eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Departamento Ingeniería de Sistemas y Automática

Escuela de Ingeniería de Bilbao

PLATAFORMA DE GESTIÓN PARA APLICACIONES IIoT CON REQUISITOS DE CALIDAD DE SERVICIO

TESIS DOCTORAL

Don. Aitor Aguirre Andueza

Directoras: Marga Marcos Muñoz

Elisabet Estévez Estévez

Bilbao, Mayo de 2017

“Investigar es lo que hago cuando no sé lo que estoy haciendo.”

(Wernher von Braun)

AGRADECIMIENTOS

Sirvan estas líneas como sincero agradecimiento a todas aquellas personas que, de una u otra forma, han contribuido a la culminación exitosa de este trabajo de investigación.

Quisiera agradecer muy especialmente la orientación, el apoyo y la supervisión de la Dra. Marga Marcos y de la Dra. Elisabet Estévez, directora y codirectora de este trabajo de investigación. Su ayuda y consejo han sido cruciales para que este trabajo haya llegado a buen puerto.

Asimismo, quisiera dar las gracias a Ikerlan por el apoyo y los medios prestados para la realización de esta tesis. También quiero agradecer a mis compañeros los ánimos recibidos, y en especial el empuje y la ilusión de Juan Pedro Uribe.

Por último, debo agradecer a mi familia el tiempo prestado y el respaldo recibido, sin los cuales esta aventura no habría sido posible.

RESUMEN

El concepto de *Internet Industrial de las Cosas (IIoT)*, la versión industrial del término *Internet de las Cosas* acuñado por Kevin Ashton en 1999, engloba ideas y paradigmas que conforman una nueva visión de la interacción entre hombres y máquinas. Se contempla un mundo en el que cualquier elemento de ámbito industrial con capacidad de procesamiento pueda estar conectado a una red cibernética global e interactuar con el resto de elementos de forma más o menos autónoma. Esta interactividad, combinada con otras disciplinas como la robótica o la inteligencia artificial, vaticina un considerable impacto social en el futuro próximo.

Para sustentar el paradigma *IIoT* se identifican una serie de retos tecnológicos. Por una parte es necesario gestionar aplicaciones geográficamente distribuidas y muy heterogéneas en cuanto a plataformas hardware, redes y protocolos de comunicación. Por otra parte, la naturaleza de las aplicaciones es intrínsecamente dinámica, y por tanto requiere de soporte para la reconfiguración dinámica y autónoma de los sistemas. Finalmente, existen una serie de requisitos no funcionales que son claves desde un enfoque industrial y que contemplan aspectos tales como la calidad de servicio, la tolerancia a fallos o la seguridad funcional.

En este contexto, desde la perspectiva de la ingeniería del software se identifica una oportunidad de investigación que supone la motivación de esta tesis. Con el objetivo de facilitar el desarrollo y soporte de aplicaciones *IIoT*, se ha concebido una plataforma de gestión de aplicaciones distribuidas basadas en componentes, que soporta la reconfiguración dinámica y autónoma de las mismas en base a criterios de optimización de los recursos y de calidad de servicio. La plataforma soporta una serie de paradigmas de comunicación y modelos de ejecución que abarcan una amplia tipología de aplicaciones. Para su validación, se ha diseñado y desarrollado un demostrador en el campo de los almacenes automatizados.

ABSTRACT

The concept of *Industrial Internet of Things* (IIoT), a specialization of the *Internet of Things* term coined by Kevin Ashton in 1999, encompasses ideas and paradigms that propose a new vision of the relationship between humans and machines. It contemplates a world in which any industrial device with processing capability can be connected to a global cybernetic network and can interact autonomously with other devices. This extensive connectivity, combined with other disciplines such as robotics or artificial intelligence, predicts a considerable social impact in the near future.

To support the IIoT paradigm several technological challenges must be addressed. On the one hand, the considered applications are geographically distributed and heterogeneous in terms of hardware platforms, networks and communication protocols. On the other hand, the nature of the applications is intrinsically dynamic, and therefore, support for dynamic and autonomous system reconfiguration is required. Finally, there are several non-functional key requirements from an industrial point of view. These requirements include aspects such as quality of service, fault tolerance or functional safety.

In this context, a research opportunity has been identified from the software engineering perspective. To facilitate the development and runtime support of IIoT applications, a platform which manages applications based on distributed components has been designed. It drives the dynamic and autonomous system reconfiguration based on resource and QoS optimization criteria. The platform supports several communication paradigms and execution models that cover a wide range of applications. For its validation, a case study in the field of automated warehouses has been designed and developed.

ÍNDICE

ÍNDICE DE CONTENIDOS

Índice de figuras

Índice de Tablas

1 INTRODUCCIÓN

1.1 Motivación

En las últimas dos décadas han surgido diversos paradigmas de computación distribuida que comparten características similares y que presentan retos comunes. Los sistemas basados en redes de sensores [1], los sistemas ciber-físicos (CPS, *cyber-physical systems*) [2] o la Internet de las Cosas (IoT, *Internet of Things*) [3] exhiben una serie de características comunes tales como la distribución geográfica, la heterogeneidad de redes, hardware y protocolos, la interoperabilidad o su naturaleza intrínsecamente dinámica. Asimismo, aspectos como la calidad de servicio (QoS, *Quality of Service*) y la gestión de recursos son particularmente relevantes en este tipo de sistemas, ya que afectan a aspectos clave tales como la experiencia de usuario, la confiabilidad o la eficiencia energética [4-6].

Por otra parte, la constante evolución tecnológica y el continuo abaratamiento de las redes de transmisión de datos, los nodos de procesamiento, sensores, y la electrónica en general, están haciendo que la idea de un mundo más inteligente basado en una red cibernética global con millones de dispositivos interconectados esté cobrando cada vez más fuerza en la industria.

El presente trabajo de investigación se enfoca precisamente en la versión industrial del internet de las cosas (IIoT, *Industrial Internet of Things*) [7], que consiste en la aplicación del concepto IoT a las tecnologías de fabricación avanzada y control industrial y se basa fundamentalmente en la automatización de la captura, comunicación y posterior tratamiento de datos de proceso provenientes de distintas máquinas. Combinando sensores, redes telemáticas y tecnologías avanzadas de análisis de datos y toma de decisiones, se observa un gran potencial en términos de control de calidad, ahorro energético, gestión logística o monitorización y trazabilidad de los procesos [8]. Esto permitiría que las empresas se centren en la resolución de problemas de alto nivel y por tanto redundaría en un ahorro de costes y disminución de las ineficiencias.

El concepto de IIoT es la base fundamental de la *fábrica inteligente*, que contempla un gran número de sistemas industriales interconectados que se coordinan para incrementar el rendimiento y la eficiencia de una instalación, disminuyendo o evitando en la medida de lo posible las paradas de producción. Un ejemplo ilustrativo de esto puede ser un equipo industrial capaz de prever fallos en sus componentes a partir del procesamiento de datos de distintos sensores, y actuar proactivamente planificando el mantenimiento preventivo de dicho componente a través de la red empresarial. De forma similar, en el ámbito del *hogar inteligente*, un ascensor puede comunicar averías, en tiempo real, a un centro de atención remoto, que a su vez redirija la incidencia al teléfono móvil de un técnico de mantenimiento que se encuentre en ese momento en una zona próxima al edificio.

Esta tipología de aplicaciones distribuidas, aunque de naturaleza muy diversa, exhiben una serie de características comunes y demandan una serie de necesidades cuya resolución constituye el reto y motivación de esta investigación. Se trata de **aplicaciones basadas en componentes software** distribuidos que se despliegan sobre una infraestructura de **nodos heterogéneos** (en términos de hardware y sistema operativo). Estos componentes de aplicación pueden ser diversos en cuanto a lenguaje de implementación, y pueden comunicarse mediante distintos protocolos de comunicación y niveles de seguridad. Asimismo, plantean la necesidad de responder con un determinado nivel de calidad de servicio en un escenario en el que deben compartir con otras aplicaciones los recursos hardware y software, que son limitados. En resumen, las aplicaciones objetivo de este trabajo de investigación demandan una serie de necesidades entre las que destacan las siguientes:

- **Distribución geográfica**. Los componentes que conforman una aplicación pueden estar desplegados en nodos geográficamente distribuidos.

- **Heterogeneidad e interoperabilidad**. Diversidad de plataformas hardware, sistemas operativos, lenguajes de implementación y protocolos de comunicación.

- **Escalabilidad**. La infraestructura hardware/software del sistema, con recursos limitados, debe poder alojar un número creciente de aplicaciones en base a la adición de nuevos recursos.

- **Servicios remotos de monitorización y control**. Gestión global y centralizada de las aplicaciones distribuidas, considerando la instalación inicial, la monitorización de los componentes de aplicación, así como el mantenimiento y actualización de los mismos.

- **Disponibilidad**. Algunas aplicaciones pueden requerir alta disponibilidad (24 horas, 365 días), incluso en fase de mantenimiento o en caso de fallo de algunos de sus componentes.

- **Seguridad funcional**. Las aplicaciones críticas como pueden ser las de salud o transporte demandan mecanismos de comunicación confiables que deben garantizar la transmisión segura de datos entre los componentes de aplicación, detectando cualquier fallo o error que pueda llevar a una operación no segura.

- **Calidad de servicio**. Las aplicaciones se ejecutan sobre una infraestructura compartida y con recursos limitados, que deben ser gestionados adecuadamente para satisfacer el grado de calidad de servicio demandado por las aplicaciones.

- **Reconfiguración dinámica.** Los sistemas pueden ser dinámicamente reconfigurados por distintos motivos, ya sean fallos de hardware o software, actualización de componentes, o instalación de nuevas aplicaciones sobre la misma infraestructura hardware. En el ámbito de este trabajo se entiende la reconfiguración dinámica como un cambio en caliente (es decir, sin recurrir a la parada del sistema) de la topología del sistema, entendida ésta como una distribución específica de los componentes software de aplicación sobre los nodos hardware que conforman la infraestructura.

- **Código heredado.** Muchas de las aplicaciones IIoT pueden requerir de la integración de software antiguo dentro del nuevo paradigma. El facilitar dicho proceso de migración puede suponer un ahorro de costes considerable.

Para resolver esta problemática se han propuesto diferentes tecnologías. Desde la perspectiva de la ingeniería del software se han desarrollado los marcos de componentes [9, 10] y las arquitecturas orientadas a servicios (SOA, *Service Oriented Architecture*) [11], que persiguen la reutilización de código y el desacoplamiento de sistemas. Algunos marcos de componentes soportan diferentes lenguajes de programación y, apoyándose en la evolución de las tecnologías de middleware de comunicación, proporcionan los niveles requeridos de abstracción y transparencia en la distribución de datos. Asimismo, apoyándose en otros paradigmas como la programación orientada a aspectos (AOP, *Aspect Oriented Programming*) [12], permiten la integración modular de funcionalidades transversales tales como la seguridad, las transacciones o el registro de trazas de proceso. Sin embargo, otros requisitos adicionales como por ejemplo la reconfiguración dinámica o el aseguramiento de la calidad de servicio han hecho patente la necesidad de integrar otras tecnologías.

En este trabajo de investigación se pretende proporcionar una solución adecuada a las necesidades de las aplicaciones previamente expuestas, en términos de soporte en tiempo de ejecución (*runtime*) de las mismas. Este soporte comprende diferentes fases del ciclo de vida de las aplicaciones, desde el registro inicial, pasando por el despliegue, el control de ejecución y la desinstalación final, poniendo especial énfasis en la gestión de los recursos y de la calidad de servicio de las aplicaciones. Asimismo, el soporte a la migración de código heredado se contempla como un gran valor añadido, debido a que las aplicaciones consideradas cuentan con una cantidad considerable de código ya desarrollado, y por tanto el facilitar la integración de dicho código en la nueva arquitectura a desarrollar es un aspecto a tener muy en cuenta a la hora de abaratar costes.

Un ejemplo enmarcado dentro del contexto industrial de internet de las cosas (IIoT) y que sirve para identificar los retos que motivan esta tesis es el de los almacenes automáticos.

