales.

3.1 Libreria OPC

El estándar aplicado a OPCSM VAI está basado en una librería de OPC Foundation denominada "Interop.OPCAutomation.dll". Esta librería se encuentra en la SDK, "OPC Core Components Redistributable" necesario para poder hacer que el software se entienda con servidores OPC.

La librería "Interop.OPCAutomation" es cargada directamente en todas las librerías que necesitan del estándar OPC, como son "outOPC", "inOPC", y en el programa principal OPCSM VAI. El resto de librerías, que deben estar instaladas en "C:\Windows\System32" son necesarias para reconocer servidores OPC, como ocurre con la librería "OCPenum.dll", que no se integra como tal en el software si no que se llama indirectamente desde "Interop.OPCAutomation", por ello debe estar instalada y registrada en el sistema operativo. A la librería de acceso a OPC, se le denomina *wrapper*.

La librería OPC está desarrollada en base a una especificación de OPC foundation, "Data Access Automation Interface Standard v2.02", es una versión antigua del estándar OPC, pero es más que suficiente para cubrir la necesidad de OPCSM VAI.

El objetivo de esta librería es permitir que OPCSM VAI pueda interactuar con los servidores OPC, sea cual sea, ya que todos siguen el mismo estándar. Se puede observar en la *figura 3.1* la jerarquía de elementos que componen un servidor OPC.

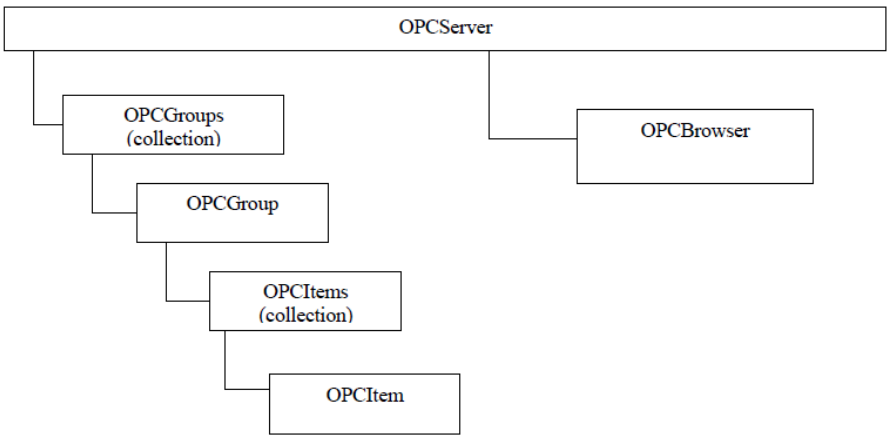


Fig. 3-1 – Modelo de un servidor OPC. Jerarquía.

En él se observa que hay una colección de grupos compuesta por grupos, una colección de ítems que cuelgan de cada grupo compuesta por ítems. Por otro lado tenemos un módulo de representación, OPCBrowser. Con este elemento es posible obtener toda la jerarquía existente en un servidor OPC, de este modo se consigue obtener el conjunto de rutas de elementos OPC (TAGs) para tenerlos disponibles en el proyecto y permitir que los elementos InOPC y OutOPC se conecten a ellos para interactuar con los PLCs de la planta real.

3.2 Integración OPC en OPCSM VAI

Se ha comentado anteriormente que OPCSM VAI necesita de un wrapper para conectar con los servidores OPC. En este caso se optó por el wrapper de OPC foundation, aunque hay otros muchos de uso libre. Esta librería se usa en el programa principal OPCSM VAI, con el objetivo de gestionar todo lo relacionado con los accesos OPC y la conexión con el servidor.

OPCSM VAI, hace uso de la librería OPC para poder detectar los servidores OPC disponibles en la red, y conectarse al servidor seleccionado para poder realizar más tarde las acciones E/S sobre este. Por otro lado, dos elementos dedicados, InOPC y OutOPC, hacen también uso de dicha librería, con el objetivo de leer o escribir en las rutas OPC seleccionadas por estos elementos en la fase de diseño del proyecto OPCSM.

Para hace uso de todo el sistema OPC, primeramente es necesaria una conexión. El programa principal hace uso de un módulo de gestión que enumera los servidores OPC, y el usuario selecciona uno de ellos. Esta información se almacena en las propiedades del proyecto para futuras cargas del proyecto (Realizando una reconexión automática)

En el código a continuación, se denomina *opcwrap* a la instancia del wrapper. Este código está disponible en el módulo de gestión OPC, *OPCManager_window* (Módulo que usa OPCSM VAI para gestionar los servidores y conexiones OPC)

```
private void LoadOPCServerList()
{
     lstOPCServerAvailable.Items.Clear();
     // Obtenemos todos los servidores OPC

     try
     {
         foreach (string s in opcwrap.GetServerList(txtMachineIP.Text))
         {
             // Servidores OPC
             opcwrap.ServerID = s;
             lstOPCServerAvailable.Items.Add(s);
         }
     }
     catch (Exception ex)
     { }
}
…
…

private void lstOPCServerAvailable_DoubleClick(object sender, EventArgs e)
{
    // Seleccionamos el servidor y conectamos
    // Con este procedimiento seleccionamos un servidor y realizamos el enlace
    // que se pasa directamente al programa principal para que todos los elementos
    // tengan disponible dicho enlace.

    try
    {
        ConnectToServer((string)lstOPCServerAvailable.SelectedItems[0]);
        //
```

```
        frmMain.MainProject.OPCServerConnection = (string)lstOPCServerAvailable.SelectedItems[0];

        frmMain.MainProject.OPCServerConnectionIP = txtMachineIP.Text;


        // Ahora reactivamos todos los elementos OPC. Esto lo hacemos para recordarle a los
        // elementos OPC que tienen una conexión con un servidor disponible para E/S.
        // Para ello recorremos todos los elementos existentes en el proyecto y a los tipo OPC
        // le pasamos el enlace del wrapper, propiedad "OPCWrapper" del elemento InOPC, OutOPC

        foreach (cDesignerOSM ob in frmMain.MainProject.Container)
        {
            foreach (Tamairon.OPCStateMatic.Elements.cObjBaseOSM el in ob.Document.Container)
            {
                if (el.NodeType == Tamairon.OPCStateMatic.Elements.ElementTypeOSM.OPCin_opc |
el.NodeType == Tamairon.OPCStateMatic.Elements.ElementTypeOSM.OPCout_opc)
                {
                    // Ya está activado. Ahora hay que conectarlo
                    el.ConnectedToOPCServer = true;

                    // Esto fuerza a generar la conexion permitiendo crear el grupo al
                    // conectar. Pasamos el wrapper al programa principal
                    el.OPCWrapper = frmMain.opcwrap;
                    // Recargamos la señal
                    string signal = el.OPCSignalToWrite;
                    el.OPCSignalToWrite = signal;
                    //
                    string signal2 = el.OPCSignalToRead;
                    el.OPCSignalToRead = signal2;

                }
            }
        }
    }
    catch
    { }
}
```

Una vez generado el enlace, cada vez que se agregue un elemento OPC tipo InOPC o OutOPC a un documento del proyecto, a este se le asignará el wrapper enlazado con el servidor OPC, para que pueda listar todas los TAGs disponibles en este, y poder de manera asignárselo al elemento OPC. De esta manera se tendría operativo el acceso a la variable OPC que hay en el servidor para leer o escribir desde el nodo OPC.

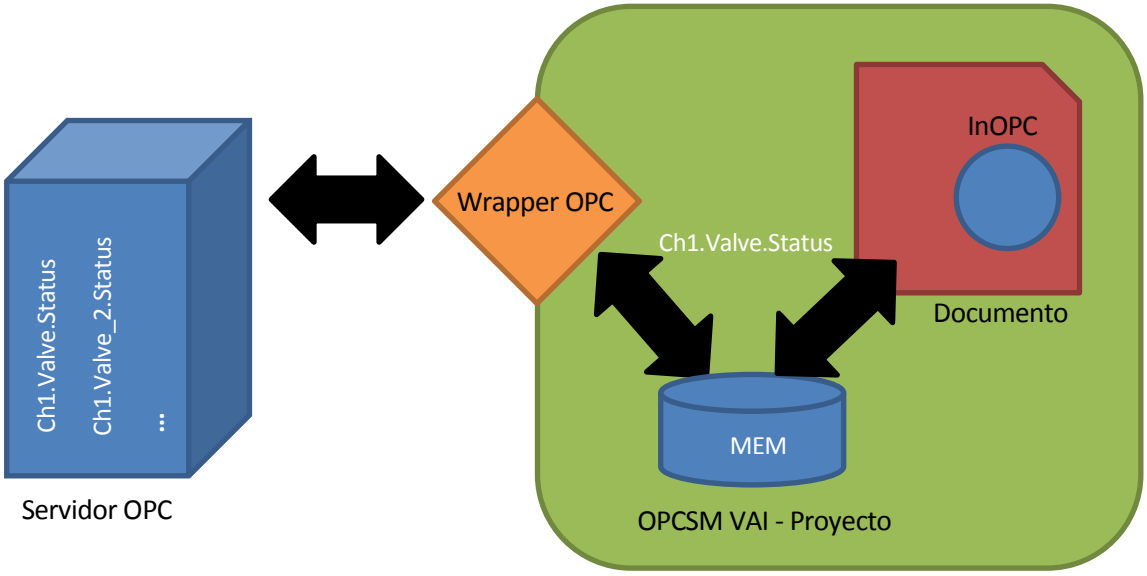


Fig. 3-2 – Detalle de enlace OPC desde nodos de un documento.

Cuando un elemento OPC tiene asignado un TAG, este se habilita en el subscriptor del servidor OPC, es decir, se le indica al servidor OPC que cierta variable va a ser leída/escrita y además que debe notificar cuando se modifique; Entonces el servidor accede a ella cuando esta variable cambia y notifica cuando otro proceso la modifique. De esta manera se evita leer todas las variables existentes en el servidor OPC cada vez, si realmente después no van a ser usadas todas, optimizando el acceso.

En el detalle de la *figura 3-2* se puede observar un enlace con el servidor OPC. El nodo InOPC tiene el TAG "Ch1.Valve.Status" de tipo bool asignado. En el modo ejecución, InOPC leerá el valor que proporciona el TAG indirectamente mediante la memoria de intercambio de OPCSM VAI, porque en realidad el wrapper accede al servidor cuando hay cambios en los TAGs habilitados para ser gestionados, y guarda el valor de cada uno de ellos en la variable interna del elemento OPC que tiene asignado ese TAG en el proyecto OPCSM. Todo esto se lleva a cabo con el código a continuación, procedimiento al que llama el subscriptor cuando tiene que notificar cambios en variables.

```
public void OnDataChange(int TransactionID, int NumItems, ref Array ClientHandles, ref Array
ItemValues, ref Array Qualities, ref Array TimeStamps)
{
    if (NumItems > 0)
    {
        for (int i = 1; i <= NumItems; i++)
        {
            // Se leen todos los items que han cambiado en el servidor
            // y que son asignados a los elementos OPC. Se actualizan en su
            // posición de memoria correspondiente.
            string valorOPC = ItemValues.GetValue(i).ToString();
            memShared.SetValueInPosition((int)ClientHandles.GetValue(i), valorOPC);
        }
    }
}
```

El procedimiento "OnDataChange", es un evento propio del wrapper OPC, que se dispara de manera automática cuando hay cambios en las variables habilitadas en el servidor OPC. Cuando ocurre esto, aprovechamos para leer el valor que ha cambiado en cada uno de los TAGs habilitados y actualizar la memoria interna de sus elementos OPC correspondientes, así cuando lo deseen los elementos InOPC podrán leer dicha información desde su variable interna.

