UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Desarrollo de soluciones con Capacidades de Computación Autónoma en el ámbito de la Robótica Colaborativa

Trabajo Fin de Máster

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

AUTOR/A: Oliver Cortés, Francisco Javier

Tutor/a: Fons Cors, Joan Josep

Cotutor/a: Vendrell Vidal, Eduardo

CURSO ACADÉMICO: 2023/2024

Valencia, «Septiembre 2024»

Este Trabajo Fin de Máster se ha depositado en el Departamento de Sistemas Informàticos y Computación de la Universitat Politècnica de València para su defensa.

Trabajo Fin de Máster
Máster Universitario en Ingeniería y Tecnología de Sistemas Software

Título: Desarrollo de soluciones con Capacidades de Computación Autónoma en el ámbito de la Robótica Colaborativa

«Septiembre 2024»

Autor(a): Francisco Javier Oliver Cortés
Director(a): Joan Josep Fons Cors
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Resum

Context i Problemàtica

A mitjan la dècada del 2000 apareix una nova família de robots: els robots col·laboratius o cobots, que pretenen fer un pas endavant en l'àmbit de l'automatització industrial. Els cobots s'uneixen als robots industrials ja existents, aportant algunes millores amb l'objectiu d'aconseguir una major taxa d'aplicació industrial de solucions robotitzades. Una d'aquestes millores està relacionada amb la seguretat a l'hora de co-existir amb operaris humans, potenciant el treball col·laboratiu amb humans.

Un altre aspecte que introduïen aquests cobots era que, gràcies a la seua mida més reduïda, podien ser utilitzats en un major ventall de tasques. Com que la seua 'petjada' (espai requerit, restriccions de seguretat, etc.) en un procés productiu és molt menor, els converteix en una eina més interessant i versàtil per a un gran nombre de treballs d'automatització.

No obstant això, hui dia encara existeixen restriccions tècniques, operatives i de seguretat per a l'ús de robots en processos de producció industrial, sobretot per a escenaris col·laboratius Humà-Robot. Desenvolupar una solució d'assistència robotitzada suposa, a més, altres tipus de costos relacionats amb la part del 'software' de control (programació de les tasques robotitzades, comunicació amb altres sistemes, adaptació específica als processos productius, etc.). Això suposa un risc quant a la viabilitat d'aquest tipus de solucions, ja que la producció en una fàbrica és canviant amb el temps i els requisits.

Hui en dia, el desenvolupament de solucions robotitzades, des d'una perspectiva del software, sol requerir implementacions molt específiques, molt fixes. És la manera de poder testar i garantir els requisits operacionals i de seguretat imposats. Però tot açò va en detriment de la flexibilitat i adaptabilitat que es podria oferir amb aquest tipus d'entorns, per a donar suport a requisits canviants en els sistemes de producció industrial.

Objectius

En aquest treball pretenem explorar la possibilitat de canviar el focus a l'hora de construir solucions robotitzades per a poder definir processos robotitzats no tan fixes, més parametritzables i dinàmics, amb millors capacitats d'adaptació, amb l'objectiu de no requerir una reimplementació continuada de les tasques robotitzades.

L'objectiu principal és plantejar com podria ser un entorn d'execució robòtica basat en el concepte de components modulars i reutilitzables, amb tasques que puguen dissenyar-se en temps d'enginyeria i que es puguen compondre en temps d'execució. Per a això, ens basarem en la idea preliminar de definir 'tasques adaptatives' (o adap-

tive ready), com una tasca que es dissenya pensant en que es podrà 'compondre i configurar' en temps d'execució en coordinació amb altres tasques, per a donar suport a processos de producció.

Intentarem avaluar amb aquest treball fins a quin punt és viable tècnicament definir aquests processos robotitzats de manera més modular, i habilitar la seua composició en última instància, en temps d'execució. No obstant això, som conscients de la dificultat tècnica, estratègica i dels riscos a nivell de compliment de normatives existents en l'actualitat, que farien inviable aquest treball. Encara així, es pretén avançar en aquest camí, amb un enfocament clar en la composicionalitat dinàmica (en temps d'execució) que ens atorga la computació autònoma.

Estratègia de Solució i Validació

Per a dur a terme el treball, primer analitzarem els llenguatges per a la descripció de tasques robotitzades més habituals en l'àmbit de la programació robòtica. Proposarem una xicoteta extensió per a poder definir parametrització en les mateixes, i dinamicitat en la relació entre tasques i subtasques. Definirem un model de tasques basat en aquest nou concepte de tasca adaptativa, que inclourà aquestes extensions, i que serà clau per a desenvolupar la solució.

En aquest treball utilitzarem els bucles de control, utilitzant la proposta FADA MAPE-K (grup Tatami/PROS Institut VRAIN). Aquesta proposta ofereix un marc conceptual per a analitzar i descriure solucions amb capacitats de computació autònoma, i proporciona un 'framework' d'implementació per a desplegar solucions auto-adaptatives.

Per a exemplificar i validar el treball, es desenvoluparà un prototip funcional que puga ser desplegat en un entorn robotitzat simulat. En aquest cas d'estudi s'exemplificarà com podrien realitzar-se aquestes tasques col·laboratives entre robots, i com s'haurien /podrien dissenyar aquests programes robotitzats més modulars i reutilitzables.

El cas d'estudi permetrà avaluar l'adaptabilitat de les solucions a través de la capa de computació autònoma implementada amb la solució FADA MAPE-K, que s'encarregarà d'auto-configurar, auto-optimar, auto-protegir i auto-curar una solució robotitzada col·laborativa com la descrita, en base a aquestes tasques col·laboratives 'adaptive ready', en temps d'execució.

Paraules clau: Robòtica Col·laborativa Humà-Robot; Tasques Adaptatives; Computació Autònoma; Bucles de control MAPE-K.

Resumen

Contexto y Problemática

A mitad de la década del 2000 aparece una nueva familia de robots, los robots colaborativos o cobots, que pretenden dar un paso adelante en el ámbito de la automatización industrial. Se unen a los robots industriales ya existentes, y aportan algunas mejoras con el objetivo de conseguir una mayor tasa de aplicación industrial de soluciones robotizadas. Una de estas mejoras está relacionada con el aspecto de seguridad a la hora de co-existir con operarios humanos, potenciando el trabajo colaborativo con humanos.

Otro aspecto que introducían estos cobots era que, gracias a su tamaño más reducido, podían ser utilizados en un mayor abanico de tareas. Debido a que su 'huella' (espacio requerido, restricciones de seguridad, etc.) en un proceso productivo es mucho menor, lo convierten en una herramienta más interesante y versátil para un gran número de trabajos de automatización.

Pero hoy en día todavía existen restricciones técnicas, operativas y de seguridad para el uso de robots en procesos de producción industrial, sobretodo para escenarios colaborativos Humano-Robot. Desarrollar una solución de asistencia robotizada supone además otro tipo de costes relacionados con la parte del 'software' de control (programación de las tareas robotizadas, comunicación con otros sistemas, adaptación específica a los procesos productivos, etc.). Esto a su vez supone un riesgo en cuanto a la viabilidad de este tipo de soluciones, ya que la producción en una fábrica es algo cambiante en el tiempo y en los requisitos.

Hoy en día, el desarrollo de soluciones robotizadas, desde una perspectiva del software, suele requerir implementaciones muy específicas, muy fijas. Es la manera de poder testear y garantizar los requisitos operacionales, y de seguridad impuestos. Pero todo esto va en detrimento de la flexibilidad y adaptabilidad que se podría ofrecer con este tipo de entornos, de cara a soportar requisitos cambiantes en los sistemas de producción industrial.

Objetivos

En este trabajo pretendemos explorar la posibilidad de cambiar el foco a la hora de construir soluciones robotizadas para poder definir procesos robotizados no tan fijos, más parametrizables y dinámicos, con mejores capacidades de adaptación, con el objetivo de no requerir una reimplementación continuada de las tareas robotizadas.

El objetivo principal es plantear cómo podría ser un entorno de ejecución robótica basado en el concepto de componentes modulares y reutilizables, con tareas que puedan diseñarse en tiempo de ingeniería y que puedan componerse en tiempo de

ejecución. Para ello, nos basaremos en la idea preliminar de definir 'tareas adaptativas' (o adaptive ready), como una tarea que se diseña pensando en que se podrá 'componer y configurar' en tiempo de ejecución en coordinación con otras tareas, para dar soporte a procesos de producción.

Intentaremos evaluar con este trabajo hasta qué punto es viable técnicamente definir estos procesos robotizados de manera más modular, y habilitando su composición en última instancia, en tiempo de ejecución. Sin embargo, somos conscientes de la dificultad técnica, estratégica y los riesgos a nivel de cumplimiento de normativas existentes en la actualidad, que harían inviable este trabajo. Aún así, se pretende avanzar en este camino, con un enfoque claro en la composicionalidad dinámica (en tiempo de ejecución) que nos otorga la computación autónoma.

Estrategia de Solución y Validación

Para llevar a cabo el trabajo, primero analizaremos los lenguajes para la descripción de tareas robotizadas más habituales en el ámbito de la programación robótica. Propondremos una pequeña extensión para poder definir parametrización a las mismas, y dinamicidad en la relación entre tareas y subtareas. Definiremos un modelo de tareas que se base en este nuevo concepto de tarea adaptativa, que incluirá estas extensiones, y que será clave para desarrollar la solución.

En este trabajo utilizaremos los bucles de control, utilizando la propuesta FADA MAPE-K (grupo Tatami/PROS Instituto VRAIN). Esta propuesta ofrece un marco conceptual para analizar y describir soluciones con capacidades de computación autónoma, y proporciona un framework de implementación para desplegar soluciones auto-adaptativas.

Para ejemplificar y validar el trabajo, se desarrollará un prototipo funcional que pueda ser desplegado en un entorno robotizado simulado. En este caso de estudio se ejemplificará cómo podrían realizarse estas tareas colaborativas entre robots, y cómo se deberían/podrían diseñar estos programas robotizados más modulares y reutilizables.

El caso de estudio permitirá evaluar la adaptabilidad de las soluciones a través de la capa de computación autónoma implementada con la solución FADA MAPE-K, que se encargará de auto-configurar, auto-optimizar, auto-proteger y auto-curar una solución una solución robotizada colaborativa como la descrita, en base a estas tareas colaborativas adaptive ready, en runtime.

Palabras clave: Robótica Colaborativa Humano-Robot; Tareas Adaptativas; Computación Autónoma; Bucles de control MAPE-K.

Abstract

Context and Problematic

In the mid-2000s, a new family of robots emerged: collaborative robots or cobots, which aimed to advance industrial automation. These cobots joined the existing industrial robots, bringing several improvements with the goal of achieving a higher industrial adoption rate for robotic solutions. One of these improvements relates to safety when coexisting with human operators, enhancing collaborative work with humans.

Another feature introduced by these cobots is their smaller size, allowing them to be used in a broader range of tasks. Since their "footprint"(required space, safety restrictions, etc.) in a production process is much smaller, they become a more interesting and versatile tool for a wide range of automation tasks.

However, there are still technical, operational, and safety restrictions today when it comes to using robots in industrial production processes, especially in collaborative human-robot scenarios. Developing a robotic assistance solution also involves additional costs related to the control software (programming of robotized tasks, communication with other systems, specific adaptation to production processes, etc.). This, in turn, presents a risk regarding the viability of such solutions, as production in a factory changes over time and with varying requirements.

Currently, developing robotic solutions from a software perspective typically requires highly specific, fixed implementations. This is the way to test and ensure operational and safety requirements. However, this comes at the expense of flexibility and adaptability that such environments could offer in supporting changing requirements in industrial production systems.

Objectives

In this work, we aim to explore the possibility of shifting the focus when building robotic solutions to define less fixed, more parameterizable and dynamic processes, with better adaptation capabilities, in order to avoid the continuous reimplementation of robotic tasks.

The main objective is to propose how a robotic execution environment could be based on the concept of modular and reusable components, with tasks that can be designed during the engineering phase and composed at runtime. To do this, we will build on the preliminary idea of defining 'adaptive-ready' tasks, as a task designed with the possibility of being 'composed and configured' at runtime in coordination with other tasks to support production processes.

With this work, we will attempt to evaluate how technically feasible it is to define these robotized processes in a more modular way and enable their composition, ultimately, at runtime. However, we are aware of the technical, strategic difficulties, and risks in terms of compliance with current regulations, which could make this work unfeasible. Nonetheless, we intend to move forward along this path, with a clear focus on dynamic compositionality (at runtime) enabled by autonomous computing.

Solution Strategy and Validation

To carry out the work, we will first analyze the most commonly used languages for describing robotized tasks in robotic programming. We will propose a small extension to allow parametrization and dynamism in the relationship between tasks and subtasks. We will define a task model based on this new concept of adaptive-ready tasks, which will include these extensions and will be key to developing the solution.

In this work, we will use control loops, employing the FADA MAPE-K proposal (Tatami/PROS group, VRAIN Institute). This proposal offers a conceptual framework for analyzing and describing solutions with autonomous computing capabilities and provides an implementation framework for deploying self-adaptive solutions.

To exemplify and validate the work, we will develop a functional prototype that can be deployed in a simulated robotic environment. In this case study, we will exemplify how these collaborative tasks between robots could be carried out and how these more modular and reusable robotized programs should/could be designed.

The case study will allow us to evaluate the adaptability of the solutions through the autonomous computing layer implemented with the FADA MAPE-K solution, which will handle the self-configuration, self-optimization, self-protection, and self-healing of a collaborative robotic solution as described, based on these adaptive-ready collaborative tasks at runtime.

Keywords: Human-Robot Collaborative Robotics; Adaptive Tasks; Autonomous Computing; MAPE-K Control Loops.

Tabla de contenidos

Índice de figuras

Capítulo 1

Introducción

Las tareas de fabricación han existido durante toda la historia del ser humano, y han evolucionado de manera constante con el objetivo de aumentar el rendimiento, abaratar costos y reducir plazos. A su vez, los avances tecnológicos también fueron progresando hasta que, en los años 60, se creó el invento que revolucionaría la forma en la que funcionaba la industria hasta el momento, se creó el que sería uno de los grandes aliados de la humanidad (al menos hasta ahora): los robots.

Estos aliados tecnológicos ofrecían diversas funcionalidades. Nos ayudaban a soldar piezas de manera automatizada o podían realizar las tareas más exigentes, como cargar objetos pesados, todo esto con un plantel más reducido de trabajadores, con menos fallos y a una mayor velocidad. Los robots cumplían con todo esto con la única necesidad de ser revisados regularmente para evitar que hicieran su tarea de manera defectuosa o que sufrieran averías.

Sin embargo, a mediados de los años 90, se creó un nuevo tipo de robot, uno que estaba especialmente diseñado para trabajar conjuntamente con humanos: los robots colaborativos [1]. Estos robots colaborativos, que a partir de ahora llamaremos cobots, fueron creados para mejorar la seguridad de las personas que trabajaban con ellos y facilitar su programación e instalación, ya que no tienen que ser tan especializados y pueden ser posicionados para hacer diversas tareas. Su principal motivo de creación era el de hacer más accesible el trabajo industrial, ya que estos eran más fáciles de trabajar con ellos, mejorar la seguridad trabajando con humanos, y hacer más económica la forma de trabajar, sobre todo para empresas no tan grandes, ya que un mismo cobot puede llevar a cabo diferentes tareas.