Los almacenes automáticos modernos presentan una serie de características y requieren de una serie de servicios que los convierten en un claro ejemplo de IIoT enfocado a la automatización industrial. Habitualmente están compuestos por cientos de componentes software distribuidos que se ejecutan sobre decenas de nodos hardware heterogéneos y que se comunican mediante diversos protocolos. Estos componentes necesitan ser monitorizados y controlados en tiempo real para poder agilizar el diagnóstico de fallos eventuales que de otra forma serían complicados de detectar e identificar. Asimismo, los almacenes automáticos demandan servicios como por ejemplo el registro y despliegue de nuevas aplicaciones en caliente o la tolerancia a fallos de ciertos componentes críticos. En este sentido, el aseguramiento de la integridad y consistencia del estado del almacén en caso de fallo requiere de ciertas funcionalidades. Por una parte, se debe garantizar la seguridad (funcional) de las comunicaciones entre los distintos componentes distribuidos del almacén. Por otra parte, se han de habilitar mecanismos que permitan restaurar íntegramente el estado previo al fallo de los componentes, compuesto por un conjunto de variables que incluyen, entre otras muchas, las órdenes de preparación de pedidos (*picking*) en curso o la ubicación de las mercancías.

Todos estos servicios automatizados de monitorización, control, redundancia y tolerancia a fallos con mantenimiento íntegro del estado (*stateful fault tolerance*) redundan en una disminución de los tiempos de parada por mantenimiento, aumentan la disponibilidad del sistema y en definitiva suponen un considerable ahorro de costes de explotación.

1.2 Objetivos

El objetivo del presente trabajo de investigación se centra en el diseño conceptual y desarrollo de una plataforma software (*middleware*) que proporcione los servicios necesarios para habilitar la gestión, en tiempo de ejecución, del ciclo de vida de aplicaciones basadas en componentes software físicamente distribuidos en nodos hardware heterogéneos. Para lograr este objetivo principal, se han identificado varios objetivos parciales que responden a las demandas de las aplicaciones expuestas en la motivación:

Soporte en tiempo de ejecución. Debe contemplar el registro, despliegue, control de ejecución y desinstalación final de las aplicaciones.

Distribución de datos e interoperabilidad. Los componentes de aplicación pueden requerir de diversos paradigmas (cliente-servidor, publicador-suscriptor) y protocolos de comunicación (servicios web, REST, DDS, etc). En este sentido, el objetivo básico que se persigue es la transparencia en la distribución de datos, haciendo que, de forma ideal, el implementador de un componente se centre en la parte funcional y sea el middleware el que proporcione los mecanismos necesarios para que la comunicación entre componentes pueda realizarse con diferentes protocolos.

Confiabilidad. Este objetivo se enfoca desde dos puntos de vista: Por una parte, desde el punto de vista de la seguridad, deben soportarse canales de comunicación que aseguren la integridad de los datos (*safety*). Por otra parte, desde el punto de vista de la disponibilidad, deben proporcionarse mecanismos de tolerancia a fallos de componentes software, ya sean debidos a caídas de los nodos hardware que los alojan o a fallos del propio software. Además, debe asegurarse la restauración del estado de la aplicación previo al fallo.

Escalabilidad. En este aspecto deben proporcionarse los mecanismos para que el sistema global, formado por múltiples aplicaciones, pueda escalarse en base a la adición de más recursos hardware. Esto implica que la ejecución de los componentes de aplicación individuales pueda ser transferida dinámicamente de un nodo hardware a otro, ya sea por motivos de disponibilidad en caso de fallo o por motivos de balance de carga en caso de pérdida de rendimiento.

Gestión de recursos y aseguramiento de la calidad de servicio. Las aplicaciones demandan un nivel de calidad de servicio que debe ser garantizado en la medida de lo posible en base a un criterio de prioridad y de disponibilidad de recursos. Esta gestión de recursos es clave para la consecución de otros objetivos anteriormente citados, como son el balance de carga y la tolerancia a fallos, e implica que deben establecerse mecanismos de soporte a la reconfiguración dinámica del sistema.

Soporte a sistemas heredados. Se debe facilitar la integración de software heredado en la nueva arquitectura. En general, la reutilización de código es uno de los *leit motiv* de la ingeniería de software basada en componentes (CBSE, *Component Based Software Engineering*). En este trabajo este requisito es crucial, ya que se pretende integrar software ya desarrollado.

Funcionalidad ampliable. La plataforma de gestión debe permitir ampliaciones futuras de su funcionalidad de forma ágil y sencilla.

Soporte a la generación automática de código. El objetivo que se persigue es el diseño de un patrón de implementación que permita generar automáticamente el código no funcional de un componente de aplicación, aumentando así la productividad del desarrollador y disminuyendo la probabilidad de errores de implementación, lo cual supone un claro aporte de valor. Este código no funcional comprende los atributos y el esqueleto de las funciones que debe implementar el componente para su funcionamiento interno y la comunicación con el resto de componentes de aplicación y la plataforma.

Cabe señalar que este último objetivo está íntimamente ligado con la disponibilidad de un conjunto de herramientas de diseño que permitan la generación automática de código a partir del modelo de componente que se presenta en este trabajo. Estas herramientas de diseño se han desarrollado en el contexto de otra tesis, englobada dentro de la misma línea de investigación del Grupo de Control e Integración de Sistemas (GCIS) del Departamento de Ingeniería de Sistemas y Automática de la Universidad del País Vasco, y con la que el presente trabajo se relaciona.

1.3 Estructura

La estructura del resto del documento es la siguiente: El Capítulo 2 – **Estado del arte** – se divide en dos grandes bloques que agrupan conceptos bien diferenciados. Por una parte se estudian una serie de marcos de componentes (*component frameworks*) existentes en la literatura, con objeto de seleccionar el marco más adecuado para los objetivos marcados en la Sección 1.2. Este análisis es especialmente importante para los objetivos relativos al soporte en tiempo de ejecución, distribución de datos, código heredado y herramientas de generación automática de código, ya que éstos son aspectos generalmente cubiertos, ya sea parcial o totalmente, por los marcos de componentes existentes. Por otro lado, también se presentan algunos conceptos relativos a la reconfiguración dinámica de aplicaciones distribuidas, y se analizan algunos trabajos de investigación centrados en sistemas dinámicamente reconfigurables y con consideraciones de calidad de servicio.

El Capítulo 3 – **Escenario general** – presenta el modelo formal del sistema, compuesto por una infraestructura de nodos hardware y las aplicaciones distribuidas que se ejecutan sobre ellos. Asimismo, se definen los requisitos de detalle que guiarán el diseño de la plataforma de gestión, obtenidos a partir de los objetivos establecidos en la Sección 1.2.

El Capítulo 4 – **Arquitectura de la plataforma DAMP** – presenta los elementos principales que componen la arquitectura de la plataforma así como el conjunto de

servicios que proporcionan, y se indica cómo esta arquitectura resuelve parte de los requisitos de detalle definidos en el Capítulo 3.

El Capítulo 5 – **Diseño de la plataforma DAMP** –detalla el diseño que posibilita la implementación de la plataforma. Asimismo, se justifica la cobertura de los requisitos que no se cubren en base a la arquitectura.

El Capítulo 6 – **Caso de estudio: almacén automático** – presenta un caso de estudio que cubre distintos requisitos de la plataforma DAMP. Centrado en el ámbito de los almacenes automáticos, se focaliza en aspectos de escalabilidad y rendimiento de la plataforma. Asimismo, incide en aspectos de tolerancia a fallos con recuperación íntegra del estado distribuido, a través de canales funcionalmente seguros.

En el Capítulo 7 – **Conclusiones y líneas futuras** - se detallan las conclusiones y aportaciones del trabajo, así como las publicaciones a las que ha dado lugar. Finalmente, se apuntan algunas líneas de investigación que podrían abordarse en futuros trabajos.

2 ESTADO DEL ARTE

2.1 Introducción

IoT es un concepto muy amplio que puede enfocarse desde múltiples perspectivas, que abarcan desde las infraestructuras de computación en la nube (*cloud computing*) y tecnologías *big data* [13-15], hasta protocolos de comunicaciones, plataformas hardware y plataformas software (*middleware*) específicamente ideadas para el desarrollo de aplicaciones IoT [16, 17]. Es por ello por lo que a continuación se realiza una breve introducción al estado del arte desde una perspectiva global, y posteriormente las secciones 2.2 y 2.3 se centran en el estado del arte de aquellos aspectos concretos que atañen a los objetivos planteados en la introducción.

Las infraestructuras de computación en la nube [18] surgieron inicialmente con una clara orientación a aplicaciones empresariales. Se presentaban como una alternativa competitiva a un enfoque tradicional que consideraba la infraestructura informática empresarial como un activo que la propia empresa debía adquirir y mantener localmente. En este sentido, los conceptos de Infraestructura como Servicio (IaaS, *Infrastructure as a Service)* [19] y Software como Servicio (SaaS, *Software as a Service*) [20] proponían la deslocalización y externalización de la infraestructura hardware y de los servicios informáticos (bases de datos, servidores web, etc.) de una empresa, y ofrecían tres ventajas principales: (1) eliminación de la necesidad de inversiones iniciales elevadas en infraestructura, (2) disminución de los costes de mantenimiento de dicha infraestructura y (3) garantías en aspectos clave como la seguridad, escalabilidad, respaldo de datos o la disponibilidad de servicio. Actualmente, estas infraestructuras de computación en la nube han evolucionado y se han adaptado a nuevos requerimientos, para convertirse en uno de los pilares fundamentales del paradigma IoT [13, 21]. Ejemplos de tal evolución son las plataformas Particle [22], Cayenne [23] o IFTTT [24], que permiten la gestión centralizada de aplicaciones IoT (desarrollo, despliegue, control remoto), así como la integración con otras plataformas y fabricantes de dispositivos.

Aunque ya están disponibles decenas de plataformas IoT basadas en la nube, el mercado para este tipo de aplicaciones continúa siendo pequeño, y se limita a aplicaciones con requisitos de rendimiento poco exigentes. Precisamente, este tipo de requisitos, junto con otros relativos a la interoperabilidad, confiabilidad, o calidad de servicio, son clave en ciertas aplicaciones industriales, y por tanto pueden limitar la adopción del paradigma IoT en su versión industrial (IIoT). Para que las aplicaciones IIoT alcancen todo su potencial se hace patente la necesidad de integrar otras tecnologías aparte de las centradas en computación en la nube [25]. En este sentido cabe destacar, por ser de especial interés para los objetivos del presente trabajo de investigación, el paradigma de computación conocido como *fog computing*, que se sitúa a medio camino entre el *cloud computing* y la computación local en el dispositivo embebido distribuido (*edge computing*) [26-28]. En un escenario en el que se prevén cientos de millones de dispositivos conectados a Internet, la aproximación *fog computing* se basa en el uso de dispositivos que hagan de puerta de enlace entre la red local y la red de área extensa. Esta es una opción viable para gestionar esta enorme cantidad de nodos, ya que estos dispositivos "de enlace", con una capacidad de computación media, pueden utilizarse para almacenar y procesar datos de forma local, liberando recursos de la nube y limitando el uso de ancho de banda de la red troncal de área extensa. Asimismo, aplicaciones de control con restricciones temporales o de calidad de servicio, pueden beneficiarse de la capacidad de operación autónoma y de las menores latencias que el *fog* ofrece frente al *cloud*. En este sentido, el *fog computing* proporciona la escalabilidad necesaria para soportar aplicaciones IIoT que demandan requisitos de calidad de servicio y además requieren de conectividad con la nube.

Otro aspecto de vital importancia para habilitar el desarrollo de aplicaciones IoT es el relativo al middleware de distribución de datos. En los últimos años hemos asistido al desarrollo de numerosos protocolos de comunicación enfocados a aplicaciones IoT. Estos protocolos consideran entre sus principales objetivos la escalabilidad, la seguridad o la eficiencia energética, y sobre todo persiguen la ligereza de las implementaciones, ya que pueden ser embebidos en dispositivos con recursos muy

limitados. Por citar algunos ejemplos, MQTT [29], ZeroMQ [30] o CoAP [31, 32] son protocolos de extrema ligereza y gran eficiencia en términos de consumo energético y ancho de banda. Otro protocolo que originalmente era algo más pesado pero que ha sabido adaptarse a los requisitos de las aplicaciones IoT es DDS [33], que ya cuenta con versiones orientadas a dispositivos embebidos con escasos recursos [34]. Este protocolo añade, a las ventajas de los anteriormente expuestos, soporte para un amplio abanico de parámetros de calidad de servicio, como por ejemplo los relativos a latencia, prioridad, consumo de ancho de banda o comunicación confiable diferida.