Realizando de este modo la operación de acceso al servidor, se evita colapsarlo ya que por cada lectura síncrona que se realizase el servidor tendría que hace un acceso exclusivo a la variable y en un número considerable de variables se perdería eficiencia, ralentizando el programa.

En el caso de la escritura, se realiza de manera asíncrona, es decir, es el elemento OPC del proyecto, OutOPC el que decide cuando se escribe. En ese momento se lanza una llamada al wrapper para escribir el TAG que tiene asignado el elemento OutOPC.

```
public void SetOPCDataNow(OPCGroup opcg, string tag, string data)
{
    // Obtiene datos de todos los items de una vez
    // y actualiza su valor en memoria para que los objetos puedan actualizarse
    //

    string GetQualityText = string.Empty;
    object value = null;
    object quality = null;
    object timestamp = null;
    short source = 2; // Es un tipo
```

```
    int IDS = 0;
    int ID = 0;
    int IDTobeWritten = 0;
    string tagID = string.Empty;
    int count = opcg.OPCItems.Count;
                        //
    for (int i = 1; i <= count; i++)
    {
        try
        {
            // Localizamos primero en el servidor el ID del TAG que se desea escribir
            // Para ello:

            // Se lee cada vez información de un elemento
            opcg.OPCItems.Item(i).Read(source, out value, out quality, out timestamp);
            // Cogemos el ID del Servidor
            IDS = opcg.OPCItems.Item(i).ServerHandle;
            // Cogemos el ID de gestión del ITEM
            ID = opcg.OPCItems.Item(i).ClientHandle;
            // Cogemos la ruta, TAG
            tagID = opcg.OPCItems.Item(i).ItemID;

            // Verificamos que es el TAG que buscamos (Que es el mismo que el elemento
            // OutOPC tiene asignado
            if (tag == tagID)
            {
                // Cogemos el server handled de dicho TAG
                IDTobeWritten = IDS;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error > " + ex.Message);
        }
    }

    // Obtenemos el manejador del ITEM para poder escribir en el
    OPCItem item = opcg.OPCItems.GetOPCItem(IDTobeWritten);

    // Según el tipo de dato del TAG, adaptamos el valor a dicho tipo de dato
    try
    {

// TYPE            ID      DESCRIPTION
//-------------------------------------------------------------------------------------------------------------------
//Boolean           11      1 Bit True or False
//Byte              17      1 Byte 0 - 255
//Byte Array*       8209    1 Byte times Array Size 0 - 255
//Short             2       2 bytes - 32768 – 32767
//Short Array*      8194    2 bytes times Array Size -32768 – 32767
//BCD               18 +    2 bytes 0 - 9999
//BCD Array*        8210    2 bytes times Array Size 0 - 9999
//Word              18      2 bytes 0 – 65535
//Word Array*       8210    2 bytes times Array Size 0 – 65535
//Long              3       4 bytes - 2,147,483,648 to 2,147,483,647
//Long Array*       8195    4 bytes times Array Size -2,147,483,648 to 2,147,483,647
//LBCD              19 +    4 bytes 0 - 99999999
//LBCD Array*       8211    4 bytes times Array Size 0 - 99999999
//Dword             19      4 bytes 0 – 4,294,967,295
//Dword Array*      8211    4 bytes times Array Size 0 – 4,294,967,295
//Float             4       4 bytes - (3.40 E38 – 1.40 E - 45) to (1.40 E - 45 – 3.40 E38)
//Float Array*      8196    4 bytes times Array Size -(3.40 E38 – 1.40 E - 45) to (1.40 E - 45 – 3.40 E38)
//Double            5       8 bytes - (1.798 E308 – 4.941 E - 324) to 4.941 E - 324 – 1.798 E308)
//Double Array*     8197    8 bytes times Array Size -(1.798 E308 – 4.941 E - 324) to 4.941 E - 324 – 1.798 E308)
//Char              16      1 Byte All Characters supported in the standard ASCII table
//String            8       1 – n Bytes All Characters supported in the standard ASCII table

        short OPCtype = item.CanonicalDataType;
        object dataToWrite = null;
        try
        {
            switch (OPCtype)
```

```
            {
                case 2:     // Short
                    dataToWrite = Convert.ToInt16(data);
                    break;
                case 3:     // Long
                    dataToWrite = Convert.ToInt32(data);
                    break;
                case 4:     // Float
                    dataToWrite = float.Parse(data);
                    break;
                case 5:     // Double
                    dataToWrite = Convert.ToDouble(data);
                    break;
                case 8:     // String
                    dataToWrite = data;
                    break;
                case 11:    // Boolean
                    if (data == "1")
                    { dataToWrite = 1; }
                    else
                    { dataToWrite = 0; }
                    //
                    break;
                case 16:    // Char
                    dataToWrite = Convert.ToChar(data);
                    break;
                case 17:    // Byte
                    dataToWrite = Convert.ToByte(data);
                    break;
                case 18:    // BCD/Word
                    dataToWrite = Convert.ToUInt16(data);
                    break;
                case 19:    //LBCD/DWORD
                    dataToWrite = Convert.ToInt32(data);
                    break;
                case 8194:  // Short Array
                    break;
                case 8195:  // Array Long
                    break;
                case 8196:  //Float Array
                    break;
                case 8197:  //Array Double
                    break;
                case 8209:  // Byte Array  - El tamaño dependerá del servidor
                    break;
                case 8210:  // BCD/Word Array
                    break;
                case 8211:  //Array LBCD/DWORD
                    break;
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Error OPC write: " + ex.Message);
        }


        // Finalmente escribimos en el TAG con el DATO adecuadamente, adaptado.
        item.Write(dataToWrite);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error > " + ex.Message);
    }
}
```

4 USO DE LA APLICACIÓN

En anteriores capítulos se ha empleado tiempo en describir aspectos técnicos y funcionales del proyecto, en parte interesantes para conocer el software, sus componentes, etc. En este capítulo se dedicarán algunos párrafos al conocimiento del software desde el punto de vista práctico, dando a conocer las pautas para crear y ejecutar un proyecto OPCSM completo con conexión a un servidor OPC.

Los apartados siguientes componen una guía de uso habitual con una explicación detallada de todas las funciones que incorpora esta primera versión funcional del entorno de desarrollo.

4.1 Instalación

4.1.1 Requisitos del sistema

Para la ejecución de OPCSM VAI es necesario disponer de un PC con plataforma Microsoft Windows 7 o superior, procesador mínimo Intel Core i3 o similar, con al menos 4 GB de RAM. Se debe tener instalado el paquete Microsoft .NET Framework v4.6

4.1.2 OPC Foundation Core Components.

Antes de arrancar el programa también es necesario tener instalado el pack "core components" en versión 32 bits desde OPC Foundation. Este pack proporciona las librerías necesarias para listar servidores (OPCENUM.exe) y el wrapper para comunicar con ellos. Este pack puede ser descargado desde su propio *site* accediendo a la siguiente dirección de internet,

https://opcfoundation.org/developer-tools/developer-kits-classic/core-components/#not-archived

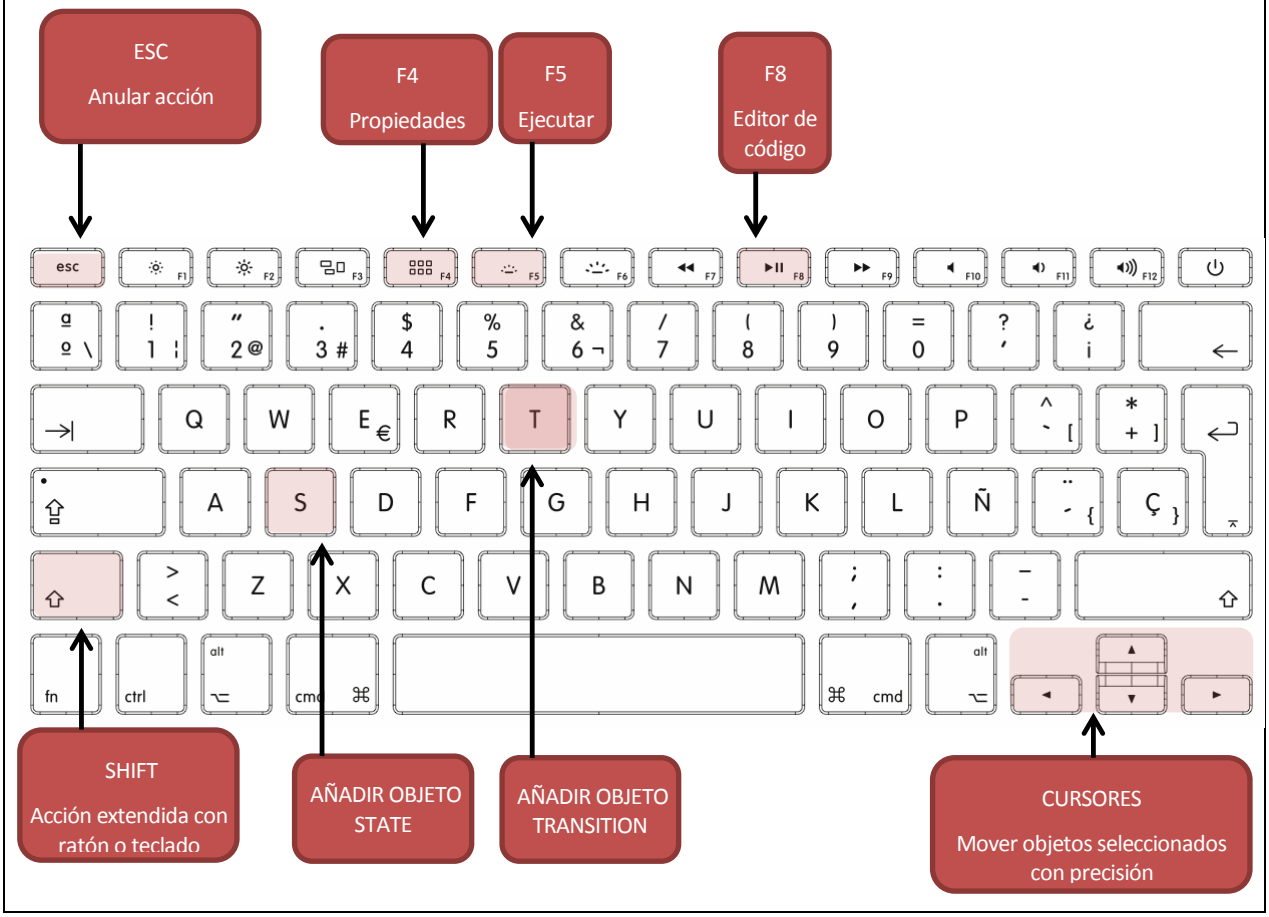
Para llevar a cabo la descarga es necesario registrarse en la web.