Pero, a pesar de facilitar mucho las tareas que tendríamos que hacer los humanos, siguen requiriendo de las personas para ser supervisados o programados. De hecho, al tratar este dato, encontramos un problema: distintas tareas necesitan distintas programaciones de los robots y cobots para ser llevadas a cabo.

Supongamos el caso del montaje de una puerta de un automóvil. Existen tareas como la alineación de piezas para su posterior fijación o tareas simples como coger o dejar una pieza antes y después de ser tratada. Para esto, existe una configuración concreta en la estación donde ocurre la tarea y en la que cada robot, cobot o incluso persona tiene asignado un proceso, ya sea soldar, coger, retirar, entre otras. Pero esta configuración es un programa que se ha dado a los robots para trabajar en esa tarea. ¿Qué pasaría si queremos cambiar la tarea? ¿Qué pasaría si quisiéramos cambiar

de esta tarea de montar puertas a otra, por ejemplo, al pulido y ajuste de chapa? Pues, para hacer esta nueva tarea, tendríamos que ajustar de nuevo manualmente la configuración de nuestros robots.

Este es el problema que queremos tratar. ¿Qué pasaría si pudiésemos ajustar de manera dinámica los recursos que tenemos y, así, agilizar el proceso de asignación de las tareas, ayudando a generar código que sea más reutilizable en los procesos, además de producir menos cantidad del mismo? Esto lo llevaremos a cabo mediante un bucle de control MAPE-K, con el que, mediante sondas que nos proporcionan información, monitores que tratan dicha información, el "knowledge" en el que se guardaría la información de los estados del bucle y las reglas de adaptación que creemos para cada tarea, podremos ajustar los componentes que necesita la tarea para llevarla a cabo.

Para esto también se enseñará el framework llamado ACR, que se ha creado para funcionar en el contexto industrial y donde se propone una solución para la manera en la que unimos diferentes componentes para llevar a cabo las tareas en función de los datos que se reporten del bucle de control MAPE-K.

1.1. Motivación

La motivación que existe detrás de este proyecto es poder ofrecer una solución ejemplificada de cómo podemos usar la estructura FaDa, propuesta por el grupo PROS/Tatami del instituto VRAIN/UPV, en un ejemplo simplificado pero real. El objetivo es dar un paso en cómo está actualmente el escenario de las adaptaciones en los sistemas que usan robots que colaboran para hacer tareas y cómo se podrían hacer estos bucles para facilitar el uso de robots y abaratar costes en la industria robótica.

Además, queremos destacar que estamos conscientes de todas las limitaciones que existen en cuanto a la seguridad y a los sistemas de adaptación de los robots. Sabemos que aún hay un arduo camino de investigación en torno a esto y que, por ende, no existe colaboración entre humanos y cobots, lo que hace imposible la forma de trabajo de humanos, cobots y robots en una misma tarea. Además, este es uno de los campos que más se está investigando en la Industria 5.0 [7]. Sin embargo, ese no es el objetivo principal del trabajo. El objetivo es demostrar cómo se puede conseguir que funcionen estas adaptaciones. Si conseguimos ayudar a que estas colaboraciones sean más seguras, tendríamos mucho camino recorrido.

1.2. Objetivos

Los objetivos de este trabajo son:

1. Estudio y definición de los distintos componentes de nuestro framework ACR que usaremos en nuestra solución, desde los robots, cobots y humanos que llevarán a cabo las tareas, hasta las propias tareas y las estaciones de trabajo, creando sus configuraciones y parámetros necesarios para su funcionamiento, además de explicar cuál fue el pensamiento para plantearlo de esa manera.
2. Se hará una estructura creando un bucle de control MAPE-K funcional. En este caso, usaremos una versión reducida del mismo, debido al gran consumo de tiempo que tendríamos si usáramos la versión completa. Usaremos el MAPE-K

Lite, que es una versión reducida de FaDa, en la que definiremos las diversas sondas, monitores, reglas y knowledges necesarios para el funcionamiento del mismo, usando OSGi en Java para generar un software flexible que pueda cambiar en tiempo de ejecución.

3. Finalmente, se utilizará la herramienta RoboDK para, usando Mosquitto para la creación de un broker MQTT y conseguir comunicarse con los scripts de Python que estarán en RoboDK, proporcionar un ejemplo visual del funcionamiento del bucle, llegando a ofrecer una simulación con robots reales de cómo funcionaría la solución.

1.3. Estructura de la memoria

La memoria estará estructurada en distintas secciones. Empezaremos explicando la situación actual de la industria y cómo el IoT se ha integrado a la misma y el rumbo que lleva, además del estado de los bucles de control MAPE-K. A continuación, explicaremos el diseño que hemos creado de nuestra solución, describiendo en primer lugar nuestro framework ACR y cómo y por qué hemos creado los distintos componentes, además de la estructura que seguirá. Explicaremos en más profundidad las herramientas que usaremos en el proyecto, desde los bucles MAPE-K y cómo funcionan, hasta cómo usaremos el programa de RoboDK y cómo hemos hecho el broker MQTT.

Después, enseñaremos el proceso de desarrollo de la solución que implementamos, cómo fue el proceso mental que seguimos para desarrollar el software. Aquí presentaremos cómo nos enfrentamos al problema de crear el MAPE-K, conectarlo con RoboDK mediante el broker MQTT y la creación de los escenarios con los robots.

Seguidamente, mostraremos los resultados que obtuvimos del desarrollo, lo cual es el ejemplo visual de cómo se pueden usar los bucles de control MAPE-K con cadenas de montaje, etc. En nuestro caso, mostraremos los ejemplos de las tareas colaborativas de pintar una pieza y de soldar una pieza.

Finalmente, concluiremos con unas ideas finales, enseñando y reflexionando sobre los resultados obtenidos y sobre cómo se plantea el futuro en este ámbito.

Capítulo 2

Estado del arte

2.1. Industria 4.0

Actualmente, la industria se encuentra en constante renovación. Gracias al Internet de las Cosas (IoT), se ha logrado un gran avance en este ámbito. Hoy en día, con la colaboración del IoT, nos encontramos en la Industria 4.0, que, según [8], se alcanzó al lograr una interacción humano-ordenador y se buscaba utilizar una red de dispositivos ciberfísicos que ayudaran a optimizar los procesos en los que se quería trabajar.

Sin embargo, ya se ha iniciado un gran avance hacia la Industria 5.0, gracias a la irrupción de la inteligencia artificial (IA). Lo que busca esta Industria 5.0 es mejorar la interacción entre humanos y máquinas. Según [8], se pretende que las máquinas inteligentes aporten su rapidez y precisión en los procesos, mientras que los humanos contribuyan con su creatividad, pensamiento crítico y mentalidad innovadora. Aquí es donde nuestro trabajo entra en juego, ofreciendo una manera de generar una estructura inteligente donde el sistema se adapte para realizar la tarea de la mejor manera posible, además de simplificar el trabajo en procesos multirrobot.

Para ello, utilizaremos dos tecnologías que han sido objeto de investigación en numerosos estudios: el bucle de control MAPE-K para la generación de nuestro software adaptativo y MQTT para la comunicación con los robots.

2.2. Software adaptativo. Bucle de control MAPE-K

2.2.1. Sistemas adaptativos

La evolución hacia la Industria 4.0 ha traído consigo una transformación radical en la forma en que las industrias operan, permitiendo la digitalización completa de procesos y la integración entre sistemas físicos y digitales. Esta forma de trabajar se caracteriza por el uso intensivo de tecnologías avanzadas como el IoT, la IA, el análisis de datos en tiempo real, la robótica avanzada y la automatización inteligente. Todo ello ha generado una necesidad urgente de soluciones tecnológicas que no solo sean capaces de automatizar tareas complejas, sino que también tengan la capacidad de adaptarse dinámicamente a las condiciones cambiantes del entorno.

Las soluciones adaptativas, en particular los sistemas auto-adaptativos, son sistemas

que ajustan automáticamente su comportamiento en respuesta a cambios internos o externos. Estos sistemas están diseñados para monitorizar constantemente el entorno, analizarlo en tiempo real y tomar decisiones por sí solos para optimizar su rendimiento, mantener la fiabilidad y cumplir con los requisitos del usuario o de la operación, sin intervención humana. Estos sistemas son extremadamente necesarios en la Industria 4.0 debido a la complejidad de los procesos y su variabilidad, donde la adaptabilidad es crucial para mejorar la eficiencia, escalabilidad y resiliencia frente a perturbaciones o fluctuaciones en las demandas del entorno.

Uno de los principales desafíos en el ámbito industrial moderno es gestionar eficientemente sistemas que están en constante evolución. Además, el sistema debe ser capaz de gestionar correctamente todos los datos generados por sensores, máquinas y otros dispositivos conectados.

Los sistemas tradicionales no adaptativos tienden a tener limitaciones importantes en su capacidad para manejar situaciones imprevistas, lo que puede llevar a procesos detenidos o incluso situaciones peligrosas.

2.2.2. Arquitecturas y Técnicas para la Adaptabilidad

El diseño y desarrollo de soluciones adaptativas es un área compleja que requiere planificación e implementación de arquitecturas flexibles. Un enfoque comúnmente utilizado para implementar sistemas adaptativos es el ciclo de control MAPE-K (Monitorización, Análisis, Planificación y Ejecución, con un Knowledge), que se ha convertido en un pilar fundamental en la creación de sistemas auto-adaptativos, como se destaca en [3], del que se hablará más adelante en profundidad.

En [3], los autores proporcionan un análisis de las diferentes técnicas y metodologías que se han desarrollado para diseñar software y sistemas auto-adaptativos. El artículo resalta que, aunque el ciclo MAPE-K es central, las soluciones adaptativas también requieren una arquitectura modular, escalable y lo suficientemente flexible como para permitir modificaciones en tiempo real sin interrumpir las operaciones críticas del sistema.

El artículo subraya la importancia de las arquitecturas basadas en componentes, que permiten una evolución gradual del sistema. Estas arquitecturas se benefician de la encapsulación, donde cada componente es autónomo y puede ser reemplazado o actualizado sin afectar al sistema en su conjunto.

Además de las arquitecturas basadas en componentes, se discuten otros enfoques, como las arquitecturas orientadas a servicios (SOA) y las arquitecturas basadas en microservicios, que son especialmente relevantes en sistemas distribuidos, ya que facilitan la adaptación y escalabilidad del sistema al permitir la integración de nuevas funcionalidades y la adaptación a diferentes cargas de trabajo sin una reconfiguración completa del sistema, pero a una menor escala que la arquitectura basada en componentes.

Una de las tendencias más significativas en el desarrollo de soluciones adaptativas para la Industria 4.0 es la integración de la inteligencia artificial (IA) y técnicas de aprendizaje automático. Estas tecnologías permiten a los sistemas no solo reaccionar ante cambios en tiempo real, sino también predecir posibles eventos antes de que ocurran. Al aprovechar grandes volúmenes de datos históricos y en tiempo real,

los algoritmos de aprendizaje automático pueden identificar patrones complejos y generar modelos predictivos que anticipan problemas operativos, como fallos en los equipos.

En [3], se enfatiza que la IA y el aprendizaje automático son esenciales para mejorar la capacidad adaptativa de los sistemas, ya que permiten una adaptación proactiva en lugar de reactiva, es decir, intentar predecir los fallos antes que tratar de resolverlos cuando ya han ocurrido.

El artículo revisado identifica una serie de métricas comúnmente utilizadas para evaluar las soluciones adaptativas, incluyendo la eficacia de la adaptación, el tiempo de respuesta y la estabilidad del sistema bajo diferentes cargas. Estas métricas deben ser observadas para asegurar la calidad del servicio y que se cumpla correctamente con la tarea.

2.2.3. MAPE-K

Comenzando con el bucle de control MAPE-K, desarrollado por IBM, esta arquitectura nos ayuda a generar un software que sea autorregulable y dinámico. Como explica [9], las categorías en las que debe ser el sistema autogestionable son:

1. Auto-configurable: El sistema debe ser capaz de adaptarse de manera dinámica a cambios en el entorno.
2. Auto-reparable: El sistema debe poder descubrir, diagnosticar y reaccionar ante casos no valorados y fallos.
3. Auto-optimizable: El sistema debe poder monitorizar y manejar los recursos de manera totalmente automática.
4. Auto-protegido: El sistema debe anticiparse, detectar, identificar y protegerse ante peligros.

El bucle de control MAPE-K debe sus siglas a “Monitor-Analyze-Plan-Execute-Knowledge”, los cuales son las distintas fases por las que trascurre bucle.

1. Monitor: Se encarga de gestionar y recolectar los datos que recibe el bucle, por ejemplo, de sondas.
2. Analyze: El sistema analiza los cambios que se han producido para llevar a cabo las adaptaciones necesarias.
3. Plan: Aquí es donde se gestionan los componentes necesarios para cumplir un objetivo.
4. Execute: Finalmente, se ejecutan los componentes para llevar a cabo la solución propuesta.
5. Knowledge: Aquí se guarda toda la información relevante para el bucle de control para poder aplicar los cambios.

En este proyecto, utilizaremos un framework proporcionado por el grupo Tatami, del Instituto VRAIN de Valencia, llamado FADA. Sin embargo, con el fin de evitar complicar demasiado la solución, ya que el objetivo de este trabajo reside más en una demostración práctica de cómo se puede trabajar con esta tecnología, utilizaremos una versión reducida del mismo, a la que denominaremos MAPE-K Lite.

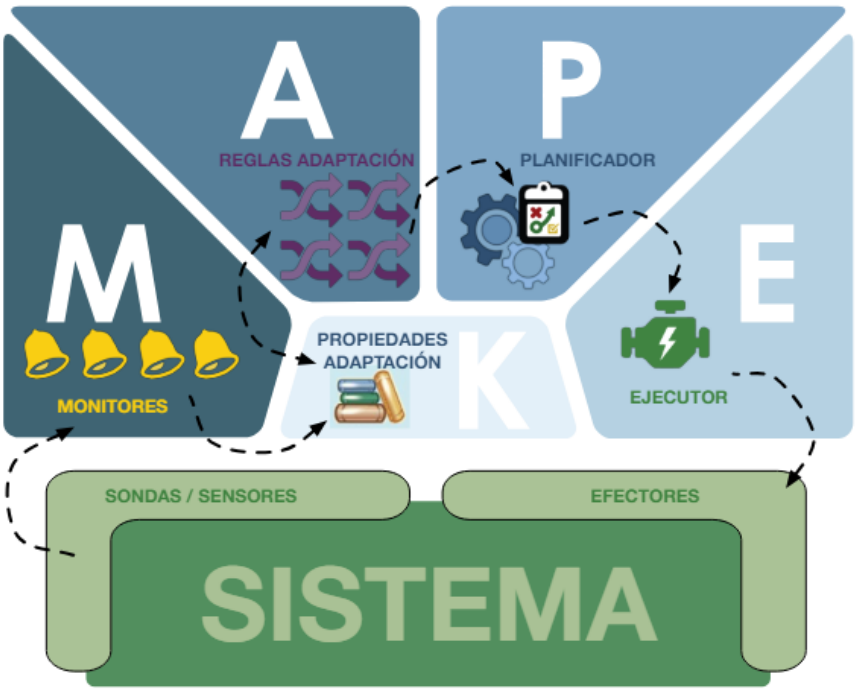

Figura 2.1: Ilustracion de bucle MAPE-K. Ilustración de [10]

2.3. Entornos de simulación para Robotización de Procesos Industriales

La robotización de procesos industriales ha sido uno de los pilares fundamentales de la Industria 4.0. Este avance ha permitido integrar robots con sistemas automatizados. La robotización no solo busca reemplazar tareas manuales o repetitivas, sino también optimizar procesos y mejorar la precisión en entornos industriales. La combinación de tecnologías como el IoT(Internet de las cosas), la IA (inteligencia artificial) y la robótica avanzada ha hecho posible la existencia de fábricas inteligentes que operan de manera autónoma.