Para resolver la problemática asociada a los objetivos planteados en la introducción se han propuesto diferentes tecnologías. Desde la perspectiva de la ingeniería del software se han desarrollado los marcos de componentes y las arquitecturas orientadas a servicios (SOA, *Service Oriented Architecture*), que persiguen la reutilización de código y el desacoplamiento de sistemas. Algunos marcos de componentes soportan diferentes lenguajes de programación y, apoyándose en la evolución de las tecnologías de middleware de comunicación, proporcionan los niveles requeridos de abstracción y transparencia en la distribución de datos. Sin embargo, otros requisitos adicionales como por ejemplo la reconfiguración dinámica [35, 36] o el aseguramiento de la calidad de servicio [37] han hecho patente la necesidad de integrar otras tecnologías.

El resto del capítulo se divide en dos bloques principales, que cubren los trabajos relacionados con los objetivos contemplados en la introducción. En la Sección 2.2 se realiza un recorrido por distintos marcos de componentes existentes en la literatura, y que en general tratan de dar una solución a los aspectos relativos al soporte en tiempo de ejecución, distribución de datos, interoperabilidad, herramientas y metodología de desarrollo. También se considera el soporte de propiedades no funcionales que proporcionan, que cobra especial relevancia a la hora de plantear los objetivos de confiabilidad y soporte a sistemas heredados.

Asimismo, otro aspecto que se ha considerado de especial relevancia es la disponibilidad de implementaciones de código abierto (open source) para los marcos de componentes analizados, así como su nivel de madurez y documentación disponible. En este sentido, se han eliminado de la comparativa aquellos marcos que o bien no están disponibles para uso inmediato, o carecen de documentación, o el soporte de herramientas no alcanza el mínimo necesario para experimentar con ellos (p.ej. ROBOCOP [38] o AUTOSAR [39]). Tampoco se consideran marcos que no ofrecen soporte para sistemas distribuidos, como por ejemplo PECOS [40], Pin [41], Koala [42] o BlueArX [10]. Por tanto, de cara a realizar un análisis comparativo, se han seleccionado ocho marcos que persiguen objetivos similares y que cubren un mayor número de necesidades planteadas en la introducción. No obstante en algunos apartados se citan otros marcos que puedan tener interés en relación a la característica concreta analizada en el apartado.

En la Sección 2.3 se analizan distintas aproximaciones a la problemática que plantean los objetivos relativos a la escalabilidad, gestión de recursos y aseguramiento de la calidad de servicio, que generalmente no son considerados por los marcos de componentes de propósito general, o bien lo son de forma parcial.

2.2 Marcos de componentes

La ingeniería del software basada en componentes (CBSE) ha demostrado ser un paradigma útil a la hora de enfrentar problemas típicos en el sector del software, como por ejemplo productividad, mantenimiento o escalabilidad [43-45]. Esto ha motivado el desarrollo de numerosos marcos de componentes, basados en diversas tecnologías, enfocados a distintos dominios [9], y con muy diverso grado de soporte al desarrollo.

Un **modelo de componente** define una serie de estándares para la implementación, nomenclatura, interoperabilidad, configuración, composición, evolución y despliegue del componente [46]. Un **marco de componentes** (*component framework*)

generalmente mapea un modelo de componente a uno o varios lenguajes de implementación, y añade un conjunto de herramientas y métodos que pretenden facilitar la gestión del ciclo de vida de las aplicaciones, desde la especificación y diseño inicial, pasando por el desarrollo, despliegue y ejecución, hasta el mantenimiento y desinstalación final [10]. Sin embargo, no todos los marcos de componentes cubren todas las fases ni proporcionan herramientas de soporte para todas ellas. Algunos modelos presentan un nivel de abstracción elevado (p.ej. Fractal [47-49]), mientras que otros llegan a especificar aspectos como por ejemplo los protocolos de comunicación a soportar, tal y como se analiza en [50]. En cuanto al soporte de requisitos no funcionales, algunos modelos de componente orientados a tiempo real permiten modelar atributos específicos de tiempo real como el periodo, tiempo de cómputo o plazo, de cara a poder realizar análisis de planificabilidad [10]. Por otra parte, algunos modelos de componente están diseñados para un dominio de aplicación específico, por ejemplo AUTOSAR [39] y BlueArX [51] en automoción, OROCOS [52] en robótica, o IEC61499 [53] en automatización industrial, mientras que otros cubren un abanico más amplio de aplicaciones y requisitos (p.ej. Fractal [54], SOFA [55] o SCA [56]).

2.2.1 Soporte al ciclo de vida

Esta sección se centra en el soporte al ciclo de vida que proporcionan los distintos marcos de componentes analizados, desde el punto de vista de los siguientes objetivos planteados en la introducción: (1) soporte en tiempo de ejecución, (2) soporte a la generación automática de código y (3) migración de sistemas heredados. La Tabla 1 resume las características de los marcos de componentes seleccionados respecto al soporte al ciclo de vida.

Para facilitar la generación automática de código, es importante que el marco de componentes considere la fase de diseño, con conceptos como por ejemplo lenguajes declarativos para la definición de la arquitectura (ADL, *Architecture Description Languaje* [57]) o anotaciones en el código, ya que hacen mucho más sencilla la implementación de herramientas de generación automática de código. Este apartado

se refleja en la columna "Especificación del diseño" de la Tabla 2.1, en la cual se indica si el marco de componentes ofrece los mecanismos suficientes para habilitar la generación automática de código mediante aplicaciones externas. Por ejemplo, Palladio [58] y KobrA [59] utilizan perfiles UML en la fase de especificación del diseño, mientras que Fractal [48] o SCA [56] tienen su propio lenguaje de modelado ADL y ofrecen la posibilidad de anotar el código. Por otra parte, SOFA [56] propone su propio meta-modelo y el estándar IEC61499 [60] define un conjunto de modelos a nivel de sistema (dispositivos, recursos y aplicaciones) y propone los diagramas de bloques funcionales (FBD, *Function Block Diagram*) como herramienta de modelado de las aplicaciones.

Tabla 2.1. Soporte al ciclo de vida

Marco de componentes	Especificación del diseño	Soporte multilenguaje	Entorno de ejecución
Fractal	ADL, anotaciones	Si	Si
SCA	ADL, anotaciones	Si	No
CCM	-	Si	Si
OSGi	-	Java	Si
EJB	-	Java	Si
SOFA	ADL	Java	Si
OROCOS	-	C/C++	Si
IEC61499	Metamodelos de dispositivos y recursos. Modelado de las aplicaciones con diagramas FBD	LD, FBD, ST, IL, SFC	Si

Por otra parte, la migración de software heredado exige que el marco de componentes ofrezca soporte multilenguaje, ya que esto reduce el esfuerzo necesario para migrar aplicaciones heterogéneas que pueden estar compuestas por componentes software implementados en distintos lenguajes. De esta forma la migración se limita a encapsular el software existente en el envoltorio de componente ofrecido por el marco. En este sentido cabe destacar que aunque algunos modelos de componentes se declaran como agnósticos o independientes del lenguaje de implementación (p.ej CCM [61] o SCA [56]), en la práctica existen implementaciones de dichos estándares para ciertos lenguajes concretos. Así, para que una distribución sea considerada conforme al estándar SCA, debe soportar mínimamente los lenguajes

C/C++, BPEL, Java y Spring. Este es el caso de Apache Tuscany [62], que ofrece una implementación de código abierto para el estándar SCA [56]. En el caso de CCM [61], se definen mapeos del estándar a distintos lenguajes, que luego pueden ser implementados por distintas distribuciones como por ejemplo OpenCCM [63], que proporciona una implementación de código abierto para el estándar CCM sobre Java. Este es también el caso de Fractal, del que existen diversas implementaciones para distintos lenguajes como por ejemplo Julia (Java) [64], Think (C/C++) [65], FracNet (.NET) [66] o Julio (Phyton) [67].

Finalmente, en relación al soporte en tiempo de ejecución, algunos marcos solo consideran el concepto de componente desde la fase de diseño hasta la fase de compilación, y no ofrecen un entorno de ejecución (*runtime*). Dicho de otra forma, los componentes se integran en fase de compilación y desaparecen como entidad independiente cuando comienza su ejecución. Esto hace que la arquitectura de la aplicación sea estática y que no se puedan controlar los componentes en tiempo de ejecución, haciendo inviable la reconfiguración dinámica del sistema. En contrapartida, esta escasa flexibilidad en tiempo de ejecución permite sistemas más predecibles, característica de gran importancia para los marcos más especializados, como por ejemplo los enfocados a automoción (AUTOSAR [39], BlueArX [10]). Sin embargo otros marcos de componentes como por ejemplo CCM [61], EJB [68], Fractal [48] o [69] permiten la gestión (adición o sustitución) de componentes en tiempo de ejecución. Todos ellos introducen conceptos similares, como por ejemplo la separación de aspectos, la diferenciación entre interfaces funcionales y de control, componentes activos y pasivos o el concepto de contenedor o membrana.

2.2.2 Distribución de datos e interoperabilidad

En esta sección se consideran una serie de conceptos directamente relacionados con los objetivos de distribución de datos e interoperabilidad. En relación a la distribución de datos se analiza el soporte nativo para distintos protocolos de comunicación, así como la facilidad de integración de otros protocolos. En relación a

la interoperabilidad con otros sistemas, se analiza el soporte que los distintos marcos proporcionan en relación a los siguientes conceptos:

Tipos de interface: Los componentes pueden ofrecer interfaces definidos mediante un conjunto de operaciones (funciones) con parámetros. También pueden presentar interfaces que se basan en la definición de ciertas estructuras de datos que se transfieren entre los puertos de salida y entrada de los componentes. Los basados en operaciones se utilizan típicamente en arquitecturas orientadas a servicios tipo cliente servidor, mientras que los basados en flujo de datos son característicos de aplicaciones de control o *streaming,* donde el flujo de datos entre componentes es unidireccional. Como ejemplos de modelos de componentes que definen sus interfaces mediante operaciones se pueden citar EJB [68] y OSGi [70]. Entre los marcos que soportan interfaces basados en flujo de datos tenemos OROCOS [52] o IEC61499 [60]. Otros marcos como Fractal [48] o Pin [41] admiten distintos estilos de comunicación.

Paradigma de comunicación: La mayoría de los marcos analizados soportan el esquema de comunicación cliente/servidor (petición / respuesta), que implica tráfico de datos bidireccional en los enlaces entre componentes. Asimismo, algunos marcos ofrecen otros patrones de interacción como por ejemplo la comunicación mediante colas de mensajes, eventos, o mediante publicación/suscripción.

Tipo de comunicación: Existen dos variantes dentro del esquema de comunicación petición / respuesta. En la comunicación asíncrona, el cliente que inicia la comunicación continúa su ejecución sin esperar a la respuesta, que llegará en un momento posterior. La comunicación síncrona implica que el cliente detiene su ejecución hasta recibir respuesta del componente servidor.

En la Tabla 2.2 se resume el soporte que ofrecen los distintos marcos seleccionados en relación a la distribución de datos e interoperabilidad. Los componentes de una aplicación distribuida necesitan intercomunicarse mediante distintos paradigmas y

protocolos de comunicación, y por tanto uno de los principales objetivos de un marco de componentes es la abstracción de los protocolos de comunicación de forma que el esfuerzo de desarrollo se concentre en los aspectos funcionales de la aplicación.

En este sentido, algunos de los marcos de componentes analizados no proporcionan de forma nativa conectores (*bindings*) para protocolos concretos, dado su elevado nivel de abstracción. Este es por ejemplo el caso de Fractal [48]. Por el contrario, puede observarse que el estándar SCA [56] destaca en el apartado de interoperabilidad al proporcionar un amplio abanico de protocolos de comunicación, lo cual facilita enormemente la integración de nuevos componentes en sistemas heredados ya existentes. Por ejemplo, si ya se cuenta con un servidor empresarial que ofrece servicios web, el consumir dichos servicios desde un nuevo componente SCA es sencillo, ya que SCA proporciona un *binding* para dicho protocolo de forma nativa.