NOTA: La versión 3.0.106 del *OPC core components*, está incluida en el paquete de instalación de OPCSM VAI

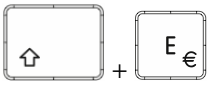
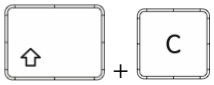
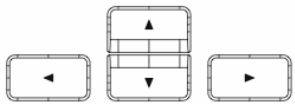
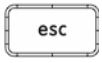
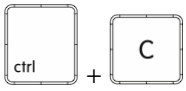
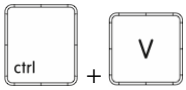
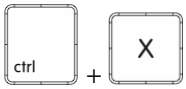
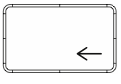
4.2 Consideraciones en el texto

A lo largo del texto se encontrarán abreviaturas relacionadas con acciones que el usuario puede llevar a cabo. En este apartado se dan a conocer para tenerlas en cuenta.

	Click-[izquierdo/derecho]	Click con el botón izquierdo ó derecho del ratón.
	Click-[izquierdo/derecho]-mantenido	Click con el botón izquierdo ó derecho del ratón pulsado y mantenido.
	Arrastre-de-ratón	Mover ratón con el botón que se haya indicado, manteniéndolo pulsado
	Tecla-xxx-mantenida	Tecla “xxx” del teclado, pulsada sin soltar hasta indicarlo.



4.3 Teclas de accesos rápido

	Muestra la ventana de propiedades del objeto seleccionado, grupo o documento activo.
	Ejecuta el diagrama de estados implementado en todos los documentos del proyecto, en modo de ejecución NORMAL.
	Ejecuta el diagrama de estados implementado en todos los documentos del proyecto, en modo de ejecución PETRI.
	Muestra la ventana de edición de código del objeto con capacidad de scripting.
	Añade un objeto STATE al documento activo
	Añade un objeto TRANSITION al documento activo
	Añade un objeto EXPRESSION al documento activo
	Añade un objeto CONDITION al documento activo
	Mueve los objetos seleccionados con precisión, en el documento activo.
	Anula la acción activada. Deselecciona los objetos seleccionados.
	Copiar los elementos seleccionados al portapapeles.
	Pegar los elementos que están en el portapapeles, en el documento activo.
	Copiar los elementos seleccionados al portapapeles, y eliminarlos del documento activo.
	Eliminar los objetos seleccionados. (Tecla DELETE o SUPR)
	Tecla “+”, aumentar zoom
	Tecla “-”, disminuir zoom

4.4 Generalidades del entorno.

El software se compone de un entorno común, formado por un área de edición, una barra de menú, una barra de herramientas y una barra de estado en la parte inferior.

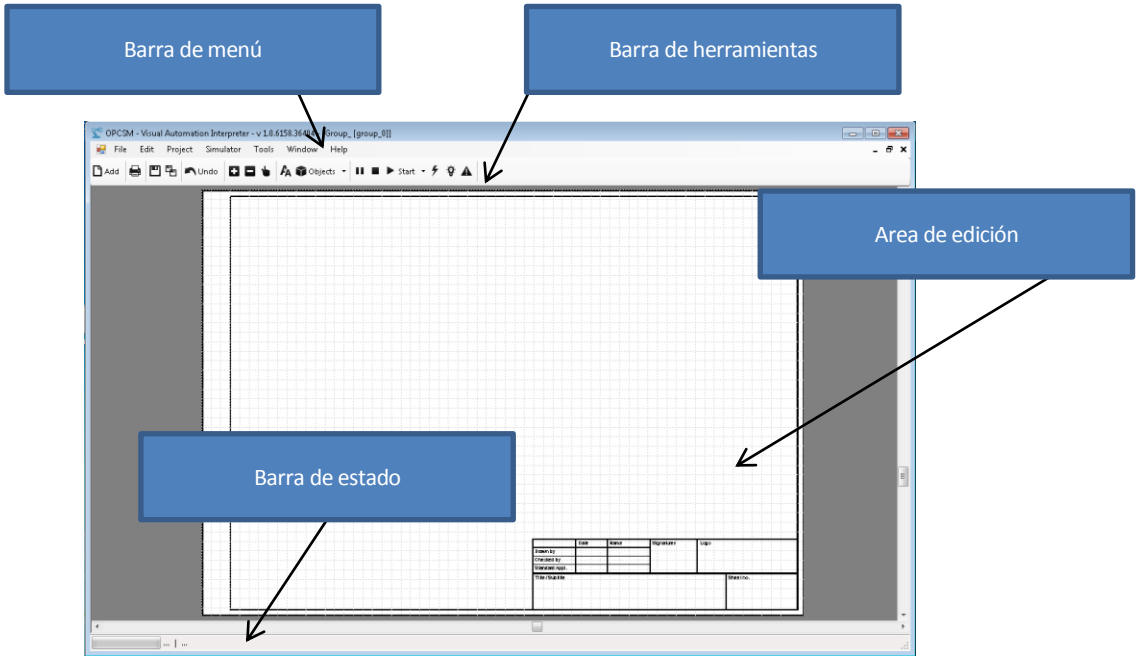


Fig. 4.3-1 – Programa principal – OPCSM VAI

- **Barra de menú** – Compone el conjunto de menús con las opciones de edición, herramientas y controles de ejecución necesarios para el desarrollo del proyecto.
- **Barra de herramientas –** Contiene una serie de enlaces rápidos a las herramientas y comandos más habituales, para agilizar el trabajo.
- **Barra de estado –** Indica el estado de edición, ejecución, errores, etc.
- **Área de edición –** Es el área de diseño, donde se localizan los documentos que se agregan al proyecto.

4.5 Crear un Sistema de estados OPC.

4.5.1 Crear un proyecto nuevo

Antes de iniciar un nuevo desarrollo es necesario generar un proyecto OPCSM. (fig. 4.3-1 y 4.3-2)

Para ello se accede al menú File → New proyect, esto mostrará una ventana de diálogo para configurar los parámetros esenciales para crear el proyecto.

- **Project title** – (Opcional) Es el título que describe el proyecto, es un campo de descripción.
- **Path of the Project** – Ruta donde se almacenará el directorio del proyecto y todos los documentos. Por defecto se genera configura en la ruta "Mis documentos" de Windows.
- **Directory name** – Directorio que contiene el fichero del proyecto y los ficheros de los documentos.
- **Filename Project** – Nombre del fichero de proyecto. Por defecto se genera uno automáticamente, pero puede ser modificado.

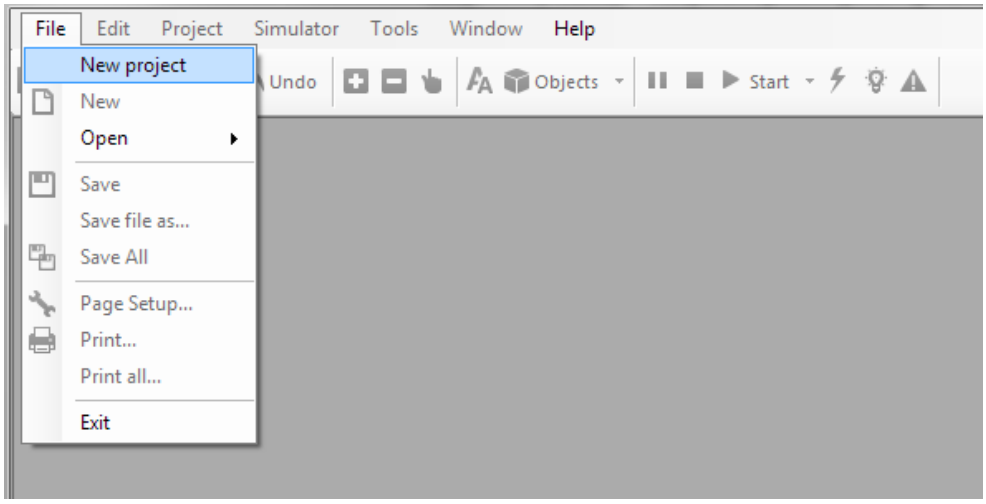


Fig. 4.4-1 – Detalle del menú FILE

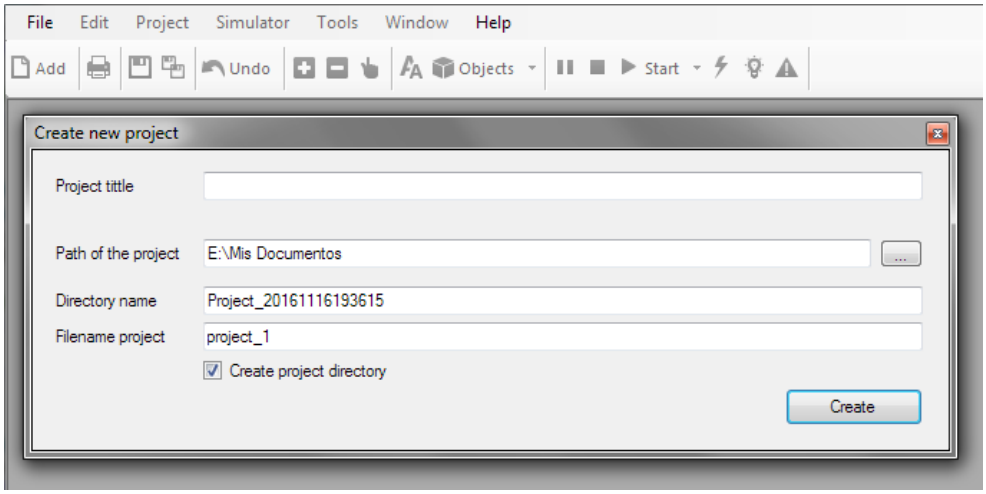


Fig. 4.4-1 – Detalle de la ventana de generación de nuevo proyecto.

Una vez finalizada la configuración de los parámetros del proyecto OPCSM, se debe pulsar CREATE para generar en el entorno el proyecto, así como el directorio y los ficheros en disco (En la ruta especificada). Con esto el entorno queda listo para comenzar a desarrollar un nuevo sistema con diagrama de estados.

Se puede acceder al resto de propiedades del proyecto, mediante el menú Project → Properties que mostrará todas las propiedades del proyecto y las cuales pueden ser modificadas a excepción del nombre del fichero de proyecto y la ruta de almacenamiento

NOTA: Para cambiar el nombre del fichero y la ruta del proyecto, debe hacerse fuera del entorno de desarrollo a través del sistema operativo renombrando el fichero y cambiando la ruta manualmente. Esto no debe aplicarse a los documentos, ya que estos SI PUEDEN ser cambiados de nombre dentro del entorno)

Cuando se abra el fichero de proyecto, automáticamente el programa detecta la nueva ruta y extrae de ella el resto de documentos.