Los objetivos principales de la robotización son mejorar la eficiencia, aumentar la precisión, reducir costes y proporcionar flexibilidad operativa.

Los entornos de simulación y programación de robots han adquirido una importancia crucial para la robotización de procesos industriales. Estas herramientas permiten diseñar, programar y simular robots de manera eficiente antes de implementarlos en entornos de producción reales. Con estas herramientas, se pueden reducir los tiempos de puesta en marcha, minimizar errores y mejorar la eficiencia de la producción. Entre los entornos más destacados se encuentran herramientas como RoboDK [17], ABB RobotStudio [4], y Siemens Tecnomatix Process Simulate [5]. En este trabajo, se utilizará RoboDK, que se verá más adelante.

Los entornos de simulación para la robotización industrial son plataformas de software que permiten diseñar y programar robots en un entorno virtual antes de su implementación física. Estas plataformas permiten la programación de robots sin necesidad de estar conectados a ellos, y ofrecen simulaciones detalladas para probar los movimientos, el alcance y las secuencias de trabajo de los robots en condiciones

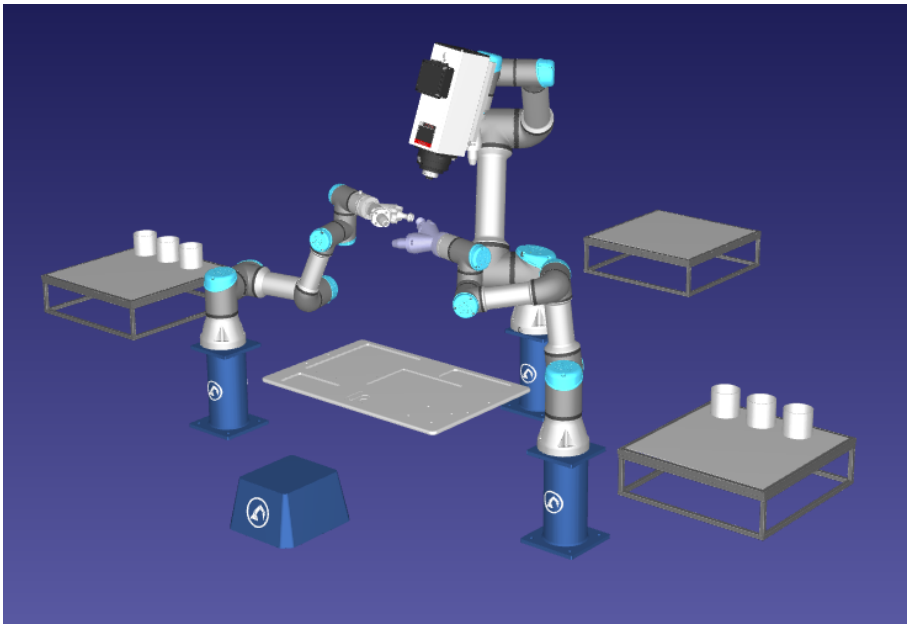

Figura 2.2: Ejemplo de estacion en RoboDK

virtuales que replican el entorno de trabajo real. Los entornos de simulación y programación permiten identificar y corregir problemas antes de que los robots trabajen en el mundo real, lo que reduce significativamente los riesgos de error, los costes asociados a las pruebas físicas y los tiempos de parada.

2.3.1. RoboDK

RoboDK [17] es uno de los software líderes de simulación y programación offline para robots industriales. Permite a los usuarios crear y probar programas de robots en un entorno virtual sin necesidad de tener el robot físicamente disponible. Este entorno es compatible con una amplia gama de marcas y modelos de robots, lo que lo convierte en una herramienta versátil.

Entre sus principales características, RoboDK ofrece la posibilidad de optimizar trayectorias, realizar análisis de colisiones y generar automáticamente el código necesario para los controladores de robots, facilitando así la implementación rápida y eficiente de proyectos robóticos.

RoboDK es fácil de usar, con una interfaz gráfica intuitiva que no requiere experiencia avanzada en robótica para comenzar, lo que ha sido una de las razones que ha llevado a su uso en este proyecto, evitando perder tiempo aprendiendo la herramienta.

2.4. MQTT

Finalmente, necesitamos una manera de comunicar nuestro sistema con nuestro robot. Para esto, utilizaremos el protocolo MQTT. Como se menciona en [11], este protocolo está diseñado para su uso en el Internet de las Cosas. Funciona mediante TCP/IP y envía texto, código binario, XML o JSON mediante un mecanismo de publicación-suscripción. Las partes que componen el protocolo son:

1. Un broker MQTT (en nuestro caso, usaremos el broker open source Mosquitto).
2. Los clientes MQTT que tienen la posibilidad de suscribirse o publicarse a tópicos (usaremos la librería PAHO para crear estos clientes que puedan realizar estas acciones).

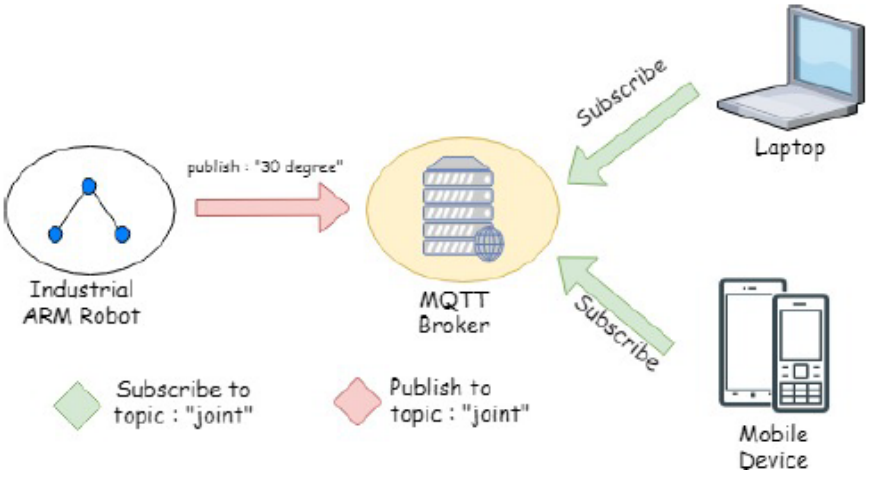

Figura 2.3: Ilustracion de funcionamiento del protocolo MQTT. Ilustración de [11]

Como podemos ver en la imagen 2.3, el funcionamiento del protocolo consiste en que varios dispositivos se suscriben al broker MQTT, mientras que otros clientes pueden publicar valores en el broker. Cuando un valor se publica en un tema particular (en el caso de la imagen, sería "joint"), todos los clientes que estén suscritos a ese tema recibirán el valor enviado por el broker. En este trabajo, funcionará de manera que, cuando se asigne una tarea a los robots (por ejemplo, recoger una pieza), al ejecutarse la tarea colaborativa y ser el turno del robot para realizar la acción asignada, podremos utilizar RoboDK para simular la tarea.

Capítulo 3

Análisis

3.1. Solución robotizada actual

En este capítulo del proyecto, explicaremos cómo hemos pensado, evaluado y trabajado en la selección de las herramientas utilizadas para la creación de nuestra solución, centrándonos en la implementación del framework ACR (Autonomous Collaborative Robots). También profundizaremos en los distintos componentes desarrollados, que se abordarán en el apartado de diseño, donde se valorarán sus conexiones y parámetros.

Comenzaremos hablando del framework ACR. Este framework, creado por Joan Josep Fons Cors e inspirado en la herramienta Papyrus, es un modelo open source utilizado para diversas aplicaciones, siendo uno de los campos más relevantes la industria. A través de ACR, se nos proporciona una forma estructurada de interactuar con los robots y todo el ecosistema industrial, desde las tareas hasta las posiciones en las que deben estar los trabajadores o robots para llevar a cabo dichas tareas.

El framework cuenta con distintos componentes, cada uno con una función específica. La estructura de estos componentes es simple: cada uno tiene conexiones definidas como Requires y Provides. Los Requires son las dependencias de servicios que ofrece otros componentes, necesarias para que un componente pueda cumplir su función. Por otro lado, los Provides son los servicios que el componente ofrece a otros elementos del sistema.

Con los componentes ACR encontramos una forma de encapsular las distintas partes de nuestra solución, facilitando la unión de los componentes necesarios en tiempo de ejecución.

El framework ACR nos permitió abordar varias cuestiones clave. Primero, necesitábamos una forma de conectar a los Task Doers, que representan a los trabajadores, ya sean robots o personas, capaces de llevar a cabo diversas tareas. Además, debíamos ser capaces de integrar las distintas Tasks, es decir, las tareas unitarias que contribuyen a la ejecución de una tarea colaborativa. Por ejemplo, al soldar dos piezas, podemos desglosar el proceso en tres tareas unitarias: suministrar (ambas piezas), soldar y, finalmente, retirar.

También necesitábamos un espacio donde llevar a cabo estas tareas y un mecanismo para ubicar a los Task Doers en dicho entorno de trabajo.

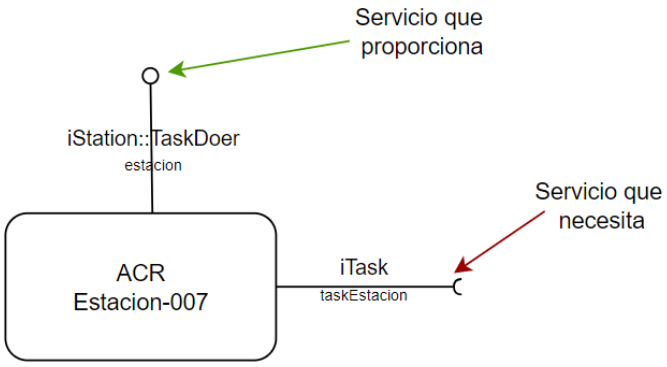

Figura 3.1: Dibujo de ejemplo de un componente

Comenzando con los Task Doers, estos pueden ser de tres tipos según la consideración que hemos adoptado para el trabajo: humanos, robots colaborativos y robots industriales. Los humanos se tratarán como componentes que nosotros, en este contexto ficticio, indicaremos cuando hayan completado su tarea. En cambio, los robots, aunque se mostrarán visualmente en RoboDK, podrían ser brazos robóticos reales utilizados para ejecutar las tareas.

Los humanos tendrán ciertos atributos que determinarán si pueden realizar una tarea o no. En nuestro caso, hemos simplificado estos filtros para que únicamente sean la altura del humano, que permite saber en qué posición operar, y las tareas que saben hacer; por ejemplo, algunos podrán pintar, otros soldar. En función de las habilidades que tengan, podrán conectarse para llevar a cabo una tarea o no. Los robots tendrán otras características, como su rango o el peso máximo que pueden soportar.

Ahora que tenemos los Task Doers, necesitamos las tareas que llevarán a cabo. Las tareas son las distintas acciones que se pueden realizar. Aquí entrarían desde pintar una pieza hasta soldarla. Como vemos en 3.2, una tarea cuenta con actividades previas y posteriores, que se realizan antes de la tarea principal (por ejemplo, cargar el robot con el gripper correcto o preparar la pintura) y después (ejemplos pueden ser apagar o desconectar el robot o aplicar alguna técnica de enfriamiento). Cabe destacar que estas tareas no son obligatorias, ya que puede que la tarea no requiera nada antes ni después. Además, debe existir alguna tarea que se ejecute en caso de que algo falle; esta es la tarea fallback (un ejemplo podría ser volver a la posición inicial o detener directamente los procesos que estén en curso, sin importar la posición).

Existen en el framework distintas tipos de Task como vemos en 3.3.

1. La tarea Task es síncrona, tiene un esquema de ejecución y su método doTheTask() en el cual podemos implementar lo que hará la tarea. Esta tarea se ejecutará en el hilo principal y avisará cuando acabe con un evento onTaskCompleted. Usaremos este tipo de tareas para las secuenciales, como las de los robots, que al acabar automáticamente notifican.
2. La tarea Async Task es asíncrona. Esta tarea extiende de Task, pero al ser

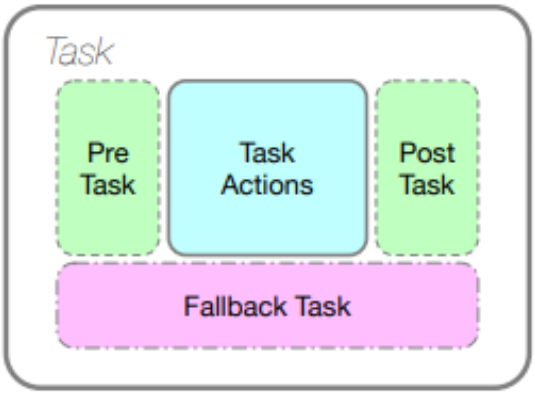

Figura 3.2: Dibujo explicativo del las partes de una task

asíncrona, se ejecuta en otro hilo para evitar bloquear el hilo principal y también emite un evento onTaskCompleted al acabar. Usaremos estas tareas en el caso de que varias se tengan que hacer a la vez, como cuando dos robots cojan una pieza simultáneamente para en el siguiente paso soldarlas.

3. La tarea Long Lived Task es asíncrona, pero a diferencia de la anterior, no emite ningún evento al acabar. Esta necesita que la tarea que la ha implementado la invoque explícitamente. Usaremos estas tareas con los humanos. Estos deben tener alguna manera de notificar que han terminado de hacer su tarea, ya sea con un botón u otro dispositivo, indicando explícitamente que han concluido.

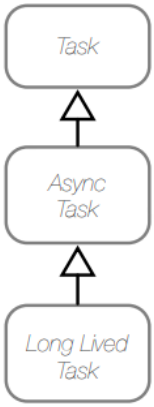

Figura 3.3: Diagrama de los tipos de task

Todas las Task, independientemente de su tipo, necesitan de alguien que las lleve a cabo, ya sea un robot industrial, colaborativo o una persona.

A continuación, presentamos las tareas colaborativas, que serían más bien el proceso que queremos realizar en su plenitud. Estas tareas extienden de las tareas antes vistas, ya que su lógica es similar: llevar una tarea a cabo. Lo único que esta no realiza la acción, sino que estará completa cuando las tareas con las que se une acaben.

