TESIS DE LA UNIVERSIDAD DE ZARAGOZA
2016
16

Carlos Catalán Cantero

Modelos y plataforma IEC 61499 adaptados al control distribuido de máquinas herramienta en sistemas de fabricación ágil

Departamento
Informática e Ingeniería de Sistemas

Director/es
Blesa Gascón, Alfonso
Colom Piazuelo, José Manuel

ISSN 2254-7606

Prensas de la Universidad
Universidad Zaragoza

ISSN 2254-7606

Universidad Zaragoza
1542

Tesis Doctoral

MODELOS Y PLATAFORMA IEC 61499 ADAPTADOS AL CONTROL DISTRIBUIDO DE MÁQUINAS HERRAMIENTA EN SISTEMAS DE FABRICACIÓN ÁGIL

Autor

Carlos Catalán Cantero

Director/es

Blesa Gascón, Alfonso
Colom Piazuelo, José Manuel

UNIVERSIDAD DE ZARAGOZA

Informática e Ingeniería de Sistemas

2015

DEPARTAMENTO DE INFORMÁTICA E INGENIERÍA
DE SISTEMAS

Modelos y plataforma IEC 61499 adaptados al control distribuido de máquinas herramienta en sistemas de fabricación ágil

Tesis doctoral

Carlos Catalán Cantero

Directores:
Dr. Alfonso Blesa Gascón
Dr. José Manuel Colom Piazuelo

Teruel, septiembre 2015

A mis padres, Felipe y Pilar

We are all ignorant, but not all ignore the same things.
Atribuida a Albert Einstein

Agradecimientos

Esta tesis es resultado del trabajo realizado a lo largo de varios años con la ayuda de un amplio grupo de personas. De ese grupo son parte importante Alfonso Blesa y Félix Serna, con los que he compartido horas frente a una pizarra, y en algunas ocasiones también frente a una máquina. Alfonso, además de "guiarme" por el buen camino, ha realizado tareas sin las que no habría podido llegar hasta aquí. Por su parte, José Manuel Colom ha creído desde el principio en este trabajo y demostrado un grado de entusiasmo, y paciencia, que agradezco enormemente. Sus conocimientos tanto de la materia específica como del mundo de la investigación han sido imprescindibles.

Igualmente, debo reconocer las aportaciones y espíritu crítico de Josep Maria Rams, así como el apoyo de todo el grupo TUROMAS. Ellos han contribuido a que esta tesis no acabe siendo únicamente un objeto de estantería. Los técnicos de laboratorio Miguel Ángel y David, además de prestarme su ayuda en esas pequeñas cosas que al final hacen que funcionen todo, han hecho este tiempo más agradable. También he tenido la suerte de contar con estudiantes que han experimentado con algunas de las ideas aportadas aquí, son: Oscar, Javier, Ángel, Julio, María, Isabel, Miguel, Tamara, Carla, Alberto y Julián.

Quiero mencionar a Santiago Velilla y Alfredo Roy, a los que debo agradecer mi pertenencia al mundo universitario. La amistad de Alfredo al comienzo de mi vida laboral previa a la universidad, en un entorno difícil, contribuyó a mi formación humana y profesional.

A nivel personal debo dar las gracias a mi compañera por su paciencia y ánimo constante, pero de manera especial por ayudarme todos los días a ser mejor persona. Y finalmente, quiero recordar a mis padres que ya no están.

Carlos Catalán
Teruel, septiembre 2015

Abstract

Manufacturing systems have evolved to follow the changing market needs. Thus, have emerged mass production systems, flexible systems and, finally, agile systems. The latter ones are supported by reconfigurable systems, which can be modified without stopping its operation, and for information and communication technologies, so they can be adapted very quickly to changes in production. To define these new systems have also arisen terms like e-manufacturing, cloud-manufacturing or Industrie 4.0.

IEC 61131 is the reference standard for the development of control software of manufacturing systems. Some studies have indicated its low suitability to fulfil the requirements and complexity of the new systems. For this reason, it has been proposed the more complex IEC 61499 standard, which defines architectures and models for distributed and reconfigurable control software. The industry demands to this standard the ability to develop control software: predictable, scalable, maintainable and extensible. However, although, has been the subject of academic works, today the standard has not acquired that ability, so it is not yet accepted by the industry. The subject of this thesis is to provide proposals to contribute that the standard reaches that ability. To this end, component-based software development methodologies and models adapted to domain are proposed. Particularly, in the domain of the control of machine tools for agile manufacturing systems. This domain has been chosen for its complexity compared with simple use cases considered in previous proposals with IEC 61499.

This work begins with the study of that domain and the establishment of the design principles of its control software. These principles are used to review the state-of-the-art of the standard. Then, a proposal of distributed control for a generic machine tool is presented. The modeling of this control allows to establish function block and execution models of IEC 61499 adapted to this domain. These models provide a design methodology and allow an implementation of the standard deterministic, efficient, scalable and that meets real-time constraints. For experimental verification a platform is required. Previous platforms of IEC 61499 are not support such models, so has been specified, designed and implemented the COSME platform. Unlike previous platforms, it incorporates features which facilitate its use in industrial environments. COSME has been developed in a project of technological cooperation and research between university and a manufacturer of machine tools, which has allowed that the platform, methodology and adapted models have been validated in real cases of use.

Resumen

Los sistemas de fabricación han ido evolucionando para adaptarse a las cada vez más cambiantes demandas del mercado, pasando de los sistemas de fabricación en masa a los sistemas flexibles y, finalmente, a los sistemas ágiles. Estos últimos están soportados por sistemas reconfigurables, capaces de ser modificados sin parar su funcionamiento, así como por tecnologías de la información y la comunicación, por lo que pueden adaptarse muy rápidamente a cambios en la producción. Para denominar estos nuevos sistemas han surgido también términos como *e-manufacturing*, *cloud-manufacturing* o *Industrie 4.0*.

El estándar actual de referencia para el desarrollo de software de control en los sistemas de fabricación es IEC 61131, del que algunos trabajos han indicado su poca adecuación frente a los requisitos de los nuevos sistemas. Por este motivo ha surgido el más complejo IEC 61499, que define arquitecturas y modelos para un software de control distribuido y reconfigurable. La industria demanda a este estándar la capacidad para desarrollar software de control: predecible, escalable, mantenible y extensible. A este respecto, aunque ha sido objeto de múltiples trabajos por parte de la comunidad académica, a día de hoy IEC 61499 no ha adquirido esa capacidad, por lo que no es aceptado todavía por la industria.

El objeto de la presente tesis es aportar propuestas que contribuyan a que el estándar alcance dicha capacidad. Con este fin, se propone el uso de metodologías y modelos de componentes software adaptados al dominio de aplicación, en particular, al control de máquinas herramienta en sistemas de fabricación ágil. Este dominio ha sido elegido por su complejidad, frente a los sencillos tipos de aplicación y casos de uso considerados en anteriores propuestas relacionadas con IEC 61499.

Para establecer dichas metodologías y modelos adaptados se estudia en primer lugar el dominio indicado, determinando los principios de diseño de su software de control. Estos principios sirven de base para efectuar una revisión del estado actual del estándar. Seguidamente, se propone y modela el control distribuido de una máquina herramienta genérica, a partir del cual se establecen los modelos de bloque función y de ejecución IEC 61499 adaptados a ese dominio. Dichos modelos facilitan el establecimiento de una metodología de diseño, a la vez que permiten una implementación del estándar determinista, eficiente, escalable y que cumple restricciones de tiempo real.

A la hora de verificar experimentalmente la metodología y los modelos adaptados es necesaria una plataforma de ejecución. Debido a que las plataformas IEC 61499 existentes no soportan dichos modelos se ha especificado, diseñado e implementado la plataforma COSME. A diferencia de anteriores plataformas, ésta incorpora características que hacen posible su empleo en entornos industriales. En este sentido, la plataforma COSME ha sido desarrollada dentro de un proyecto de investigación, transferencia y colaboración tecnológica entre la universidad y un grupo industrial fabricante de máquinas herramienta. Dicho proyecto ha permitido que esta plataforma, la metodología y los modelos adaptados hayan sido validados en casos de uso reales.

Índice general

Índice de figuras

Índice de tablas

Lista de acrónimos

AMS	*Agile Manufacturing System*	Sistema de fabricación que pueden adaptarse a nuevos productos sin parar la producción
CFB	*Composite Function Block*	Bloque función IEC 61499 compuesto por varios FB simples
CNC	*Computer Numerical Control*	Máquina herramienta que realiza procesos de mecanizado de manera automática.
CPS	*Cyber-Physical Systems*	Sistema de control embebido con capacidad de conexión a Internet
DBMS	*Database management systems*	Sistema gestor de base de datos
DMS	*Dedicated Manufacturing System*	Sistema de fabricación en masa de productos
COSME	*COntrol System and Modeling Environment*	Plataforma para el desarrollo de aplicaciones de control distribuido con IEC 61499
ECC	*Execution Control Chart*	Máquina de estados que define el comportamiento de un FB simple IEC 61499
FB	*Function Block*	Bloque función simple IEC 61499
FBD	*Function Block Diagram*	Lenguaje de programación gráfico definido en IEC 61131
FBDK	*Function Block Development Kit*	Entorno de desarrollo para IEC 61499
FBN	*Function Block Network*	Red de bloques función IEC 61499
FMS	*Flexible Manufacturing System*	Sistema de fabricación que pueden adaptarse a nuevos productos
GEMMA	*Guide dÉtude des Modes de Marches et d'arrêts)*	Guía de puesta en marcha y modos de operación de automatismos
HMI	*Human Machine Interface*	Aplicación de interfaz entre personas y computadores
HMS	*Holonic Manufacturing System*	Sistema de fabricación propuesto dentro de los sistemas de fabricación ágiles
IEC	*International Electrotechnical Commission*	Organismo encargado del establecimiento de estándares en ingeniería eléctrica

IL	*Instruction List*	Lenguaje de programación definido en IEC 61131
IP	*Internet Protocol*	Protocolo de comunicación en Internet
IMS	*Intelligent Manufacturing System*	Sistema de fabricación propuesto dentro de los sistemas de fabricación ágiles
IoT	*Internet of Things*	Internet que incluye la conexión de sistemas embebidos (ver CPS)
JIT	*Just in time*	Compilación "al vuelo" efectuada por la máquina virtual de Java)
LD	*Ladder Diagram*	Lenguaje de programación gráfico definido en IEC 61131
MDE	*Model-driven Engineering*	Metodología de la ingeniería de software
MES	*Manufacturing Execution Systems*	Aplicación de gestión de producción
PID	*Proportional Integral Derivative*	Algoritmo usado en controles de posición o velocidad
PLC	*Programmable Logic Controllers*	Computadores de control industrial
RMS	*Reconfigurable Manufacturing System*	Sistemas de fabricación con capacidad de modificar sus características sin detener su funcionamiento
SCADA	*Supervisory Control And Data Acquisition*	Sistema de supervisión de aplicaciones de control
SFC	*Sequential Function Chart*	Lenguaje de programación gráfico definido en IEC 61131
SIFB	*Service Interface Function Block*	Bloque función IEC 61499 para interfaz con servicios de la plataforma de ejecución
SoC	*System on Chip*	Computador integrado en un único chip
ST	*Structured Text*	Lenguaje de programación definido en IEC 61131
QoS	*Quality of Service*	Calidad de los servicios de una red de comunicación desde el punto de vista de los usuarios
WCET	*Worst Case Execution Time*	Tiempo de ejecución utilizado para determinar el cumplimiento de *deadlines* en sistemas de tiempo real

Capítulo 1

Introducción

1.1. Sistemas de fabricación ágil

La industria de manufactura ha alcanzado una gran madurez tecnológica que permite la fabricación de grandes series de productos a bajo coste gracias a los sistemas de fabricación flexible (*Flexible Manufacturing System* o FMS). Estos sistemas se basan en el empleo de elementos como máquinas herramienta y robots, agrupados en celdas/líneas de fabricación distribuidas por la factoría y unidas por elementos de transporte de piezas y/o productos. El control de todos estos elementos se realiza mediante algún tipo de computador, frecuentemente los denominados controladores lógicos programables (*Programmable Logic Controllers* o PLC).

Los FMS han supuesto un gran avance en la tecnología de manufactura, aunque su flexibilidad tiene costes asociados que no existen en los anteriores sistemas de fabricación dedicados (*Dedicated Manufacturing System* o DMS). Dichos costes vienen derivados por el denominado tiempo de transición (*changeover time*) [5], que contabiliza el tiempo requerido para pasar a fabricar un nuevo producto. Este tiempo es la suma del tiempo de adaptación de la celda/línea, junto con sus sistemas de transporte, y del tiempo requerido para alcanzar unos niveles de productividad y calidad similares a los anteriores (*ramp-up*). Su duración puede ir desde unas pocas horas hasta varias semanas, durante las cuales la producción está parada. Un ejemplo típico de este último caso es la adaptación de las líneas de fabricación de automóviles.

En la actualidad la industria de manufactura está condicionada por un gran mercado global, caracterizado por el acortamiento en el ciclo de vida de los productos y por las demandas por parte de los consumidores de productos diferenciados, aún a bajo coste. Este escenario ha hecho necesario el empleo en los sistemas de fabricación de tecnologías y formas de producción industrial que vayan más allá de los FMS.

El visionario estudio *21st Century Manufacturing Enterprise Strategy: An Industry Led View* [6] introdujo en 1991 un nuevo paradigma: la **fabricación ágil**. El estudio indicaba la necesidad de nuevos sistemas de fabricación ágiles (*Agile Manufacturing System* o AMS) que respondieran rápidamente y con satisfacción a

las necesidades cambiantes de los consumidores. Para alcanzar esos objetivos eran necesarios nuevos métodos de organización industrial que permitieran la fabricación bajo pedido de pequeños lotes de productos, incluso de hasta una unidad, manteniendo o mejorando los costes y la calidad obtenidos con los métodos ya existentes. Según [7] este paradigma se define de la siguiente manera:

"The capability of an organization, by proactively establishing virtual manufacturing with an efficient product development system, to (i) meet the changing market requirements, (ii) maximize customer service level and (iii) minimize the cost of goods, with an objective of being competitive in a global market and for an increased chance of long-term survival and profit potential. This must be supported by flexible people, processes and technologies."

Los requisitos fundamentales de los sistemas de fabricación organizados bajo este nuevo paradigma han sido indicados por [8]:

- Integración total de sistemas hardware y software heterogéneos dentro de la empresa o la cadena de suministros.
- Arquitectura abierta para acomodar nuevos subsistemas hardware o software o desmantelar subsistemas existentes "al vuelo".
- Comunicación y cooperación eficiente y efectiva entre departamentos dentro de la empresa o las empresas participantes.
- Consideración de los factores humanos dentro de los sistemas de fabricación.
- Rápida respuesta frente a cambios y perturbaciones internas y externas.
- Tolerancia a fallos, mediante la detección y recuperación, minimizando su impacto.

Con el objeto de desarrollar sistemas de fabricación que cumplan estos requisitos han surgido diferentes iniciativas. En concreto, la Comisión Europea ha lanzado el proyecto Manufuture [9] para desarrollar nuevas estrategias que hagan posible a la manufactura en Europa competir a nivel global. Con este fin propone una *"Adaptive manufacturing focuses on agility and anticipation to permit flexible, small-scale or even single-batch production, through integration of affordable intelligent technologies and process control for optimal efficiency."*. Por su parte, *Industrie 4.0* [10], auspiciada por el gobierno alemán, persigue la creación de factorías inteligentes mediante la interconexión de sus sistemas a través de la denominada Internet de las Cosas (*Internet of Things* o IoT) [11].

Otra iniciativa, está vez desde la propia industria, es el programa de investigación y desarrollo *Intelligent Manufacturing Systems* (IMS) [12]. En el ámbito académico han surgido propuestas como los sistemas de producción evolutivos [13], donde *"Evolvability means the ability to co-evolve with the continuously changing production requirements."* O el concepto de *e-manufacturing*, que persigue que la factoría sea una parte más del proceso de comercio electrónico mediante el uso de las tecnologías de Internet; definida en [14] de la siguiente manera: *"E-manufacturing includes the ability to monitor the plant floor assets and predict the variation and performance loss to dynamically reschedule production and*

maintenance operations, and synchronize related and consequent actions to achieve a complete integration between manufacturing systems and upper-level enterprise applications." Más recientemente ha surgido la denominada *Cloud-manufacturing* [15]. Todas estas propuestas e iniciativas hacen un uso masivo de las tecnologías de la información y la comunicación.

Dentro de estas iniciativas han surgido los sistemas de fabricación reconfigurables (*Reconfigurable Manufacturing System* o RMS) y los sistemas de fabricación holónicos (*Holonic Manufacturing System* o HMS). Debido a su importante influencia en el contexto de este trabajo dichos sistemas son descritos seguidamente con un mayor detalle.

1.1.1. Sistemas de fabricación reconfigurables

Los RMS pretenden aunar las ventajas de los DMS y los FMS sin sus inconvenientes. Como se ha indicado anteriormente, los DMS aunque facilitan costes por producto muy bajos y una gran productividad tienen poca escalabilidad y una rigidez que les impide reaccionar a cambios en la demanda. Por su parte, los FMS sí tienen una flexibilidad que facilite su respuesta a esos cambios, pero sus costes por producto pueden ser más altos y tienen una menor productividad.

Los RMS se proponen en 1999 en el influyente trabajo [5], donde un sistema de este tipo se define como: *"A Reconfigurable Manufacturing System (RMS) is designed at the outset for rapid change in structure, as well as in hardware and software components, in order to quickly adjust production capacity and functionality within a part family in response to sudden changes in market or in regulatory requirements."*

El objetivo principal en estos sistemas es acortar los tiempos de transición, incluso hasta llegar a cero. Es decir, pretenden que el proceso de fabricación esté detenido el menor tiempo posible durante el cambio de un tipo de producto a otro. Con el fin de alcanzar este objetivo, los RMS deben estar diseñados para reconfigurar de manera fácil y escalable las celdas/líneas tanto mecánicamente como a nivel de comportamiento.

La reconfiguración de los sistemas de fabricación debe ser considerada en tiempo de diseño y de fabricación. En el primer caso, la reconfiguración tiene que ver con la adaptabilidad que facilita la reacción a los futuros cambios. Por ejemplo, el diseño de los elementos del sistema de manera que sean reusables. Aunque, es el segundo caso el que marca la diferencia entre este tipo de sistemas de fabricación y los anteriores. La reconfiguración en tiempo de fabricación consiste en la adaptabilidad para cambios "al vuelo", sean éstos planificados por modificaciones en los planes de producción o sobrevenidos por fallos y contingencias.

El comportamiento de los RMS viene definido por el software de control, que debe ser adaptable. La adaptabilidad del software puede lograrse mediante cambios en tiempo de ejecución de parámetros que modifiquen el comportamiento. Un alto grado de adaptabilidad implica la inclusión en tiempo de diseño de acciones que permitan responder a casi cualquier requisito futuro del sistema. Lo que conlleva un software muy difícil de diseñar y mantener, y por tanto con un alto coste.

Por este motivo, la adaptabilidad del software de control de los RMS se logra mediante la capacidad del software para ser modificado en tiempo de ejecución,

facilitando así la respuesta a nuevos requisitos no previstos en tiempo de diseño. Este tipo de software se denomina **software reconfigurable**.

En [16] se han identificado tres tipos de reconfiguración del software de control con un nivel de complejidad creciente:

- **Reconfiguración simple**: El software puede ser reconfigurado manualmente mediante la modificación en ejecución del código. Esta reconfiguración puede llevar el sistema a un estado inseguro o inestable por eventos imprevistos derivados de ésta.
- **Reconfiguración dinámica**: El software puede ser también modificado manualmente en ejecución, aunque ahora su comportamiento permanece estable y predecible. Para ello, las modificaciones son estructurales y realizadas mediante operaciones transaccionales o políticas de cambio que garanticen la no violabilidad de la seguridad o consistencia del sistema.
- **Reconfiguración inteligente**: Es el propio software quien decide automáticamente cuando realizar los cambios. Su base es la reconfiguración dinámica unida al uso de técnicas de inteligencia artificial, habitualmente mediante sistemas multiagente.

La reconfiguración simple es posible desde hace mucho tiempo en los PLC empleados en el software de control de los FMS. Aunque normalmente los cambios en ejecución realizados con estos dispositivos son de tipo menor, quedando en cualquier caso el comportamiento del sistema en manos de la experiencia del técnico que los realiza.

Por el contrario, en los RMS se considera que los cambios deben poder ser efectuados por un operario de producción (reconfiguración dinámica), en vez de por un técnico (reconfiguración simple). Y en algunos casos incluso sin necesidad de intervención humana (reconfiguración inteligente). Una panorámica completa sobre los sistemas reconfigurables se presenta en [17].

1.1.2. Sistemas de fabricación holónicos

Estos sistemas han surgido dentro del ya citado programa *Intelligent Manufacturing Systems*, donde colaboran grandes compañías mundiales, así como algunas universidades. Entre sus resultados más importantes está la propuesta de los nuevos paradigmas de fabricación: fractal, biónica y holónica. En [18] se presenta una comparación entre estos paradigmas y son referenciadas las primeras experiencias desarrolladas.

La principal característica de los *sistemas de fabricación fractales* es la organización de la factoría en niveles recursivamente autosemejantes, algunos trabajos sobre el tema son [19], [20] y [21]. Por su parte, los *sistemas de fabricación biónicos* intentan aplicar los principios y métodos encontrados en la naturaleza desde el nivel celular hasta las sociedades de seres vivos, en [22] se presenta una panorámica de este paradigma.

La repercusión de los dos primeros paradigmas ha sido limitada, en cambio, los *sistemas holónicos* sí han tenido una influencia posterior mayor. Este paradigma

surge a partir del concepto de holón propuesto en [23]. Un holón se define como una unidad básica de organización de naturaleza híbrida que representa un elemento físico o lógico. Este carácter híbrido viene determinado por el hecho de ser a la vez un todo y una parte de un sistema mayor. Es decir, un holón puede ser parte de otro holón. La organización holónica aporta estabilidad de cara a las perturbaciones, adaptabilidad y flexibilidad, así como eficiencia en el uso de los recursos disponibles. Basado en este concepto entre 1991 y 2004, dentro del IMS, fueron propuestos los HMS [24]. Estos sistemas pretenden combinar la estabilidad de las organizaciones jerárquicas ("de arriba abajo") y la flexibilidad de las no jerárquicas ("de abajo arriba").

Los HMS han sido objeto de un importante trabajo por parte de la comunidad académica, ver los trabajos [25] y [26] y más recientemente [16], [27], [28] y [29]. Las propuestas de HMS presentadas en estos trabajos se basan en sistemas multiagente reconfigurables. La complejidad de aplicar restricciones *hard real-time* en estos sistemas [30] hace que la mayoría de ellas aborden únicamente los niveles de planificación y control de producción.

Los HMS son reconfigurables de forma dinámica o inteligente, por lo que sus sistemas de control no pueden realizarse mediante PLC, los cuales solamente permiten la reconfiguración simple. Por este motivo, uno de los resultados más importantes de este tipo de sistemas ha sido la propuesta de arquitecturas y modelos para el desarrollo de software de control reconfigurable [25], materializadas en el estándar IEC 61499 [31]. Dada la importancia de dicho estándar en este trabajo, sus principales características serán abordadas más adelante.

1.1.3. Las máquinas herramienta en el contexto de los sistemas de fabricación ágil

Una máquina herramienta es parte integrante de una celda/línea de fabricación junto con los elementos de ensamblado, transporte y almacenamiento; con los que tiene de trabajar de manera fuertemente coordinada. Su misión es la fabricación de piezas y/o productos mediante diferentes procesos de mecanizado. Estos procesos pueden ser de torneado, fresado, taladrado, corte, etc. e implican el movimiento de piezas y/o herramientas de mecanizado. Los elementos que constituyen este tipo de máquinas son: mecánicos, neumáticos, hidráulicos, eléctricos y electrónicos. El diseño de este tipo de máquinas herramienta viene determinado por un conjunto de requisitos que tienen en consideración aspectos como: a) tipos de mecanizado; b) movimientos de herramientas, materiales, piezas y productos; c) número de ejes de posicionamiento (determinados por el punto anterior); d) modos de operación; e) integración con sistemas de información de la factoría y f) ratios de producción. Desde hace ya tiempo su funcionamiento está automatizado mediante un computador y un software de control. Un importante ejemplo de máquina herramienta son los denominados controles numéricos (*Computer Numerical Control* o CNC), los cuales realizan procesos de mecanizado. En la actualidad estos dispositivos pueden llegar a tener una elevada complejidad al combinar distintos procesos.

La mayoría de CNC son programados mediante el lenguaje definido en el estándar ISO 6983 [32], conocido también como G-Code o RS274D. Los progra-

mas en dicho lenguaje consisten en órdenes de movimientos y operaciones que la máquina debe efectuar. Este lenguaje carece de características habituales en lenguajes de alto nivel (p.e. estructuras de control o identificadores en lenguaje natural). Algunos fabricantes de CNC han intentado paliar esta carencia mediante extensiones propias que han conducido a una falta de portabilidad del código. La obsolescencia de este estándar ha llevado a la aparición de un nuevo estándar para la programación de los CNC denominado ISO 14649 [33] (también conocido por STEP-NC). La complejidad de diseño de los CNC, junto con el empleo de un lenguaje de bajo nivel con extensiones no estandarizadas, hace que su flexibilidad y capacidad de adaptación sea limitada, como indican los trabajos [5], [34] y [35].

La **reconfigurabilidad** de las máquinas herramientas es la característica clave en los sistemas de fabricación ágil. En el trabajo que propuso los RMS [5], o de manera más reciente en [36], ya se indican los principios de diseño de las máquinas herramientas reconfigurables. Estas máquinas deben tener una estructura modular, donde cada módulo físico (p.e. herramienta, eje) dispone de interfaces mecánicas, de potencia (eléctrica, neumática e hidráulica) y de control estandarizadas. De esta manera, es posible modificar rápidamente el comportamiento de la máquina mediante el simple reemplazo de uno o más de sus módulos. Con un enfoque similar en [35] se sugiere el empleo de elementos mecatrónicos autoadaptativos.

La reconfigurabilidad puede ser usada como guía a la hora de determinar otras características fundamentales de estas máquinas herramienta. La primera de ellas es la **escabilidad** que facilita la adaptación del sistema ante incrementos en la demanda de producción. Una segunda característica es la **gestión de cambios de producción** planificados o sobrevenidos. Para ello, es necesaria la comunicación con aplicaciones externas de supervisión SCADA (*Supervisory Control And Data Acquisition*) o de gestión de producción como MES (*Manufacturing Execution System*). Estas últimas encargadas de la planificación, la calidad o la trazabilidad de la fabricación.

Una tercera característica fundamental es una **gestión de fallos** encaminada a miminizar las paradas no programadas. Un mecanismo para ello es la redundancia dentro de la máquina herramienta o, en un nivel superior, dentro de la celda/línea de fabricación. En el primer caso, los módulos que componen la máquina pueden ser sustituidos en acciones de reconfiguración, disminuyendo así las paradas de producción debidas a fallos [25]. Otro mecanismo es reducir el número de fallos mediante el mantenimiento preventivo y predictivo de la máquina [35]. El primero consiste en la computación del número de ciclos o tiempo de operación de elementos críticos que sufren un desgaste o pérdida de propiedades conocido, con el fin de conocer el momento de su sustitución. Por su parte, en el mantenimiento predictivo los fallos son inferidos, mediante sistemas basados en reglas, a partir de la monitorización del estado de determinados elementos de la máquina. Y un tercer mecanismo es la gestión jerarquizada de los fallos. Finalmente, la última característica fundamental de estas máquinas es un **comportamiento seguro** y fiable en todo momento, que evite daños en personas y bienes materiales.

Tras estas consideraciones, a continuación enumeramos las características de una máquina herramienta en el contexto de los AMS:

- Realización de movimientos y posicionamientos precisos de herramientas y/o piezas.
- Reconfiguración dinámica y escalable de elementos del sistema de control.
- Gestión de los modos de operación (marcha, paradas programadas y no programadas).
- Gestión de fallos en relación con los modos de operación y la reconfiguración.
- Trabajo coordinado entre elementos de la celda/línea de fabricación.
- Mantenimiento preventivo y predictivo.
- Comportamiento seguro y fiable.
- Integración con sistemas de información de la factoría, a través de la comunicación con aplicaciones externas SCADA y MES.

1.2. Principios generales de diseño del software de control en sistemas de fabricación ágil

Hemos visto en la introducción anterior que los AMS están constituidos por elementos (p.e. máquinas herramienta) distribuidos físicamente por la factoría, organizados para trabajar de una manera estrechamente coordinada y con una estructura modular que puede ser modificada sin parar el proceso de producción. Teniendo en cuenta esta constitución establecemos seguidamente los principios generales de diseño del software de control de los sistemas de fabricación ágil:

1. **Uso de sistemas abiertos y estándares**: Los sistemas abiertos ofrecen un conjunto de especificaciones disponibles a la industria, materializada en muchos casos en forma de estándares, que facilitan la portabilidad, la interoperabilidad y la reconfigurabilidad del software en entornos heterogéneos. El uso de las arquitecturas y modelos definidos por los estándares evitan el diseño "artesanal" del software de control. Algo todavía frecuente en muchas ocasiones [37], pero que las características de los AMS hacen mucho más difícil.

2. **Uso del paradigma de programación orientada a componentes**: El uso de componentes es frecuente en las disciplinas de ingeniería cuando alcanzan la madurez (p.e. la ingeniería electrónica). Sus principales ventajas son el acortamiento de los tiempos de desarrollo y la mejora de la fiabilidad, debido al uso de elementos previamente probados. En el caso de la ingeniería de software, además de las indicadas ventajas, este paradigma es un mecanismo que facilita la reconfiguración dinámica. Según Szyperski [38],

un componente software se define como: *"A unit of composition with contractually specified interfaces and explicit context dependencies that can be installed independently and is subject to composition by thirds parties."*

Las aplicaciones basadas en este paradigma están formadas por una red de instancias interconectadas, que siguen unas reglas de composición definidas por el modelo de componente. Así como, unas reglas de despliegue, instalación y ejecución definidas por la plataforma subyacente. La reconfiguración de estas aplicaciones es posible modificando dicha red mediante operaciones en tiempo de ejecución de: creación, eliminación, reemplazo y reconexión. Las instancias son creadas a partir de tipos de componentes ya existentes o de nuevos tipos añadidos también en tiempo de ejecución.

3. **Arquitectura orientada al control distribuido**: El control de máquinas herramienta de los FMS se realiza habitualmente mediante un único controlador que accede a la interfaz de proceso a través de módulos de entrada/salida o buses de campo. Estos buses son redes de comunicación con un software especializado y un hardware también especializado (p.e. CANopen, Profibus, SERCOS) o estándar (p.e. EtherCAT). Por otro lado, la coordinación entre máquinas herramienta a nivel de celda/línea se realiza a través de la información de proceso adquirida vía sensores, y en algunos casos también mediante señales de campo cableadas entre controladores.

   Los características de los AMS hacen que un control de este tipo dificulte la reconfiguración y limite la escalabilidad, especialmente si el número de tareas a ejecutar y/o de entradas/salidas es elevado. Por ello, estos sistemas requieren de un control distribuido entre múltiples elementos interconectados, a efectos de su coordinación, mediante una red de comunicación y donde no existe ningún controlador central. El número de controladores (i.e. granularidad) está relacionado con la cantidad de conexiones y la complejidad de éstos. Aspectos que influyen en el cumplimiento de los requisitos de los AMS. Así, un número de controladores más elevado facilita la reconfigurabilidad, la tolerancia a fallos y la escalabilidad. También mejora los tiempos de respuesta del control, por la mayor sencillez de los controladores y el acceso más cercano a la interfaz de proceso. Por el contrario, dificulta su coordinación.

Como resultado de la aplicación de estos principios de diseño el desarrollo de software de control para AMS es más complejo que en los FMS. Complejidad que lleva consigo un incremento de los costes de desarrollo. Según indica [39], en los próximos años estos costes pueden alcanzar hasta el 80 % de los costes totales (actualmente son entre el 40 y el 50 %). Para hacer frente a esta situación puede ser necesario el uso de metodologías de la ingeniería de software, ver por ejemplo la propuesta de [40] para el desarrollo de software de control basado en ingeniería dirigida por modelos (*Model-driven engineering* o MDE). Un inconveniente es la poca familiaridad de los ingenieros de control, encargados del desarrollo de este tipo de aplicaciones, con dichas metodologías. Algunos trabajos, como [41], [42] y [2], en su momento ya han afirmado la necesidad de realizar esfuerzos al respecto en la formación de estos ingenieros.

1.3. Estándar IEC 61499 para el software de control de sistemas de fabricación ágil

1.3.1. Estándar IEC 61131

Los sistemas de control de los FMS se basan en el empleo de PLC. Estos dispositivos realizan un control centralizado y cíclico (*scan*) consistente en: a) adquisición del estado de los sensores; b) ejecución de los algoritmos de control y c) actualización del estado de los actuadores. Los PLC están implementados con hardware específicamente diseñado para tener un alto grado de fiabilidad y disponibilidad en entornos industriales adversos (eléctrica y medioambientalmente). Actualmente, también es habitual el uso de dispositivos denominados SoftPLC. En este caso, el PLC es una aplicación ejecutada sobre un PC compatible dotado de un sistema operativo de tiempo real. Su empleo está cada vez más extendido debido a la disminución de costes que conlleva el uso de hardware estándar, aún en versiones industriales robustas.

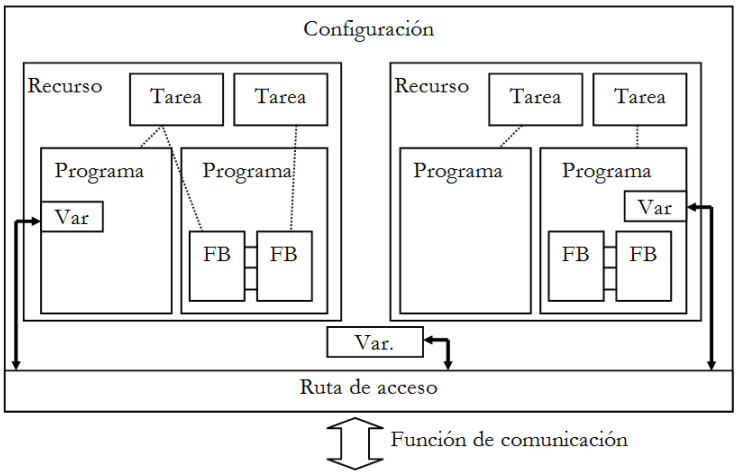


Figura 1.1: Modelo software de IEC 61131

Las reglas para los sistemas basados en PLC han sido establecidas por el estándar IEC 61131 [43], ampliamente aceptado y usado hoy en día por la industria. Este estándar data de principios de los años noventa del siglo pasado y ha representado un gran avance, al facilitar el paso de modelos software propietarios a un modelo abierto que permite la portabilidad de las aplicaciones. Dicho modelo software (ver Fig. 1.1) está definido a partir de:

- **Elementos de alto nivel**: configuraciones, recursos y tareas.
- **Unidades de organización**: funciones, bloques función y programas.
- **Tipos de datos y variables**.

Las configuraciones especifican tipos de sistemas PLC compuestos por: procesadores, memoria, E/S, etc. Dentro de los cuales hay recursos representando los elementos que ejecutan los programas, y donde se definen las tareas de ejecución. Los programas pueden estructurarse mediante funciones predefinidas (p.e. lógicas, aritméticas) o definidas por el programador, no conteniendo en ambos casos información de estado, y mediante bloques función (*Function Block* o FB). Los FB son mecanismos de encapsulación que disponen de datos de entrada y salida, así como de variables internas para almacenar su estado. En el estándar existen FB predefinidos para implementar algunas funcionalidades básicas (p.e. biestables, contadores), además de un conjunto de tipos de datos elementales (p.e. enteros, reales, cadenas de caracteres). Finalmente, las variables pueden ser locales, externas, globales o remotas.

IEC 61131 define cuatro lenguajes de programación, denominados: lista de instrucciones (*Instruction List* o IL), texto estructurado (*Structured Text* o ST), diagrama de contactos (*Ladder Diagram* o LD) y diagrama de bloques función (*Function Block Diagram* o FBD) [44]. Los lenguajes IL y ST son textuales; siendo el primero similar a un lenguaje ensamblador y el segundo al lenguaje Pascal. Por su parte, LD es un lenguaje gráfico que proviene de la antigua lógica de relés, por lo que resulta sencillo de usar por técnicos sin formación en programación. Finalmente, FDB es similar a los diagramas utilizados en ingeniería electrónica o de procesos. El estándar también dispone de un lenguaje para representar comportamientos secuenciales mediante grafos o diagramas de función secuencial (*Sequential Function Chart* o SFC) [44]. En un SFC se definen un conjunto de etapas y transiciones interconectadas mediante enlaces directos. Cada etapa tiene asociado un conjunto de acciones y cada transición un conjunto de condiciones lógicas que determinan la evolución del grafo.

Según indica [45], los PLC y el estándar IEC 61131 tienen serias limitaciones para el cumplimiento de los requisitos de control de los AMS. Como se ha indicado, los PLC solamente permiten la reconfiguración simple. Algo que puede dar lugar a situaciones impredecibles y se considera, en general, una técnica arriesgada. Otra limitación es un modelo de ejecución cíclico que no encaja bien con el control distribuido, ya que puede llevar a grandes *overheads* por la coordinación entre controladores [2] o, en el peor de los casos, a provocar funcionamientos incorrectos según el orden de ejecución. Estas limitaciones han hecho que IEC 61131 no se considere adecuado para el control de los AMS.

1.3.2. Estándar IEC 61499

Uno de los resultados ya indicados de los HMS ha sido el estándar IEC 61499 (ver [31] y para un estudio más descriptivo [2]). Surgido en su primera versión en 2005 y en su segunda en 2012, define una arquitectura y unos modelos software para el desarrollo de aplicaciones reconfigurables de control distribuido. Dicha arquitectura está organizada jerárquicamente mediante los modelos de sistema, dispositivo y recurso. Apoyándose sobre los cuales se establecen los de aplicación y FB, éste último una extensión del anterior FB IEC 61131. Estos modelos son descritos de manera breve a continuación:

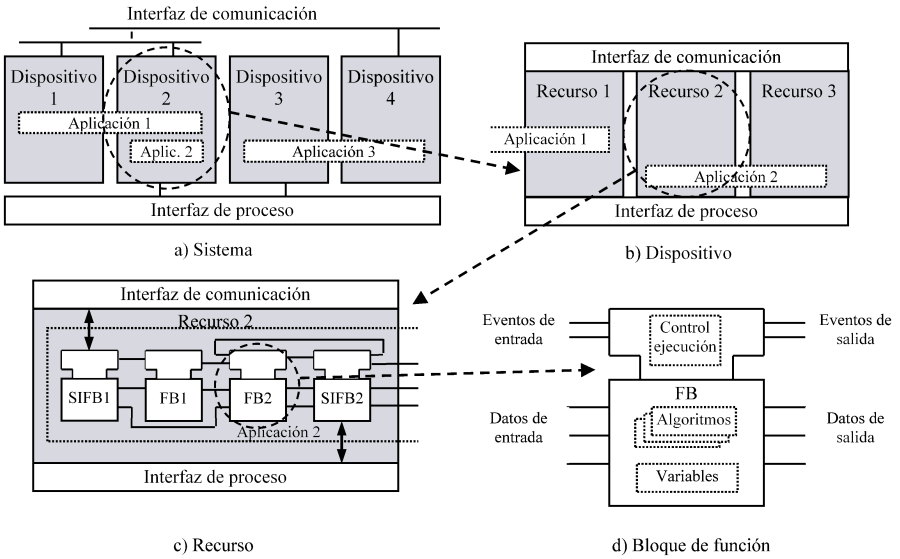


Figura 1.2: Modelos de IEC 61499

- **Sistema**: Un sistema consiste en una colección de dispositivos interconectados mediante alguna red de comunicación. La funcionalidad es modelada por aplicaciones que pueden residir en uno o, de manera distribuida, en varios dispositivos (ver Fig. 1.2a).

- **Dispositivo**: Representa un equipo con capacidad de contener recursos y ejecutar las aplicaciones (ver Fig. 1.2b). Un dispositivo está constituido por interfaces de comunicaciones y proceso, un gestor de dispositivo y cero o más recursos. La interfaz de comunicación suministra servicios de comunicaciones al dispositivo y a las aplicaciones residentes en él. Por su parte, la interfaz de proceso permite el acceso a los sensores y actuadores. Los dispositivos deben gestionar los recursos de manera que no se interfieran y operen como elementos independientes entre sí. Un dispositivo puede no contener explícitamente ningún recurso, pasando en este caso a ser considerado él mismo como un recurso. Con el objeto de la reconfigurabilidad, el gestor del dispositivo administra los recursos y las aplicaciones residentes en él, efectuando operaciones de creación, inicialización, parada, arranque y eliminación. El gestor de dispositivo debe trabajar de manera que las aplicaciones no implicadas no se vean afectadas.

- **Recurso**: Representa un entorno para la ejecución independiente de las aplicaciones o parte de ellas (ver Fig. 1.2c). Éstas se componen de FB que el recurso puede crear, eliminar y conectar entre sí. El estándar organiza el flujo de ejecución de las aplicaciones mediante eventos, siendo el recurso el encargado de gestionar su entrega. Los recursos tienen acceso a las interfaces de comunicación y proceso del dispositivo.

- **Aplicación**: Representa el principal elemento de modelado de la funcionalidad de control. Las aplicaciones se organizan mediante redes de FB (*Function Network Block* o FBN) distribuidas en diferentes recursos, siendo los FB elementos atómicos respecto de la distribución. Desde el punto de vista del estándar, y si consideramos que no existen retrasos significativos o pérdidas de comunicación, una FBN distribuida en recursos ubicados en distintos dispositivos tiene el mismo comportamiento que una localizada en un único recurso. El estándar permite que la asignación de las FBN, o parte de ellas, a los dispositivos y recursos pueda ser realizada después de la fase de modelado.

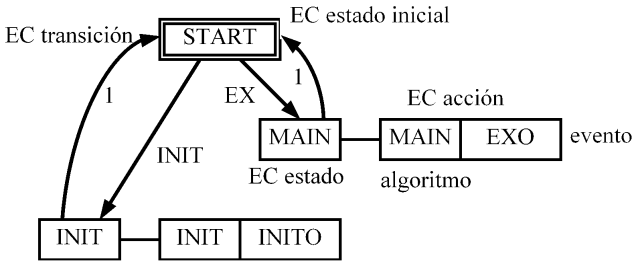


Figura 1.3: Ejemplo de ECC IEC 61499

- **Bloque función**: Unidad básica de modelado constituida por: eventos de entrada y salida, datos de entrada y salida, un control de ejecución (*Execution Control* o EC), algoritmos y variables internas (ver Fig. 1.2d). Los algoritmos contienen las acciones que realiza el FB, mientras que las variables representan el estado interno del FB. Por su parte, los eventos de entrada controlan el flujo de ejecución de los FB según lo especificado en un grafo de control (*Execution Control Chart* o ECC) (ver Fig. 1.3). Como resultado de esta ejecución pueden generarse eventos de salida que corresponden a eventos de entrada en otros FB.

  Los FB pueden agruparse en tres categorías: FB básicos, FB compuestos (*Composite Function Block* o CFB) y FB de interfaz de servicio (*Service Interface Function Block* o SIFB). Los CFB están formados por una red de FB y gestionan las conexiones internas de datos y eventos de esta red igual que un recurso. Representan un mecanismo de encapsulación y son elementos atómicos no distribuibles. Por su parte, los SIFB permiten encapsular las interfaces con primitivas de servicios específicos no contemplados en el estándar. Dependiendo del tipo de evento que provoque su ejecución existen dos tipos de SIFB: a) por eventos externos (p.e. interrupciones hardware de interfaz de proceso) y b) por eventos internos del propio recurso de ejecución (p.e. eventos de puesta en marcha `E_RESTART` o periódicos `E_CYCLE`).

1.4. IEC 61499 en relación con los principios de diseño del software de control en sistemas de fabricación ágil

Desde el comienzo IEC 61499 ha sido origen de múltiples trabajos por parte de la comunidad académica. La panorámica más importante hasta el momento acerca del estándar es [2], trabajo que ha tenido una réplica en [46]. A continuación, se realiza una revisión del estándar en relación con los principios, anteriormente establecidos, para el diseño del software de control de estos sistemas de fabricación (ver Sección 1.2). Algunas de las cuestiones serán tratadas como mayor nivel de detalle en capítulos posteriores.

1.4.1. Uso de sistemas abiertos y estándares

IEC 61499 no ha sido todavía aceptado por la industria de la automatización. Los retos para dicha aceptación fueron ya marcados hace tiempo por [47]: a) escalabilidad, b) mantenibilidad, c) predecibilidad y d) extensibilidad. Todo ellos fuertemente orientados al uso del estándar en aplicaciones reales, y por lo tanto complejas, de control industrial. Por diversas razones, como se verá a lo largo de este trabajo, estos retos están pendientes a día de hoy. Una de estas razones es la definición textual (i.e. no formal) que el estándar hace de su modelo de ejecución orientada a eventos. Esta definición es fuente de ambigüedades semánticas, identificadas hace tiempo en trabajos como [48] ó [49], que han llevado a diferentes interpretaciones a la hora de su implementación. La segunda versión del estándar (2012) ha intentando dar respuesta a esas ambigüedades, aunque todavía hay cuestiones que afectan a los retos indicados y serán discutidas más adelante.

Otro motivo para la no aceptación es la falta de casos de uso reales que validen el estándar. Un motivo para ello es la escasez de plataformas de desarrollo profesionales, en lo que [2] llama "*The chicken and egg problem*". El primer entorno de desarrollo para IEC 61499 ha sido *Function Block Development Kit* (FBDK), asociado a su *runtime* FBRT [50], este último de uso limitado al no ser de tiempo real. Más recientemente, han surgido propuestas en el ámbito académico, entre otras el entorno de desarrollo 4DIAC [51] [52] y su *runtime* FORTE, o con una menor repercusión [53]. El problema, como indica [2], es que estas plataformas han sido usadas únicamente con sencillas aplicaciones en laboratorios de investigación, y además no tienen el grado de madurez de las herramientas y plataformas profesionales que ya existen para PLC. Dos intentos para solventar esta cuestión son los primeros entornos comerciales: ISaGRAF [54], basado en un entorno ya existente para IEC 61131 (ver una comparación con FBDK en [55]), y nxtStudio de nxtControl [56]. Aunque este autor no ha encontrado trabajos donde se describa su empleo en casos de uso real. En este sentido, es importante la realización de experiencias de uso real en el ámbito del control industrial que permitan demostrar su validez, uno de los objetivos marcados en el presente trabajo. Experiencias que posteriormente es necesario divulgar en foros profesionales [57]. Como indica [58], es posible que todas estas cuestiones estén contribuyendo a que diez años después de su aparición IEC 61499 no haya sustituido a IEC 61131.

1.4.2. Paradigma de programación orientada a componentes

De los modelos definidos por IEC 61499 el más importante es el FB. Hay discusión en la comunidad académica acerca de si un FB es un componente software o no. Por ejemplo, para [59] presenta carencias semánticas en su definición de la interacción entre FB, afirmando que no es un componente desde el punto de vista de *programming in the large*. En cambio, [60] dice que la inclusión de la ejecución en su interfaz (datos y eventos) y un comportamiento bien definido por la ECC permiten considerarlo como tal. Por su parte, [48] indica que un FB es un elemento análogo (pero no idéntico) a un componente. En el contexto del presente trabajo se atribuye al FB este carácter, al considerar que cumple la definición de componente software aceptada mayoritariamente hoy en día (ver Sección 1.2).

Las aplicaciones IEC 61499 son modeladas mediante FBN, no siendo este modelado una tarea sencilla. El carácter de componente software del FB hace que éstos deban ser diseñados teniendo en cuenta su reusabilidad e interacción en contextos diferentes [38]. Con este fin, dividir diseños monolíticos en partes, cada una de las cuales se convierte en uno o más FB no es una metodología adecuada. El estándar no ofrece modelos, guías o metodologías que faciliten la fase de modelado de la FBN (p.e. la identificación de tipos de FB). Algo que, en cambio, sí hace en las fases de distribución, despliegue o reconfiguración. Este hecho se puso de manifiesto desde la aparición del estándar, ver por ejemplo [41].

Como se ha indicado de manera general en la Sección 1.2, es necesaria la adopción de metodologías y herramientas de la ingeniería de software a la hora diseñar las FBN. Pueden ser ejemplos el diseño dirigido por modelos (*model-driven design*) o el uso de lenguajes de modelado como UML (*Unified Modeling Language*). Diversos trabajos han realizado propuestas en este sentido, ver por ejemplo [61] ó [62]. Es destacable la temprana propuesta [63], presentada durante las discusiones iniciales del grupo de trabajo que elaboró el estándar, donde se estudian diversos patrones de diseño para IEC 61499. Para una amplia revisión del estado del arte sobre esta cuestión ver [64].

Uno de los aspectos mas relevantes de la consideración del FB como componente software es su modelo de ejecución. Los estándares discutidos en este trabajo siguen dos modelos distintos: a) por *scan* en el caso de IEC 61131, donde los FB son ejecutados de manera cíclica en una secuencia predefinida; y b) por eventos en el caso de IEC 61499, donde los FB son ejecutados únicamente cuando reciben eventos de entrada (ver Subsección 1.3.2). El estándar IEC 61499 ha traído al dominio de las aplicaciones de control el paradigma de la programación orientada a eventos, usado ya en otros dominios: señales vs. espera ocupada en la ejecución de procesos, encuesta vs. interrupciones en operaciones de entrada/salida, etc.

La ejecución por *scan* es menos eficiente que la ejecución por eventos. La razón es su ejecución cíclica de los algoritmos de control, haya o no datos nuevos que procesar. Por el contrario, en IEC 61499 los eventos facilitan la ejecución de los algoritmos de control únicamente cuando haya datos nuevos o situaciones a las que dar respuesta. No obstante, como se verá más adelante, la gestión de los eventos tiene *overheads* a considerar.

Se han propuesto diferentes modelos o aproximaciones de la ejecución por eventos de IEC 61499. Atendiendo a la asignación de recursos de ejecución (tareas, procesos o threads) pueden clasificarse de la siguiente manera:

1. FB [55], [65] y [66]
2. Recursos [48], [49] y [67]
3. Cadenas de eventos [53] y [68]
4. Eventos y algoritmos [69]

Otra clasificación es la propuesta por [70] a partir del método de invocación de los FB:

1. Secuencial [48], [53] y [69]
2. Cíclica [66], [55] y [67]
3. Paralela sobre hardware específico [71]
4. *Non-Preemptive Multi-Threaded Resource* [72]

La definición no formal del estándar plantea algunas ambigüedades semánticas referidas a la invocación de los FB o la evolución en el ECC [48]. En [2] y [73] se discuten posibles modelos de ejecución para el estándar, mostrando las distintas interpretaciones que producen estas ambigüedades que, como se ha indicado, serán discutidas posteriormente.

1.4.3. Arquitectura orientada al control distribuido

En los sistemas distribuidos son fundamentales los aspectos relacionados con la comunicación. En el caso de los sistemas de control distribuido esta comunicación debe ser de tiempo real. En IEC 61499 ha sido propuesto el uso de diferentes redes y protocolos específicos: RT-CORBA [74], MODBUS-TCP [75], RTPS [76] y CIP [77]. No obstante, como en otros ámbitos, hay una convergencia al empleo de los estándares de comunicación de propósito general *Ethernet* e IP (*Internet Protocol*). En particular, el estándar IEEE 802.1D [78] ofrece mecanismos de QoS (*Quality of Service*) a nivel de MAC (*Media Access Control*) que permiten una comunicación de tiempo real sobre redes *Ethernet*. Este estándar se ha materializado en dispositivos denominados *Switched Ethernet* que evitan las colisiones entre mensajes, y por lo tanto los retardos de comunicación no deterministas. Con ello es posible calcular el peor retardo de comunicación, y así aplicar este tipo de redes a sistemas de control distribuido de tiempo real [79]. El uso de estos dispositivos ha sido propuesto en [80] para implementar las comunicaciones en aplicaciones IEC 61499, evitando así el uso de redes y protocolos específicos.

Los modelos propuestos por el estándar están definidos teniendo en cuenta la ejecución distribuida de las FBN. Éstas pueden ser desplegadas sobre varios recursos, que a su vez pueden desplegarse sobre distintos dispositivos. Por motivos de reusabilidad la asignación de las FBN, o parte de ellas, a los recursos puede ser

posterior a su diseño. De hecho, el estándar indica que el comportamiento de una FBN no debe verse alterado por el hecho de estar distribuida, salvo en lo relativo a los retardos o la fiabilidad de la comunicación. A este respecto, el estándar establece los siguientes pasos a seguir para el diseño de una FBN distribuida:

1. Diseño de la FBN sin considerar los aspectos relacionados con la distribución.

2. División de la FBN en partes, las cuales son desplegadas sobre los recursos. Las conexiones de eventos y datos entre recursos son establecidas con enlaces de comunicación, abstraídos mediante SIFB predefinidos en IEC 61499. Dos tipos de enlaces son considerados en el estándar. El primero está basado en el modelo publicación-subscripción y permite únicamente comunicaciones unidireccionales. El segundo está basado en el modelo cliente-servidor y permite comunicaciones bidireccionales. En ambos casos, el estándar define diferentes tipos de transacciones: conexión, desconexión y transferencias; dejando para los detalles de implementación los tipos de datos y el tratamiento de errores.

Los trabajos [68], [70] y [81] han indicado la existencia de algunos problemas con esta aproximación de diseño establecida por el estándar. Un caso es el indicado en [81], donde se muestra el diferente comportamiento de una FBN según se ejecute en uno o en dos recursos, lo que obliga a una modificación de su diseño. Es decir, la manera de desplegar (i.e. distribuir) la FBN sí debe ser tenida en cuenta, más allá de la inserción de SIFB en los enlaces de comunicación.

Un análisis más detallado sobre el diseño de aplicaciones de control distribuido con IEC 61499 puede ser efectuado desde la perspectiva del cumplimiento de los principios, generalmente aceptados, de diseño de los sistemas distribuidos [82]: transparencia, fiabilidad, rendimiento, escalabilidad, seguridad, flexibilidad, heterogeneidad y emulación de sistemas legados.

- **Transparencia**: Este principio se asocia con la ocultación del sistema distribuido subyacente. Existen diferentes formas de transparencia: acceso, localidad, replicación, fallo, migración, concurrencia, prestaciones y escalabilidad. Contemplamos en primer lugar la transparencia de acceso, que permite acceder a los recursos locales y remotos de manera idéntica (no confundir aquí con la definición de recurso de IEC 61499). Si consideramos las conexiones de eventos y datos como el recurso al cual acceder, el estándar no cumple este principio, ya que las conexiones entre FB remotos deben ser realizadas en tiempo de diseño insertando SIFB según uno de los dos modelos de comunicación establecidos. Para lograr esa transparencia las dos soluciones propuestas son: insertar automáticamente los SIFB por medio del entorno de desarrollo una vez establecida la correspondencia entre FB y recursos (aquí en el sentido del estándar) [81]; y no usar SIFB, reemplazados por alguna capa de la plataforma subyacente [53]. Tampoco la transparencia de localidad se cumple, al estar los FB y los recursos fuertemente vinculados a los dispositivos. Otro tipo de transparencia es la de replicación como mecanismo para incrementar la fiabilidad del sistema. En este sentido, y considerando los FB como recursos a replicar, se han presentado trabajos

como [83] y [84]. En relación con la transparencia frente a fallos, por ejemplo en [83] se estudian distintos algoritmos maestro-esclavo de recuperación; y respecto de la transparencia de migración en [27] se recopilan diferentes propuestas. La transparencia de concurrencia está relacionada con el modelo de ejecución, aspecto que ha sido tratado anteriormente. Finalmente, la transparencia de prestaciones y escalabilidad son abordados más adelante.

- **Fiabilidad**: En los FMS la fiabilidad es considerada principalmente desde el punto de vista del hardware de los PLC, ya que tradicionalmente el software ejecutado en estos dispositivos ha sido relativamente simple. Por el contrario, los requisitos de control de los AMS llevan a un software más complejo. Concretamente, en IEC 61499, la ejecución distribuida, la orientación a eventos y las ambigüedades semánticas hacen más difícil asegurar la fiabilidad [37]; por lo que es especialmente importante el modelo o aproximación seguida a la hora de implementar la ejecución por eventos. En este sentido, desde hace algún tiempo se han abordado las cuestiones relacionadas con la verificación y validación formal de FBN en IEC 61499 [2], y mas recientemente [85], [86] y [87]. Por su parte, [88] propone la replicación de FBs como mecanismo de tolerancia a fallos.

- **Rendimiento y escalabilidad**: En las aplicaciones IEC 61499 estos principios tienen que ver con el tamaño de la FBN, el número de controladores, así como con la comunicación entre ellos. Por ejemplo, en [76] se analizan las prestaciones y escalabilidad en IEC 61499 en relación con el empleo de protocolos de comunicación estándar. Aunque son los dos últimos factores los que tienen más influencia en los principios indicados. De manera general, éstos son difíciles de alcanzar en aplicaciones reales, compuestas por grandes FBN [47] que demandan un uso elevado de recursos de la plataforma subyacente. Trabajos como [89] y [90] ó [91] han abordado esta cuestión.