En la figura 4.3-3 se puede ver la ventana de propiedades del proyecto con mayor detalle.

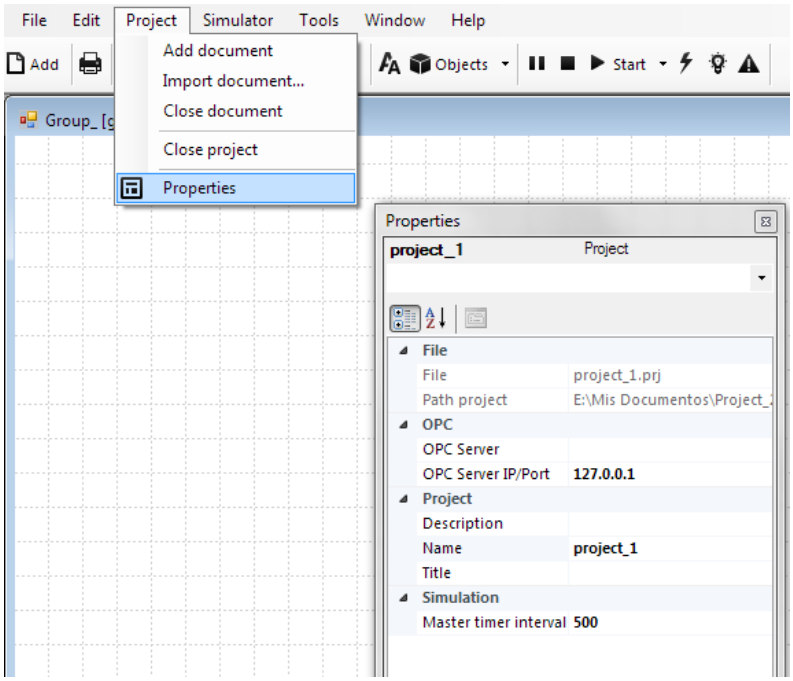


Fig. 4.4-3 – Detalle de la ventana de propiedades del proyecto

Se pueden observar las propiedades del proyecto en las que destacaremos,

- **OPC Server** – Información del servidor OPC al que se conecta el sistema
- **OPC Server IP/Port** – IP del servidor en red, donde están los servidores OPC instalados.
- **Description** – Campo de información resumen del proyecto (Opcional)
- **Name** – Nombre del proyecto
- **Title** – Título del proyecto (Opcional)
- **Master timer interval [ms]** – Configura la base de tiempo del reloj de simulación.

4.5.2 Crear un sistema

4.5.2.1 Configuración del documento

El siguiente paso en el desarrollo del proyecto, es crear el sistema. Este se organizará dentro de documentos que no son más que hojas donde se agregan elementos interconectados. El proyecto puede estar formado por todos los documentos que sean necesarios, con el objetivo de organizar el sistema lo mejor posible.

Cuando se ha generado por primera vez el proyecto, se crea automáticamente una hoja nueva (documento), lista para comenzar a insertar elementos e interconectarlos. Esta hoja se puede configurar para caracterizarla, agregando información en su cajetín, como es, título del proyecto, firma, fecha de creación, autor, logotipo, etc.

Para ello se hará click con el botón derecho del ratón encima de una zona libre del documento, y se debe seleccionar la opción *Properties* del menú desplegable, esta acción mostrará las propiedades del proyecto (figura 4.3-4)

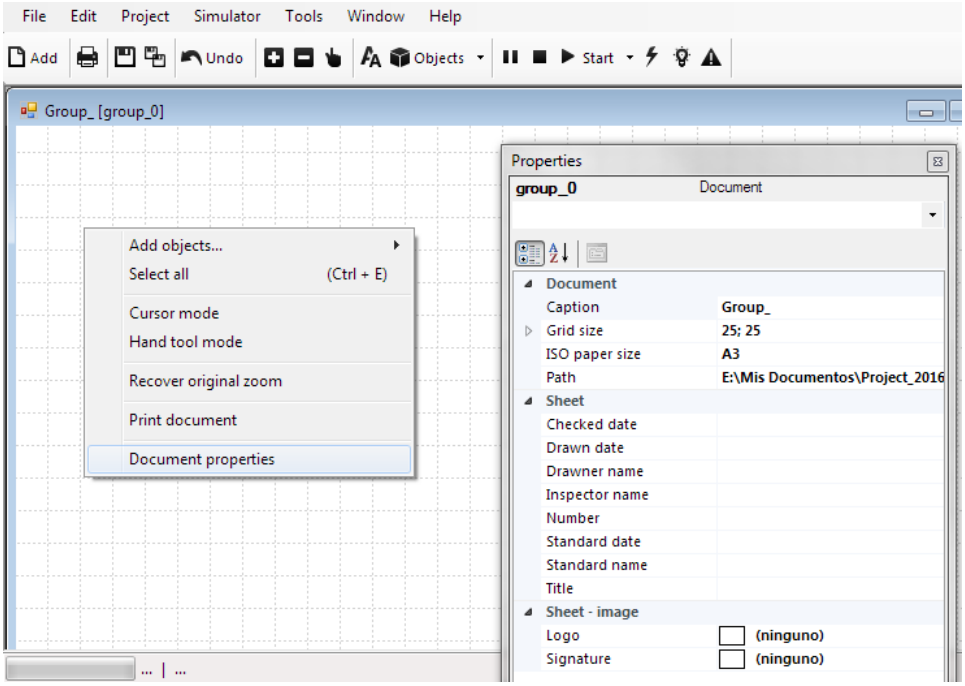


Fig. 4.4-4 – Detalle de la ventana de propiedades del documento

En la ventana propiedades se pueden observar los siguientes campos que pueden configurarse para caracterizar el documento.

- **Caption** – Configura el sufijo que identifica la ventana en la ventana de exploración del proyecto.
- **Grid size** – Configura el tamaño de la *grid* del documento.
- **ISO paper size** – Configura el tamaño del papel del documento ajustándolo a la norma ISO seleccionada.
- **Path** – Indica la ruta donde el fichero .xml del documento, está contenido, es un campo de solo lectura.

En las secciones *Sheet* y *Sheet – image* se pueden ver campos relativos a los datos que se muestran en el cajetín.

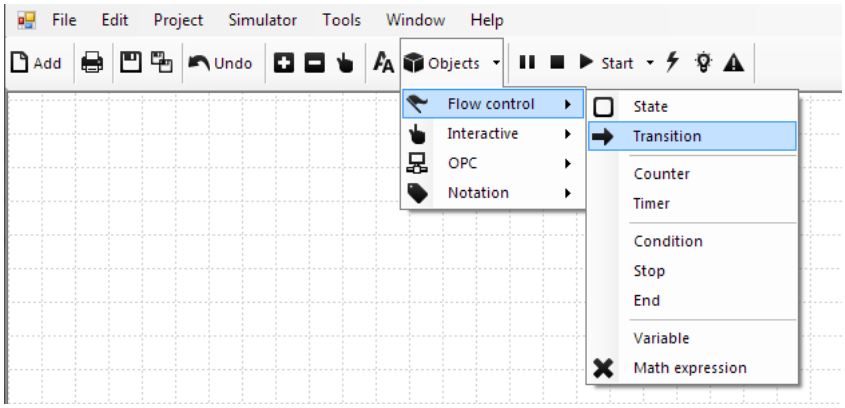
La configuración del documento puede hacerse indistintamente en el momento que se desee, no es necesario para poder comenzar a desarrollar el sistema.

4.5.2.2 Inserción y edición de elementos para el sistema

En este paso se comenzará a insertar elementos que compondrán el sistema, en el documento. Para ello se dispone de varias posibilidades, mediante teclas de acceso rápido, barra de menú o menú contextual. Además el área de documento posee un interfaz muy versátil para realizar la edición del sistema durante su implementación, pudiendo realizar acciones con el ratón y teclado conjuntamente, que será detallado en sucesivos apartados.

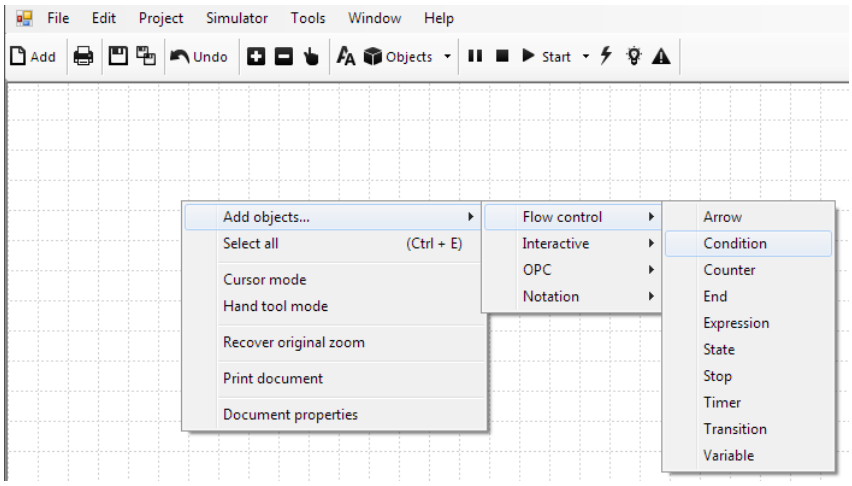
Para la inserción de elementos puede hacerse mediante,

- **Barra de herramientas**



Mediante el botón *Objects* se puede acceder a varios grupos de elementos que han sido descritos en el capítulo 2 desde el punto de vista más técnico, pero que da una idea de su posible uso. Desde este botón es posible agregar elementos al documento activo en ese momento.

- **Menú contextual**



Otro modo es mediante el menú contextual, haciendo click con el botón derecho sobre una zona libre del documento y seleccionando la opción, *Add objects…* que permitirá desplegar igualmente un grupo con los tipos de elementos y a su vez acceder a los elementos de cada tipo, de manera similar al caso anterior.

- **Teclado**

Existen teclas de acceso rápido a los elementos más importantes, se pueden observar en la tabla siguiente.

Tecla	Descripción
S ó Shift + S	Agrega un elemento STATE
T ó Shift + T	Agrega un elemento TRANSITION
Shift + E	Agrega un elemento EXPRESSION
Shift + C	Agrega un elemento CONDITION

Una vez el elemento está insertado, aparecerá en el documento y es posible realizar acciones de edición sobre él; Seleccionarlo, moverlo, cambiar su tamaño, modificar sus propiedades. Es posible copiarlo, cortarlo y pegarlo, eliminarlo, etc.

- **Selección simple**

  Para seleccionar un elemento, simplemente se hace click sobre él. En programa lo detecta y lo sombrea. En el elemento aparecerá además el testigo rojo para cambio de tamaño.

  NOTA: Para deseleccionar un elemento, basta con hacer click en una zona libre del documento o pulsar la tecla ESC.