Tabla 2.2. Distribución de datos e interoperabilidad

Marco de componentes	Tipo de interface	Paradigma de comunicación	Tipo de comunicación	Protocolos nativos
Fractal	Basado en operaciones	Múltiple	Síncrona Asíncrona	No
SCA	Basado en operaciones	Petición respuesta Colas Mensajes	Síncrona Asíncrona	SOAP-WS [71], JMS [72], JCA [73], RMI [74]
CCM	Basado en operaciones Basado en puertos	Petición-respuesta Eventos	Síncrona Asíncrona	CORBA [75]
OSGi	Basado en operaciones	Petición respuesta	Síncrona	DOSGi [70]
EJB	Basado en operaciones	Petición respuesta	Síncrona Asíncrona	RMI
SOFA	Basado en operaciones	Múltiple	Síncrona Asíncrona	CORBA, RMI
OROCOS	Basado en puertos	Flujo de datos	Síncrona Asíncrona	CORBA
IEC61499	Basado en puertos	Eventos Flujo de datos	Síncrona	TCP/UDP

OROCOS [52] proporciona de forma nativa un middleware de distribución basado en CORBA, aunque pueden implementarse conectores para otros protocolos. 4DIAC [60], una implementación de código abierto del estándar IEC61499, proporciona

conectores para diversos protocolos como por ejemplo Modbus, OPC-DA, MQTT o Ethernet Powerlink.

2.2.3Soporte para propiedades no funcionales

El soporte que ofrece un marco de componentes para la gestión de las propiedades no funcionales es de vital importancia para cubrir algunos aspectos de los objetivos relativos a confiabilidad, modularidad y en general, cualquier aspecto relativo a la calidad de servicio [76]. En concreto, aquellos aspectos transversales que atañen a la confiabilidad, como por ejemplo la seguridad funcional o la transmisión segura de datos, pueden enfocarse de forma más eficaz si el marco favorece la modularidad a través de la separación de aspectos.

Por ejemplo, SCA [56] propone una aproximación similar a la programación orientada a aspectos (AOP) para especificar e implementar las propiedades no funcionales de los componentes de aplicación y sus conectores. A través de los conceptos de servicio no funcional (*intent*) y política (*policy*), se pueden especificar requisitos no funcionales (también referidos como calidades de servicio), por ejemplo relativos a seguridad funcional, persistencia, autenticación, integridad, etc. Separando la implementación de los aspectos funcionales de los no funcionales, se consigue ganar en eficiencia, ya que las políticas se definen una vez y pueden aplicarse a distintos componentes y conectores a través de anotaciones en el código, lo cual facilita considerablemente el desarrollo de las aplicaciones.

El marco de componentes GCM [77] puede integrarse con SCA [56] y combina el patrón de *objeto activo* [78, 79] con la arquitectura MAPE (*Monitor-Analyze-Plan-Execute*) propuesta por IBM para abordar el paradigma de computación autónoma [80]. Al estar basado en Fractal [81], incorpora el concepto de contenedor (también llamado membrana), que encapsula el componente funcional dotándolo de capacidades de introspección y control. De esta forma, la gestión de las propiedades no funcionales del componente puede externalizarse y desacoplarse del código funcional.

OROCOS [52] soporta la especificación de propiedades de tiempo real (prioridad, periodo) a través del concepto de *TaskContext*, que es el equivalente en OROCOS al concepto de *objeto activo*. Este marco orientado a robótica proporciona determinismo siempre y cuando se ejecute sobre un sistema operativo de tiempo real. 4DIAC [60] también proporciona ejecución determinista de los bloques funcionales del estándar IEC61499 [53], que son accesibles asimismo través de puertos de control.

En general los marcos de componentes proponen diferentes aproximaciones en cuanto al modelo de gestión de propiedades no funcionales [82]. La responsabilidad de esta gestión en general se reparte entre los propios componentes y el entorno de ejecución o middleware subyacente. En el modelo de gestión *endógeno* se asume que en los componentes de aplicación hay que insertar código (adecuadamente encapsulado) específico para la gestión de las propiedades no funcionales. Por el contrario, la aproximación *exógena* considera los componentes de aplicación como unidades completamente independientes conformadas únicamente por código específico de negocio (funcional). En este caso el código no funcional asociado a los componentes se deposita en el contenedor que los aloja.

Tabla 2.3. Soporte para propiedades no funcionales

Marco de componentes	Gestión de propiedades no funcionales	Dominio
Fractal	Exógena / Colaboración	Propósito general
SCA	Exógena / Colaboración	Propósito general
CCM	Exógena / Orquestación	Propósito general
OSGi	Endógena / Colaboración	Propósito general
EJB	Exógena / Orquestación	Propósito general
SOFA	Endógena / Orquestación	Propósito general
OROCOS	Endógena / Colaboración	Robótica
IEC61499	Endógena / Colaboración	Automatización

Por otra parte, las propiedades no funcionales pueden gestionarse por *colaboración* o por *orquestación*. En el primer caso, la plataforma de ejecución no proporciona soporte para la gestión de propiedades no funcionales, que recae exclusivamente en los componentes de aplicación (endógena) o en su contenedor (exógena). En el

segundo caso, la plataforma de ejecución gestiona todo lo relativo a la gestión de las propiedades no funcionales. La Tabla 2.3 resume los enfoques de los distintos modelos analizados, tanto en relación al modelo de gestión de las propiedades no funcionales como en el ámbito de aplicación de cada uno de ellos (de propósito general o específico de dominio). Puede concluirse que prácticamente todos los marcos proporcionan cierto soporte para la gestión de propiedades no funcionales. Ejemplos típicos son la autenticación, persistencia, consumo de recursos o restricciones temporales. Algunos ofrecen soporte formal para la especificación de propiedades no funcionales (p.ej. Fractal [83] o SOFA [84-86]) y muy pocos soportan la composición de propiedades no funcionales de una aplicación completa en base a las propiedades de sus componentes [9].

2.3 Reconfiguración dinámica de aplicaciones distribuidas con requisitos de QoS

En el contexto del presente trabajo de investigación, el concepto de reconfiguración se refiere a los siguientes casos:

- modificación de la topología del sistema. Entendido como una modificación de la distribución de los componentes software en la infraestructura de nodos hardware distribuidos.

- modificación de los atributos (parametrización) de los componentes: Estos atributos o propiedades pueden ser funcionales (p.ej. modo de funcionamiento de un componente) o no funcionales (p.ej. periodo de un componente de ejecución periódica).

En la Sección 2.3.1 se exponen algunos conceptos previos relativos a la reconfiguración, y en la Sección 2.3.2 se realiza un recorrido por algunos trabajos de

investigación centrados en la reconfiguración dinámica de aplicaciones con requisitos de QoS.

2.3.1 Tipos de reconfiguración

2.3.1.1 Reconfiguración estática vs. Dinámica.

Se dice que un marco de componentes soporta reconfiguración estática cuando para poder reconfigurar una aplicación primero hay que pararla, después modificar su configuración, y finalmente proceder a su reinicio con la nueva configuración. Este es el caso de SCA [56], que basa sus capacidades de configuración en archivos estáticos XML que describen la configuración de la aplicación mediante un lenguaje ADL específico. Estos ficheros definen los componentes de aplicación, con sus interfaces y conectores, en tiempo de diseño. Pero no se proporciona un entorno de ejecución que permita el control de los componentes como entidades independientes en tiempo de ejecución. Este es también el caso de otros modelos de componentes como por ejemplo Wright, Darwin o ACME analizados en [50].

En cambio, otros marcos de componentes cuentan con un entorno de ejecución que permite la reconfiguración dinámica (en tiempo de ejecución) de las aplicaciones. Este es el caso de SOFA [55] o Fractal [47, 48]. Cabe destacar, por tener ciertas similitudes con la propuesta realizada en este trabajo de investigación, el marco de componentes FraSCAti [87, 88]. Este marco proporciona una implementación del estándar SCA [56], pero frente a otras implementaciones del mismo estándar como Tuscany [62], añade soporte (no exigido por el estándar SCA [56]) para permitir la reconfiguración dinámica de las aplicaciones, aprovechando el hecho de que está implementado sobre el propio Fractal [48]. Los componentes de una aplicación FraSCAti admiten la inyección de *scripts* de reconfiguración que les permiten realizar operaciones de reconfiguración como por ejemplo la modificación de conectores o atributos del componente.

2.3.1.2 Reconfiguración funcional vs. No funcional

En el contexto de este trabajo de investigación, se entiende que una reconfiguración es **funcional** cuando la funcionalidad del sistema se ve alterada como respuesta a demandas del usuario o de las propias aplicaciones. Por ejemplo, la activación de una nueva aplicación por parte del usuario supone un cambio en la funcionalidad global del sistema, y a su vez implica un cambio en la topología del sistema, derivada de la instanciación de los componentes distribuidos que conforman la aplicación.

Sin embargo, una reconfiguración **no funcional** no implica cambios en la funcionalidad del sistema. Un ejemplo de reconfiguración no funcional puede ser la actualización de un componente de aplicación por una versión más reciente. La funcionalidad ofrecida sigue siendo la misma pero su configuración ha cambiado debido al cambio de versión del componente. Un fallo en un nodo hardware es otro ejemplo de reconfiguración no funcional en el cual se modifica la topología del sistema, ya que los componentes software que se estaban ejecutando en el nodo caído deben pasar a ejecutarse en otros nodos que tengan recursos disponibles.

Como ejemplo de reconfiguración no funcional disparada por una aplicación puede considerarse una aplicación de video vigilancia formada por dos componentes. El primer componente gestiona un sensor de presencia y el segundo componente recibe las imágenes de una videocámara y las graba en disco. La calidad de grabación de las imágenes depende de si el sensor detecta presencia o no. Cuando el sensor detecta presencia, el componente de video demanda más recursos de computación para poder grabar las imágenes a una mayor tasa de fotogramas por segundo. Suponiendo que no haya recursos suficientes para atender esta demanda, el sistema podría decidir el apagado de otras aplicaciones menos prioritarias, lo cual supone una reconfiguración de la topología del sistema.

2.3.1.3 Reconfiguración sin estado vs. con estado

En el contexto de este trabajo de investigación, consideramos que se ejecuta una reconfiguración **con estado** (*stateful*) cuando el sistema es capaz de instanciar un componente inicializándolo a un estado concreto. Este estado estará definido por un conjunto de variables específicas de dicho componente. Por el contrario, una reconfiguración **sin estado** (*stateless*) siempre instancia los componentes con un estado inicial definido en la configuración estática. Esto quiere decir que ante eventos de reconfiguración no funcional como por ejemplo la caída de un nodo, un sistema que soporte la reconfiguración con estado podría instanciar los componentes afectados en nodos alternativos, y recuperar el estado de la aplicación previo al fallo.

En este sentido, el objetivo relativo a la confiabilidad, sobre todo desde la perspectiva de la disponibilidad, está muy relacionado con el soporte a la reconfiguración con estado. Un ejemplo claro puede encontrarse en el caso de estudio relativo al ámbito de los almacenes automáticos. Frente a aproximaciones basadas en gestionar eventuales fallos del maestro de tráfico de un grupo de ascensores mediante modos de funcionamiento degradados [89], pueden considerarse otras alternativas más ambiciosas que contemplen la redundancia de dicho componente como mecanismo de tolerancia a fallos con mantenimiento íntegro del estado y las prestaciones. Esto requeriría que ante una caída del maestro principal, el de reserva tome el control de forma inmediata, partiendo del estado exacto que tenía el principal.

En la literatura existen distintas aproximaciones al problema del mantenimiento del estado de las aplicaciones [35, 37, 90], que por otra parte puede ser **local** o **distribuido**. En el caso del estado distribuido la restauración del estado de la aplicación distribuida se complica porque puede ser necesario instanciar un componente con el estado de otro componente remoto. En la siguiente sección se incluye el mantenimiento del estado en el análisis de los diferentes trabajos de investigación, por ser de vital importancia para acometer con éxito los objetivos

relativos a la confiabilidad, escalabilidad y gestión de la calidad de servicio de las aplicaciones.