Tendremos que unir las tareas necesarias con la tarea colaborativa, de modo que

al realizar, por ejemplo, la tarea de soldar varias piezas, esta pueda tener una de las configuraciones que incluya las Task que Suministrar1, Suministrar2, Soldar y Retirar al final. Todas las tareas colaborativas deben estar conectadas a las Task, ya que estas serán las que tengan asignados los Task Doers y realizarán las operaciones que tienen asignadas.

Ahora, lo siguiente que se nos plantea es un lugar donde deberemos llevar a cabo las tareas. Para ello, creamos el componente Station. Una Station es el lugar donde emplazaremos a los Task Doers. Pero para ello, también necesitamos saber qué posiciones están disponibles en la Station, ya que debemos ser capaces de verificar que los trabajadores se encuentran en el lugar correcto y preparados en la Station para llevar a cabo la tarea. Por esta necesidad, nacen también los Placements.

Una Station tendrá diversos Placements, dependiendo de las tareas que suponemos que podrán llevar a cabo en la Station. Por ejemplo, una Station que podrá ser usada para soldar piezas, pero también para pintar, necesitará que existan las posiciones de suministrar, tanto 1 como 2, la de soldar y la de retirar para la tarea de soldar varias piezas. Asimismo, deberá disponer de las posiciones de suministrar y retirar como antes, pero también una de pintar, para llevar a cabo la segunda tarea. En ese caso, antes de llevar a cabo una tarea colaborativa,

3.2. Tipos de adaptaciones

A continuación, se analizarán los diferentes tipos de adaptaciones que los robots industriales y colaborativos pueden implementar para mejorar su eficiencia y su manera de interactuar. Estas adaptaciones se centran en tres áreas principales: el entorno, las tareas y las personas.

Una de las formas más básicas de adaptación que un sistema robótico debe realizar es a su entorno de trabajo, en particular, a los recursos disponibles y las condiciones que afectan su operativa. Se ha valorado que algunas de las cosas a las que debe estar pendiente el sistema son:

1. Recursos dinámicos: Los robots deben ser capaces de ajustarse a la disponibilidad de materiales y herramientas en tiempo real. Por ejemplo, en un entorno industrial, los robots adaptativos pueden detectar la falta de una pieza y ajustarse para realizar otras tareas mientras se soluciona el problema, lo que ayuda a mejorar los tiempos del proceso.

2. Entornos cambiantes: Además de los recursos, el entorno físico puede sufrir modificaciones. Las posiciones de los objetos, si varían, pueden hacer que los robots tengan que reajustar sus movimientos o comportamientos.

Otra forma de adaptación que se debe tener en cuenta es la capacidad de los robots para adaptarse a las tareas que se les asignan. Esto se hace para garantizar siempre la mejor adaptación en correspondencia con la productividad y el rendimiento. Por ejemplo:

1. Flexibilidad en las tareas: Un robot adaptativo debe ser capaz de cambiar el modo en que realiza una tarea según las circunstancias. Por ejemplo, al soldar una pieza, el robot puede ajustar la velocidad o el ángulo en función del tipo de

material o del espacio disponible para realizar la tarea. Esto permite a los robots ser más eficientes y reducir errores.

2. Optimización basada en KPIs: En un entorno productivo, los robots pueden recibir datos en tiempo real sobre indicadores clave de rendimiento (KPIs), que son parámetros como la velocidad de producción, la calidad del producto o el uso de energía. A partir de estos datos, los robots pueden ajustar sus operaciones para maximizar la productividad o mejorar la calidad.
3. Automatización de mantenimiento preventivo: Los robots también pueden adaptar su comportamiento en función de fallos o situaciones anómalas que pueden llegar a detectar tanto en ellos mismos como en la cadena de producción, pudiendo ajustar sus operaciones o incluso solicitar tareas de mantenimiento.

Figura 3.4: Colaboración de robot y personas [6]

Además, los robots colaborativos no solo interactúan con otros robots o máquinas, sino también con personas, por lo que es fundamental que estos robots se adapten a las características físicas y las necesidades de los humanos con los que trabajan.

1. Detección de humanos: Los robots adaptativos deben ser conscientes de la presencia y las acciones de los trabajadores humanos. A través de sensores, estos robots deben poder detectar a los humanos en su entorno inmediato y ajustar su comportamiento para evitar accidentes o mejorar la colaboración.
2. Ergonomía: Los robots deben poder adaptarse a las características individuales de los trabajadores, como su altura o fuerza. En tareas colaborativas donde se requiere que un robot y un humano trabajen juntos, el robot puede ajustar su posición o velocidad para mejorar la ergonomía del trabajador.
3. Aprendizaje colaborativo: Los robots también pueden aprender de los patrones de trabajo de los humanos. Si un trabajador prefiere realizar una tarea de cierta manera o a un ritmo particular, los robots adaptativos pueden ajustar su comportamiento para alinearse con estas preferencias.

Y aunque existen muchas otras adaptaciones que se pueden explorar, estas deberían ser básicas para la creación de un sistema adaptativo en el que se usen robots que pueden llegar a trabajar con personas.

3.3. Retos y riesgos

Si bien la adaptabilidad promete aumentar la eficiencia y flexibilidad de los robots industriales, surgen varios obstáculos que deben abordarse, como normativas, riesgos de seguridad y la falta de estándares industriales. Exploramos cómo estos desafíos impactan en la creación de soluciones basadas en el framework ACR.

El principal reto para la incorporación de capacidades adaptativas en robots industriales reside en las normativas y los riesgos de seguridad. Los entornos industriales requieren sistemas robustos y predecibles que operen bajo condiciones estrictamente controladas. Sin embargo, los robots adaptativos, que ajustan sus comportamientos de forma dinámica en respuesta a los cambios del entorno, añaden una capa de incertidumbre que las regulaciones actuales no cubren de manera adecuada. Las normativas actuales están diseñadas para robots con comportamientos predefinidos y no contemplan las implicaciones de la adaptabilidad.

Un riesgo evidente es que un robot adaptativo pueda tomar decisiones inesperadas que comprometan la seguridad de los trabajadores o dañen el entorno de trabajo. En sistemas colaborativos, donde los humanos interactúan de cerca con los robots, es necesario que existan medidas preventivas ante fallos en la adaptabilidad. A nivel de ciberseguridad, la conectividad necesaria para la adaptabilidad, especialmente con el uso de IoT o soluciones en la nube, puede incrementar la vulnerabilidad del sistema ante ataques, un tema que hay que tratar con extremo cuidado.

Otro problema que podemos encontrar es la interoperabilidad entre robots, ya que los fabricantes pueden trabajar con tecnologías propias al diseñar sus productos, lo que complica la integración. A esto hay que añadir que la mayoría de robots industriales del mercado actual no fueron concebidos con la idea de que fuesen adaptativos, por lo que la mayoría solo pueden llevar a cabo una tarea de manera repetitiva; sin embargo, este punto puede llegar a ser solucionado en muchos casos por los cobots.

Muchas de las soluciones robóticas actuales no están equipadas con sensores o tecnologías avanzadas necesarias para captar información en tiempo real y adaptar sus operaciones. En nuestro proyecto, aunque utilizamos simuladores como RoboDK para visualizar la interacción con los Task Doers, los robots reales suelen carecer de la flexibilidad necesaria para integrarse plenamente con un framework adaptativo como el ACR.

Con esto vemos que, actualizando las normativas e integrando las necesidades para llevar a cabo los sistemas adaptativos, como sensores en los robots, podemos hacer que estos sistemas puedan aprovechar plenamente el potencial de los robots en esta Industria 4.0.

Capítulo 4

Diseño

A continuación, empezaremos la parte en la que explicaremos cómo hemos diseñado nuestro proyecto, ya que, a la hora de trabajar con los bucles de control, estos controlan los componentes de un sistema; pero, para ello, debemos crear un sistema que controlar.

En este apartado, explicaremos cada uno de los componentes de nuestro framework, además de ilustrarlos con una representación gráfica de cómo se podría representar un componente ACR con sus Requires y Provides, que son los servicios que necesitan para funcionar y el servicio que ofrece, respectivamente. Además, posteriormente, explicaremos las sondas, monitores, reglas y knowledge que tendrá nuestro bucle MAPE-K, anteriormente explicado.

4.1. Componentes ACR

En este punto, vamos a explicar los distintos componentes identificados en nuestro sistema. Estos se conectarán o desconectarán entre ellos en función de las necesidades del sistema y de cómo se adapte.

La mejor parte de trabajar de esta manera es la autonomía que ofrece, debido a que cada parte está desarrollada por separado, lo que nos brinda la posibilidad de crear las configuraciones necesarias conectando los componentes que se deseen.

Esta posibilidad facilita mucho la forma de trabajar clásica, haciendo la lógica mucho más sencilla, ya que ya no se tienen que gestionar las distintas conexiones entre los componentes o piezas. El tema de las conexiones se encargará del bucle MAPE-K en función de los parámetros que le estén llegando, produciendo de manera automática las conexiones necesarias para llevar a cabo la tarea.

4.1.1. Task

Aquí veremos cómo hemos compuesto algunas de nuestras Task. En 4.1 podemos ver cómo son nuestros componentes Task que hemos creado para que sean realizadas por los cobots. Al ser estos iguales a los que existen para los componentes Task que llevarían a cabo humanos y robots industriales, únicamente variando el servicio que proporcionan, y para simplificar la ilustración, se ha decidido no mostrarlos. Este necesita estar conectado a la tarea colaborativa que se va a llevar a cabo; en nuestro

caso, sería la tarea soldarPiezaColaborativa. Además de eso, alguien debe llevarla a cabo; por ello, es necesaria la conexión con un Task Doer que se encuentre en posición y vaya a realizarla.

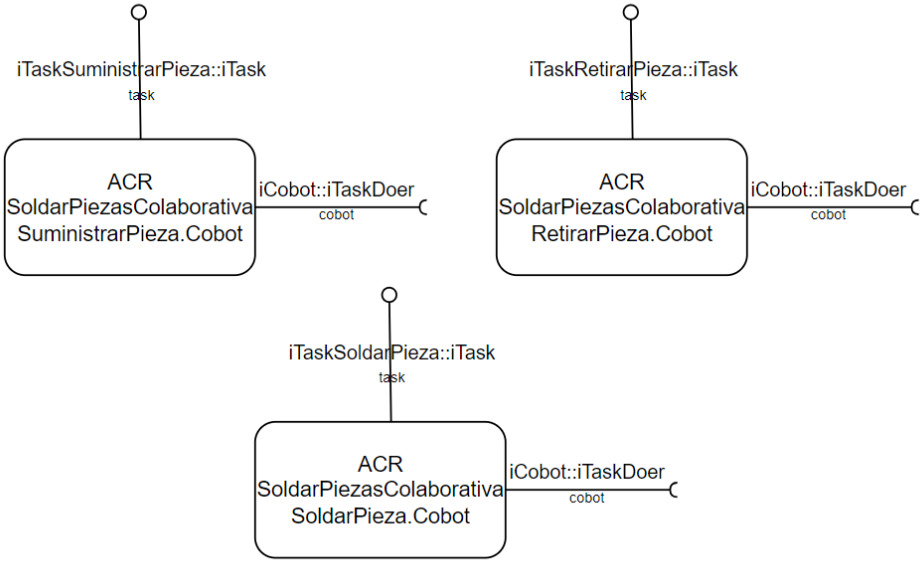

Figura 4.1: Ilustración del componente de la tarea retirar.

4.1.2. Task Colaborativa

Estas tareas necesitan para funcionar de un puesto de trabajo donde llevar a cabo dicha tarea, que es la station que proporcionamos. Además, también necesita que se le adjunten las distintas tareas que se requieren para llevar a cabo el proceso. En este caso mostrado de ejemplo 4.2, necesitamos que se adjunten dos tareas de suministrar, una de soldar y otra de retirar, pero no se destaca de quién debe ser, es decir, las tareas pueden ser llevadas a cabo en la combinación que se encuentre disponible. Si existe una tarea que tiene preparado un cobot para llevarla a cabo, por ejemplo, soldar, se hará unión con el componente que tiene la tarea SoldarPieza.Cobot. Lo mismo ocurre con las otras tareas necesarias y con los robots industriales y humanos.

Cabe destacar que estas son las piezas en torno a las que girarán nuestras adaptaciones, ya que si hemos adaptado otra tarea colaborativa que también realiza el proceso de soldar piezas, y esta tiene a sus trabajadores preparados para llevar a cabo las tareas, se elegirá esta para llevar a cabo la tarea. Si vemos como ejemplo 4.3, observamos que esta realiza las tareas de soldar y retirar en dos Tasks, mientras que en nuestra alternativa, se hacen con una sola. Con esto, nuestro bucle verá en todo momento los componentes que se encuentran disponibles, y en el momento en que alguna de las tareas colaborativas se convierta en una tarea posible de llevar a cabo, se elegirá esa.

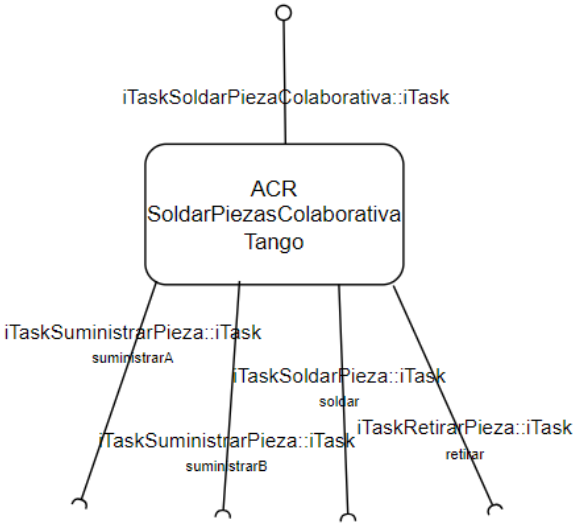

Figura 4.2: Ilustración del componente de la tarea colaborativa.

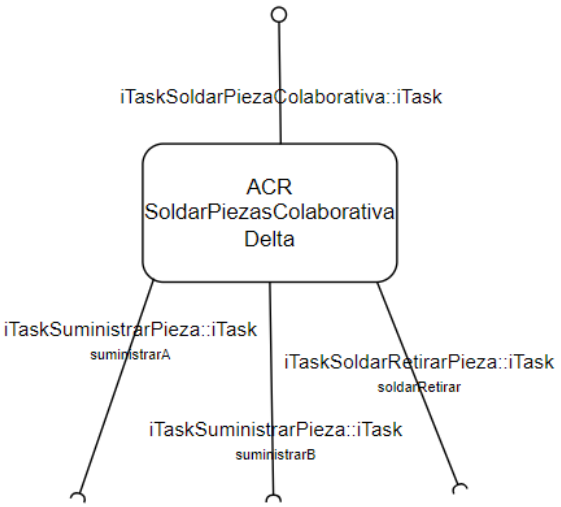

Figura 4.3: Ilustración del componente de la tarea colaborativa alternativa.

4.1.3. Task Doer

Estos componentes serían básicamente los que se encargarían de llevar a cabo las distintas tareas que se pueden producir en una tarea colaborativa. Siguiendo con el ejemplo anterior del proceso de soldar piezas, dos Task Doers se encargarían de suministrar las piezas que se van a soldar, otro llevaría a cabo la acción de soldar y el último retiraría la pieza ya soldada. Como podemos ver, no existe nada predefinido sobre qué Task Doer debe hacer qué tarea a priori, por lo que en muchos casos, varios Task Doers podrán realizar la misma tarea. Así, se contemplará cuál esté disponible en cada momento y cuál sea más válido.