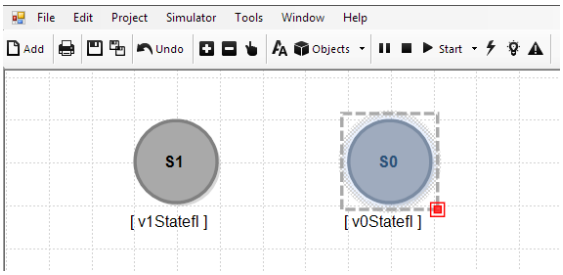


Fig. 4.4-5 – Detalle de elemento no seleccionado (izquierda) y seleccionado (derecha)

En la figura 4.3-5 se puede observar la selección simple de un elemento (lado derecho), el elemento queda sombreado y rodeado de un área punteada. Cuando un elemento está seleccionado se pueden hacer varias operaciones sobre el:

- **Cambio de tamaño**

  Hacer *click-izquierdo-mantenido* sobre el testigo rojo y *Arrastre-de-ratón* hacia abajo para agrander o hacia arriba para achicar.

- **Mover**

  Hacer *click-izquierdo-mantenido* encima del objeto y *arrastre-de-ratón*, para mover el objeto.

- **Movimiento de precisión.**

  Mediante las teclas cursores cuando el objeto está seleccionado.

- **Copiar, Cortar, Marcar, Eliminar y ver Propiedades.**

  Estas acciones pueden realizarse desde el menú contextual (C*lick-derecho* sobre el objeto seleccionado) mediante la barra de menús, con el menú *Edit* o mediante el teclado. fig. 4.3-6

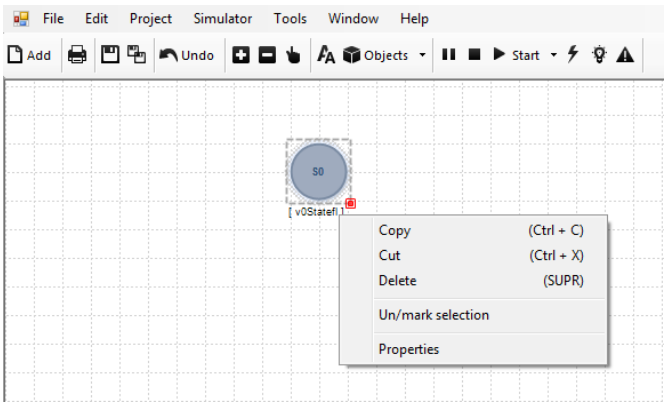


Fig. 4.4-6 – Detalle de opciones de edición.

- **Selección múltiple**

  La selecciona múltiple, puede ser realizada de varias maneras, mediante teclado y ratón, o solo con el ratón.

  - **Selección múltiple con rectángulo de selección.**

    Para ello hacer *click-izquierdo-mantenido*, en una zona libre del documento y *arrastre-de-ratón*, hasta capturar los elementos con el rectángulo azulado que se forma. Para realizar la selección soltar el botón y los elementos contenidos en el rectángulo, serán seleccionados.

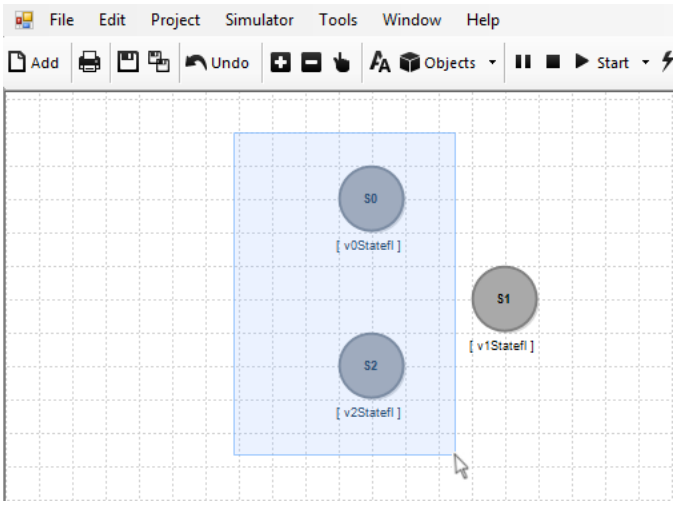


Fig. 4.4-7 – Detalle de selección múltiple

  - **Selección múltiple con teclado y click simple de ratón.**

    Para ello se va seleccionando el objeto haciendo *click-izquierdo* sobre él, con *tecla-shift-mantenida*, de este modo se van agregando objetos al grupo de selección múltiple.

  - **Selección múltiple con teclado y rectángulo de selección.**

    Para ello se van seleccionando objetos con *tecla-shift-mantenida* y realizando rectángulos de selección, de este modo se van agregando objetos al grupo de selección múltiple.

Cuando están los elementos seleccionados, es posible realizar las siguientes acciones.

- **Mover**

  Hacer *click-izquierdo-mantenido* encima del objeto y *arrastre-de-ratón*, para mover el objeto.

- **Movimiento de precisión.**

  Mediante las teclas cursores cuando el objeto está seleccionado.

- **Copiar, Cortar, Marcar, Eliminar y ver Propiedades.**

  Estas acciones pueden realizarse desde el menú contextual (Haciendo *click-derecho* sobre el objeto seleccionado), mediante la barra de menús, con el menú *Edit* o mediante el teclado. Las opciones son *Copy*, *Cut*, *Mark*, *Delete* y *Properties.* Ver fig. 4.3-6

NOTA: No es posible realizar cambio de tamaño de los objetos seleccionados en una selección múltiple.

NOTA(2): La selección múltiple mediante rectángulo de selección también captura elementos enlazados, pudiendo seleccionar un diagrama completamente terminado.

NOTA(3): Cuando se copia/pega un elemento, se duplica, esto quiere decir que adopta todas sus propiedades incluso su variable. Conviene modificar la variable asignada al elemento si no se desea tener un comportamiento duplicado a la hora de ejecutar la red de estados.

- **Enlace simple**

Para realizar un enlace simple entre dos objetos, hay que entrar en el *modo de enlace*, haciendo doble-click con botón izquierdo del ratón sobre el elemento de partida (Sin necesidad de mantener pulsado el botón) y movemos hasta el elemento de llegada, haciendo click-simple sobre este último. Si el enlace es correcto, ambos elementos se unirán con una flecha, en el caso de no ser correcto la flecha de enlace se vuelve de color rojo y el extremo se convierte en un círculo indicando que se debe seleccionar otro objeto o cancelar la operación (Pulsando ESC o haciendo click en una zona libre del documento).

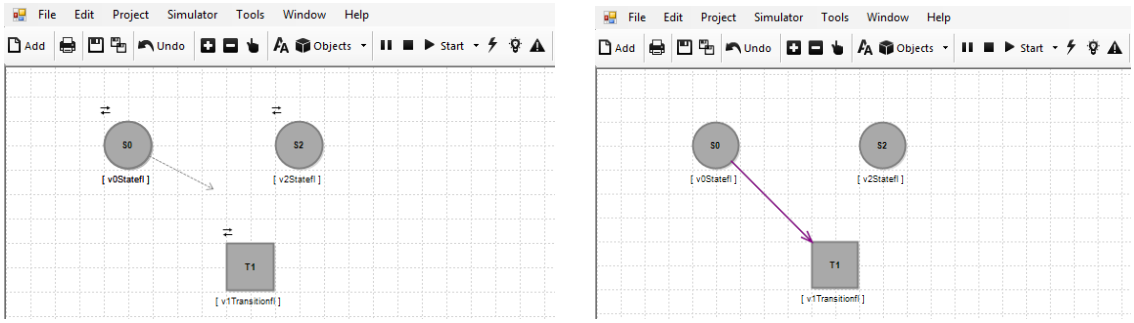


Fig. 4.4-8 – Detalle de enlace en curso (izquierda) y realizado (derecha).

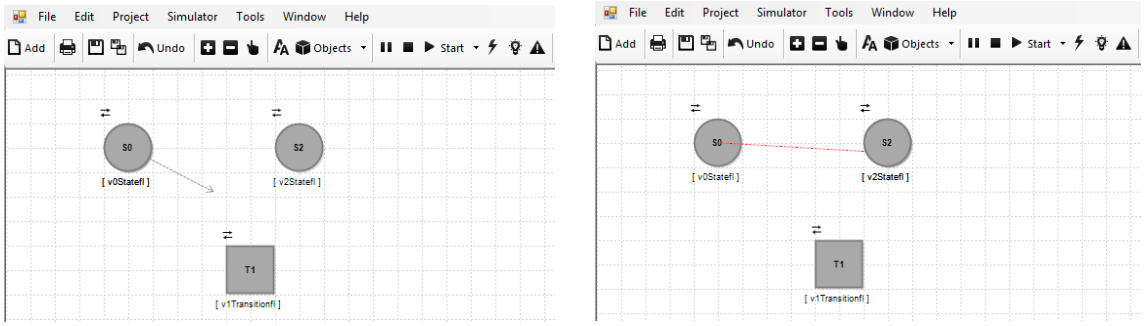


Fig. 4.4-8 – Detalle de enlace en curso (izquierda) y fallido (derecha).

NOTA: No es posible conectar/enlazar dos objetos del mismo tipo base, es decir objetos de tipo base estado u objetos de tipo base transición, siempre deben estar intercalados.

NOTA(2): Los objetos de tipo interactivo, no son enlazables.

Otro modo de realizar un enlace simple mediante ratón y teclado es de la siguiente forma,

- **Enlace uno-a-uno**

1. Salir del modo de enlace (Pulsar ESC)
2. Seleccionar un elemento de partida.
3. Pulsar la tecla CTRL y mantener.
4. Hacer click sobre el elemento de llegada y soltar CTRL.

- **Enlace múltiple**

  El enlace múltiple se lleva a cabo combinando acciones de teclado y ratón, sin necesidad de entrar en el modo de enlace simple (Doble-click sobre el elemento de partida)

  - **Enlace todos-a-uno**

    5. Seleccionar un grupo de elementos de partida.
    6. Pulsar la tecla CTRL y mantener.
    7. Hacer click sobre el elemento de llegada y soltar CTRL.

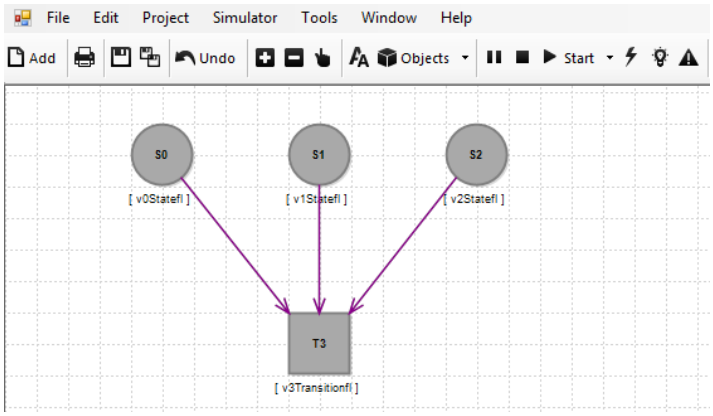


Fig. 4.4-9 – Detalle de enlace todos a uno.