2.3.2 Calidad de servicio y gestión de recursos

En la Sección 2.3.1 se han comentado diversos motivos que pueden provocar la reconfiguración de un sistema. Independientemente del evento que lo origine, el proceso de reconfiguración debe ser gestionado de acuerdo a unas determinadas políticas que aseguren, en la medida de lo posible, la calidad de servicio de las aplicaciones implicadas. Por tanto, el proceso de reconfiguración normalmente va ligado a la existencia de mecanismos de gestión de recursos y calidad de servicio que piloten dicho proceso.

A este respecto, los marcos de componentes analizados en la Sección 2.1 no ofrecen soporte nativo para dicha gestión. Debido a ello en esta sección se analizan otros trabajos de investigación que se centran en la gestión de la calidad de servicio para sistemas distribuidos dinámicamente reconfigurables. Estos trabajos, resumidos en la Tabla 2.4, generalmente consideran modelos de componentes abstractos orientados a la simulación y testeo de estrategias de reconfiguración. Salvo los basados en CCM [91-94], no contemplan la adopción de un marco de componentes estándar. En este sentido, se identifica la oportunidad de combinar las ventajas que proporciona un marco de componentes estándar con una gestión adecuada de los recursos y de la calidad de servicio.

Tabla 2.4. Soporte para reconfiguración y gestión de recursos

	Reconfiguración con estado	Estado distribuido	Gestión de recursos	Modelo de componente estándar
Li [35, 37]	Sí	No	-	No
Valls [95-97]	No	No	Sí	No
Romero [98]	Sí	No	Sí	No
Almeida [99]	No	No	Sí	No
SLAstic [100, 101]	No	No	Sí	No
Schmidt [91-94]	No	No	No	Sí
Orlic [102]	No	No	Sí	No
GCM [77]	Sí	No	No	No
Hammer [90]	Sí	Sí	No	No

En [35, 37], Li proporciona un buen estado del arte centrado en el aseguramiento de la calidad de servicio durante la reconfiguración de sistemas basados en componentes. En este caso, el aseguramiento de la QoS se centra en reducir al mínimo el tiempo necesario para ejecutar una reconfiguración del sistema, minimizando así el tiempo de inactividad del sistema durante la reconfiguración. Por otra parte se trata de evitar, en la medida de lo posible, que el proceso de reconfiguración afecte a las transacciones en curso. Para soportar la reconfiguración con estado, se introduce el concepto de *versionado*. Este concepto supone que los componentes antiguos (a sustituir) y los nuevos (sustituyen a los antiguos) pueden coexistir durante el periodo de reconfiguración. Este enfoque no es válido para nuestras necesidades ya que cubre exclusivamente reconfiguraciones planificadas (actualizaciones de componentes), y no trata la reconfiguración motivada por fallos de hardware o software, en la que un componente puede desaparecer de forma imprevista. Por otra parte, Li propone el uso de memoria compartida cuando el tamaño del estado a transferir es alto. Esta solución mejora el rendimiento de forma considerable, pero no es viable cuando el estado es distribuido, ya que no considera la transferencia de estado en aquellos casos en los que la réplica del componente fallido debe instanciarse en un nodo remoto.

Otros trabajos [95-97] están enfocados a la optimización del grafo de ejecución de aplicaciones basadas en servicios. Gestionan los recursos disponibles (CPU, memoria) y consideran parámetros de calidad de servicio específicos de aplicación (p.ej. los fotogramas por segundo o la resolución en aplicaciones de video). Proponen la utilización de componentes específicos de aplicación para asistir al middleware de plataforma en la búsqueda del grafo de ejecución óptimo. Se centran en la reconfiguración del grafo de ejecución en el contexto de una aplicación basada en servicios sin estado, y tratan de elegir la implementación de servicio más adecuada basándose en una serie de parámetros de calidad de servicio. Sin embargo, la reconfiguración funcional de las conexiones entre servicios no se detalla. Se presenta el concepto de *gestor de recursos*, que es el responsable de gestionar eventos como

por ejemplo sobrecargas o fallos de nodos hardware, aunque no se detallan los mecanismos utilizados para la detección de tales eventos.

En [96] se presenta un algoritmo basado en una caracterización de la calidad de servicio en base a consumo de recursos de cómputo y memoria, aunque limitado a servicios sin estado. La selección entre las implementaciones de servicio disponibles se optimiza de acuerdo a diferentes criterios, como por ejemplo el plazo de ejecución del grafo completo de principio a fin (*end to end deadline*). Las diferentes demandas de calidad de servicio (y por tanto de recursos) de un mismo servicio funcional están representadas mediante las diferentes implementaciones del mismo. Estas implementaciones de servicio, una vez desplegadas, tienen un conjunto de parámetros de calidad de servicio que es fijo y no cambia.

En [97] se presenta el modelo de componente *iLaser*, que incluye un puerto de control para permitir su reconfiguración. En [98], se propone un marco de componentes basado en Java que soporta el reemplazo dinámico de componentes locales con estado, pero no considera el problema del estado distribuido.

En [99] se presenta el concepto de factor de utilización local (LUB, *Local Utilization Bound*), que sirve para poder realizar análisis de planificabilidad en tiempo de ejecución, debido al bajo coste computacional que supone su cálculo. Se consigue un elevado rendimiento porque la reconfiguración se limita a un conjunto de configuraciones que han sido previamente calculadas en tiempo de diseño. Este trabajo está orientado a tareas locales, es decir, no considera el concepto de componente.

El marco de componentes *SLAstic* se presenta en [100, 101]. Este marco adopta el modelo de componente *Palladio* [103] y considera tres aspectos de la reconfiguración dinámica: (1) asignación de recursos de computación (nodos hardware) a los componentes software, (2) la migración de componentes, y (3) el balance de carga. Su objetivo es optimizar los recursos de hardware existentes a la vez que se asegura el

cumplimiento de un contrato de nivel de servicio (SLA, *Service Level Agreement*), ya sea evitando la infrautilización en periodos de baja demanda de recursos o la sobrecarga en situaciones de estrés. Se especifica la funcionalidad algoritmo de gestión de recursos, pero no se detalla su diseño. No se contempla ningún mecanismo específico para soportar la transferencia de estado distribuido.

En [91-93], se presenta un middleware basado en el modelo de componente de CORBA CCM (Lightweight CORBA Component Model), que adjudica recursos físicos (CPU, memoria, ancho de banda) a los componentes de aplicación. Propone un gestor de recursos multicapa (*MLRM*) compuesto por varios módulos que ofrecen funcionalidades como por ejemplo monitorización de recursos, control de admisión o algoritmos de composición. Considera aspectos de QoS como la disponibilidad o la seguridad, pero no detalla los mecanismos de bajo nivel que los implementan. A través de una API de gestión de recursos detallada en [91], las aplicaciones pueden preguntar por la disponibilidad de recursos y proceder a la reserva de los mismos. Sin embargo, no proporciona un algoritmo específico de gestión de recursos, se limita a proporcionar los mecanismos para que sea el usuario el que integre el suyo propio. Por otra parte, CORBA no está siendo utilizado de forma extendida en la industria, y su ámbito de aplicación es relativamente reducido [104]. En [94] se presenta una mejora del estándar CORBA D&C [105], centrada en el despliegue y activación determinista de aplicaciones basadas en CORBA, aunque no se considera ningún mecanismo de gestión de recursos dinámico. En [106] se propone un servicio CORBA que implementa el paradigma FTT (*Flexible Time Triggered*) [107], orientado a aplicaciones distribuidas deterministas que pueden ser dinámicamente reconfiguradas. Para ello, propone un *orquestador* capaz de añadir y eliminar tareas distribuidas y de reconfigurar de forma acorde la planificación del sistema.

Otro marco de componentes con gestión de recursos de infraestructura se presenta en [102]. Propone contratos de demanda de recursos para gestionar los recursos demandados por los servicios. Debido a su orientación a servicios, no contempla componentes con estado.

En [77] se presenta un marco basado en el modelo de componente GCM (*Grid Component Model*), que es una adaptación de Fractal [48] para supercomputación distribuida. Está basado en objetos activos altamente desacoplados denominados "componentes autónomos", cada uno de ellos desplegado en una máquina virtual de Java independiente. No soporta aspectos relativos a calidad de servicio y no detalla mecanismos específicos para soportar la transferencia de estado distribuido.

En [108] se detalla un marco de componentes orientado a tiempo real que se basa en una ejecución secuencial orquestada por un gestor central. Proporciona determinismo y reduce el acoplamiento de componentes, pero no soporta reconfiguración dinámica.

En [90] la transferencia de estado es dependiente de la aplicación, es decir, los componentes de aplicación deben proporcionar métodos de acceso de forma que el middleware de gestión pueda acceder a su estado interno para reinyectarlo en su réplica. Una aproximación similar se presenta en [109]. Es decir, no se proporciona una forma genérica para que los componentes puedan reportar su estado a la plataforma.

2.4 Conclusiones

Los marcos de componentes analizados en la Sección 2.2 persiguen objetivos similares y atienden algunas de las demandas planteadas en el capítulo de introducción. Sin embargo, ninguno de ellos cubre todas las necesidades identificadas. Concretamente, las principales carencias se identifican en relación a los objetivos de confiabilidad, escalabilidad, gestión de recursos y aseguramiento de la QoS. Por esta razón, en la Sección 2.3 se han analizado una serie de trabajos que se centran en estos aspectos. Desafortunadamente, o bien no consideran el concepto de componente en absoluto, o si lo hacen se refieren a modelos abstractos *ad-hoc* orientados a la demostración de sus estrategias de reconfiguración y aseguramiento de la QoS.

En la Tabla 2.5 se resumen las necesidades y objetivos planteados en la introducción como motivación del presente trabajo de investigación, y se indica la sección en la que se consideran dentro del capítulo del estado del arte. Por otra parte, en la Tabla 2.6 se indica la cobertura que proporcionan los marcos de componentes analizados para dichos objetivos y necesidades. Esta cobertura puede existir o no, o proporcionarse de forma parcial.

Tabla 2.5. Resumen de objetivos y necesidades

Objetivo	Necesidades asociadas	Sección
Soporte en tiempo de ejecución	Servicios de monitorización/control	2.2.1
Distribución de datos e interoperabilidad	Distribución geográfica Heterogeneidad e interoperabilidad Código heredado	2.2.2
Confiabilidad	Disponibilidad Seguridad funcional	2.2.3 2.3.2
Escalabilidad	Escalabilidad Calidad de Servicio Reconfiguración dinámica	2.3.2
Gestion de recursos y aseguramiento QoS	Escalabilidad Calidad de Servicio Reconfiguración dinámica	2.3.2
Soporte a sistemas heredados	Heterogeneidad e interoperabilidad Código heredado	2.2.1
Funcionalidad ampliable	Escalabilidad	2.2.3
Soporte generación automática de codigo	Heterogeneidad e interoperabilidad Código heredado	2.2.1

El marco más completo en cuanto a distribución de datos e interoperabilidad con sistemas ya existentes es SCA. El resto proveen de mayor o menor número de protocolos de comunicación, salvo Fractal, que está en un nivel mayor de abstracción y no especifica soporte para protocolos concretos.

Cabe señalar que el objetivo relativo a la confiabilidad, en los términos expuestos en la introducción, no está soportado de forma nativa por ninguno de los marcos. Sin embargo, tal como se ha comentado en la Sección 2.2.3, todos ellos proporcionan de una u otra forma los mecanismos para la integración de propiedades no funcionales, y por tanto habilitan la eventual implementación de los mecanismos de comunicación

segura demandados en la introducción. En relación al soporte para sistemas heredados, tres de los marcos son multilenguaje y por tanto facilitan la migración de código heredado. La generación automática de código es más fácil en algunos de ellos gracias a que contemplan lenguajes ADL u otros tipos de modelado de la arquitectura.