De manera predeterminada, para nuestro caso, nosotros hemos generado tres tipos de Task Doers, que pueden ser robots industriales, robots colaborativos o humanos.

4.1.3.1. Humano

Este componente es fácil de adivinar qué es. Los trabajadores que pueden llevar a cabo tareas deben ser también considerados como componentes de nuestro sistema, ya que ejercen una función en este ecosistema. En nuestro caso de componentes, hemos tenido que considerar las limitaciones que puede tener una persona. Estas limitaciones, que hemos valorado de una manera básica, han sido las habilidades que tiene la persona (no puedes poner a una persona que no sabe pintar a pintar una pieza), además del rango en el que puede trabajar (si la persona mide 1.70 m, no podrá trabajar, por ejemplo, a una altura de 2.00 m).

Sabemos que existen muchos otros atributos que podríamos incluir, pero el objetivo del trabajo no es hacer un caso que se ajuste totalmente a la realidad de la industria, debido a que carecemos de conocimiento en el ámbito, sino más bien tomar algunos como ejemplo para ejemplificar cómo funcionarían estos.

Como podemos ver en 4.4, un Human necesita proporcionar una interfaz TaskDoer. Esta se podrá conectar posteriormente y ser usada por una Task, pero este deberá estar conectado a un iStationPlacement, el cual representa que está en la posición adecuada para llevar a cabo la tarea. Un ejemplo será que, a la hora de soldar, la persona no puede estar en la posición donde se va a llevar a cabo la tarea de retirar la pieza, sino que, si quiere llevar a cabo una tarea, debe estar donde se va a realizar esa tarea.

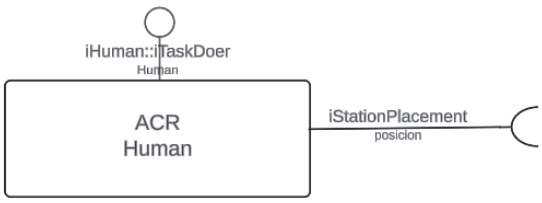

Figura 4.4: Ilustración del componente de humano.

4.1.3.2. Robot industrial

Al igual que hemos visto con el humano, el robot industrial también es un componente TaskDoer que puede llevar a cabo tareas. Estos representan los robots usados en la industria.

Como se podrá intuir, un robot también tendrá sus características para saber qué tareas puede llevar a cabo y cuáles no. En el caso de los robots, hemos considerado como atributos limitantes su rango espacial y de peso. Además, en este caso, hemos creado la posibilidad de parametrizar de manera dinámica las posiciones que tendrá el robot para llevar a cabo la tarea. Esto se debe a que, en la tarea, las posiciones en las que tendrá que llevar a cabo la tarea dependerán del punto de referencia que hayamos tomado, además de parámetros como dónde ha dejado la pieza el robot anterior o dónde está el objeto al inicio.

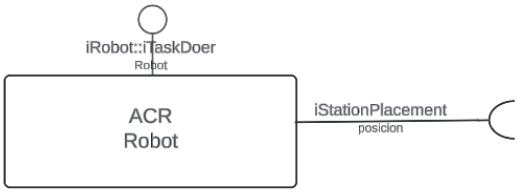

Figura 4.5: Ilustración del componente de robot industrial.

4.1.3.3. Robot colaborativo

Los siguientes son los robots colaborativos, que no son más que otra versión de los robots anteriormente explicados. Estos representan los robots que pueden colaborar con las personas por sus características, pero en términos de limitaciones que les hemos introducido y de movimiento son idénticos a los que hemos creado anteriormente.

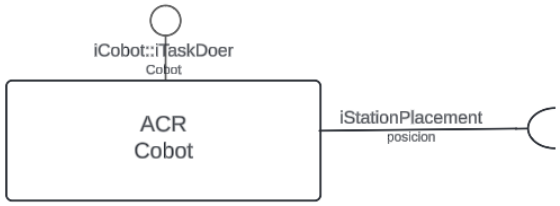

Figura 4.6: Ilustración del componente de robot colaborativo.

4.1.3.4. Estación

Este es el componente donde se llevará a cabo la tarea colaborativa o el proceso. Por ello, la estación no necesita de un iStationPlacement para trabajar.

Además, las distintas posiciones que existirán para llevar a cabo una tarea, los Placements, también se conectarán a la interfaz que la estación proporciona. Esto hace que solo se pueda llevar a cabo una tarea colaborativa si los respectivos Placements que la tarea necesita están conectados a la estación y si esos Placements tienen un TaskDoer que pueda realizar la tarea.

4.1.4. Placement

A continuación, hablaremos del componente Placement, el cual se encarga de indicar cuáles son las posiciones disponibles en la estación para que los TaskDoer lleven a cabo las tareas.

Cada vez que se quiera llevar a cabo una tarea colaborativa, se verificarán las posiciones que están conectadas a la estación y si son las necesarias para llevar a cabo la tarea, además de comprobar si existe un TaskDoer en esa posición conectado a la tarea correspondiente.

Pongamos un ejemplo para clarificarlo. En el caso de soldar piezas, existirán cuatro posiciones: PosicionSuministrar1, PosicionSuministrar2, PosicionSoldar y Posicion-

Figura 4.7: Ilustración del componente de estación de trabajo.

Retirar. Para que una regla de adaptación se pueda llevar a cabo, lo que deberá ocurrir es que, por ejemplo, el TaskDoer que vaya a llevar a cabo la soldadura esté en la PosicionSoldar y además esté conectado con el componente Task que realiza esa tarea. Esta forma de trabajar se ha creado de esta manera para que se pueda tener una noción de cuándo el TaskDoer está preparado y en posición para llevar a cabo su tarea.

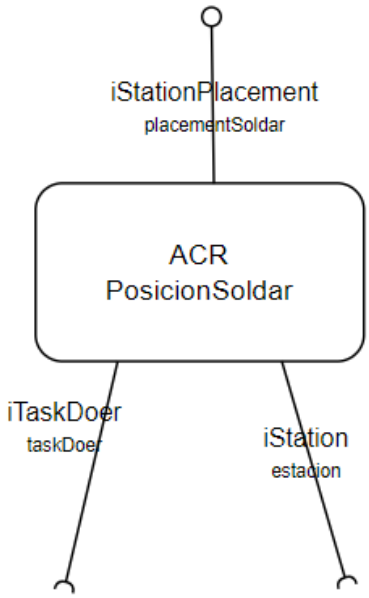

Figura 4.8: Ilustración del componente de las posiciones de la estación.

4.1.5. Representación visual

Para llegar a ver más claro cómo se pueden unir estos componentes, se pueden explicar como si fuesen piezas de construcción, por lo que podemos describirlo de esa manera, como se puede observar en el ejemplo de 4.9.

Este ejemplo plantea cómo, con este framework, se construye sobre la Station, que es la base de todos los demás componentes y a partir de la cual comienza. A esta

se le deben conectar los Placements que tendrá, sobre los cuales se colocarán los Task-doers que vayan a realizar las tareas, y después las Task que se van a llevar a cabo.

En la parte posterior, situándose sobre la Station y llegando hasta las Task superiores, se encuentra la Task colaborativa, que llega hasta arriba debido a que necesita de estas Task que ofrecen los Task-doers para funcionar.

Como se puede observar, algunas de estas piezas están apiladas, por lo que se puede ver que, en el caso de que desconectemos, por ejemplo, un Placement, también lo harán tanto el Task-doer como la Task que se iba a llevar a cabo, que es el planteamiento que también ofrece el sistema.

En el caso de que se quiera trabajar con otra pieza o componente, esta manera de trabajar facilita muchísimo el proceso, ya que se simplifica la forma de conectar de nuevo los componentes. Además, no hay que cambiar ninguna lógica; sino únicamente usar el nuevo componente y hacer que lo detecte el sistema, y ya el sistema MAPE-K ajustará la configuración para cambiar estas conexiones.

En 4.9 se ve un sistema que podemos imaginar que está preparado para llevar a cabo la tarea colaborativa, ya que sus relaciones están bien establecidas; por ejemplo, cada Placement tiene su Task-doer y la Task colaborativa tiene las tres Task necesarias para funcionar. En el caso de que alguna no estuviese, sería necesario conectar algún componente que sea válido para ese servicio necesario para funcionar.

Por ello, concluimos que trabajar de esta manera, encapsulando los objetos dentro de estos componentes ACR autónomos, ayuda a que la manera de trabajar reduzca la carga de la creación de la lógica de relación de los componentes, ya que todos los componentes están creados de la misma manera. De esta manera, se pueden facilitar acciones como hacer los cambios necesarios en estos, entre otras acciones.

4.2. MAPE-K

A continuación, se explicarán todos los componentes necesarios para nuestro bucle de control. Desde las sondas que reportarán los datos, hasta los monitores que tratarán la información, las reglas de adaptación que diseñemos y el knowledge donde se guardará esta información.

A través de este bucle de control es como se puede adaptar el sistema, haciendo que, a través de las sondas, monitores, reglas de adaptación y propiedades de adaptación, se intente encontrar una configuración que sea válida para llevar a cabo una Task o para detener una Task en el caso de que algo falle, como que un robot deje de estar en su Placement.

4.2.1. Sondas

Ahora enseñaremos las sondas que hemos creado para nuestra solución. Las sondas en la estructura MAPE-K pueden ser cualquier dispositivo que reporte valores a nuestro bucle de control. En nuestro caso, hemos considerado unas sondas simples para la solución de nuestro proyecto. Aunque en este caso las sondas están incrustadas en el sistema, hay que destacar que también podrían ser externas, siendo, por ejemplo, sensores:

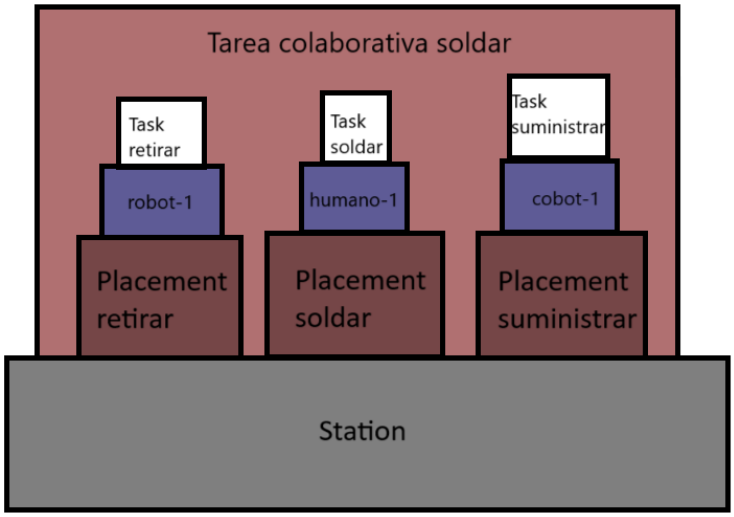

Figura 4.9: Ejemplo visual con bloques de como funcionaria un sistema usando el framework

1. Sonda: Station-N-RequiredTask-probe.
   Descripción:Esta sonda reporta el estado de una estación para llevar a cabo una tarea y es la que detecta qué tarea se quiere llevar a cabo en la misma.
   Monitor: Station-N-RequiredTask-monitor.
   Datos: Devuelve información de la tarea que se quiere llevar a cabo en la estación.

2. Sonda: Placement-probe.
   Descripción: Esta sonda es la que se ocupará de controlar los Placements de la Station, verificando si detecta que se ha colocado un Task-doer en alguno de los Placements para después reportarlo.
   Monitor: Task-AsignDoer-monitor.
   Datos: Reporta el Task-Doer que se ha posicionado al Placement.

4.2.2. Monitores

Los monitores son los componentes de nuestro bucle de control que recibirán los valores de las sondas y los filtrarán para ver y conseguir solo aquellos valores que puedan llegar a ser relevantes para llevar a cabo alguna regla de adaptación.

1. Monitor: Task-AsignStation-monitor
   Descripción: Este monitor realiza las correspondientes validaciones para saber si se puede llevar a cabo la asignación de una Task a una Station.
   Afecta Propiedades de Adaptación: station-*/task-assignation
   Acciones:
   SI (Task != null && Station.Exists)
   ACTUALIZA-KNOWLEDGE station-*/task-assignation = Station-* + Task

2. Monitor: Placement-monitor
   Descripción: Este monitor guarda en la regla de adaptación la última configuración que le llega de la sonda si el formato es correcto. En el knowledge, guardaremos únicamente la última configuración que se nos reporta, ya que las anteriores ya se han comprobado y no sería necesario volver a verificarlas, dado que los componentes ya han registrado que existe un Task-doer en esa posición.
   Afecta Propiedades de Adaptación: station-*/placement-configuration
   Acciones:
   SI (Task-Doer != null && Placement.Exists)
   ACTUALIZA-KNOWLEDGE station-*/placement-configuration = Station-* + Task-Doer + Placement

4.2.3. Reglas

En el caso de que hayamos considerado la tarea de soldar piezas, hemos creado dos reglas de adaptación que tienen dos configuraciones distintas para llevar a cabo la tarea: una en la que las tareas de soldar y retirar son realizadas por dos TaskDoer diferentes, y otra en la que ambas son consideradas como una única tarea y, por ende, realizadas por un único TaskDoer.

Cabe destacar que las configuraciones serán variables, ya que las tareas podrán ser realizadas por varios tipos de TaskDoer en muchos casos (por ejemplo, la tarea de soldar, que podrá ser llevada a cabo por cualquier tipo de TaskDoer). La comprobación de que el TaskDoer que se asigna para realizar la tarea se encuentre entre los que son válidos se llevará a cabo en la implementación.

4.2.3.1. SoldarPiezaColaborativa.Tango

Esta configuración para llevar a cabo la tarea de soldar piezas cuenta con 4 tareas: las dos de suministrar piezas, que se realizarán de forma asíncrona; la de soldar, que podrá ser síncrona si la realiza un robot, o Long Lived Task si la lleva a cabo una persona; y la de retirar, que será igual que la tarea de soldar. Dependiendo del TaskDoer que vaya a realizar la Task, deberemos seleccionar para hacer la unión una u otra tarea.

En 4.10 se muestra un ejemplo de una configuración posible. Como podemos ver, la tarea de retirar la va a realizar un robot industrial, el ABB_IRB_1600_6, por lo que el componente Task que debemos elegir para nuestra configuración deberá ser el que haga que ese TaskDoer ejecute la Task, en este caso, SoldarPiezasColaborativasRetirarPieza.Robot.

Deberemos hacer la unión entre todas las posiciones necesarias para llevar a cabo la tarea con la estación, y estas, a su vez, deberán unirse con los TaskDoer que se posicionarán en esas posiciones y llevarán a cabo las tareas.

Finalmente, se hará la conexión de cada una de las tareas con la tarea colaborativa, que podrá ser de distinto tipo dependiendo de quién la realice, pero siempre tiene que implementar la interfaz de la tarea que se necesite.