  - **Enlace uno-a-todos**

    8. Seleccionar el grupo de elementos de llegada.
    9. Pulsar la tecla CTRL+SHIFT y mantener.
    10. Hacer click sobre el elemento de partida y soltar CTRL+SHIFT.

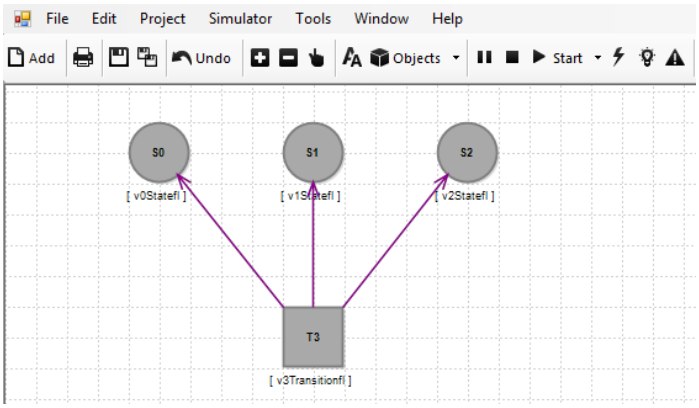


Fig. 4.4-10 – Detalle de enlace uno a todos.

4.5.3 Definir expresiones y condiciones

Otra de las características del entorno de desarrollo es la posibilidad de generar expresiones matemáticas de cierta versatilidad, para realizar cálculos y comparaciones. Estas opciones pueden llevarse a cabo con los elementos EXPRESSION y CONDITION.

Para hacer más sencilla la labor de escritura de las expresiones, está disponible un editor de código al que se puede acceder de la siguiente manera:

1. Seleccionar el elemento con capacidad de scripting.
2. Pulsar la tecla de función F8 ó mediante el menú contextual en la opción "Code Editor"

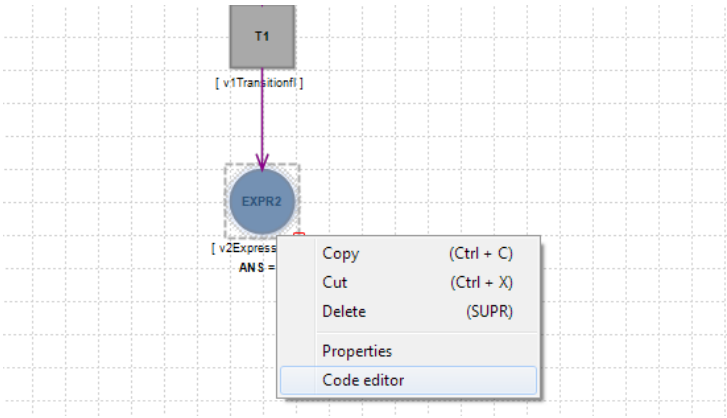


Fig. 4.4-9 – Detalle del menú y opción "Code editor"

y se mostrará la ventana ilustrada a continuación,

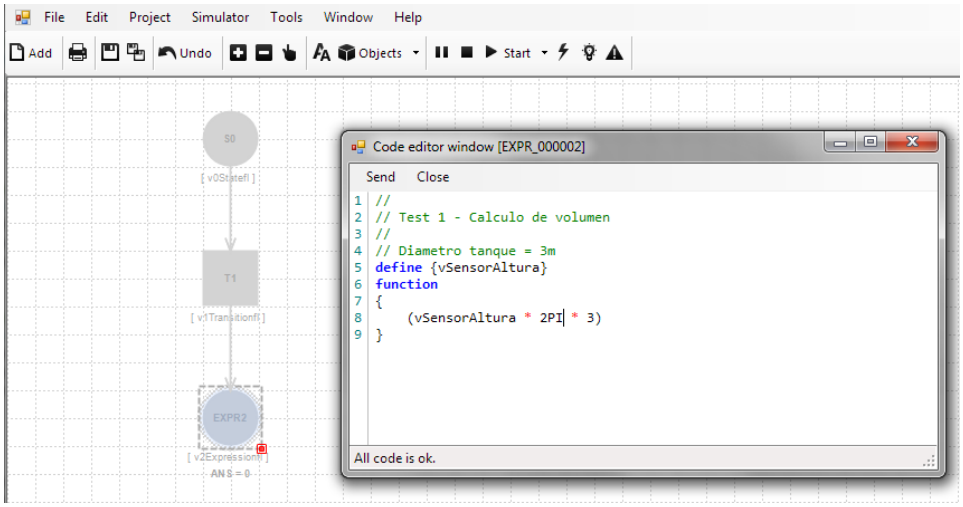


Fig. 4.4-10 – Detalle de la ventana editor de código.

En ella se inserta el código que se ejecutará cuando se dispare en nodo tipo "Expression" o "Condition"

4.5.4 Programas para elementos tipo Script.

Se ha visto antes que los elementos *Expression* y *Condition* poseen la característica de contener código, y este se ejecuta para resolver fórmulas matemáticas. A continuación se verá cómo implementarlas.

Un programa puede contener tres secciones, definición, constantes y función.

Definición – Define las variables que serán usada en la expresión implementada en el campo "function". Esta sección queda liderada por el comando, "define" y es obligatorio establecerlo para poder resolver la expresión contenida en function, si esta posee variables, para su resolución.

Constantes – Define las constantes que pueden usarse en la expresión. Esta sección queda definida por el comando "constant" y es necesaria si se va a hacer uso de constantes.

Función – Define la expresión matemática o condición a ser ejecutada. Está definida por el comando "function"

Comentarios – Define la sección de comentarios. Un comentario comienza por "//"

A continuación se ve un ejemplo de estas secciones.

```
//
// Sección de comentarios, liderada por doble barra
//
define
{
    v1,v2,v3
}

// Define la sección de constantes
const
{
    total=12.78
}

// Define la función matemática a ejecutar
function
{
    v2 = cos(v1)*total;
    v3 = v2 + v1;
}
```

Es posible ver más información sobre los comandos en el capítulo 2.

4.5.4.1 Consideraciones de la sección function.

La asignación de resultados a variables puede ser dos tipos, dependiendo si la expresión tiene que ser asignada a otra variable distinta de la variable interna del elemento, (Se hace uso del símbolo "="), o si el resultado va directamente a la variable interna del elemento *expression* o *condition,* denominada asignación directa.

Asignación directa – Para este tipo de expresiones, no es necesario usar el símbolo "=" para realizar la asignación del resultado a la variable interna, basta con poner la expresión directamente, el resultado pasará a la variable interna. Véase el ejemplo siguiente, el resultado de la expresión (v1+2)*v3 se insertará en la variable interna del elemento expression.

```
function
{
    (v1+2)*v3;
}
```

Otro ejemplo para el elemento *condition*

```
function
{
    ((v1 > 2) & (v2 > 3));
}
```

Una asignación a otra variable puede verse en el ejemplo a continuación, el resultado por ejemplo de v2+v1 se pasará a v3.

```
function
{
    v2 = cos(v1)*total;
    v3 = v2 + v1;
}
```

4.5.5 Configurar el acceso al servidor OPC

Un proyecto OPCSM puede ser conectado a un servidor OPC y enviar y recibir datos de este, mediante los elementos InOPC y OutOPC. Para conseguir esta funcionalidad, hay que definirle al proyecto una conexión con el servidor OPC. Esta configuración se hace mediante el "OPC manager" (Se puede ver en la figura a continuación)

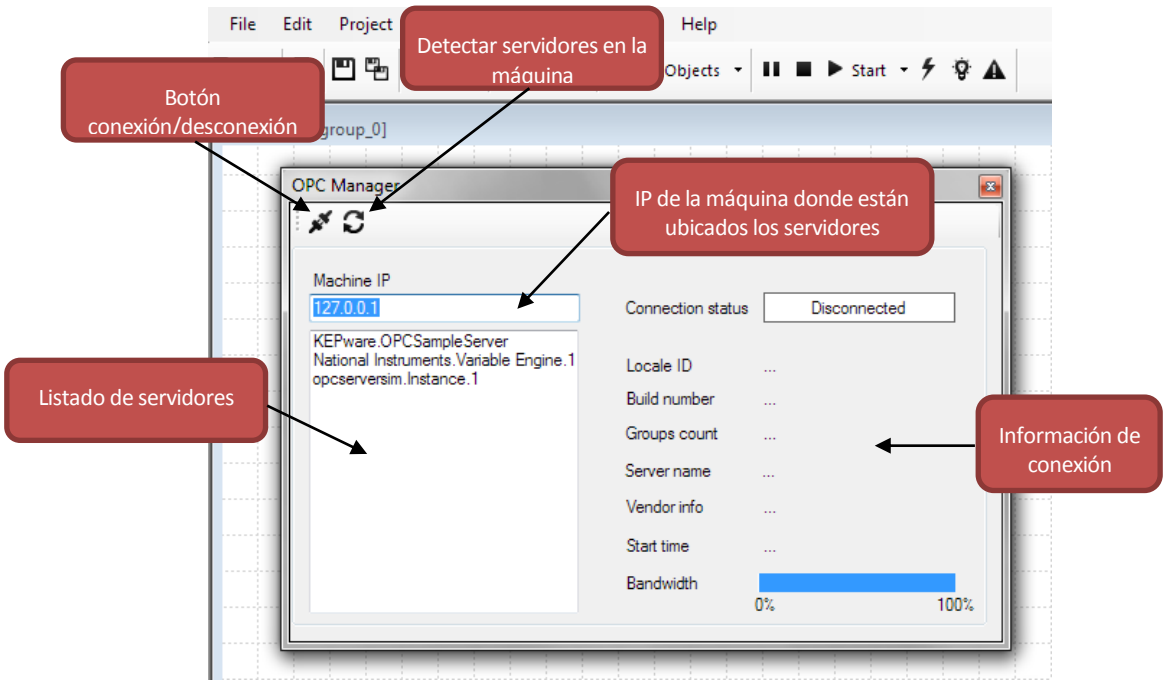


Fig. 4.4-11 – Detalle de la ventana de configuración del servidor OPC.

La ventana se compone da varias partes. El listado de servidores, información de conexión con el servidor, barra de acceso rápido, configuración de la IP de la máquina que contiene los servidores.

Para realizar una conexión, basta con seleccionar un servidor de los contenidos en la lista, haciendo doble-click y se realizará la conexión. También es posible seleccionarlo y pulsar el botón de "conexión/desconexión". Una vez establecida la conexión con el servidor, se obtendrá información relativa a este (Se puede observar en la figura 4.4-12)

Cuando esto ocurre, OPCSM le asigna a los elementos tipo OPC ya insertados, el wrapper para poder interactuar con el servidor. Lo mismo ocurre cuando se insertan elementos nuevos, cuando una conexión ya ha sido establecida. Para el funcionamiento de los elementos InOPC y OutOPC se puede obtener más información, en el capítulo 2.