  Estos principios están relacionados con el cumplimiento de las restricciones de tiempo real. Dicho cumplimiento requiere la determinación del tiempo de ejecución del peor caso (*Worst Case Execution Time* o WCET). Existen algunas propuestas al respecto en IEC 61499. Por ejemplo, en [92] se presenta una propuesta basada en *model-checking* y en [53] se determina el WCET para un *framework* específico. Por su parte, en [93] se propone un método general a partir del uso de modelos software de los FB. Para un estudio sobre la ejecución de tiempo real en IEC 61499 ver [4].

- **Seguridad**: Los aspectos de seguridad en las comunicaciones son importantes en un sistema de control distribuido, de cara a evitar riesgos en personas y equipamientos. Dos son los ámbitos en IEC 61499 relacionados con este principio: la comunicación entre controladores y la comunicación con las aplicaciones externas. Aunque el estándar considera los aspectos de comunicación, y por lo tanto de seguridad, dependientes de la implementación, algunos trabajos han presentado propuestas relacionadas con este principio [94] [95]

- **Flexibilidad**: La flexibilidad la podemos contemplar como la característica de los sistemas distribuidos que permite la incorporación de nuevas funcionalidades de forma transparente al usuario y con mínimas interrupciones en el funcionamiento. Este principio queda perfectamente cubierto por IEC 61499, uno de cuyos objetivos principales es alcanzar un grado máximo de flexibilidad mediante la reconfigurabilidad. Algunos trabajos representativos son [30] donde se presenta la primera plataforma para desarrollo de aplicaciones reconfigurables con el estándar y [96], que aborda cuestiones metodológicas relacionadas con la reconfiguración, aportando algunos datos experimentales sobre el comportamiento en laboratorio de una aplicación reconfigurable. Por su parte, en [97] se propone el uso de arquitecturas orientadas a servicios (*Service-Oriented Architectures* o SOA) como medio para alcanzar la flexibilidad en este tipo de aplicaciones.

- **Heterogeneidad**: El principal trabajo inicial relacionado con principio es [72], donde estudian la interoperabilidad entre dispositivos heterogéneos respecto de los modelos de ejecución y la portabilidad de código y datos. El trabajo indica la dificultad del cumplimiento de este principio debido a las ambigüedades semánticas del estándar. Mucho más recientemente, en [98] se estudia la misma cuestión a la vista de la segunda versión del estándar, indicando que aunque la mayoría de las citadas ambigüedades han sido resueltas todavía quedan problemas potenciales en relación con este principio.

- **Emulación de sistemas legados**: Existe una gran cantidad de aplicaciones realizadas con el anterior estándar IEC 61131, por lo que en algunos casos puede ser necesaria su emulación o migración a IEC 61499. Trabajos relacionados con este principio son [99], [100] y [101]; donde se proponen metodologías generales para una migración manual. Por su parte, en [102] se presenta un caso particular de migración y en [66] un generador de aplicaciones basado en el modelo de ejecución en *scan*, utilizado en los PLC, que facilita la migración. En este sentido, en [103] se discute las relaciones entre los dos estándares en términos de semántica de la ejecución, tipos de datos, etc. Finalmente, en [104] se propone una aproximación a la migración basada en el empleo de ontologías que representan programas, tareas, funciones, etc.

1.5. Plataformas y casos de uso IEC 61499

Son escasos los trabajos publicados que han propuesto plataformas o presentado casos de uso con IEC 61499. Particularmente en estos segundos, para el control de elementos complejos como máquinas herramienta. Algunas de las razones para este hecho han sido ya discutidas en la Subsección 1.4.1. Entre las plataformas más importantes presentadas hasta el momento destacamos a [51], [53] y [54]; cuyas principales características serán discutidas en capítulos posteriores.

Al respecto de los casos de uso, IEC 61499 ha sido aplicado a un sistema automatizado de transporte de equipajes de un aeropuerto [105], al control de sistemas de distribución de energía eléctrica [106], [107], [108] y a domótica en grandes edificios [109]. En el dominio de aplicación considerado en este trabajo podemos citar a

[49], que presenta una línea de fabricación automatizada de calzado; a [110] donde se identifican alternativas para el control de máquinas herramienta; a [111] que propone un CNC de arquitectura abierta y a [112] con el control reconfigurable de líneas de fabricación. Más recientemente, en [113] se propone el uso de arquitecturas orientadas a servicios para monitorizar y planificar la producción de máquinas herramienta. En [114] IEC 61499 se modela el control de máquinas herramienta atendiendo a la demanda de energía durante los procesos de mecanizado. Mientras que, en [115] se implementan *Cyber-Physical Systems* (CPS) basados en IEC 61499 mediante plataformas hardware de bajo coste y en [116] se propone una metodología para el control de CNC adaptativos. Más referencias a otros trabajos con propuestas similares pueden ser obtenidas en [2].

Todos los casos de uso citados tienen en común el hecho de ser únicamente prototipos de laboratorio no validados en condiciones de uso real. Por otra parte, no se aportan pruebas o resultados experimentales acerca del rendimiento, escalabilidad, etc.

1.6. Conclusiones y propuestas

El objeto de este trabajo es el **control distribuido de máquinas herramienta en AMS** siguiendo el estándar IEC 61499. Este tipo de máquinas herramienta son elementos complejos encargados de fabricar piezas y/o productos mediante procesos de mecanización. Las características de dichos elementos han hecho que su software de control incremente la **complejidad** ya existente en el caso de los FMS. La característica principal que distingue a estos elementos de los utilizados en los anteriores FMS es la **capacidad de reconfigurar "al vuelo"** (i.e. sin parar la producción) sus partes mecánicas, eléctricas, electrónicas, así como su software de control. En este último caso, la reconfiguración es efectuada modificando el código en ejecución, como en los PLC utilizados en los FMS, aunque con el requisito añadido de la garantía de **seguridad y consistencia** del sistema una vez realizada la modificación.

En este capítulo introductorio han sido establecidos los principios generales de diseño del software de control que facilitan el cumplimiento de los requisitos de los AMS: a) uso de **sistemas abiertos** y **estándares**, b) paradigma de **programación orientada a componentes** y c) **arquitectura orientada al control distribuido**.

En el caso de los FMS el estándar IEC 61131 ha definido los modelos, arquitecturas y lenguajes de programación del software de los PLC. Años después de su aparición en 1991 el estándar está fuertemente extendido en la industria y hoy en día forma parte de los conocimientos básicos de cualquier técnico o ingeniero de control. Aunque trabajos ya citados han indicado su poca adecuación para su empleo en los AMS, debido su modelo de control centralizado y de ejecución cíclica.

Por este motivo, ha surgido en 2005 el estándar IEC 61499, que define modelos y arquitecturas para el desarrollo del software de control distribuido y reconfigurable requerido en los AMS. El elemento fundamental de modelado es el FB, considerado como un componente software es una extensión del FB definido en IEC 61131. La

principal característica del nuevo estándar es el paso a un modelo de ejecución por eventos. Este modelo facilita la ejecución distribuida de las FBN que constituyen el software de control.

Por otra parte, la complejidad del control distribuido de los AMS hace necesario al empleo de modelos y metodologías de la ingeniería de software. En relación con dicha cuestión en este capítulo se han indicado dos problemas relevantes: a) la falta de conocimiento sobre ellas por parte de muchos ingenieros de control y b) la carencia en IEC 61499 de metodologías o guías específicas, especialmente en las fases iniciales de diseño.

Además de los aspectos metodológicos son también importantes las cuestiones relacionadas con el control distribuido requerido por los AMS. Al respecto, en este capítulo se ha aportado un análisis sobre el diseño de dicho control con IEC 61499, desde el punto de vista de los principios aceptados de diseño de los sistemas distribuidos: transparencia, fiabilidad, rendimiento, escalabilidad, seguridad, flexibilidad, heterogeneidad y emulación de sistemas legados.

Con todas las consideraciones indicadas en mente el objeto de este trabajo es la propuesta de:

Modelos y plataforma IEC 61499 adaptados al control distribuido de máquinas herramienta en sistemas de fabricación ágil.

Estos modelos siguen los principios de diseño establecidos en el presente trabajo para este tipo de sistemas. El uso de modelos adaptados es una solución original a las carencias metodológicas del estándar que facilita el desarrollo del software de control de estos complejos sistemas, a la vez que permite cumplir sus estrictos requisitos de ejecución. Esta propuesta permite enfrentar el estándar a un tipo de aplicación de control industrial real y compleja, aportando soluciones a los retos pendientes que la industria demanda a IEC 61499. En este sentido, parte del trabajo ha sido desarrollado dentro de un proyecto de investigación, transferencia y colaboración tecnológica entre la universidad y un grupo industrial fabricante de máquinas herramienta. Dicho proyecto ha permitido que las propuestas hayan sido validadas en casos de uso reales. A continuación se detallan las propuestas realizadas en esta tesis:

- Conceptuales

    1. *Introducción de metodologías procedentes de la ingeniería de software en el control de máquinas herramienta en sistemas de fabricación ágil.*
    Como resultado de los requisitos de estos nuevos sistemas (p.e. reconfigurabilidad) su software del control se ha vuelto más complejo, indicando algunos trabajos que su coste puede llegar en un futuro cercano a ser un 80 % del coste total. Por este motivo, frente a un diseño "artesanal" es cada vez más necesario el empleo de metodologías de la ingeniería

de software y de paradigmas como la programación orientada a componentes.

Es esta tesis se han introducido estas metodologías y paradigmas por primera vez para el control de máquinas herramienta de los sistemas de fabricación indicados.

2. *Establecimiento de modelos y metodologías software para el control distribuido y reconfigurable con IEC 61499 de sistemas complejos*

   El estándar IEC 61499 ha sido propuesto para el software de control distribuido y reconfigurable en los sistemas de fabricación ágil. Su principal elemento de modelado es un bloque función, extensión del definido en el anterior IEC 61131, que utiliza un modelo de ejecución por eventos. En la actualidad, la mayoría de la comunidad académica considera este elemento un componente, tal y como lo define la ingeniería de software. En este sentido, para el tipo de software de control considerado no puede partirse de la simple división de un diseño monolítico en diferentes partes. Es necesario, el establecimiento de modelos, guías y metodologías que faciliten este diseño. Algo contemplado por IEC 61499 en las fases de distribución, despliegue y reconfiguración, pero no en las fases iniciales (p.e. identificación de los tipos de componentes). Está situación se ve agravada especialmente en el control de sistemas complejos.

   En este tesis se propone el establecimiento de modelos de componentes software adaptados al dominio de aplicación. Esta es una solución original a las carencias metodologías del estándar que facilita el desarrollo del software de control. El uso de estos modelos permite una semántica clara y unas reglas simples para la composición de las redes distribuidas de bloques función, lo que facilita el establecimiento de metodologías de diseño.

3. *Aplicación de IEC 61499 al control distribuido y reconfigurable de máquinas herramientas en sistemas de fabricación ágil*

   Son escasos los trabajos publicados que han aplicado este estándar a dominios de control industrial, y los que lo han hecho consisten en sencillos casos de uso o prototipos de laboratorio no validados en condiciones de uso real. En cambio, sí existen trabajos en otros dominios como la domótica de grandes edificios o el control de sistemas distribución eléctrica.

   En esta tesis se aplica por primera vez IEC 61499 al dominio del control de máquinas herramientas reales. Este tipo de dispositivos son complejos, en particular, para en su control deben tenerse en cuenta: a) la reconfigurabilidad para facilitar cambios en la producción, b) una gestión de fallos encaminada a minimizar las paradas no programadas, c) el mantenimiento preventivo y predictivo, d) la escalabilidad para una adecuada respuesta ante incrementos en la producción y e) un comportamiento fiable y seguro que evite daños en personas y bienes.

   En este contexto, se ha revisado el estado actual de IEC 61499 en relación con los principios de diseño del software de control del tipo de

sistemas considerado: a) el uso de estándares abiertos, b) el uso del paradigma de programación orientada componentes y c) el uso de arquitecturas orientadas al control distribuido, en particular, el cumplimiento de los principios de Liu para el diseño de sistemas distribuidos.

- Metodológicas

  1. *Modelo de bloque función adaptado al control de las máquinas herramienta consideradas*

     Para el establecimiento de este modelo se ha partido de una máquina herramienta genérica mediante la cual se han caracterizado y modelado la arquitectura, las conexiones entre sus diferentes partes y las restricciones de tiempo real de su control distribuido reconfigurable. Esto ha permitido identificar los bloques funcionales, así como sus relaciones estructurales y su comportamiento dinámico,

     El modelo de bloque función adaptado es la base para el diseño de todos los tipos de bloques función en el dominio. Los requisitos para su establecimiento han sido: a) facilitad y al mismo tiempo versatilidad y b) tener una implementación eficiente, escalable y que cumpla las restricciones de tiempo real del dominio.

  2. *Metodología para la construcción de redes mediante los bloques función adaptados*

     Como se ha indicado anteriormente, IEC 61499 no contempla modelos, guías o metodologías que faciliten las fases iniciales de diseño, únicamente se dan unas breves indicaciones: a) diseñar la red de bloques función sin considerar los aspectos de su distribución entre controladores, b) dividir la red en subredes y desplegarlas entre éstos y c) establecer las conexiones inter-controlador mediante uno de los dos tipos de bloques función de comunicación establecidos por el estándar.

     El modelo del bloque función adaptado propuesto en el punto anterior ha permitido establecer una metodología que facilita la construcción de redes distribuidas de bloques función, ya que todos los tipos de bloques y sus conexiones se diseñan a partir de dicho modelo.

  3. *Modelo de ejecución de estas redes distribuidas de bloques función que cumplen con los requisitos de las máquinas herramientas consideradas*

     Los requisitos de cualquier modelo de ejecución de IEC 61499 deben garantizar un comportamiento determinista (o predecible), el cumplimiento de las restricciones de tiempo real y tener una implementación eficiente. Los modelos propuestos hasta la fecha pueden agruparse en dos clases: los que atienden los eventos por orden de generación (modelos secuenciales) y los que atienden los eventos por un orden de ejecución de los bloques función establecido en tiempo de diseño (modelos cíclicos). En la práctica, los primeros deben limitar el tipo de conexiones en las redes distribuidas de bloques función para garantizar un comportamiento determinista y una implementación eficiente, a pesar de lo cual pueden no cumplir las restricciones de tiempo real. Por su parte, los

segundos resultan muy similares al modelo de ejecución de IEC 61131, y por lo tanto tienen sus mismos inconvenientes. Tanto una como otra clase de modelos de ejecución han sido validados únicamente en casos de uso sencillos y con redes distribuidas de bloques función de pequeño tamaño.

En esta tesis se propone un modelo de ejecución determinista, eficiente y que cumple las restricciones de tiempo real adaptado al control de las máquinas herramienta consideradas, las cuales requieren de redes con un gran número de bloques función. Para establecer el modelo de ejecución indicado se ha propuesto un nuevo criterio de atención a los eventos: su orden de prioridad, establecido a partir del modelo de bloque función adaptado.

- Herramientas

  1. *Especificación y diseño de la plataforma de ejecución COSME*

     El número de plataformas propuestas para IEC 61499 es pequeño, la mayoría no han salido del ámbito académico de los laboratorios de investigación y han sido utilizadas en sencillos casos de uso. Respecto de las plataformas comerciales existentes no hay publicados casos de uso de una cierta complejidad en dominios del control industrial, lo que es un hecho relevante en un estándar reciente como este.

     La verificación experimental de los modelos adaptados propuestos requiere de una plataforma de ejecución. Debido a que las plataformas existentes no soportan dichos modelos se ha especificado y diseñado la plataforma COSME. Esta plataforma permite la ejecución, depuración, despliegue y monitorización del software de control. Su principal elemento es un *runtime* que implementa el modelo de ejecución adaptado. Las pruebas realizadas en laboratorio con COSME han aportado resultados experimentales que verifican que las redes construidas con el modelo de bloque función adaptado tienen un comportamiento determinista y que cumple las restricciones de tiempo real, así como una implementación eficiente que permite una ejecución escalable. A diferencia de las plataformas propuestas hasta el momento para IEC 61499, la validación final de COSME ha sido realizada en un caso de uso real: una compleja máquina herramienta comercial.

  2. *Especificación y diseño del entorno de desarrollo Domiciano*

     Los entornos de desarrollo propuestos para IEC 61499 adolecen de lo ya indicado para las plataformas de ejecución. Por este motivo, se ha especificado y diseñado un entorno de desarrollo propio, dentro de la plataforma COSME, denominado Domiciano. Este entorno oculta la complejidad propia del estándar facilitando así el diseño de las redes distribuidas de bloques función. Por otro lado, también incorpora herramientas de desarrollo habituales en entornos comerciales de IEC 61131, una importante carencia de otros entornos de IEC 61499. Esta herramienta ha sido utilizada para el desarrollo del software de control de la máquina herramienta comercial antes indicada.

El resto de capítulos de este trabajo están organizados como sigue:

- En el Capítulo 2 se especifican, en primer lugar, las funcionalidades específicas, genéricas y de control de una máquina herramienta genérica en sistemas de fabricación ágil. A continuación, se propone un control distribuido para dicha máquina determinando su arquitectura, granularidad y caracterizando los tipos de conexiones entre sus diferentes partes. Este control es modelado identificando los bloques que componen cada una de las funcionalidades especificadas, así como sus relaciones estructurales y dinámicas. A partir de este modelado se propone el modelo de FB adaptado y una metodología de diseño de FBN. Finalmente, se presentan algunos casos de uso significativos.

- El Capítulo 3 comienza abordando las cuestiones relacionadas con la ejecución de FBN en IEC 61499, en particular, los problemas derivados de la interpretación del estándar, así como del orden de ejecución de los FB. A continuación, se analizan las aproximaciones y *runtimes* propuestos hasta la fecha para el estándar. Seguidamente, se realiza la propuesta de modelo de ejecución adaptado, discutiendo el cumplimiento de los requisitos del dominio de aplicación considerado. Para finalizar, se analizan cuestiones fundamentales relacionadas con la implementación de este modelo de ejecución.

- En el Capítulo 4 se establecen en primer lugar los principios de diseño de la plataforma COSME. A continuación, se propone la arquitectura de un controlador en esta plataforma y se detalla su implementación, especialmente en relación con el modelo de ejecución adaptado. Seguidamente, se presenta el entorno Domiciano asociado a la plataforma y el *workflow* para el desarrollo de aplicaciones de control. Para ilustrar el funcionamiento de ambos se presenta el caso de uso real de una máquina herramienta de corte de vidrio. Finalmente, se analizan brevemente otras plataformas relevantes propuestas para IEC 61499.

- El Capítulo 5 empieza discutiendo cuestiones previas relacionadas con las pruebas efectuadas a la plataforma COSME. Seguidamente, se describe un primer conjunto de pruebas efectuadas con una FBN sintética, analizando los resultados experimentales obtenidos. A continuación, se presentan y discuten las pruebas realizadas con el caso de uso real de control de la mesa de corte de vidrio presentado en el capítulo anterior. Los resultados obtenidos en ambos casos indican el cumplimiento de los objetivos marcados para la plataforma, y por lo tanto la validez de los modelos adaptados. En particular, se verifica el cumplimiento de las restricciones de tiempo real y la escalabilidad de la plataforma. Finalmente, se discuten pruebas similares realizadas con otras importantes plataformas propuestas para el estándar.

- En el Capítulo 6 se presentan las conclusiones generales y las principales contribuciones de este trabajo, indicando las distintas publicaciones a las que han dado lugar. Para acabar, se proponen líneas de trabajo futuras.

Capítulo 2

Modelo de bloque función IEC 61499 adaptado

2.1. Introducción

Las aplicaciones IEC 61499 están constituidas por FBN distribuidas. En aplicaciones complejas, como las máquinas herramienta consideradas, estas redes pueden llegar a estar formadas por un gran número de FB de distinto tipo. Así, el diseño de las FBN no es una tarea sencilla y como indica [41], incluso pueden ser insuficientes para capturar completamente la estructura y el comportamiento de las aplicaciones de control. Este diseño debe efectuarse teniendo en cuenta los principios en los que se basa IEC 61499: los sistemas distribuidos, la programación orientada a eventos y a componentes (ver Sección 1.4). Para ello, el estándar no establece ninguna metodología o guía, habiéndose propuesto el uso de metodologías de la ingeniería de software (ver Sección 1.4.2), por otra parte, poco conocidas por muchos de los ingenieros de control (ver Sección 1.2).

En este trabajo se propone el uso de modelos de FB adaptados al dominio de aplicación para facilitar el diseño de FBN. Estos modelos deben ser la base para el diseño de todos los tipos de FB que constituyen una FBN en dicho dominio. Los requisitos principales para el establecimiento de un modelo adaptado son: a) facilidad y al mismo tiempo versatilidad para el diseño en el dominio considerado y b) implementación eficiente, escalable y cumpliendo las restricciones de tiempo real impuestas por el dominio.

En el presente capítulo se realiza una propuesta original de modelo de FB IEC 61499 adaptado al dominio de las máquinas herramienta en sistemas de fabricación ágil. Para su definición, en primer lugar se especificarán las funcionalidades y el control de una máquina herramienta genérica en ese dominio. Y en segundo lugar, se realizará una propuesta de control distribuido para dicha máquina. Este control será modelado identificando los bloques que componen cada una de las funcionalidades específicas y genéricas, así como las relaciones estructurales y dinámicas de dichos bloques. Finalmente, a partir de este modelado se definirá el modelo de FB adaptado propuesto.

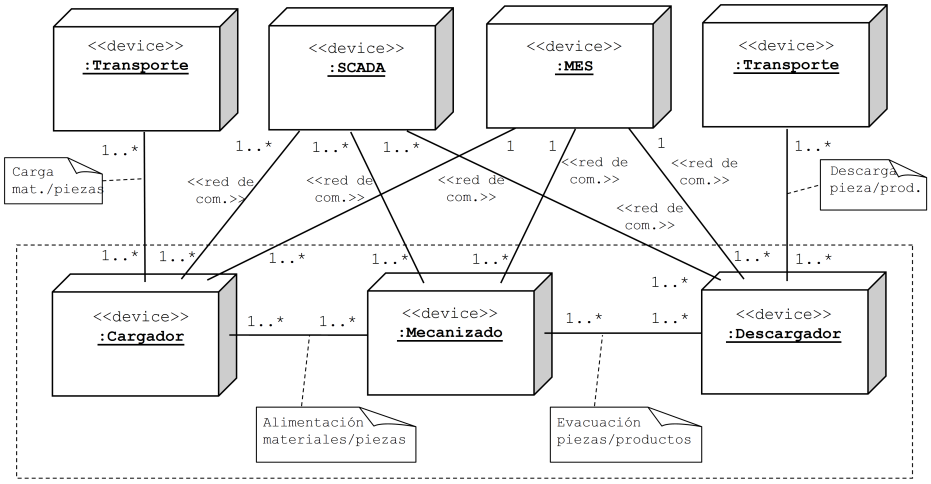


Figura 2.1: Diagrama de despliegue UML de una máquina herramienta genérica en un sistema de fabricación ágil

2.2. Control de una máquina herramienta genérica en sistemas de fabricación ágil

Las máquinas herramienta en los sistemas de fabricación ágil son sistemas complejos en cuyo diseño e implementación intervienen muchas tecnologías. Este tipo de máquinas ha sido descrito brevemente en la Sección 1.1.3, destacando entre sus características específicas más relevantes las siguientes: a) la capacidad para trabajar de manera coordinada con otros elementos de la celda o línea de fabricación, b) la reconfiguración de sus elementos, c) la adecuada gestión de fallos para reducir paradas no programadas y d) la integración en los sistemas de información de la factoría. A partir de sus características se han establecido en la Sección 1.2 los principios para su control: a) el uso de estándares y sistemas abiertos; b) el paradigma de programación orientada a componentes y c) un control distribuido.

Para identificar las funcionalidades de estas máquinas se va a considerar la máquina genérica representada, mediante un diagrama de despliegue UML, en la Fig. 2.1. Esta máquina está compuesta por los siguientes elementos principales: a) `Transporte`, que representa la interfaz de entrada y salida con los sistemas de transporte de piezas y/o productos acabados de la línea de fabricación; b) `Mecanizado`, que representa los elementos de mecanizado (p.e. en caso de una célula de fabricación); c) `Cargador`, que representa los alimentadores de los materiales o piezas a mecanizar; d) `Descargador`, que representa los evacuadores de los productos o piezas ya mecanizados, subproductos o restos de materiales; e) `SCADA` y f) `MES`, que representan a estos tipos de aplicaciones externas.

Las funcionalidades de estos elementos pueden descomponerse en una parte específica, determinada por el tipo concreto de máquina, y una parte genérica.

2.2.1. Funcionalidades específicas

Estas funcionalidades son diseñadas e implementadas dependiendo del proceso a realizar por la máquina herramienta, y vienen determinadas por sus operaciones de mecanizado específicas (p.e. cortado, taladrado, fresado, etc.). Pueden agruparse en dos categorías, ambas con restricciones de control de tiempo real.

1. **Automatización**: Corresponde con las secuencias (p.e. movimientos de piezas, productos o herramientas) realizadas por los diferentes elementos de la máquina, funcionalidad implementada típicamente por dispositivos PLC en los FMS. En base a la máquina genérica considerada identificamos en esta categoría los siguientes tipos de secuencias:

   - **Carga**: Alimentan a la máquina con las piezas a mecanizar suministradas desde los sistemas de transporte de la línea de fabricación.
   - **Mecanizado**: Realizan los movimientos de piezas, productos o herramientas para facilitar las operaciones de mecanizado específicas de la máquina.
   - **Descarga**: Evacuan las piezas mecanizadas (o productos) hacia los sistemas de transporte o almacenamiento de la línea de fabricación.

2. **Posicionamiento preciso**: Corresponde con el posicionamiento preciso de piezas y/o herramientas para las operaciones de mecanizado, funcionalidad implementada típicamente por dispositivos CNC en los FMS. Identificamos en esta categoría las siguientes funcionalidades:

   - **Posicionamiento de ejes**: Realizan el posicionamiento preciso en un eje del sistema de referencia.
   - **Posicionamiento de grupos de ejes**: Agrupan dos o más ejes con el fin de realizar movimientos coordinados sobre varios de ellos (p.e. un eje X y un eje Y para realizar mecanizados de formas geométricas).
   - **Cálculos de trayectorias**: Realizan cálculos relacionados con las trayectorias que siguen los distintos ejes de la máquina durante el posicionamiento. Por ejemplo, a partir de una forma geométrica a mecanizar dividen ésta en puntos intermedios y calculan las velocidades y aceleraciones para realizar los movimientos entre puntos.

2.2.2. Funcionalidades genéricas

Están determinadas por las características, indicadas en el capítulo anterior, exigidas a las máquinas herramienta en los sistemas de fabricación considerados, ver por ejemplo [35]. Pueden ser agrupadas dependiendo de si tienen o no restricciones de control de tiempo real:

- **Con restricciones de tiempo real**:
  - **Interfaz de proceso**: Acceso a los dispositivos sensores y actuadores. En este tipo de sistemas esta funcionalidad se realiza mediante buses de campo (p.e. CANopen, AS-I, Profibus, EtherCAT).
  - **Modos de operación**: Respecto del proceso de fabricación la máquina puede estar operando en diferentes modos: marcha manual, automática o semiautomática; inicialización; parada programada o no programada; etc. Estos modos, así como el paso de uno a otro, pueden ser establecidos siguiendo, por ejemplo, la guía GEMMA (*Guide dEtude des Modes de Marches et d'arrêts*) [117].
  - **Gestión de fallos**: Corresponde con la operación de la máquina cuando alguno de sus elementos falla. Esta funcionalidad se relaciona con los modos de operación indicados en el caso anterior, en particular con las paradas no programadas. La minimización de estas paradas es un requisito de diseño.
  - **Gestión de mantenimiento preventivo**: Computa tiempos, número de ciclos, etc. en el funcionamiento de las partes de la máquina que sufren procesos de desgaste; con el fin de conocer el momento de realización de algún tipo de intervención, como por ejemplo su sustitución.
  - **Gestión de mantenimiento predictivo**: Monitoriza el estado de la máquina con el fin de poder inferir (p.e. mediante sistemas basados en reglas) posibles futuros fallos.
- **Sin restricciones de tiempo real**:
  - **Interfaces de usuario**: Permite a los operadores humanos interactuar con la máquina. Estas interfaces pueden ser locales, ubicadas en algún dispositivo HMI (*Human Machine Interface*) en la propia máquina y/o remotas, normalmente integradas en sistemas SCADA ubicados en planta, oficina, etc.
  - **Gestión de datos de aplicación**: Almacena y/o procesa los datos sobre el propio funcionamiento de la máquina (p.e. paradas no programadas, datos mantenimiento preventivo y predictivo).
  - **Gestión de datos de producción**: Almacena y/o procesa los datos relacionados con la producción (p.e. número y tipo de piezas producidas), suministrándolos a las aplicaciones de gestión de producción.
  - **Comunicación con aplicaciones externas**: Se encarga de la comunicación con aplicaciones de tipo: SCADA, MES, etc. De esta manera, la máquina herramienta se integra en el sistema de información de la factoría.

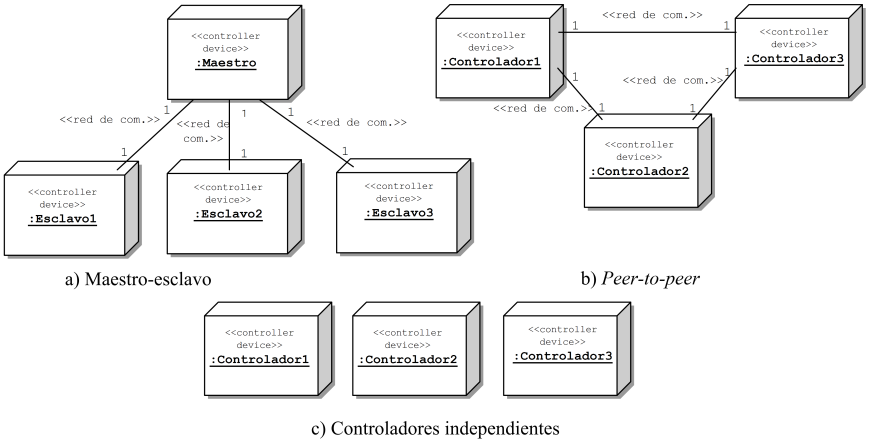


Figura 2.2: Tipos de arquitecturas de control distribuido

2.2.3. Aspectos relacionados con el control distribuido de la máquina herramienta genérica

El control distribuido de este tipo de máquinas tiene especificidades que es necesario considerar a la hora de su modelado. Podemos agruparlas en los tres siguientes aspectos:

- **Arquitectura**. Existen tres tipos de arquitecturas de control distribuido: maestro-esclavo, *peer-to-peer* y controladores independientes (Fig. 2.2). En el primer tipo hay un controlador maestro que dirige (u orquesta) a controladores esclavos siguiendo una lógica de control (o de "negocio") para determinar el funcionamiento de la máquina, los controladores esclavos no tienen comunicación entre sí. Por el contrario, en el segundo tipo no hay un controlador maestro, sino que los controladores se comunican entre ellos con el fin de establecer el funcionamiento coordinado (o coreografiado), estando la lógica de control distribuida. Finalmente, en el tercer tipo la lógica también está distribuida, pero los controladores operan de manera independiente coordinados únicamente a partir de información de estado del proceso adquirida vía sensores y, en algunos casos, también por señales de campo cableadas.

  Las características de los sistemas de fabricación ágiles, indicadas en el primer capítulo (p.e. gestión de fallos, reconfigurabilidad); junto con aspectos relacionados con la ingeniería de software (p.e. reusabilidad, escalabilidad), hacen más adecuado la arquitectura *peer-to-peer* para el control distribuido de la máquina herramienta. Un trabajo al respecto en el contexto del IEC 61499 es presentado en [118], donde se estudian los tres tipos de arquitecturas sobre un caso de estudio, destacando las ventajas de la arquitectura *peer-to-peer*.

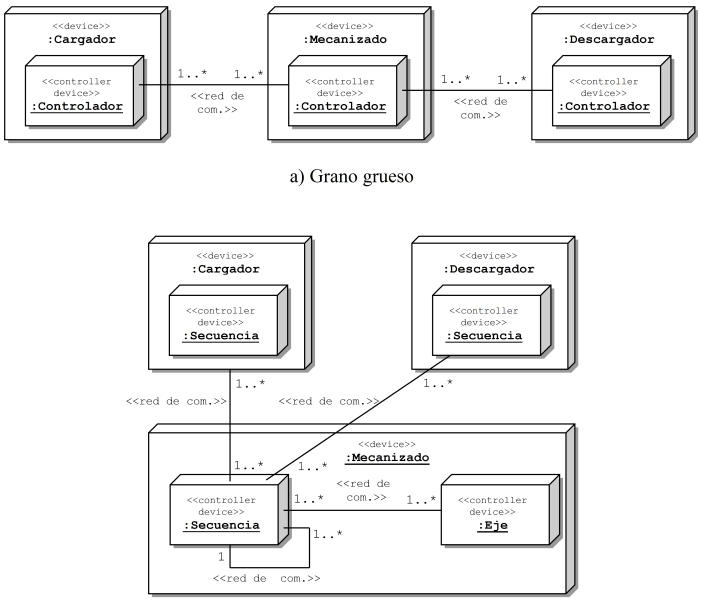


Figura 2.3: Granularidad del control distribuido de la máquina herramienta genérica

- **Granularidad** (o número de controladores). Las funcionalidades del control identificadas en la sección anterior pueden ser distribuidas (desplegadas) sobre un menor (grano más grueso) o mayor (grano más fino) número de controladores. Desde un punto de vista general, y teniendo en cuenta el cumplimiento de las restricciones de control de tiempo real, la granularidad está condicionada por el compromiso entre la complejidad de los controladores y los retardos de comunicación. Así, una mayor granularidad hace que los controladores tengan menos funcionalidades asignadas, por lo que requieren menos recursos computacionales y pueden ser más sencillos, como contrapartida el número de conexiones entre ellos es mayor, haciendo que el control pueda resultar más complejo (p.e. por cuestiones de sincronización). Por el contrario, una menor granularidad incrementa la complejidad de los controladores, al tener más funcionalidades a cargo, aunque disminuye el número de conexiones entre controladores. En este caso, los mensajes que intercambian los controladores tienen un mayor nivel semántico.

  La granularidad está relacionada con el despliegue de las funcionalidades sobre los controladores. En el dominio considerado, y para el caso de grano grueso, cada elemento principal de la máquina puede desplegarse sobre un controlador (Fig. 2.3a). En este caso, los controladores intercambian mensajes para sincronizar (coreografiar) las operaciones o movimientos de piezas y productos durante el proceso de fabricación. Estos mensajes pueden indicar

estados de disponibilidad, operaciones en proceso, fallos, etc. Por ejemplo, un cargador puede indicar al elemento de mecanizado que una nueva pieza está disponible para ser cargada para su mecanizado. Por otra parte, en un despliegue de grano fino las funcionalidades de cada elemento (p.e. secuencias, control de ejes) pueden desplegarse sobre diferentes controladores (Fig. 2.3b). Un ejemplo de mensaje es la indicación por parte de una secuencia al control de un eje de las coordenadas del próximo posicionamiento a realizar. Finalmente, otro aspecto que puede afectar a la granularidad es la reconfigurabilidad. Por ejemplo, una máquina herramienta puede tener partes reconfigurables que realicen operaciones de mecanizado complejas, y por lo tanto requieran su propio controlador.

- **Comunicación entre controladores**. Una aproximación al diseño de arquitecturas de control distribuido, y en general de cualquier tipo de arquitectura, es el diseño multicapa. En el dominio considerado los conexiones entre capas corresponden con los conexiones entre las diferentes partes del control distribuido. Según la granularidad, un mayor o menor número de estas conexiones puede tener un ámbito intra-controlador o inter-controlador. La identificación y caracterización de estas conexiones depende del dominio de aplicación. Una propuesta de diseño multicapa en aplicaciones basadas en IEC 61499 es presentada en [119], donde se establecen tres capas o niveles: interfaz de proceso (acceso a sensores/actuadores), operaciones (secuencias y posicionamientos) y aplicación (lógica de control o de "negocio"). Otra propuesta relacionada es presentada en [76], donde se establece el concepto de "caso de comunicación", similar al de caso de uso, característico de la especificación de requisitos en ingeniería de software.

2.2.4. Tipos de conexiones en el control distribuido de la máquina herramienta genérica

Como se ha indicado, las conexiones entre las diferentes partes en una aplicación de control distribuido pueden ser identificadas y caracterizadas a partir del dominio de aplicación. En el caso de las máquinas herramienta en sistemas de fabricación ágil estas conexiones pueden caracterizarse por los siguientes aspectos: a) latencia típica, b) modelo de comunicación: periódico o aperiódico, c) restricciones de tiempo real y d) ancho de banda. Así, identificamos los siguientes tipos de conexiones (Tabla 2.1):

1. **Control de aplicación**: Corresponde a las funcionalidades de modos de operación y gestión de fallos. Este tipo de conexión tiene restricciones *soft real-time*. Los mensajes pueden tener longitudes típicas de algunos bytes y las latencias típicas puede ser hasta de 100 ms, con un modelo de comunicación aperiódico (p.e. "en marcha", "fallo cargador por presión de aire").

2. **Control de proceso**: Corresponde a las funcionalidades de realización de secuencias automáticas y posicionamiento preciso. A diferencia del caso anterior este tipo de conexión tiene restricciones *hard real-time*, con longitudes

Tipo de conexión	**Latencia típica**	**Modelo de comunic.**	**Tiempo real**	**Ancho de banda**	**Ejemplo**
Control de aplicación	$\leq 100ms$	Aperiódico	*Soft*	Bytes	Gestión de modos de marcha
Control de proceso	1-10 ms	Periódico	*Hard*	Bytes	Bucle de control PID
Interfaz de proceso	$\leq 1ms$	Periódico o aperiódico	*Hard*	Bytes	Acceso a bus campo
Comandos de proceso	10-100 ms	Aperiódico	*Soft*	Bytes	Comando *"ir a X Y"*
Sincronización de proceso	$\leq 100ms$	Aperiódico	*Soft*	Bytes	*"Pieza disponible"*
Interfaz de usuario	100-1000 ms	Periódico o aperiódico	No	KBytes	Supervisión/alarmas
Datos de aplicación	100-1000 ms	Periódico	No	KBytes	Datos de uso para mantenimiento preventivo
Datos de producción	Segundos	Periódico	No	MBytes	Recepción órdenes de trabajo

Tabla 2.1: Tipos de conexiones en el control distribuido de la máquina herramienta genérica

de los mensajes también de algunos bytes. Las latencias típicas están entre 1 y 10 ms, y dependen del tipo de proceso a controlar. Por ejemplo, los posicionamientos precisos pueden exigir valores menores de 1 ms, mientras que en las secuencias automáticas pueden ser suficientes valores de hasta 10 ms. Finalmente, el modelo de comunicación es periódico (p.e. bucle de control de un PID).

3. **Interfaz de proceso**: Corresponde a la conexion con sensores y actuadores mediante un bus de campo. La comunicación tiene también restricciones *hard real-time*, con longitudes de los mensajes de bytes. Las latencias dependerán del protocolo del bus de campo, pudiendo llegar a ser menores de 1 ms y el modelo de comunicación puede ser periódico o aperiódico (p.e. atender las interrupciones de un controlador hardware del bus de campo) dependiendo igualmente de dicho protocolo.

4. **Comandos de proceso**: Corresponde a comandos de ejecución de secuencias automáticas o posicionamientos precisos. Esta comunicación tiene también restricciones *soft real-time*, con longitudes de los mensajes de algunos bytes. Las latencias pueden ser algo mayores que en el caso anterior, con valores típicos de milisegundos, dependiendo en todo caso de la velocidad de ejecución de las operaciones de mecanizado y movimiento. El modelo de comunicación es aperiódico (p.e. "alinear pieza" o "ir a posición X Y").

5. **Sincronización de proceso**: Este tipo corresponde a la lógica de control para el funcionamiento coordinado (o coreografiado) de los distintos procesos ejecutados en la máquina herramienta. Las características de la comunicación

son iguales al tipo anterior. Un ejemplo puede ser un mensaje del tipo "pieza mecanizada".

6. **Interfaces de usuario**: Comunicación con una interfaz local o con aplicaciones externas de supervisión (p.e. SCADA). Esta comunicación no tiene restricciones de tiempo real, y los mensajes pueden tener longitudes de cientos de bytes, así como latencias hasta de 1000 ms. El modelo de comunicación puede ser periódico (p.e. información de estado) o aperiódico (p.e. notificación de alarmas).

7. **Datos de aplicación**: Este tipo de conexión consiste en el envío de información sobre el funcionamiento de la aplicación a algún sistema gestor de base de datos (*Database Management Systems* o DBMS). La comunicación no tiene restricciones de tiempo real, mientras que el modelo de comunicación es periódico. La latencia depende del tiempo de respuesta del DBMS y puede llegar a valores en el entorno del segundo. Un ejemplo es el almacenamiento de datos de utilización de la máquina herramienta para su mantenimiento preventivo.

8. **Datos de producción**: Corresponde con la comunicación con aplicaciones externas de gestión de producción (p.e. MES). Igual que en el tipo anterior esta comunicación es periódica sin restricciones de tiempo real. Las latencias pueden ser de segundos y los mensajes pueden tener tamaño de hasta MBytes. Ejemplos pueden ser: órdenes de trabajo, ratios de producción, etc.

Tipo de conexión	**Ámbito**
Control de aplicación	Inter-controlador
Control de proceso	Intra-controlador
Interfaz de proceso	Intra-controlador
Comandos de proceso	Intra-controlador
Sincronización de proceso	Inter-controlador
Interfaz de usuario	Intra-controlador/Aplicación externa
Datos de aplicación	Intra-controlador/Aplicación externa
Datos de producción	Aplicación externa

Tabla 2.2: Ámbito de los tipos de conexiones en el control distribuido propuesto para la máquina herramienta genérica

2.3. Propuesta de control distribuido de la máquina herramienta genérica

En la sección anterior se ha presentado una máquina herramienta genérica (ver Fig. 2.1) en un sistema de fabricación ágil. A partir de ella se han establecido los requisitos para su control distribuido, que contemplan: a) las funcionalidades específicas y genéricas de la máquina herramienta, b) los aspectos específicos de este control distribuido y c) los tipos de conexiones entre las diferentes partes de

este control. A partir de estos requisitos se presenta una propuesta para el control distribuido de dicha máquina (ver Fig. 2.4) basada en:

1. Una **arquitectura *peer-to-peer*** (ver Fig 2.2b) que elimina la existencia de un controlador maestro, con el fin de facilitar la reconfigurabilidad de la aplicación de control y tener una mayor tolerancia a fallos.

2. Un **control distribuido de grano grueso** (ver Fig. 2.3a) que despliega sobre un controlador la funcionalidad de cada uno de los elementos principales de la máquina herramienta: cargador, mecanizado, y descargador. Esta granularidad disminuye la complejidad de dicho control al limitar las necesidades de sincronización, a la vez que permite un grado de reconfigurabilidad suficiente.

   En cada controlador se establecen dos entornos de ejecución, uno con restricciones de tiempo real y el otro sin ellas. En el primer entorno se ejecutan las secuencias de automatización y el posicionamiento preciso de ejes, comunicadas mediante mensajes de control de proceso con la interfaz de proceso, que a su vez accede al controlador de bus de campo. Estas secuencias y posicionamientos están comandados mediante mensajes de control de aplicación y comando de proceso por una secuencia principal, encargada de la lógica de control intra-controlador. Por su parte, la lógica de control inter-controlador se establece mediante mensajes de control de aplicación y de sincronización de proceso entre las secuencias principales de cada controlador.

   En el segundo entorno sin restricciones de tiempo real se ejecutan: a) la gestión de datos de aplicación y producción, b) la comunicación con las aplicaciones externas y c) la interfaz de usuario. Estas funcionalidades interaccionan con la secuencia principal a través de mensajes de datos de aplicación y producción, así como de interfaz de usuario.

3. Unas **conexiones** entre las diferentes partes de la aplicación de control distribuido caracterizadas por los tipos de conexión identificados (ver Tabla 2.1). Estas conexiones tienen unos ámbitos establecidos a partir de la granularidad indicada en el punto anterior (ver Tabla 2.2).

A continuación se va a modelar el control distribuido de la máquina herramienta genérica propuesto. Para ello, se identifican los bloques que componen cada una de sus funcionalidades específicas y genéricas (p.e. secuencias, control de ejes, comunicación) a partir de los siguientes casos significativos:

- Una interfaz de proceso con un bus de campo.
- Una secuencia automática y el control de un eje comandados por una secuencia principal.
- La inicialización y finalización de la aplicación de control.
- La gestión de fallos de funcionamiento.
- La comunicación con otro controlador y con una aplicación externa.

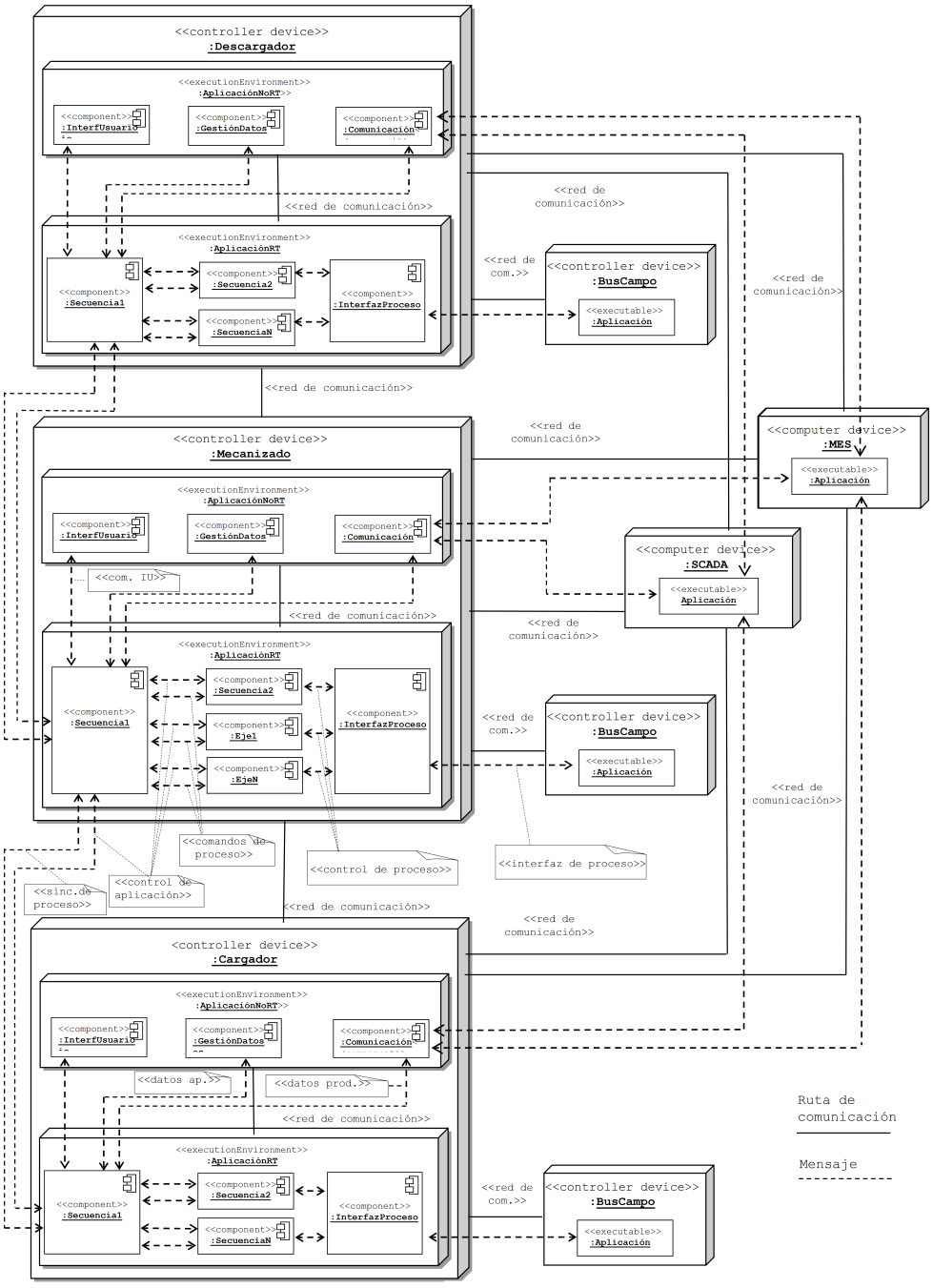


Figura 2.4: Propuesta de control distribuido de la máquina herramienta genérica

En cada uno de estos casos se modelarán los aspectos estructurales y dinámicos (o de comportamiento) mediante el lenguaje UML de la siguiente forma:

1. El **modelo estructural** representado con un diagrama de clase.
2. El **modelo dinámico** representado con un diagrama de colaboración.

   La existencia de restricciones de tiempo real hace adecuado utilizar para estos diagramas el perfil UML/SPT (*Profile for Schedulability, Performance and Time*) [120]. Los elementos en estos diagramas de colaboración se representarán con los siguientes estereotipos:

   - «`SAsituation`»: Entorno de ejecución de tiempo real.
   - «`SAschedRes`»: Tarea planificable.
   - «`SAaction`»: Acción ejecutada en una tarea.
   - «`SAresource`»: Recurso accedido por acciones concurrentes.
   - «`SAtrigger`»: Evento que causa la ejecución de una secuencia de acciones.

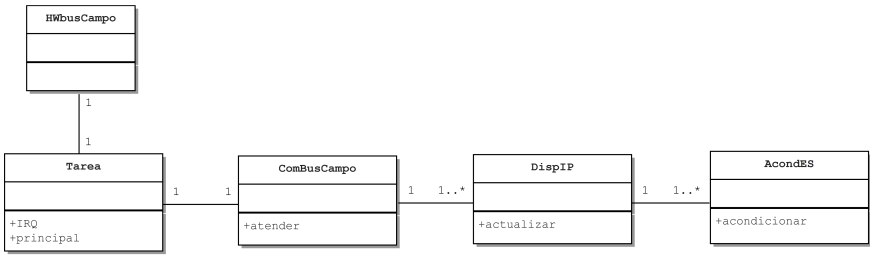


Figura 2.5: Modelo estructural de la interfaz de proceso

2.3.1. Interfaz de proceso con un bus de campo

Esta funcionalidad tiene como misión la adquisición del estado de los sensores y la actualización del estado de los actuadores que permiten controlar el proceso. El acceso a dichos dispositivos se asume que es realizado mediante buses de campo. Los bloques identificados son:

- **Comunicación con bus de campo**: Implementa el protocolo de comunicación con el bus de campo. Debe acceder al dispositivo hardware específico de dicho bus, que debe ser atendido síncrona o asíncronamente (mediante interrupciones hardware) lo antes posible.
- **Dispositivos de interfaz de proceso**: Dispositivos de la interfaz de proceso que pueden representar desde sensores todo/nada que requieren un simple bit (p.e. un sensor de presencia todo/nada), hasta dispositivos que requieren complejas estructuras de datos y lógica de control (p.e. un servomotor).

- **Almacenamiento y acondicionamiento de la interfaz de proceso**: Almacena la imagen de estado del proceso, accedida por la interfaz de proceso, para su utilización durante el ciclo de control. Además, puede realizar acondicionamiento (p.e. linealización, conversiones, contado de pulsos) o generar información de estado añadida (p.e. cálculo de flancos).

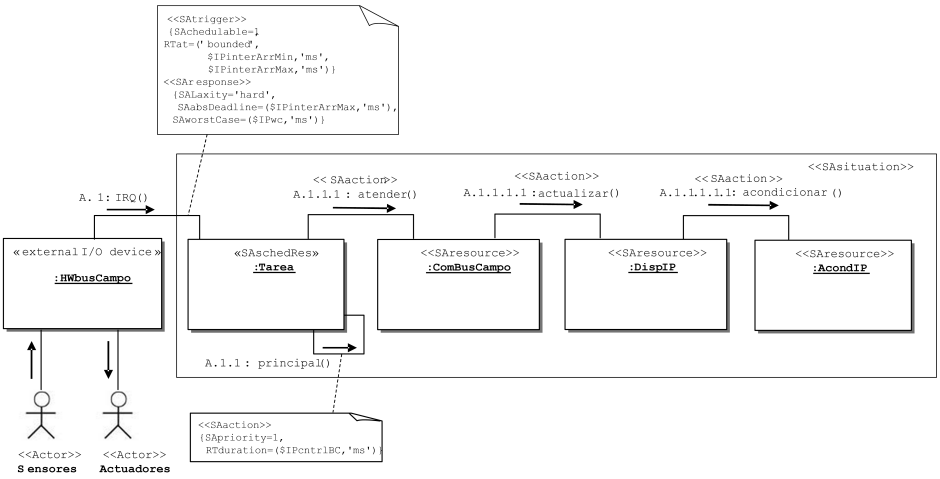


Figura 2.6: Modelo dinámico del caso de la interfaz de proceso con un bus de campo

El modelo estructural de esta funcionalidad es representado por el diagrama de clases mostrado en la Fig. 2.5. Por su parte, el modelo dinámico en el caso de la interfaz de proceso con un bus de campo es representado por el diagrama de colaboración mostrado en la Fig. 2.6. El bloque `Tarea` representa la tarea encargada de recibir, a través del evento `IRQ`, las interrupciones generadas por el dispositivo hardware del bus de campo. Este evento provoca la ejecución de la acción `principal`, la cual realiza la comunicación con el bus de campo mediante la acción `atender` en el bloque `ComBusCampo`. Así como, la atención a los dispositivos y el acondicionamiento de la interfaz de proceso, mediante las acciones `actualizar` y `acondicionar`, en los bloques `DispIP` y `AcondIP` respectivamente.

2.3.2. Secuencias automáticas

Las secuencias se encargan del movimiento automático de piezas o elementos de la máquina para la carga de piezas, su mecanizado o su descarga. Estas secuencias se comandan en cada controlador por una secuencia principal, estableciéndose así dos niveles en la lógica de control (ver subsecciones 2.2.3 y 2.3). Los bloques funcionales identificados son:

- **Secuencia**: Implementa una máquina de estados finita (en automatización industrial frecuentemente programada mediante el lenguaje GRAFCET [121] o su derivado SFC [44]).

- **Almacenamiento y acondicionamiento de la interfaz de proceso**: Bloque ya descrito en la Subsección 2.3.1.

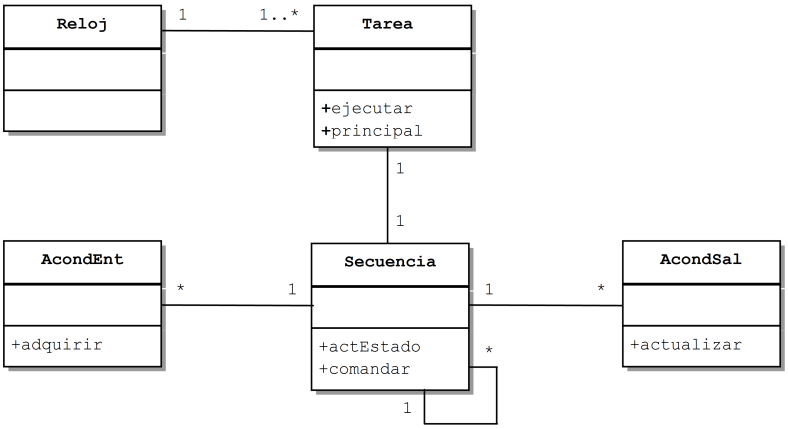


Figura 2.7: Modelo estructural de las secuencias automáticas

La Fig. 2.7 muestra el modelo estructural que representa esta funcionalidad. Mientras que, la Fig. 2.8 muestra el modelo dinámico del caso de una secuencia automática comandada desde una secuencia principal. En este caso, los bloques `Tarea` corresponden a dos tareas de tiempo real invocadas periódicamente desde un bloque `Reloj`, a través de un evento `ejecutar`. Estas tareas ejecutan una acción `actEstado` en el correspondiente bloque `Secuencia`, que hace evolucionar su máquina de estados. En el caso de la secuencia automática (parte inferior de la Fig. 2.8), la imagen de estado del proceso es adquirida y actualizada mediante las acciones `adquirir` y `actualizar` de los bloques `AcondEnt` y `AcondSal` respectivamente, en un tipo de conexión de control de proceso (ver Tabla 2.1).

Por su parte, la secuencia principal controla a la secuencia automática mediante la acción `comandar`, en un tipo de conexión de comando de proceso (ver Tabla 2.1). En el bloque `Secuencia` inferior esta acción y la acción `actEstado` pueden acceder a recursos compartidos de manera concurrente, por lo que es necesario establecer algún mecanismo de exclusión mutua.

Las secuencias principales de cada controlador se comunican entre sí mediante un tipo de conexión de sincronización a proceso, a efectos de establecer el control distribuido *peer-to-peer*. Este caso será considerado posteriormente en la comunicación entre controladores (ver Subsección 2.3.6).