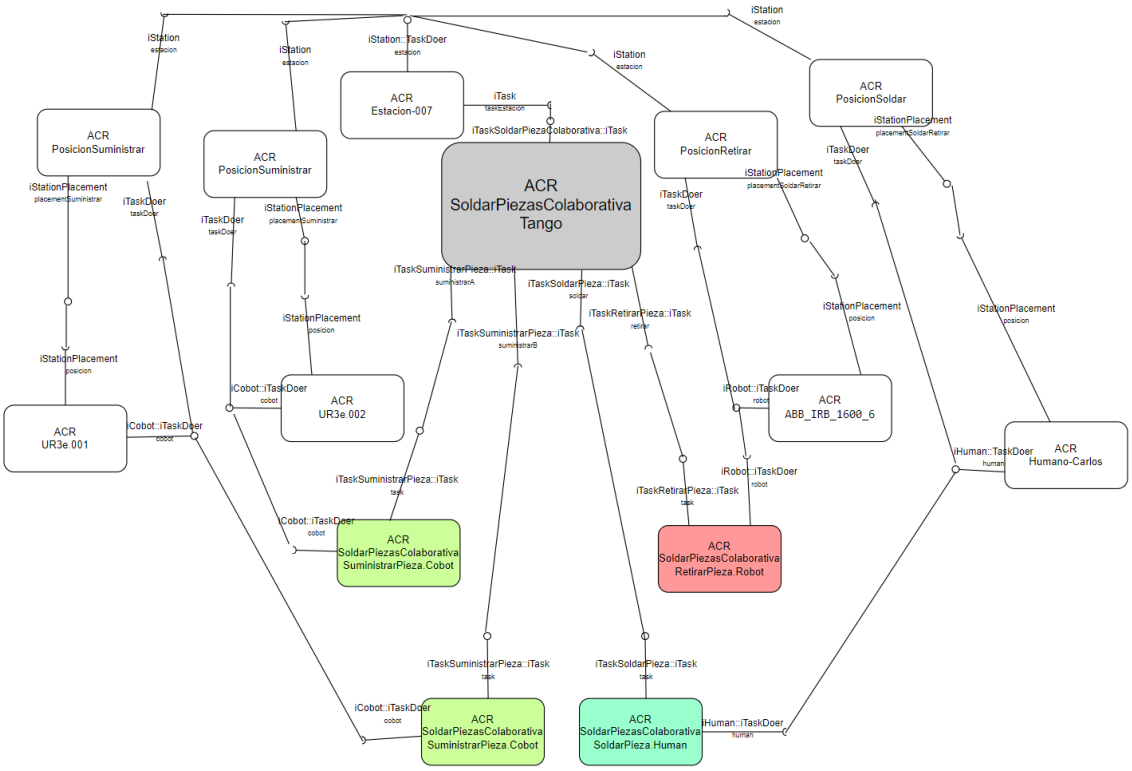

Figura 4.10: Ilustración de la composición de componentes de la tarea soldar piezas denominada 'Tango'

4.2.3.2. SoldarPiezaColaborativa.Delta

Esta configuración contará con 3 tareas en lugar de las 4 de la regla anterior. Las 2 de suministrar se mantendrán igual; sin embargo, las tareas de retirar y soldar se juntarán y se realizarán en una única tarea, que, dependiendo de si la lleva a cabo un humano o un robot, será síncrona en el caso del robot o Long Lived en el caso de la persona.

En 4.11 se ha presentado una configuración posible en la que la tarea es síncrona y es realizada por un robot industrial.

Se necesitarán hacer las mismas conexiones que en 4.10, a excepción de que habrá 3 posiciones en lugar de 4 y que cambiaremos las Tasks de retirar y soldar por una sola que hará ambas tareas. El resto sigue la misma lógica.

4.2.4. Knowledge

Por último, hablaremos del knowledge, que se resumiría en el espacio donde el bucle de control puede guardar información que será revisada por monitores o reglas para valorar si es necesaria una adaptación o no.

La forma en la que se ha trabajado considera la menor cantidad de propiedades de knowledge para facilitar la comprensión del código, además de que no serían totalmente necesarias todas las propiedades, ya que cada una se comprueba en su momento. Sin embargo, en el caso de que se hubiesen puesto más propiedades (un ejemplo sería poner una propiedad para cada placement), también sería válido.

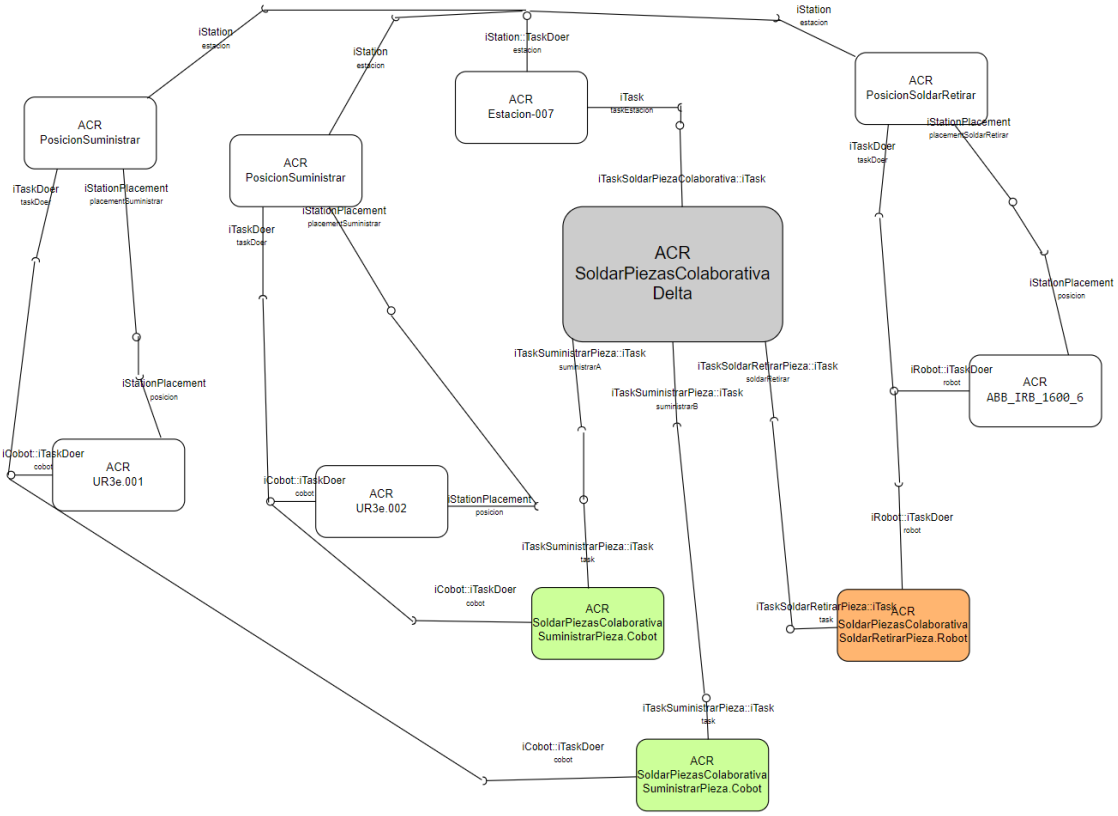

Figura 4.11: Ilustración de la composición de componentes de la tarea soldar piezas denominada 'Delta'

1. Propiedad: station-007/task-assignation.
   Descripción: Esta es la propiedad donde se guardará la última Task colaborativa que se haya reportado a la Station.
   Tipo de Dato: Tarea colaborativa (String).

2. Propiedad: station-007/placement-configuration.
   Descripción: En esta propiedad se guardará la última configuración de los Placements que se reporta. Si, por ejemplo, en la sonda se informa que en el Placement "Suministrar"se ha posicionado un robot, eso será lo que se guardará en esta propiedad. Sin embargo, si en el siguiente dato reportado por la sonda se indica que el Placement Retirar"ha sido ocupado por un robot, se sobrescribirá el valor de la propiedad, ya que se habrán realizado las comprobaciones necesarias con los datos anteriores. De esta manera, evitamos la necesidad de colocar una propiedad de knowledge para cada Placement que hayamos creado.
   Tipo de Dato: Task (String).

Capítulo 5

Desarrollo

En este capítulo, se explicará cómo se ha pasado de las ideas de los componentes, tanto ACR como del bucle MAPE-K, al proyecto, en temas de implementación, además de explicar el porqué de cada forma de implementación. También se explicará cómo se ha trabajado con RoboDK.

5.1. Bucle de control MAPE-K

5.1.1. Creación de los componentes ACR y del bucle de control MAPE-K

En un primer momento, comentar que el IDE que usaremos para nuestro proyecto será Eclipse [13], en particular su versión para RCP y RAP. Lo usaremos debido a las facilidades que nos aporta con sus Launch Configurations para lanzar los paquetes necesarios para nuestro proyecto, permitiendo crear y gestionar el bucle de control mediante el framework OSGi [14].

Lo siguiente que hacemos es crear nuestra Launch Configuration. En él será donde incluyamos los paquetes que debe tener el proyecto, incluyendo los paquetes del grupo Tatami, que facilita todo el proceso de trabajar con el bucle de control MAPE-K.

Dentro del código, se deberán crear cada uno de los componentes que pueden ser necesarios para alguna configuración o para alguna de las posibilidades de ejecución. Estos componentes se podrían crear de manera dinámica, pero para facilitar las cosas y debido a que este es un caso ejemplificado, los crearemos desde el principio.

En este caso, lo que se ha creado para el funcionamiento han sido los Task Doer, los Placements, la Station y las distintas posibilidades de Task (que la misma tarea la haga un robot o una persona ya supone que debemos crear esa Task para el robot y para la persona, ya que las ejecuciones serán distintas).

Seguidamente, lo que se hará será crear las propiedades adaptativas, monitores, sondas y reglas, y posteriormente inicializarlas en lo que llamaremos el activador del bucle de control MAPE-K. Aquí es donde añadiremos los componentes al bucle para que funcionen como el bucle lo necesita.

El sistema es adaptativo, por lo que cada vez que cambien los inputs, se reconfigurará en tiempo real para intentar encontrar una configuración válida que permita llevar a

cabo la Task colaborativa.

Las reglas de adaptación se crean para establecer las configuraciones y unir a los componentes entre sí cuando se satisfaga la condición establecida. En este caso, de la manera en que se creó el knowledge, que reporta si existe un Task Doer en un Placement, cada vez que cambia esta propiedad de adaptación, se guarda que ese Placement ya está ocupado y es válido para llevar a cabo esa configuración.

Si en algún momento un Task Doer abandona uno de los Placements, la configuración pasaría a ser no válida y, por ende, pasaría a no tener una configuración. Si está en mitad de llevar a cabo una Task, la abortaría.

Ejemplificándolo de alguna manera, si nuestra regla de adaptación tuviese un Placement que es "suministrar", otro que es "soldar" y un último que es "retirar", la primera vez se reportaría que en el Placement "suministra" ya existe un Task Doer para llevar a cabo la tarea. Si el Task Doer es válido, se marcará el Placement como válido. Así con cada uno de los Placements, y finalmente, cuando todos los Placements son válidos 5.2, se harán los bindings de los componentes ARC, que recordaremos que es la forma que tenemos de unir los Task Doers, Tasks, etc. (En este caso, se unirán los Task Doers a los Placements y a las Tasks correspondientes) 5.3. La Station ya estaría preparada para llevar a cabo la tarea cuya configuración es válida.

```
(Station N RequiredTask Probe) Reporting measure: [Station Task Configuration] Station: station-007, Task: soldar-piezas
Received measure: [Station Task Configuration] Station: station-007, Task: soldar-piezas
```

Figura 5.1: Aquí se recibiría que la Station-007 va a hacer la Task colaborativa soldar

```
Checking Tango Configuration ...
Placement <suministrar1> satisfies doer's requirement ...
Placement <suministrar2> satisfies doer's requirement ...
Placement <soldar> satisfies doer's requirement ...
Placement <retirar> satisfies doer's requirement ...
Tango Configuration is VALID ...
```

Figura 5.2: Aquí vemos que ha encontrado una configuración valida

```
DEPLOY ACR.SoldarPiezasColaborativa.Soldar.Humano.station-007_1.0.0
Performing the DEPLOY of ACR.SoldarPiezasColaborativa.Soldar.Humano.station-007_1.0.0
[ACR.SoldarPiezasColaborativa.Soldar.Humano.station-007] DEPLOYING ...
DEPLOY ACR.SoldarPiezasColaborativa.Quitar.Robot.station-007_1.0.0
Performing the DEPLOY of ACR.SoldarPiezasColaborativa.Quitar.Robot.station-007_1.0.0
[ACR.SoldarPiezasColaborativa.Quitar.Robot.station-007] DEPLOYING ...
BIND station-007_1.0.0::req_StationTask -C   o- ACR.SoldarPiezasColaborativa.Tango.station-007_1.0.0::prov_service
Performing the BIND of station-007::req_StationTask -C --- o- ACR.SoldarPiezasColaborativa.Tango.station-007::prov_service
[station-007] BINDING SERVICE ...
```

Figura 5.3: Aquí encontramos alguno de los deploy y bindings de los componentes ACR

5.1.2. Api MQTT

La forma en la que se llevará a cabo la comunicación entre nuestro bucle de control y la aplicación de RoboDK será mediante MQTT. Para ello, se ha usado el broker Mosquitto[15] para ayudar a implementar este protocolo.

Para ayudar con la creación de los temas que se usarán en el proyecto, se ha usado MQTTfx[16], la cual es una herramienta que facilita el trabajo de enviar mensajes mediante este protocolo. Además, se ha utilizado para el testeo y para simular las publicaciones que mandarían los robots o personas cuando están en posición para realizar una tarea, o cuando se quiere seleccionar una Station para que haga una Task colaborativa, por ejemplo.

Dicho esto, los temas tendrán una estructura similar a esta: "station/station-007/pintar-piezas/suministrar/config", donde cambiaremos los parámetros como la Station, la Task colaborativa o la Task que necesitemos en cada caso (la única excepción serán los temas para preparar una Station para llevar a cabo una Task, que seguirán la estructura: station/station-007/config). Los últimos parámetros de los temas a los que se enviarán las publicaciones y a los que se suscribirán estarán diferenciados por 3 tipos:

1. Config: Estos temas serán usados para la configuración de los Placements. En ellos, se enviarán los datos del Task Doer que va a llevar a cabo la tarea. Esto simularía cuando se posiciona al Task Doer en un Placement para llevar a cabo la Task. Este mensaje lo publicaremos a través de MQTTfx 5.4 y lo recibirán tanto el bucle de control a través de las sondas, que comprobará las configuraciones para ver si alguna ya es válida y está lista para ser ejecutada, como RoboDK, que posicionará un Task Doer en la posición que se ha mandado.