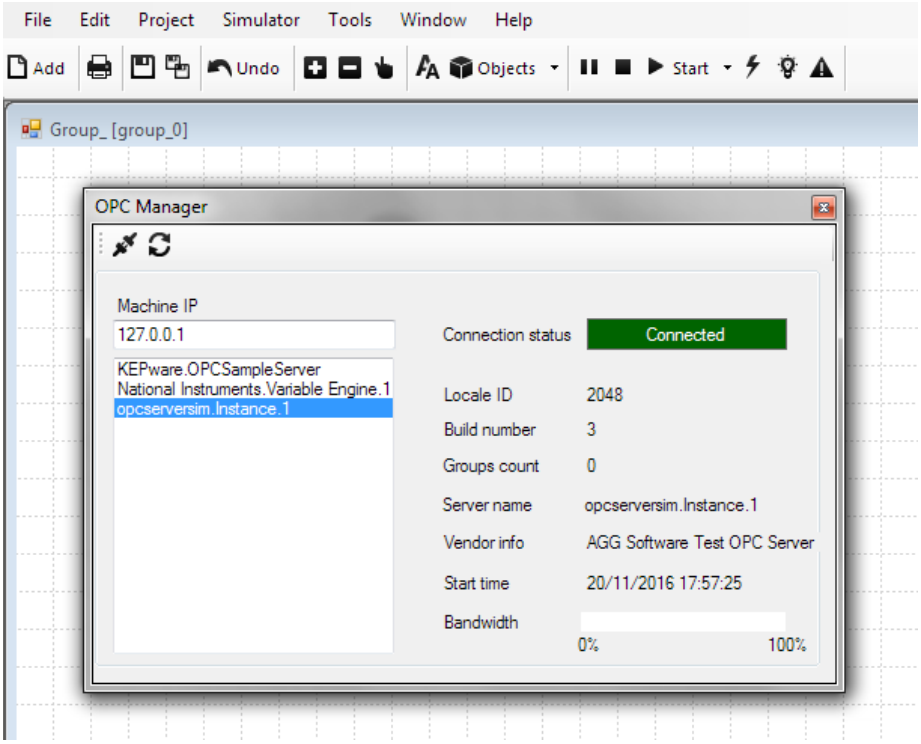


Fig. 4.4-12 – Detalle de la conexión con el servidor OPC.

4.5.6 Ejecutar el sistema.

Esta sería la fase final del desarrollo, para comprobar que el sistema funciona acorde a las premisas que se le hayan establecido. La acción de ejecución se realiza mediante la barra de botones superior o mediante las teclas de función F5, F6 y SHIFT+F5. Hay varios modos de ejecución, modo normal y modo Petri.

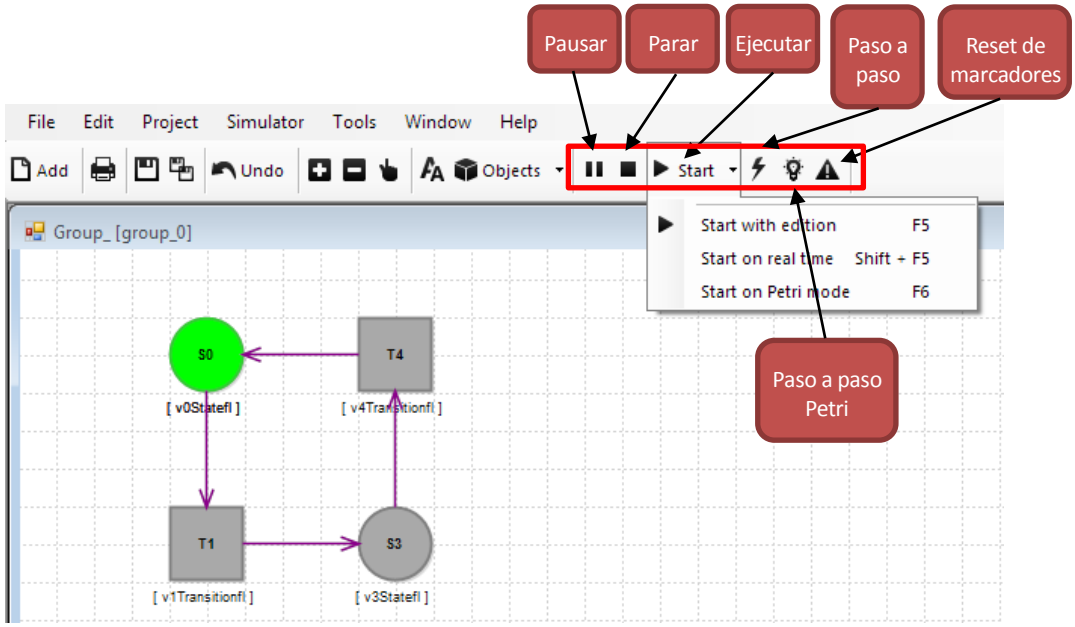


Fig. 4.4-13 – Detalle del menú ejecución.

Para ejecutar el sistema en modo normal, basta con pulsar la tecla de función F5 o la opción de menú "Start with edition", para el caso de ejecutar el sistema de estados con la filosofía de las redes de Petri, se pulsará la tecla de función F6 o selección de la opción "Start on Petri mode".

También es posible para el sistema, o pausarlo. Existen también el modo de ejecución paso a paso, y la ejecución con o sin depuración.

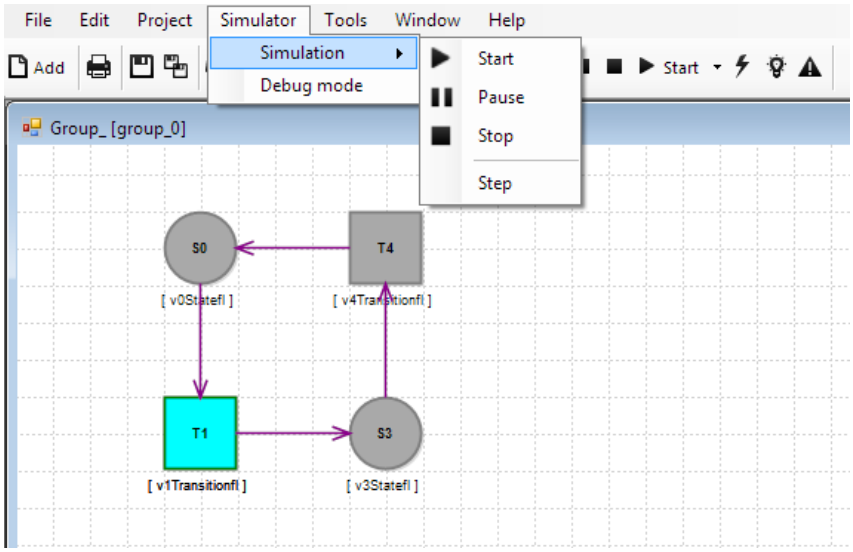


Fig. 4.4-14 – Detalle del menú ejecución.

Para activar el modo depuración basta acceder al menú "Simulator" y pulsar la opción "Debug mode". Cuando el modo debug está activado, es posible trazar la ejecución de un sistema mediante la ventana "Debug console" del menú "Window".

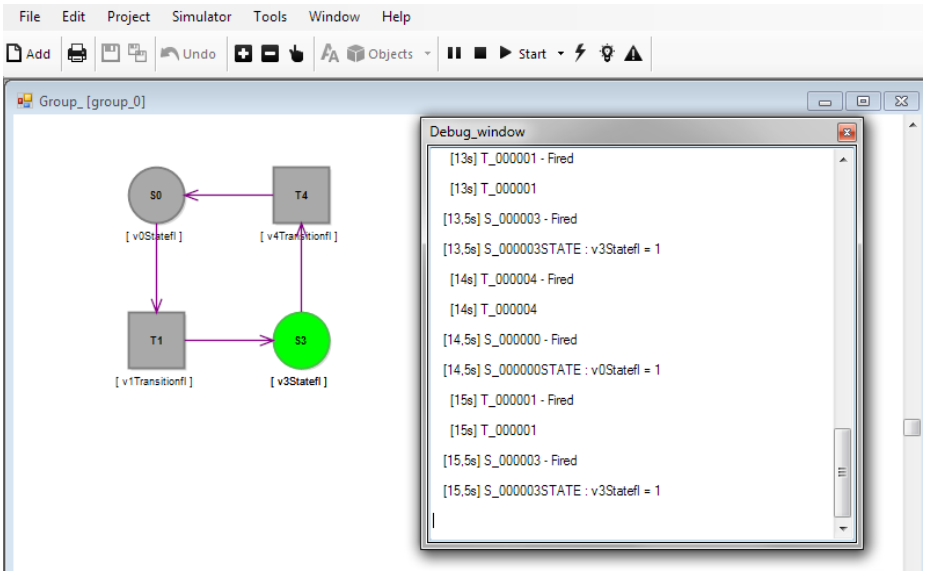


Fig. 4.4-15 – Detalle del modo debug y la ventana de depuración.

4.5.6.1 **Modos de ejecución.**

En este apartado se explicará cada uno de los funcionamientos.

Modo Normal – En este caso el motor de ejecución disparará los elementos que contienen el marcador. Si se observa la fig. 4.4-13 el elemento S0 esta sombreado en color verde, esto significa que el motor de ejecución lanzará dicho elemento en un flanco de reloj. Cuando un elemento es lanzado o disparado, se realiza una

ejecución de la función que tiene programada realizar, por ejemplo el elemento de la fig. 4.4-13 al ser disparado pondrá en su variable interna un valor constante definido o en otro caso el valor "1", cuando está marcado; En caso de no estar marcado, será "0".

Cuando un elemento se ha ejecutado después de ser disparado, pasa el testigo a los elementos siguientes según el sentido de la flecha, a los que está conectados. En el ejemplo de la fig. 4.4-13, una vez ejecutado el testigo pasará a la transición T1, y será disparado. Así sucesivamente durante la ejecución del sistema.

Modo Petri – En este modo se sigue la filosofía de ejecución de redes de Petri, disparando las transiciones que están habilitadas para ser ejecutadas. Una transición está habilitada cuando los elementos tipo estado conectados a ella, poseen el testigo. El modo de ejecución Petri se ha implementado para compatibilizar el entorno con sistemas basados en redes de Petri.

5 CONCLUSIONES

El proyecto descrito en este documento presenta una herramienta de desarrollo basada en un interfaz gráfico útil para implementar procesos automáticos, agilizando su desarrollo gracias a las herramientas de edición que posee y mejorando la compresión del sistema, ya que la presentación en nodos y el seguimiento de la ejecución por paso de testigo, permiten ver como el sistema está trabajando en todo momento. A lo largo de la implementación del proyecto se han encontrado numerosas dificultades, dado que conseguir que una aplicación tenga un interfaz amigable para el usuario y que permita simplificar las tareas de implementación y edición, supone tener detrás una compleja programación. En ocasiones los distintos componentes del proyecto han tenido que ser re-especificados y re-escritos cuando la funcionalidad comenzó a aumentar y se deseaba proporcionar un interfaz más ágil cada vez. Uno de las partes impactadas por este hecho, fue el motor de decisiones que lidera la funcionalidad del interfaz gráfico, para conocer en cada momento cual es la intención del usuario, de ahí la sencillez de edición conseguida.