Tabla 2.6. Cobertura de objetivos

	Objetivos				
	Soporte en tiempo de ejecución	Distribución e Interoperabilidad	Soporte a sistemas heredados	Funcionalidad ampliable	Generación automatica de código
Fractal	√	-	√	√	√
SCA	-	√	√	√	√
CCM	√	Parcial	√	√	-
OSGi	√	Parcial	-	√	-
EJB	√	Parcial	-	√	-
SOFA	√	Parcial	-	√	√
OROCOS	√	Parcial	-	√	-
IEC61499	√	Parcial	-	√	√

Por otra parte, la necesidad de poder escalar el sistema para hacer frente a la integración dinámica de nuevas aplicaciones, requiere de una gestión de los recursos que permita controlar el nivel de QoS de las mismas. Estos objetivos no están cubiertos por los marcos de componentes analizados. Asimismo, los trabajos de investigación analizados en la Sección 2.3 tampoco cubren en su totalidad estas necesidades. Por ejemplo, el objetivo de confiabilidad requiere de soporte para poder restaurar, en caso de fallo, el estado de una aplicación distribuida. Sólo uno de los trabajos analizados [90] proporciona soporte para ello, pero sin embargo no contempla mecanismos de gestión de recursos, necesarios ante eventos de reconfiguración dinámica.

En conclusión, se identifica la oportunidad para el diseño y desarrollo de una plataforma que dé respuesta a todos los objetivos planteados en la introducción. Los requisitos de detalle de dicha plataforma, junto con el modelo formal del sistema, se especifican en el capítulo siguiente.

7 CONCLUSIONES Y LÍNEAS FUTURAS

7.1 Conclusiones

El concepto de *Internet Industrial de las Cosas* responde a una nueva concepción de la interacción entre los humanos y las máquinas. En este paradigma, cualquier *cosa* que tenga capacidad de procesamiento y que esté conectada a la red es susceptible de participar en un entorno colaborativo global. Esta incipiente integración de las *cosas* en una red cibernética global puede tener gran potencial como herramienta transformadora de la industria y como catalizador de nuevas aplicaciones y servicios que aún hoy no imaginamos. En definitiva, se contempla que este paradigma tenga un elevado impacto social en el medio plazo.

Para sustentar la *Internet Industrial de las Cosas* se necesitan nuevas tecnologías en muy diversas áreas, que van desde las telecomunicaciones hasta la energía o el software, campo en el que se enmarca el presente trabajo de investigación. Desde este punto de vista, las aplicaciones *IIoT* exhiben una serie de características que han marcado los objetivos de investigación. Se trata de aplicaciones basadas en componentes geográficamente distribuidos y heterogéneos en cuanto a las plataformas hardware y software sobre las que corren o los lenguajes en los que se implementan. Normalmente la interoperabilidad entre esos componentes se basa en distintos protocolos de comunicación, y demandan ciertos requisitos no funcionales de escalabilidad, disponibilidad, seguridad y de gestión de recursos y QoS.

Aunque existen algunos marcos de componentes que cubren parcialmente estas necesidades, se identifican algunas carencias en relación a los objetivos de confiabilidad, escalabilidad, gestión de recursos y aseguramiento de la QoS.

En este contexto, este trabajo propone una plataforma de gestión de aplicaciones basadas en componentes software distribuidos. Esta plataforma, denominada **DAMP**, proporciona soporte en tiempo de ejecución para las aplicaciones, y ofrece una serie de servicios que responden a las necesidades de las aplicaciones objetivo.

En concreto, la principal aportación de este trabajo de tesis es la concepción de una plataforma de gestión de aplicaciones que proporciona las siguientes funcionalidades y características:

- Modelo de componente:
  - Multilenguaje y multiplataforma
  - Modelo de ejecución pasivo y activo
  - Diseño orientado a la generación automática de código e integración de código heredado
- Paradigmas de comunicación cliente-servidor y publicador-suscriptor
- Comunicaciones seguras
- Registro e instalación de nodos de infraestructura y aplicaciones
- Monitorización de recursos y aplicaciones
- Reconfiguración dinámica automatizada para los tipos de eventos:
  - Evento funcional: disparado por las aplicaciones cuando modifican su demanda de QoS
  - Evento no funcional: fallo en un nodo hardware
- Tolerancia a fallos con mantenimiento del estado

Las aportaciones que conforman la plataforma DAMP han dado lugar a un conjunto de resultados que han sido publicados en dos revistas con índice de impacto y presentados en siete conferencias internacionales de reconocido prestigio en el campo de la investigación. En concreto se han realizado las siguientes publicaciones:

1. A. Agirre, J. Parra, A. Armentia, E. Estévez, and M. Marcos. "QoS Aware Middleware Support for Dynamically Reconfigurable Component Based IoT Applications". *International Journal of Distributed Sensor Networks,* vol. 2016, pp. 1-17, Art. no. 2702789. JCR_2015: 0.906 (Telecommunications: T2/Q3).

2. A. Agirre, J. Parra, A. Armentia, A. Ghoneim, E. Estévez, and M. Marcos. "QoS management for dependable sensory environments". *Multimedia Tools and*

Applications, pp. 1-23, 2015. DOI: 10.1007/s11042-015-2781-4. JCR_2015: 1.331 (Computer Science, Software Engineering T1/Q2).

3. A. Agirre, J. Parra, E. Estevez, and M. Marcos. "QoS aware platform for dependable sensory environments". In: *IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, Chengdu, China, 2014, pp. 1-5.

4. A. Agirre, E. Estevez, and M. Marcos. "Resource management support for SCA based distributed applications". In: *IEEE 19th Conference on Emerging Technologies & Factory Automation (ETFA),* Barcelona, Spain, 2014, pp. 1-4.

5. A. Agirre, J. Perez, R. Priego, M. Marcos, and E. Estevez. "SCA extensions to support safety critical distributed embedded systems". In: *IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA),* Cagliari, Italy, 2013, pp. 1-4.

6. A. Agirre, M. Marcos, and E. Estevez. "Distributed Applications Management Platform Based on Service Component Architecture" in *IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA),* Krakow, Poland, 2012, pp. 1-4.

7. A. Agirre, E. Estévez, and M. Marcos. "Fault tolerant component management platform over Data Distribution Service" in *1st IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control (CESCIT)*, Wurzburg, 2012, pp. 218-223.

8. A. Agirre, M. Marcos, and E. Estevez. "Distributed component management platform for QoS enabled applications", in *IEEE 16th Conference on Emerging Technologies & Factory Automation (ETFA),* Toulouse, France, 2011, pp.1-4.

9. A. Agirre, E. Estévez, and M. Marcos. "QoS enabled application management platform over DDS" presented at the Proceedings of the Middleware Workshop on Posters and Demos Track, Lisbon, Portugal, 2011.

Con respecto a la integración de la plataforma de gestión con un entorno de desarrollo integrado que permite el modelado de las aplicaciones y la monitorización y control de las mismas, se han realizado las siguientes publicaciones:

1. Armentia, A., Agirre, A., Estévez, E., Pérez, J., and Marcos, M. "Model Driven Design Support for Mixed-Criticality Distributed Systems". In: *19th World Congress of the International Federation of Automatic Control (IFAC)*. Cape Town, South Africa, 2014, Elsevier, pp. 4441–4446.

2. Armentia, A., Sarachaga, I., García de Albéniz, O., Estévez, E., Agirre, A., and Marcos, M. "Achieving Reconfigurable Service Oriented Applications Using Model Driven Engineering". In: *16th IEEE International Conference on Emerging Technologies & Factory Automation (ETFA)*. Toulouse, France, 2011, IEEE, pp. 1–4.

Finalmente, se ha colaborado con otros autores en la adaptación de la plataforma DAMP al ámbito de la automatización industrial, dando como resultado la siguiente publicación:

1. R. Priego, A. Agirre, E. Estévez, D. Orive, and M. Marcos. "Middleware-based Support for Reconfigurable Mechatronic Systems" presented at the *2nd Conference on Embedded Systems, Computational Intelligence and Telematics in Control (CESCIT 2015)*, Maribor, Slovenia, 2015.

7.2 Líneas futuras

Durante la realización de este trabajo de investigación se han identificado una serie de puntos de mejora y trabajos futuros que pueden dar continuidad a la línea de investigación iniciada.

Aunque la plataforma DAMP se ha enfocado inicialmente como una plataforma de gestión que da cobertura a aplicaciones de propósito general, y de facto se ha

contrastado en entornos aplicativos de tipo empresarial, se considera interesante su adaptación a ámbitos de aplicación más específicos, como pueden ser la automatización industrial o las redes de sensores. En este sentido, el explorar la posibilidad de "aligerar" la plataforma para adaptarla a dispositivos embebidos con recursos limitados puede resultar de gran interés.

El presente trabajo de investigación no contemplaba el algoritmo de composición como un objetivo de investigación en sí mismo. Lo que se ha buscado es proporcionar un diseño modular que permita llegado el caso sustituir el algoritmo actual por otro que persiga distintos objetivos. En este sentido se identifica un gran margen de mejora en la optimización del algoritmo actual o en el desarrollo de nuevos algoritmos que implementen distintas políticas de gestión de recursos. Asimismo, cabe contemplar aspectos como la reconfiguración del sistema en tiempos acotados.

En la versión actual de la plataforma se han explorado protocolos sobre Ethernet IP. Sin embargo, la integración de dispositivos de bajo consumo y recursos limitados, como por ejemplo las redes de sensores, demanda otro tipo de tecnologías de comunicación que ahorren energía aún a costa de disminuir el ancho de banda. En este sentido, la plataforma actual podría extenderse para integrar nuevos protocolos de red de área extensa (WAN) que están orientados a la Internet Industrial de las Cosas, como pueden ser LoRa, Sigfox, LTE o NarrowBand IoT. Asimismo, protocolos cableados clásicos como por ejemplo CAN también son susceptibles de ser integrados en DAMP, ya que cubren sistemas distribuidos embebidos de uso común en la industria (automatización, transporte, etc.).

A un nivel superior, se contempla la posibilidad de integrar en la plataforma modelos de información basados en estándares industriales como por ejemplo OPC-UA. La información disponible en la base de datos del sistema podría ofrecerse a través de este tipo de protocolos para su posterior explotación en sistemas tipo SCADA o en plataformas de computación en la nube (minería de datos, *Big Data Analysis*), que conforman otra de las piedras angulares del paradigma IIoT.

REFERENCIAS

[1] K. A. Delin and S. P. Jackson, "Sensor Web for in situ exploration of gaseous biosignatures," in *Aerospace Conference Proceedings, 2000 IEEE*, 2000, vol. 7, pp. 465-472 vol.7.

[2] (2008). *Cyber-Physical Systems (CPS)*.

[3] K. Ashton, "That 'Internet of Things' Thing," *RFID Journal,* 2009.

[4] Z. Longhao, R. Trestian, and G. M. Muntean, "E²DOAS: User experience meets energy saving for multi-device adaptive video delivery," in *Computer Communications Workshops (INFOCOM WKSHPS), 2015 IEEE Conference on*, 2015, pp. 444-449.

[5] S. Pollin, M. Timmers, and L. Van der Perre, "Anticipative Energy and QoS Management: Systematically Improving the User Experience," in *Software Defined Radios*(Signals and Communication Technology: Springer Netherlands, 2011, pp. 87-108.

[6] Z. Yuhao, M. Halpern, and V. J. Reddi, "Event-based scheduling for energy-efficient QoS (eQoS) in mobile Web applications," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, 2015, pp. 137-149.

[7] "Industrial Internet Consortium," ed, 2014.

[8] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective," *IEEE Internet of Things Journal,* vol. 1, no. 4, pp. 349-359, 2014.

[9] I. Crnkovic, S. Sentilles, A. Vulgarakis, and M. R. V. Chaudron, "A Classification Framework for Software Component Models," *Software Engineering, IEEE Transactions on,* vol. 37, no. 5, pp. 593-615, 2011.

[10] T. Pop, P. Hnětynka, P. Hošek, M. Malohlava, and T. Bureš, "Comparison of component frameworks for real-time embedded systems," (in English), *Knowledge and Information Systems,* pp. 1-44, 2013/04/02 2013.

[11] M. P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions," in *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, 2003, pp. 3-12.

[12] G. Kiczales *et al.*, "Aspect-oriented programming," in *ECOOP'97 — Object-Oriented Programming: 11th European Conference Jyväskylä, Finland, June 9–13, 1997 Proceedings*, M. Akşit and S. Matsuoka, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 220-242.

[13] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "Integration of Cloud computing and Internet of Things: A survey," *Future Generation Computer Systems,* vol. 56, pp. 684-700, 3// 2016.