2.3.3. Control de ejes

El control de ejes recibe comandos de proceso con movimientos u otras acciones a realizar. Los movimientos se ejecutan mediante controles de posición y/o velocidad, implementados con PID. Los bloques identificados son los siguientes:

- **PID**: Implementa un controlador proporcional integral derivativo para el control de posición o velocidad de un eje.

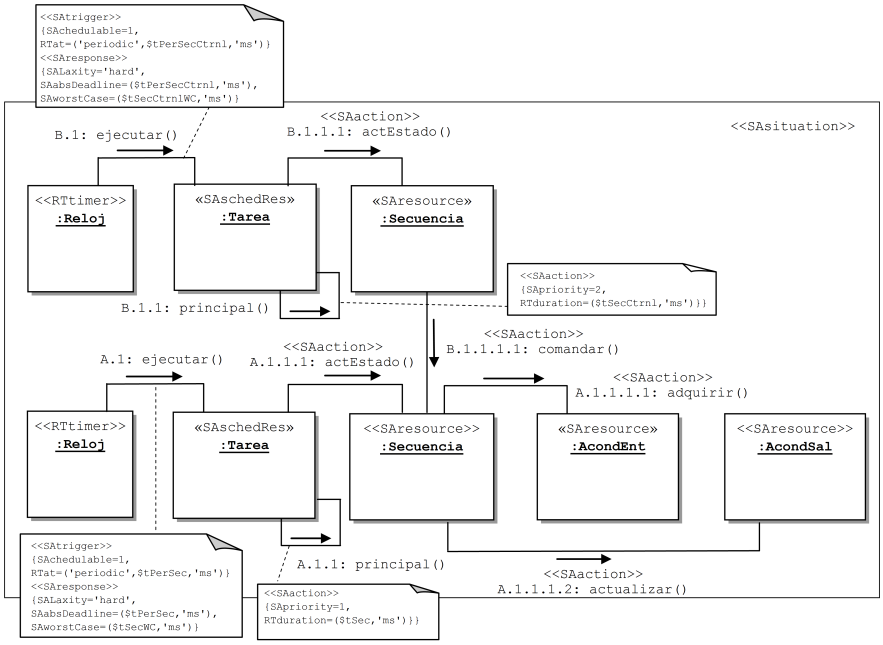


Figura 2.8: Modelo dinámico del caso de una secuencia automática comandada por una secuencia principal

- **Almacenamiento y acondicionamiento de la interfaz de proceso**: Bloque ya descrito en la Subsección 2.3.1.
- **Secuencia**: Bloque ya descrito en la Subsección 2.3.2.
- **Control de eje**: Recibe comandos de una secuencia para la realización de movimientos o acciones de parada, búsqueda de cero, etc. Estos comandos se convierten en posicionamientos que se comunican a uno o más PID, esto último en el caso de movimientos coordinados sobre varios ejes.
- **Cálculo de trayectorias**: Calcula velocidades y aceleraciones de una lista recibida con posiciones a alcanzar. En el caso de formas geométricas esta lista puede ser elaborada en este propio bloque.

El modelo estructural de esta funcionalidad se muestra en la Fig. 2.9, mientras que la Fig. 2.10 muestra el modelo dinámico en el caso de control de un eje. En este control el bloque `Tarea` de la parte inferior representa una tarea de tiempo real activada periódicamente por un bloque `Reloj`. Esta tarea ejecuta el control de posición (o velocidad) representado por `PID`, que accede a la interfaz de proceso a través de `AcondEnt` y `AcondSal`.

Por su parte, el bloque `Tarea` superior representa otra tarea periódica activada por un segundo bloque `Reloj`, la cual ejecuta la acción `actEstado` en `ControlEje`, con el que se dan los comandos de posicionamiento al PID. `ControlEje` también

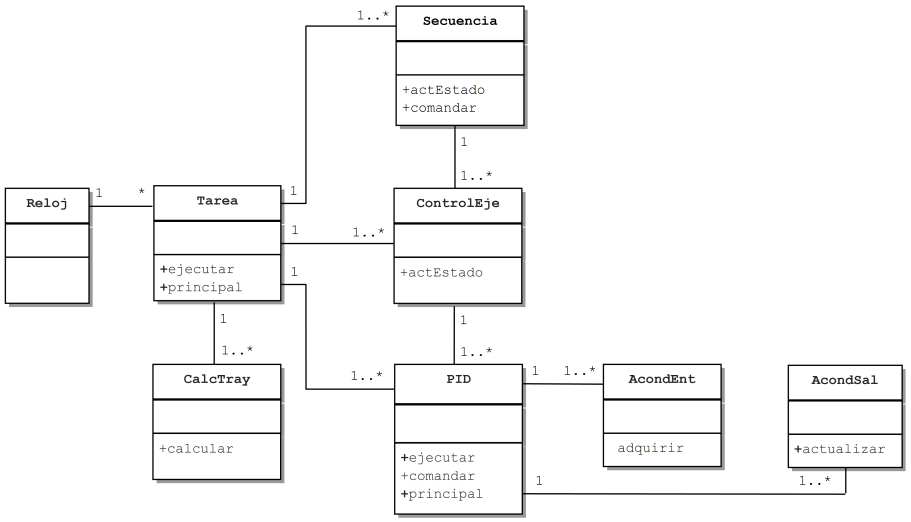


Figura 2.9: Modelo estructural del control de ejes

genera un evento aperiódico `ejecutar` que activa una tercera tarea, encargada de efectuar los cálculos de trayectoria con el bloque `CalcTray`. Dichos cálculos son en coma flotante y pueden requerir un uso elevado de recursos computacionales, por lo que esta tarea se ejecuta sin restricciones de tiempo real y con baja prioridad.

Como en el caso anterior de las secuencias el acceso concurrente de las acciones `actEstado` y `comandar` debe estar protegido mediante algún mecanismo de exclusión mutua. En este caso, esta situación se produce en los bloques `ControlEje` y `PID`. Finalmente, el bloque `Tarea` superior también ejecuta la acción `actEstado` del bloque `Secuencia`, que realiza las secuencias de movimientos con `ControlEje`.

2.3.4. Inicialización y finalización

Las funcionalidades contempladas hasta el momento corresponden con la operación durante el funcionamiento normal (o nominal) de la máquina herramienta genérica. Se van a considerar ahora las funcionalidades de inicialización y de finalización de la aplicación de control. En la primera se pueden distinguir dos casos: inicialización en frío e inicialización en caliente (ver IEC 61131-3, Apartado 2.4.2 [44]). Los bloques identificados son los siguientes:

1. **Inicialización en frío**: Pone la aplicación en un estado inicial predeterminado. Para ello, los atributos (variables) de la aplicación que representan su estado toman como valor inicial el predeterminado en su definición, o si no lo hay, el valor por defecto para su tipo de dato (p.e. cero en caso de tipos numéricos).

2. **Inicialización en caliente**: Pone a la aplicación en el mismo estado anterior a la parada, tomando los atributos el valor que tenían en ese momento.

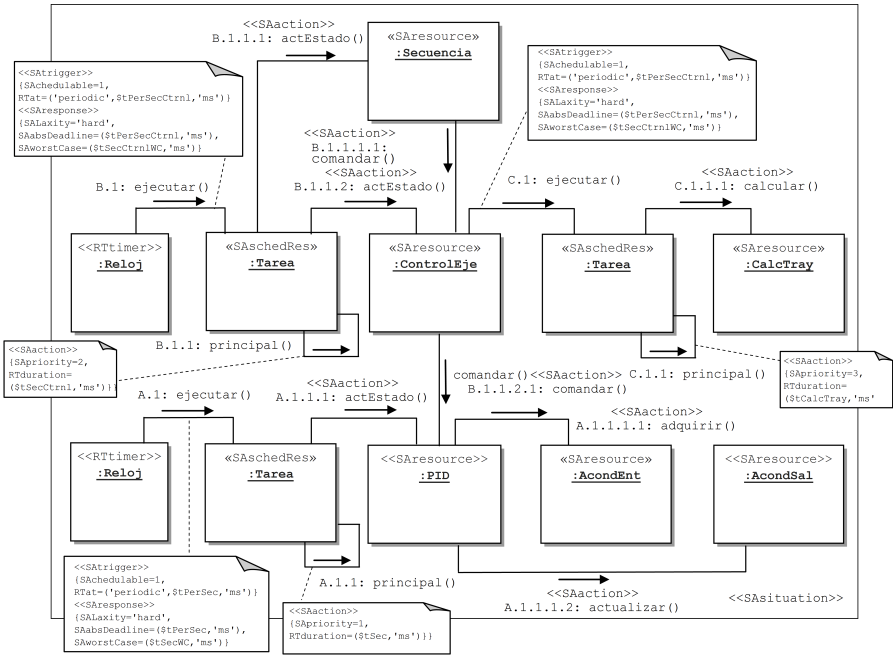


Figura 2.10: Modelo dinámico del caso de control de un eje

Para ello, es necesario que la plataforma de ejecución ofrezca servicios de almacenamiento (persistencia).

3. **Finalización**: Realiza acciones para dejar el sistema en un estado seguro antes de parar la ejecución de la aplicación, o antes de que una parte de ésta vaya a ser sustituida dentro de un proceso de reconfiguración. Ejemplos de estas acciones son: desactivación de actuadores y liberación de recursos del sistema (p.e. memoria, E/S).

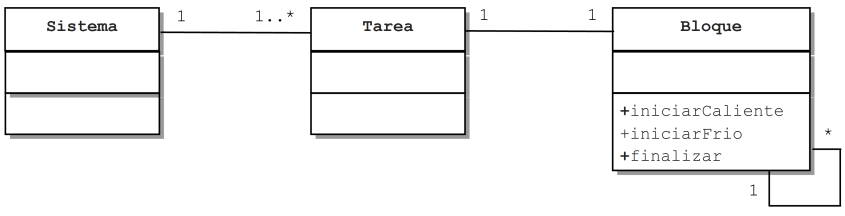


Figura 2.11: Modelo estructural de inicialización y finalización

El modelo estructural se muestra en la Fig. 2.11, donde `Bloque` representa un bloque funcional genérico y `Tarea` una tarea aperiódica que será invocada por la plataforma de ejecución representada por el bloque `Sistema`. Por su parte, el modelo dinámico para el caso de inicialización y finalización de una secuencia es

mostrado en la Fig. 2.12. Los tres bloques `Tarea` corresponden con otras tantas tareas no concurrentes que ejecutan las acciones `iniciarCaliente`, `iniciarFrio` y `finalizar` del bloque `Secuencia`. Este último, a su vez, ejecuta las acciones con igual nombre de `AcondEnt` y `AcondSal`, que inicializan o finalizan la interfaz de proceso asociada a la secuencia.

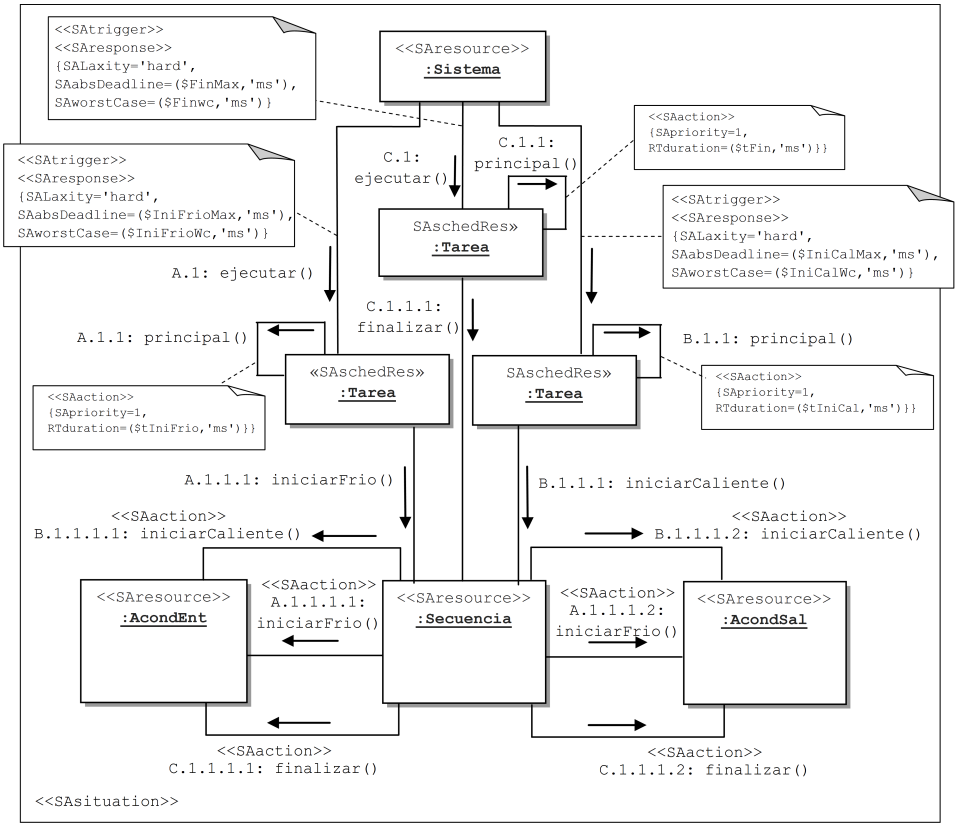


Figura 2.12: Modelo dinámico del caso de inicialización y finalización de una secuencia

2.3.5. Gestión de fallos

Durante la operación de la máquina herramienta pueden darse fallos de funcionamiento (p.e. *"pieza atascada"*, *"fallo presión neumática"*, *"tiempo de carga sobrepasado"*) que producen paradas o una disminución de la producción. La adecuada gestión de estos fallos puede minimizar dichos efectos. Con este fin se propone el concepto de **zona** [122], el cual representa una agrupación de elementos cuyos fallos son gestionados de manera conjunta e independiente, aunque coordinada, del resto de elementos de la máquina herramienta. Las zonas se relacionan de manera jerárquica, al objeto de establecer una política de propagación de fallos.

El funcionamiento de esta gestión jerarquizada de los fallos es como sigue:

1. En un instante determinado se produce un fallo en un elemento de una zona de la máquina herramienta.
2. Dicho elemento notifica a su zona su estado de fallo, así como el nivel de importancia del mismo. En estado de fallo el elemento puede realizar acciones de parada de movimientos, desactivación de actuadores o *data logging* (para posteriores análisis del fallo).
3. La zona notifica al resto de sus elementos el fallo y el elemento que lo ha causado. Los elementos notificados dependiendo del tipo e importancia del fallo pueden seguir en funcionamiento normal, pasar a un funcionamiento degradado o en fallo. En funcionamiento degradado, por ejemplo, pueden limitarse movimientos o velocidades de partes móviles de la máquina.
4. Dependiendo también del tipo de fallo y su importancia la zona lo propaga a sus zonas superiores e inferiores en la jerarquía, que a su vez pueden igualmente propagarlo o no.
5. Una vez que la causa del fallo ha desaparecido la zona puede indicar a sus elementos y zonas superiores e inferiores la vuelta al funcionamiento normal, mediante una notificación de recuperación de fallo.

Una sencilla jerarquía de zonas en la máquina herramienta genérica consiste en establecer una zona por cada uno de sus elementos principales: carga, mecanizado y descarga; más una zona general de nivel superior que agrupe toda la máquina (ver Fig. 2.13). Dependiendo de la complejidad de la máquina (p.e. si tiene varias fases de mecanizado) pueden establecerse jerarquías con un mayor número de zonas y/o niveles.

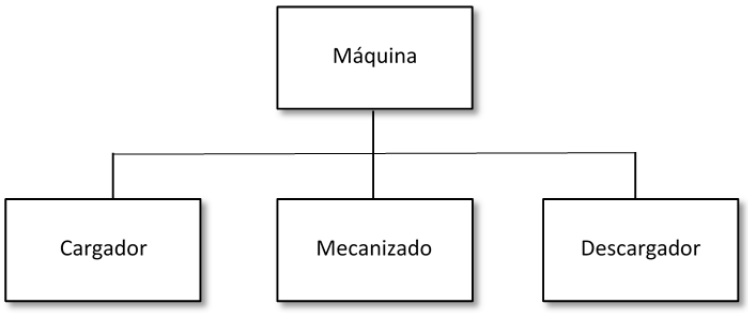


Figura 2.13: Jerarquía de zonas en la máquina herramienta genérica

Con esta división un escenario (o caso de uso) de la gestión jerarquizada de fallos es el siguiente:

1. Se produce un fallo en un elemento de la alimentación neumática del cargador.

2. La zona del cargador recibe la notificación del fallo y lo notifica al resto de elementos del cargador. Los elementos con actuadores neumáticos pasan a funcionamiento en fallo, el resto puede seguir en funcionamiento normal.

3. La zona del cargador notifica el fallo a la zona general de la máquina.

4. Debido a que este fallo no afecta al resto de la máquina la zona general no lo propaga a las zonas inferiores de mecanizado y descargador, que pueden seguir en funcionamiento normal (p.e. mecanizando y descargando las piezas en curso).

5. Una vez que desaparece el fallo en el elemento de alimentación neumática, y tras una acción de "rearme" de un operador humano, la zona del cargador lo notifica a sus elementos para que, si es el caso, vuelvan a funcionamiento normal.

6. La zona del cargador notifica a la zona general de la máquina la recuperación del fallo.

El único bloque específico identificado en esta funcionalidad es:

- **Zona de fallo**: Gestiona los fallos de un grupo de elementos de la máquina herramienta. Las zonas pueden notificar fallos o recibir notificaciones de fallos de otras zonas.

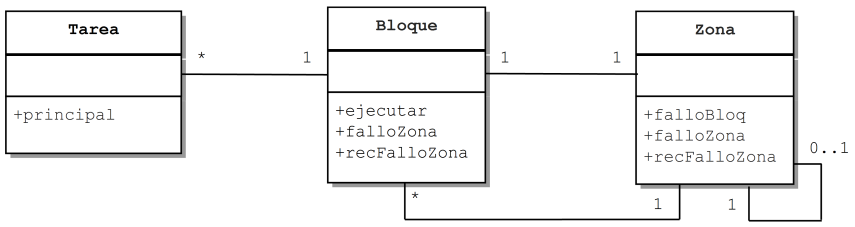


Figura 2.14: Modelo estructural de la gestión de fallos

El modelo estructural de la gestión jerarquizada de fallos se muestra en la Fig. 2.14, donde el bloque `Tarea` representa una tarea con restricciones de ejecución en tiempo real que ejecuta acciones en `Bloque`. Éste representa un bloque funcional genérico (p.e. una secuencia) que controla un elemento de una zona, representada por el bloque `Zona`. La zona puede tener relación con otras zonas.

Un modelo dinámico para el caso de tres zonas y dos niveles jerárquicos, donde las zonas de nivel inferior gestionan los fallos de dos secuencias se muestra en la Fig. 2.15. El bloque `Tarea` ejecuta la acción `ejecutar` del bloque `Secuencia` superior. Éste en un momento dado puede notificar a su bloque `Zona` un fallo de funcionamiento, que lo notifica al otro bloque `Secuencia` de la zona y al segundo bloque `Zona` superior. Esta última zona puede notificar el fallo al bloque `Zona` inferior, en cuyo caso ésta lo notifica también a sus dos bloques `Secuencia`. Una vez subsanado el fallo la primera zona lo notifica a sus elementos, así como a la zona superior, volviendo todos los elementos a su funcionamiento normal.

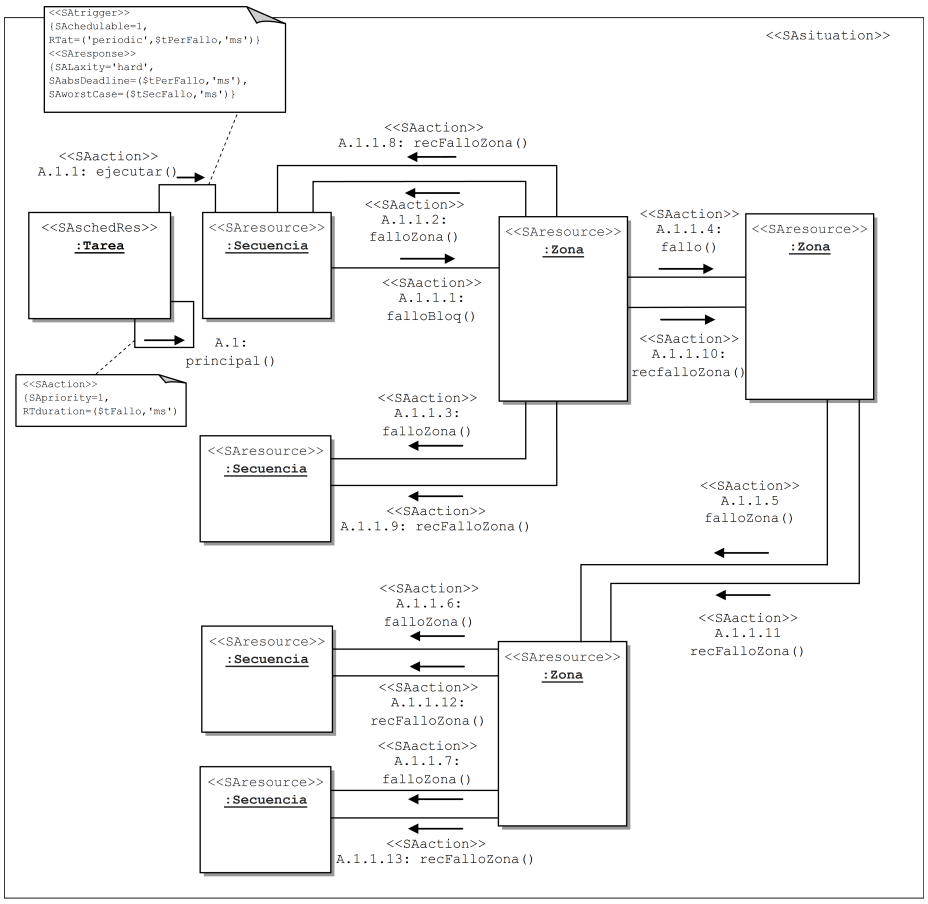


Figura 2.15: Modelo dinámico de una gestión de fallos con tres zonas y cuatro secuencias

2.3.6. Comunicación con otros controladores y aplicaciones externas

La comunicación entre controladores corresponde a los tipos de conexión de control de aplicación y sincronización de proceso (ver Subsección 2.2.4). El modelo estructural de esta funcionalidad es mostrado en la Fig. 2.16, donde el bloque `Secuencia`, invocada por el bloque `Tarea`, intercambia mensajes de control de aplicación y sincronización de proceso con secuencias principales en otros controladores mediante el bloque `Comunicación`. El modelo dinámico de un caso de sincronización entre dos de estas secuencias es mostrado en la Fig. 2.17.

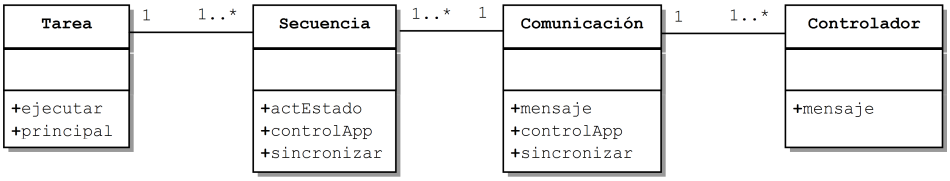


Figura 2.16: Modelo estructural de la comunicación entre controladores

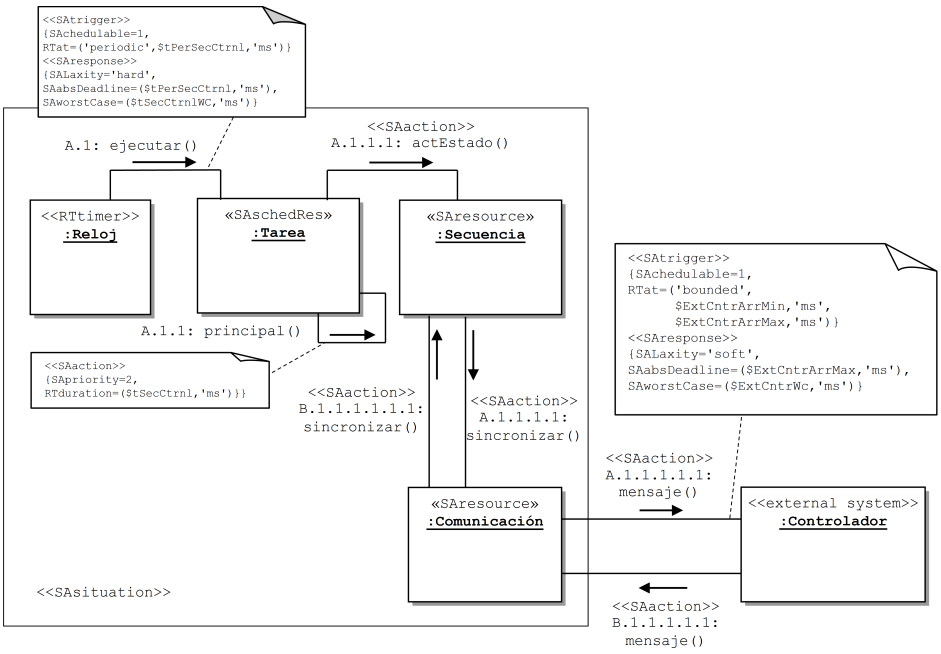


Figura 2.17: Modelo dinámico del caso de la sincronización entre dos secuencias principales

Respecto a la comunicación con aplicaciones externas, ésta corresponde a los tipos de conexión de interfaz de usuario, datos de aplicación y datos de producción

(ver Subsección 2.2.4). En este caso, el modelo estructural, similar al anterior, y el modelo dinámico son mostrados en la Fig 2.18 y la Fig. 2.19 respectivamente.

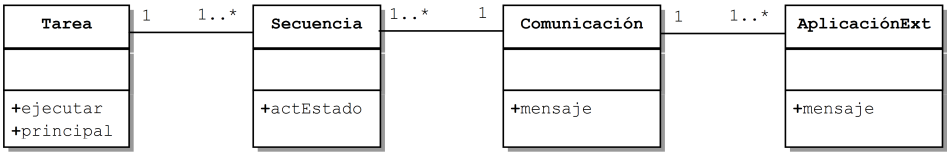


Figura 2.18: Modelo estructural de la comunicación con aplicaciones externas

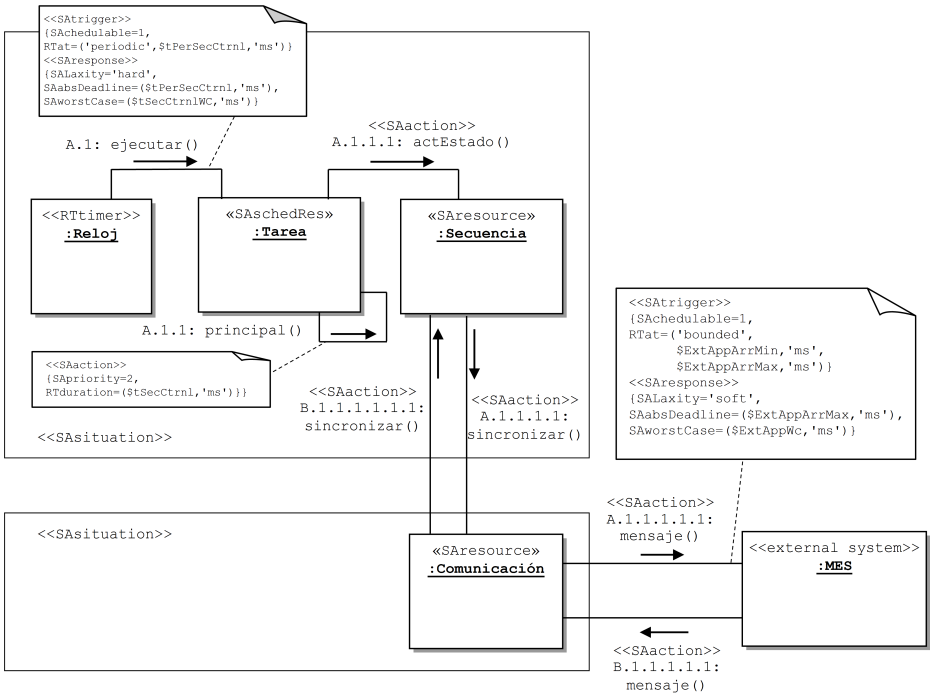


Figura 2.19: Modelo dinámico del caso de comunicación con una aplicación externa

2.4. Propuesta de modelo de FB IEC 61499 adaptado

Una vez modelado el control distribuido de la máquina herramienta genérica en un sistema de fabricación ágil, a continuación se propone un modelo de FB IEC 61499 adaptado a dicho dominio. El objetivo principal, como se indicó al principio de este capítulo, es facilitar el diseño con dicho estándar del control de este tipo de complejas máquinas que pueden tener: a) docenas de ejes y secuencias automáticas, que pueden ser reconfigurables; b) un funcionamiento coordinado (o "coreografiado") con otras máquinas o elementos en la celda/línea de fabricación y c) una integración con los sistemas de información de la factoría. Un segundo objetivo del modelo adaptado propuesto es permitir una implementación eficiente y escalable de este tipo de aplicaciones de control, cuestión que será abordada en el siguiente capítulo de este trabajo.

Antes de empezar la discusión sobre la propuesta de modelo adaptado indicar, como cuestión previa, un cambio de terminología respecto a los términos de dispositivo y recurso establecidos por el estándar (ver 1.3.2). El motivo es la posible confusión a lo largo del trabajo con sus significados más comúnmente empleados. Así, por claridad, se adopta el término **controlador** como sinónimo de dispositivo y el término ***runtime*** como sinónimo de recurso.

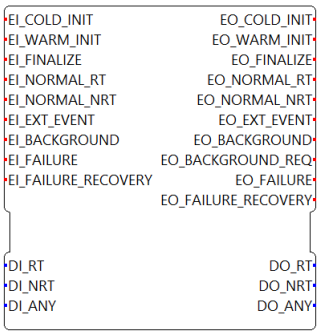


Figura 2.20: Modelo de FB adaptado

2.4.1. Modelo de FB IEC 61499 adaptado

Según el estándar un FB está constituido por: eventos de entrada y salida, datos de entrada y salida, un control de ejecución (*Execution Control* o EC), algoritmos y variables. El **primer principio** para la propuesta de modelo adaptado (ver Fig. 2.20) es la predefinición de los algoritmos y los eventos de entrada y salida, a partir del control de la máquina herramienta genérica modelado en la sección anterior. En cambio, el modelo adaptado no predefine los datos de entrada y salida, ni las acciones a realizar en cada uno de sus algoritmos, que serán definidos dependiendo de la funcionalidad concreta de cada tipo de FB. Por su parte, el comportamiento del modelo de FB adaptado está también predefinido y es representado por el

ECC (*Execution Control Chart*) mostrado en la Fig. 2.21 (en forma de diagrama de estados UML por facilidad de representación). Sus estados y subestados son los siguientes:

- ***Non Operational***: Representa el estado del FB antes de pasar a subestado operacional. Las acciones contenidas en sus subestados corresponden con su inicialización y finalización, y deben ejecutarse lo antes posible.
  - ***Initialization***: Establece el estado inicial del FB, representado por sus variables internas. El tipo de inicialización es determinado por la plataforma de ejecución y puede ser (ver subsección 2.3.4):
    - ***Cold Init***: Inicializa el FB al estado inicial establecido en tiempo de diseño.
    - ***Warm Init***: Restaura el estado que tenía el FB justo antes de la última parada de la aplicación.
  - ***Finalize***: Estado final del FB antes de ser finalizado (y eliminado). Puede contener acciones como la liberación de recursos asignados por la plataforma de ejecución, la desactivación de actuadores, etc.
- ***Operational***: Representa el estado del FB en modo operativo donde atiende los eventos internos y externos que determinan las acciones a ejecutar.
  - ***Internal Event***: Estado al que pasa el FB una vez inicializado, contiene las acciones que atienden los eventos internos. Su ejecución puede requerir el cumplimiento de distintos *deadlines*, por lo que son agrupadas en subestados con diferentes restricciones de tiempo real, representadas en el diagrama mediante transiciones producidas por los eventos predefinidos. Dichos subestados son:
    - ***Idle***: Estado donde el FB espera la ocurrencia de un evento.
    - **RT**: Estado que contiene las acciones con restricciones *hard real-time* ejecutadas en el control de proceso, con los siguientes subestados:
      - ***Normal***: Acciones periódicas de funcionamiento normal como son: secuencias automáticas (ver Subsección 2.3.2) o control de ejes (ver Subsección 2.3.3).
      - ***Failure***: Acciones periódicas de funcionamiento en modo fallo (ver subsección 2.3.5).
      - ***Recovery***: Acciones de recuperación a realizar después de un fallo, y justo antes de volver a ***Normal*** (ver Subsección 2.3.5).
    - **NRT**: Contiene las acciones periódicas con restricciones *soft real-time* (o *near real-time*). Por ejemplo, las secuencias de sincronización o comando de proceso (ver Subsección 2.3.2).
    - ***Background***: Acciones a ejecutar cuando sea posible, es decir, sin restricciones de tiempo real. Por ejemplo, cálculos de trayectorias en el control de ejes (ver Subsección 2.3.3).

- ***External Event***: Acciones para la atención a eventos externos, normalmente deben ejecutarse lo antes posible. Por ejemplo, interrupciones hardware desde los dispositivos de interfaz de proceso (ver Subsección 2.3.1).

El **segundo principio** seguido para la propuesta de modelo adaptado es asociar con cada algoritmo del FB un evento de entrada, que provoca su ejecución; y un evento de salida, generado al final de esta ejecución. Con este principio, una FBN compuesta por FB basados en el modelo adaptado se construye conectando los eventos de entrada y salida asociados al mismo tipo de algoritmo (ver Fig. 2.22). De esta manera, se establecen cadenas de eventos en una topología ***daisy-chain***, que provoca la ejecución secuencial de los algoritmos de un mismo tipo en los distintos FB. Esta ejecución secuencial hace que las FBN puedan ser distribuidas de manera sencilla incorporando SIFB de tipo *Publish/Subscribe* en las conexiones inter-controlador.

En este sentido, se establece una correspondencia entre las cadenas de eventos predefinidas en el modelo adaptado y los tipos de conexión previamente identificados en el control distribuido de la máquina herramienta genérica (ver Subsección 2.2.4), representada en la Tabla 2.3.

Cadena de eventos	**Tipo de conexión**	**Datos**	**Inter-controlador**
`COLD_INIT`	Control de aplicación	No	Sí
`WARM_INIT`	Control de aplicación	No	Sí
`FINALIZE`	Control de aplicación	No	Sí
`RECOVERY`	Control de aplicación	No	Sí
`FAILURE`	Control de aplicación	Sí	Sí
`NORMAL_RT`	Control de proceso	Sí	No
`EXT_EVENT`	Interfaz de proceso	Sí	No
`NORMAL_NRT`	Comando y Sincronización de proceso, Interfaz de usuario, Datos de aplicación y producción	Sí	Sí
`BACKGROUND`	Control de proceso, Datos de aplicación y producción	Sí	Sí

Tabla 2.3: Correspondencia entre cadenas de eventos y tipos de conexión

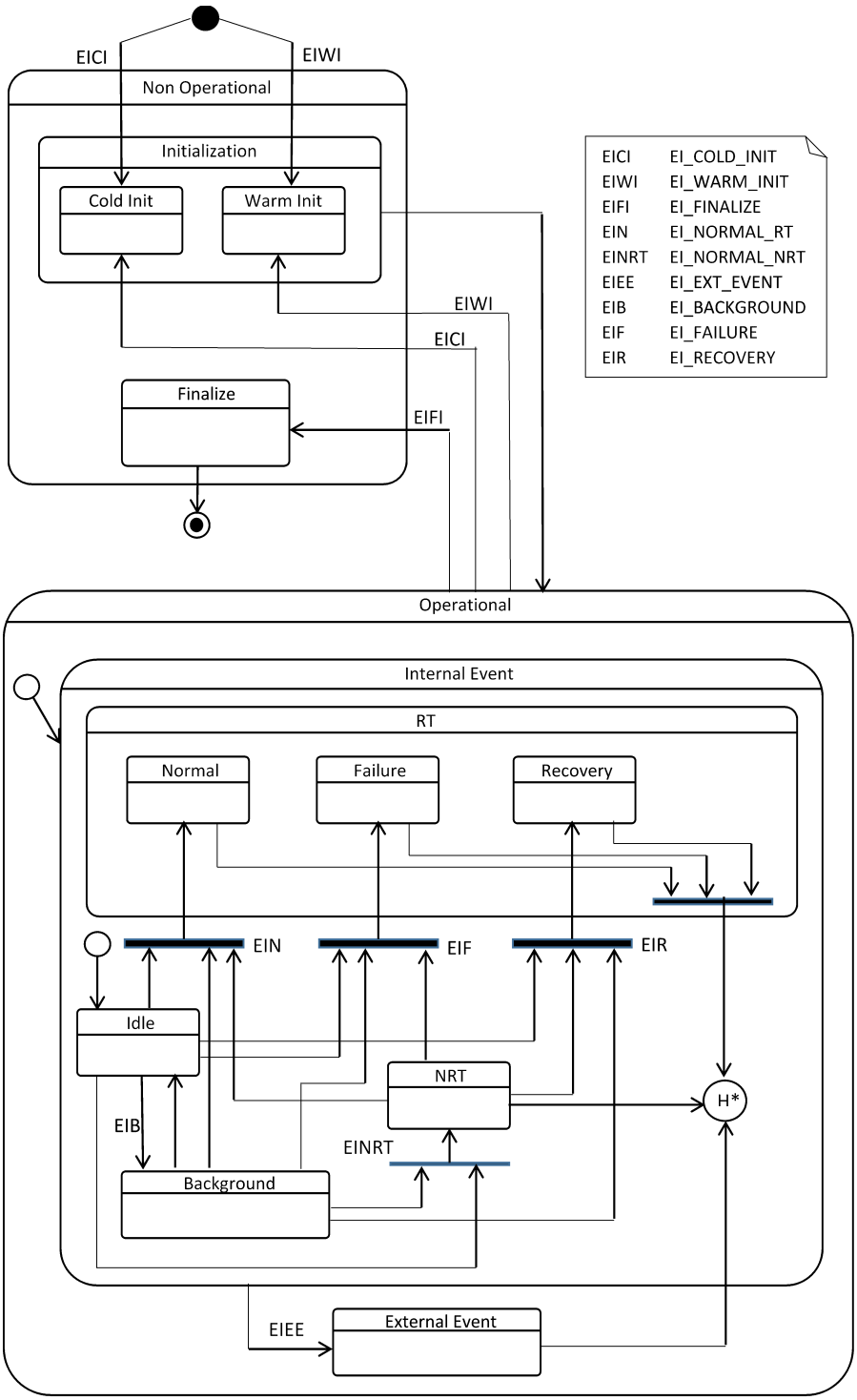


Figura 2.21: ECC del modelo de FB adaptado (como diagrama de estados UML)

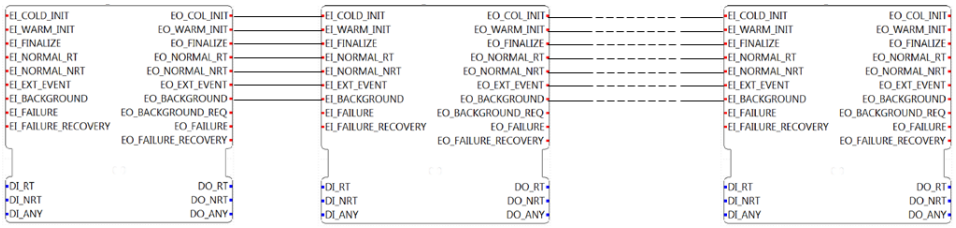


Figura 2.22: FBN con conexiones de eventos en *daisy-chain*

2.4.2. Metodología para el diseño de FBN con el modelo de FB adaptado

El estándar IEC 61499 únicamente da unas breves indicaciones de los pasos a seguir para el diseño de FBN distribuidas (ver Subsección 1.4.3), que básicamente consisten en: a) diseñar la FBN sin considerar los aspectos relacionados con la distribución, b) dividir la FBN en subFBN que son asignadas a cada uno de los controladores (recursos) y c) establecer el modelo de comunicación en las conexiones inter-controlador incorporando SIFB de tipo *Publish/Subscribe* o *Client/Server*.

Estas breves indicaciones puede ser particularizadas para el modelo de FB adaptado en una sencilla metodología que consta de los siguientes pasos:

1. Identificar los tipos de subFBN que componen la red distribuida de FB. Esta identificación puede partir de patrones de diseño clasificados por las funcionalidades: a) interfaz de proceso, b) control y automatización, c) gestión de fallos y d) interfaz con aplicaciones externas. Al respecto, el autor ha contribuido en la presentación de diferentes propuestas de patrones de diseño, ver [122], [123] y [124], así como en la elaboración de guías de diseño [125].

2. Identificar en cada subred los tipos de FB, definiendo para cada uno de ellos: a) los datos de entrada y salida (los eventos de entrada y salida ya están predefinidos), b) las variables internas y c) las acciones en los algoritmos predefinidos. Un objetivo de diseño es que estos tipos de FB sean reusables, salvo quizá los tipos que implementen los niveles más altos de la lógica de control (o de "negocio") de la aplicación.

3. Distribuir las subFBN entre los distintos controladores.

4. Establecer los *daisy chains* indicados en la Sección 2.4 considerando las dependencias entre datos. Es decir, en cada *daisy chain* los FB productores de datos deben estar antes que los FB consumidores de los mismos.

5. Incorporar SIFB de comunicación *Publish/Subscribe* en las conexiones inter-controlador de los *daisy chain* teniendo en cuenta sus datos asociados.

6. Incorporar SIFB de comunicación también del tipo *Publish/Subscribe* en las conexiones con las aplicaciones externas.

2.4.3. Casos de uso

A modo de ejemplos de diseño se describen los casos de uso más significativos para el control de las máquinas herramienta consideradas, ya caracterizados previamente (ver Sección 2.3). En cada caso, las subFBN se representan de forma simplificada, por lo que únicamente se muestran las conexiones de datos y eventos más relevantes.

2.4.3.1. Inicialización y finalización

Para inicializar la aplicación (ver Subsección 2.3.4) se establecen dos *daisy chains* a partir de los eventos `EI_COLD_INIT` y `EO_COLD_INIT`, `EI_WARM_INIT` y `EO_WARM_INIT`. Y para la finalización un *daisy chain* con los eventos `EI_FINALIZE` y `EO_FINALIZE`. Un ejemplo de esta subred es mostrado en la Fig. 2.23. El evento origen en los tres casos es generado por la plataforma de ejecución, situación representada mediante el FB estándar `E_RESTART`. No hay conexiones de datos entre los bloques función.

En este caso de uso se considera que todos los bloques función son inicializados y finalizados a la vez, justo antes de empezar ejecutar los algoritmos de control periódicos. En el caso de aplicaciones reconfigurables los bloques función pueden ser creados o finalizados en distintos momentos, por lo que deben establecerse diferentes *daisy chain* de inicialización y finalización de las subFBN reconfigurables.

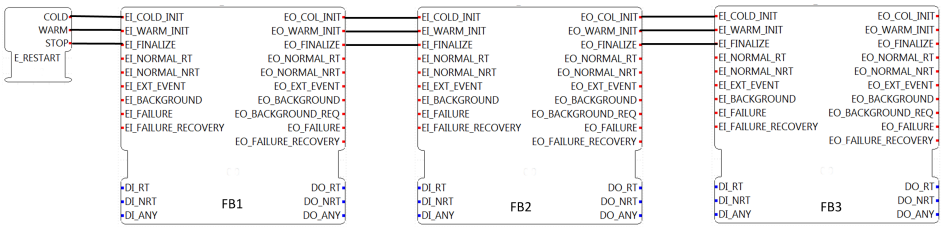


Figura 2.23: SubFBN de inicialización y finalización

2.4.3.2. Interfaz de proceso

El caso de uso para la interfaz de proceso (ver Subsección 2.3.1) corresponde con la subred mostrada en la Fig. 2.24. Consta de tres FB y un SIFB, denominado `FB_IO`. Este último representa la comunicación con el controlador externo del bus de campo. Dicho SIFB genera un evento cada vez que el bus de campo demanda ser atendido. Este evento es la fuente de un *daisy chain* del tipo `EXT_EVENT` compuesto por: `CTRL_FIELDBUS`, encargado de las comunicación con el bus de campo; `IO_DEVICE`, que modela un supuesto dispositivo de entrada/salida y finalmente `IO_ACOND` que realiza el acondicionamiento de la entrada/salida del dispositivo. Estos FB también tienen conexiones entre sus datos mostradas en la misma figura.

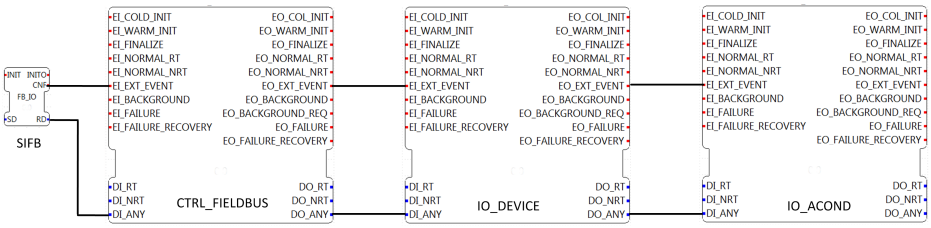


Figura 2.24: SubFBN de interfaz de proceso

2.4.3.3. Control de un eje

Este caso, más complejo que los dos anteriores, es mostrado en la Fig. 2.25. En el control de un eje (ver Subsección 2.3.3) existen tres *daisy chains*:

1. `NORMAL_NRT`: Empieza en el FB `CONTROL`, que realiza las funciones de secuencia de control del eje, ésta es ejecutada cada vez que recibe un evento generado periódicamente por la plataforma de ejecución mediante el SIFB estándar `E_CYCLE`. Esta cadena finaliza en la función `NORMAL_NRT` del FB `PID`, desde donde se dan comandos de proceso al algoritmo de control.

2. `BACKGROUND`: Iniciada por el evento `EO_BACKGROUND_REQ` del FB `CONTROL`, con el fin de solicitar la realización de cálculos en segundo plano en el FB `TRAJ_CAL` relacionados con trayectorias en los movimientos del eje. Mediante el evento de salida `EO_BACKGROUND` el FB `CONTROL` recibe la notificación de que dichos cálculos han finalizado.

3. `NORMAL_RT`: Iniciada por un evento periódico generado por el SIFB estándar `RT_E_CYCLE`. En esta cadena se realiza la adquisición de la imagen de proceso, almacenada en el FB `IN_ACOND`; después se ejecuta el algoritmo de control mediante el FB `PID` y finalmente se actualiza la imagen de proceso con el FB `OUT_ACOND`.

Estas tres cadenas tienen diferente prioridad de ejecución. La más prioritaria es `NORMAL_RT` que tiene restricciones *hard real-time*, por lo que su ejecución puede suspender la ejecución del resto de cadenas. Mientras que, la menos prioritaria es `BACKGROUND`.

2.4.3.4. Gestión de fallos mediante zonas

La subred mostrada en la Fig. 2.26 representa la gestión de fallos (ver Subsección 2.3.5) en una zona compuesta por los bloques función `FB1` y `FB2`. Dichos FB comunican una situación de fallo a su zona, representada por `ZONE_1`, a través de eventos de tipo `EO_FAILURE`. Dichos eventos están conectados mediante el FB estándar `E_MERGE`, que permite tener dos eventos fuente en un mismo evento de entrada. Una vez recibido uno de dichos eventos la zona lo comunica, mediante el FB estándar `E_SPLIT`, que convierte una evento de entrada en varios eventos de salida, a todos sus FB y a su zona de nivel superior en la jerarquía de gestión de fallos, representada por `ZONE_2`. Esta zona, dependiendo de la política de

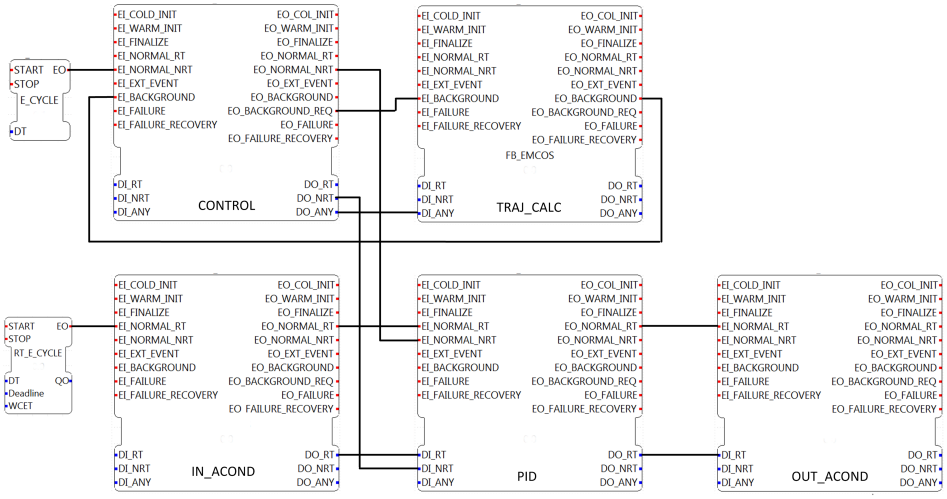


Figura 2.25: SubFBN de control de un eje

propagación, lo comunica a sus zonas en niveles inferiores (situación no mostrada en la figura). Una vez que la situación de fallo ha desaparecido, y tras una orden de rearme (no mostrada), la zona lo indica a sus FB y su zona de nivel superior mediante un evento de tipo `EO_FAILURE_RECOVERY`.

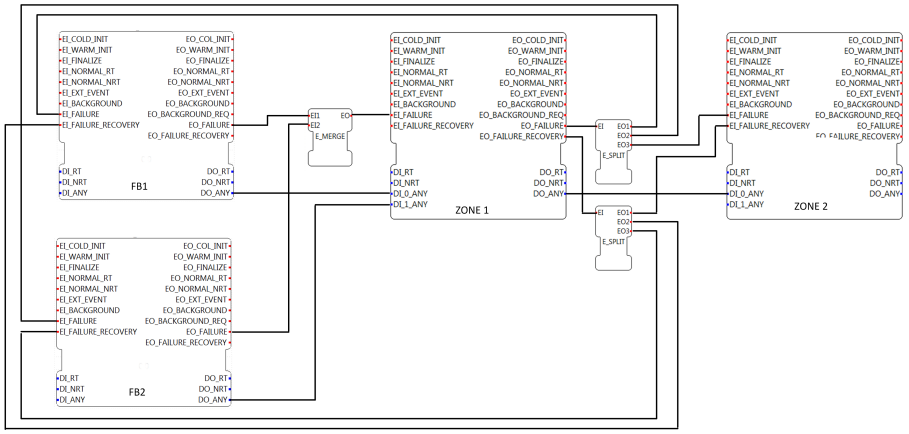


Figura 2.26: SubFBN de zona de fallos

2.4.3.5. Comunicaciones inter-controlador y con aplicaciones externas

El último caso de uso es una FBN distribuida entre varios controladores (ver Subsección 2.3.6). En la Fig. 2.27 se muestra la subred para la comunicación entre dos controladores intermedios de un *daisy-chain*. Como se ha indicado en la

metodología de diseño (ver Subsección 2.4.2) en las conexiones inter-controlador de los *daisy chain* deben insertarse SIFB del tipo *Publish/Subscribe*. En el caso mostrado esta cadena es del tipo `NORMAL_NRT`. Por ejemplo, una sincronización de proceso entre dos secuencias de control.

Un caso particular de comunicación inter-controlador puede producirse si el diseño de la FBN requiere que un FB utilice datos producidos por bloques función ejecutados posteriormente (o "aguas abajo"). La solución es emplear SIFB sin conexión con el evento de entrada como se muestra en la misma figura. En este caso, el dato consumido por el FB habrá sido producido el ciclo anterior.

En la misma subred se muestra también la comunicación con dos aplicaciones externas, una interfaz de usuario y una gestión de producción. Esta comunicación se realiza mediante dos SIFB de comunicación denominados `HMI` y `MES` incluidos en el *daisy chain* `NORMAL_NRT`.

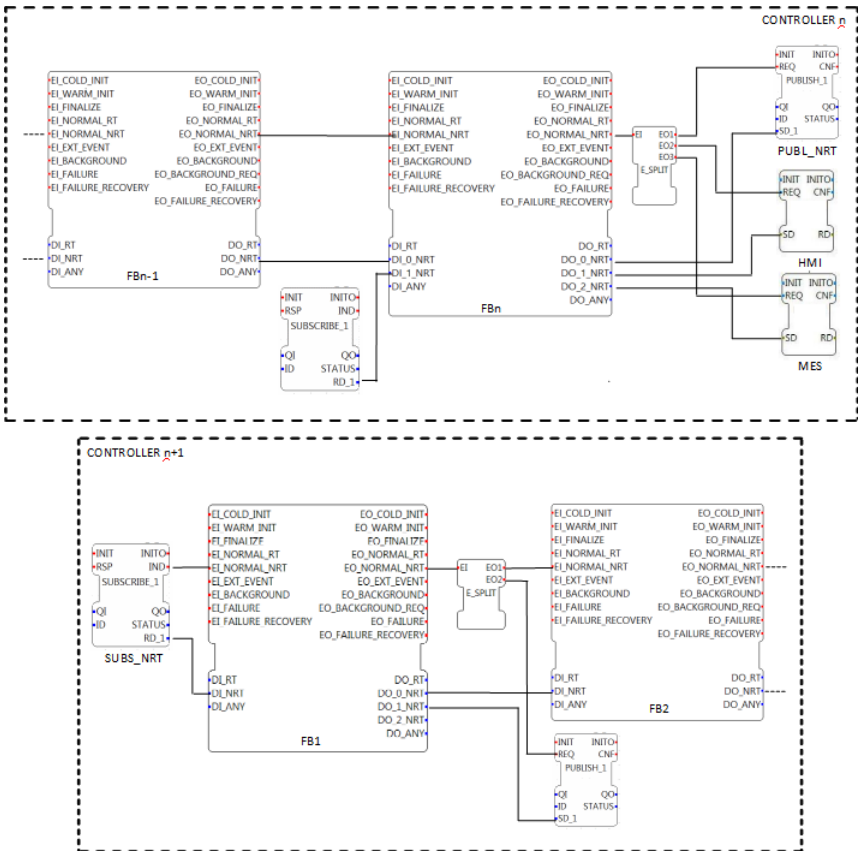


Figura 2.27: Subred de comunicación inter-controlador y con dos aplicaciones externas

2.5. Conclusiones

El paso del estándar IEC 61131 a IEC 61499 puede ser costoso para muchos ingenieros de control. En particular, el nuevo estándar exige conocimientos de sistemas distribuidos, paradigmas como la programación orientada a eventos y la programación orientada a componentes, así como de metodologías y lenguajes típicos de la ingeniería de software (p.e. UML). Esta situación se ve agravada si los sistemas a controlar son complejos. Para facilitar el paso al nuevo estándar en este trabajo se realiza una propuesta original consistente en el empleo de modelos de FB adaptados al dominio de aplicación. Estos modelos tienen una semántica clara y unas reglas sencillas para la composición de las FBN distribuidas, las cuales constituyen las aplicaciones de control en IEC 61499.

Para definir el modelo de FB adaptado a este dominio en primer lugar se han caracterizado estos sistemas, desde el punto de vista de su control, considerando una máquina herramienta genérica. En particular, se han identificado sus funcionalidades, así como discutido aspectos relacionados con su control distribuido como la arquitectura, granularidad y los tipos de conexión, que han sido caracterizados para este dominio. En segundo lugar, se ha presentado una propuesta de control distribuido de la máquina herramienta genérica considerada. Este control ha sido modelado a partir de casos de uso, en los que se han identificado los bloques funcionales del control y establecido sus relaciones estructurales y su comportamiento dinámico.

A continuación, se ha presentado la propuesta de modelo FB adaptado. La base principal para su establecimiento es la predefinición, a partir del dominio de aplicación modelado, de los algoritmos, los eventos de entrada y salida y el ECC del FB. Con este modelo las FBN se construyen conectando los eventos de los FB en cadena en una topología *daisy-chain*. Estas cadenas de eventos corresponden con los tipos de conexión previamente caracterizados. Finalmente, se ha presentado una sencilla metodología de diseño de FBN, así como algunos casos de uso significativos.

El modelo de FB adaptado propuesto facilita el diseño de aplicaciones en IEC 61499. En primer lugar, porque las FBN se diseñan únicamente con tipos de FB basados en este modelo. En cada uno de los cuales el ingeniero de control debe determinar las acciones a realizar en las situaciones habituales, ya predefinidas, en el dominio. En segundo lugar, por la facilidad de diseño de las FBN, mediante la conexión en cadena de los eventos predefinidos del mismo tipo. Y en tercer lugar, por el modelo de ejecución secuencial establecido con estas cadenas de eventos, similar al del estándar IEC 61131, y por lo tanto conocido por estos ingenieros.

De manera más general, el uso de modelos adaptados en IEC 61499 tiene como ventajas la mejora de los procesos y tiempos de diseño, al encapsular los FB y sus patrones derivados las funcionalidades del dominio considerado. Por ejemplo, esta encapsulación facilita la adaptación a los cambios tecnológicos disminuyendo costes en procesos de re-ingeniería.