Pese al gran esfuerzo que ha supuesto llevar a cabo esta herramienta, bien planificada en un principio llega a ser gratificante y da belleza al resultado final, pues permite conseguir funcionalidad muy precisa, y ser mejorada sin una excesiva complejidad. En definitiva puede decirse que el desarrollo de este proyecto ha permitido adquirir una experiencia más profunda no solo en el sentido de la programación, si no en el concepto de ergonomía y aspectos relacionados con la experiencia del usuario que a veces se olvida en herramientas de ingeniería con un carácter tan técnico y especializado.

Para finalizar se plasman a continuación posibles líneas futuras, que podrían seguirse partiendo de este software que aumentarían y mejorarían tanto su funcionalidad como utilidad.

- Mejorar el entorno gráfico basándolo en DirectX, para acelerarlo mediante tarjeta gráfica, ya que se ha hecho uso del GDI+ que carece de este aspecto.
- Mejorar el entorno de ejecución e implementación, añadiendo nuevos tipos de datos, como array, tratamiento de cadenas, etc.
- Añadir más elementos interactivos, slides, depositos, válvulas, etc para hacer más real el sistema implementado.
- Hacer uso de OPC UA, más avanzado que OPC Clásico.
- Permitir conectar con varios servidores OPC a la vez, asignando a los elementos InOPC y OutOPC, a donde realizar sus operaciones.
- Crear un motor de ejecución en plataforma que permita implementar un sistema de tiempo real. Esto podría ser implementado en sistemas operativos Linux/Unix.
- Crear un compilador para poder ejecutar el programa desarrollado de manera autónoma en otra plataforma, esto eliminaría la característica de seguimiento de testigo, pero permitiría ejecutar los sistemas desarrollados como un autómata. Esto por ejemplo da la idea de generar un compilador para Arduino, dado que es un hardware de bajo coste y poder ejecutar el diagrama de estados generado directamente, consiguiendo automatismos de bajo coste, para soluciones en los que se pueda aplicar.
- Crear un OPCSM BOX (Hardware con motor OPCSM) para ejecutar directamente el sistema implementado en él y creando una centralita de E/S, OPC con múltiples conexiones WiFi, discretas, etc. para control domótico, industrial de bajo coste.
- Añadir elementos para control remoto y mediante una aplicación en Android poder interactuar con el sistema implementado.

6 REFERENCIAS

1. OPC Foundation – *The industrial interoperability standard* – https://opcfoundation.org/

2. Mathos parser .NET library – *Mathematical Expression Parser* – Ethan Dagner – United State – https://mathosparser.codeplex.com/

3. FastColoredTextBox .NET library – *Fast Colored TextBox for Syntax Highlighting. The text editor component for .NET* – Pavel Torgashop – https://github.com/PavelTorgashov/FastColoredTextBox

4. OPC Data Access Wrapper class – Sourceforge – Developer site – https://sourceforge.net/projects/opcdawrapper/

5. Stack overflow – *Developers Site* – http://stackoverflow.com/

6. MSDN Library – *Microsoft Developer Network* – https://msdn.microsoft.com/

7. OPCConnect.com – *OPC Connect site* – http://www.opcconnect.com/dotnet.php

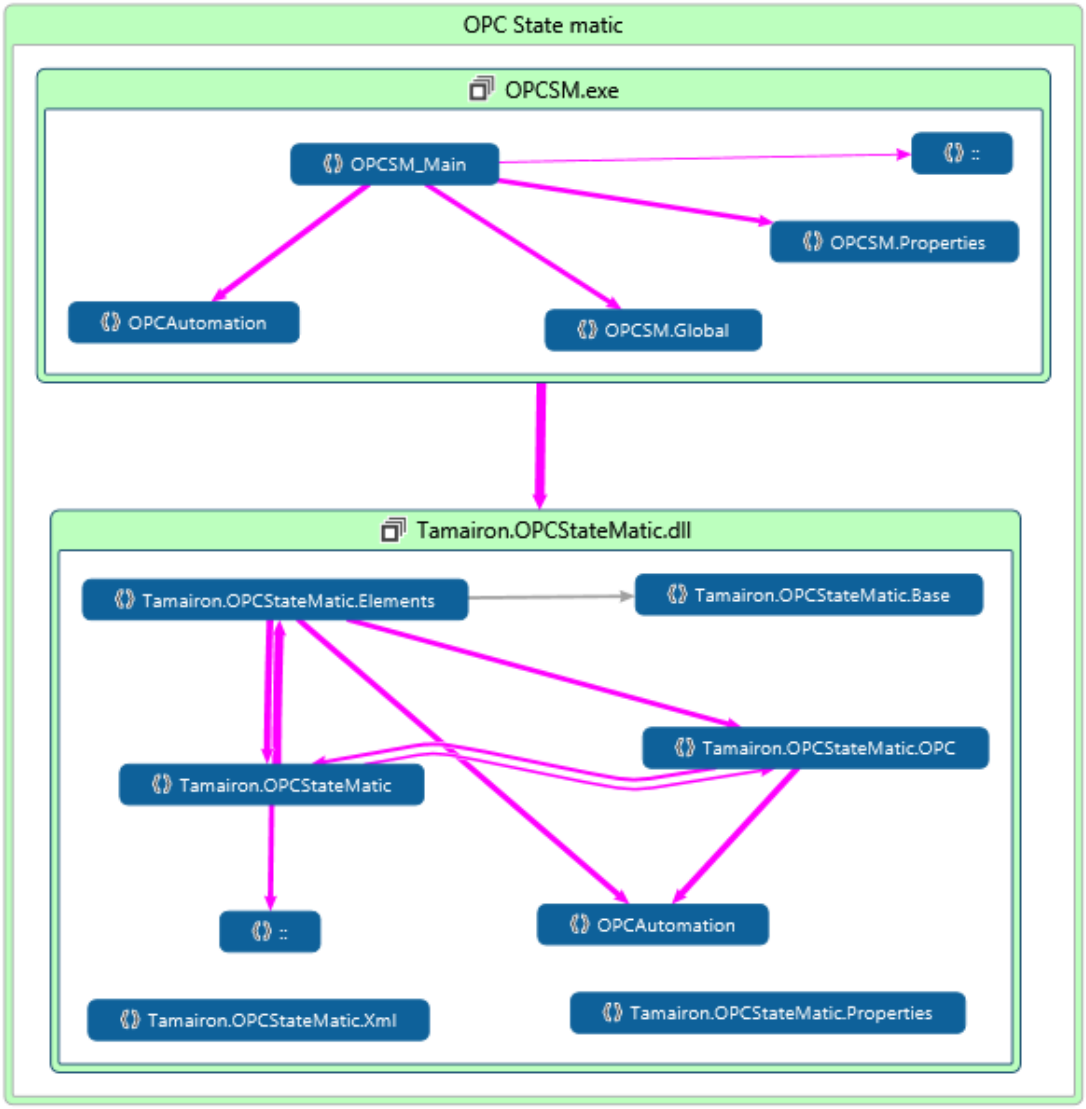
8. Kepware ServerEX – *Kepware* – Servidor OPC – http://www.kepserverexopc.com/kepware-kepserverex-features/

9. GrayBox Simulator – *Graybox Software* – Free OPC Server – http://www.gray-box.net/download_graysim.php

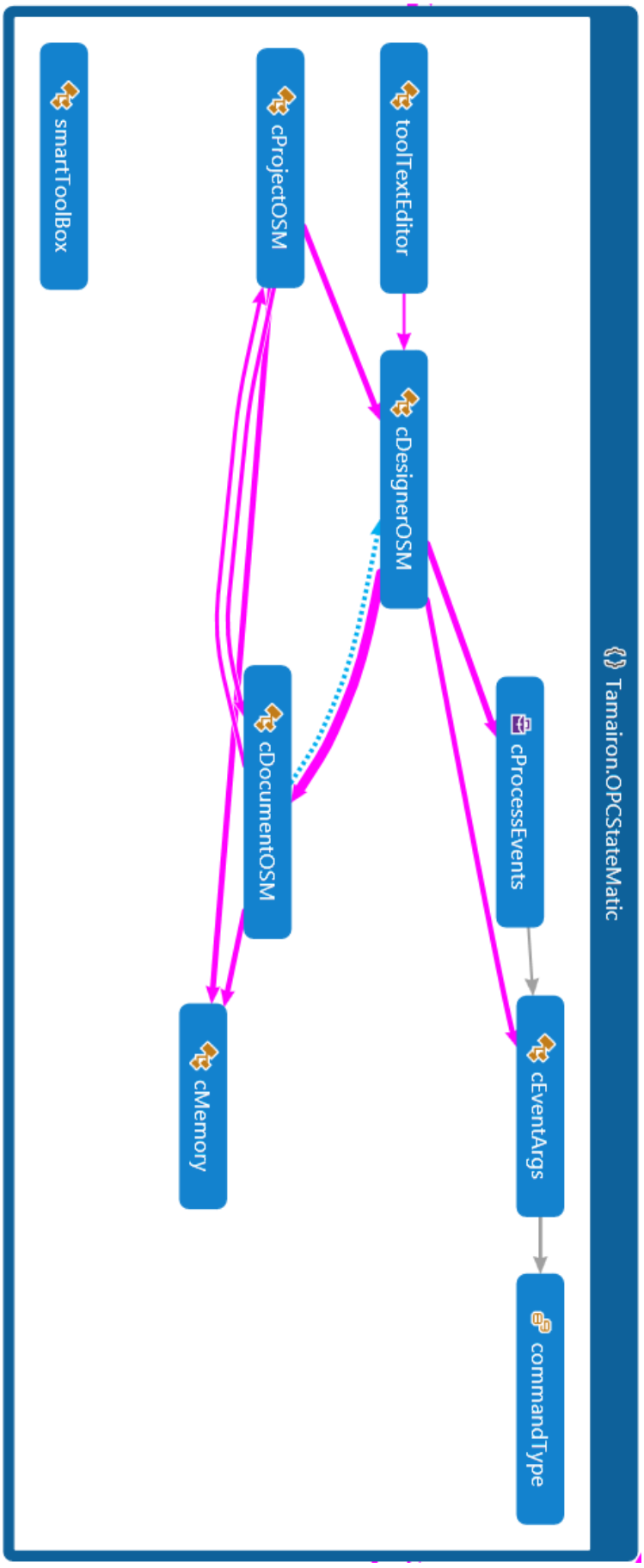
7 ANEXO

7.1 Principales mapas del código

7.1.1 OPCSM.exe y librería principal OPCStateMatic.dll



7.1.2 Conjunto de objetos que forman OPCStateMatic



7.1.3 Librería de elementos activos OPCStateMatic.Elements