[14] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, "The rise of “big data” on cloud computing: Review and open research issues," *Information Systems,* vol. 47, pp. 98-115, 1// 2015.

[15] C. Liu, C. Yang, X. Zhang, and J. Chen, "External integrity verification for outsourced big data in cloud and IoT: A big picture," *Future Generation Computer Systems,* vol. 49, pp. 58-67, 8// 2015.

[16] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A Survey," *IEEE Internet of Things Journal,* vol. 3, no. 1, pp. 70-95, 2016.

[17] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "A Survey of Middleware for Internet of Things," in *Recent Trends in Wireless and Mobile Networks: Third International Conferences, WiMo 2011 and CoNeCo 2011, Ankara, Turkey, June 26-28, 2011. Proceedings*, A. Özcan, J. Zizka, and D. Nagamalai, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 288-296.

[18] B. P. Rimal, E. Choi, and I. Lumb, "A Taxonomy and Survey of Cloud Computing Systems," presented at the Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC, 2009.

[19] S. S. Manvi and G. Krishna Shyam, "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey," *Journal of Network and Computer Applications,* vol. 41, pp. 424-440, 5// 2014.

[20] D. Ma, "The Business Model of "Software-As-A-Service"," in *IEEE International Conference on Services Computing (SCC 2007)*, 2007, pp. 701-702.

[21] P. Persson and O. Angelsmark, "Calvin – Merging Cloud and IoT," *Procedia Computer Science,* vol. 52, pp. 210-217, 2015/01/01 2015.

[22] Particle. (2016). Available: www.particle.io

[23] Cayenne. (2016). Available: http://www.cayenne-mydevices.com/

[24] S. Gulwani, J. Hernandez-Orallo, E. Kitzelmann, S. H. Muggleton, U. Schmid, and B. Zorn, "Inductive programming meets the real world," *Commun. ACM,* vol. 58, no. 11, pp. 90-99, 2015.

[25] D. Floyer, "The Vital Role of Edge Computing in the Internet of Things," WIKIBON, Ed., ed: WIKIBON, 2015.

[26] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," presented at the Proceedings of the first edition of the MCC workshop on Mobile cloud computing, Helsinki, Finland, 2012.

[27] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog Computing: A Platform for Internet of Things and Analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*, N. Bessis and C. Dobre, Eds. Cham: Springer International Publishing, 2014, pp. 169-186.

[28] L. M. Vaquero and L. Rodero-Merino, "Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing," *SIGCOMM Comput. Commun. Rev.,* vol. 44, no. 5, pp. 27-32, 2014.

[29] *Message Queuing Telemetry Transport (MQTT)*, 2015.

[30] P. Hintjens, I. O'Reilly Media, Ed. *ZeroMQ: Messaging for Many Applications*. O'Reilly, 2013.

[31] C. Bormann, A. P. Castellani, and Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," *IEEE Internet Computing,* vol. 16, no. 2, pp. 62-67, 2012.

[32] *The Constrained Application Protocol (CoAP)*, 2014.

[33] OMG, "Data Distribution Service for Real-time Systems v1.2," ed, 2007.

[34] RTI. (2016). *Connext DDS Micro*. Available: https://www.rti.com/products/micro.html

[35] W. Li, "Evaluating the impacts of dynamic reconfiguration on the QoS of running systems," *Journal of Systems and Software,* vol. 84, no. 12, pp. 2123-2138, 2011.

[36] M. García Valls and P. Basanta Val, "Comparative analysis of two different middleware approaches for reconfiguration of distributed real-time systems," *Journal of Systems Architecture,* vol. 60, no. 2, pp. 221-233, 2// 2014.

[37] W. Li, "QoS assurance for dynamic reconfiguration of component-based software systems," (in English), *IEEE Transactions on Software Engineering,* vol. 38, no. 3, pp. 658-676, 2012, Art. no. 5740932.

[38] H. Maaskant, "A Robust Component Model for Consumer Electronic Products," in *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*, P. van der Stok, Ed. Dordrecht: Springer Netherlands, 2005, pp. 167-192.

[39] *Autosar website*. Available: http://www.autosar.org/

[40] O. Nierstrasz *et al.*, "A Component Model for Field Devices," in *Component Deployment: IFIP/ACM Working Conference, CD 2002 Berlin, Germany, June 20–21, 2002 Proceedings*, J. Bishop, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 200-209.

[41] S. Hissam, J. Ivers, D. Plakosh, and K. C. Wallnau, "Pin Component Technology (V1.0) and Its C Interface," CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST2005.

[42] R. v. Ommering, F. v. d. Linden, J. Kramer, and J. Magee, "The Koala component model for consumer electronics software," *Computer,* vol. 33, no. 3, pp. 78-85, 2000.

[43] P. Mohagheghi, "The Impact of Software Reuse and Incremental Development on the Quality of Large Systems," PhD, Department of Computer and Information Science Norwegian University of Science and Technology 2004.

[44] L. Kung-Kiu and W. Zheng, "Software Component Models," *Software Engineering, IEEE Transactions on,* vol. 33, no. 10, pp. 709-724, 2007.

[45] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[46] G. T. Heineman and W. T. Councill, *Component-Based Software Engineering: Putting the Pieces Together* Addison-Wesley Longman Publishing Co., 2001.

[47] G. Blair, T. Coupaye, and J.-B. Stefani, "Component-based architecture: the Fractal initiative," *Annals of Telecommunications,* vol. 64, no. 1, pp. 1-4, 2009.

[48] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J. B. Stefani, "The FRACTAL component model and its support in Java," *Software - Practice and Experience,* vol. 36, no. 11-12, pp. 1257-1284, 2006.

[49] E. Bruneton, T. Coupaye, and J.-B. Stefani, "Recursive and Dynamic Software Composition with Sharing," presented at the Seventh International Workshop on Component-Oriented Programming (WCOP 2002), 2002.

[50] P. Hnetynka, L. Murphy, and J. Murphy, "Comparing the service component architecture and fractal component model," *Computer Journal,* vol. 54, no. 7, pp. 1026-1037, 2011.

[51] K. Ji Eun, O. Rogalla, S. Kramer, and A. Hamann, "Extracting, specifying and predicting software system properties in component based real-time embedded software development," in *Software Engineering - Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, 2009, pp. 28-38.

[52] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the OROCOS project," 2003, vol. 2, pp. 2766-2771.
[53] *IEC 61499: Function Blocks for Industrial Process Measurement and Control Systems, Parts 1 - 4*, 2004/2005.
[54] *The Fractal Component Model Specification*, 2004.
[55] M. Malohlava, P. Hnetynka, and T. Bures, "SOFA 2 Component Framework and Its Ecosystem," *Electronic Notes in Theoretical Computer Science,* vol. 295, no. 0, pp. 101-106, 5/9/ 2013.
[56] *Service Component Architecture*, 2007.
[57] E. Cavalcante, F. Oquendo, and T. Batista, "Architecture-Based Code Generation: From π-ADL Architecture Descriptions to Implementations in the Go Language," in *Software Architecture: 8th European Conference, ECSA 2014, Vienna, Austria, August 25-29, 2014. Proceedings*, P. Avgeriou and U. Zdun, Eds. Cham: Springer International Publishing, 2014, pp. 130-145.
[58] S. Becker, H. Koziolek, and R. Reussner, "The Palladio component model for model-driven performance prediction," *Journal of Systems and Software,* vol. 82, no. 1, pp. 3-22, 1// 2009.
[59] C. Atkinson *et al.*, "Modeling Components and Component-Based Systems in KobrA," in *The Common Component Modeling Example: Comparing Software Component Models*, A. Rausch, R. Reussner, R. Mirandola, and F. Plášil, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 54-84.
[60] T. Strasser *et al.*, "Framework for Distributed Industrial Automation and Control (4DIAC)," in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, 2008, pp. 283-288.
[61] *OMG CORBA Component Model v4.0*, 2011.
[62] S. Laws, M. Combellack, R. Feng, H. Mahbod, and S. Nash, *Tuscany SCA in Action*. 2011.
[63] ObjectWeb. (2016). *OpenCCM - The Open CORBA Component Model Platform*. Available: http://openccm.ow2.org/
[64] O. Consortium. (2009). *Julia*. Available: http://fractal.ow2.org/julia/index.html
[65] O. Consortium. (2005). *Think*. Available: http://think.ow2.org/
[66] O. Consortium. (2009). *FracNet*. Available: http://www-adele.imag.fr/fractnet/
[67] O. Consortium. (2009). *Julio*. Available: http://www.lifl.fr/~marvie/software/julio.html
[68] *JSR 345: Enterprise JavaBeansTM,Version 3.2 EJB Core Contracts and Requirements*, 2013.
[69] A. Plsek, F. Loiret, P. Merle, and L. Seinturier, "A component framework for java-based real-time embedded systems," presented at the Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware, Leuven, Belgium, 2008.
[70] *OSGi Service Platform Core Specification, Version 4.3*, 2011.
[71] *SOAP Version 1.2*, 2007.
[72] *Java(TM) Message Service Specification Final Release 1.1*, 2002.
[73] *JSR-000322 Java EE Connector Architecture 1.6 1.6 Final Release*, 2009.
[74] *Java Remote Method Invocation*, 2016.

[75] *Common Object Request Broker Architecture (CORBA) Specification, Version 3.1*, formal/2008-01-08, 2008.

[76] D. A. D'Mello and V. S. Ananthanarayana, "Dynamic selection mechanism for quality of service aware web services," *Enterprise Information Systems,* vol. 4, no. 1, pp. 23-60, 2010.

[77] F. Baude, L. Henrio, and C. Ruz, "Programming distributed and adaptable autonomous components—the GCM/ProActive framework," *Software: Practice and Experience,* pp. n/a-n/a, 2014.

[78] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Volume 2, Patterns for Concurrent and Networked Objects*. Wiley, 2000.

[79] L. Braubach and A. Pokahr, "Addressing Challenges of Distributed Systems Using Active Components," in *Intelligent Distributed Computing V: Proceedings of the 5th International Symposium on Intelligent Distributed Computing – IDC 2011, Delft, The Netherlands – October 2011*, F. M. T. Brazier, K. Nieuwenhuis, G. Pavlin, M. Warnier, and C. Badica, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 141-151.

[80] P. Horn, *Autonomic Computing: IBM's Perspective on the State of Information Technology*. 2001.

[81] A. Rausch, R. Reussner, R. Mirandola, and F. Plasil, *The Common Component Modeling Example: Comparing Software Component Models*. Springer Publishing Company, Incorporated, 2008, p. 460.

[82] W. Emmerich, M. Aoyama, and J. Sventek, "The impact of research on the development of middleware technology," *ACM Trans. Softw. Eng. Methodol.,* vol. 17, no. 4, pp. 1-48, 2008.

[83] P. Merle and J.-B. Stefani, "A formal specification of the Fractal component model in Alloy," 2008-00-00, Available: http://hal.inria.fr/inria-00338987.

[84] T. Bureš, P. Hnětynka, and F. Plášil, "SOFA 2.0: Balancing advanced features in a hierarchical component model," 2006, pp. 40-48.

[85] J. Kofron, "Behavior Protocols Extensions," Software Engineering, Charles University in Prague, 2007.

[86] F. Plasil and S. Visnovsky, "Behavior protocols for software components," *IEEE Transactions on Software Engineering,* vol. 28, no. 11, pp. 1056-1076, 2002.

[87] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J. B. Stefani, "Reconfigurable SCA Applications with the FraSCAti Platform," in *Services Computing, 2009. SCC '09. IEEE International Conference on*, 2009, pp. 268-275.

[88] L. Seinturier, P. Merle, R. Rouvoy, D. Romero, V. Schiavoni, and J.-B. Stefani, "A component-based middleware platform for reconfigurable service-oriented architectures," *Software: Practice and Experience,* pp. n/a-n/a, 2011.

[89] C. P. Shelton and P. Koopman, "Improving system dependability with functional alternatives," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2004, pp. 295-304.

[90] M. Hammer and A. Knapp, "Correct Execution of Reconfiguration for Stateful Components," *Electronic Notes in Theoretical Computer Science,* vol. 260, no. 0, pp. 91-108, 1/1/ 2010.