Capítulo 3

Modelo de ejecución de redes de bloques función IEC 61499 adaptado

3.1. Introducción

En el capítulo anterior se ha presentado una propuesta para el control distribuido de una máquina herramienta genérica en sistemas de fabricación ágil. Las principales características de esta propuesta son:

- Control distribuido *peer-to-peer* con un controlador por cada uno de los elementos principales de la máquina herramienta: cargador, mecanizado y descargador.
- Conexiones entre las diferentes partes del control distribuido basadas en los tipos de conexión identificados para el dominio de aplicación:
  - Control de aplicación.
  - Control e interfaz de proceso.
  - Comando y sincronización de proceso.
  - Interfaz de usuario, datos de aplicación y producción.
- Restricciones de tiempo real de las diferentes tareas de control a ejecutar:
  - *Hard real-time*. Secuencias automáticas y movimientos de ejes (control e interfaz de proceso).
  - *Soft real-time*. Modos de operación, lógica de control intra e inter-controlador (control de aplicación, comando y sincronización de proceso).
  - *Sin restricciones de tiempo real*. Interfaz de usuario, gestión de datos de aplicación y producción.

Esta propuesta de control ha permitido modelar un control distribuido de la máquina herramienta genérica identificando los bloques funcionales, así como sus relaciones estructurales y dinámicas. Con esta base se ha propuesto a continuación un modelo de FB IEC 61499 adaptado a dicho dominio (ver Sección 2.4), a partir del cual es posible construir las FBN adaptadas. El siguiente paso es determinar la manera de ejecutar estas FBN, cuestión abordada en el presente capítulo.

Como hemos visto, el modelo de FB adaptado predefine: algoritmos, eventos y ECC del FB; estableciendo además una correspondencia entre los eventos de entrada y salida. De manera general, con este modelo una FBN se construye conectando eventos de entrada y salida homólogos. Mediante estas conexiones se establecen cadenas de eventos (en adelante *daisy-chains*) que determinan un esquema de composición de las FBN (ver Fig. 2.22). Este esquema constituye la base del modelo de ejecución adaptado, que de forma más precisa podemos especificar como sigue:

1. Una FBN está compuesta por FBs basados en el modelo adaptado con sus eventos conectados en *daisy-chain*.
2. Un *daisy-chain* se forma conectando un evento de entrada con su evento de salida homólogo en el FB que contiene el siguiente algoritmo a ejecutar en la cadena.
3. El orden de ejecución se determina en tiempo de diseño en base a las dependencias productor/consumidor en los datos de entrada y salida.
4. El número de *daisy-chain*, no así su longitud, está determinado por los eventos predefinidos en el modelo de FB adaptado.
5. Para cumplir con el comportamiento del modelo de FB adaptado determinado en su ECC, en particular, las diferentes restricciones de tiempo real, los eventos de entrada tienen el siguiente orden de prioridad:
   a) `EI_COLD_INIT`, `EI_WARM_INIT` y `EI_FINALIZE` (más prioritarios)
   b) `EI_EXT_EVENT`
   c) `EI_NORMAL_RT`, `EI_FAILURE` y `EI_FAILURE_RECOVERY`
   d) `EI_NORMAL_NRT`
   e) `EI_BACKGROUND` (menos prioritario)
6. Las conexiones inter-controlador en los *daisy-chains* son efectuadas mediante SIFB de tipo *Publish/Subscribe*.

A la hora de definir el modelo de ejecución adaptado debemos establecer los requisitos que éste debe cumplir. Para ello, es necesario tener en cuenta las características de la propuesta de control distribuido de la máquina herramienta genérica, así como el modelo de FB adaptado. Además, es necesario también tener en cuenta dos características más, relevantes en este tipo de máquinas:

- Su operación en entornos donde hay personas y bienes materiales, por lo que deben tener un funcionamiento totalmente seguro.

- El hecho de que pueden llegar a tener un número elevado de secuencias automáticas y posicionamientos precisos, que además pueden ser reconfigurados según las necesidades de producción.

A partir de todo lo anterior, los **requisitos** que debe cumplir el **modelo de ejecución adaptado** al control de máquinas herramienta en sistemas de fabricación ágil son los siguientes:

1. Garantizar un comportamiento determinista de las FBN, construidas mediante el modelo de FB adaptado, de manera que la máquina herramienta tenga un comportamiento seguro.

2. Permitir el cumplimiento de las distintas restricciones de tiempo real establecidas en la propuesta de control de la máquina herramienta genérica.

3. Tener una implementación eficiente y escalable que permita adaptarse a incrementos en el número de secuencias y posicionamientos precisos a controlar.

4. Facilitar el diseño de las FBN a partir del modelo de FB adaptado.

En el resto del capítulo se discuten, en primer lugar, los aspectos relacionados con la ejecución de FBN en IEC 61499. Posteriormente, se analizan los modelos de ejecución y *runtimes* propuestos hasta la fecha. Y finalmente, se presenta la propuesta de modelo de ejecución adaptado.

3.2. Ejecución de redes de bloques función IEC 61499

En el primer capítulo de este trabajo se ha introducido el estándar IEC 61499 (ver 1.3.3) y sus elementos principales: bloques función, dispositivos, recursos. Así como, se ha discutido su uso para el desarrollo de aplicaciones de control desde el punto de vista de los principios de diseño aceptados para los sistemas distribuidos [82]. También se han referenciado las principales propuestas presentadas para su implementación e indicado el problema más importante que tiene este estándar: las ambigüedades semánticas presentes en su primera edición. Como veremos seguidamente, estas ambigüedades pueden ocasionar FBN no portables o en el peor de los casos con comportamientos no deterministas. Es decir, una misma FBN, en iguales condiciones de partida, puede comportarse de forma diferente en recursos (*runtimes* o plataformas de ejecución) distintos o incluso en el mismo recurso.

El FB es el elemento a partir del cual se diseñan las aplicaciones de control con IEC 61499. Este elemento resulta familiar para los ingenieros de control con experiencia en PLCs, ya que extiende los bloques funcionales definidos en el lenguaje FBD (*Function Block Diagram*) del anterior estándar IEC 61131. Como ya se ha indicado, la principal diferencia es que en IEC 61499 los FB son ejecutados siguiendo los principios de la programación orientada a eventos. Se considera que

este es el mecanismo clave para el desarrollo de las aplicaciones de control distribuido [2]. Este tipo de aplicaciones son más difíciles de llevar a cabo con IEC 61131, al no estar diseñado para ello (ver Sección 1.3.1).

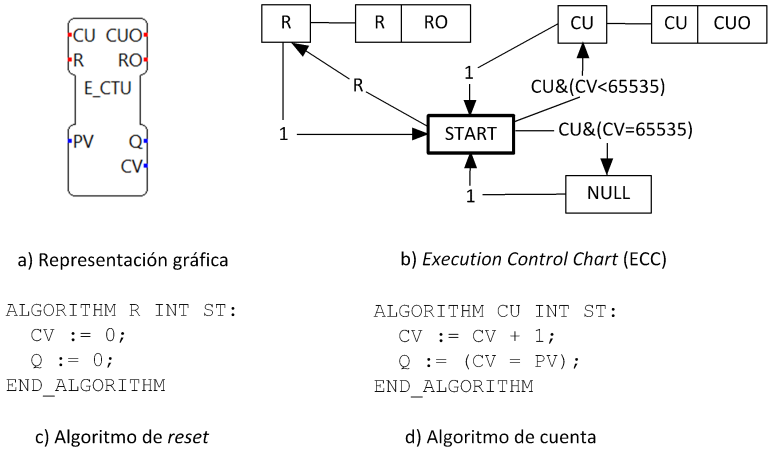


Figura 3.1: Ejemplo de FB IEC 61449: Contador ascendente [1]

En la Figura 3.1 se muestra un ejemplo de FB que define un contador ascendente. La interfaz del FB con sus eventos y datos de entrada/salida es mostrada en (a), mientras que el comportamiento del FB está determinado por el EC (*Execution Control*) mediante el ECC (*Execution Control Chart*) mostrado en (b). Dependiendo de qué evento de entrada ocurra se produce el incremento o la puesta a cero del contador, así como la generación del correspondiente evento de salida para indicar la acción realizada. Los algoritmos con estas dos acciones son mostrados en (c) y (d), escritos en lenguaje estándar ST (*Structured Text*) [44]. El recurso es el elemento de IEC 61499 encargado de planificar la activación de los FB con el objeto de que sean ejecutados los algoritmos de respuesta a los eventos de entrada.

3.2.1. Problemas derivados de la interpretación de IEC 61499

Un aspecto importante antes de determinar la implementación de IEC 61499 es estudiar los problemas derivados de la interpretación de su modelo de ejecución por eventos. Los trabajos [48] [81] [126] [72], y más recientemente [58] [46], han discutido estos problemas, a los que una segunda versión de estándar ha intentado dar respuesta. De manera breve, estos problemas son los siguientes:

- **Ejecución de FB en caso de la ocurrencia simultánea de dos o más eventos**: Una primera cuestión a discutir es qué ocurre cuando un FB es activado y tiene más de un evento de entrada que atender. En el ejemplo de la Fig. 3.1, ¿el contador debe ser incrementado o puesto a cero?. En este caso puede servirnos lo que dice la primera edición del estándar en su Apartado 5.2.2.2, Tabla 1 (c) en lo relativo a la evaluación de las transiciones en el ECC:

"This operation consists of evaluating the conditions at all the EC transitions following the active EC state and clearing the first EC transition (if any) for which a TRUE condition is found. "Clearing the EC transition" consists of deactivating its predecessor EC state and activating its successor EC state. The order in which the transition conditions are evaluated corresponds to the order in which they are declared following the textual syntax defined in B.2.1, or equivalently in the XML syntax defined in IEC 61499-2."

Es decir, si el ECC del FB está en `START` el contador será puesto a cero si la transición `R` está declarada antes que `CU&(CV<65535)` o incrementado en caso contrario. Algo que por otra parte, no queda reflejado en la FBN, haciendo que ésta no represente de manera explícita una decisión de diseño relevante. Este tipo de definiciones textuales son la fuente de las ambigüedades semánticas reseñadas anteriormente. Para resolver esta cuestión la nueva versión hace dos modificaciones: a) únicamente un evento puede ser entregado en cualquier instante dado por el recurso al FB en su activación, y b) puede indicarse explícitamente en la FBN el orden de evaluación de las transiciones.

El problema con la primera modificación es qué ocurre si se producen varios eventos simultáneamente. Una solución es elegir uno de ellos (p.e. por importancia) y descartar el resto. Esto tiene como inconvenientes que alguna situación relevante no sea contemplada, la pérdida de datos o incluso un comportamiento no determinista de la FBN [2]. Otra solución es guardar los eventos en un cola para ir procesándolos uno tras otro más tarde (solución discutida más adelante).

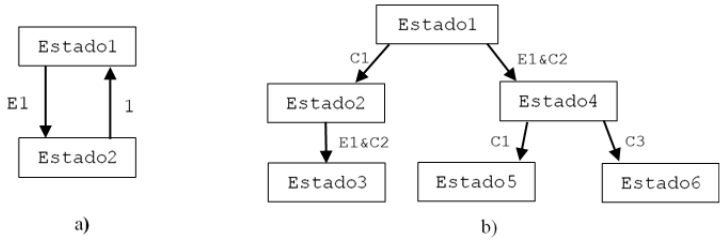


Figura 3.2: Ambigüedad por la duración de los eventos

- **Vigencia de los eventos**: En la primera versión del estándar no está definida de manera explícita la vigencia de los eventos al evaluar las transiciones en un ECC. Consideremos los fragmentos de ECC mostrados en la Fig 3.2, donde `Ei` representa un evento de entrada, `Ci` una condición de guarda booleana y `1` una condición siempre verdadera; también asumiremos que las transiciones han sido declaradas en orden de izquierda a derecha. En el caso a), si `E1` no es consumido al realizar el paso del `Estado1` al `Estado2` el ECC entrará en un bucle infinito. Consideremos ahora el caso b), suponiendo que el ECC está en `Estado1` y la transición `C1` está declarada antes que `E1&C2`. Si `C1` y `C2` son verdaderas cuando `E1` ocurre, el ECC pasará al `Estado2`; pero

el paso a `Estado3` no está claro, ya que no está definido en el estándar si `E1`, que ha provocado la activación del FB, debe ser consumido en un cambio de estado aunque no forme parte de la transición que ha producido dicho cambio.

Este problema ha sido resuelto en la segunda versión al indicar que un evento de entrada es válido únicamente la primera vez que se produce la evaluación de las transiciones. Si se produce un cambio de estado solamente las condiciones posteriores sin evento serán evaluadas, permitiendo, en su caso, nuevos cambios de estado. Además, en la primera evaluación puede ser tomada una condición aunque no tenga evento. Para el ejemplo considerado, en el caso a) no se producirá el bucle infinito, y en el caso b) se pasará al `Estado2` pero no al `Estado3`, al haber sido ya consumido el evento en la primera transición.

- **Evaluación de las transiciones que no incluyen eventos**: Consideremos ahora de nuevo el caso de la Fig 3.2 b), donde el ECC está en `Estado1`, `C2` es verdadera y `C1` es falsa en el momento que ocurre `E1`. Entonces, se produce un cambio a `Estado4`, pudiéndose considerar a continuación dos interpretaciones distintas según la primera versión del estándar:

  1. Las transiciones sin eventos son evaluadas una sola vez finalizada la ejecución del estado. El inconveniente con esta interpretación es que puede ocurrir una situación donde el ECC quede bloqueado. En el caso indicado, esto ocurre si `C1` y `C3` son falsas en el momento de finalizar la ejecución de `Estado4`.
  2. Las transiciones sin eventos son evaluadas cada vez que se produce un evento de entrada. En el caso indicado se saldrá del `Estado4` si `C1` y/o `C3` son verdaderas al producirse cualquier evento de entrada, evitándose el bloqueo.

  Esta segunda interpretación es la considerada en la nueva versión del estándar, que permite evaluar transiciones sin eventos cuando el FB es activado. El problema es que el consumo de eventos en transiciones de las que no forman parte no está representado explícitamente en la FBN, lo que dificulta su legibilidad.

3.2.2. Problemas derivados del orden de ejecución de los FB

El orden de ejecución de los FB es sin duda el problema más importante relacionado con la interpretación del estándar. La primera versión de éste no indica de manera clara cómo deben ser activados los FB en una FBN. Es claro que este orden debe tener en cuenta las dependencias entre FB productores de eventos y FB consumidores (el *event flow* según el estándar). Pero una FBN puede tener también conexiones en paralelo o en *feedback*. Igualmente tampoco define de manera precisa cuándo deben ser generados los eventos de salida, se han identificado tres posibilidades [48]: a) al finalizar la acción asociada; b) al finalizar todas las acciones del estado y c) al finalizar la ejecución del FB.

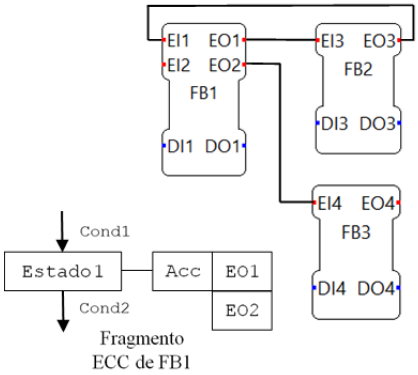


Figura 3.3: Ambigüedad por el orden de ejecución de FB: Caso 1

Consideremos el caso de FBN mostrado en la Fig. 3.3, asumiendo que los FB se activan en orden de izquierda a derecha y de arriba a abajo, pueden ejecutarse concurrentemente, y que los eventos de salida son generados al finalizar su acción asociada. En `Estado1` se generan los eventos de salida `EO1` y `EO2` que deben activar FB2 y luego FB3. Pero FB2 a su vez genera un nuevo evento de salida `EO3`, que vuelve a activar FB1. Si éste no ha finalizado la primera ejecución se produce una **condición de carrera**. Esta condición puede ser evitada si asumimos que los eventos se generan al finalizar la ejecución del FB. El problema ahora es que FB3 entra en situación de **inanición**, ya que nunca es ejecutado debido al orden de activación asumido.

En este caso la nueva versión del estándar indica que los eventos de salida son generados tan pronto es realizada su acción asociada. Una manera de evitar entonces la condición de carrera es no permitir la ejecución concurrente de los FB, algo que puede interpretarse en la nueva versión, al indicar que estos elementos son **atómicos**. Además, con esta modificación una planificación adecuada de la ejecución de los FB por parte del recurso puede evitar situaciones de inanición como la de FB3. Un inconveniente de esta restricción es que no aprovecha la mejora de prestaciones obtenida en los actuales procesadores *multi-core* [81].

Otro ejemplo de problema relacionado con el orden de ejecución puede encontrarse en el caso mostrado en la Fig. 3.4 a). En este fragmento de FBN tenemos a FB4 que usa datos de FB2 y FB3, mientras que su activación depende del evento de salida de este último. Con la interpretación de la primera versión de IEC 61499, mencionada en el caso anterior, no está claro qué datos utilizará FB4, ya que FB2 y FB3 pueden acabar en distinto momento. La ejecución no concurrente de los FB y la generación de uno en uno de los eventos de salida, como indica la segunda versión, hace que esta situación no determinista sea evitada. Aunque, ésto es así únicamente si todos los FB están en el mismo dispositivo. En caso de no serlo, Fig. 3.4 b), el comportamiento de nuevo será no determinista, ya que recursos alojados en dispositivos diferentes pueden planificar la ejecución de sus FB de manera distinta. El problema es que el estándar indica que el comportamiento de una FBN no debe verse afectado (salvo por retardos o fiabilidad de la comuni-

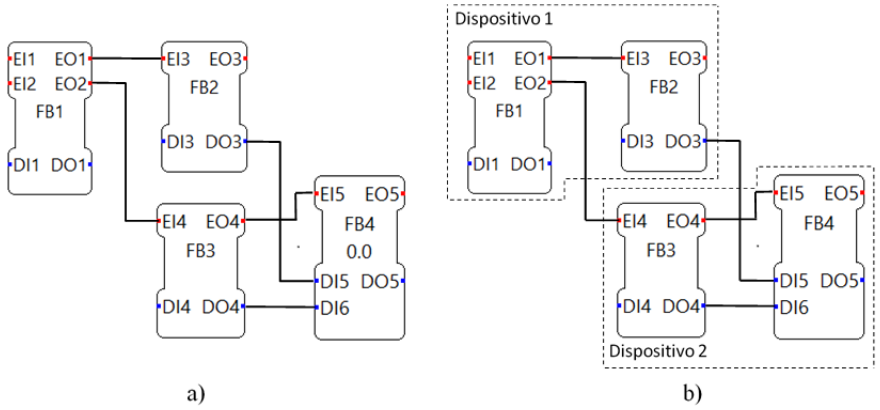


Figura 3.4: Ambigüedad por el orden de ejecución de FB: Caso 2

cación) por el hecho de estar ésta distribuida entre varios dispositivos (Apartado 4.6 del estándar). Es decir, la FBN puede diseñarse y luego distribuirse (desplegarse) sin más modificaciones que colocar SIFB de comunicación en los enlaces inter-controlador. Como vemos, en el caso descrito esta indicación del estándar no se cumple. Para solucionarlo debe modificarse la FBN, por ejemplo incluyendo (ver Fig. 3.5) una sincronización por *rendezvous* mediante `E_REND` [81].

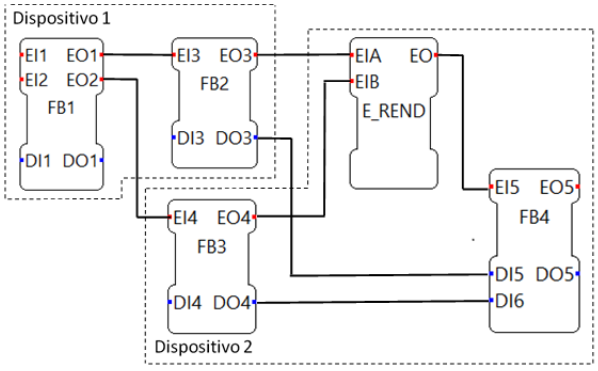


Figura 3.5: Caso 2 con sincronización mediante *rendezvous*

En IEC 61499 las conexiones de eventos entre FB son el medio que tiene el diseñador para definir explícitamente el orden de ejecución de los FB en la FBN [1]. En este sentido, la segunda versión del estándar no ha resuelto totalmente las dudas semánticas detectadas en la primera versión, las cuales generaban comportamientos indeterministas. Así, todavía pueden darse situaciones donde una FBN tenga este tipo de comportamientos. Lo que según [2] depende de la manera de implementar el modelo de eventos. Una implementación habitual es almacenar los eventos por orden de antigüedad en una cola, mediante la que el recurso determina el orden de ejecución de los FB (ver Fig. 3.6). El problema es que la topología y el

timing de una FBN puede ocasionar una pérdida de eventos si se excede la capacidad de esta cola. Dando así lugar a resultados distintos con iguales condiciones de entrada, es decir a, **indeterminismo**. Como indica [72], el uso de colas para almacenar eventos puede acabar en una explosión combinatoria de estados, haciendo difícil predecir si, y cuándo, puede producirse un comportamiento inseguro del sistema. Por otra parte, es necesario almacenar en la cola los eventos junto con sus datos asociados. En caso contrario, en el momento de ser atendido un evento los datos empleados no serán consistentes. Este almacenamiento conjunto de eventos y datos exige un uso elevado de recursos del sistema, principalmente de memoria, en FBN de gran tamaño [127].

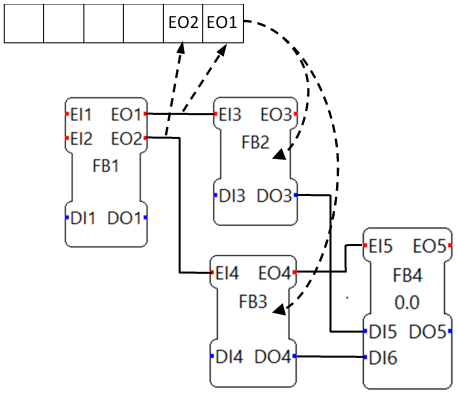


Figura 3.6: Almacenamiento de eventos en una cola

Una solución práctica para tener una FBN determinista es la mostrada en la Fig. 3.7 [2]. Su principio de funcionamiento es el muestreo periódico de las entradas externas mediante el FB `INPUTS`, el cual es activado por eventos generados periódicamente por el FB `RT_E_CYCLE`. Si `INPUTS` detecta cualquier cambio en las entradas lo notifica, mediante el evento `CHG`, al FB `CONTROLLER`; que ejecuta los algoritmos de control. Si como resultado se produce un cambio en el estado del proceso este FB lo notifica al FB `OUTPUTS` para que sea reflejado en las salidas externas. Esta solución es criticada en [1], al indicar que parece más cercana al modelo de *scan* de IEC 61131 que al modelo de eventos de IEC 61499. En particular, hace notar que en este caso los eventos son únicamente empleados para ordenar la ejecución secuencial de los FB, siendo toda la información relacionada con la aplicación manejada por medio de conexiones de datos. El propio autor de la propuesta indica que *"...this solution does not seem to be as exciting as compare to the pure interrupt-driven (in which case the E_CYCLE would not be required),..."*. Otros intentos por incrementar el determinismo en FBN han llevado a la propuesta de diferentes modelos de ejecución, discutidos brevemente a continuación.

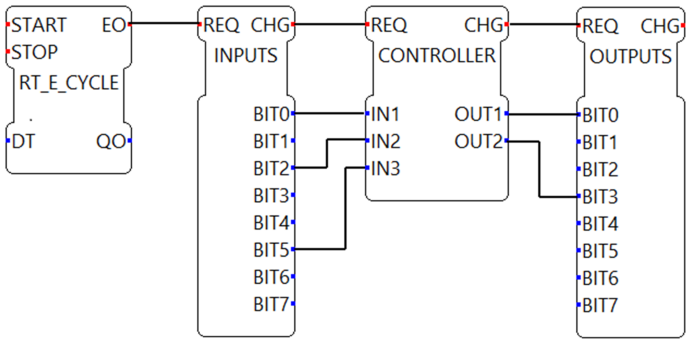


Figura 3.7: Solución práctica para tener una FBN determinista [2]

3.2.2.1. Modelos para determinar el orden de ejecución de los FB

Como se indicó en el Capítulo 1 (ver Sección 1.3.4), el *runtime* FBRT [50] es la primera implementación de IEC 61499. Su modelo de ejecución ha sido analizado en [65] y en [72], donde es denominado **NPMTR** (*Non-Preemptive Multi-Threaded Resource*). En este modelo los eventos son interpretados como llamadas directas a un método en el FB destino, el cual ejecuta el ECC. El orden de ejecución de los FB está determinado por la aproximación *depth-first search* con el que son realizadas estas llamadas al seguir las conexiones entre FB. Este modelo puede presentar un comportamiento no determinista en la ejecución de FBN con conexiones en paralelo o en *feedback* [65].

Una importante propuesta es el modelo de ejecución **secuencial** presentado en [48]. Este modelo tiene unas reglas simples para determinar la ejecución de los FB:

1. No se permite la ejecución concurrente de FB.
2. Los eventos de salida son generados secuencialmente.
3. Los FB son ejecutados en el orden en que han sido generados sus correspondientes eventos de entrada.

Con estas reglas se pretende tener un modelo de ejecución predecible y sencillo de implementar (requiere una única tarea). Este modelo ha tenido una gran influencia en la segunda versión del estándar, por lo que sus principales inconvenientes han sido ya indicados anteriormente: a) la necesidad de almacenar los eventos en una cola, lo que ocasiona posibles pérdidas de éstos, y por lo tanto puede llevar a comportamientos no deterministas de la aplicación; y b) la determinación del orden de ejecución de los FB únicamente en FBN no distribuidas [81].

Respecto del primer inconveniente, un intento de incrementar el determinismo en las FBN, como se indica en [2], es el modelo de ejecución **cíclico** propuesto en [66] y [67]. En él el orden de ejecución de los FB debe ser determinado en tiempo de diseño. Este modelo es similar al modelo de *scan* de IEC 61131.

Por otra parte, respecto del segundo inconveniente los autores del modelo secuencial han propuesto el modelo de ejecución **paralelo** [128], que preserva la semántica en FBN distribuidas [70]. Este modelo es derivado del secuencial a partir de la modificación de las reglas para permitir la ejecución simultánea de FB, así como también la generación simultánea de eventos de salida. Otro motivo a favor de este modelo es que la ejecución de un único FB en cada momento está pensada para arquitecturas *mono-core*, algo que puede estar desfasado actualmente. El modelo de ejecución paralelo permite una implementación más eficiente en las modernas arquitecturas *multi-core* o también en dispositivos realizados mediante hardware programable (p.e. FPGAs) [71].

Otro modelo propuesto es el síncrono [129], basado en considerar que los eventos ocurren en una secuencia discreta de instantes de tiempo, denominados *ticks*. De esta manera, los autores demuestran que las FBN pueden tener un comportamiento determinista. Los FB son módulos que se ejecutan concurrentemente en *pipelines* productor-consumidor.

En la segunda versión de IEC 61499 han sido considerados los modelos secuencial, cíclico y paralelo. Cada modelo determina un orden de ejecución de los FB distinto (ver un ejemplo en [70]), por lo que una misma FBN puede producir resultados diferentes. Es decir, aunque parece que las principales ambigüedades semánticas del estándar han sido resueltas, el problema de la no portabilidad de las FBN persiste. Una propuesta que pretende paliar esta situación se presenta en [130]. Básicamente consiste en modificar (refactorizar) la FBN siguiendo unos patrones de diseño, con el fin de obtener un comportamiento equivalente en cualquier modelo. Esta modificación es sistemática, por lo que el proceso de refactorización puede automatizarse.

3.2.3. Aproximaciones a la implementación de IEC 61499

Además del modelo de ejecución que determina el orden de ejecución de los FB, pueden contemplarse diferentes aproximaciones a la hora de implementar *runtimes* para IEC 61499. El primer trabajo que estudió esta cuestión con cierto nivel de detalle fue [65], donde se analizan siete de estas aproximaciones organizadas a partir de dos criterios:

1. Si hay un orden de ejecución prefijado de los FB o éste depende de la ocurrencia de eventos.

2. Si la implementación es o no multitarea.

Para poder compararlas se realizan tests con FBN sintéticas utilizando FBDK, en el cual es modificado el *runtime* según la aproximación probada. Comparando resultados los mejores tiempos de ejecución se obtienen con el orden de ejecución determinado por los eventos, y una implementación multitarea. Los tests revelan también que la topología de la FBN influye en los rendimientos obtenidos.

Por su parte, en [73] se describen varias aproximaciones agrupadas utilizando como criterio si una tarea es asignada a:

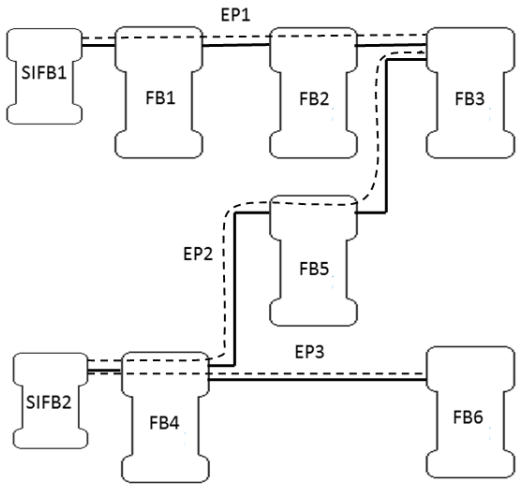


Figura 3.8: Ejemplo de *event-paths* en una FBN [3]

1. Todos los FB.
2. Cada FB.
3. Un subconjunto de los FB.
   a) Con prioridad por tarea.
   b) Con prioridad por *event-path*.
4. Un *event-path*.

Aunque no se muestran tests que permitan comparar dichas aproximaciones, el trabajo indica que la primera aproximación es ineficiente en FBN complejas (i.e. con una gran número de conexiones de eventos y datos), mientras que la segunda tiene importantes costes si el número de FB es elevado. Por el contrario, la asignación de partes de la FBN (bien un subconjunto de FB o bien un *event-path*) a tareas distintas resulta una manera eficiente de implementar un *runtime* para IEC 61499. En alguna de las aproximaciones aparece el concepto de *event-path*, presentado en [3]. Éste representa un camino en las conexiones de eventos que atraviesa varios FB y se inicia por un evento fuente externo generado por un SIFB (ver Fig 3.8). Si varios *event-paths* atraviesan un mismo FB debe establecerse algún mecanismo de exclusión mutua para el acceso a las variables internas. El inconveniente es que éste puede afectar al cumplimiento de los *deadlines* de las tareas (por inversión de prioridad).

Finalmente, en [68] se hace un estudio similar al anterior donde se analizan cuatro aproximaciones a partir del concepto *execution context* (que representa un procesador, tarea o *thread*), encargado de planificar la ejecución de los FB; y cuyas conclusiones son similares al anterior trabajo. Dichas aproximaciones son analizadas brevemente a continuación:

- *Execution context* asignado a un FB: Esta aproximación es poco eficiente debido a que si la FBN es grande habrá también un gran número de *execution context*. Lo que tiene costes en *overheads* generales y en particular por sincronización, algo ya mostrado en los resultados obtenidos previamente por [65].

- *Execution context* asignado a un recurso: Esta aproximación puede tener una implementación compleja si distintas partes de la FBN, alojadas en un mismo recurso, deben ser ejecutadas con diferentes restricciones de tiempo real. Una solución en este caso es implementar el *execution context* mediante un proceso del sistema operativo, el cual a su vez puede tener varios *threads* (a modo de *sub-execution contexts*). Esta solución, además de compleja, tiene costes elevados en términos de uso de memoria y procesador, especialmente si varios recursos ocupan un mismo dispositivo.

- *Execution context* asignado a un dispositivo: En principio esta solución puede ser contraria al estándar, que hace responsable al recurso de planificar la ejecución de FB. Pero en caso de dispositivos sobre plataformas hardware sencillas, donde el recurso puede fundirse con el dispositivo en un único elemento, es una solución adecuada.

- *Execution context* asignado a un área de la FBN: Es una solución que permite cumplir las restricciones de tiempo real de manera eficiente. En particular, se propone la asignación a una *event-chain*, conceptos similar al *event-path* referenciado anteriormente. Como se indicó en el capítulo anterior (ver Sección 2.4) este concepto es la base para la propuesta del modelo adaptado de ejecución mediante *daisy-chains*, objeto de este capítulo.

3.2.4. Principales runtimes propuestos para IEC 61499

El primer *runtime* propuesto para IEC 61499 ha sido FBRT [50], dentro del entorno de desarrollo FBDK. Referenciado anteriormente al presentar su modelo de ejecución NPMTR, está realizado en Java y no cumple restricciones de tiempo real. Ha tenido importancia como referencia en trabajos de la comunidad académica del IEC 61499 y en aplicaciones de demostración. Un *runtime* desarrollado específicamente para una aplicación real se ha propuesto en [49]. Está desarrollado en lenguaje C sobre RTAI, una versión del *kernel* de Linux que cumple restricciones de tiempo real. Su modelo de ejecución es secuencial asignando un *thread* y una cola de eventos a cada uno de los recursos que pueden estar contenidos en un dispositivo. Por ello, los FB de un mismo recurso se ejecutan por el orden de inserción en la cola de sus eventos de entrada.

Por otra parte, en [53] se ha propuesto un *runtime* que cumple restricciones de tiempo real y soporta la reconfigurabilidad de aplicaciones, denominado RTAI-AXE. Al igual que el anterior está desarrollado sobre RTAI, pero en lenguaje C++. El modelo de ejecución parte del concepto de FBC (*Function Block Container*), el cual representa un módulo que agrupa FB que tienen las mismas restricciones de tiempo real. Cada FBC tiene un *thread* asignado, así como una cola de eventos.

También existen módulos para gestionar las conexiones de eventos y datos, a efectos de permitir la reconfigurabilidad de la FBN. Igualmente, con tal fin los tipos de FB están implementados mediante librerías dinámicas de Linux, con el fin de poder ser cargados y descargados de memoria en tiempo de ejecución en un repositorio de *FB-Types*. El trabajo presenta resultados con el análisis de prestaciones del *runtime* implementado que demuestran un comportamiento predecible, salvo en un número pequeño de casos, lo que es atribuido al comportamiento ocasional de RTAI ("*latency killers*" indicados en su manual de usuario). Estos resultados no se comparan con los de otros *runtimes* propuestos hasta ese momento, debido a que, según los autores de esta propuesta, en el momento de su publicación no existen medidas de prestaciones de dichos *runtimes*.

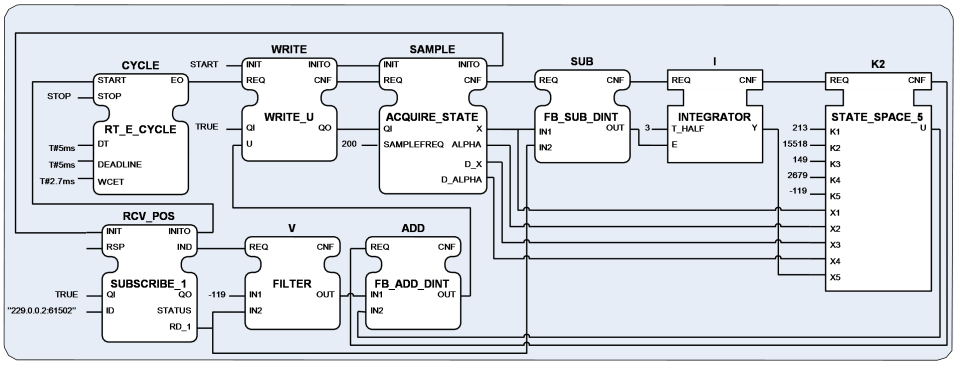


Figura 3.9: FBN para el control de un péndulo invertido con patrón de conexión `REQ-CNF` [4]

El *runtime* FORTE [4] es una de las más importantes implementaciones del modelo de ejecución secuencial descrito anteriormente. Está realizada en C++ y ha sido portado a varias plataformas ARM e Intel con sistemas operativos de tiempo real propietarios y de código libre. En este *runtime* una FBN es dividida en *event-chains*, que pueden tener diferentes restricciones de tiempo real. Cada *event-chain* tiene asignado un *thread* y una cola de eventos. Como todas las implementaciones que utilizan una cola de eventos un aspecto relevante es su dimensionado, de manera que se evite la pérdida de estos. Por otra parte, el uso de *event-chain* puede ocasionar WCET (*Worst Case Execution Time*) elevados, lo que conlleva restricciones en el diseño de la FBN. Por ejemplo, evitar *paths* alternativos o condicionales. En general, se recomienda seguir el patrón de conexión `REQ-CNF` (ver Fig. 3.9). Donde el evento de entrada `REQ` representa la demanda de ejecución de un conjunto de acciones al FB, al que debe seguir el evento de salida `CNF`, que representa la confirmación de esta ejecución. En [4] se muestra los resultados obtenidos con una aplicación sintética con diferente número de *event-chain* (hasta diez de ellos). También se presenta una prueba con una sencilla aplicación real, el control de un péndulo invertido (ver Fig. 3.9). En ambos casos, con las pruebas realizadas, el sistema tiene un comportamiento predecible; mientras que los tiempos de ejecución son algo mayores que en plataformas PLC equivalentes. Los

autores indican que ésto se debe al *overhead* derivado de la ejecución de los FB. Este *runtime* está incluido en la herramientas 4DIAC y en una extensión comercial del fabricante de sistemas de automatización nxtControl, que serán reseñadas en el capítulo posterior.

El primer *runtime* comercial es el incluido en la plataforma ISaGRAF [54]. Este *runtime* sigue el modelo de ejecución cíclico. Así, los FB son invocados en un orden fijado en tiempo de diseño sin que la existencia de eventos de entrada influya en esta invocación. Si un FB no tiene eventos de entrada cuando es ejecutado, se pasa al siguiente. De hecho, este es un *runtime* IEC 61131 sobre el que se ejecutan aplicaciones IEC 61499. Para un análisis detallado de su modelo de ejecución ver [55]. Finalmente, en [2] pueden ser obtenidas referencias a otros *runtimes* que han tenido menor repercusión.

3.3. Propuesta de modelo de ejecución para IEC 61499 adaptado

En la introducción de este capítulo se han indicado los requisitos a cumplir por el modelo de ejecución adaptado al dominio establecido, de manera resumida: a) el modelo debe garantizar un comportamiento determinista de la FBN adaptadas, b) el modelo debe garantizar el cumpliendo de las restricciones de tiempo real establecidas y c) el modelo debe tener una implementación eficiente y escalable.

Con el fin de determinar el modelo de ejecución adaptado se ha discutido en primer lugar la ejecución de FBN en IEC 61499, así como los modelos de ejecución y *runtimes* propuestos hasta la fecha para el estándar. Las principales conclusiones de esta discusión pueden resumirse en dos puntos fundamentales:

- IEC 61499 es complejo de implementar.

  El estándar no está definido de manera formal derivando en ambigüedades semánticas que provocan distintas interpretaciones, las cuales pueden ocasionar FBN no portables o con comportamientos indeterministas. Aunque la segunda versión de IEC 61499 ha resuelto las principales ambigüedades de la primera versión, aún puede haber casos donde se den este tipo de comportamientos. Éstos son inaceptables en cualquier aplicación de control, ya que pueden ocasionar un funcionamiento inseguro del sistema. Dicha situación puede darse especialmente en el modelo secuencial, el más fiel al "espíritu" del estándar [2]. Para "incrementar" el determinismo de las FBN se han propuesto modelos secuenciales que limitan el diseño de las FBN (p.e. mediante el uso de patrones `REQ-CNF`), en la práctica, dichos modelos son similares al *scan* de IEC 61131. Otra solución ha sido la propuesta de modelos cíclicos, donde el orden de ejecución de los FB no viene determinado por los eventos, sino que es definido en tiempo de diseño. De nuevo, estos modelos son también similares al anterior estándar. En general, la implementación de IEC 61499 es compleja, especialmente en modelos de ejecución puramente orientados a eventos.

- IEC 61499 puede no ser más eficiente que IEC 61131.

  En la introducción a este trabajo se ha indicado que la ejecución por eventos de IEC 61499 es más eficiente que la ejecución en *scan* de IEC 61131 (ver Sección 1.4.2). Pero, a la vista de los modelos de ejecución presentados hasta la fecha para IEC 61499, esta afirmación no es necesariamente cierta. Por un lado, los modelos de ejecución secuenciales han mostrado que los *overheads*, debidos a la gestión de eventos, disminuyen o incluso eliminan las diferencias en los tiempos de ejecución con respecto al IEC 61131. Mientras que, por otro lado, los modelos cíclicos tienen un principio de funcionamiento similar al anterior estándar, por lo que su eficiencia es también similar.

Con estos dos puntos en mente, a continuación, confrontaremos los requisitos del modelo de ejecución adaptado con los modelos de ejecución propuestos en IEC 61499 (ver Subsección 3.2.2.1), clasificados según su segunda versión:

- *Modelo cíclico.* En este modelo todos los eventos de un FB son atendidos conjuntamente al ejecutar el FB, estando el orden de ejecución de los FB definido en tiempo de diseño. El modelo adaptado, por el contrario, requiere atender separadamente los eventos por orden de prioridad, con el fin de garantizar el cumplimiento de las diferentes restricciones de tiempo real. Por este motivo, el modelo cíclico es descartado.

- *Modelo secuencial.* En este modelo los FB se ejecutan en un orden establecido por el orden de generación de los eventos. Por otra parte, la ejecución de los FB no es concurrente, es decir, hasta que un FB no finaliza la atención a un evento no puede ejecutarse un FB distinto para atender otro evento. De nuevo, esta característica no es compatible con el establecimiento de un orden de prioridad en los eventos. Ambas características pueden condicionar el cumplimiento de las restricciones de tiempo real del modelo adaptado, por lo que el modelo secuencial es descartado.

- *Modelo paralelo.* Este modelo es una ampliación del modelo secuencial para mantener su semántica en FBN distribuidas. Al igual que el anterior, el orden de ejecución de los FB viene establecido por el orden de generación de los eventos y no permite la ejecución concurrente de FB en un mismo recurso. Como en el caso anterior este modelo es descartado.

Seguidamente confrontamos los requisitos del modelo de ejecución adaptado con los modelos de ejecución para IEC 61499 clasificados, como indica [68] (ver Sección 3.2.3), por la asignación de *execution contexts* a:

- *FB.* Si el número de FB es elevado este modelo tiene como principal inconveniente sus *overheads*. Las FBN en el dominio considerado pueden ser de gran tamaño, por lo que este modelo puede ser poco eficiente, y por lo tanto es descartado.

- *Recursos.* En el caso de FBN que tengan partes con diferentes restricciones de tiempo real este modelo puede tener una implementación compleja y,

como en el caso anterior, con elevados *overheads*. Un requisito del modelo de ejecución adaptado es el cumplimiento de las distintas restricciones de tiempo real en la FBN, lo que hace que el modelo considerado pueda ser poco eficiente, con lo que es descartado.

- *Dispositivos*. Análogo al caso anterior de *execution context* asignado a un recurso.
- *Áreas de la FBN*. Al igual que en la asignación a recursos este modelo permite ejecutar FBN que contengan partes con diferentes restricciones de tiempo real, aunque de manera más eficiente. A la hora de asignar estas partes a *execucion contexts* pueden seguirse dos criterios: a) asignar un subconjunto de FB y b) asignar una cadena de eventos. Este segundo caso corresponde con el modelo de ejecución adaptado, donde los *daisy-chains* tienen diferentes restricciones de tiempo real.

  La eficiencia y comportamiento de las FBN en este caso depende fuertemente del tipo de implementación seguida, especialmente si está basada en el empleo de colas para almacenar eventos y sus datos. Los inconvenientes de las colas han sido ya indicados: a) *overheads* y b) posibilidad de pérdida de eventos, que puede derivar en comportamientos indeterministas de las FBN. Estos inconvenientes se acentúan en FBN con un número elevado de FB, como las del dominio considerado.

  Un tipo particular de implementación de este modelo es la denominada "*Event-path per task, event path-based priority*", identificada en [73]. Como su nombre indica, en esta implementación las cadenas de eventos (aquí denominadas *event-path*) son asignadas a tareas de diferente prioridad. Se puede considerar que estas cadenas de eventos "atraviesan" los FB mediante algún mecanismo de paso de mensajes síncrono o asíncrono. El mecanismo de paso de mensajes síncrono es destacable, ya que no requiere del uso de colas.

La asignación de *execution contexts* a cadenas de eventos tiene ventajas en el caso de FBN con un número elevado de FB y restricciones de tiempo real, como son las del dominio considerado. A continuación, se discute con más detalle el cumplimiento de los requisitos del modelo adaptado de ejecución en un supuesto de: a) *execution contexts* asignados a áreas de la FBN, b) implementación "*Event-path per task, event path-based priority*", donde un *event-path* corresponde a un *daisy-chain* del modelo adaptado; y c) paso de mensajes síncrono:

1. Comportamiento determinista de las FBN adaptadas.

   El supuesto evita la pérdida de eventos por el uso de colas y por ello, el indeterminismo que esta pérdida puede ocasionar. Por otra parte, el hecho de asignar *daisy-chains* a tareas de diferente prioridad hace que una adecuada planificación de su ejecución permita garantizar el comportamiento determinista de la FBN.

2. Cumplimiento de las restricciones del tiempo real establecidas.

   Al igual que en caso anterior, una adecuada planificación de las tareas que ejecutan cada *daisy-chain* permite garantizar el cumplimiento de este requisito.

3. Implementación eficiente y escalable.

   Una posible implementación para los *daisy-chain*, en el supuesto de paso de mensajes síncrono, consiste en una secuencia de llamadas a los algoritmos en los FB. Esta sencilla implementación tiene *overheads* limitados.

4. Facilitar el diseño de las FBN adaptadas.

   En algunos de los modelos anteriores propuestos el despliegue de FBN sobre los controladores requiere, además de SIFB en las conexiones inter-controlador, modificaciones posteriores en la red que garanticen su correcto funcionamiento. Por ejemplo, por motivos de sincronización (ver Subsección 3.2.2). El modelo adaptado de ejecución elimina la necesidad de este tipo de modificaciones debido a la ejecución secuencial de los *daisy-chain*, facilitando así el diseño de las FBN.

Además del cumplimiento de los requisitos del modelo adaptado de ejecución deben ser discutidas otras cuestiones derivadas de la elección del supuesto indicado:

- Los FB no son atómicos.

  El motivo es que la ejecución de un algoritmo en un *daisy-chain* puede ser interrumpida por un evento de más prioridad. A este respecto, el estándar establece la atomicidad de los FB, aunque lo hace de una manera ambigua. Por ejemplo, indica que los recursos pueden planificar la ejecución multitarea de un algoritmo. También indica que los eventos pueden tener atributos, en particular, el de prioridad. El propio estándar hace referencia a este atributo para determinar la prioridad de ejecución de un algoritmo en recursos con *preemptive multitasking* [1].

  En este sentido, han sido presentadas propuestas similares para la implementación del modelo de eventos en IEC 61499, como *event-path* [3] y *event-chain* [68] ya referenciadas anteriormente.

- Incumplimiento de restricciones de tiempo real por inversión de prioridad.

  Los FB son "atravesados" por los *daisy-chains* durante la ejecución de las tareas de tiempo real. Si hay tareas que comparten datos o variables internas de los FB deben utilizarse mecanismos de exclusión mutua. Dichos mecanismos pueden derivar en situaciones de inversión de prioridad de las tareas, que deben tenerse en cuenta para el cumplimiento de las restricciones de tiempo real establecidas en el dominio.

- Incumplimiento de restricciones de tiempo real por bloqueo a tareas menos prioritarias

  Si un *daisy-chain* tiene tiempos de ejecución largos puede bloquear la ejecución de otros de menor prioridad, que pueden no cumplir sus *deadline*. De

nuevo, una adecuada planificación de tareas en tiempo de diseño debe permitir evitar estas situaciones. Por otra parte, debe establecerse como limitación de diseño la "*execution in short*" de los algoritmos.

La discusión anterior indica que el supuesto contemplado es una adecuada propuesta como modelo de ejecución adaptado, por lo que a continuación se aborda su implementación.

3.4. Implementación del modelo de ejecución para IEC 61499 adaptado

En el contexto de este trabajo los FB IEC 61499 son considerados componentes software (ver discusión en Subsección 1.4.2). Atendiendo a la terminología propuesta en [131] una de las características que definen a un componente software es su carácter activo o pasivo, definido de la siguiente manera:

- **Componente activo**: El componente tiene el control de su activación. Es decir, las tareas que ejecutan las funciones (o métodos en términos de programación orientada a objetos) del componente son creadas y activadas desde el propio componente.
- **Componente pasivo**: El componente no tiene el control de su activación. En este caso, las tareas indicadas son creadas y ejecutadas externamente. Es decir, el componente es un contenedor de funciones (o métodos).

En la propuesta de modelo de ejecución adaptado los *daisy-chain* son asignados a tareas, creadas y ejecutadas externamente a los FB. Por lo que, según la clasificación anterior, el FB adaptado tiene el carácter de **componente pasivo**.

3.4.1. Ejecución por eventos *daisy-chains* intra-controlador

La asignación de los *daisy-chain* a tareas permite una sencilla implementación del modelo de eventos IEC 61499 para las conexiones intra-controlador. Ésta implementación, basada en un paso de mensajes síncrono, consiste en una ejecución secuencial de los algoritmos que componen el *daisy-chain* mediante llamadas directas en el orden establecido en éste. Esta implementación es similar a la del modelo de ejecución NPMTR del *runtime* FBRT (ver Subsección 3.2.2.1). En la Fig. 3.10 se representa un ejemplo de ejecución de los *daisy-chain* `NORMAL_RT` y `NORMAL_NRT` en una FBN compuesta por tres FB.

3.4.2. Ejecución por eventos *daisy-chains* inter-controlador

Según el modelo de FB adaptado todos los *daisy-chain* pueden ser inter-controlador, excepto `EXT_EVENT` y `NORMAL_RT` (ver Tabla 2.3) por sus restricciones *hard real-time*. Como se ha indicado en la definición de dicho modelo en estas conexiones se utilizan SIFB *Publish/Subscribe* (ver Sección 2.4).

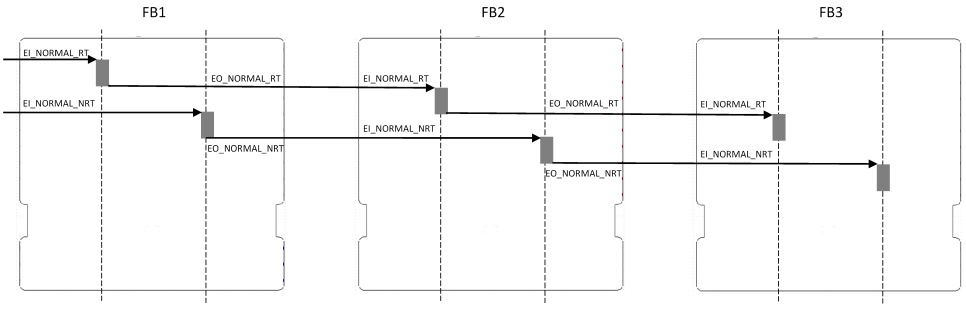


Figura 3.10: Ejecución intra-controlador de `NORMAL_RT` y `NORMAL_NRT`

Para la implementación en este caso el último FB del *daisy-chain* en un controlador llama a una función en el SIFB *Publish*. Esta llamada provoca, a través del *runtime* y de la red de comunicación, la ejecución de una función en el SIFB *Subscribe* que continua la secuencia de llamadas en los FB en el siguiente controlador. En la Fig. 3.11 se muestra la ejecución del *daisy-chain* `NORMAL_NRT` (ver Subsección 2.4.3.5).

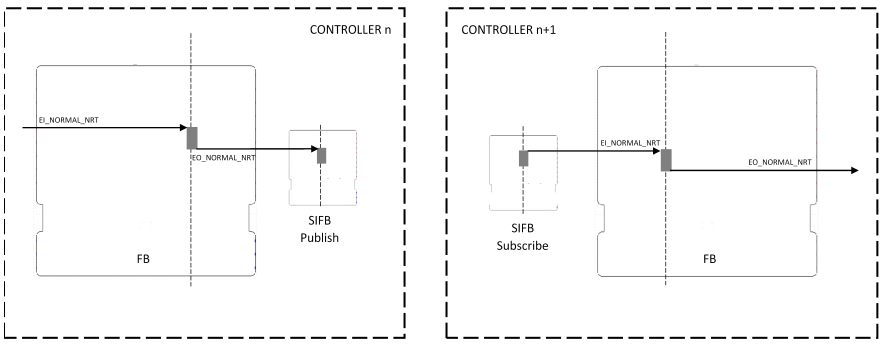


Figura 3.11: Ejecución inter-controlador de `NORMAL_NRT`

3.4.3. Conexiones de datos de entrada y salida en el FB adaptado

El mecanismo de comunicación por paso de mensajes síncrono no requiere el empleo de *buffers* (en este caso una cola) para asegurar la consistencia entre eventos y datos. Por ello, en el modelo adaptado las conexiones intra-controlador en los datos de los FB se implementan mediante simples referencias de memoria (i.e. punteros). Así, un dato de entrada es una referencia al dato de salida al que está conectado (ver Fig. 3.12). Por su parte, las conexiones inter-controlador se implementan mediante referencias entre datos de los FB y de los correspondientes SIFB *Publish* o *Subscribe*.

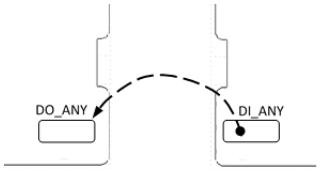


Figura 3.12: Conexiones de datos en el FB adaptado

3.4.4. Control de ejecución del FB adaptado

A la hora de implementar el control de ejecución (en adelante EC) del FB adaptado hay que tener en cuenta que este FB tiene un carácter de componente pasivo, por lo que el control debe efectuarse exteriormente por el propio *runtime*. En este sentido, IEC 61499 no requiere que la implementación del EC sea literal (p.e. mediante una máquina de estados), como indica [72], sino que es suficiente un funcionamiento equivalente.

Daisy-chain	**Tipo**	**Prioridad**	**Tiempo real**	**Origen**
`COLD_INIT`	Aperiódico	5	*Hard*	*runtime* (SIFB)
`WARM_INIT`	Aperiódico	5	*Hard*	*runtime* (SIFB)
`FINALIZE`	Aperiódico	5	*Hard*	*runtime* (SIFB)
`NORMAL_RT`	Periódico	3	*Hard*	*runtime* (SIFB)
`NORMAL_NRT`	Periódico	2	*Soft*	*runtime* (SIFB)
`EXT_EVENT`	Periódico/Esporádico	4	*Hard*	SIFB
`BACKGROUND`	Aperiódico	1	No	cualquier FB
`FAILURE`	Periódico	3	*Hard*	*runtime* (SIFB)
`RECOVERY`	Aperiódico	3	*Hard*	*runtime* (SIFB)

Tabla 3.1: Características de ejecución de las tareas asociadas a los *daisy-chain* (valores altos de prioridad indican una prioridad alta)

El primer paso para la implementación del EC del FB adaptado es caracterizar las tareas asociadas a los *daisy-chain*, las cuales son la base del modelo de ejecución adaptado. Esta caracterización parte del ECC predefinido para el modelo de FB adaptado (ver Fig. 2.21), que determina el comportamiento del FB. En la Tabla 3.1 se muestra dicha caracterización, discutida a continuación por el orden de prioridad en la ejecución de las tareas.

Las tareas de los *daisy-chain* que corresponden a la inicialización y finalización (ver Subsección 2.4.3.1): `COLD_INIT`, `WARM_INIT` y `FINALIZE`, son ejecutadas de manera aperiódica por el *runtime* a través de un SIFB. Esta ejecución no es concurrente y su tiempo de activación no es conocido, aunque puede existir un tiempo máximo para completarla (p.e. en reconfiguración). La siguiente tarea por orden de prioridad correspondiente a `EXT_EVENT`, encargado de atender los eventos externos a la FBN (ver Subsección 2.4.3.2). Estos eventos son convertidos por el *runtime* en eventos internos mediante un SIFB. En el dominio considerado los eventos externos tienen que ver con el controlador de bus de campo. Dependiendo del tipo de bus los eventos externos pueden ser periódicos o esporádicos.

Un nivel menor de prioridad que el anterior corresponde con las tareas que ejecutan las acciones periódicas en funcionamiento normal (ver Subsección 2.4.3.3) o en fallo (ver Subsección 2.4.3.4), que corresponden a los *daisy-chain*: `NORMAL_RT` y `FAILURE` respectivamente. Estas tareas son activadas por el *runtime* a través de un SIFB que genera eventos periódicos. En el caso del modo de recuperación de fallos, la tarea que corresponde al daisy-chain `RECOVERY` es aperiódica, al no ser conocido el tiempo que llevará al FB salir del modo fallo. Aunque, puede existir también un tiempo máximo para completar su ejecución.

La tarea que ejecuta las acciones periódicas de menor prioridad en funcionamiento normal corresponde al *daisy-chain* `NORMAL_NRT` (ver Subsección 2.4.3.3), y es activada por el *runtime* mediante otro SIFB que genera eventos periódicos.

Finalmente, la tarea para realizar las acciones no periódicas sin restricciones de tiempo real corresponde con el *daisy-chain* `BACKGROUND` (ver Subsección 2.4.3.3). A diferencia de las anteriores puede ser activada desde cualquier FB.