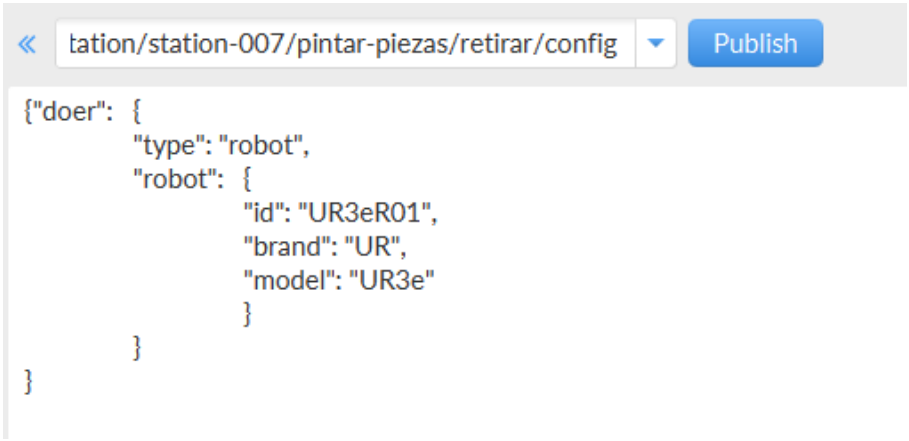

Figura 5.4: Ilustración de como se vería un mensaje de ejemplo de config enviado a través de MQTTfx

2. Action: Estos temas serán usados por el bucle de control para enviar a RoboDK cuándo iniciar una tarea de un Task Doer. El código de estos mensajes está situado dentro de las Tasks, para que sean enviados únicamente cuando las comprobaciones son correctas y se está llevando a cabo la ejecución correctamente.5.5
3. Status: Estos son enviados desde RoboDK para comunicarle al bucle de control el estado de las Tasks (si en un pick & drop se está ejecutando el pick o si ya ha acabado totalmente la Task) para que el bucle considere qué hacer. La estructura sería igual a la de los temas Action, sustituyendo Action por Status.

En la herramienta de MQTTfx se crearán y se guardarán las publicaciones que se

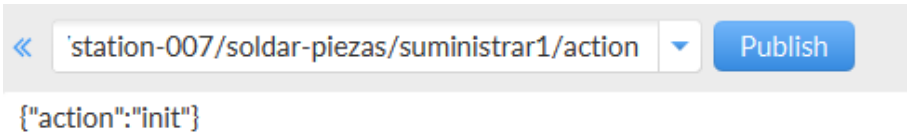

Figura 5.5: Ilustración de como se vería un mensaje de ejemplo de action enviado a través de MQTTfx

usarán para colocar los Task Doers en los Placements, además de asignar a la Station la Task colaborativa 5.6. Además, durante todo el proceso ha sido usada para labores de testeo de funcionamiento.

Figura 5.6: Ejemplos de publicaciones guardados en MQTTfx

5.1.3. Acción de las Task

A continuación, se debe crear la lógica dentro de las Tasks. En este caso, como lo que se quiere es demostrar a través de RoboDK, como ejemplo visual, cómo funcionaría este proyecto, lo que se debe hacer es que cada Task envíe un evento a un tema al que estará suscrito el proyecto de RoboDK, con el que empezará a simular los movimientos.

Entonces, con esto, se han configurado las Tasks de la siguiente manera:

1. Task colaborativa: Estas Tasks son las que se ejecutan cuando una Station está preparada con una configuración válida para llevarla a cabo. Su lógica interna contiene el orden en el que se tienen que ejecutar las Tasks, ya que no siempre deben ser una después de otra. Un ejemplo sería que en un pick & place, antes de hacer el place, se deba hacer otra Task, como pintar o soldar. Este tipo de cosas se programarán dentro de esta Task.

2. Task: Dentro de estas Tasks encontramos la lógica con los mensajes para que en RoboDK se lleven a cabo los movimientos de los Task Doers para realizar cada acción, junto a los mensajes de error o de confirmación, entre otras cosas.

5.2. RoboDK

A continuación, en esta sección se mostrara como se ha trabajado con RoboDK.

5.2.1. Creación de los robots

Como se ha trabajado, ha sido creando una plataforma alrededor de la cual se iban a posicionar los robots, los cuales se han obtenido de la página oficial de RoboDK [17], ya que esta cuenta con una extensa librería donde se pueden encontrar diversos tipos de robots, además de agarres u objetos, entre muchas otras cosas que podemos usar en nuestros proyectos.

Entonces, el primer paso fue crear dichos robots, de los cuales se consideraron los que podrían posicionarse en cada Placement. Un ejemplo es que, aunque en el Placement ”Retirar"se pudiera colocar cualquier robot, ya fuera colaborativo o no, para simplificar, consideraremos que solo se pueden colocar UR3 y UR5. De esta manera, simplificamos el trabajo, ya que solo se tendrían que ajustar estos casos. Se hace de esta manera porque el objetivo del trabajo no es ofrecer una solución completa que sirva para cualquier robot, sino una ejemplificación visual de que puede funcionar.

Por ello, lo que se hará será crear 2 Task-doers por posición, y se colocarán invisibles. De esta manera, lo que se hará cuando llegue la publicación de que se ha posicionado un robot en un Placement será descubrir al robot, simulando que se ha instalado ahí.

Cabe destacar un tema, y es sobre los nombres de los Task-doers, en concreto de los robots y cobots, ya que estos deben ser los mismos que los inicializados en el bucle de control MAPE-K. Esto es importante porque, posteriormente, se verá que cuando se envía desde nuestra herramienta MQTTfx una publicación al tema ”config", lo que hacemos es configurar que el Task-doer está en el Placement. Esa publicación la recibe tanto el bucle como RoboDK, por lo que deben tratar los mismos nombres para que ambos funcionen con las mismas publicaciones.

5.2.2. Control del movimiento

Ahora lo que se tendría que hacer con cada uno de los Placements es elegir uno de los dos Task-doers e ir marcando las posiciones que serán necesarias para trabajar con ellos.

De manera general, se encuentran tres posiciones que serían generales y mínimas, pero que podrían aumentar dependiendo de la Task que lleven a cabo, como es el caso de este proyecto. Estas posiciones mínimas serían la "Safety". 5.7, "Pre*Accion*"5.8, "*Accion*"5.9.

Como se ha mencionado, ese es el mínimo de posiciones que tendrá un robot que vaya a llevar a cabo una tarea, pero en caso de que este tenga una tarea que se realice en varias posiciones, o que necesite de varias posiciones para trabajar, se deberán también crear siguiendo el mismo procedimiento.

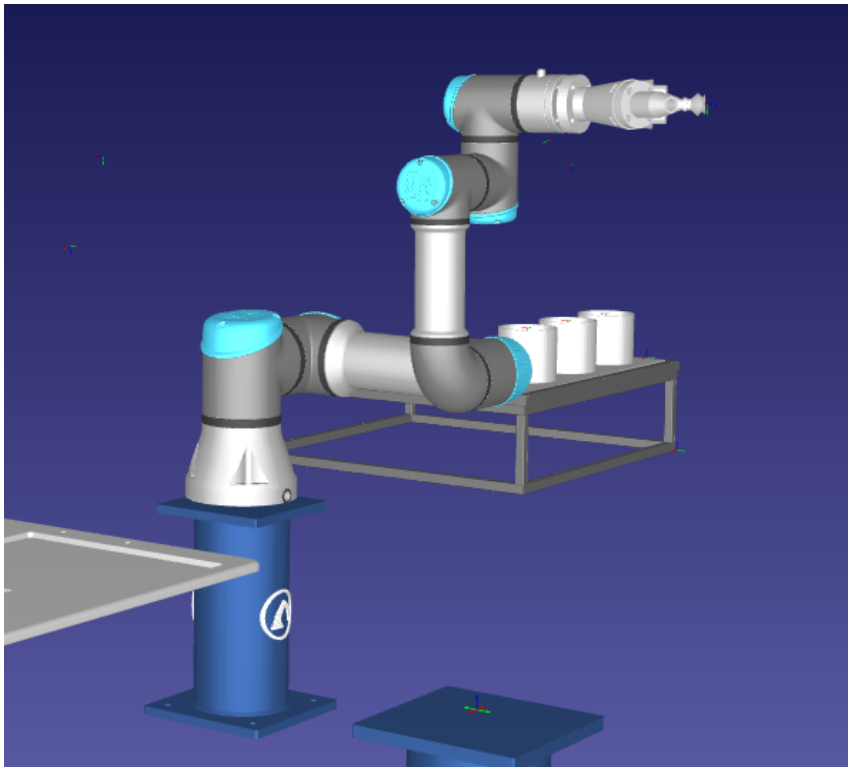

Figura 5.7: Robot en su posición 'safety'

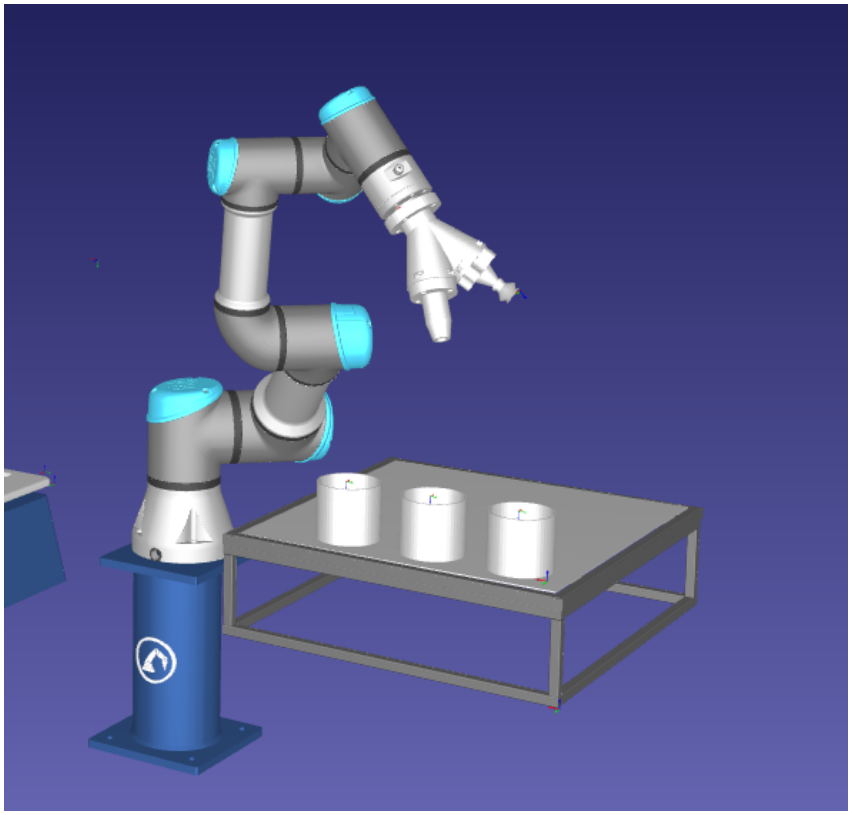

Figura 5.8: Robot en su posición anterior a la posición donde hará la acción

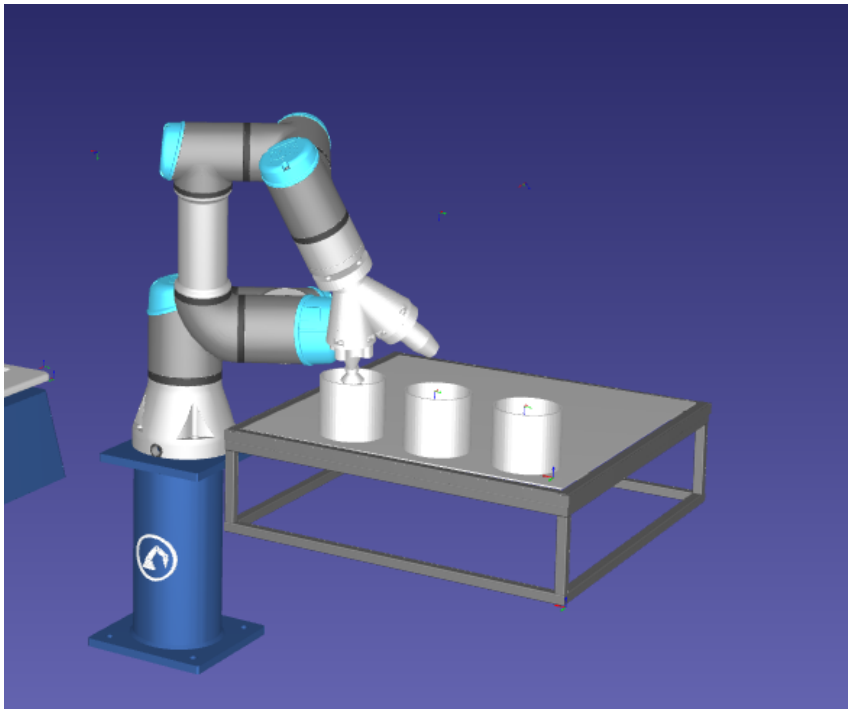

Figura 5.9: Robot en la posición donde llevara a cabo la acción

Para saber cuáles serían los Placements sin que los robots estén posicionados, se han creado unas plataformas donde se podrá ver dónde se situarán los Task-doers.

De la manera en que funciona el proyecto, en el que para llevar a cabo una tarea debe haber una configuración válida, no es necesario que se cubran todos los Placements con un Task-doer para funcionar; únicamente se deben situar en aquellos que sean necesarios para la Task.

Una vez situados todos los robots y guardadas sus posiciones, pasaremos a crear la lógica de movimiento de estos, además de la lógica de envío de publicaciones al broker Mosquitto.

Para esto, se han creado una serie de scripts en Python. En uno de ellos, se han almacenado las coordenadas de las posiciones necesarias para llevar a cabo las tareas (ya que, independientemente del robot, si van a realizar la misma tarea, se considerará para simplificar que las posiciones con las que trabajarán serán las mismas).

Después, se crea el script donde haremos la conexión al broker Mosquitto y donde se suscribirá el programa a los temas. A la vez, se desarrollará el script que tratará las publicaciones que lleguen (si es un config, volver a hacer visible al robot que llegó por la publicación, o si es un action, llevar a cabo el movimiento de la Task).

En el ultimo script sera donde creemos la lógica de los movimientos. A través de los movimientos en los ejes que nos proporciona RoboDK, haremos que los robots se muevan entre las posiciones que se usen.

Con todo esto, solo seria necesario lanzar el script donde tengamos el código de suscripción al broker MQTT, y a partir de el, cuando el programa reciba las publicaciones, empezara la Task.

5.3. Responsabilidades

En esta sección, es importante destacar las responsabilidades de cada uno de los componentes y cómo se organizan dentro del sistema. El trabajo de los componentes ACR y del bucle MAPE-K no es manejar la lógica detallada de las acciones específicas que realiza el Task-Doer (como soldar o retirar una pieza), sino que su papel se centra en orquestar las acciones que se van a llevar a cabo. El ”cómo"los robots ejecutan esas acciones se encuentra en los propios robots. En este caso de simulación, esa lógica estará en los scripts dentro del software RoboDK.

En otras palabras, cuando hablamos de tareas colaborativas, como soldar piezas o cualquier otra tarea similar en el framework ACR, lo que estamos representando no es el control detallado de cómo el robot realiza la acción de soldadura, sino que estamos invocando esa tarea en el robot. La implementación real de la soldadura se encuentra dentro del robot, y lo que estamos haciendo en el framework ACR es simplemente lanzar el evento o instrucción que activa esa tarea en el robot.