[91] N. Roy, N. Shankaran, and D. C. Schmidt, "Bulls-eye - A resource provisioning service for enterprise distributed real-time and embedded systems," in *OTM Confederated International Conferences, CoopIS, DOA, GADA, and ODBASE 2006* vol. 4276 LNCS - II, ed. Montpellier, 2006, pp. 1843-1861.

[92] N. Shankaran *et al.*, "A framework for (re)deploying components in distributed real-time and embedded systems," presented at the Proceedings of the 2006 ACM symposium on Applied computing, Dijon, France, 2006.

[93] V. Subramonian *et al.*, "The design and performance of component middleware for QoS-enabled deployment and configuration of DRE systems," *Journal of Systems and Software,* vol. 80, no. 5, pp. 668-677, 2007.

[94] W. R. Otte, A. Gokhale, and D. C. Schmidt, "Efficient and deterministic application deployment in component-based enterprise distributed real-time and embedded systems," (in English), *Information and Software Technology,* vol. 55, no. 2, pp. 475-488, 2013.

[95] M. Garcia Valls, I. R. Lopez, and L. F. Villar, "iLAND: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems," *Industrial Informatics, IEEE Transactions on,* vol. 9, no. 1, pp. 228-236, 2013.

[96] I. Estevez-Ayres, P. Basanta-Val, M. Garcia-Valls, J. A. Fisteus, and L. Almeida, "QoS-Aware Real-Time Composition Algorithms for Service-Based Applications," *Industrial Informatics, IEEE Transactions on,* vol. 5, no. 3, pp. 278-288, 2009.

[97] M. García-Valls, P. Basanta-Val, and I. Estévez-Ayres, "A component model for homogeneous implementation of reconfigurable service-based distributed real-time applications," in *10th Annual International Conference on New Technologies of Distributed Systems, NOTERE'10*, Tozeur, 2010, pp. 267-272.

[98] J. C. Romero and M. García-Valls, "Scheduling component replacement for timely execution in dynamic systems," *Software: Practice and Experience,* pp. n/a-n/a, 2013.

[99] L. Almeida, S. Fischmeister, M. Anand, and I. Lee, "A dynamic scheduling approach to designing flexible safety-critical systems," presented at the Proceedings of the 7th ACM & IEEE international conference on Embedded software, Salzburg, Austria, 2007.

[100] A. v. Hoorn, M. Rohr, A. Gul, and W. Hasselbring, "An adaptation framework enabling resource-efficient operation of software systems," presented at the Proceedings of the Warm Up Workshop for ACM/IEEE ICSE 2010, Cape Town, South Africa, 2009.

[101] R. Von Massow, A. Van Hoorn, and W. Hasselbring, "Performance simulation of runtime reconfigurable component-based software architectures," in *5th European Conference on Software Architecture, ECSA 2011* vol. 6903 LNCS, ed. Essen, 2011, pp. 43-58.

[102] B. Orlic, I. David, R. Mak, and J. Lukkien, "Dynamically Reconfigurable Resource-Aware Component Framework: Architecture and Concepts," in *Software Architecture*, vol. 6903, I. Crnkovic, V. Gruhn, and M. Book, Eds. (Lecture Notes in Computer Science: Springer Berlin Heidelberg, 2011, pp. 212-215.

[103] F. Brosch, H. Koziolek, B. Buhnova, and R. Reussner, "Architecture-Based Reliability Prediction with the Palladio Component Model," *IEEE Transactions on Software Engineering,* vol. 38, no. 6, pp. 1319-1339, 2012.
[104] M. Henning, "The Rise and Fall of CORBA," *Queue,* vol. 4, no. 5, pp. 28-34, 2006.
[105] *Deployment and Configuration of Component-based Distributed Applications Specification v4.0*, 06-04-02, 2006.
[106] I. Calvo, L. Almeida, A. Noguero, F. Pérez, and M. Marcos, "A flexible time-triggered service for real-time CORBA," *Computer Standards & Interfaces,* vol. 36, no. 3, pp. 531-544, 3// 2014.
[107] P. Pedreiras, P. Gai, L. Almeida, and G. C. Buttazzo, "FTT-ethernet: A flexible real-time communication protocol that supports dynamic QoS management on ethernet-based systems," *IEEE Transactions on Industrial Informatics,* vol. 1, no. 3, pp. 162-172, Aug 2005.
[108] S. Richter, M. Wahler, and A. Kumar, "A Framework for Component-Based Real-Time Control Applications," in *13th Real-Time Linux Workshop, Prague, Czech Republic*, Prague, 2011.
[109] M. Wegdam, "Dynamic Reconfiguration and Load Distribution in Component Middleware (PhD Thesis)," University of Twente, Enschede, The Nederlands, 2003.
[110] *The Real-Time Publish-Subscribe Wire Protocol. DDS Interoperability Wire Protocol Specification (DDS-RTPS) v2.1*, 2009.
[111] *RFC 3550: RTP: A Transport Protocol for Real-Time Applications*, 2003.
[112] J. C. Knight, "Safety critical systems: challenges and directions," in *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, 2002, pp. 547-550.
[113] F. Carvalho, S. R. L. Meira, B. Freitas, and J. Eulino, "Embedded Software Component Quality and Certification," in *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*, 2009, pp. 420-427.
[114] *Service Component Architecture Assembly Model Specification Version 1.1*, 2011.
[115] *Apache Tuscany*. Available: http://tuscany.apache.org/
[116] *SCA Policy Framework Version 1.1*, 2011.
[117] A. Armentia, "Ingeniería Basada en Modelos aplicada a Sistemas Distribuidos Sensibles al Contexto," Ingeniería de Sistemas y Automática (UPV/EHU) ETSI Bilbao, Bilbao, 2016.
[118] S. K. Baruah, A. Burns, and R. I. Davis, "Response-Time Analysis for Mixed Criticality Systems," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, 2011, pp. 34-43.
[119] *Data Distribution Service for Real-time Systems v1.2*, 2007.
[120] *IEC 61784-3-3: Industrial communication networks - Profiles - Part 3-3: Functional safety fieldbuses - Additional specifications for CFP 3*, 2007.
[121] R. Rekik and S. Hasnaoui, "Application of a CAN BUS transport for DDS middleware," in *2nd International Conference on the Applications of Digital Information and Web Technologies, ICADIWT 2009*, London, 2009, pp. 766-771.
[122] A. Armentia, A. Agirre, E. Estevez, J. Pérez, and M. Marcos, "Model Driven Design Support for Mixed-Criticality Distributed Systems," presented at the

19th World Congress of the International Federation of Automatic Control, Cape Town, South Africa, 2014.

[123] *Data Distribution Service (DDS) v1.4*, 2015.

GLOSARIO

GLOSARIO

ADL	Lenguaje de descripción de la arquitectura (*Architecture Description Language*)
AGV	Vehículos de guiado automático (*Automatic Guided Vehicles*)
AOP	Programación orientada a aspectos (*Aspect Oriented Programming*)
API	Interfaz de programación de aplicaciones (*Application Programming Interface*)
AUTOSAR	Arquitectura abierta para automoción (*Automotive Open System ARchitecture*)
BD	Base de Datos
BPEL	Lenguaje de ejecución de procesos de negocio (*Business Process Execution Language*)
CAN	*Controller Area Network*
CASE	Ingeniería de software asistida por computador (*Computer Aided Software Engineering*)
CBSE	Ingeniería del software basada en componentes (*Component-Based Software Engineering*)
CCM	Modelo de componentes CORBA (*Corba Component Model*)
COAP	Protocolo para aplicaciones con recursos limitados (*Constrained Application Protocol*)

CORBA	*Common Object Request Broker Architecture*
CORBA D&C	*Corba Deployment and Configuration*
CPS	Sistemas ciber-físicos (*Cyber Physical Systems*)
CPU	Unidad Central de Proceso (*Central Processing Unit*)
CRC	Chequeo de redundancia cíclica (*Cyclic Redundancy Check*)
DAMP	Plataforma de gestión de aplicaciones distribuidas (*Distributed Applications Management Platform*)
DDS	*Data Distribution Service*
DOSGi	*Distributed OSGi*
EJB	*Enterprise Java Beans*
E2ERt	Tiempo de respueta de punto a punto (*End to End Response Time*)
FDB	Diagrama de bloques funcionales (*Function Block Diagram*)
FPS	Imágenes Por Segundo (*Frames Per Second*)
FTT	*Flexible Time Triggered*
GCIS	Grupo de Control e Integración de Sistemas
GCM	*Grid Component Model*
HW	*Hardware*
IaaS	Infraestructura como servicio (*Infrastructure as a Service*)

IDE	Entorno de desarrollo integrado (Integrated Development Environment)
IDL	Lenguaje de descripción de interfaces (Inteface Description Language)
IFTTT	Si esto… entonces aquello (*If This Then That*)
IoT	Internet de las cosas (*Internet of Things*)
IIoT	Internet industrial de las cosas (*Industrial Internet of Things*)
IP	*Internet Protocol*
JCA	*Java Enterpise Edition Connector Architecture*
JMS	Servicio de mensajería de Java (*Java Message Service*)
LAN	Red de área local (*Local Area Network*)
LTE	*Long Term Evolution*
MAPE	Monitorizar, analizar, planificar y Ejecutar (Monitor Analyze Plan Execute)
MDE	Ingeniería Dirigida por Modelos (*Model-Driven Engineering*)
MQTT	*Message Queue Telemetry Transp ort*
MLRM	Gestión de recursos multicapa (Multi Layer Resource Management)
M2M	Máquina a máquina (*Machine to Machine*)

MW_DAEMON	Demonio gestor de nodo (*Middleware Daemon*)
MW_DAEMON	Gestor de plataforma (*Middleware Manager*)
OASIS	*Organization for the Advancement of Structured Information Standards*
OPC-DA	*Ole (Object linking and embedding) for Process Control Data Access*
OPC-UA	*Ole (Object linking and embedding) for Process Control Unified Architecture*
OpenCCM	Modelo de componentes CORBA abierto (*Open Corba Component Model*)
OROCOS	Software abierto para control de robots (*Open RObot COntrol Software*)
OSGi	*Open Services Gateway Initiative*
OSI	Interconexión de Sistemas Abiertos (*Open System Interconnection*)
PC	Computador personal (Personal Computer)
PECOS	Sistema de componentes pervasivo (*PErvasive COmponent System*)
QoS	Calidad de Servicio (*Quality of Service*)
REST	Transferencia de Estado Representacional (*Representational State Transfer*)

RFID	Identificación por radio frecuencia (*Radio Frecuency IDentification*)
RMI	Invocación de método remoto (Java) (*Remote Method Invocation*)
ROBOCOP	Arquitectura software abierta y basada en components para dispositivos configurables (*Robust Open Component Based Software Architecture for Configurable Devices*)
RTA	Análisis de tiempo de respuesta (*Response Time Analysis*)
RTI	*Real Time Innovations*
RTP	Protocolo de transporte de tiempo real (*Real time Transport Protocol*)
RTPS	Protocolo publicador suscriptor de tiempo real (*Real Time Publish Subscribe*)
SaaS	Software como servicio (*Software as a Service*)
SCA	*Service Component Architecture*
SCADA	Control de supervisión y adquisición de datos (Supervisory Control and Data Adquisition)
SCL	Capa de comunicaciones con seguridad funcional integrada (*Safety Communication Layer*)
SGA	Sistema de gestión de almacenes
SLA	*Service Level Agreement*
SOAP	Protocolo simple de acceso a objetos (*Simple Object Access Protocol*)
SOA	Arquitectura orientada a Servicio (*Service Oriented Architecture*)

SOFA	*SOFtware Appliances*
SW	*Software*
TCP	Protocolo de control de transmisión (*Transmission Control Protocol*)
UDP	*User Datagram Protocol*
UML	*Unified Modeling Language*
UPNP	*Universal Plug&Play*
VPN	Red privada virtual (*Virtual Private Network*)
WAN	Red de área extensa (*Wide Area Network*)
WCET	Tiempo de ejecución de peor caso (*Worst Case Execution Time*)
WS	Servicio web (*Web Service*)
WSDL	Lenguaje de descripción de servicios web (*Web Service Description Language*)
XML	*eXtensible Markup Lenguaje*
4DIAC	*For Distributed Industrial Automation and Control*