3.4.5. Planificación de la ejecución de las tareas asociadas a los *daisy-chains*

El segundo paso para implementar el EC es discutir los aspectos relacionados con la planificación de la ejecución de las tareas asociadas a los *daisy-chains*, con el fin de garantizar que se cumplen las restricciones de tiempo real requeridas por el modelo de ejecución adaptado. A efectos de dicha planificación consideraremos únicamente las tareas periódicas asociadas a: `EXT_EVENT`, `NORMAL_RT` y `NORMAL_NRT`.

Los *daisy-chain* relacionados con la gestión de fallos: `FAILURE` y `RECOVERY` no están asignados a una tarea específica, sino a la misma tarea que `NORMAL_RT`. Si observamos el ECC del FB adaptado (ver Fig. 2.21) comprobamos que los estados **Normal**, **Failure** y **Recovery** se engloban dentro de un estado denominado **RT**. Estos estados tienen restricciones *hard real-time* con el mismo nivel de prioridad y son mutuamente excluyentes. Es decir, el FB en un momento dado está en modo normal, en modo fallo o en modo recuperación de fallo. Por ello, su ejecución puede asignarse a una misma tarea, eliminando así *overheads* innecesarios. Por su parte, las tareas asociadas al resto de *daisy-chain* son aperiódicas, por lo que no puede planificarse su ejecución.

Como se ha indicado anteriormente la tarea asociada a `EXT_EVENT` puede ser periódica o esporádica. En sistemas de tiempo real las tareas esporádicas son un caso particular de las tareas aperiódicas, la diferencia es que en las esporádicas hay un tiempo mínimo entre activaciones consecutivas [132]. Así, desde el punto de vista de su planificación, podemos considerar periódica a la tarea asociada a `EXT_EVENT` si tomamos como periodo su tiempo mínimo de activación.

Para la planificación de las tareas periódicas consideradas se sigue un esquema *rate monotonic*, típico en sistemas de tiempo real. En él cada tarea tiene una prioridad fija donde periodos más pequeños implican prioridades mayores. Las tareas asociadas a `EXT_EVENT`, `NORMAL_RT` tienen restricciones *hard real-time*, por lo que deben cumplir sus deadlines en todo momento. Por el contrario, la tarea asociada a `NORMAL_NRT` tiene restricciones *soft real-time* (ver Subsección 2.3.2), por lo que su *deadline* puede ser sobrepasado ocasionalmente. En cualquier caso,

puede determinarse el cumplimiento de los *deadlines* mediante la ecuación de Liu y Layand [133]:

$$U = \sum_{i=1}^{n} (\frac{C_i}{T_i}) < n(2^{\frac{1}{n}} - 1) \tag{3.1}$$

Donde U representa el total de utilización del procesador en tanto por uno, n el número de *daisy-chain*, C_i el WCET (*Worst Case Execution Time*) y Ti el periodo de cada uno de ellos. En este caso $n = 3$, por lo que si $U < 0{,}78$, los tres cumplirán sus *deadlines*. El WCET de un *daisy-chain* i formado por N FB está determinado por la ecuación:

$$C_i = \sum_{n=1}^{N} C_{FB_n} + C_{dc} \tag{3.2}$$

Donde C_{FB_n} representa el WCET del FB n-ésimo, mientras que C_{dc} representa el tiempo de *overhead* debido a la llamada en secuencia de los algoritmos de los FB en el *daisy-chain*, este tiempo es igual en todos los *daisy-chain*. Los valores de C_{FB_n} y C_{dc} dependen en gran medida del hardware, sistema operativo y plataforma de ejecución subyacentes y pueden ser obtenidos para una FBN concreta por métodos experimentales o analíticos. Los primeros tienen el inconveniente de saber si efectivamente se ha medido el caso peor. Mientras que los segundos están dificultados por el uso en los procesadores de caches, *pipelines*, ejecución especulativa, etc. Para un estudio sobre la obtención de WCET en sistemas de tiempo real ver [134].

En cualquiera de los casos, la manera de programar las tareas periódicas puede facilitar la obtención del WCET. Por ejemplo, dichas tareas deben tener un tiempo de ejecución lo más corto posible ("*execution in short*"). En concreto, deben evitarse los esquemas iterativos no acotados (p.e. de espera ocupada), situación de la tarea periódica representada en el Fig. 3.13a. Para ello, pueden utilizarse esquemas condicionales que comprueban la condición esperada en cada ejecución periódica (ver Fig. 3.13b) o máquinas de estados programadas (ver Fig. 3.13c).

Como ya se indicó anteriormente en el modelo de ejecución adaptado propuesto puede darse el problema de la inversión de prioridad en las tareas periódicas. Este fenómeno se da en tareas que son bloqueadas por otras de menor prioridad al utilizar mecanismos de exclusión mutua en el acceso a datos compartidos. En el peor de los casos la inversión de prioridad puede ocasionar *deadlocks*, en el caso menos malo incrementar el WCET de las tareas de mayor prioridad. Por este motivo, se modifica la ecuación 3.2 que determina el WCET para incluir este tiempo de bloqueo, denominado B_i, quedando:

$$C_i = \sum_{n=1}^{N} C_{FB_n} + C_{dc} + B_i \tag{3.3}$$

Han sido propuestos varios métodos para disminuir e incluso evitar alguno de los efectos que provoca la inversión de prioridad. Uno de ellos es el denominado *priority inheritance*. Este método establece un límite superior del tiempo de

```
void real_time_task() {
  ...
  while(1) {
    ...
    while(! condition); // waiting condition
    ...
    sleep(next_activation_time);
  }
}
```

a) Tarea periódica con espera ocupada

```
void real_time_task() {
  ...
  while(1) {
    ...
    if (condition) {
      ...;
    }
    ...
    sleep(next_activation_time);
  }
}
```

b) Tarea periódica sin espera ocupada (esquema condicional)

```
void real_time_task() {
  ...
  while(1)
    ...
    switch(state) {
      case STATE1:
        actions;
        if (condition2)
          state = STATE2;
        break;
      case STATE2: ...
      ...
    }
    sleep(next_activation_time);
  }
}
```

c) Tarea periódica sin espera ocupada (máquina de estados)

Figura 3.13: Programación de tareas periódicas de tiempo real con y sin espera ocupada

bloqueo producido por la inversión de prioridad y puede calcularse mediante la ecuación [132]:

$$B_i = \sum_{k=1}^{K} uso(k,i)C(k)$$

Donde B_i representa el límite superior del tiempo de bloqueo de la tarea i y K el número de secciones críticas; siendo $uso(k,i) = 1$ si la sección crítica k es usada al menos por una tarea de menor prioridad, y al menos una tarea con una prioridad mayor o igual. En caso contrario, $uso(k,i) = 0$. Por su parte, $C(k)$ es el WCET de la sección crítica k.

El segundo método se denomina *inmediate priority ceiling* y en él el límite superior del tiempo de bloqueo se calcula mediante la ecuación [132]:

$$B_i = \max_{k=1}^{K} uso(k,i)C(k) \tag{3.4}$$

Donde ahora B_i es igual al mayor $C(k)$ de las tareas de prioridad menor a la tarea i. Las ventajas de este segundo método son dos: disminuye el límite superior del tiempo de bloqueo y evita posibles *deadlocks* [132]. Por ello, es el método elegido para implementar el acceso compartido en el FB adaptado. Las secciones críticas de éste corresponden con accesos a variables internas, así como a datos de entrada y salida en el FB adaptado. En principio, estos accesos son debidos a las tareas de los *daisy-chain* concurrentes: `EXT_EVENT`, `NORMAL_RT`, `NORMAL_NRT` y `BACKGROUND`. Aunque, debemos considerar que esta concurrencia depende del tipo de FB adaptado (ver casos de uso en Subsección 2.4.3). El tipo más desfavorable corresponde al FB que guarda y acondiciona la imagen de proceso (`IO_ACOND` en Fig. 2.24), utilizada por los bloques de control de secuencias y posicionamiento de ejes (`PID` en Fig. 2.25). En este tipo de FB se ejecutan de manera concurrente y con restricciones *hard real-time* únicamente las tareas de los *daisy-chain*: `EXT_EVENT` y `NORMAL_RT`.

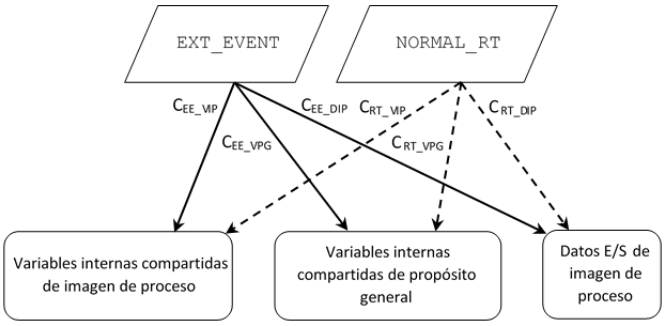


Figura 3.14: Supuesto de secciones críticas y accesos para FB `IO_ACOND`

Con el fin de hacer B_i lo más pequeño posible debe minimizarse la cantidad y tamaño de las secciones críticas. Una sencilla recomendación de diseño es que cada *daisy-chain* tenga variables específicas, es decir, no compartidas con el resto. En el

caso del FB indicado anteriormente podemos disminuir el tamaño de las secciones críticas estableciendo una división en las variables internas y datos compartidos entre imagen de proceso y propósito general (ver Fig. 3.14). Utilizando entonces la ecuación 3.4 puede determinarse B_i para este supuesto, que para el *daisy-chain* de mayor prioridad `EXT_EVENT`, será igual a máx($C_{RT\_VIP}, C_{RT\_VPG}, C_{RT\_DIP}$). Donde C_{RT_VIP}, C_{RT_VPG} y C_{RT_DIP} corresponden con el WCET de las secciones críticas indicadas anteriormente. La ecuación 3.3 mediante la cual se facilita la determinación, experimental o analíticamente, del WCET para `EXT_EVENT` queda como sigue:

$$C_{EE} = \sum_{n=1}^{N} C_{FB_n} + C_{dc} + max(C_{RT\_VIP}, C_{RT\_VPG}, C_{RT\_DIP}) \tag{3.5}$$

Para determinar el WCET en el caso de las conexiones inter-controlador debemos modificar la ecuación 3.3 para incluir los tiempos debidos a la comunicación entre controladores. Estos tiempos dependerán del tipo de red utilizada, en el caso del modelo adaptado de ejecución se ha decidido no utilizar protocolos específicos de tiempo real, sino el estándar de comunicaciones en red local *Ethernet* y dispositivos de tipo *Switched Ethernet*, el cual permite cumplir las restricciones de tiempo real (ver Subsección 1.4.3). Para el tipo de conexión inter-controlador más habitual que correspondiente al *daisy-chain* `NORMAL_NRT` la indicada ecuación como sigue:

$$C_{NRT} = \sum_{n=1}^{N} C_{FB_n} + C_{dc} + C_{BACK\_VPG} + D_{NRT} \tag{3.6}$$

Donde C_{dc} ahora debe incluir el *overhead* en el *runtime* (p.e. hardware, sistema operativo, etc.) debido a la comunicación. Por su parte, C_{BACK_VPG} representa el posible tiempo de bloqueo de `NORMAL_NRT` por parte de `BACKGROUND`. Finalmente, D_{NRT} representa el *delay* por la comunicación *Ethernet*, que incluye el tiempo de propagación de la señal y el procesamiento de las tramas en el dispositivo *Switched Ethernet*. Este *delay* puede calcularse a partir del trabajo presentado en [79], aplicado en comunicaciones de tiempo real para IEC 61499 en [80].

3.5. Conclusiones

El objetivo de este capítulo ha sido proponer un modelo de ejecución de las FBN diseñadas con el modelo de FB adaptado. La ejecución de estas redes, al igual que cualquier aplicación de control, debe garantizar un comportamiento determinista, el cumplimiento de las restricciones de tiempo real y la eficiencia en el uso de recursos del sistema subyacente.

Como primer paso se ha discutido la ejecución de FBN a partir del modelo de eventos definido por IEC 61499. Esta es una definición no formal, por lo que puede llevar a diferentes interpretaciones (o semánticas). Al respecto, se han presentado algunos ejemplos mostrando ambigüedades en el estándar que derivan en comportamientos indeterministas de las FBN.

A continuación, se han discutido los modelos de ejecución propuestos hasta la fecha en IEC 61499, así como los principales *runtimes* que los soportan. Por la manera de atender los eventos esos modelos pueden agruparse en dos categorías. La primera corresponde a los fieles al "espíritu" del estándar, los cuales atienden los eventos por su orden de generación (p.e. modelo secuencial). En la práctica, en estos modelos deben ponerse limitaciones al diseño de las FBN (p.e. patron `REQ-CNF`), con el fin de garantizar un comportamiento determinista o evitar WCETs elevados. La segunda categoría corresponde a modelos donde los eventos son atendidos por un orden de ejecución de los FB, establecido en tiempo diseño (p.e. modelo cíclico); siendo como resultado de ello muy similares a IEC 61131.

Para el modelo de ejecución adaptado se ha tenido especialmente en mente el cumplimiento de las restricciones de tiempo real en FBN con un número elevado de FB, requisito para el control de máquinas herramienta en el dominio considerado. Por ello, se ha establecido un nuevo criterio a la hora de atender los eventos: su orden de prioridad.

El modelo de ejecución propuesto parte de las cadenas de eventos (*daisy-chains*) predefinidas en el modelo de FB adaptado, que son asignadas a tareas de tiempo real. La ejecución de estas tareas se planifica según las prioridades de los eventos establecidas en el modelo indicado. Cada una de estas tareas efectúa una secuencia de llamadas directas a los algoritmos de los FB. Con ello se elimina la necesidad del uso de colas de eventos, evitando así el indeterminismo que pueden ocasionar. Por otra parte, este modelo facilita la obtención de los WCETs en las FBN, aspecto relevante para determinar el cumplimiento de las restricciones de tiempo real. En ese sentido, se ha analizado de manera general el modelo de ejecución adaptado con el fin de poder determinar dichos WCETs. El siguiente paso es proponer una plataforma de ejecución que permita el desarrollo y ejecución de las FBN diseñadas con los modelos adaptados, objeto del siguiente capítulo.

Capítulo 4

Plataforma IEC 61499 adaptada

4.1. Introducción

En los capítulos anteriores se ha caracterizado el control distribuido de máquinas herramienta en sistemas de fabricación ágil, así como propuesto modelos de FB y de ejecución IEC 61499 adaptados a este dominio. Las plataformas existentes para este estándar no soportan dichos modelos, por lo que se propone a continuación la plataforma **COSME** (*COntrol System and Modeling Environment*), que tiene como objeto el desarrollo y la ejecución de esta clase de aplicaciones de control. La plataforma se compone, principalmente, de un entorno de ejecución (*runtime*) y un entorno de desarrollo (en adelante IDE por *Integrated Development Environment*) denominado **Domiciano**. Este entorno permite al ingeniero de control: diseñar, implementar, desplegar y depurar el funcionamiento de las aplicaciones de control en el dominio indicado.

La plataforma COSME está guiada por los principios establecidos en el primer capítulo de este trabajo (ver Sección 1.4), en relación con el software de control IEC 61499 en sistemas de fabricación ágil. Teniendo en consideración los modelos adaptados propuestos dichos principios se concretan como sigue:

- *Facilidad de diseño de las FBN*

  El diseño de estas redes con IEC 61499 es complejo debido a la falta de metodologías o guías en el estándar y a la novedad que introducen los principios sobre los que se apoya (ver secciones 1.3.2 y 1.4). En este trabajo se ha propuesto el uso de modelos IEC 61499 adaptados al dominio de aplicación, mediante los que es posible ocultar parte de la complejidad intrínseca del estándar. Con este fin, estos modelos predefinen los eventos y algoritmos en los FB, facilitando así el establecimiento de **patrones de diseño** para los casos de uso habituales en el dominio, lo que evita el diseño desde cero de las FBN.

Por otra parte, el principio de **transparencia** de los sistemas distribuidos, relacionado aquí en el sentido de ocultar a los diseñadores el sistema distribuido subyacente, también contribuye a facilitar el diseño de las FBN. Este principio no se cumple en IEC 61499, como se ha discutido en el capítulo anterior (ver Sección 3.2). Los motivos son: a) la necesidad de insertar SIFB en las conexiones inter-controlador y b) en algunos casos las FBN pueden requerir modificaciones para que su comportamiento no sea alterado por el hecho de estar distribuida (ver Subsección 3.2.2). Por el contrario, los modelos adaptados al predefinir los eventos de los FB facilitan la inserción automática, mediante el IDE Domiciano, de SIFB en las conexiones inter-controlador. Así mismo, evitan modificaciones posteriores de la FBN para mantener su comportamiento en una ejecución distribuida. Por ello, la plataforma contribuye al cumplimiento del mencionado principio de transparencia, y de esta manera a facilitar el diseño de las FBN.

- *Fiabilidad*

  El modelo de ejecución por eventos hace más difícil el desarrollo de aplicaciones fiables, a diferencia de la ejecución en *scan* de IEC 61131. Un motivo es el comportamiento no determinista de algunas de las implementaciones de IEC 61499 propuestas hasta la fecha. En cambio, el modelo de ejecución adaptado facilita la determinación del comportamiento de la FBN mediante una adecuada planificación de las tareas asociadas a los *daisy-chains*, contribuyendo así a la fiabilidad. Contribuye también a este principio el hecho de evitar el diseño desde cero de las FBN.

- *Rendimiento*

  En IEC 61499 se afirma que el modelo de eventos es más eficiente que el modelo de ejecución en *scan*. Pero los modelos propuestos hasta la fecha para su implementación no ha demostrado de manera clara esta afirmación [1]. En particular, el modelo secuencial, el más fiel al estándar, tiene *overheads* derivados de la gestión de eventos que inciden en su eficiencia. Además, en este modelo es difícil determinar el WCET de las FBN, recomendándose a tal fin el uso de sencillos patrones de conexionado de los eventos (p.e. `REQ-CNF`). Por otra parte, el modelo cíclico, propuesto para garantizar un comportamiento determinista, tiene en la práctica una eficiencia similar al de *scan*. En el caso del modelo adaptado de ejecución se simplifica, mediante los *daisy-chains*, la gestión de eventos disminuyendo así sus *overheads*. Igualmente, este modelo permite la determinación del WCET de una manera más sencilla que en el modelo secuencial.

- *Escalabilidad*

  En IEC 61499 la escalabilidad depende principalmente de dos factores: a) el número de controladores (granularidad) y b) el tamaño de las subFBN en cada controlador. En el dominio considerado el número de controladores es pequeño y limitado, no así el tamaño de las subFBN que puede ser grande. En este caso, la escalabilidad depende de la exploración eficiente de los recursos de la plataforma hardware y del sistema operativo subyacentes.

Al respecto, la ejecución no concurrente de los FB prescrita por el modelo de ejecución secuencial, con el fin de incrementar su determinismo, impide el aprovechamiento eficiente de dichos recursos. La mayoría de propuestas presentadas hasta la fecha para implementar IEC 61499 han sido evaluadas en FBN de pequeño tamaño o sin considerar su escalabilidad (p.e. [4] y [53]). Pero las aplicaciones reales, como las del dominio considerado, pueden tener FBN con un elevado número de FB. Por ejemplo, máquinas herramienta con decenas de ejes y movimientos a controlar, cuyo número además puede variar debido a la reconfiguración. En este sentido, los recursos exigidos por el modelo adaptado dependen en menor medida que otras implementaciones del estándar propuestas del tamaño de las FBN, lo que facilita la escalabilidad.

- *Seguridad de las comunicaciones*

  En IEC 61499 este objetivo no es tenido en cuenta, indicando que es "dependiente de la implementación" y debe ser abstraído por el diseñador mediante los SIFB de comunicación. En nuestro caso, las conexiones predefinidas por el modelo adaptado facilitan el establecimiento de mecanismos y políticas de seguridad específicos. En concreto, por motivos de eficiencia la comunicación de tiempo real entre controladores puede ser efectuada mediante redes dedicadas cableadas y sin encriptación de datos. Mientras que, las comunicaciones con aplicaciones externas, sin restricciones de tiempo real, puede ser realizada mediante redes de propósito general cableadas o inalámbricas, con encriptación de datos y conexiones autenticadas.

- *Soporte a funcionalidades genéricas del dominio de aplicación*

  Las aplicaciones para el control de máquinas herramienta en sistemas de fabricación ágil deben incluir funcionalidades genéricas, por ejemplo, el soporte para el mantenimiento preventivo y predictivo. En este caso, el modelo de FB adaptado permite incorporar código para obtener datos sobre la contabilidad de uso de elementos de la máquina. Otra funcionalidad genérica de este dominio es la comunicación con aplicaciones de tipo MES encargadas de la planificación, calidad o trazabilidad de la producción. Los modelos adaptados facilitan al IDE Domiciano la inserción automática de SIFB, como en el caso de las conexiones inter-controlador, para la comunicación con este tipo de aplicaciones.

4.2. Plataforma COSME

La plataforma COSME tiene como un objetivo principal la ejecución de FBN para el control distribuido de máquinas herramienta en sistemas de fabricación ágil, estas redes se construyen a partir de los modelos de FB y ejecución adaptados propuestos anteriormente. Los requisitos para el diseño e implementación de la plataforma son los siguientes:

1. Arquitectura orientada a cumplir los requisitos del domino de máquinas herramienta en sistemas de fabricación ágil (ver Sección 2.3).

2. Soporte a la ejecución de las FBN diseñadas a partir de los modelos de FB (ver Sección 2.4) y de ejecución (ver Sección 3.3) adaptados a dicho dominio.

3. Cumplimiento de los principios de: facilidad de diseño, fiabilidad, rendimiento, escalabilidad, seguridad de las comunicaciones y soporte a las funcionalidades genéricas del dominio de aplicación (ver Sección 4.1).

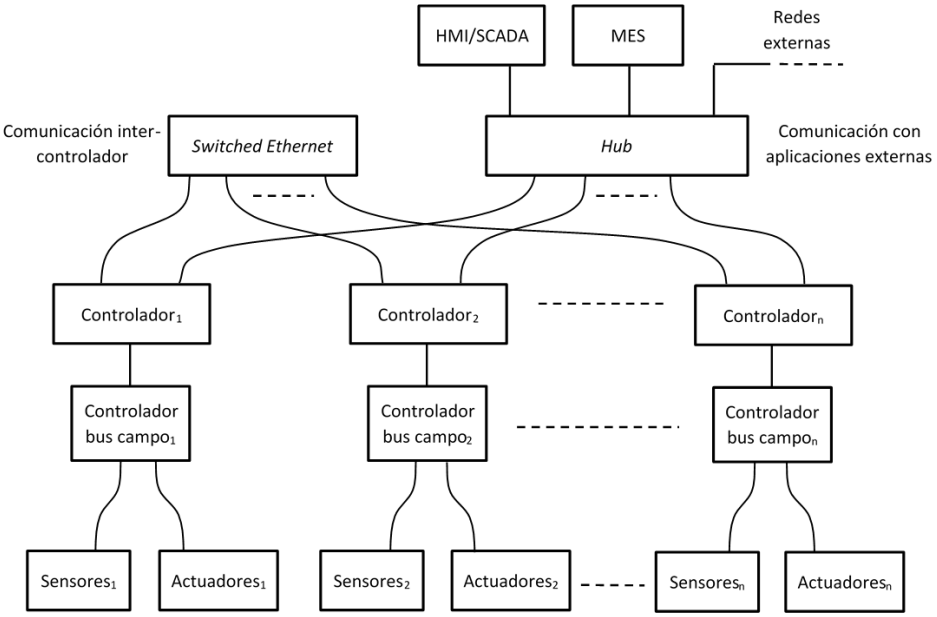


Figura 4.1: Arquitectura de una aplicación de control distribuido COSME

4.2.1. Arquitectura de una aplicación de control COSME

Una aplicación de control distribuido COSME (ver Fig. 4.1) está compuesta por n controladores conectados entre sí, donde cada controlador está conectado a su vez a un controlador de bus de campo para la interfaz de proceso, así como a otros dispositivos que ejecutan aplicaciones externas de (p.e. HMI, MES). A la hora de determinar la arquitectura de la plataforma vamos a considerar en primer lugar los aspectos relacionados con la comunicación entre los diferentes controladores y dispositivos. Particularmente, en lo relativo a las restricciones de tiempo real y la seguridad de las comunicaciones. En este sentido, como se ha indicado anteriormente, en IEC 61499 la comunicación es "dependiente de la implementación" y sus detalles deben ser abstraídos mediante SIFB (ver Subsección 1.4.3).

4.2.1.1. Comunicación en conexiones inter-controlador

Esta comunicación corresponde con las conexiones que se establecen en los *daisy-chains* en una FBN distribuida y tiene restricciones de tiempo real (ver Sec-

ción 2.2.4). En la plataforma COSME no se utilizan dispositivos hardware, buses o protocolos de comunicación específicos de tiempo real. Para el cumplimiento de estas restricciones se emplea únicamente una red *Ethernet* dedicada cableada y dispositivos de tipo *Switched Ethernet* (ver Subsección 1.4.3).

Respecto de la seguridad de comunicaciones en este tipo de conexiones, el empleo de una red dedicada y cableada imposibilita su acceso remoto, quedando la seguridad comprometida únicamente por posibles accesos físicos a la red.

4.2.1.2. Comunicación en conexiones con la interfaz de proceso

Igual que la anterior esta comunicación tiene restricciones de tiempo real. Como se ha indicado para este tipo de conexión en COSME se utilizan buses de campo (ver Sección 2.3). Estos buses son redes específicas utilizadas para el acceso a los sensores y actuadores distribuidos. El controlador accede a dicha red a través de un controlador de bus de campo.

Las consideraciones relacionadas con la seguridad de las comunicaciones son las mismas que en el caso de las conexiones inter-controlador.

4.2.1.3. Comunicación en conexiones con aplicaciones externas

La comunicación con los dispositivos que ejecutan las aplicaciones externas no tiene restricciones de tiempo real. Por este motivo, estas conexiones son realizadas en COSME mediante una segunda red local no dedicada, establecida con algún dispositivo de tipo *Hub*, que habitualmente estará conectada a Internet.

El aspecto más importante en este caso es la seguridad de las comunicaciones. Por ello, la plataforma COSME utiliza conexiones autenticadas con las aplicaciones externas y encriptación de datos.

4.2.2. Arquitectura de un controlador COSME

Desde un punto de vista software una plataforma es una capa intermedia entre las aplicaciones y el sistema subyacente (hardware, sistema operativo, etc.), que tiene como objeto suministrar los servicios necesarios para la ejecución e interoperabilidad de las primeras. En nuestro caso, las aplicaciones de control IEC 61499 están constituidas por FBN que pueden estar distribuidas entre diferentes controladores. Por ello, cada controlador debe tener una arquitectura orientada a proporcionar los servicios requeridos para ejecutar dicha clase de aplicaciones:

- Un entorno de ejecución (o *runtime*) de los FB.
- Interoperabilidad intra e inter-controlador entre FB.
- Comunicación con interfaz de proceso.
- Comunicación con aplicaciones externas.
- Reconfiguración de la FBN.

Teniendo en mente estos servicios un controlador COSME está compuesto por los siguientes elementos (ver Fig. 4.2):

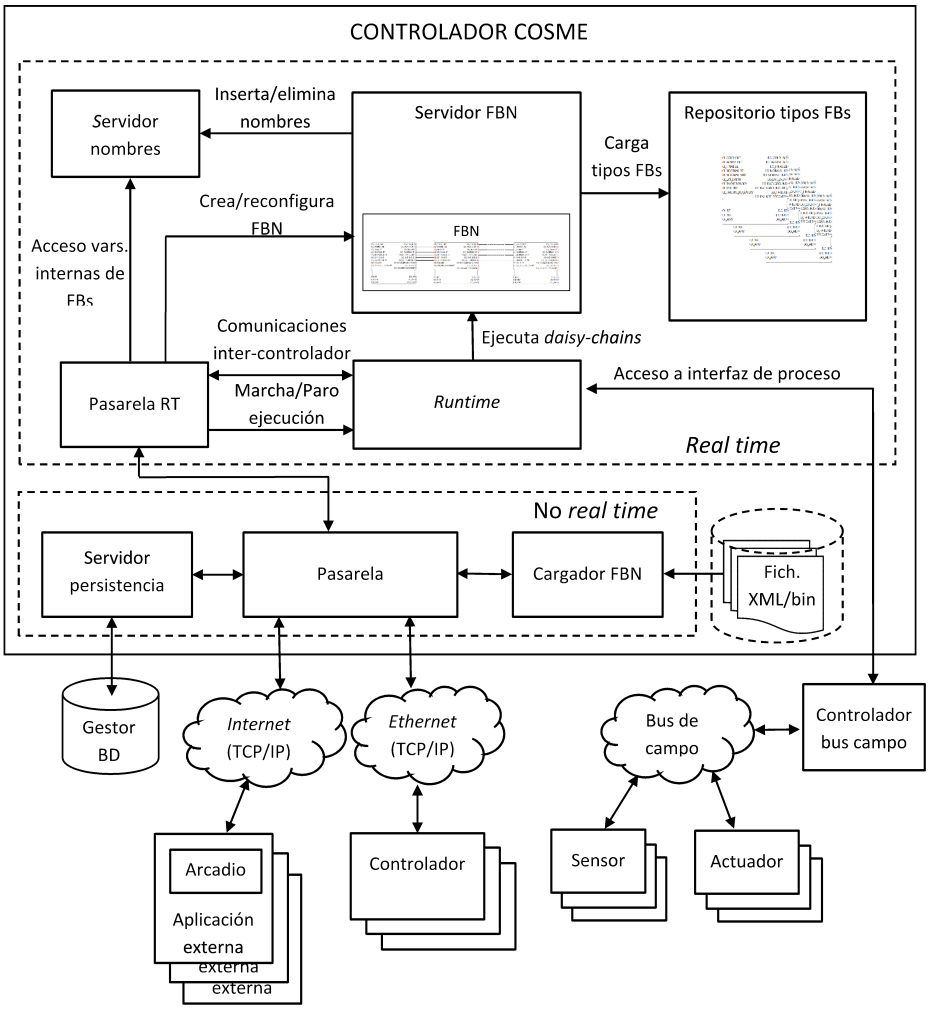


Figura 4.2: Arquitectura de un controlador COSME

- **Servidor de FBN**: Almacena la FBN, proporcionando los servicios para su creación y modificación. En particular, permite: instanciar y destruir FB, establecer conexiones en la FBN, cargar y descargar los tipos de FB desde un repositorio. La modificación de la FBN puede hacerse en tiempo de ejecución, haciendo posible así su reconfiguración.

- ***Runtime***: Ejecuta los *daisy-chain* de la FBN siguiendo el modelo de ejecución adaptado propuesto. Este elemento también se comunica con el controlador de bus de campo con el fin de acceder a la interfaz de proceso.

- **Servidor de nombres**: Establece y controla el acceso a todos los elementos que componen la FBN (tipos de FB, FB, conexiones, variables internas de FB, etc.). De esta manera se facilita la creación y reconfiguración de la FBN, así como el acceso a variables internas de los FB por parte de aplicaciones externas, herramientas de *debug*, etc. Para ello, este servidor contiene un repositorio de nombres (referencias) que permite su: inserción, eliminación, acceso o modificación.

- **Cargador de FBN**: A partir de información almacenada en ficheros carga y pone en ejecución la FBN en el controlador. Igualmente, efectúa la descarga de la FBN y la parada de su ejecución. Estos servicios los realiza a través del **Servidor FBN**.

- **Pasarela**: Realiza la comunicación en los conexiones inter-controlador de la FBN, así como la comunicación con las aplicaciones externas.

- **Servidor de persistencia**: Realiza las funciones de interfaz entre el controlador y una aplicación externa gestor de base de datos. Mediante este servicio puede almacenarse el estado de la aplicación para arranques en caliente y datos generados por funcionalidades genéricas (p.e. mantenimiento preventivo y predictivo, gestión de producción).

4.2.3. Implementación de un controlador COSME

El requisito principal para la implementación de un controlador COSME es el soporte de tiempo real por parte del sistema operativo subyacente. Actualmente existen diferentes sistemas operativos de tiempo real tanto comerciales (Windows Embedded Compact, VxWorks o QNX), como *open source* basados en el sistema operativo Linux.

En el caso de estos últimos, aunque Linux ha sido diseñado con propósito general y no da soporte de tiempo real, desde hace algún tiempo se han desarrollado variantes (*patches*) que modifican el *kernel* de ese sistema a fin de dotarle de tal característica. Ejemplos relevantes son: RTAI, Xenomai y PREEMPT_RT. El principio de funcionamiento de los dos primeros es el mismo, y básicamente consiste en incorporar una capa intermedia (o *microkernel*) entre el hardware y el *kernel* de Linux. Esta capa ejecuta las tareas de tiempo real a la vez que virtualiza el hardware que ve dicho *kernel*, que pasa a ser una tarea más, de menor prioridad, a ejecutar. Por su parte, la variante PREEMPT_RT es una modificación oficial, y

opcional, del *kernel* de Linux que lo convierte en un sistema de tiempo real. Su empleo está menos extendido que las dos primeras variantes y algunos trabajos han indicado que no es 100 % *hard real-time* [135].

Otra posibilidad distinta para disponer de soporte de tiempo real es Real Time Java. Aunque, al ser Java una plataforma virtual debe ser ejecutada sobre un sistema operativo real, en este caso alguno de los anteriormente indicados, por lo que tiene mayores latencias.

Finalmente, en relación con los sistemas operativos de tiempo real, es necesario mencionar el estándar POSIX 1003. Este estándar define un API (*Application Programming Interface*) para asegurar la compatibilidad y portabilidad del código fuente en sistemas de tipo Unix. La mayoría de los sistemas mencionados anteriormente cumplen total o parcialmente este estándar. Resultan de especial interés las partes del estándar 1003.1b y 1003.1c, relacionadas con extensiones de tiempo real y *pthreads* respectivamente.

En la subsección anterior se han identificado los elementos que componen la arquitectura de un controlador COSME y como se relacionan entre sí. Estos elementos están agrupados en dos partes (ver Fig. 4.2): a) elementos con restricciones de tiempo real relacionados con la ejecución de la FBN como ***Runtime***, **Servidor de FBN** y **Servidor de nombres**; y b) elementos que no tienen dichas restricciones. La comunicación entre las dos partes del controlador se realiza a través de la **Pasarela**, que se divide a su vez en dos.

Esta arquitectura ha sido implementada sobre Linux mediante los lenguajes C y Java. El primero ha sido usado por eficiencia en los elementos que se ejecutan con restricciones de tiempo real. Mientras que, el segundo ha sido usado por facilidad de desarrollo en los elementos sin esas restricciones. Se han realizado dos implementaciones alternativas a partir de sistemas basados en Linux:

- *Implementación COSME-RTAI*

  Los elementos del controlador se realizan mediante módulos del *kernel* de Linux ejecutados en una versión de este sistema modificada con el *patch* de RTAI. Estos módulos son partes de sistema operativo que pueden ser cargadas o descargadas de memoria en tiempo de ejecución. Las funciones relacionadas con la ejecución de tiempo real de la FBN han sido implementadas mediante el API de RTAI. La principal ventaja de trabajar en el espacio del *kernel* es tener menores latencias que en el espacio de usuario. Los inconvenientes, la no portabilidad del código fuente y una menor robustez, ya que un fallo grave de la aplicación puede hacer que caiga el sistema.

- *Implementación COSME-POSIX-PREEMPT_RT*

  En esta implementación los elementos se realizan mediante el API estándar de POSIX 1003, ejecutándose en el espacio de usuario de un sistema Linux cuyo *kernel* ha sido modificado con el *patch* PREEMPT_RT. Las ventajas son la portabilidad del código y una mayor robustez de la aplicación de control. Por el contrario, esta implementación tiene latencias mayores que COSME-RTAI y, como se ha indicado anteriormente, no cumple al 100 % restricciones *hard real-time*.

A continuación se discute brevemente por separado los aspectos más relevantes de la implementación de cada uno de los elementos de la plataforma.

4.2.3.1. Servidor de FBN

Los servicios que este elemento de la plataforma proporciona son:

- Carga y descarga tipos de FB.
- Creación y destrucción de FB.
- Establecimiento y modificación de conexiones entre FB.

La implementación de este elemento debe ser abordada desde el punto de vista de la reconfiguración de la FBN. En la introducción a este trabajo (ver Sección 1.1.1) se han descrito los diferentes tipos de reconfiguración: simple, dinámica e inteligente. La versión actual de este elemento proporciona servicios (o primitivas) para la reconfiguración simple. Estos servicios son la base para la reconfiguración dinámica mediante un elemento Gestor de Reconfiguración, a incorporar en la siguiente versión de la plataforma.

La FBN se almacena en memoria mediante estructuras de datos dinámicas de tipo lista. Lo que permite que la creación/eliminación de FB y el establecimiento de conexiones sean realizados con simples operaciones de inserción, eliminación, etc. sobre estas estructuras. La manera de realizar la carga y descarga de los tipos de FB depende de la implementación de la plataforma. En el caso de COSME-RTAI cada tipo de FB es un módulo de *kernel* de Linux. El **Servidor de FBN** carga los módulos correspondientes a los tipos de FB necesarios al iniciar la creación de la FBN, y su caso durante una reconfiguración. Por su parte, en COSME-POSIX-PREEMPT_RT cada tipo de FB es una librería dinámica de Linux. Al igual que los módulos de *kernel* estas librerías pueden ser cargadas o descargadas de memoria durante la ejecución.

4.2.3.2. *Runtime*

Este elemento es el encargado de ejecutar la FBN desplegada en el controlador a partir del modelo adaptado de ejecución propuesto en el capítulo anterior. Siendo dicho modelo una de las partes relevantes del presente trabajo se discutirá la implementación de este elemento con un nivel de detalle mayor que el resto de elementos de la plataforma. Por motivos de sencillez la implementación discutida corresponde a COSME-POSIX-PREEMPT_RT, siendo ésta similar en COSME-RTAI. Además, nos referiremos aquí a la ejecución intra-controlador de las FBN, dejando para la discusión de la **Pasarela** la ejecución inter-controlador.

En el capítulo anterior se ha indicado el carácter de componente pasivo de los FB adaptados, por lo que el control de su ejecución es implementado por el propio ***Runtime*** (ver Sección 3.4). El código simplificado con la implementación de un FB genérico se muestra en las Figs. 4.3, 4.4 y 4.5. Así, en el fichero `fbX.h` se declara la estructura del tipo genérico de FB, que contiene las variables internas

dependientes de la plataforma, así como las variables internas y datos de entrada/salida dependientes de la aplicación. Por su parte, el fichero `fbX.c` contiene las funciones del tipo genérico de FB dependientes de la plataforma:

- Inicialización del tipo de FB: `init_fbX_t`.

  Muestra un ejemplo de establecimiento secciones de críticas tal como han sido descritas en la Subsección 3.4.5 para los *daisy-chain* `EXT_EVENT` y `NORMAL_RT`.

- Obtención del modo de estado del FB: `get_RT_state`.

  Devuelve el valor de la variable que indica si el FB está en modo normal, fallo o recuperación. Se utiliza en la implementación de la gestión de fallos descrita más adelante.

Mientras que las funciones que corresponden a los algoritmos del tipo FB, predefinidos por el modelo de FB adaptado son:

1. Inicialización y finalización: `cold_init_fbX`, `warm_init_fbX` y `finalize_fbX`.
2. Funcionamiento normal: `normal_RT_fbX`, `normal_NRT_fbX` y `background_fbX`.
3. Atención a eventos externos: `ext_event_fbX` .
4. Gestión de fallos: `failure_fbX` y `recovery_fbX`.

El código simplificado del ***Runtime*** es mostrado en las Figs. 4.6, 4.7 y 4.8, donde las tareas asociadas a los *daisy-chain* corresponden con *pthreads*, asignados como se muestra en la Tabla 4.1.

Daisy-chain	***Thread***
`COLD_INIT`	`pthr_cold_init`
`WARM_INIT`	`pthr_warm_init`
`FINALIZE`	`pthr_finalize`
`NORMAL_RT`	`pthr_RT`
`FAILURE`	`pthr_RT`
`RECOVERY`	`pthr_RT`
`NORMAL_NRT`	`pthr_NRT`
`EXT_EVENT`	`pthr_ext_event`
`BACKGROUND`	`pthr_back`

Tabla 4.1: Asignación de los *daisy-chain* a *threads*

El fichero `runtime.h` declara la estructuras: `fb_type`, que define el modelo de FB adaptado y `fb_chain`, que define un FB que forma parte de un *daisy-chain*. Mientras que, el fichero `runtime.c` define las variables dependientes de la plataforma y las funciones:

- Auxiliares: `wait_period` y `wait_ext_event`.

  Espera el cumplimiento de un periodo establecido y la aparición de un evento externo respectivamente.

```
/*
 * fbX.h
 * FB X type definition
 */
typedef struct fbX{
  // platform dependent vars
  int RT_state;
  int *ei_recovery, *ei_failure;
  fb_chain_t fbcs_back[MAX_FBCS];
  ...
  // internal vars
  ...
  // data inputs
  ...
  // data outputs
  ...
} fbX_t;
```

```
/*
 * fbX.c
 * FB X type definition
 */

#include "fbX.h"

// platform dependent vars
pthread_mutex_t mtx_vip, mtx_vpg, mtx_dip;
pthread_mtxattr_t mtxattr;
...

// platform dependent functions
...

void init_fbX_t() {
  ...
  pthread_mutexattr_init(&mtxattr);
  pthread_mutexattr_setprotocol(
     &mtxattr, PTHREAD_PRIO_PROTECT);
  pthread_mutexattr_setprioceiling(
     &mtxattr, EXT_EVENT_PRIORITY);
  pthread_mutex_init(&smtx_vip, &mtxattr);
  pthread_mutex_init(&smtx_vpg, &mtxattr);
  pthread_mutex_init(&smtx_dip, &mtxattr);
  pthread_mtxattr_destroy(&mtxattr);
  ...
}
```

Figura 4.3: Código simplificado del FB adaptado (Parte 1/3)

```
int get_RT_state(void *fb) {
  fbX_t *this = (fbX_t *)fb;
  return this->RT_state;
}

void cold_init_fbX(void *fb) {
  fbX_t *this = (fbX_t *)fb;
  ...
}

void warm_init_fbX(void *fb) {
  fbX_t *this = (fbX_t *)fb;
  ...
}

void finalize_fbX(void *fb) {
  fbX_t *this = (fbX_t *)fb;
  ...
}

void normal_RT_fbX(void *fb, int RT_period) {
  fbX_t *this = (fbX_t *)fb;
  ...
  // mutual exclusion zones
  pthread_mutex_lock(&mtx_vip);
  ...
  pthread_mutex_unlock(&mtx_vip);
  ...
  pthread_mutex_lock(&mtx_vpg);
  ...
  pthread_mutex_unlock(&mtx_vpg);
  ...
  pthread_mutex_lock(&mtx_vip);
  ...
  pthread_mutex_unlock(&mtx_vip);
  ...
  if (condicion)
    background_req(this->fbcs_back);
  ...

  if (failure_condition || *this->ei_failure)
    this->RT_state = FAILURE;
  ...
}
```

Figura 4.4: Código simplificado del FB adaptado (Parte 2/3)

```
void normal_NRT_fbX(void *fb, int NoRT_period) {
  fbX_t *this = (fbX_t *)fb;
  // normal NoRT actions
  ...
}
void ext_event_fbX(void *fb) {
  fbX_t *this = (fbX_t *)fb;
  ...
  // mutual exclusion zones
  pthread_mutex_lock(&mutex_vip);
  ...
  pthread_mutex_unlock(&mutex_vip);
  ...
  pthread_mutex_lock(&mutex_vpg);
  ...
  pthread_mutex_unlock(&mutex_vpg);
  ...
  pthread_mutex_lock(&mutex_vip);
  ...
  pthread_mutex_unlock(&mutex_vip);
  ...
}
void background_fbX(void *fb) {
  fbX_t *this = (fbX_t *)fb;
  ...
}

void failure_fbX(void *fb, int RT_period) {
  fbX_t *this = (fbX_t *)fb;
  ...
  if (*this->ei_recovery)
     this->RT_state = RECOVERY;
}

void recovery_fbX(void *fb, int RT_period) {
  fbX_t *this = (fbX_t *)fb;
  ...
  this->RT_state = NORMAL;
}
```

Figura 4.5: Código simplificado del FB adaptado (Parte 3/3)

- Creación y finalización de la FBN: `init_fbn` y `finalize_fbn`.

  Crea y destruye respectivamente los *pthreads* que corresponden a las tareas asociadas a cada *daisy-chain*

- *Daisy-chain* de inicialización y finalización: `cold_init_task`, `warm_init_task` y `finalize_task`.

  Ejecuta una sola vez los *daisy-chains* indicados al crear y finalizar la FBN.

- Atención a eventos externos: `ext_event_task`.

  Ejecuta el *daisy-chain* indicado cuando se produce un evento externo. La espera de este evento está representada por la función `wait_ext_event`.

- *Daisy-chain* de funcionamiento normal *soft real-time*: `NRT_periodic_task`.

  Ejecuta periódicamente con restricciones *soft real-time* el *daisy-chain*.

- *Daisy-chain* de funcionamiento normal sin restricciones de tiempo real: `background_task`.

  Ejecuta periódicamente con la menor prioridad los *daisy-chain* de *background* activados desde el FB.

- Funcionamiento normal, fallo y recuperación de fallo: `RT_periodic_task`.

  Los tres *daisy-chain* que corresponden a estos modos de funcionamiento están asignados a una misma tarea periódica (ver Subsección 3.4.5). Una variable de estado indica al principio de su ejecución en cuál de los modos está el FB y llama a su correspondiente función. El paso de un modo a otro se realiza mediante la variable de estado `RT_state` en las funciones del FB dependiendo de la aplicación, como se muestra en el codigo simplificado del FB (Figs. 4.3, 4.4 y 4.5). En concreto, el paso de modo normal a modo fallo puede ser provocado por una condición interna en el FB o por la entrada `EI_FAILURE` del FB (que indica un fallo en la zona a la que pertenece el FB). Por su parte, el paso a modo recuperación es provocado por la entrada `EI_RECOVERY`. En este modo el FB permanece durante una sola ejecución, volviendo al modo normal al finalizar.

Como se ha indicado anteriormente, la implementación COSME-RTAI es similar a COSME-POSIX-PREEMPT_RT. En ella el ***Runtime*** es un módulo del *kernel* de Linux, siendo las tareas asociadas a los *daisy-chain* que corresponden con tareas de tiempo real (*rt_tasks*) creadas y planificadas mediante el API específica de RTAI.

Finalmente, el ***Runtime*** también debe facilitar el acceso a la interfaz de proceso por parte de los FB (ver Subsección 2.24). Este acceso está encapsulado en un SIFB implementado para cada tipo distinto de bus de campo utilizado en el sistema. En la implementación COSME-POSIX-PREEMPT_RT la comunicación con el controlador de este bus se realiza, dentro del mencionado SIFB, mediante las llamadas al sistema de POSIX 1003 para acceso a dispositivos de E/S: *open*, *read*, *write* e *ioctl*. Por su parte, en COSME-RTAI la ejecución en el espacio del *kernel* de Linux permite dicha comunicación a través de las instrucciones de acceso directo al hardware de E/S: *inb* y *outb*.

```
/*
 * runtime.h
 */
typedef struct fb_type {
  void (*cold_init)(void *fb);
  void (*warm_init)(void *fb);
  void (*finalize_init)(void *fb);
  void (*ext_event)(void *fb);
  void (*normal_RT)(void *fb, RT_period);
  void (*normal_NRT)(void *fb, NoRT_period);
  int (*get_RT_state)(void *fb);
  ...
} fb_type_t;

typedef struct fb_chain {
  void *fb;
  fb_type_t *fb_type;
} fb_chain_t;
```

```
/*
 * runtime.c
 */
fb_chain_t fbcs[MAX_FBCS], fbcs_ext_event[MAX_FBCS];
pthread_t pthr_cold_init, pthr_warm_init, pthr_finalize,
          pthr_RT, pthr_NRT, pthr_ext_event;

void wait_ext_event() {
  ...
}
int wait_period(struct timespec *next, Long period) {
  timespec_add_us(next, period);
  clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, next, NULL);
}
void init_fbn() {
  ...
  if (cold_init_condition) {
    pthread_create(&pthr_cold_init, NULL,
      (void *)&cold_init_task, NULL);
    pthread_join(pthr_cold_init, NULL);
  }
  else if (warm_init_condition) {
    pthread_create(&pthr_warm_init, NULL,
      (void *)&warm_init_task, NULL);
    pthread_join(pthr_warm_init, NULL);
  }
```

Figura 4.6: Código simplificado del *runtime* para la ejecución intra-controlador (Parte 1/3)

```
  // Periodic tasks
  pthread_create(&pthr_RT, NULL,
     (void *)&RT_periodic_task, NULL);
  pthread_create(&pthr_NRT, NULL,
     (void *)&NRT_periodic_task, NULL);
  ...
}
void finalize_fbn() {
  pthread_t pthr_finalize;

  pthread_cancel(pthr_RT);
  pthread_join(pthr_RT, NULL);
  pthread_cancel(pthr_NoRT);
  pthread_join(pthr_NoRT, NULL);

  if (finalize condition)
    pthread_create(&pthr_finalize, NULL,
       (void *)&finalize_task, NULL);
  ...
}
void cold_init_task() {
  for(fb_chain_t *fbc: fbc_init)
    fbc->fb_type->cold_init(fbc->fb);
}
void warm_init_task() {
  for(fb_chain_t *fbc: fbcs_init)
    fbc->fb_type->cold_warm(fbc->fb);
}
void finalize_task() {
  for(fb_chain_t *fbc: fbcs_fin)
    fbc->fb_type->cold_finalize(fbc->fb);
}
void ext_event_task() {
  struct sched_param param;
  param.sched_priority = EXT_EVENT_PRIORITY;
  sched_setscheduler(0, SCHED_RR, &param);
  while(1) {
    wait_ext_event();
    for(fb_chain_t *fbc: fbcs_ext_event)
      fbc->fb_type->ext_event(fbc->fb);
  }
}
void RT_periodic_task() {
  struct sched_param param; struct timespec next;
  param.sched_priority = RT_PRIORITY;
  sched_setscheduler(0, SCHED_FIFO, &param);
```

Figura 4.7: Código simplificado del *runtime* para la ejecución intra-controlador (Parte 2/3)

```
  clock_gettime(CLOCK_REALTIME, &next);
  while(1) {
    for(fb_chain_t *fbc: fbcs_normal)
      switch(fbc->fb_type->get_RT_state(fbc->fb) {
        case FAILURE:
          fbc->fb_type->failure(fbc->fb,RT_PERIOD);
          break;
        case RECOVERY:
          fbc->fb_type->recovery(fbc->fb,RT_PERIOD);
          break;
        case NORMAL:
          fbc->fb_type->normal_RT(fbc->fb,RT_PERIOD);
          break;
      }
    wait_period(&next, RT_PERIOD);
  }
}
void NRT_periodic_task() {
  struct sched_param param;
  struct timespec next;
  param.sched_priority = NRT_PRIORITY;
  sched_setscheduler(0, SCHED_FIFO, &param);

  clock_gettime(CLOCK_REALTIME, &next);
  while(1) {
    for(fb_chain_t *fbc: fbcs_normal)
      fbc->fb_type->normal_NRT(fbc->fb, NoRT_PERIOD));
    wait_period(&next, NoRT_PERIOD);
  }
}
int background_req(fb_chain_t *fbcs) {
  static pthread_t pthr_back;

  if(pthread_tryjoin_np(pthr_back,NULL) == EBUSY)
    return 0;

  pthread_create(&pthr_back, NULL,
     (void *)&background_task, (void *)fbcs);
  return 1;
}
void background_task(void *fbcs_back) {
  for(fb_chain_t *fbc: (fb_chain_t *)fbcs_back)
    fbc->fb_type->background(fbc->fb);
}
```

Figura 4.8: Código simplificado del *runtime* para la ejecución intra-controlador (Parte 3/3)

4.2.3.3. Servidor de nombres

Los servicios que proporciona este repositorio de nombres al resto de elementos de la plataforma son:

- Inserción y eliminación de nombres.
- Referencia de un nombre.
- Tipo de un nombre.

El repositorio está implementado mediante una estructura de datos de tipo tabla *hash*, donde un nombre es a su vez una estructura de datos que contiene:

- Un identificador, utilizado como clave de búsqueda en la tabla *hash*.
- Una referencia (puntero) a un elemento de la FBN.
- La clase de nombre: tipo de FB, FB, variable interna, etc.
- Las propiedades: público/privado y modificable/no modificable; que indican la posibilidad de acceso y modificación respectivamente del valor referenciado desde una aplicación externa.
- Límites inferior y superior, en el caso de variables internas de tipo numérico.

4.2.3.4. Pasarela

Al igual que en el caso del ***Runtime*** este elemento es parte fundamental para el modelo adaptado de ejecución, por lo que su implementación será discutida con un nivel detalle mayor, también en este caso a partir de la implementación COSME-POSIX-PREEMPT_RT.

Para la comunicación con los controladores y aplicaciones externas (p.e. HMI, MES) la **Pasarela** utiliza una arquitectura cliente/servidor con mensajes de petición/respuesta. Estos mensajes contienen los identificadores de cliente y servidor, una marca de tiempo (*timestamp*) y una petición de servicio. Los tipos de peticiones están relacionados con:

- Creación y reconfiguración de la FBN.
- Puesta en marcha y parada de la aplicación.
- Lectura y escritura de variables internas (públicas y modificables) de los FB.
- Comunicación *daisy-chains* inter-controlador.
- Soporte herramientas de despliegue y *debug* de FBN.

Como ya se ha indicado este elemento está dividido en dos partes: **Pasarela RT**, ejecutada con restricciones *hard real-time* en una *rt_task* (COSME-RTAI) o en un *pthread* (COSME-POSIX-PREEMPT_RT); y **Pasarela**, ejecutada en Java. La comunicación entre ambas partes se realiza a través de dos *FIFOS* del sistema.

Pasarela se encarga de establecer y mantener las conexiones con los controladores y aplicaciones externas. Dichas conexiones son realizadas mediante *network sockets* y a través de ellas se reciben las peticiones de servicio. Las peticiones son pasadas a **Pasarela RT**, que las interpreta, requiere el servicio al elemento correspondiente de la plataforma y devuelve su respuesta a **Pasarela**. Ésta envía dicha respuesta al controlador o aplicación externa solicitante.

Con el fin de facilitar el desarrollo de estas aplicaciones externas se ha implementado una librería (o *middleware*) en Java, denominada **Arcadio** [125]. Esta librería encapsula el protocolo de comunicación de COSME ocultando sus detalles a dichas aplicaciones (ver Fig. 4.2).

En el caso de la comunicación inter-controlador, las peticiones corresponden con los mensajes de publicación generados por el ***Runtime*** al ejecutar cada *daisy-chain*. Estos mensajes son pasados por **Pasarela RT** a **Pasarela** para su envío al siguiente controlador y contienen los datos publicados, así como el nombre del *daisy-chain* a ejecutar. En caso de los datos suscritos en controladores "aguas arriba" (ver Subsección 2.4.3.5) el mensaje contiene únicamente los datos publicados.

El codigo simplificado del ***Runtime*** para la ejecución inter-controlador en el caso de `NORMAL_NRT` se muestra en las Figs. 4.9, 4.10 y 4.11, para el resto de *daisy-chains* distribuidos es similar. Respecto de la ejecución intra-controlador ahora este *daisy-chain* es activado mediante las funciones `wait_nrt_activation` y `nrt_activation`, y la variable de condición `cond_wait_nrt`. En el controlador que inicia la ejecución del *daisy-chain* la activación se produce periódicamente en el *pthread* `pthr_ecycle`. Por su parte, en el resto de controladores la activación se produce al recibir un mensaje de subscripción en la función `subscribe_nrt`.

En relación con el cumplimiento de las restricciones *soft real-time* del *daisy-chain* `NORMAL_NRT` debe tenerse en cuenta que: a) la comunicación inter-controlador se realiza con dispositivos *Switched Ethernet* que evitan los comportamientos no deterministas de las redes *Ethernet*, b) este *daisy-chain* tiene tiempos de ciclo relativamente grandes (hasta 100 ms) y c) la **Pasarela** se ejecuta en una configuración de la máquina virtual Java que minimiza sus *overheads*.

4.2.3.5. Cargador de FBN

Al comenzar el ***Runtime*** su ejecución este elemento realiza una secuencia de arranque para la carga y puesta en marcha, en frío o en caliente, de la FBN (ver Subsección 2.3.4). Para ello, lee ficheros binarios y XML que representan los tipos de FB y la FBN, previamente generados y desplegados en cada controlador por el IDE Domiciano. La secuencia está compuesta por peticiones de servicio del **Cargador de FBN**, a través de la **Pasarela**, a diferentes elementos de la plataforma y consta de los siguientes pasos:

1. Carga de tipos de FB (a **Servidor de FBN**).
2. Creación de los FB (a **Servidor de FBN**).
3. Establecimiento del estado inicial en frío o en caliente de los FB (a **Servidor de nombres** y **Servidor de Persistencia**).

```
pthread_t pthr_cold_init, pthr_warm_init, pthr_finalize,
          pthr_RT, pthr_NRT, pthr_ext_event, pthr_ecycle;

pthread_mutex_t mutex_nrt;
pthread_cond_t cond_wait_nrt;
int nrt_suspend;

int wait_period(struct timespec *next, long period) {
  timespec_add_us(next, period);
  clock_nanosleep(CLOCK_REALTIME, TIMER_ABSTIME, next, NULL);
}
void wait_nrt_activation() {
  pthread_mutex_lock(&mutex_nrt);
  nrt_suspend = 1;
  do {
    pthread_cond_wait(&cond_wait_nrt, &mutex_nrt);
  } while(nrt_suspend);
  pthread_mutex_unlock(&mutex_nrt);
}
void nrt_activation() {
  pthread_mutex_lock(&mutex_nrt);
  nrt_suspend = 0;
  pthread_mutex_unlock(&mutex_nrt);
  pthread_cond_signal(&cond_wait_nrt);
}
void publish_nrt() {
  ...     // data out
  send_message_controller();
}
// called by comm runtime
void subscribe_nrt() {
  receive_message_controller();
  ...     // data in
  nrt_activation();
}
// called by comm runtime
void subscribe_data() {
  receive_message_controller();
  ...     // data in
}
void init_fbn() {
  ...
  // Init
  if (cold init condition) {
    pthread_create(&pthr_cold_init, NULL,
       (void *)&cold_init_task, NULL);
    pthread_join(pthr_cold_init, NULL);
  }
```

Figura 4.9: Código simplificado del *runtime* para ejecución inter-controlador de `NORMAL_NRT` (Parte 1/3)

```
  else if (warm init condition) {
    pthread_create(&pthr_warm_init, NULL,
       (void *)&warm_init_task, NULL);
    pthread_join(pthr_warm_init, NULL);
  }
  // Periodic tasks
  pthread_create(&pthr_RT, NULL,(void *)&RT_periodic_task, NULL);
  pthread_create(&pthr_NRT, NULL,
    (void *)&NRT_periodic_task, NULL);
  pthread_create(&pthr_ecycle, NULL,
    (void *)&ecycle_periodic_task, NULL);
  ...
}
void finalize_fbn() {
  pthread_t pthr_finalize;

  pthread_cancel(pthr_RT);
  pthread_join(pthr_RT, NULL);

  pthread_cancel(pthr_ecycle);
  pthread_join(pthr_ecycle, NULL);

  pthread_cancel(pthr_NoRT);
  pthread_join(pthr_NoRT, NULL);

  if (finalize condition)
    pthread_create(&pthr_finalize, NULL,
      (void *)&finalize_task, NULL);
  ...
}
void ecycle_periodic_task() {
  ...
  struct sched_param param;
  struct timespec next;
  param.sched_priority = ECYCLE_PRIORITY;
  sched_setscheduler(0, SCHED_FIFO, &param);

  clock_gettime(CLOCK_REALTIME, &next);

  while(1) {
   if (is_first_controller())
     nrt_activation();
    wait_period(&next, ECYCLE_PERIOD);
  }
}
```

Figura 4.10: Código simplificado del *runtime* para ejecución inter-controlador de `NORMAL_NRT` (Parte 2/3)

```
void NRT_periodic_task() {
  ...
  struct sched_param param;
  struct timespec next;
  param.sched_priority = NRT_PRIORITY;
  sched_setscheduler(0, SCHED_FIFO, &param);

  while(1) {
    wait_nrt_activation();

    for(fb_chain_t *fbc: fbcs_normal)
      fbc->fb_type->normal_NRT(fbc->fb, NoRT_PERIOD));

    publish_nrt();
  }
}
```

Figura 4.11: Código simplificado del *runtime* para ejecución inter-controlador de `NORMAL_NRT` (Parte 3/3)

4. Establecimiento de conexiones intra-controlador (a **Servidor de FBN**).
5. Establecimiento de conexiones inter-controlador (a **Servidor de FBN** y **Pasarela**).
6. Recuperación de variables internas persistentes de los FB (a **Servidor de Persistencia**).
7. Ejecución de funciones de inicialización en frio o caliente de los FB (a ***Runtime***).
8. Inicio de ejecución de las funciones normales periódicas (a ***Runtime***).

Por su parte, la parada de la ejecución de la FBN, en frío o en caliente, es a demanda de una aplicación externa (p.e. HMI, IDE Domiciano) y la secuencia de pasos es la siguiente:

1. Parada de ejecución de las funciones normales periódicas (a ***Runtime***).
2. Ejecución de las funciones de finalización (a ***Runtime***).
3. Almacenamiento de las variables internas persistentes de los FB (a **Servidor de Persistencia**).
4. Cierre de las conexiones inter-controlador (a **Pasarela**).
5. Almacenamiento del estado de los FB en caso de parada en caliente (a **Servidor de Persistencia**).
6. Destrucción de los FB (a **Servidor de FBN**).
7. Descarga de tipos de FB (a **Servidor de FBN**).

4.2.3.6. Servidor de persistencia

Los servicios concretos que proporciona a los elementos que requieren de persistencia de datos son:

1. Escritura persistente de variables internas de los FB.
2. Escritura persistente periódica de variables internas de los FB.
3. Lectura persistente de variables internas de los FB.

Para prestar estos servicios se usa un gestor de base de datos relacional (DBMS). El **Servidor de persistencia** actúa de interfaz entre el DBMS y los elementos de la plataforma, con el fin de abstraer los detalles del primero. En la versión actual de COSME se ha empleado el DBMS Java Apache Derby, aunque pueden ser empleados otros (p.e. MySQL), debido a es "ligero" y puede ser incluido en el propio fichero ejecutable de la **Pasarela**.

4.3. IDE Domiciano

COSME incorpora el IDE Domiciano mediante el que es posible diseñar, implementar, desplegar y probar las FBN que constituyen las aplicaciones de control ejecutadas en esta plataforma. El estándar IEC 61499 se basa en el paradigma de la programación orientada a componentes, uno de los principios generales para el diseño del software de control en sistemas de fabricación ágil (ver Sección 1.2). Este paradigma es un medio para abordar la complejidad creciente del software y limitar los costes de desarrollo.

Una metodología de la ingeniería del software que hace uso del citado paradigma es la denominada *Generative Programming* [136]. Su principal aportación es el paso de una construcción de sistemas software basada en la reutilización de componentes previamente desarrollados, a una construcción mediante componentes generados automáticamente bajo demanda, a partir de modelos de alto nivel representados de manera textual o gráfica. Esta metodología es seguida en Domiciano, donde los FB (i.e. componentes) son generados automáticamente por este IDE a partir del modelo de FB adaptado.

Para el desarrollo de las FBN Domiciano incluye elementos con los que se puede:

- Diseñar FBN con FB definidos mediante el modelo de FB adaptado.
- Desplegar FBN sobre los controladores.
- Depurar el funcionamiento de FBN.

Domiciano está implementado en Java, por lo que es ejecutable en distintos sistemas operativos. Su diseño es modular y puede ser extendido de manera sencilla para añadir nuevos lenguajes para la programación de los tipos de FB. Los editores gráficos y de código C están realizados mediante bibliotecas con licencia GPL del IDE de Java Netbeans. Para el despliegue de las FBN en los controladores se usan protocolos SAMBA o SCP y aplicaciones de ejecución remota de comandos Linux como *plink*.

4.3.1. Elementos arquitectónicos del IDE

4.3.1.1. Editor de tipos de FB

Este editor permite definir tipos de FB a partir del modelo de FB adaptado, representado en un fichero XML que contiene la plantilla (o *template*) con su código fuente. Cada uno de los tipos de FB se guarda también en un fichero XML, estos ficheros se almacenan en un biblioteca de tipos. El primer paso para crear un tipo de FB es declarar sus variables internas, así como los datos de entrada y salida. En cada una de estas declaraciones debe especificarse:

1. *Nombre*
2. *Tipo*, que puede ser predefinido (entero, real, etc.) o bien definido previamente mediante un editor de tipos de variables internas y datos.
3. *Valor inicial* asignado a la variable o dato en el arranque en frío.
4. *Longitud* en cadenas de caracteres.
5. *Vector*, indicando su número de dimensiones y tamaño.
6. *Público*, para permitir el acceso desde aplicaciones externas vía **Pasarela** y **Servidor de nombres**.
7. *Modificable*, para permitir la modificación de su valor, en el caso de variables o datos declarados previamente como públicos.
8. *Límite inferior* y *límite superior*, para establecer un rango de valores, en el caso de variables o datos previamente declarados como públicos y modificables.
9. *Persistente*, para almacenar automáticamente el dato o variable.
10. *Periodo*, en caso de variable o dato persistente indica el tiempo a transcurrir entre cada almacenamiento.

Seguidamente el diseñador debe escribir el código de los algoritmos de los FB. Al respecto, Domiciano puede incluir editores de código para diferentes lenguajes de programación, siendo en la versión actual estos lenguajes C e IEC 61131 FBD (*Function Block Diagram*). En este segundo caso, el IDE incorpora un editor gráfico y una biblioteca de componentes estándar (funciones numéricas, contadores, temporizadores, etc.). Por otra parte, la programación de sistemas secuenciales se efectúa con el lenguaje IEC 61131 SFC (o GRAFCET) mediante un editor gráfico específico. Las acciones en las etapas del SFC, así como las transiciones entre etapas pueden programarse en cualquiera de los lenguajes incluidos en el IDE.

4.3.1.2. Editor de FBN

Domiciano incluye un editor gráfico con el objeto de permitir el diseño de las FBN. El modelo general de FBN está especificado en un fichero XML que contiene la configuración inicial de la red (p.e. subFBN y tipos de FB predefinidos). Al igual que en el caso anterior, la FBN se guarda en un fichero XML conteniendo tipos e instancias de FB, conexiones, orden de ejecución y distribución sobre los controladores.

Para facilitar el diseño de estas redes, especialmente en el caso de un gran número de FB, el editor permite definir elementos, denominados capas y vistas, que facilitan la organización de la representación gráfica de la FBN. En las capas se representan gráficamente partes de la FBN (i.e. subFBN) como la interfaz de proceso, el control de ejes o la gestión de fallos. Por su parte, las vistas representan conjuntamente una o más de estas capas.

El diseño de una FBN consiste en instanciar FB de los tipos ya definidos, realizar las conexiones entre sus datos e indicar su distribución. Por su parte, las conexiones de eventos siguen el modelo de ejecución adaptado, por lo que se ocultan en la representación de la FBN, teniendo únicamente el diseñador que establecer el orden de ejecución de los FB en los *daisy-chain*. Al respecto, las conexiones intra-controlador e inter-controlador se realizan de manera indistinta en esta fase, ya que en la fase posterior de despliegue se incorporará de manera automática la funcionalidad de los SIFB de comunicación (ver Subsección 3.4.2).

4.3.1.3. Generador de código

Este elemento del IDE es el encargado de generar automáticamente el código de cada uno de los tipos de FB previamente definidos, a partir de los ficheros XML que los representan. El resultado producido por el generador son ficheros con el código fuente en lenguaje C para cada tipo de FB. Antes, el diseñador debe especificar la versión de ***Runtime*** que ejecutará dicho código: COSME-POSIX-PREEMPT_RT o COSME-RTAI. En la fase posterior de despliegue se realizará la compilación localmente en cada uno de los controladores. Así, es posible tener diferentes ***Runtimes*** que ejecuten la FBN distribuida. Además del código C son generados también ficheros *makefile*, utilizados para automatizar la compilación mediante la herramienta estándar *make*.

4.3.1.4. Herramienta de despliegue

Una vez diseñada la FBN y generado el código de los tipos de FB esta herramienta se encarga de desplegar la FBN sobre los controladores. El despliegue consiste primero en la transferencia de los ficheros XML de cada subFBN, así como de los ficheros con el código fuente de los tipos de FB y *makefiles* necesarios para la compilación local. Y seguidamente en la invocación, de forma remota, del compilador en cada controlador para obtener el código ejecutable de los FB.

Una vez desplegada correctamente la FBN la herramienta permite poner en marcha su ejecución mediante la correspondiente petición, a través de la **Pasarela**, al **Cargador de FBN** en cada uno de los controladores. Antes de empezar la

ejecución las pasarelas establecerán las conexiones inter-controlador con los datos almacenados en los ficheros XML de las subFBN.

4.3.1.5. Herramienta de depuración (*debugger*)

Domiciano incorpora una herramienta para facilitar el acceso, a través de la **Pasarela**, a variables internas y datos de entrada y salida de los FB. Mediante este acceso es posible comprobar el funcionamiento de la FBN, y por lo tanto efectuar la depuración de errores (o *debugging*). Las variables y datos son visualizados agrupados en tablas o en gráficas (en el caso de valores numéricos) creadas mediante la propia herramienta. El periodo de actualización o refresco de sus valores se puede establecer en cada tabla o gráfica independientemente. Este periodo es normalmente más grande que el periodo de ejecución de las tareas de tiempo real, por lo que determinados valores pueden no ser muestreados. Por ello, es también posible registrar el valor de variables y datos en cada ciclo de ejecución de tiempo real durante un tiempo establecido. El modo de funcionamiento es similar al de los analizadores lógicos utilizados en sistemas electrónicos digitales.

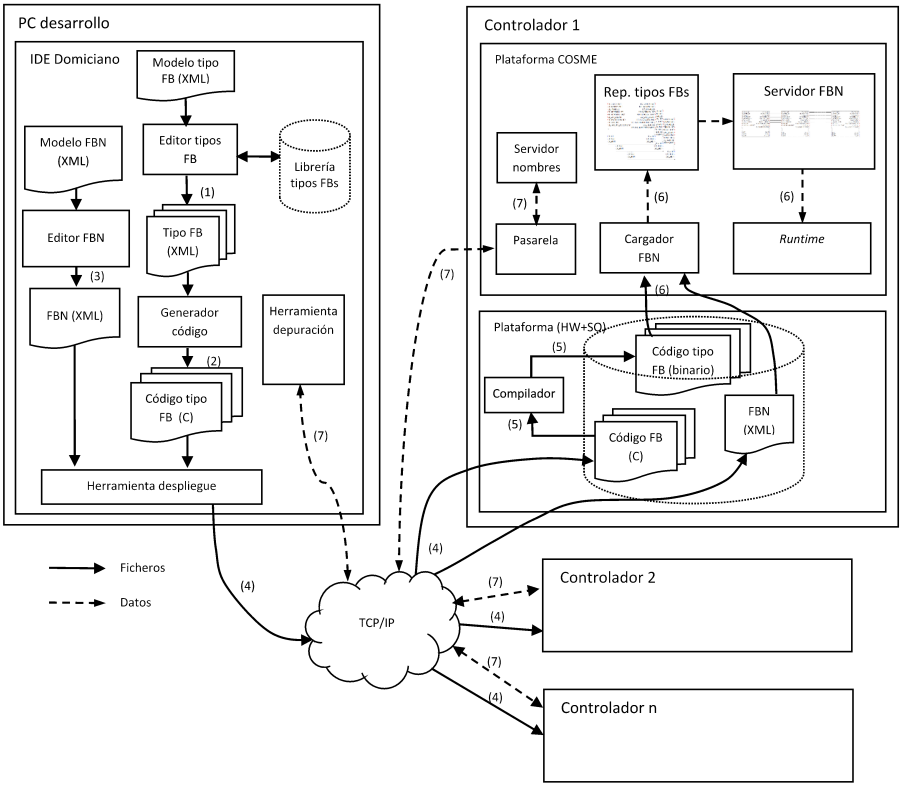


Figura 4.12: *Workflow* para el desarrollo de FBN con Domiciano

4.3.2. Desarrollo de FBN con Domiciano

El desarrollo de FBN con Domiciano sigue un *workflow* (ver Fig. 4.12) que consta de los siguientes pasos:

1. Definición de los tipos de FB.
2. Generación de los tipos de FB.
3. Diseño de la FBN.
4. Despliegue de la FBN.
5. Compilación local de los tipos FB.
6. Carga y ejecución en cada controlador.
7. Depuración (*debugging*).

4.3.2.1. Definición y generación de tipos de FB

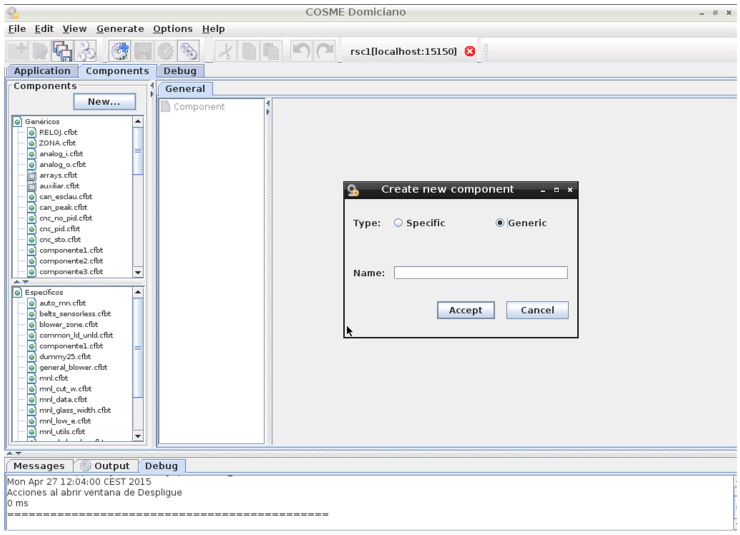
En este paso el diseñador define los tipos de FB que serán posteriormente instanciados durante el diseño de la FBN (ver Fig. 4.13a). Los tipos de FB son almacenados en una biblioteca organizados en dos clases: a) específicos y b) genéricos. Los tipos específicos corresponden a funcionalidades dependientes de la aplicación (p.e. lógica de control principal), por lo que serán utilizados únicamente en una FBN en particular. Por su parte, los tipos genéricos corresponden a funcionalidades que son reutilizables en diferentes FBN (p.e. PID).

Para definir un tipo de FB el diseñador debe primero declarar las variables internas y los datos de entrada y salida (ver Fig. 4.14a y b). Estos elementos pueden ser declarados de tipos simples predefinidos en el lenguaje (enteros, coma flotante, etc.) o de tipos estructurados, los cuales son definidos previamente mediante un editor específico incluido en el IDE (ver Fig. 4.13b).

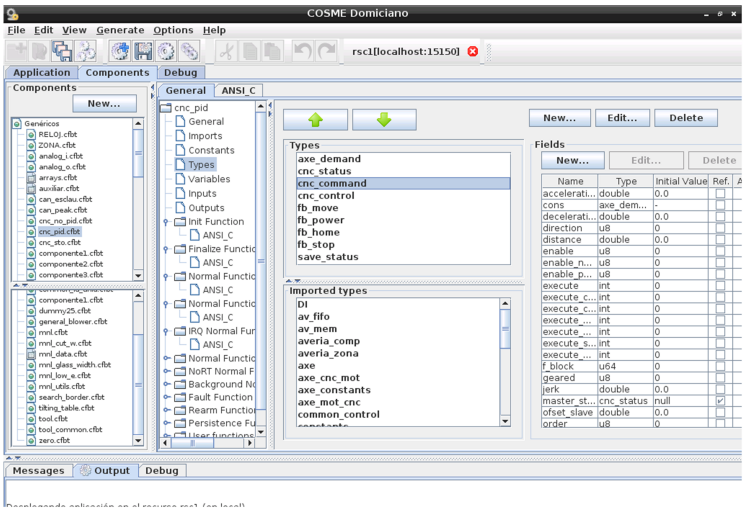
Seguidamente el diseñador debe programar las acciones, en lenguaje C en la versión actual de Domiciano, a realizar en los algoritmos (o funciones) de los FB (ver 4.15a). Como se indicó anteriormente, es posible también utilizar el lenguaje SFC para la programación de sistemas secuenciales (ver 4.15b), habitual en plataformas de PLCs comerciales. Una vez definido el FB puede realizarse una compilación en el equipo de desarrollo, con el fin de adelantar la detección de errores sencillos sintácticos. El último paso es generar el FB, que es almacenado en la librería de tipos mediante un fichero XML.

4.3.2.2. Diseño de la FBN

Este paso comienza creando una nueva FBN, a la que se le asigna un nombre, a partir de una configuración inicial (modelo de FBN) (ver Fig. 4.16a). A continuación el diseñador instancia y realiza las conexiones entre FB únicamente para los datos de entrada y salida, al estar las conexiones de eventos predefinidas por el modelo de ejecución (ver Fig. 4.16b).

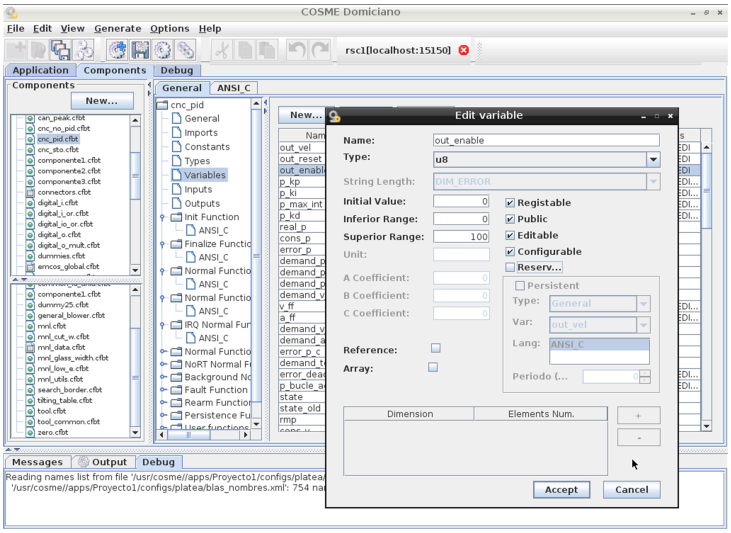


a) Definición de un tipo de FB

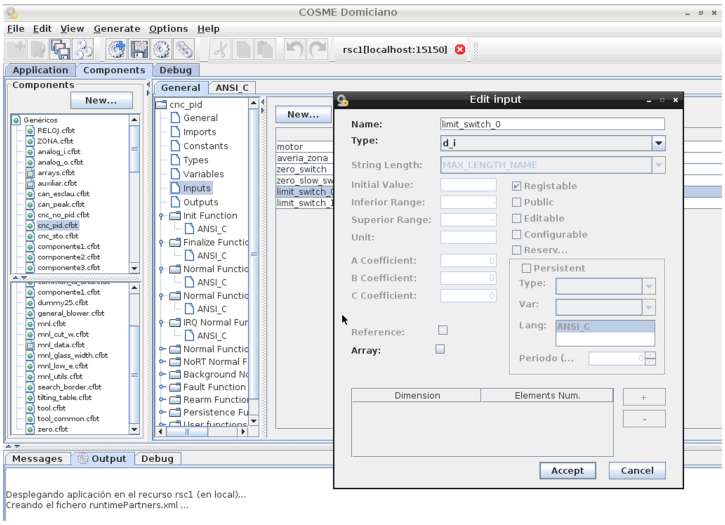


b) Definición de un tipo de variable o dato

Figura 4.13: Definición de tipos de FB (1/2)

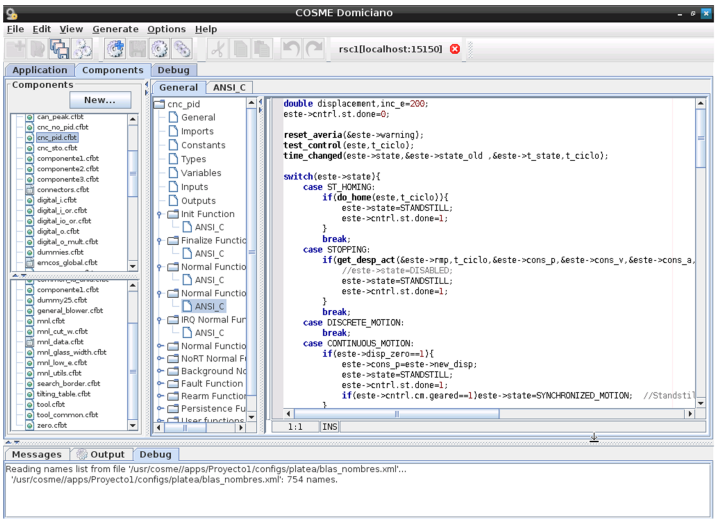


a) Definición de una variable interna

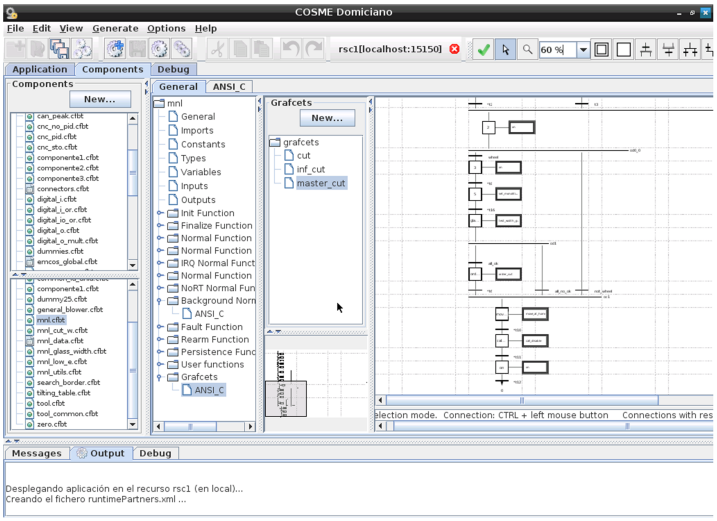


b) Definición de un dato de entrada

Figura 4.14: Definición de tipos de FB (2/2)

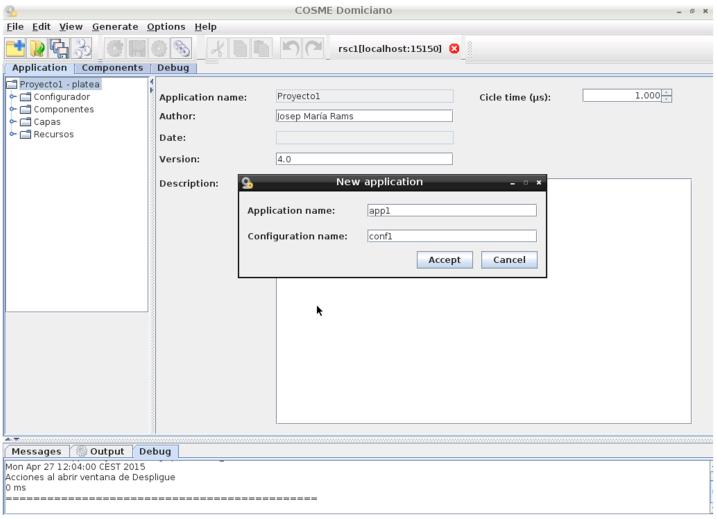


a) Programación algoritmo FB (lenguaje C)

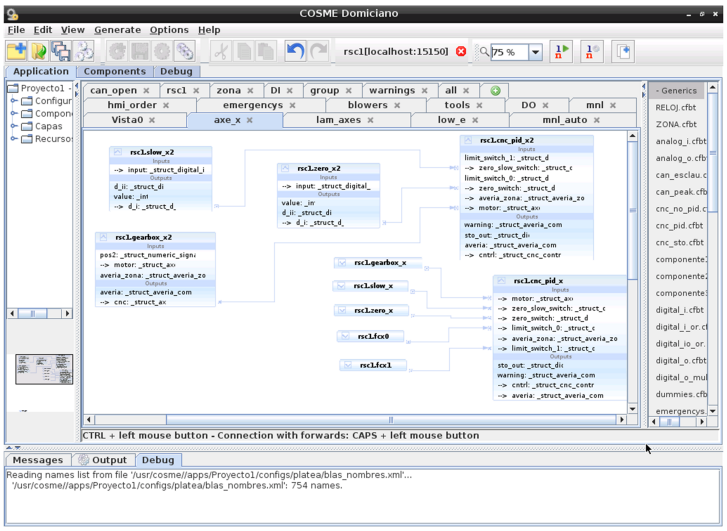


b) Programación algoritmo FB (lenguaje SFC)

Figura 4.15: Programación de algoritmos del FB

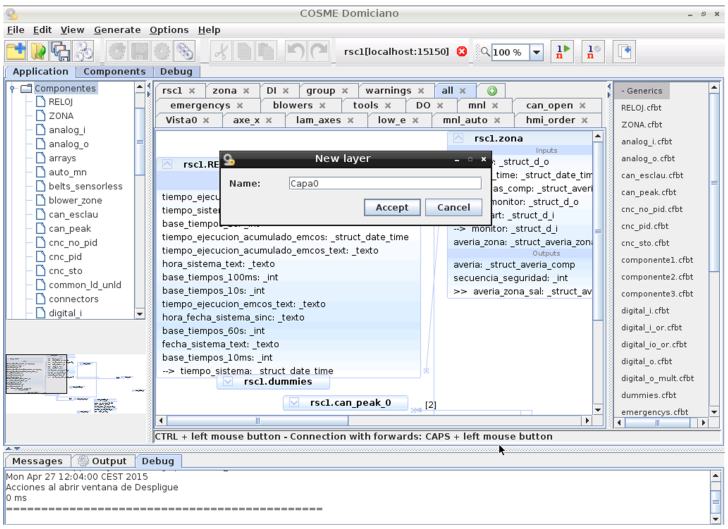


a) Definición FBN

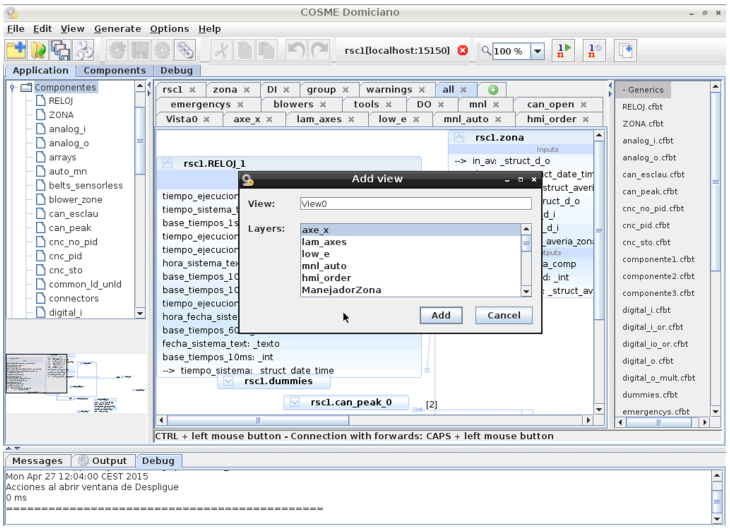


b) Diseño FBN

Figura 4.16: Diseño de la FBN (1/2)



a) Definición de una capa



b) Definición de una vista

Figura 4.17: Diseño de la FBN (2/2)

Si el tamaño de la FBN es grande (por número de FB y/o conexiones de datos) su diseño puede resultar difícil. Por este motivo, el editor gráfico facilita la representación de la FBN mediante capas y vistas, como se ha indicado anteriormente (ver Figs. 4.17a y b).

Una vez diseñada la FBN debe indicarse el orden de ejecución de los FB en los *daisy-chain* (ver Fig. 4.18a), tras lo cual únicamente queda generar la FBN que es almacenada en un fichero XML (ver Fig. 4.18b).

4.3.2.3. Despliegue y depuración de la FBN

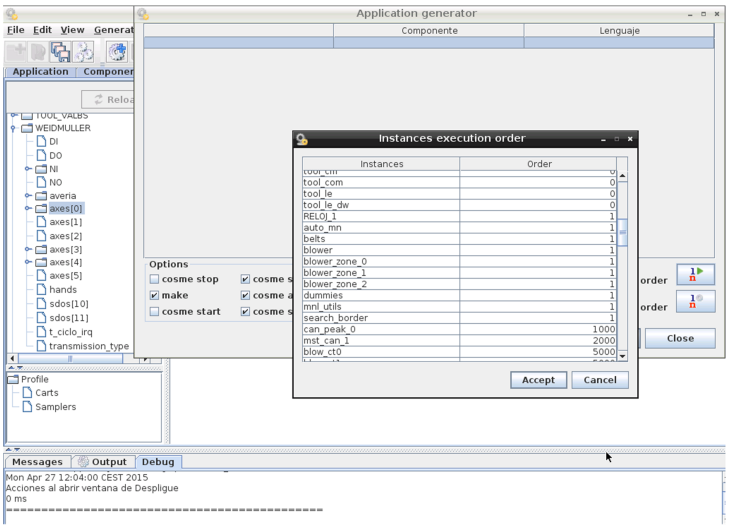
El siguiente paso en el desarrollo de la FBN consiste en desplegar la FBN en los controladores. Para ello, el diseñador debe indicar los datos de conexión (IP, puerto, login, etc.) con los controladores, que serán guardados en un perfil para sucesivas conexiones (ver Fig. 4.19a). El despliegue comienza con la transferencia de los ficheros con el código de los FB y el fichero XML de la FBN mediante uno de los dos protocolos disponibles: SAMBA o SCP. Una vez transferidos estos ficheros debe realizarse la compilación en cada controlador. Para ello, Domiciano invoca remotamente el compilador local.

Finalmente, con las FBN desplegadas en cada controlador la aplicación de control distribuido es puesta en ejecución mediante una petición al **Cargador de FBN**, que sigue la secuencia de pasos descrita en la Subsección 4.2.3.5.

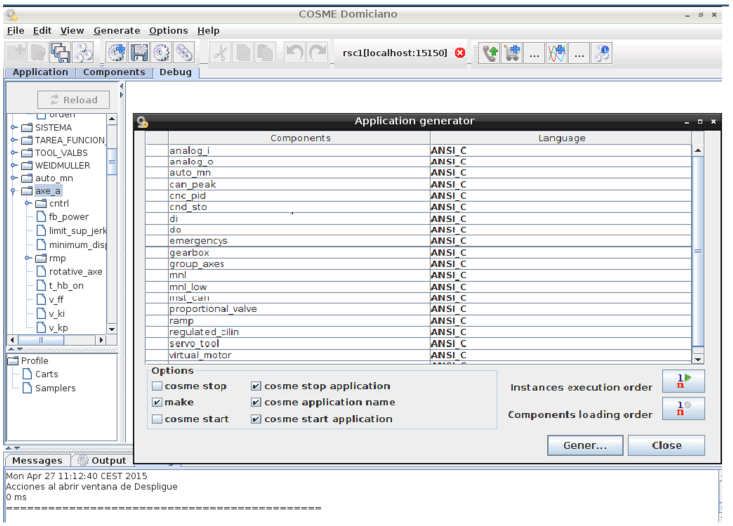
El último paso de depuración (ver Fig. 4.19b) se realiza observando de manera remota el estado de las variables internas, así como de los datos de entrada y salida mediante la **Pasarela** y el **Servidor de nombres**, como se ha descrito en las Subsecciones 4.2.3.3 y 4.2.3.4.

4.4. Caso de uso de la plataforma COSME

La plataforma COSME es utilizada actualmente por el grupo TUROMAS, fabricante internacional ubicado en Aragón de máquinas herramienta de corte, almacenamiento y manipulación de vidrio [137]. Dicha plataforma ha sido desarrollada dentro de un proyecto de colaboración tecnológica e investigación entre universidad y industria. El caso de uso que vamos a describir aquí, de manera breve, corresponde a una máquina herramienta para el corte de vidrio monolítico y laminado fabricada por el citado grupo (ver Fig. 4.20). Las características de una máquina de esta clase vienen dadas por los tipos de materiales a cortar, en este caso: vidrio monolítico y vidrio laminado. Se denomina vidrio monolítico al constituido por una única hoja de vidrio plano, que puede recibir diversos tratamientos térmicos que le dotan de mayor resistencia a tensiones mecánicas (flexión, choque, etc.) y térmicas. Por su parte, el segundo tipo está compuesto por dos hojas de vidrio monolítico separadas por una lámina de material plástico denominado butiral de polivinilo (*Polyvinyl butyral* o PVB). Dicha lámina impide el desprendimiento de fragmentos si se produce una rotura. El vidrio laminado se emplea por normativa de seguridad en edificaciones (p.e. ventanas, escaparates) para evitar daños en personas.

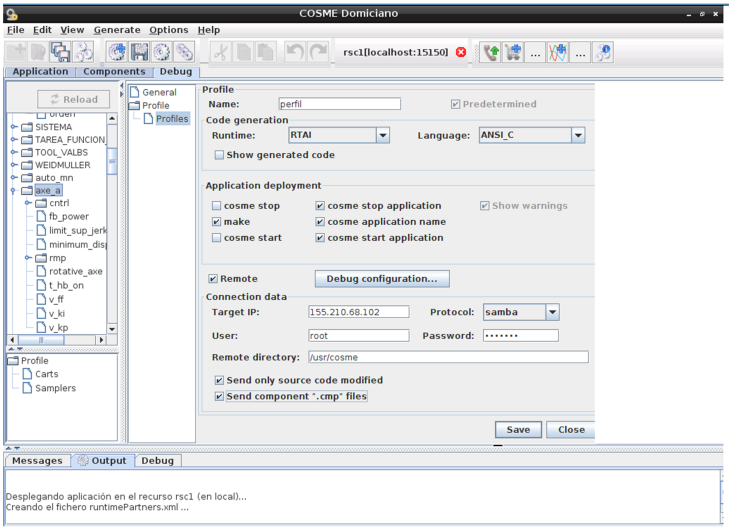


a) Definición orden de ejecución FB

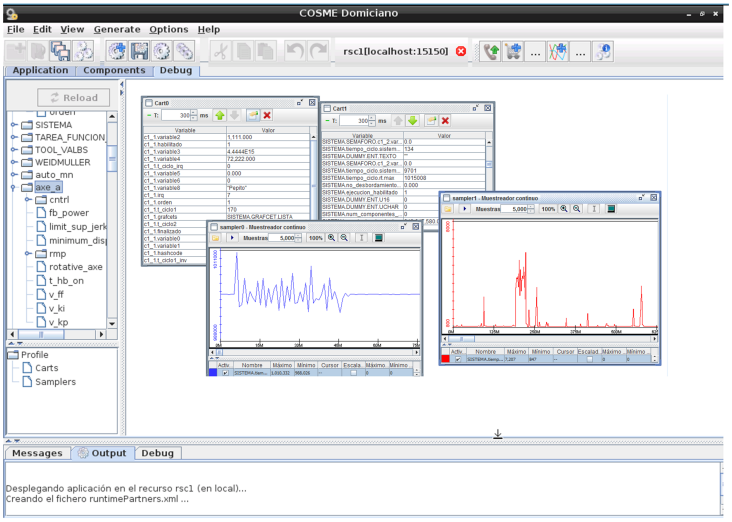


b) Generación de la FBN

Figura 4.18: Orden de ejecución de FB, generación, despliegue y depuración de la FBN (1/2)



a) Despliegue de la FBN



b) Depuración funcionamiento de la FBN

Figura 4.19: Orden de ejecución de FB, generación, despliegue y depuración de la FBN (2/2)

Figura 4.20: Máquina de corte de vidrio monolítico y laminado (Cortesía de TUROMAS)

El proceso de corte es distinto para cada tipo de vidrio. En el caso del vidrio monolítico se marcan las piezas a obtener en cada hoja mediante una herramienta, dotada de una punta de un material de elevada dureza, que va moviéndose de manera automática. Posteriormente, las piezas se obtienen mediante simples operaciones de manipulación por un operario. En el caso del vidrio laminado, además del marcado, cada línea de corte debe calentarse durante unos segundos para fundir el PVB. Debido a dicha operación, ahora es el vidrio el que se mueve automáticamente por la máquina, en vez de la herramienta de marcado, al estar la resistencia calefactora fija.

La máquina herramienta está compuesta por un cargador automático de hojas, una zona de corte para los dos tipos de vidrio indicados y una zona de descarga manual de piezas cortadas. La máquina tiene nueve ejes de posicionamiento a controlar, con una precisión menor de 1 mm y una aceleración de hasta 22 m/s^2 en el caso de la herramienta de marcado. Además de varias decenas de secuencias automáticas para movimientos de carga, así como para pre-posicionamientos de vidrio y herramientas. La Fig. 4.21 muestra los distintos *workflows* de materiales, piezas y procesos durante la operación de la máquina herramienta, así como las funcionalidades implementadas.

El desarrollo de la aplicación de control de máquina de corte ha sido realizado a partir de los siguientes requisitos generales:

- Control de tiempo real de movimientos y posicionamientos precisos.
- Los especificados por diversas normativas de seguridad de personas, nacio-

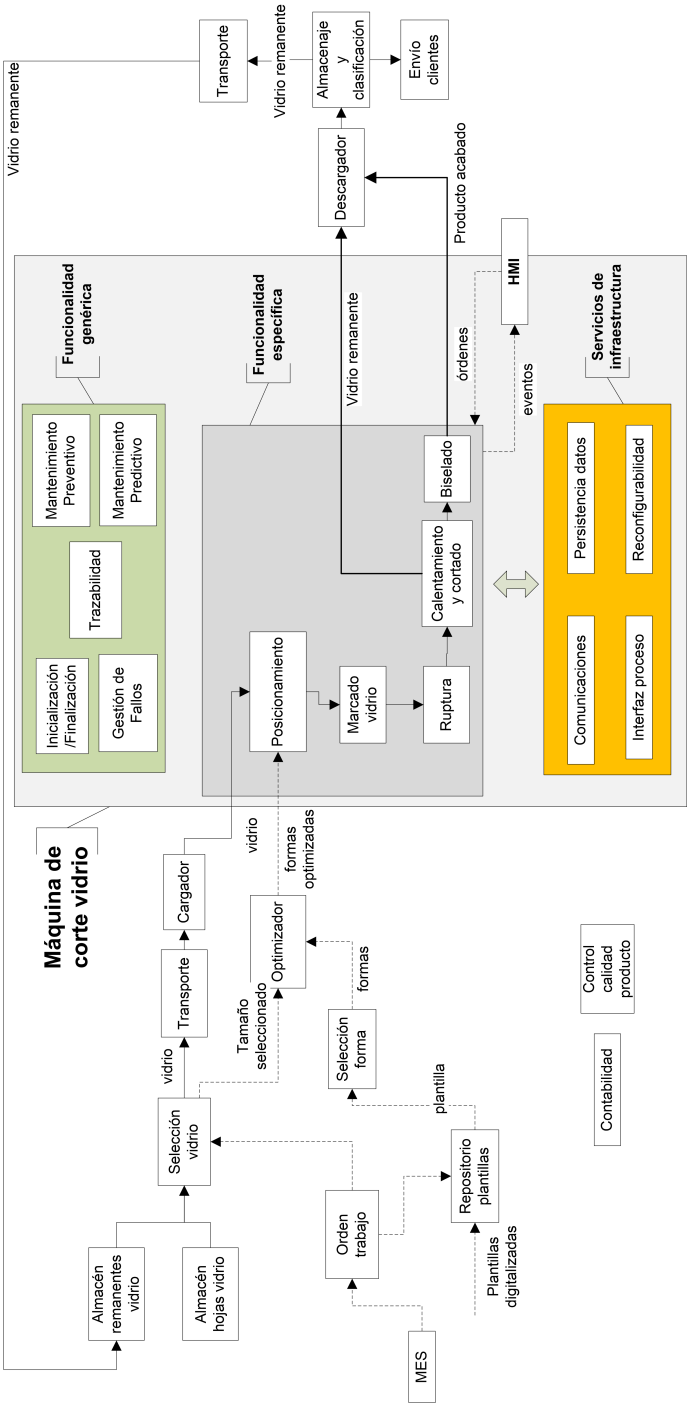


Figura 4.21: *Workflows* de materiales, piezas y procesos de la máquina de corte

nales e internacionales, debido a las operaciones efectuadas manualmente en la máquina (p.e. descarga de piezas cortadas).

- La operación mediante una aplicación externa HMI (ver Fig. 4.23).
- La integración con los sistemas de información de la factoría a través de una aplicación MES.

La aplicación de control, descrita con más detalle en el siguiente capítulo, está compuesta por distintos elementos arquitectónicos (ver Fig. 4.22) que implementan las funcionalidades específicas y genéricas de la máquina. Agrupadas según la propuesta presentada en la Sección 2.2 son las siguientes:

- **Cargador**
  - **Genéricas**
    - *Interfaz de proceso* mediante un controlador bus de campo CANopen, una isla de válvulas neumáticas y E/S digitales.
    - *Gestión de fallos* de una zona general.
    - *Gestión de mantenimiento preventivo y predictivo.*
  - **Específicas**
    - *Secuencias automáticas* para movimiento de hojas para la carga de la mesa de corte.
- **Mesa de corte**
  - **Genéricas**
    - *Interfaz de proceso* mediante un controlador bus de campo CANopen, una isla de válvulas neumáticas, E/S digitales y nueve *drives* para otros tantos servomotores *brushless.*
    - *Gestión de fallos* de zonas: general, laminado, monolítico, descarga y pulsadores de emergencias.
    - *Gestión de mantenimiento preventivo y predictivo.*
    - *Trazabilidad.*
  - **Específicas**
    - *Secuencias automáticas* para movimientos de entrada de hojas del cargador, herramienta de corte, hojas de laminado y descarga de piezas cortadas.
    - *Posicionamiento preciso* de herramientas de corte de vidrio monolítico y de hojas de vidrio laminado.

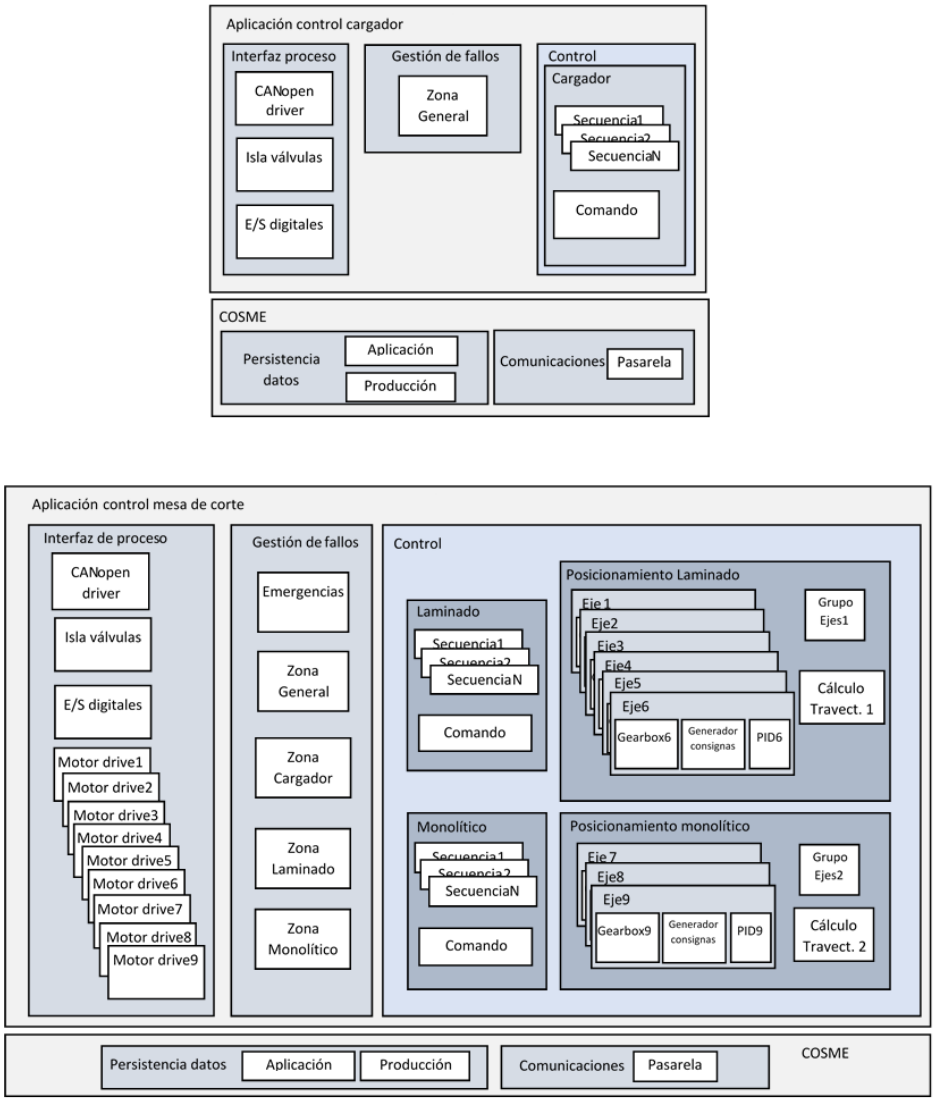


Figura 4.22: Elementos arquitectónicos de la máquina de corte agrupados jerárquicamente por funcionalidad

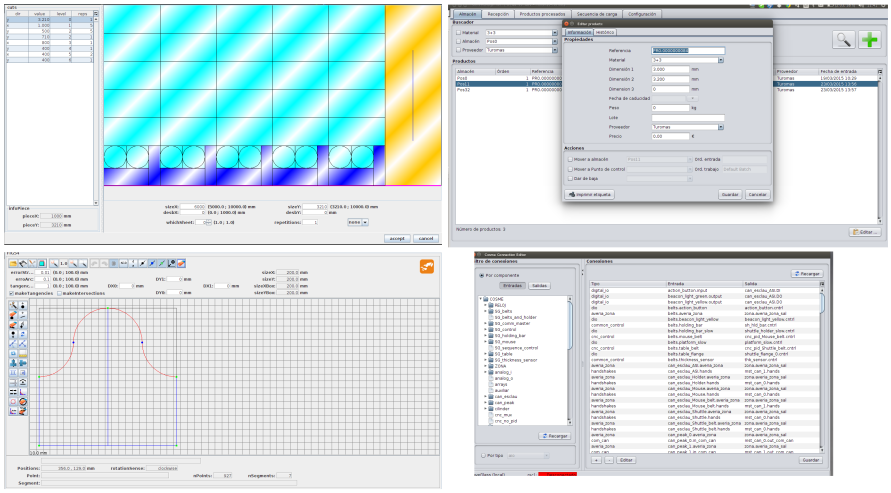


Figura 4.23: Interfaz de usuario de la máquina herramienta de corte de vidrio (Cortesía de TUROMAS)

4.5. Disponibilidad de la plataforma COSME

La plataforma COSME ha sido desarrollada a lo largo de varios años dentro de un proyecto de investigación, transferencia y colaboración tecnológica entre la Universidad de Zaragoza y el grupo TUROMAS. Este proyecto se ha materializado en forma de diversos contratos OTRI (Oficina de Transferencia de Resultados de Investigación). La participación de un fabricante de máquinas herramienta ha contribuido a evaluar la plataforma en casos de uso reales de control industrial, uno de los cuales ha sido descrito anteriormente.

La disponibilidad de COSME está sujeta a los acuerdos marcados por los indicados contratos. El autor, junto con otros miembros de su grupo de investigación, trabaja actualmente en una versión que pueda ponerse a disposición de la comunidad de IEC 61499.

4.6. Revisión de las plataformas más relevantes propuestas para IEC 61499

El número de plataformas propuestas hasta la fecha para IEC 61499 es pequeño (ver Subsección 1.4.1), y solamente cuatro de ellas son relevantes a día de hoy: FBDK, ISaGRAF, 4DIAC y nxtControl. La primera tiene importantes limitaciones, como el no poder ejecutar FBN con restricciones de tiempo real, por lo que pretende ser más un “demostrador” de IEC 61499 que una plataforma para el desarrollo de aplicaciones reales. De hecho, durante los primeros años de vida del estándar FBDK ha sido la plataforma de experimentación de referencia en muchos de los trabajos académicos relacionados con él.

Las limitaciones de uso de esta plataforma hacen que, a efectos de discusión y comparación con la plataforma COSME, sea de interés reseñar únicamente las otras tres plataformas restantes. Los modelos de ejecución de estas plataformas han sido ya anteriormente discutidos (ver Sección 3.1).

Los trabajos que han analizado las plataformas propuestas [2] [98] destacan su uso mayoritario únicamente en el ámbito académico de los laboratorios de investigación. Este hecho puede explicar porque los mismos trabajos indican también la falta de herramientas de desarrollo (p.e. *debugging*) que faciliten su utilización en casos de uso reales. Esta carencia es un grave inconveniente, ya que los ingenieros de control están habituados al empleo de plataformas comerciales de PLCs, muchas de ellas basadas en IEC 61131, que proporcionan estas herramientas. No obstante, debe indicarse que dos de las tres plataformas indicadas, ISaGRAF y nxtStudio, sí las incluyen. Aunque, este autor no ha encontrado casos de uso reales de control industrial con estas plataformas.

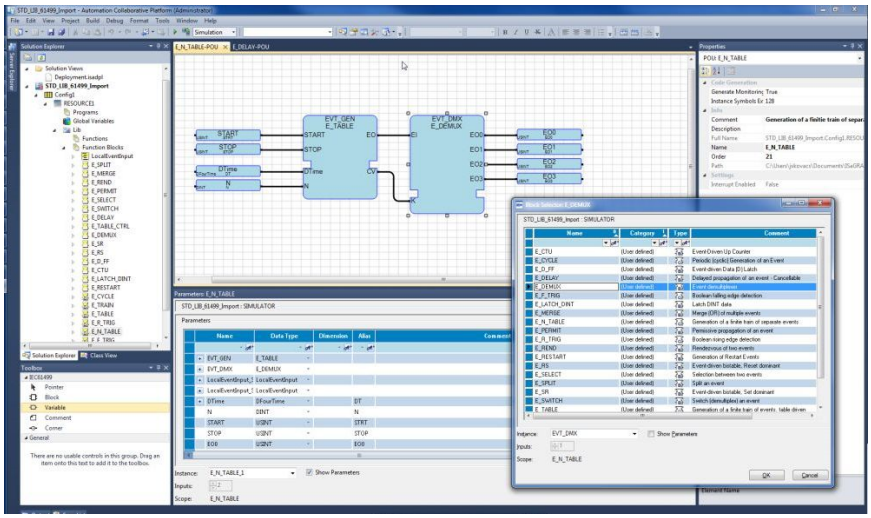


Figura 4.24: IDE ISaGRAF

4.6.1. Plataforma ISaGRAF

La primera plataforma comercial para IEC 61499 ha sido ISaGRAF [54] de ICS Triplex y pertenece a la categoría de *Soft-PLCs*. Denominada así porque implementa un PLC sobre un PC, a diferencia de las plataformas hardware y sistemas operativos propietarios empleados en los PLCs clásicos. Esta plataforma parte de una anterior para IEC 61131 y ha ido paulatinamente convergiendo al nuevo estándar. Su modelo de ejecución cíclico es compatible con la ejecución de aplicaciones IEC 61131 sobre el mismo *runtime* (ver Subsección 3.2.4). ISaGRAF representa una solución pragmática que tiene por objeto introducir gradualmente el nuevo estándar sobre la base del anterior.

Su IDE incluye soporte y herramientas para *debugging*, así como para el despliegue de las FBN. Varias aplicaciones de demostración son descritas en [89]. Por su parte, en [105] se presenta un prototipo del sistema de control de equipajes de un aeropuerto. Este autor no ha podido encontrar casos de uso con esta plataforma en aplicaciones reales de control industrial con IEC 61499.

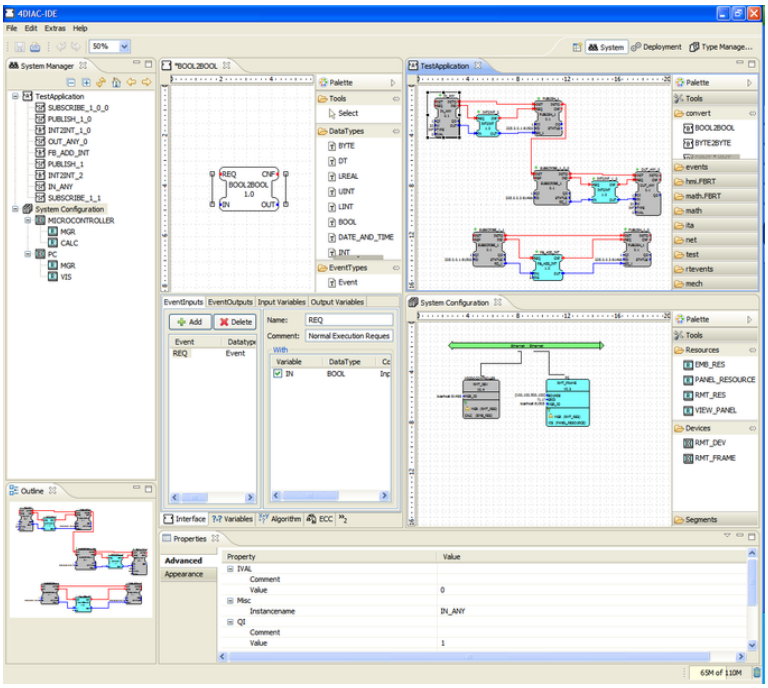


Figura 4.25: IDE 4DIAC

4.6.2. Plataforma 4DIAC

Desde la comunidad académica en unión con *partners* industriales ha surgido como iniciativa de software abierto la plataforma 4DIAC [51]. Consta de un IDE (ver Fig. 4.25) y un *runtime*, denominado FORTE, basado en el modelo de ejecución secuencial (ver Subsección 3.2.2.1) y desarrollado en C++. Ambos están disponibles para sistemas Windows y Linux. Las principales carencias de 4DIAC son la necesidad de mejoras en el IDE que reduzcan los tiempos de desarrollo. así como el paso del laboratorio a la industria [98] que haga posible evaluar su uso en aplicaciones reales.

Es destacable que los líderes de la iniciativa están haciendo un importante esfuerzo por crear una comunidad de desarrolladores y usuarios [52]. Así, 4DIAC parece haber sustituido el papel de FBDK como plataforma de experimentación para IEC 61499. Quizá por este motivo los casos de uso más importantes hasta la fecha [51] están circunscritos únicamente a laboratorios de investigación.

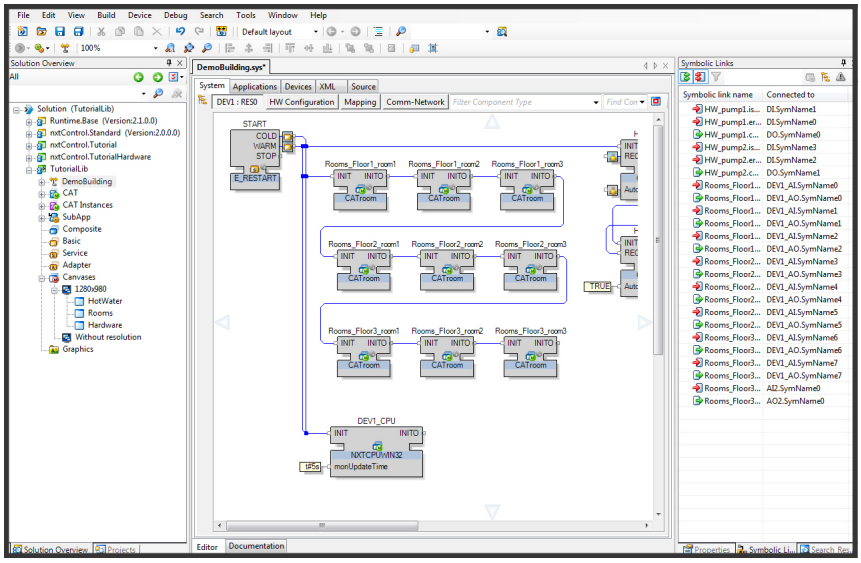


Figura 4.26: IDE nxtStudio

4.6.3. Plataforma nxtStudio

El fabricante de equipos para automatización nxtControl [56] es uno de los *partners* que participan en el desarrollo de 4DIAC y FORTE. Y de ellos ha surgido como producto comercial la plataforma compuesta por el IDE nxtStudio (ver Fig. 4.26) y su *runtime* nxtRT61499F. Su orientación profesional hace que incorpore un conjunto de biblioteca de bloques función y herramientas para *debugging*, así como para el despliegue de aplicaciones de control distribuido. Su IDE está disponible para sistemas Windows, mientras que su *runtime* lo está para sistemas operativos Windows Embedded, VxWorks o Linux. Los principales casos de uso pueden obtenerse en [56], la gran mayoría en el ámbito de la domótica de grandes edificios, lo que no permite evaluar su empleo en aplicaciones de control industrial.

4.7. Conclusiones

Las propuestas de modelos para IEC 61499 deben ir acompañadas de plataformas que permitan su evaluación experimental. La mayoría de plataformas propuestas hasta la fecha para el estándar han verificado su funcionamiento únicamente en prototipos o sencillas aplicaciones de control limitadas al ámbito académico. En realidad, estas plataformas parecen haber sido desarrolladas más bien como demostradores tecnológicos. Un paso más allá de la verificación es la validación, para lo que las propuestas, y el propio estándar, deben ser enfrentados a casos de uso reales y complejos, especialmente en dominios del control industrial.

Esta cuestión, añadida a los indicadas en capítulos anteriores, puede estar contribuyendo a la no aceptación de IEC 61499 hasta el momento por la industria.

Para el desarrollo de la plataforma COSME, dentro de un proyecto de colaboración tecnológica, transferencia e investigación, se ha tenido en mente tanto la verificación experimental de los modelos adaptados como su empleo por ingenieros de control en casos de uso reales. Con ello, se pretende contribuir al acercamiento de IEC 61499 a la industria, y por lo tanto a su aceptación. En este sentido, la plataforma propuesta es similar a la plataforma ISaGRAF, aunque con el hecho diferencial de los modelos adaptados.

Los requisitos fundamentales para su desarrollo han sido: facilidad de diseño, fiabilidad, rendimiento y escalabilidad. El cumplimiento de estos requisitos es posible gracias al uso de los citados modelos. La plataforma COSME está constituida por un *runtime* que implementa el modelo de ejecución adaptado propuesto en el anterior capítulo, y por el entorno de desarrollo Domiciano. Este último oculta la complejidad propia del estándar facilitando así el diseño de las FBN distribuidas. Por otro lado, también incorpora herramientas (p.e. de depuración) habituales en entornos comerciales de IEC 61131, una importante carencia de otros entornos de desarrollo de IEC 61499.

Se ha presentado un caso de uso que manifiesta la validez del empleo de la plataforma en aplicaciones de control industrial de máquinas herramienta. No obstante, queda presentar pruebas y resultados experimentales que permitan verificar el cumplimiento de los tres últimos requisitos indicados en el párrafo anterior. Estas pruebas son el objeto del siguiente capítulo.

Capítulo 5

Pruebas y resultados experimentales

5.1. Introducción

En capítulos anteriores se han indicado las ventajas del uso en IEC 61499 de los modelos adaptados propuestos en este trabajo, frente a los modelos anteriores del estándar. Especialmente, dada su complejidad y requisitos establecidos, para el control de máquinas herramienta en sistemas de fabricación ágil. Los modelos adaptados se han plasmado en la plataforma COSME, como medio para el desarrollo y ejecución de esta clase de aplicaciones de control. En este capítulo se presentan las pruebas, y resultados experimentales, realizadas con el objeto de comprobar el correcto funcionamiento de esta plataforma, así como el cumplimiento de los requisitos establecidos en la propuesta de modelo de ejecución adaptado (ver Sección 3.1). Relacionados con la ejecución de las FBN estos requisitos son los siguientes:

1. Comportamiento determinista.
2. Cumplimiento de las restricciones de tiempo real.
3. Escalabilidad.

Los dos primeros requisitos están fuertemente relacionados, como se ha indicado en la propuesta del modelo de ejecución adaptado (ver Sección 3.3). El motivo es la implementación de este modelo mediante FB pasivos y la ejecución de cada *daisy-chain* en una tarea de tiempo real. Esta implementación de IEC 61499 hace innecesario el empleo de colas de eventos, causa de posibles comportamientos indeterministas de las FBN como ha sido discutido anteriormente.

En el caso del modelo adaptado de ejecución el comportamiento determinista de las FBN depende únicamente de una planificación de las tareas, antes indicadas, que garantice el cumplimiento de sus *deadlines*, o lo que es igual, de las restricciones de tiempo real de los *daisy-chain*.

Dicho cumplimiento va a ser determinado, mediante las pruebas realizadas, para los *daisy-chain* periódicos `EXT_EVENT`, `NORMAL_RT` y `NORMAL_NRT`. Los dos primeros tienen restricciones *hard real-time*, por lo que los tiempos de ejecución empleados para esta determinación corresponderán a los WCETs medidos experimentalmente. Respecto de `NORMAL_NRT`, éste se ejecuta de manera distribuida con restricciones *soft real-time* (ver Subsección 2.2.4). La definición de *soft real-time* corresponde a un tipo de ejecución en el cual una tarea cumple su *deadline* la mayor parte de las ocasiones. El número de incumplimientos por encima del cual el funcionamiento del sistema se considera incorrecto depende de la clase de aplicación. En este caso, consideraremos estas restricciones como 95 % *real-time*. Es decir, la ejecución distribuida de `NORMAL_NRT` cumplirá las restricciones de tiempo real si el *deadline* se cumple el 95 % de las veces. Por ello, se tomará como WCET el valor máximo después de despreciar el 5 % de los tiempos de ejecución más altos medidos experimentalmente.

Respecto del tercer requisito, la escalabilidad de las FBN es considerada aquí en relación con el tamaño de las subFBN en cada controlador (ver Subsección 4.1). En particular, respecto del número de FB, ya que los modelos adaptados limitan la cantidad y el tipo de cadenas de eventos (*daisy-chains*).

Las pruebas y resultados experimentales presentados en este capítulo han sido realizados con las dos implementaciones de la plataforma COSME (ver Subsección 4.2.3), ejecutadas en plataformas de pruebas distintas. Estas pruebas han consistido en la medida de tiempos de ejecución en dos escenarios distintos:

1. FBN sintéticas ejecutadas con la implementación de la plataforma COSME POSIX-PREEMPT_RT en un sencillo computador SoC (*System on Chip*).

2. FBN para el control distribuido de una mesa de corte de vidrio laminado y monolítico ejecutada con la implementación COSME RTAI en un computador industrial.

El uso de programas sintéticos (i.e. con cargas de trabajo "artificiales") es habitual en el análisis de prestaciones de arquitecturas y plataformas software. Facilitando su empleo la realización de las pruebas y, si están estandarizados, la comparación de prestaciones. En el caso de IEC 61499 en [138] se han propuesto diferentes clases de FBN sintéticas con este fin. Debido al modelo de ejecución del estándar que asume dicho trabajo estas FBN no son aplicables al modelo de ejecución adaptado, por lo que en este caso se ha diseñado una FBN sintética específica. Esta FBN permite la realización de pruebas con redes de distinto tamaño, con el objeto de medir el rendimiento y determinar la escalabilidad de la plataforma COSME.

De cualquier manera, la mejor medida de las prestaciones y del rendimiento de una plataforma viene dada por su utilización en aplicaciones reales. En el capítulo anterior se ha presentado un caso de uso real con la plataforma COSME: el control distribuido de una mesa de corte de vidrio monolítico y laminado fabricada por la empresa TUROMAS (ver Sección 5.1). En este capítulo se presentan los resultados experimentales de las pruebas realizadas con esta máquina herramienta.

5.1.1. Medida de tiempos de ejecución en la plataforma COSME

El empleo de herramientas software, denominadas *profilers*, es frecuente en el análisis de prestaciones de un programa o aplicación. Estas herramientas determinan el número de veces que se ejecuta el código, así como sus tiempos y flujos de ejecución (*call graphs*). Para realizar estas funciones los *profilers* necesitan incorporar código específico al propio analizado, lo que afecta al error de medida de los tiempos de ejecución.

Otro método de medida de tiempos de ejecución consiste en utilizar una salida digital de algún dispositivo de E/S, puesta a nivel alto al comienzo del código analizado y a nivel bajo al final. El tiempo de ejecución puede ser medido conectando esa salida a un analizador lógico u osciloscopio. Este método es más preciso que el anterior, aunque requiere considerar los tiempos de establecimiento de la salida digital, dificulta las pruebas y los dispositivos indicados no siempre están disponibles.

Finalmente, el tercer método para medir tiempos de ejecución es utilizar un reloj suministrado por el sistema operativo. En el caso de sistemas de tipo Unix el acceso a dicho reloj lo presta la función:

```
int clock_gettime(clockid_t clk_id, struct timespec *tp)
```

Esta función devuelve el tiempo de un reloj especificado del sistema. El reloj usado habitualmente para el propósito requerido es `CLOCK_MONOTONIC_RAW`, un contador monótono creciente no afectado por ajustes ni cambios de hora en el sistema y que tiene una resolución de nanosegundos.

Este tercer método va a ser el empleado para la medida de tiempos de ejecución en el código de la plataforma COSME implementado en lenguaje C. La diferencia entre el valor del reloj al comienzo y al final de la ejecución del código analizado corresponde al tiempo de ejecución. Los resultados serán almacenados en variables, que serán accedidas a través de la **Pasarela** mediante la herramienta de *debug* del IDE Domiciano (ver Sección 4.3.1.5).

El elemento **Pasarela** de la plataforma está implementado parcialmente en Java. Para medir tiempos de ejecución en este lenguaje, al igual que antes, se accederá al valor de un reloj del sistema. En este caso, mediante el método `nanoTime()` de la clase `System` del paquete estándar `lang` de Java. Dicho método devuelve un valor con una resolución de nanosegundos, aunque la precisión depende del reloj del sistema subyacente y puede ser menor.

5.2. Pruebas con una FBN sintética

Se ha diseñado una FBN sintética desplegada sobre tres controladores (ver Fig. 5.1) para medir los tiempos de ejecución de las tareas asociadas a los *daisy-chain* periódicos. En cada uno de los controladores el *daisy-chain* `EXT_EVENT` está formado por un FB que simula la atención a los eventos externos generados, a través del SIFB `FB_IO`, por un supuesto bus de campo. Mientras que, `NORMAL_RT` y `NORMAL_NRT` están formados en cada prueba por un número diferente de FB, simulando distinto número de secuencias automáticas y controles de ejes.

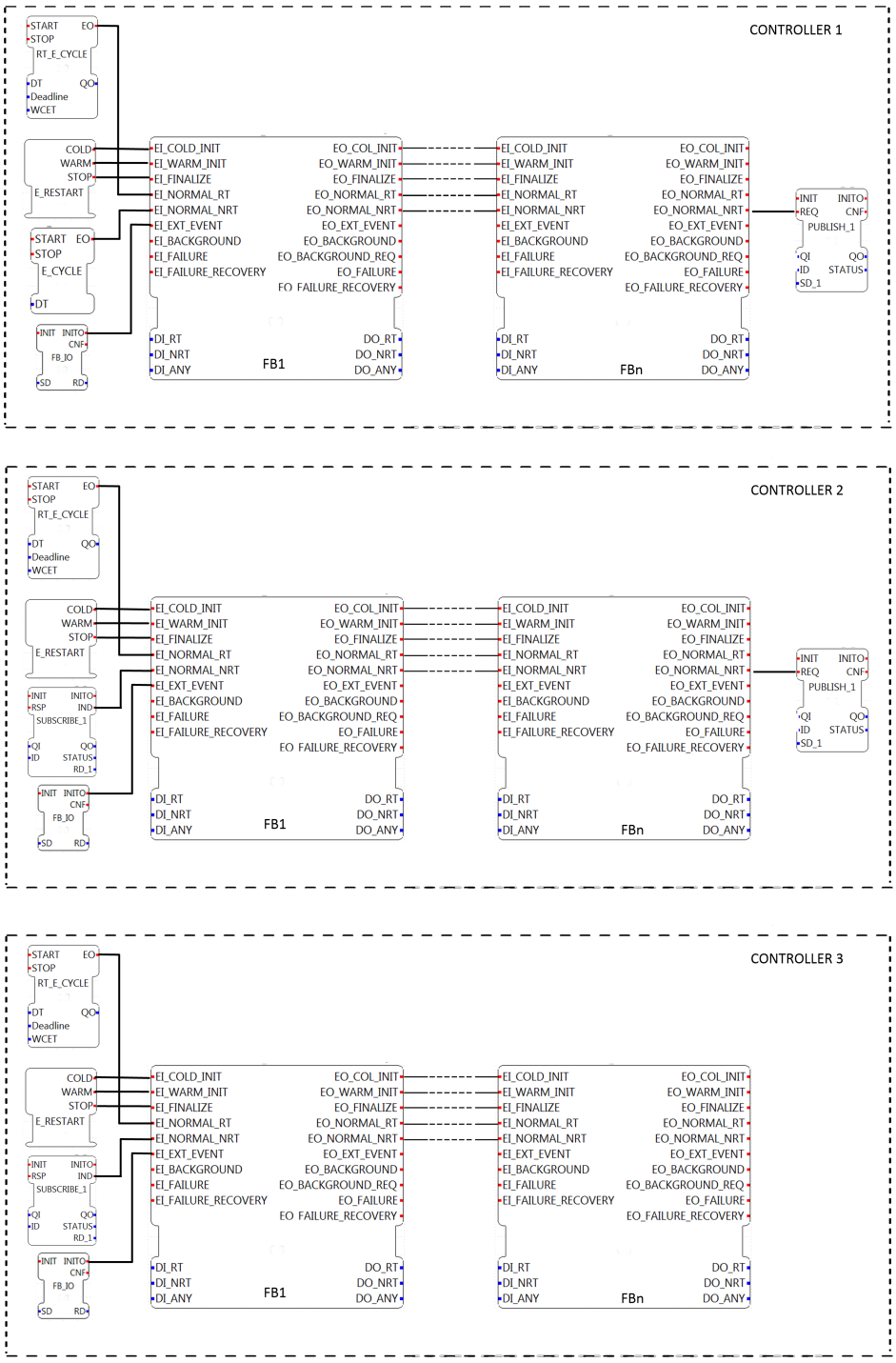


Figura 5.1: FBN sintética desplegada sobre tres controladores

Los *daisy-chain* son ejecutados mediante eventos generados por distintos SIFB. En concreto, `COLD_INIT`, `WARM_INIT` y `FINALIZE` a partir de eventos del SIFB `E_RESTART`, mientras que `NORMAL_RT` es ejecutado por eventos periódicos deL SIFB `RT_E_CYCLE`. Finalmente, `NORMAL_NRT` está distribuida en los tres controladores y su ejecución periódica comienza en el primero de ellos mediante el SIFB `E_CYCLE`, continuando en el resto de controladores mediante SIFB `PUBLISH` y `SUBSCRIBE`.

Se han establecido distintas cargas de trabajo sintéticas en los FB, con el objeto de ejecutar la FBN en cada prueba con diferentes niveles de exigencia. En particular:

1. En vacío: Las funciones no realizan ninguna acción con el fin de conocer el *overhead* de ejecución de los *daisy-chains*.
2. Cálculos con variables internas de los FB de tipo `long`.
3. Cálculos con vectores de 5 y 50 K de variables internas de los FB de tipo `long`.
4. Cálculos con variables internas de los FB de tipo `double`.
5. Cálculos con vectores de 5 y 50 K de variables internas de los FB de tipo `double`.

Estas cargas pretenden simular el empleo de los recursos de la plataforma hardware, así como del sistema operativo. Por ejemplo, el cálculo con variables enteras correspondería a la ejecución de secuencias automáticas y el cálculo con coma flotante a los controles de ejes. Por su parte, el diferente tamaño de los vectores pretende comprobar la influencia del sistema de memoria de la plataforma hardware (p.e. caches).

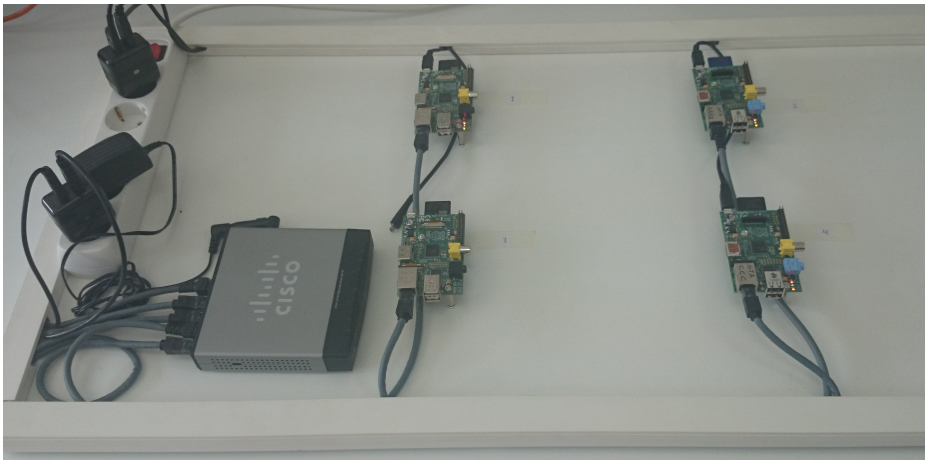


Figura 5.2: Plataforma de pruebas FBN sintética

5.2.1. Plataforma de pruebas

La plataforma de pruebas (*testbed*) consta de cuatro controladores, de los que se utilizarán únicamente tres en la prueba (ver Fig. 5.2). Los controladores están implementados mediante un sencillo computador basado en un SoC (*System on Chip*) denominado Raspberry Pi, dotado de un sistema operativo de tiempo real. Las conexiones inter-controlador son realizadas mediante una red dedicada cableada establecida con un dispositivo *Switched Ethernet*. Por otro lado, los computadores están también conectados a una red inalámbrica de propósito general para la comunicación con la herramienta de *debug* de Domiciano, con el fin de acceder a los resultados de las mediciones. Los datos detallados de la plataforma de pruebas son los siguientes:

- *Denominación computador*: Raspberry Pi modelo B con SoC Broadcom BCM2835
- *CPU*: Un *core* ARM 1176JZF-S a 700 MHz
- *Memoria*: 256 MB
- *Sistema operativo*: Raspbian 3.8.13 PREEMPT_RT armv6l
- *Java*: JVM HotSpot 1.7.0
- *Implementación COSME POSIX-PREEMPT_RT*
- *Switched Ethernet*: CISCO SLM2008 1Gbps

5.2.2. Resultados de la ejecución de *daisy-chain* intra-controlador

El **escenario 1** de la pruebas consiste en la ejecución intra-controlador de los *daisy-chains* `EXT_EVENT` y `NORMAL_RT` con periodos fijos de 2 y 15 ms respectivamente. Por su parte, `NORMAL_NRT` se ejecuta con dos periodos diferentes de 50 y 100 ms. El *daisy-chain* `EXT_EVENT` tiene una longitud de 1 FB y su carga de trabajo consiste en cálculos con números enteros en variables internas simples del FB. Mientras que, `NORMAL_RT` tiene una longitud de 10 a 100 FB en cada prueba y su carga de trabajo consiste en cálculos con números enteros en variables internas simples y en vectores de longitud 5K y 50K.

Con el fin de limitar la cantidad de casos a probar se ha considerado que la longitud de `NORMAL_NRT` en cada prueba dependa de la longitud de `NORMAL_RT`. En concreto, la longitud del primero será el 20 % de la longitud del segundo. Dicho valor es una relación estimada entre el número de FB de los dos *daisy-chain*, basada en los casos de uso caracterizados en el capítulo 2 de este trabajo. Respecto de la carga de trabajo de `NORMAL_NRT`, ésta consiste en cálculos con números enteros en variables internas simples.

Los tiempos de ejecución de este *daisy-chain* pueden estar influidos por la ejecución de `NORMAL_RT`, que tiene mayor prioridad. Por este motivo, y para determinar en mejor medida esta influencia, a la hora de obtener los WCETs de `NORMAL_NRT` se han contemplado las dos mayores cargas de trabajo de `NORMAL_RT`.

Los tiempos de ejecución medidos en los tres *daisy-chain* corresponden a los WCETs obtenidos con la herramienta de *debug* de Domiciano y se representan en las Tablas 5.1, 5.2 y 5.3. El número de valores muestreados puede calcularse a partir del tiempo de duración de la prueba T_p y el periodo de los *daisy-chain* T_i:

$$muestras_i = T_p/T_i \tag{5.1}$$

Para una duración de cada prueba de 30 min el número aproximado de valores muestreados en cada *daisy-chain* es:

$$muestras_{EE} = 30 * 60/(2 * 10^{-3}) = 900000$$

$$muestras_{RT} = 30 * 60/(15 * 10^{-3}) = 120000$$

$$muestras_{NRT} = 30 * 60/(50 * 10^{-3}) = 36000$$

Número FB	**1**
overhead	82
`long`	134

Tabla 5.1: Tiempos de ejecución `EXT_EVENT`, escenario 1 (valores en μs)

Número FB	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
overhead	334	427	534	639	727	875	977	1151	1293	1429
`long`	365	469	658	718	939	1161	1293	1476	1664	1886
`long[5K]`	491	572	746	958	1137	1322	1584	1899	2227	2558
`long[50K]`	627	809	1155	1498	1829	2220	2938	3427	3881	4301

Tabla 5.2: Tiempos de ejecución `NORMAL_RT`, escenario 1 (valores en μs)

Número FB	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
`RTlong[5K]` 50ms	1410	1356	1202	1199	1252	1325	1669	2035	2360	2610
`RTlong[5K]` 100ms	723	1021	984	1089	1286	875	921	916	875	1056
`RTlong[50K]` 50ms	889	1113	1391	1749	2247	2516	2826	3231	3597	4255
`RTlong[50K]` 100ms	676	961	753	587	761	913	738	641	953	994

Tabla 5.3: Tiempos de ejecución `NORMAL_NRT`, escenario 1 (valores en μs)

El **escenario 2** de la prueba es igual que el escenario anterior para la ejecución intra-controlador, longitudes y periodos de los *daisy-chain*, así como para las cargas de trabajo de `EXT_EVENT` y `NORMAL_NRT`. La diferencia en este escenario es la carga

Número FB	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
overhead	334	427	534	639	727	875	977	1151	1293	1429
`double`	658	886	1252	1594	2059	2430	2940	3382	3772	4161
`double[5K]`	901	1184	2018	2525	3096	3767	4586	5169	6270	6545
`double[50K]`	1085	1627	2370	3144	4105	4924	5468	6097	6846	7594

Tabla 5.4: Tiempos de ejecución `NORMAL_RT`, escenario 2 (valores en μs)

Número FB	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
`RTdouble[5K]` 50ms	776	1534	2020	2596	3462	4113	5289	5872	6273	7090
`RTdouble[5K]` 100ms	918	809	879	865	1417	3989	4313	4890	5848	6367
`RTdouble[50K]` 50ms	927	2157	2743	3436	4137	5248	6121	7117	7857	8369
`RTdouble[50K]` 100ms	1177	1993	2585	3268	4079	4979	5956	6875	7620	8122

Tabla 5.5: Tiempos de ejecución `NORMAL_NRT`, escenario 2 (valores en μs)

de `NORMAL_RT`, consistente ahora en cálculos con números en coma flotante. Por su parte, para obtener los WCETs de `NORMAL_NRT` se han contemplado, al igual que antes, las dos mayores cargas de trabajo de `NORMAL_RT` en este escenario. Los WCETs obtenidos se representan en las Tablas 5.4 y 5.5. Los resultados obtenidos serán discutidos más adelante.

5.2.3. Resultados de la ejecución de *daisy-chain* inter-controlador

Se obtendrán ahora los tiempos de ejecución y los retardos de comunicación relacionados con la ejecución distribuida de los *daisy-chain*. En el caso particular de la FBN sintética, `NORMAL_NRT` es el único de ellos que tiene conexiones inter-controlador, por lo que será el objeto de las pruebas realizadas.

La **Pasarela** es el elemento de la plataforma encargado de la comunicación inter-controlador, como se indicó en la Subsección 4.2.3.4 este elemento se ha implementado parcialmente en Java. Para determinar sus tiempos de ejecución debemos considerar el rendimiento de la máquina virtual de Java JVM HotSpot, que depende fundamentalmente del *Garbage collector* y del JIT (*just-in-time compiler*). El primero es un proceso automático de la máquina virtual encargado de la liberación de memoria mediante la eliminación de objetos no referenciados. Por su parte, el JIT está relacionado con la manera en que la máquina virtual ejecuta el código binario Java. Éste es interpretado cada vez a código nativo de la plataforma hardware subyacente, salvo cuando una misma parte (p.e. un método de un objeto) es ejecutada un número de veces determinado, momento en que es compilado *just in time* por el JIT. Este tipo de compilación disminuye, aún considerando sus *overheads*, los tiempos de ejecución de los programas escritos en Java. El umbral a partir del cual actúa el JIT se denomina *CompileThreshold*.

Las pruebas realizadas pretenden determinar la configuración de los dos elementos indicados de la JVM HotSpot con menores *overheads*, al ejecutar la **Pasarela** en las conexiones inter-controlador de `NORMAL_NRT`. El resto de opciones de configuración de la máquina virtual de Java son las establecidas por defecto. En concreto, se miden los tiempos de ejecución con las siguientes configuraciones:

1. Configuración por defecto.
2. *Garbage collector* `ConcMarkSweepGC`, `CompileThreshold` = 1, 100, 1000 y 10000.
3. *Garbage collector* `ParallelGC`, `CompileThreshold` = 1, 100, 1000 y 10000.
4. *Garbage collector* `SerialGC`, `CompileThreshold` = 1, 100, 1000 y 10000.
5. *Garbage collector* `G1GC`, `CompileThreshold` = 1, 100, 1000 y 10000.

En las Tablas 5.6, 5.7, 5.8, 5.9 y 5.10 se indican los valores medios, la desviación estándar, así como los WCETs del 95 % y del 100 % de las muestras para las configuraciones indicadas y un periodo de `NORMAL_NRT` de 50 ms. Al igual que antes, el número de valores muestreados puede calcularse a partir del tiempo de duración de la prueba y la ecuación 5.1. Así, con una duración de 30 minutos tenemos un valor aproximado de:

$$muestras_{NRT} = 30 * 60/(50 * 10^{-3}) = 36000$$

Configuración	**Media**	**Desv.Std**	**95 %**	**100 %**
Por defecto	7,76	12,27	19,18	725,19

Tabla 5.6: Tiempos de ejecución Pasarela Configuración 1 (valores en ms)

Configuración	**Media**	**Desv.Std**	**95 %**	**100 %**
`ConcMarkSweepGC`, `CompileThreshold = 1`	15,11	23,84	49,04	659,96
`ConcMarkSweepGC`, `CompileThreshold = 100`	8,49	8,24	20,34	337,81
`ConcMarkSweepGC`, `CompileThreshold = 1000`	7,13	2,84	11,32	54,52
`ConcMarkSweepGC`, `CompileThreshold = 10000`	9,40	4,37	14,26	188,13

Tabla 5.7: Tiempos de ejecución Pasarela Configuración 2 (valores en ms)

Configuración	**Media**	**Desv.Std**	**95 %**	**100 %**
`ParallelGC`, `CompileThreshold = 1`	7,73	5,04	13,35	188,01
`ParallelGC`, `CompileThreshold = 100`	7,93	5,04	14,84	189,25
`ParallelGC`, `CompileThreshold = 1000`	8,50	9,22	16,73	189,25
`ParallelGC`, `CompileThreshold = 10000`	8,09	5,00	14,94	184,67

Tabla 5.8: Tiempos de ejecución Pasarela Configuración 3 (valores en ms)

Configuración	Media	Desv.Std	95 %	100 %
`SerialGC, CompileThreshold = 1`	8,41	9,38	16,31	313,47
`SerialGC, CompileThreshold = 100`	7,31	6,50	13,47	337,97
`SerialGC, CompileThreshold = 1000`	9,61	12,74	19,91	639,29
`SerialGC, CompileThreshold = 10000`	9,92	5,79	17,71	199,85

Tabla 5.9: Tiempos de ejecución Pasarela Configuración 4 (valores en ms)

Configuración	Media	Desv.Std	95 %	100 %
`G1GC, CompileThreshold = 1`	8,44	9,02	15,55	294,09
`G1GC, CompileThreshold = 100`	7,35	4,41	13,60	180,71
`G1GC, CompileThreshold = 1000`	10,35	11,28	25,09	200,91
`G1GC, CompileThreshold = 10000`	18,79	13,82	41,62	228,30

Tabla 5.10: Tiempos de ejecución Pasarela Configuración 5 (valores en ms)

En las conexiones inter-controlador se utiliza un dispositivo *Switched Ethernet* para la comunicación en red de los controladores. Bajo el concepto de *Quality of Service* (QoS) en estos dispositivos puede configurarse su funcionamiento de dos modos: a) establecimiento de prioridades en los mensajes y b) garantía de ancho de banda. Para el cumplimiento de las restricciones *soft real-time* en `NORMAL_NRT` consideraremos el primer modo, ya que el segundo está más indicado para comunicaciones de audio o vídeo.

En particular, en el dispositivo CISCO SLM2008 de la plataforma COSME se ha configurado QoS en el modo *Port-based priority*, donde a cada uno de sus ocho puertos se le asigna una prioridad: *low*, *normal*, *medium* o *high*. En este caso todos los puertos han sido configurados con prioridad *high*. Esta configuración tiene dos modos distintos relacionados con el funcionamiento de las colas de mensajes: *Strict Priority* y *Weighted Round-Robin* (WRR), ambos son contemplados en las pruebas efectuadas.

Para medir los retardos de comunicación se ha modificado la FBN sintética (ver Fig. 5.1) de manera que el último FB del primer controlador tiene una dato de entrada generado por el primer FB del segundo controlador en una conexión "aguas abajo" (ver Fig 2.27). El retardo es obtenido como resultado del tiempo transcurrido desde el envío del mensaje con el evento de salida hasta la recepción del mensaje con el dato de entrada en el primer controlador, descontando los tiempos de ejecución de la **Pasarela** y la ejecución intra-controlador de `NORMAL_NRT` en el segundo controlador. Este retardo es el tiempo de ida y vuelta por lo que el retardo buscado es la mitad de dicho valor.

Los valores medios, la desviación estándar, así como los WCETs del 95 % y del 100 % de las muestras para las configuraciones indicadas se indican en la Tabla 5.11. Con una duración de la prueba también de 30 minutos el número de muestras es similar al indicado en la prueba de ejecución con la **Pasarela**. Estos resultados son discutidos a continuación.

Configuración	Media	Desv.Std	95 %	100 %
QoS desactivada	1,11	1,61	2,47	29,18
QoS activada y *Strict Priority*	1,07	3,48	2,36	25,17
QoS activada y WRR	0,97	1,10	1,73	17,07

Tabla 5.11: Retardos de comunicación en el *Switched Ethernet* para `NORMAL_NRT` (valores en ms)

5.2.4. Discusión de los resultados

5.2.4.1. Cumplimiento de las restricciones de tiempo real

A partir de los resultados obtenidos podemos determinar si se cumplen las restricciones de tiempo real en la FBN. Como se ha indicado en la introducción a este capítulo, en el modelo adaptado de ejecución propuesto este cumplimiento garantiza un comportamiento determinista de la FBN.

En primer lugar, consideramos la ejecución periódica de `EXT_EVENT`, `NORMAL_RT` y `NORMAL_NRT` (este último en el caso intra-controlador); comprobando el cumplimiento de sus *deadlines* mediante la ecuación de Liu y Layand (ver Subsección 3.4.5). En el caso más desfavorable de las pruebas efectuadas, y para periodos de cada *daisy-chain* de 2, 15 y 50 ms respectivamente, tenemos WCETs de 0,134 ms, 7,594 ms y 8,369 ms, obteniendo un factor de utilización de:

$$U = \frac{0{,}134}{2} + \frac{7{,}594}{15} + \frac{8{,}369}{50} = 0{,}741$$

El resultado de U obtenido es menor que el límite de 0,78 que marca la citada ecuación, por lo que se cumplen las restricciones de tiempo real de la FBN sintética en el caso más desfavorable de los considerados, aunque con un margen pequeño. Dicho cumplimiento está condicionado al hecho de haber medido realmente los WCETs, algo que únicamente podría garantizarse con una duración infinita de los experimentos. En las pruebas aquí realizadas esta duración es un compromiso entre la obtención de un número de muestras significativo y su realización práctica. Por este motivo, en el caso de una FBN real deben contemplarse márgenes mayores. Considerando un diseño óptimo en cuanto al número y carga de trabajo de los FB, una solución es modificar el periodo de ejecución de `NORMAL_NRT`, ya que este *daisy-chain* permite periodos de ejecución más grandes (ver Sección 2.4.1). Por ejemplo, si modificamos su periodo a 100 ms tenemos un margen mayor, en concreto:

$$U = \frac{0{,}134}{2} + \frac{7{,}594}{15} + \frac{8{,}369}{100} = 0{,}657$$

En segundo lugar, consideramos la ejecución distribuida de `NORMAL_NRT`. Debido a sus restricciones *soft real-time* contemplamos los WCETs del 95 % en la conexión inter-controlador, asumiendo las configuraciones más favorables de JVM HotSpot y del *Switched Ethernet*. En el primero esta configuración corresponde a un *Gargage Collector* `ConcMarkSweepGC` y un JIT con `CompileThreshold = 1000`. Mientras que en el segundo, la configuración más favorable es con QoS activada y colas de mensajes en WRR.

Considerando la misma carga de trabajo en los tres controladores y los mismos retardos en las dos conexiones inter-controlador, el WCET total de `NORMAL_NRT` en la FBN sintética puede calcularse, de manera aproximada, mediante la ecuación:

$$WCET_{NRT} = (WCET_C + WCET_{PAS}) * 3 + WCET_{SE} * 2$$

Donde $WCET_C$ corresponde a la ejecución del *daisy-chain* en un controlador, $WCET_{PAS}$ a la ejecución de la **Pasarela** y $WCET_{SE}$ al retardo de comunicación en el *Switched Ethernet*. En el caso de estos dos últimos con las configuraciones indicadas tenemos $WCET_{PAS}$ = 11,32 ms y $WCET_{SE}$ = 1,73 ms. Por su parte, para $WCET_C$ asumimos los tiempos de ejecución correspondientes al mayor número de FB y carga de trabajo de `NORMAL_RT`, en los dos escenarios y periodos contemplados.

Con un periodo de 50 ms y una carga de `NORMAL_RT` de cálculos con vectores de enteros de 50K tenemos:

$$WCET_{NRT} = (4{,}255 + 11{,}32) * 3 + 1{,}73 * 2 = 50{,}185\ ms$$

Para la misma carga de trabajo con un periodo de 100 ms:

$$WCET_{NRT} = (0{,}994 + 11{,}32) * 3 + 1{,}73 * 2 = 40{,}402\ ms$$

Con un periodo de 50 ms y una carga de `NORMAL_RT` de cálculos en coma flotante con vectores de 50K tenemos:

$$WCET_{NRT} = (8{,}369 + 11{,}32) * 3 + 1{,}73 * 2 = 62{,}527\ ms$$

Para la misma carga de trabajo con un periodo de 100 ms:

$$WCET_{NRT} = (8{,}122 + 11{,}32) * 3 + 1{,}73 * 2 = 61{,}786\ ms$$

Comprobamos que para las dos cargas de trabajo con el periodo de 50 ms no se cumple el *deadline*. Estos casos son muy desfavorables y pueden no ser realistas, ya que no todos los controladores tendrán una FBN de igual tamaño (100 FB en este caso), ni todos los FB de los *daisy-chain* tendrán la máxima carga de trabajo. Un escenario estimado más ajustado a la realidad puede ser el siguiente:

- Controladores 1 y 3 responsables de las funcionalidades de carga y descarga en la máquina herramienta: longitud de `NORMAL_RT` igual a 30 FB y carga de trabajo de cálculos con enteros con vectores de 5K. Según la estimación del 20 % `NORMAL_NRT` tiene un longitud de 6 FB, no cambiando su carga de trabajo.

- Controlador 2 responsable de la funcionalidad de mecanizado en la máquina herramienta: longitud de `NORMAL_RT` igual a 60 FB y carga de trabajo de cálculos en coma flotante con vectores de 5K. Según la estimación del 20 % `NORMAL_NRT` tiene ahora un longitud de 12 FB, sin cambios tampoco en su carga de trabajo.

El WCET total de `NORMAL_NRT` en la FBN sintética puede calcularse en este escenario, de manera aproximada, mediante la ecuación:

$$WCET_{NRT} = WCET_{C1-3} * 2 + WCET_{C2} + WCET_{PAS} * 3 + WCET_{SE} * 2$$

Para un periodo de `NORMAL_NRT` de 50 ms tenemos:

$$WCET_{NRT} = 1{,}202 * 2 + 4{,}113 + 11{,}32 * 3 + 1{,}73 * 2 = 43{,}937\ ms$$

Y para un periodo de 100 ms:

$$WCET_{NRT} = 0{,}984 * 2 + 3{,}989 + 11{,}32 * 3 + 1{,}73 * 2 = 43{,}377\ ms$$

Comprobamos que en el escenario considerado se cumplen los *deadlines* para los dos periodos de 50 y 100 ms de `NORMAL_NRT`.

5.2.4.2. Escalabilidad

Pasamos ahora a analizar el resultado más relevante de estas pruebas: la escalabilidad de la plataforma COSME en relación con el incremento del número de FB que forman un *daisy-chain* (ver Sección 5.1). Al respecto, se han considerado `NORMAL_RT` y `NORMAL_NRT`, ya que son los *daisy-chain* que pueden variar en mayor medida su longitud en una FBN real. En el caso de `EXT_EVENT`, la atención al bus de campo del controlador puede ser efectuada mediante un número pequeño y limitado de FB, en la FBN sintética este número es de uno. Por su parte, como se ha indicado anteriormente, en la FBN sintética se ha considerado que `NORMAL_NRT` tiene en cada prueba el 20 % de FB de `NORMAL_RT`.

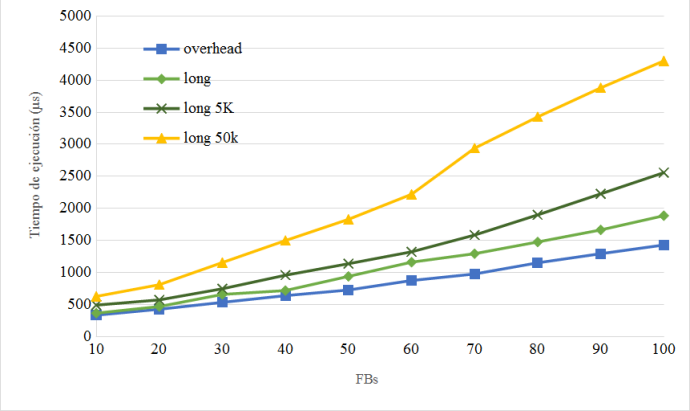


Figura 5.3: Escalabilidad de `NORMAL_RT`, escenario 1

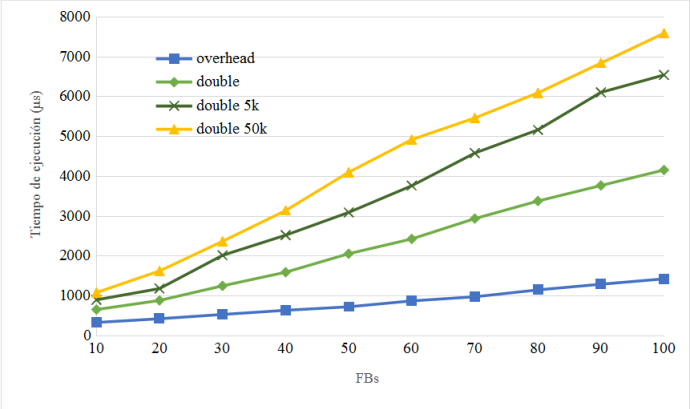


Figura 5.4: Escalabilidad de `NORMAL_RT`, escenario 2

En el caso de `NORMAL_RT` las Figs. 5.3 y 5.4 representan gráficamente los tiempos de ejecución mostrados en las Tablas 5.2 y 5.4. Los resultados indican el crecimiento lineal de estos tiempos conforme aumenta el número de FB para las distintas cargas de trabajo, lo que permite comprobar el rendimiento predecible de la plataforma COSME y por lo tanto su buena escalabilidad para este *daisy-chain* con restricciones *hard real-time*.

Por su parte, las Figs. 5.5 y 5.6 representan gráficamente los tiempos de ejecución de `NORMAL_NRT` mostrados en las Tablas 5.3 y 5.5. En este caso, se comprueba claramente la dependencia entre los tiempos obtenidos y la carga de trabajo de `NORMAL_RT`. En particular, y aunque el numero de FB de `NORMAL_NRT` va aumentando progresivamente (con carga constante cada uno), en la primera figura se observa que sus tiempos de ejecución únicamente empiezan a crecer linealmente a partir de 12 FB, para una carga de `NORMAL_RT` de `RTlong5K` y un periodo de `NORMAL_NRT` de 50 ms. De manera similar, en la segunda figura el crecimiento empieza a ser lineal a partir de 8 FB, para la carga `RTdouble5K` y periodo de 100 ms respectivamente. Al igual que en el caso anterior, las pruebas con este *daisy-chain* muestran la buena escalabilidad de la plataforma.

Finalmente, es necesario indicar que en las pruebas realizadas se han producido algunos valores de tiempos de ejecución irregulares (no mostrados en las tablas anteriores), en un numero inferior al 0,5‰ de las medidas tomadas. Este comportamiento afecta al cumplimiento de las restricciones de tiempo real y puede ser achacado al uso del computador Raspberry Pi y del sistema operativo Raspbian modificado para tiempo real con el *patch* PREEMPT_RT. Otros trabajos han indicado situaciones similares, por ejemplo [135]. Versiones futuras de dicho *patch* deberían disminuir estas irregularidades. En cualquier caso, en aplicaciones reales se requiere el empleo de arquitecturas hardware más potentes y sistemas operativos con especificaciones *hard real-time*.

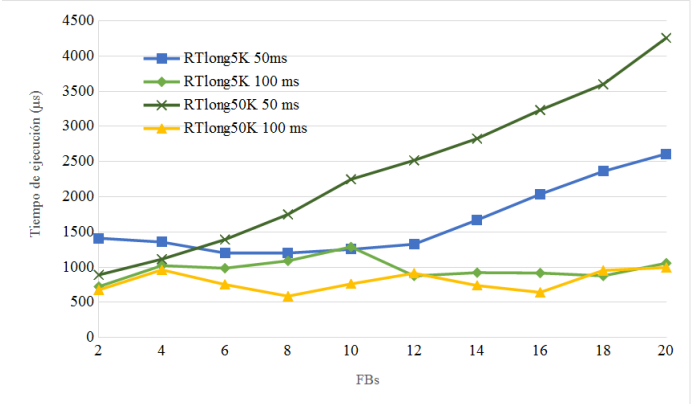


Figura 5.5: Escalabilidad de `NORMAL_NRT`, escenario 1

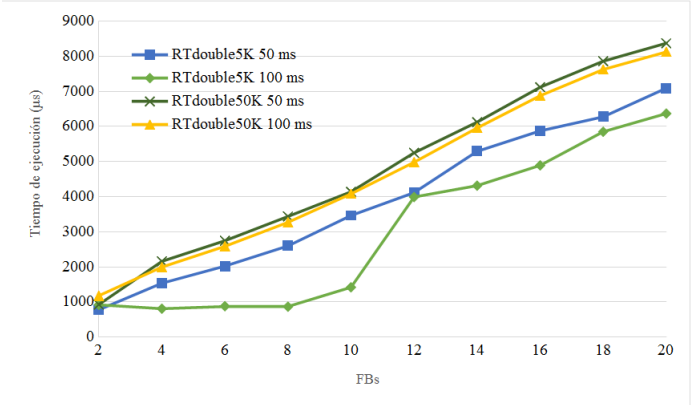


Figura 5.6: Escalabilidad de `NORMAL_NRT`, escenario 2

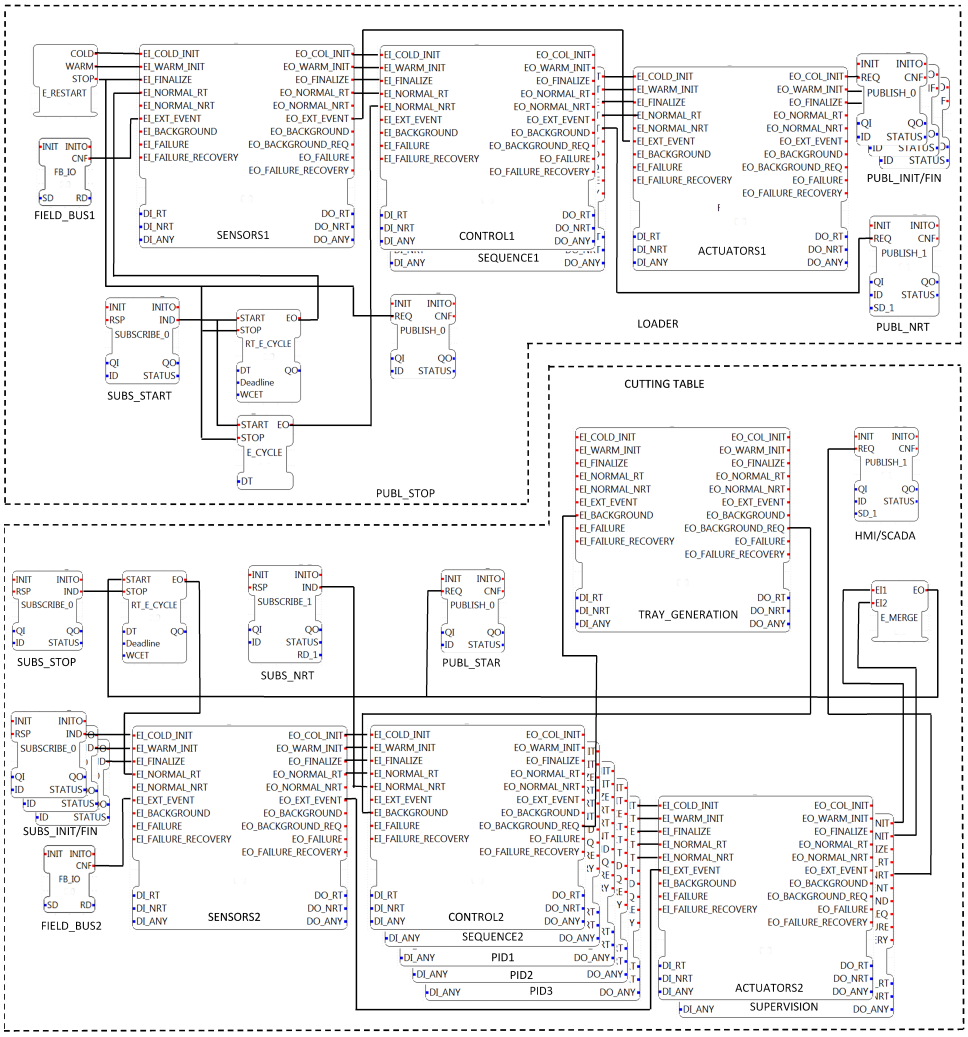


Figura 5.7: FBN simplificada para el control de la mesa de corte

5.3. Pruebas con una FBN de una aplicación real

La FBN de la aplicación real corresponde a la mesa de corte de vidrio monolítico y laminado fabricada por TUROMAS y presentada en el capítulo anterior (ver Sección 5.1). Para el control de esta máquina los ingenieros de control de dicha empresa han diseñado, mediante el IDE Domiciano, una FBN distribuida en dos controladores: uno para el cargador y otro para la mesa de corte y descargador. La razón para no utilizar un tercer controlador es la mayor sencillez de la operación de descarga en esta máquina. Cada uno de los controladores está conectado a un bus de campo CANopen para la interfaz de proceso. Mientras que, únicamente el segundo de ellos está conectado a una interfaz de usuario (o HMI).

La FBN tiene un número aproximado de 200 FB de 35 tipos distintos que controlan 70 secuencias automáticas, programadas en el lenguaje estándar SFC, y nueve ejes de posicionamiento preciso. Esta red está compuesta por distintas subredes similares a los casos de uso presentados en la Subsección 2.4.3.

Una versión simplificada de dicha FBN se muestra en la Fig. 5.7, donde se representan únicamente algunos FB y *daisy-chains*, sin conexiones de datos. En concreto: interfaz de proceso (`FIELD_BUS`), acondicionamiento de E/S (`SENSORS` y `ACTUATORS`), controles de ejes (`PID`, `TRAY_GENERATION`), secuencias automáticas (`SEQUENCE`), secuencias de control y de sincronización de proceso (`CONTROL`) y supervisión y acondicionamiento de datos de HMI (`SUPERVISION`).

Los *daisy-chain* `EXT_EVENT` y `NORMAL_RT` no están distribuidos, mientras que `COLD_INIT`, `WARM_INIT`, `FINALIZE` y `NORMAL_NRT` sí lo están; estableciéndose la conexión inter-controlador a través de SIFB `PUBLISH` y `SUBSCRIBE`. La ejecución periódica de `NORMAL_RT` y `NORMAL_NRT`, con periodos de 3 y 50 ms, comienza una vez ejecutado `COLD_INIT` o `WARM_INIT` en los dos controladores. Por su parte, `EXT_EVENT` tiene un periodo de 1 ms. Finalmente, se muestra también la conexión con una aplicación HMI/SCADA realizada al final de la ejecución de `NORMAL_NRT` con otro SIFB `PUBLISH`.

5.3.1. Plataforma de ejecución

Esta plataforma consta de dos controladores, implementados con un computador industrial dotado de un sistema operativo de tiempo real. Las conexiones inter-controlador son realizadas mediante una red dedicada cableada establecida con un dispositivo *Switched Ethernet*. Por otra parte, los computadores están también conectados a una segunda red cableada de propósito general para la comunicación con las aplicaciones externas de HMI y MES, así como con el IDE Domiciano. Los datos detallados de esta plataforma son los siguientes:

- *Denominación computador*: PC 620 Automation B&R
- *CPU*: Un *core* Intel Celeron a 1.1 GHz
- *Memoria*: 1 GB
- *I/O*: Una CANopen Master Card PCI-784 Adlink, dos X20 Remote I/O B&R, dos islas de válvulas s0700 SMC y nueve Sdrive Power Stages

- *Sistema operativo*: Linux Debian con *kernel* 2.6.31 modificado por RTAI
- *Java*: JVM HotSpot 1.6.0
- *Implementación COSME-RTAI*
- *Switched Ethernet*: Dos CISCO SLM200 Gigabit Smart Switch

5.3.2. Pruebas y discusión de resultados

En este caso las pruebas han sido realizadas en las instalaciones de la empresa fabricante de la máquina herramienta, donde se ha ejecutado la aplicación de control en modos de operación y carga de trabajo similares a los nominales de la mesa de corte. A diferencia de las pruebas con la FBN sintética hay menos posibilidades de modificar el número o carga de trabajo de los FB, debido a la implementación y la operación de la máquina. En concreto, el único grado de libertad posible es modificar el número de ejes habilitados en un momento dado.

Los tiempos de ejecución medidos son los WCETs obtenidos con la herramienta de *debug* de Domiciano y están representados (con valores en μs) en las Tablas 5.12, 5.13, 5.14 y 5.16.

El número aproximado de valores muestrados puede calcularse, igual que en las pruebas anteriores, a partir del tiempo de duración de la prueba (30 minutos) y el periodo de los *daisy-chain*, teniendo:

$$muestras_{EE} = 30 * 60/(1 * 10^{-3}) = 1800000$$

$$muestras_{RT} = 30 * 60/(3 * 10^{-3}) = 600000$$

$$muestras_{NRT} = 30 * 60/(50 * 10^{-3}) = 36000$$

Funcionalidad	**Tiempo de ejecución**
I/O CANopen	19,227

Tabla 5.12: Tiempo de ejecución de `EXT_EVENT` (valor en μs)

Funcionalidad	**Tiempo de ejecución**
I/O CANopen	109,851
I/O Acond Drive	69,631
I/O Acond E/S	1,003
PID	8,773
SFC corte	45,635
SFC carga	25,892

Tabla 5.13: Tiempos de ejecución unitarios de `NORMAL_RT` (valores en μs)

Funcionalidad	Tiempo de ejecución
Cargador	37,350
Mesa de corte	259,891

Tabla 5.14: Tiempo de ejecución de `NORMAL_NRT` (valores en μs)

Al igual que en la prueba con la FBN sintética, en primer lugar se determina el cumplimiento de las restricciones de tiempo real de los *daisy-chain* periódicos: `EXT_EVENT`, `NORMAL_RT` y `NORMAL_NRT` (este último en el caso intra-controlador). Para ello, comprobamos el cumplimiento de los *deadlines* mediante la ecuación de Liu y Layand (ver Subsección 3.4.5). Se hace únicamente en el caso de la mesa de corte, ya que el cargador tiene una funcionalidad más sencilla y por lo tanto menores tiempos de ejecución. Tomando los tiempos de ejecución mayores de las Tablas 5.12, 5.14 y 5.16, así como los periodos de 1, 3 y 50 ms tenemos:

$$U = \frac{0{,}019}{1} + \frac{1{,}670}{3} + \frac{0{,}259}{50} = 0{,}58$$

El resultado obtenido de U es menor que el límite de 0,78 establecido por la ecuación de Liu y Layand, cumpliendo por lo tanto los *daisy-chain* sus *deadlines*.

Elemento	Media	Desv.Std	95 %	100 %
Pasarela	2,58	0,65	10,19	37,24
Switched Ethernet	1,13	1,32	2,01	18,53

Tabla 5.15: Tiempos de ejecución en Pasarela y retardos de comunicación de *Switched Ethernet* para `NORMAL_NRT` (valores en ms)

En el caso de la ejecución inter-controlador de `NORMAL_NRT`, se consideran los tiempos indicados en las Tablas 5.14 y 5.15 para ambos controladores, con unas configuraciones de JVM HotSpot y del *Switched Ethernet* iguales que en las pruebas con la FBN sintética, tenemos:

$$T_{NRT} = 0{,}037 + 0{,}260 + 10{,}19 * 2 + 2{,}01 = 24{,}107\ ms$$

Este tiempo es menor que el periodo de 50 ms establecido para `NORMAL_NRT`, cumpliendo por lo tanto este *daisy-chain* su *deadline*.

Como se ha indicado, el factor considerado para determinar la escalabilidad de COSME con esta FBN es el número de ejes, teniendo en cuenta que su control incluye también la ejecución de algunas secuencias automáticas asociadas. La Fig. 5.8 representa gráficamente los tiempos de ejecución de `NORMAL_RT` representados en la Tabla 5.16. La variación de los ejes en grupos de tres es debida a la implementación y operación de la máquina. La mencionada gráfica indica el rendimiento predecible de la plataforma COSME, y por lo tanto, también en este caso su buena escalabilidad.

Por su parte, no se han detectado valores irregulares en los tiempos de ejecución, a diferencia de las pruebas efectuadas con la FBN sintética. Los motivos

Número de ejes	**3**	**6**	**9**
I/O CANopen, acond Drive y E/S, secuencias	646,345	1285,607	1670,541

Tabla 5.16: Tiempos de ejecución de `NORMAL_RT` por número de ejes (valores en μs)

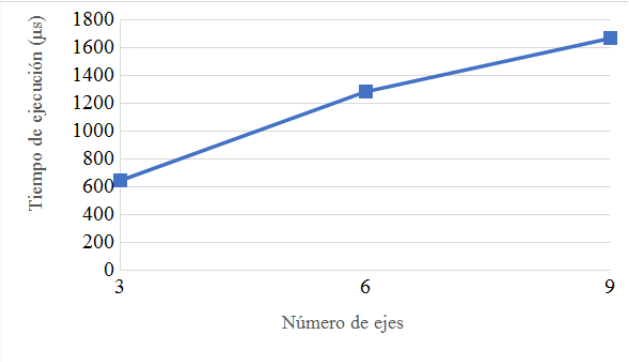


Figura 5.8: Escalabilidad por número de ejes de `NORMAL_RT`

pueden achacarse al empleo en esta ocasión de un computador específicamente diseñado para uso industrial, así como al *patch* de tiempo real RTAI para el sistema operativo Linux. Ambos más adecuados para la ejecución con restricciones *hard real-time*, que el computador Raspberry Pi y el *patch* PREEMPT_RT usados en la anterior plataforma de pruebas.

5.4. Comparativa con otras plataformas

En capítulos anteriores se han analizado las principales plataformas para IEC 61499 (ver subsecciones 3.2.4 y 4.6), así como las aproximaciones propuestas para la implementación de este estándar (ver Subsección 3.2.3). A este respecto, son pocos los trabajos que han presentado resultados experimentales que permitan determinar el rendimiento, el cumplimiento de las restricciones *hard real-time* o la escalabilidad de sus propuestas. A continuación se discuten brevemente tres trabajos relevantes, comparando las pruebas y resultados que presentan con los obtenidos con la plataforma COSME.

En [65] se presenta uno de los primeros análisis publicados sobre las diferentes aproximaciones posibles para la implementación de IEC 61499. La plataforma de pruebas es un PC compatible con Windows XP y el *runtime* FBRT [50], al que se modifica para estudiar seis implementaciones distintas. Al estar dicho *runtime* implementado en Java no es posible la ejecución de tiempo real. Las pruebas se efectúan con una FBN sintética y con una FBN que controla una sencilla máquina herramienta. Ambas redes no están distribuidas y siguen el patrón de conexión de cadenas de eventos `REQ-CNF` (ver Subsección 3.2.4). La FBN sintética está constituida por hasta 50 de dichas cadenas de 50 FB de longitud, ejecutadas concurrentemente, siendo la carga de trabajo cálculos en coma flotante. Entre las

implementaciones estudiadas destacamos la ejecución de las cadenas de eventos mediante llamadas directas, similar a la implementación de los *daisy-chain* del modelo adaptado de COSME. Los resultados obtenidos indican que esta implementación tiene los mejores tiempos de ejecución y escalabilidad, siempre que no haya eventos simultáneos o bucles en sus conexiones. Finalmente, se discuten los diferentes comportamientos de la segunda FBN obtenidos con las distintas implementaciones, aunque no se aportan resultados experimentales acerca de sus tiempos de ejecución o escalabilidad. Las diferencias respecto a las pruebas realizadas con la plataforma COSME son la ejecución sin restricciones de tiempo real y no distribuida, debido al empleo del *runtime* FBRT. Además, únicamente se aportan resultados acerca de tiempos de ejecución y escalabilidad en el caso de la FBN sintética.

El segundo trabajo es [4], e implementa una plataforma que sigue el modelo secuencial propuesto para IEC 61499 (ver subsecciones arriba indicadas). En este caso, se utilizan distintas plataformas de pruebas consistentes en tres sencillos controladores, uno con un procesador x86 y dos con un ARM7, dotados de sistemas operativos de tiempo real específicos. Al igual que el trabajo anterior, las pruebas se efectúan con una FBN sintética y una FBN real, ambas no distribuidas. La primera FBN está constituida por hasta diez cadenas de eventos de dos FB de longitud cada una, ejecutadas de manera concurrente, en este caso los FB implementan un sencillo contador. Los resultados obtenidos indican el cumplimiento de las restricciones de tiempo real, así como una buena escalabilidad. Por su parte, la FBN real estudiada (ver Fig. 3.9) realiza el control de un péndulo invertido, y como en el trabajo anterior sigue el patrón de conexión `REQ-CNF`. Se indica el correcto funcionamiento y el cumplimiento de las restricciones de tiempo real, aunque no se presentan resultados sobre tiempos de ejecución. A diferencia de las pruebas realizadas con COSME las FBN estudiadas en este trabajo son de pequeño tamaño, según se indica, debido a las limitaciones de las plataformas de pruebas empleadas; y como en el caso anterior su ejecución tampoco está distribuida.

El último de los trabajos aquí reseñados es [53], el cual presenta una plataforma que implementa un modelo de ejecución propio. En este modelo los FB son agrupados por criterios de ejecución de tiempo real, cada uno de estos grupos es ejecutado mediante una tarea independiente. La plataforma de pruebas es un PC compatible con Linux y RTAI, que ejecuta una FBN para el control de un brazo robótico siguiendo también el patrón de conexión `REQ-CNF`. Se indica el cumplimiento de las restricciones de tiempo real, aunque no se presentan tiempos de ejecución o datos para su comprobación. Finalmente, debido a la realización de las pruebas con una aplicación de control específica no se presentan experimentos que determinen la escalabilidad de la plataforma.

5.5. Conclusiones

De manera general, los modelos de ejecución y las plataformas IEC 61499 deben garantizar el comportamiento determinista de las FBN, así como el cumplimiento de las restricciones de tiempo real. Ambos requisitos pueden verse afectados por el tamaño de las FBN, debido a mecanismos de gestión de eventos (p.e. colas) u

overheads de la plataforma que impidan, a partir de un cierto tamaño de la red, cumplir dichos requisitos.

Se establece así como tercer requisito de ejecución en IEC 61499: la escalabilidad. Los trabajos que hasta la fecha han analizado esta cuestión lo han hecho, principalmente, desde el punto de vista del número de controladores, pero no en relación con el tamaño de las subFBN que se ejecutan en cada controlador. Este es un escenario diferente al contemplado en la propuesta para el control distribuido de máquinas herramienta en sistemas de fabricación ágil, donde el número de controladores es pequeño, pero no así el tamaño de las subFBN.

En este capítulo se han presentado, y discutido, distintas pruebas y resultados experimentales, con el fin determinar si las FBN diseñadas mediante los modelos adaptados cumplen los tres requisitos indicados. Dichas pruebas se han realizado en primer lugar con una FBN sintética distribuida, ejecutada sobre controladores implementados con un sencillo computador SoC. Esta FBN ha facilitado el estudio de casos con diferente número de FB y cargas de trabajo, donde se ha observado la buena escalabilidad de la plataforma COSME, y por tanto su predecible rendimiento al ejecutar FBN con un número elevado de FB. Dicha característica de la plataforma repercute, como se ha indicado al principio de estas conclusiones, en el cumplimiento de las restricciones de tiempo real, y por lo tanto en una ejecución determinista, requisito fundamental en cualquier aplicación de control.

Un segundo objetivo de este capítulo ha sido validar el funcionamiento general de la plataforma. Para ello, se han realizado pruebas con la aplicación de control distribuido de una mesa de corte de vidrio monolítico y laminado, desarrollada por ingenieros de la empresa fabricante de la máquina herramienta, comprobando el correcto funcionamiento de COSME. Debido a la operación de dicha máquina estas pruebas han tenido menor grado de libertad que en el caso anterior, lo que ha limitado el estudio de diferentes casos con la FBN. Aún así, se ha podido determinar también el cumplimiento de los requisitos establecidos, en particular de la escalabilidad, revelando la idoneidad de los modelos adaptados propuestos para IEC 61499 en un caso de uso real de control industrial, y por lo tanto complejo.

Como hemos visto al final del presente capítulo, las plataformas propuestas hasta el momento que han presentado pruebas y resultados experimentales han sido evaluadas únicamente con aplicaciones sintéticas o sencillos casos de uso de laboratorio. A este respecto, para la aceptación de IEC 61499 por la industria es fundamental enfrentar al estándar con casos de uso reales, uno de los principales objetivos de la plataforma COSME.

Capítulo 6

Conclusiones

6.1. Conclusiones

Los propuestas presentadas en esta tesis están enmarcadas en el control distribuido de máquinas herramienta en sistemas de fabricación ágil. Estos sistemas, una evolución de los anteriores sistemas de fabricación flexible, surgen a consecuencia del acortamiento del ciclo de vida de los productos. Su agilidad es el resultado de emplear sistemas reconfigurables capaces de modificar su estructura y comportamiento sin necesidad de parar su funcionamiento, así como del uso masivo de tecnologías de la información y la comunicación.

Las máquinas herramienta son elementos importantes de las celdas/líneas de cualquier sistema de fabricación. En este trabajo se han identificado las características de estas máquinas que hacen posible cumplir los requerimientos de los nuevos sistemas de fabricación. Dichas características son: a) la estrecha coordinación con otros elementos de la celda/línea de fabricación; b) la reconfigurabilidad; c) una gestión de fallos encaminada a reducir paradas no programadas y d) la integración en los sistemas de información de la factoría. De estas características la más distintiva es la reconfigurabilidad, gracias a una estructura modular y un comportamiento modificable mediante el cambio en tiempo de ejecución de su software de control.

A continuación, se han establecido los principios generales de diseño del software de control en los sistemas de fabricación ágil. El primero, el uso de estándares y sistemas abiertos que eviten un diseño "artesanal", dada la complejidad de los sistemas a controlar. El segundo, un paradigma de programación orientada a componentes como mecanismo para facilitar la reconfiguración dinámica, además del acortamiento de los tiempos de desarrollo y la mejora de la fiabilidad. Y el tercero, y último, arquitecturas orientadas al control distribuido que faciliten la reconfiguración dinámica y escalable, así como el trabajo coordinado de los elementos que constituyen las celdas/líneas de fabricación, en particular, de las máquinas herramienta.

IEC 61499 es el estándar propuesto para cumplir los requisitos del software de control de estos nuevos sistemas. Su revisión en relación con los principios de diseño anteriormente establecidos ha permitido discutir los principales retos y cuestiones pendientes del estándar, los cuales hacen que a día de hoy todavía no haya sido aceptado por la industria. Esos retos tienen que ver con su capacidad para hacer frente a aplicaciones reales y complejas de control industrial de manera: predecible, escalable, mantenible y extensible. En relación con las cuestiones pendientes, la primera es la dificultad de implementación del estándar, en parte por las ambigüedades en la definición de su modelo de ejecución por eventos, tardíamente solventadas en gran medida en la última edición del estándar. La segunda cuestión es la escasez de plataformas y entornos de desarrollo, que además en la mayoría de los casos carecen de la madurez de las existentes para PLC. Estas plataformas han sido empleadas en casos de uso de laboratorio, que no tienen una dificultad suficiente para verificar que el estándar, y las propias plataformas, cumplen los retos antes indicados. Y la tercera cuestión es, precisamente, la mayor complejidad de los sistemas a controlar, que hace necesario el empleo de metodologías y herramientas de la ingeniería software, por otra parte, desconocidas por la mayoría de los ingenieros de control responsables del diseño de tales aplicaciones. A este respecto, IEC 61499 no ofrece metodologías o guías, especialmente para las fases iniciales de ese diseño.

Los retos y cuestiones pendientes indicados han llevado a proponer como aportaciones originales de esta tesis modelos de FB y de ejecución IEC 61499, así como una plataforma, que contribuyan a hacer frente a estos retos y así facilitar la adopción del estándar por la industria. Dichos modelos y plataforma están adaptados al dominio del control distribuido de las máquinas herramienta en sistemas de fabricación ágil.

Para establecer dichos modelos el primer paso ha sido caracterizar el dominio considerado mediante una máquina herramienta genérica y presentar una propuesta para su control distribuido. Esta propuesta consiste en una arquitectura *peer-to-peer* de grano grueso, donde han sido identificados y caracterizados los tipos de conexiones entre las partes de dicho control y establecidas las distintas restricciones de tiempo real. Seguidamente este control ha sido modelado mediante casos de uso significativos, identificando los bloques funcionales y estableciendo sus relaciones estructurales y su comportamiento dinámico.

A partir de ahí se ha establecido el modelo de FB IEC 61499 adaptado en base a dos principios: 1) la predefinición de algoritmos, eventos y de su comportamiento a partir de las diferentes prioridades de cada evento; y 2) la asociación de cada algoritmo con un evento de entrada y uno de salida, con el fin de establecer cadenas de eventos homólogos en una topología en *daisy-chain*. Los algoritmos y el comportamiento predefinidos corresponden con las funcionalidades identificadas en el modelado. Mientras que las cadenas de eventos corresponden con los tipos de conexiones entre las partes del control distribuido. Este modelo de FB adaptado tiene una semántica clara y unas reglas simples para la composición de FBN distribuidas, las cuales han permitido establecer una metodología para su diseño.

El siguiente punto abordado ha sido determinar el modelo de ejecución de tales FBN adaptado al dominio. Sus requisitos más importantes son garantizar un comportamiento determinista (o predecible), el cumplimiento de las diferentes restricciones de tiempo real establecidas en la propuesta anterior de control distribuido y tener una implementación eficiente. Como paso previo, se ha discutido la ejecución de FBN en IEC 61499, en particular, los principales problemas derivados de sus ambigüedades semánticas, y las diferentes aproximaciones y modelos propuestos hasta la fecha. A este respecto, hay dos clases fundamentales de modelos, los que atienden los eventos por orden de generación (modelos secuenciales) y los que atienden los eventos por un orden de ejecución de los FB establecido en tiempo de diseño (modelos cíclicos). En la práctica, los primeros deben limitar el tipo de conexiones en las FBN para garantizar un comportamiento determinista y una implementación eficiente, a pesar de lo cual pueden no cumplir las restricciones de tiempo real. Por su parte, los segundos resultan muy similares al modelo de ejecución de IEC 61131, y por lo tanto tienen sus mismos inconvenientes.

Con el objeto de cumplir los requisitos del modelo adaptado se ha establecido un nuevo criterio para atender los eventos: su orden de prioridad. A partir de este criterio se ha determinado el modelo de ejecución adaptado, consistente en asignar cada uno de los *daisy-chain* predefinidos por el modelo de FB adaptado a una tarea, que invoca secuencialmente los algoritmos asociados de los FB. La ejecución de estas tareas es planificada según las prioridades de los eventos. Este modelo de ejecución ha permitido una implementación de IEC 61499 determinista, al evitar las colas de eventos usadas en otras propuestas, así como eficiente. Además, también ha hecho posible el cumplimiento de las restricciones de tiempo real del dominio.

Una vez establecidos los modelos de FB y ejecución IEC 61499 adaptados, el siguiente paso ha sido la plataforma COSME, constituida por un *runtime* y el entorno de desarrollo Domiciano, como medio para verificar experimentalmente dichos modelos. El *runtime* implementa el modelo de ejecución adaptado, mientras que, por su parte, el entorno facilita el diseño de FBN ocultando la complejidad propia del estándar e incorporando herramientas similares a las de entornos comerciales existentes de PLC. Las pruebas realizadas en laboratorio con COSME han mostrado resultados experimentales que verifican el cumplimiento de los requisitos indicados con anterioridad. A diferencia de anteriores propuestas, la validación final de la plataforma ha sido efectuada en un caso de uso real y complejo desarrollado por ingenieros de control.

Para concluir finalmente, esta tesis es una aproximación pragmática a IEC 61499 que propone el uso de modelos adaptados como una solución original para facilitar el diseño, así como una implementación del estándar determinista, eficiente y escalable. Cuestiones que forman parte de los retos pendientes que tiene este estándar para ser aceptado por la industria.

6.2. Líneas de trabajo futuras

Entre las posibles líneas futuras con las que continuar el trabajo realizado en esta tesis se encuentran:

- Sobre cuestiones generales de IEC 61499
  - Ejecución de tiempo real de FBN
    Actualmente el estándar considera únicamente modelos de ejecución que atienden los eventos por orden de generación (modelos secuenciales) o por orden de ejecución de los FB establecido en tiempo de diseño (modelos cíclicos). Ambos tienen el inconveniente de que las restricciones de tiempo real durante la ejecución de una FBN pueden no cumplirse, especialmente si hay áreas o grupos de FB con distintas restricciones. Como este trabajo ha mostrado, para este cumplimiento hay que considerar la prioridad como nuevo criterio a la hora de atender los eventos. Con este fin, es necesario trabajar en una propuesta de modificación de IEC 61499 que incluya modelos de ejecución basados en este criterio.
    Por otra parte, está la cuestión de la atomicidad de los FB. La segunda versión del estándar incide en ella de manera menos ambigua que en la primera versión. En cualquier caso, dicha atomicidad está fuertemente influida por la preferencia del estándar por los modelos secuenciales de ejecución, ya que simplifican la implementación y facilitan el determinismo. Los propios autores de estos modelos reconocen que tienen como handicap la ejecución con restricciones de tiempo real. El hecho de ser éste un requisito fundamental de cualquier software de control indica la necesidad de realizar trabajos que estudien esta cuestión.
  - Determinismo vs. colas de eventos
    Como hemos visto, los modelos de ejecución del estándar que atienden los eventos por orden de generación hacen uso de colas. Su principal inconveniente es las situaciones de indeterminismo a las que pueden dar lugar durante la ejecución de las FBN, especialmente en redes complejas con un gran número de FB o de conexiones. La solución propuesta en los trabajos que ha identificado este problema es poner limitaciones de diseño, particularmente, en el tipo de conexiones de eventos (p.e. patrón `REQ-CNF`). Aunque algunos trabajos han abordado cuestiones de verificación formal relacionadas con la ejecución en IEC 61499, debe hacerse una mayor esfuerzo por estudiar las repercusiones del uso de estas colas. En relación con el punto anterior, dicho estudio debe incluir el uso de colas con prioridad.
  - Generalización de modelos adaptados
    Los modelos de FB y ejecución de IEC 61499 propuestos en este trabajo están adaptados al dominio del control distribuido de máquinas herramienta en sistemas de fabricación ágil. Aunque estos modelos tienen la versatilidad suficiente para ser empleados en otros dominios de aplicación, una línea de trabajo es el establecimiento de reglas que permitan la derivación de otros modelos adaptados.

- Sobre la plataforma COSME

  - Reconfigurabilidad dinámica

    La versión actual de la plataforma incorpora servicios para la reconfiguración simple de la FBN. Como recordamos, este tipo de reconfiguración puede llevar al sistema a un funcionamiento inseguro. Por este motivo, la siguiente versión de la plataforma debe incorporar un elemento denominado **Gestor de reconfiguración** que permita reconfigurar dinámicamente la FBN, garantizando así la consistencia del sistema. Un mecanismo para ello es la realización de operaciones de reconfiguración transaccionales.

  - Versión *open source*

    La disponibilidad actual de la plataforma está condicionada por acuerdos de transferencia y colaboración tecnología con empresas privadas. La siguiente versión a desarrollar debe emplear licencias *open source* para la puesta a libre disposición de la comunidad de IEC 61499.

6.3. Publicaciones

Las publicaciones indicadas a continuación contienen todas las aportaciones de esta tesis. En ellas, han contribuido también, aunque de manera parcial y en menor medida, otros autores, como se detallará de manera particular en cada publicación. Respecto de los directores de la tesis firmantes en cada publicación su labor ha sido la de guía, asesoramiento y validación.

6.3.1. Capítulos de libros

- **C. Catalán**, A. Blesa, F. Serna y J.M. Colom, "An adapted design methodology to the IEC 61499 for distributed control applications of machine tools", en *Distributed Control Applications: Guidelines, Design Patterns, and Application Examples With The IEC 61499*, CRC Press/Taylor and Francis Group.

  En esta publicación se presentan los modelos IEC 61499 adaptados que permiten el establecimiento de una metodología original para el control distribuido de máquinas herramienta. Dichos modelos y metodología han sido establecidos por el autor, para lo que ha contado con la colaboración de F. Serna en la parte de modelado del control distribuido de una máquina genérica de este tipo.

  Participación por invitación personal al autor de esta tesis.

  Indicios de calidad:

  El libro reúne las principales aportaciones de los últimos años sobre el tema, estando prevista su publicación en noviembre de 2015. Sus editores son dos de los más importantes investigadores sobre IEC 61499:

Alois Zoitl
http://www.fortiss.org/en/about-us/people/alois-zoitl/
Miembro del IEC SC 65B WG 15 (i.e. IEC 61499) y con más de 100 publicaciones sobre este estándar

Thomas Strasser
http://www.ait.ac.at/profile/detail/Strasser-Thomas/
Miembro del IEC SC 65B WG 15 (i.e. IEC 61499) y con más de 40 publicaciones sobre el estándar

6.3.2. Conferencias internacionales

- **C. Catalán**, F. Serna, A. Blesa, J.M. Colom, y J.M. Rams, "COSME: A distributed control platform for communicating machine tools in Agile Manufacturing Systems", *IEEE Conference on Emerging Technologies & Factory Automation*, Toulouse (Francia), 2011

  En esta publicación la aportación del autor ha sido el establecimiento de los principios de diseño de software de control para el tipo de máquinas considerado, la especificación y diseño de la arquitectura de la plataforma COSME, así como del *runtime* que implementa el modelo de ejecución adaptado propuesto. Por su parte, F. Serna ha colaborado en la parte de la plataforma relacionada con las comunicaciones inter-controlador y con aplicaciones externas. Finalmente, J.M Rams ha contribuido con el diseño de la red de bloques función que constituyen la aplicación de control utilizada para el test de la plataforma.

  Indicios de calidad: Qualis B1, Google scholar: h5-index: 20, 1 cita (sin autocitas), en esta edición el índice de aceptación de regular papers fue del 29,64 %.

- **C. Catalán**, F. Serna, T. Civera, A. Blesa y J.M. Rams, "Control design for machine tools using Domiciano, an IDE based on software components", *IEEE Conference on Emerging Technologies & Factory Automation*, Toulouse (Francia), 2011

  En esta publicación se presenta el entorno de desarrollo Domiciano, el cual forma parte de la plataforma COSME. El autor ha contribuido en su especificación y diseño, en particular, en el *workflow* de desarrollo de aplicaciones, en la generación de código y en la interfaz de usuario. Por su parte, F. Serna ha colaborado en la especificación y diseño de la depuración de aplicaciones y de la comunicación con la plataforma de ejecución. Finalmente, T. Civera y J.M. Rams han contribuido en la implementación y validación del entorno, respectivamente.

  Indicios de calidad: Qualis B1, Google scholar: h5-index: 20, 1 cita (sin autocitas), en esta edición el índice de aceptación de work-in-progress papers fue del 73,02 %.

- **C. Catalán**, F. Serna, A. Blesa, J.M. Rams y J.M. Colom, "Communication types for manufacturing systems. A proposal to distributed control

system based on IEC 61499", *IEEE Conference on Automation Science and Engineering*, Triestre (Italia), 2011

La contribución del autor en este trabajo ha consistido en especificar los tipos de comunicación que han llevado a una propuesta original para el control distribuido de máquinas herramienta. Por su parte, F. Serna ha colaborado en las comunicaciones inter-controlador de la aplicación de test y J.M. Rams en el diseño y validación de esa aplicación.

Indicios de calidad: CORE B, Qualis B1, Google scholar: h5-index: 14, 1 cita (sin autocitas), en esta edición el índice de aceptación de regular papers fue del 57,65 %.

- **C. Catalán**, F. Serna, A. Blesa y J.M. Rams, "IEC 61499 execution model based on life cycle of function blocks", *IEEE Conference on Emerging Technologies & Factory Automation*, Bilbao, 2010

  En este trabajo se presenta la versión no distribuida de los modelos de bloque función y ejecución IEC 61499 adaptados. El resto de autores han contribuido con la aplicación de test utilizada.

  Indicios de calidad: Qualis B1, Google scholar: h5-index: 20, 1 cita (sin autocitas), en esta edición el índice de aceptación de regular papers fue del 29,79 %.

6.3.3. Conferencias nacionales

- **C. Catalán**, A. Blesa y F. Serna "Industria 4.0 en el Grado de Ingeniería Electrónica y Automática", *Jornadas sobre la Enseñanza Universitaria de la Informática*, Andorra La Vella (Principado de Andorra), 2015

  Este trabajo pretende abrir un debate sobre los contenidos de la enseñanza en el ámbito de la informática que debe recibir este tipo de graduados, a la vista del incremento de la complejidad del software de control en el nuevo tipo de sistemas de fabricación indicado. El autor ha contribuido con diferentes propuestas para la modificación de los planes de estudio. El resto de autores han contribuido con el análisis de los actuales planes de estudio de este grado en España.

6.4. Otras publicaciones relacionadas

Las siguientes publicaciones contienen otras aportaciones estrechamente relacionadas con la tesis en las cuales el autor ha contribuido en menor medida que el autor principal.

6.4.1. Capítulos de libros

- F. Serna, **C. Catalán**, A. Blesa, J.M. Colom y J.M. Rams, "Control software for a cutting glass machine tool built using the COSME platform. A

case study", en *Distributed Control Applications: Guidelines, Design Patterns, and Application Examples With The IEC 61499*, CRC Press/Taylor and Francis Group. Publicación prevista en noviembre 2015

En esta publicación se presenta el principal caso de estudio de la plataforma COSME. La contribución del autor ha consistido en el establecimiento de los modelos de bloque función y ejecución IEC 61499 adaptados que constituyen la base de dicha plataforma, así como en la propuesta de control distribuido de la máquina indicada.

Participación por invitación personal al autor de esta tesis.

Indicios de calidad:

Este capítulo será publicado en el libro ya reseñado anteriormente.

6.4.2. Conferencias internacionales

- A. Blesa, C. Larrea, **C. Catalán**, J.M. Colom, F. Serna y J.M. Rams, "Function Blocks for the design of control applications based on EtherCAT fieldbus", *IEEE Conference on Industrial Technology*, Sevilla, 2015

  En esta publicación se propone el uso del bus de campo EtherCAT para la interfaz de proceso en la plataforma COSME. El autor ha colaborado en la especificación y diseño de los bloques función propuestos.

  Indicios de calidad: Qualis B3, Google scholar: h5-index: 18

- F. Serna, **C. Catalán**, A. Blesa, J.M. Rams y J.M. Colom, "Predictive Maintenance Surveyor Design Pattern for Machine Tools Control Software Applications", *IEEE Conference on Emerging Technologies & Factory Automation*, Toulouse (Francia), 2011

  Una parte importante dentro del proyecto de investigación, transferencia y colaboración tecnológica indicado previamente es el establecimiento de patrones de diseño para el control distribuido de máquinas herramienta. En esta publicación se establece uno de esos patrones de diseño, en particular, para el mantenimiento predictivo de tales máquinas. El autor ha contribuido en el diseño este patrón mediante el modelo de bloque función adaptado propuesto en esta tesis.

  Indicios de calidad: Qualis B1, Google scholar: h5-index: 20, 2 citas (sin autocitas), en esta edición el índice de aceptación de regular papers fue del 29,64 %.

- F. Serna, **C. Catalán**, A. Blesa, J.M. Rams y J.M. Colom, "Control software for a cutting glass machine tool built using the COSME platform. A case study", *IEEE Conference on Automation Science and Engineering*, Triestre (Italia), 2011

  Versión reducida de la publicación del segundo capítulo del libro reseñado anteriormente.

  Indicios de calidad: CORE B, Qualis B1, Google scholar: h5-index: 14, en esta edición el índice de aceptación de regular papers fue del 57,65 %.

- F. Serna, **C. Catalán**, A. Blesa, J.M. Rams y J.M. Colom, "Preventive Maintenance Manager design pattern for component based machine tools", *IEEE Conference on Industrial Informatics*, Lisboa (Portugal), 2011

  Esta publicación corresponde con un segundo patrón de diseño para el control distribuido de máquinas herramienta. La contribución del autor ha sido igual que en el caso anterior.

  Indicios de calidad: Qualis B3, Google scholar: h5-index: 17

- F. Serna, **C. Catalán**, A. Blesa y J.M. Rams, "Design Patterns for Failure Management in IEC 61499 Function Blocks", *IEEE Conference on Emerging Technologies & Factory Automation*, Bilbao, 2010

  Esta publicación corresponde con un tercer patrón de diseño para el control distribuido de máquinas herramienta. La contribución del autor ha sido igual que en los casos anteriores.

  Indicios de calidad: Qualis B1, Google scholar: h5-index: 20, 4 citas (sin autocitas), en esta edición el índice de aceptación de regular papers fue del 29,79 %.

6.4.3. Congresos industriales internacionales

- F. Serna, **C. Catalán**, A. Blesa, J.M. Colom y J.M. Rams, "COSME: a framework for agile manufacturing systems", *IEC 61499 Day: Industrial Use Cases of Distributed Intelligent Automation in SPS/IPC/Drives Exhibition for electric automation technology*, Nuremberg (Alemania), 2011

  Participación por invitación personal al autor de esta tesis.

  Indicios de calidad:

  La plataforma COSME ha resultado de interés para la industria, lo que se manifiesta por la invitación a presentarla en el primer *Workshop* sobre el estado del arte IEC 61499, celebrado en la feria más importante a nivel europeo en su campo. Este evento ha sido organizado por Valeriy Vyatkin (http://www.ltu.se/staff/v/valvya-1.96049?l=en), uno de los principales investigadores sobre IEC 61499 con más de 150 publicaciones, y esponsorizado por las dos empresas más importantes fabricantes de equipos de automatización con dicho estánd

6.4.4. Conferencias nacionales

- A. Blesa, E. Alcalá, A. Azuara, **C. Catalán** y F. Serna "Sistema de tiempo real con Raspberry Pi", *Seminario Anual e Automática y Electrónica Industrial*, Zaragoza, 2015

  La contribución del autor ha sido la adaptación de la plataforma COSME al computador Raspberry Pi.

- A. Blesa, **C. Catalán** y F. Serna, "Programación basada en componentes para la docencia de sistemas electrónicos de control: aplicación de estándares", *Congreso de Tec. Aplicadas a la Enseñanza de la Electrónica*, Madrid, 2010

  La contribución del autor ha sido la propuesta del empleo IEC 61499 para la realización de prácticas de la materia indicada.

Bibliografía

[1] K. Thramboulidis, "IEC 61499: Back to the well proven practice of IEC 61131?," in *Proc. 17th IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1–8, 2012.

[2] V. Vyatkin, *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design (2nd edition)*. O3NEIDA and Instrumentation Society of America (ISA), 2011.

[3] G. Doukas and K. Thramboulidis, "A Real-Time Linux Execution Environment for Function-Block Based Distributed Control Applications," in *Proc. 3rd IEEE International Conference on Industrial Informatics*, pp. 56–61, 2005.

[4] A. Zoilt, *Real-Time Execution for IEC 61499*. O3NEIDA and Instrumentation Society of America (ISA), 2009.

[5] Y. Koren, U. Heisel, F. Jovane, T. Moriwak, G. Pritsshow, G. Ulsoy, and H. van Brussel, "Reconfigurable manufacturing systems," *Annals of the International Institution for Production Engineering Research (CIRP)*, vol. 48, no. 2, pp. 527–539, 1999.

[6] R. N. Nagel and R. Dove, *21st Century Manufacturing Enterprise Strategy: An Industry-Led View*. Diane Pub Co, 1991.

[7] Y. Y. A Guneskaran, "Agile manufacturing: A taxonomy of strategic and tecnological imperatives," *International Journal of Production Research*, vol. 40, no. 6, pp. 1357–1385, 2002.

[8] W. Shen, Q. Hao, and D. H. H Yoon, "Applications of agent-based systems in intelligent manufacturing: An updated review," *Advanced Engineering Informatics*, vol. 20, no. 4, pp. 415–431, 2006.

[9] European Commision, *Report of the High-Level-Group, MANUfuture: Strategic Research Agenda, Assuring the future of Manufacturing in Europe*, (accesible en www.manufuture.org/manufacturing/wp-content/uploads/Manufuture-SRA-web-version.pdf) ed., 2006.

[10] M. Hermann, T. Pentek, and B. Otto, *Design Principles for Industrie 4.0 Scenarios: A Literature Review*. Technische Universität Dortmund, 2015.

[11] L. D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[12] "Intelligent Manufacturing Systems, www.ims.org (accedida en mayo 2015)."

[13] R. Frei, J. Barata, and M. Onori, "Evolvable Production Systems C ontext and Implications," *in Proc. IEEE International Symposium on Industrial Electronics*, pp. 3233–3238, 2007.

[14] M. Koc, J. Ni, J. Lee, and P. Bandyopadhyay, "Introduction of e-manufacturing," *Transactions of the North American Manufacturing Research*, vol. 6, 2003.

[15] D. Wu, M. J. Greer, D. W. Rosen, and D. Schaefer, "Cloud manufacturing: Strategic vision and state-of-the-art," *Journal of Manufacturing Systems*, vol. 32, no. 4, pp. 564–579, 2013.

[16] R. Brennan, P. Vrba, P. Tichy, A. Zoitl, C. Snder, T. Strasser, and V. Marik, "Developments in dynamic and intelligent reconfiguration of industrial automation," *Computers in Industry*, vol. 59, no. 6, pp. 533–537, 2008.

[17] Y. Koren and M. Shpitalni, "Design of reconfigurable manufacturing systems," *Journal of Manufacturing Systems*, vol. 29, no. 4, pp. 130–141, 2010.

[18] A. Tharumajah and A. W. y L Nemes, "Comparison of emerging manufacturing concepts," in *Proc. IEEE International Conference on Systems, Man, and Cybernetics*, pp. 325–331, 1998.

[19] K. Ryu, Y. Son, and M. Yung, "Modeling and specifications of dynamic agents in fractal manufacturing systems," *Computers in Industry*, vol. 52, no. 2, pp. 161–182, 2003.

[20] J. Mun, K. Ru, and M. Jung, "Self-reconfigurable software architecture: Design and implementacion," *Computers & Industrial Engineering*, vol. 51, no. 1, pp. 163–173, 2006.

[21] M. Canavesio and E. Martnez, "Enterprise modeling of a project-oriented fractal company for smes networking," *Computers in Industry*, vol. 58, no. 8-9, pp. 794–813, 2007.

[22] P. Leitao, J. Barbosa, and D. Trentesaux, "Bio-inspired multi-agent systems for reconfigurable manufacturing systems," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 5, pp. 934–944, 2012.

[23] A. Koestler, *The Ghost in the Machine (1990 reprint ed.)*. Penguin Group, 1967.

[24] Intelligent Manufacturing Systems, *Report HMS PHASE I AND II: Holonic Manufacturing Systems*, 2011, (accesible en www.ims.org/2011/11/hms--phase-i-and-ii-holonic-manufacturing-systems).

[25] J. Christensen, "Holonic manufacturing systems: Initial architecture and standards directions," in *in Proc. First European Conference on Holonic Manufacturing Systems*, pp. 1–20, 1994.

[26] J. Mathews, "Organizational foundations of intelligent manufacturing systems- the holonic viewpoint," *Computer Integrated Manufacturing*, vol. 8, no. 4, pp. 237–243, 1995.

[27] P. Leitao, "Agent-based distributed manufacturing control: A state-of-the-art survey," *Engineering Applications of Artificial Intelligent*, vol. 22, no. 7, pp. 979–991, 2009.

[28] R. Brennan and W. G. y K Hall, "Forward-Special Issue on Industrial Applications of Holonic Manufacturing Systems," *IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and reviews*, vol. 41, no. 1, pp. 1–3, 2011.

[29] F. Hsieh and C. Chiang, "Collaborative composition of processes in holonic manufacturing systems," *Computers in Industry*, vol. 62, no. 1, pp. 51–64, 2011.

[30] R. Brennan, X. Zhang, Y. Xu, and D. Norrie, "A reconfigurable concurrent function block model and its implementation in real-time Java," *Journal of Integrated Computer-Aided Engineering*, vol. 9, pp. 263–279, 2002.

[31] International Electrotechnical Commission, *Function blocks Part 1: Architecture, IEC 61499-1 Standard*, 2.0 ed., 2005.

[32] International Standard Organization, *Numerical Control of Machines - Program Format and Definition of Address Words, Part 1. Data Format for positioning, line motion and contouring control systems, ISO 6983-1 Standard*, 1982.

[33] International Standard Organization, *Data Model for Computerized Numerical Controllers, Part 1. Overview and fundamental principles, ISO 14649-1 Standard*, 2003.

[34] X. Xu and S. Newman, "Making CNC machine tools open, interoperable and intelligent-a review of technologies," *Computers in Industry*, vol. 57, no. 3, pp. 141–152, 2006.

[35] S. Mekid, P. Pruschek, and J. Hern¡ndez, "Beyond intelligent manufacturing: A new generation of flexible intelligent NC machines," *Mechanism and Machine Theory*, vol. 44, no. 2, pp. 466–476, 2009.

[36] J. Padayachee and G. Bright, "Modular machine tools: Design and barriers to industrial implementations," *Journal of Manufacturing Systems*, vol. 31, no. 2, pp. 92–102, 2012.

[37] M. Colla, T. Leidi, and M. Semo, "Design and implementation of industrial automation control systems: A survey," *in Proc. 7th IEEE International Conference on Industrial Informatics*, pp. 570–575, 2009.

[38] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 2002.

[39] M. Strasser, T.and Rooker, I. Hegny, M. Wenger, A. Zoitl, L. Ferrarini, A. Dede, and M. Colla, "A research roadmap for model-driven design of embedded systems for automation components," in *in Proc. 7th IEEE International Conference on Industrial Informatics*, pp. 564–569, 2009.

[40] D. Hästbacka, T. Vepsäläinen, and S. Kuikka, "Model-driven development of industrial process control applications," *The Journal of Systems and Software*, vol. 84, no. 7, pp. 1100–1113, 2011.

[41] K. Thramboulidis, "IEC 61499 in Factory Automation," in *in Proc. IEEE International Conference on Industrial Electronics, Technology and Automation*, pp. 115–124, 2005.

[42] A. Zoitl and V. Vyatkin, "Face to face: IEC 61499 architecture for distributed automation: The glass half full view," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 7–23, 2009.

[43] International Electrotechnical Commission, *Programmable controllers - Part 1: General information, IEC 61131-1 Standard*, 2.0 ed., 2003.

[44] International Electrotechnical Commission, *Programmable controllers - Part 3: Programming Languages, IEC 61131-3 Standard*, 2.0 ed., 2003.

[45] R. Breannan, "Toward real-time distributed intelligent control: A survey of research themes and applications," *IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews*, vol. 37, no. 6, pp. 744–765, 2007.

[46] K. Thramboulidis, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State of the Art Review - A Different View," *Hindawi Publishing Corporation Journal of Engineering*, vol. Article ID 638521, 2013.

[47] K. Hall, R. Staron, and A. Zoitl, "Challenges to Industry Adoption of the IEC 61499 Standard on Event-based Function Blocks," in *Proc. 5th IEEE International Conference on Industrial Informatics*, pp. 823–828, 2007.

[48] V. Vyatkin and V. Dubinin, "Sequential axiomatic model for execution of basic function blocks in IEC 61499," in *Proc. 5th IEEE International Conference on Industrial Informatics*, pp. 1183–1188, 2007.

[49] M. Colla, A. Brusaferri, and E. Carpanzano, "Applying the IEC 61499 Model to the Shoe Manufacturing Sector," *in Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1301–1308, 2006.

[50] "Function Block Development Kit disponible en www.holobloc.com/doc/fbdk/index.htm (accedida en mayo 2015)."

[51] "4DIAC Framework for Distributed Industrial Automation and Control disponible en www.fordiac.org (accedida en mayo 2015)."

[52] A. Zoitl, T. Strasser, and A. Valentini, "Open source initiatives as basis for the establishment of new technologies in industrial automation: 4DIAC a case study," in *Proc. IEEE Symposium on Industrial Electronics*, pp. 3817–3819, 2010.

[53] G. Doukas and T. K., "A real-time-linux-based framework for model-driven engineering in control and automation," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 3, pp. 914–924, 2011.

[54] "ISaGRAF Workbench, www.isagraf.com (accedida en mayo 2015)."

[55] V. Vyatkin and J. Chouinard, "On Comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations," in *Proc. 6th IEEE Conference on Industrial Informatics*, pp. 289–294, 2008.

[56] "Engineering tool nxtSTUDIO, www.nxtcontrol.com (accedida en mayo 2015)."

[57] "IEC 61499 Day: Industrial Use Cases of Distributed Intelligent Automation in SPS/IPC Drives Exhibition for electric automation technology, Nuremberg, (Germany)," 2011.

[58] K. Thramboulidis, "IEC 61499 vs. 61131: A Comparison Based on Misperceptions," *Journal of Software Engineering and Applications*, vol. 6, pp. 405–415, 2013.

[59] K. Thramboulidis, "Face to face; IEC 61499 function block model: Facts and fallacies," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 7–26, 2009.

[60] C. Sünder, A. Zoitl, J. Christensen, H. Steininger, and J. Fristshce, "Considering IEC 61131-3 and IEC 61499 in the context of component frameworks," in *Proc 6th IEEE International Conference on Industrial Informatics*, pp. 277–282, 2008.

[61] A. Zoitl and H. Prahofer, "Guidelines and Patterns for Building Hierarchical Automation Solutions in the IEC 61499 Modeling Language," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 4, pp. 2387–2396, 2011.

[62] C.-H. Yang, V. Vyatkin, and C. Pang, "Model-Driven Development of Control Software for Distributed Automation: A Survey and an Approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 3, pp. 292–305, 2014.

[63] J. H. Christensen, "Design patterns for systems engineering with IEC 61499," *Verteilte Automatisierung - Modelle und Methoden fur Entwurf, Verifikation, Engineering und Instrumentierung, Ch. Döschner, ed. Magdeburg, Germany: Otto-von-Guericke-Universität*, 2000.

[64] V. Vyatkin, "Software Engineering in Industrial Automation: State-of-the-Art Review," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234–1249, 2013.

[65] L. Ferrarini and C. Veber, "Implementation approaches for the execution model of IEC 61499 applications," in *Proc. 2nd IEEE International Conference on Industrial Informatics*, pp. 612–617, 2004.

[66] J. Lastra, L. Godinho, A. Lobov, and R. Tuokko, "An IEC 61499 application generator for scan-based industrial controllers," in *Proc. 3rd IEEE International Conference on Industrial Informatics*, pp. 80–85, 2005.

[67] P. Tata and V. Vyatkin, "Proposing a novel IEC 61499 runtime framework implementing the cyclic execution semantic," in *Proc. 7th IEEE International Conference on Industrial Informatics*, pp. 416–421, 2009.

[68] T. Strasser, A. Zoitl, J. Christensen, and C. Sünder, "Design and Execution Issues in IEC 61499 Distributed Automation and Control Systems," *IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews*, vol. 41, no. 1, pp. 41–51, 2011.

[69] G. Cengic, O. Ljungkrantz, and K. Akesson, "Formal Modeling of Function Block Applications Running in IEC 61499 Execution Runtime," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1269–1276, 2006.

[70] V. Vyatkin, "The IEC 61499 standard and its semantics," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 40–48, 2009.

[71] D. O'Sullivan and D. Heffernan, "VHDL architecture for IEC 61499 funcion blocks," *IET Computers and Digital Techniques*, vol. 6, no. 6, pp. 515–524, 2010.

[72] C. Sünder, A. Zoitl, J. Christensen, V. Vyatkin, R. Brennan, A. Valentini, L. Ferrarini, T. Strasser, J. Lastra, and F. Aunger, "Usability and Interoperability of IEC 61499 based distributed automation systems," in *Proc. IEEE International Conference on Industrial Informatics*, pp. 31–37, 2006.

[73] G. Doukas and K. Thramboulidis, "Implementation model alternatives for IEC 61499 Function Block Networks," in *Proc. 6th IEEE International Conference on Industrial Informatics*, pp. 295–300, 2008.

[74] K. Thramboulidis, G. Doukas, and T. Psegianakis, "An IEC-compliant field device model for distributed control applications," in *Proc. 2nd IEEE International Conference on Industrial Informatics*, pp. 277–282, 2004.

[75] C. Schwab, M. Tangermann, A. Lüder, A. Kalogeras, and L. Ferrarini, "Mapping of IEC 61499 function blocks to automation protocols within the TORERO approach," in *Proc. 2nd IEEE International Conference on Industrial Informatics*, pp. 149–154, 2004.

[76] S. Sierla, J. Peltola, and K. Koskinen, "Real-time middleware for the requirements of distributed process control," in *Proc. 3rd IEEE International Conference on Industrial Informatics*, pp. 1–6, 2005.

[77] F. Weehuizen, A. Brown, and C. S. y O. Hummer, "Implementing IEC 61499 communication with the CIP protocol," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 498–501, 2007.

[78] IEEE Computer Society, *IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges*, 2004.

[79] K. Lee and M. Lee, "Worst Case Communication Delay of Real-Time Industrial Switched Ethernet With Multiple Levels," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 5, pp. 1669–1676, 2006.

[80] A. Schimmel and A. Zoitl, "Real Time Communication for IEC 61499 in Switched Ethernet Networks," in *Proc. International Congress on Ultra Modern Telecommunications and Control Systems and Workshops*, pp. 406–411, 2010.

[81] W. Wagner, J. Bohl, and G. Frey, "An IEC 61499 interpretation and implementation focused on usability," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 184–191, 2008.

[82] M. Liu, *Distributing Computing: Principles and Applications*. Pearson Education, 2004.

[83] R. Froschauer, F. Auinger, G. Grabmair, and T. Strasser, "Automatic control aplication recovery in distributed IEC 61499 based automation and control systems," in *Proc. IEEE International Workshop on Distributed Intelligent Systems: Collective Intelligent and Its Applications*, pp. 103–108, 2006.

[84] A. Santos and M. de Sousa, "Replications in distributed systems using IEC 61499," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1–8, 2010.

[85] L. Lednicki and K. Sandstrom, "Device Utilization Analysis for IEC 61499 Systems in Early Stages of Development," in *Proc. 18th IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1–8, 2013.

[86] H. Prahofer and A. Zoitl, "Verification of Hierarchical IEC 61499 Component Systems with Behavioral Event Contracts," in *Proc. 11th IEEE International Conference on Industrial Informatics*, pp. 578–585, 2013.

[87] R. Hametner, I. Hegny, and A. Zoitl, "A Unit-Test Framework for Event-Driven Control Components Modeled in IEC 61499," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1–8, 2014.

[88] M. de Sousa, "Guaranteeing Replica Determinism on IEC 61499," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1–7, 2014.

[89] J. Chouinard, J. Lavalle, J. Lalibert, N. Landreaud, K. Thramboulidis, P. Bettez-Poirier, F. Desy, F. Darveau, N. Gendron, and C. Trang, "An IEC 61499 configuration with 70 controllers; challenges, benefits and a discussion on technical decisions," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, 2007.

[90] J. Yan and V. Vyatkin, "Distributed Execution and Cyber-Physical Design of Baggage Handling Automation with IEC 61499," in *Proc. 9th IEEE International Conference on Industrial Informatics*, pp. 573–578, 2011.

[91] R. Sinha, K. Johnson, and R. Calinescu, "A Scalable Approach for Re-Configuring Evolving Industrial Control Systems," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1–8, 2014.

[92] M. Kuo, L. Yoong, S. Andalam, and P. Roop, "Determining the worst-case reaction time of IEC 61499 function blocks," in *Proc. 8th IEEE International Conference on Industrial Informatics)*, pp. 1104–1109, 2010.

[93] L. Lednicki and J. Carlson, "Handling Cyclic Execution Paths in Timing Analysis of Component-based Software," in *Proc. 40th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 178–182, 2014.

[94] C. Schwab, M. Tangermann, and A. Lüder, "Integration of scalable safety and security actions in IEC 61499 control applications," in *Proc. 6th IFAC International Conference on Field bus Systems and Their Applications*, pp. 1–8, 2005.

[95] I. Delamer and J. Martinez, "Information security for reconfigurable manufacturing systems using networked embedded controllers," in *Proc. 12th IFAC International Symposium on Information Control Problems in Manufacturing*, pp. 131–136, 2006.

[96] T. Strasser, I. Mller, C. Sünder, and O. H. y H. Uhrmann, "Modeling of Reconfiguration Control Applications based on the IEC 61499 Reference Model for Industrial Process Measurement and Control Systems," in *Proc. IEEE International Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications*, pp. 127–132, 2006.

[97] W. Dai, V. Vyatkin, J. H. Christensen, and V. Dubinin, "Bridging Service-Oriented Architecture and IEC 61499 for Flexibility and Interoperability," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 771–781, 2015.

[98] C. Pang, S. Patil, C.-W. Y. V. Vyatkin, and A. Shalyto, "A Portability Study of IEC 61499: Semantics and Tools," in *Proc. 12th IEEE International Conference on Industrial Informatics*, pp. 440–445, 2014.

[99] T. Hussein and G. Frey, "Migration of a PLC Controller to an IEC 61499 Compliant Distributed Control System: Hands-on Experiences," in *Proc.*

IEEE International Conference on Robotics and Automation, pp. 3984–3989, 2005.

[100] C. Gerber, H. Hanisch, and S. Ebbinghaus, "From IEC 61131 to IEC 61499 for distributed systems: a case study," *EURASIP Journal on Embedded Systems*, pp. 1–8, Article ID 231630, 2008.

[101] M. Wenger, A. Zoitl, C. Sünder, and H. Steininger, "Transformation of IEC 61131-3 to IEC 61499 based on a model driven development approach," in *Proc. 7th IEEE International Conference on Industrial Informatics*, pp. 715–720, 2009.

[102] W. Dai and V. Vyatkin, "A case study on migration from IEC 61131 PLC to IEC 61499 function block control," in *Proc. 7th IEEE International Conference on Industrial Informatics*, pp. 79–84, 2009.

[103] A. Zoitl, T. Strasser, C. Snder, and T. Baier, "Is IEC 61499 in harmony with IEC 61131-3?," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 49–55, 2009.

[104] W. Dai, V. N. Dubinin, and V. Vyatkin, "Migration From PLC to IEC 61499 Using Semantic Web Technologies," *IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews*, vol. 44, no. 3, pp. 277–291, 2014.

[105] G. Black and V. Vyatkin, "Intelligent Component-Based Automation of Baggage Handling Systems With IEC 61499," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 2, pp. 337–351, 2010.

[106] V. Vlad, C. D. Popa, R. D. Pentiuc, and C. Buzduga, "Control Architecture for Power Distribution Systems Based on IEC 61850, IEC 61499 and Holonic Concepts," in *Proc. International Conference and Exposition on Electrical and Power Engineering*, pp. 132–136, 2014.

[107] G. Zhabelova, C.-W. Yang, S. Patil, C. Pang, J. Yan, A. Shalyto, and V. Vyatkin, "Cyber-Physical Components for Heterogeneous Modelling, Validation and Implementation of Smart Grid Intelligence," in *Proc. 12th IEEE International Conference on Industrial Informatics*, pp. 411–417, 2014.

[108] N. Kashyap, C.-W. Yang, S. Sierla, and P. G. Flikkema, "Automated Fault Location and Isolation in Distribution Grids With Distributed Control and Unreliable Communication," *IEEE Transactions on Industrial Informatics*, vol. 62, no. 4, pp. 2612–2619, 2015.

[109] A. Mousavi and V. Vyatkin, "Energy Efficient Agent Function Block: A semantic agent approach to IEC 61499 function blocks in energy efficient building automation systems," *Automation in Construction*, vol. 54, pp. 127–142, 2015.

[110] N. Hagge and B. Wagner, "Implementation alternatives for the OMAC state machines using IEC 61499," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 215–220, 2008.

[111] M. Minhat, V. Vyatkin, X. Xu, S. Wong, and Z. Al-Bayaa, "A novel open CNC architecture based on STEP-NC data model and IEC 61499 function blocks," *Robotics and Computer-Integrated Manufacturing*, vol. 25, no. 3, pp. 560–569, 2009.

[112] H. Xuemei, "Distributed and Reconfigurable Control of Lower Level Machines in Automatic Production Line," in *Proc. 8th World Congress on Intelligent Control and Automation*, pp. 2339–2344, July 2010.

[113] L. Wang, "Machine availability monitoring and machining process planning towards Cloud manufacturing," *CIRP Journal of Manufacturing Science and Technology*, vol. 6, no. 4, pp. 263–273, 2013.

[114] T. Peng, X. Xu, and L. Wang, "A novel energy demand modelling approach for CNC machining based on function blocks," *Journal of Manufacturing Systems*, vol. 33, no. 1, pp. 196–208, 2014.

[115] M. V. García, F. Pérez, I. Calvo, and G. Morán, "Building Industrial CPS with the IEC 61499 Standard on Low-cost Hardware Platforms," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1–4, 2014.

[116] L. Wang, "An overview of function block enabled adaptive process planning for machining," *Journal of Manufacturing Systems*, vol. 35, pp. 10–25, 2015.

[117] Agence nationale pour le DEveloppment de la Production Automatisée (ADEPA) France, *GEMMA (Guide d'Étude des Modes de Marches et d'Arrêts)*, 1981.

[118] M. Sorouri, S. Patil, and V. Vyatkin, "Distributed Control Patterns for Intelligent Mechatronic Systems," in *Proc. 10th IEEE International Conference on Industrial Informatics*, pp. 259–264, 2012.

[119] V. Vyatkin, M. Hirsch, and H.-M. Hanisch, "Systematic Design and Implementation of Distributed Controllers in Industrial Automation," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 633–640, 2006.

[120] Object Management Group, Inc, *UML Profile for Schedulability, Performance, and Time Specification*, version 1.1 ed., 2005.

[121] International Electrotechnical Commission, *GRAFCET specification language for sequential function charts, IEC 60848 Standard*, 2.0 ed., 2002.

[122] F. Serna, C. Catalán, A. Blesa, and J. Rams, "Design patterns for failure management in IEC 61499 function blocks.," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1–7, 2010.

[123] F. Serna, C. Catalán, A. Blesa, J. Colom, and J. Rams, "Predictive maintenance surveyor design pattern for machine tools control software applications," in *Proc. 16th IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 1–7, 2011.

[124] F. Serna, C. Catalán, A. Blesa, J. Rams, and J. Colom, "Preventive maintenance manager design pattern for component based machine tools," in *Proc. 9th IEEE International Conference on Industrial Informatics*, pp. 615–620, july 2011.

[125] F. Serna, C. Catalán, A. Blesa, J. Rams, and J. Colom, "Control software design for a cutting glass machine tool based on the COSME platform. Case study," in *Proc. IEEE International Conference on Automation Science and Engineering*, pp. 501–506, 2011.

[126] G. Cengic and K. Aesson, "On Formal Analysis of IEC 61499 Applications, Part B: Execution Semantics," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 2, pp. 145–154, 2010.

[127] W. E. Rumpl, F. Auinger, C. Dutzler, and A. Zoitl, "Platforms for Scalable Flexible Automation Considering the Concepts of IEC 61499," in *Proc. Fifth IFIP/IEEE International Conference on Information Technology for Balanced Automation Systems in Manufacturing and Services: Knowledge and Technology Integration in Production and Services: Balancing Knowledge in Product and Service Life Cycle*, pp. 237–246, 2002.

[128] V. Vyatkin, V. Dubinin, C. Veber, and L. Ferrarini, "Alternatives for Execution Semantics of IEC 61499," in *Proc. 5th IEEE International Conference on Industrial Informatics*, 2007.

[129] L. H. Yoong, P. Roop, V. Vyatkin, and Z. Salcic, "Synchronous Execution of IEC 61499 Function Blocks Using Esterel," in *Proc. 5th IEEE International Conference on Industrial Informatics*, pp. 1189–1194, 2007.

[130] V. Dubinin and V. Vyatkin, "Semantics-robust design patterns for IEC 61499," *IEEE Journal on Industrial Informatics*, vol. 8, no. 2, pp. 279–290, 2012.

[131] S. V. Baelen, D. Urting, W. V. Belle, V. Jonckers, T. Holvoet, Y. Berbers, and K. D. Vlaminck, "A Toward a unified terminology for component-based development," in *Proc. Fifth Workshop on component-oriented programming. European conference on object-oriented programming*, 2000.

[132] A. Burns and A. Wellings, *Real-Time Systems and Programming Languages Ada 95, Real-Time Java and Real-Time POSIX*. Addison Wesley Longmain, 2001.

[133] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the Association for Computing Machinery*, vol. 20, pp. 46–61, January 1973.

[134] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenstrom, "The worst-case execution-time problem overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, pp. 46–61, 2008.

[135] J. H. Brown and B. Martin, "How fast is fast enough? Choosing between Xenomai and Linux for real-time applications," in *Twelfth Real Time Linux Workshop*, 2010.

[136] K. Czarnecki and U. W. Eisenecker, *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., 2000.

[137] "TUROMAS, www.turomas.com (accedida en mayo 2015)."

[138] C. Sünder, A. Zoitl, T. Strasser, and J. Brunnenkreef, "Benchmarking of IEC 61499 runtime environments," in *Proc. IEEE International Conference on Emerging Technologies & Factory Automation*, pp. 474–481, 2007.