Sin embargo, las APIs que ofrecen hoy en día los fabricantes de robots son muy limitadas en cuanto a qué tan inteligentes pueden llegar a ser, lo que representa una gran limitación a la hora de introducir lógica en ellos, ya que están diseñados únicamente para ejecutar órdenes básicas, como movimientos lineales o axiales.

Debido a que los robots no cuentan con la capacidad de tener una lógica que les permita coordinarse, es donde entra el bucle MAPE-K, que se encarga de planificar, coordinar y decidir cuándo se deben ejecutar las tareas del robot. Mientras que el robot solo ejecuta la tarea concreta (por ejemplo, "soldar"), el bucle MAPE-K se asegura de que las tareas se realicen en el orden correcto y de forma adecuada.

En el futuro, si los robots evolucionan hacia sistemas más inteligentes, podrían manejar tareas completas internamente, en lugar de depender de un sistema, como el bucle MAPE-K, para decirles qué hacer. Este avance permitiría que el sistema externo solo tuviera que supervisar que todo esté en orden, y el robot se encargaría de hacer su trabajo de manera autónoma.

Capítulo 6

Resultados

En este capítulo, se mostrará un ejemplo de ejecución con un caso de uso propuesto; en este caso, será la simulación de la soldadura de piezas.

Para ello, iniciaremos las tres piezas angulares del proyecto 6.1, que serán tanto Eclipse, con el proyecto que contiene el bucle de control MAPE-K, como el proyecto de RoboDK, en el que están los robots y scripts creados, e inicializaremos el broker de Mosquitto. Para las señales que simularían las posiciones de los Task-doer, como se ha mencionado previamente, se usará MQTTfx.

Figura 6.1: Herramientas que se usaran

Para lanzar el proyecto, lo que deberemos hacer será conectar MQTTfx al broker que se ha creado en local 6.2 y ejecutar tanto el script en RoboDK como la launch configuration en el proyecto de Eclipse, donde tendremos los paquetes necesarios para trabajar 6.3. En esta launch configuration se podrían configurar parámetros en caso de que se quisiera que ya hubiese alguna configuración cargada, pero en esta demostración desde cero, iremos cargando los robots en sus Placements correspondientes paso a paso.

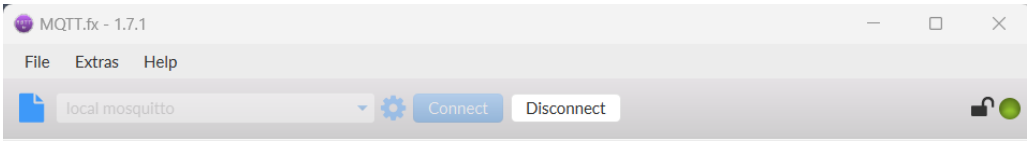

Figura 6.2: MQTTfx conectado al broker local

Cuando se hagan estos pasos, en Eclipse se verá cómo se han desplegado todos los componentes necesarios que creamos desde un inicio, como las sondas, monitores 6.4 o Task-doers y Placements 6.5. Al mismo tiempo, RoboDK quedará escuchando, preparado para ejecutar las acciones cuando reciba la publicación correspondiente.

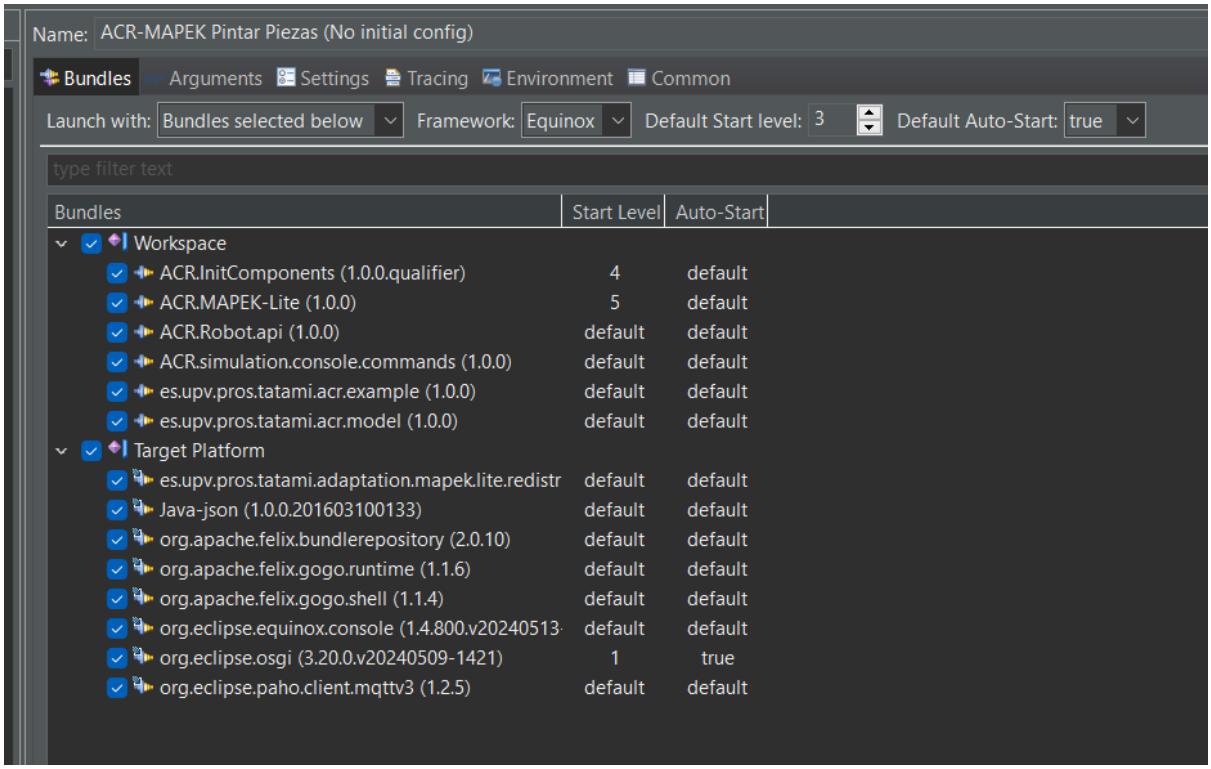

Figura 6.3: Launch configuration en Eclipse

```
[TRACE] ...components.arc.AdaptationRuleARC: [Station N Tarea Soldar Piezas Adaptation Rule (v2)] BINDING SERVICE ...
[TRACE] ...components.arc.AdaptationRuleARC: [Station N Tarea Soldar Piezas Adaptation Rule (v2)] DEPLOYING ...
[TRACE] ...les.impl.arc.MonitoringModuleARC: [MonitoringModule] GET SERVICE SUPPLY ...
[TRACE] ...ifacts.components.arc.MonitorARC: [Station N Task Assignation Event Monitor] BINDING SERVICE ...
[TRACE] ...ifacts.components.arc.MonitorARC: [Station N Task Assignation Event Monitor] DEPLOYING ...
[TRACE] ...les.impl.arc.MonitoringModuleARC: [MonitoringModule] GET SERVICE SUPPLY ...
[TRACE] ...ifacts.components.arc.MonitorARC: [Station N Placement Configuration Monitor] BINDING SERVICE ...
[TRACE] ...ifacts.components.arc.MonitorARC: [Station N Placement Configuration Monitor] DEPLOYING ...
```

Figura 6.4: Se inicializan correctamente los componentes del bucle de control

En este ejemplo, usaremos los Placements Suministrar 1, Suministrar 2, Soldar y Retirar en este primer caso. Por ello, a través de MQTTfx, se comenzarán a enviar las publicaciones para dichos Placements. Se colocará un robot UR3 en cada uno de los Suministrar y un UR5 en el Placement Retirar, mientras que en el Placement de Soldar simularemos que hay una persona.

Lo primero será enviar a través de MQTTfx que la Task que se va a llevar a cabo es la de soldar piezas, para que el bucle de control guarde en la Station que es la Task que se está pensando en llevar a cabo.

Posteriormente, enviaremos los Task-doers a los Placements, lo que hará que vayan apareciendo los robots en RoboDK 6.6 6.7. Cuando estén todos los Placements necesarios, en Eclipse, donde tenemos el bucle de control, se seleccionará la configuración válida y se podrá ejecutar.

Con toda la configuración preparada, ya se puede poner en marcha la Task colaborativa. Se enviará para iniciar una publicación mediante MQTTfx que recibirá el proyecto de Eclipse para empezar a ejecutar la Task colaborativa de soldadura. Es-

```
[TRACE] ...mponents.arc.StationPlacementARC: [suministrar1] DEPLOYING ...
[TRACE] ...mponents.arc.StationPlacementARC: [suministrar2] DEPLOYING ...
[TRACE] ...mponents.arc.StationPlacementARC: [quitar] DEPLOYING ...
[TRACE] ...mponents.arc.StationPlacementARC: [soldar] DEPLOYING ...
[TRACE] ...mponents.arc.StationPlacementARC: [solret] DEPLOYING ...
[TRACE] ...r.robots.components.arc.RobotARC: [UR3eS01] DEPLOYING ...
[TRACE] ...r.robots.components.arc.RobotARC: [UR3eS02] DEPLOYING ...
```

Figura 6.5: Se inicializan correctamente los componentes ARC

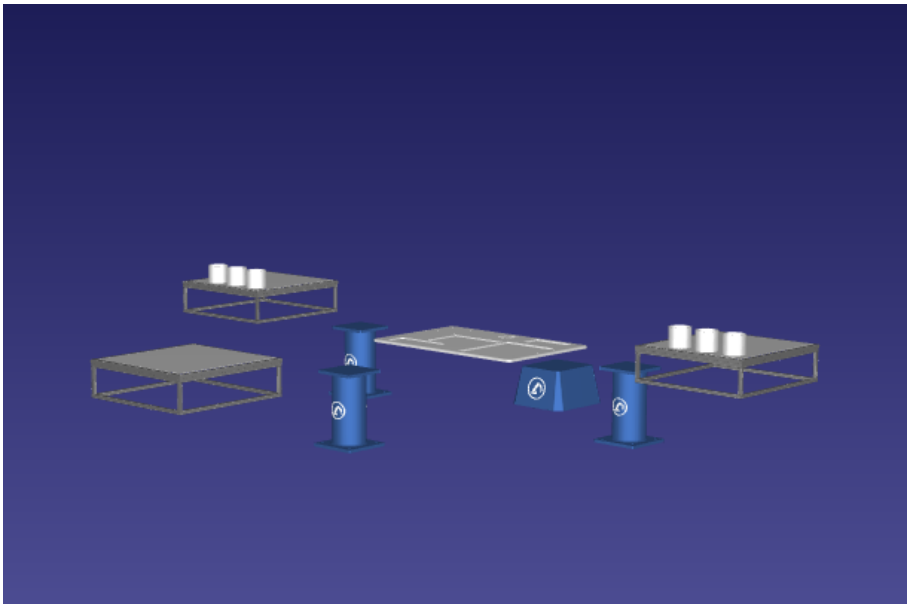

Figura 6.6: Estación sin los robots cargados

ta ejecutará la primera Task que se lleve a cabo, que en este caso será la Task de suministrar 1 y suministrar 2, cada una llevada a cabo por un robot.

Una vez que los robots han realizado la acción de pick & drop de su respectiva pieza, se llevará a cabo la Task de soldar, que en el caso de este proyecto se simula que lo hace una persona. Para esto, los robots se quedan en un estado inmóvil, esperando el input recibido a través de una publicación donde reportamos al bucle MAPE-K que ya se ha completado la acción (el humano deberá enviar la notificación en un tiempo límite).

Una vez que el bucle recibe la confirmación de que se ha hecho de manera satisfactoria, el tercer robot realizará el movimiento correspondiente para retirar la pieza a la zona final. Una vez que termine el movimiento, comenzará de nuevo con la cadena.

Aquí, si se da el caso de que, por ejemplo, el robot que está llevando a cabo la Task de retirar deje de estar disponible, la Task quedaría suspendida y la configuración sería inválida, por lo que el bucle de control se quedaría a expensas de que haya un Task-doer disponible para llevar a cabo la Task y poder completarla.

En el caso de que la configuración no sea válida, se podría hacer otra configuración, como podría ser el caso de que el robot que está en el Placement de retirar se sitúe en el Placement de soldar y retirar (que en RoboDK sería la misma plataforma que

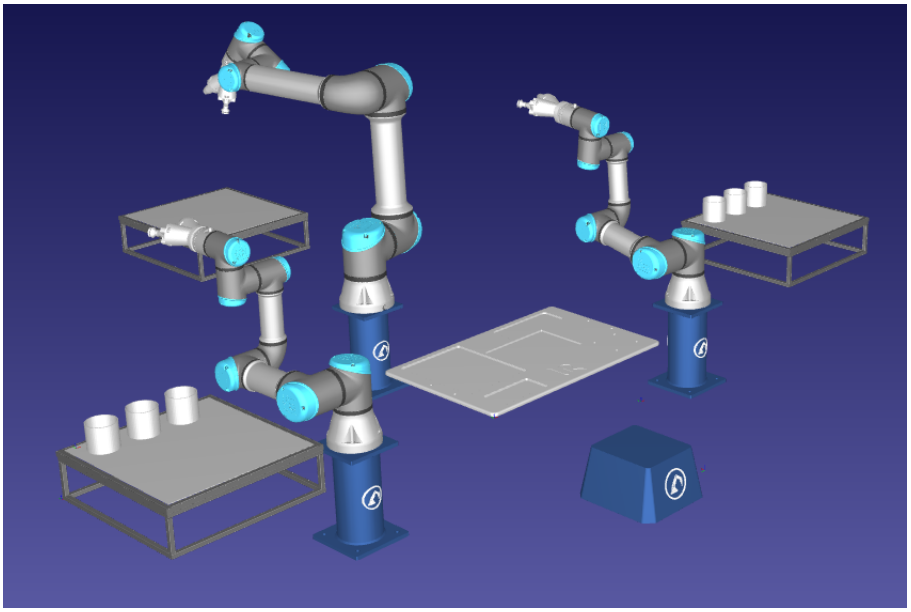

Figura 6.7: Estación con los robots cargados

retirar, pero realizaría ambas Tasks). En este caso, el bucle vería que hay una configuración distinta, pero válida, ya que cubre todas las posiciones necesarias para su funcionamiento, y la establecería como la configuración con la que ejecutar la Task colaborativa.

A pesar de que se pueda llevar a cabo el cambio de configuración, se sigue realizando la misma Task colaborativa, la de soldar, y ese es el gran logro de este proyecto: conseguir que se pueda adaptar para hacer una tarea con alguna de las múltiples configuraciones que se puedan aportar.

Capítulo 7

Conclusiones

En este TFM, se ha explorado el uso de soluciones de computación autónoma en el contexto de la robótica colaborativa, enfatizando cómo la tecnología puede contribuir a mejorar la flexibilidad y adaptabilidad de los sistemas industriales. A través del marco ACR (Autonomous Collaborative Robots) y la arquitectura MAPE-K, hemos propuesto y desarrollado una solución que facilita la integración de tareas colaborativas entre robots y humanos en entornos industriales.

Uno de los mayores logros de este trabajo ha sido demostrar que es posible construir un sistema robotizado colaborativo con capacidad de adaptación en tiempo de ejecución. Esto se ha logrado gracias al bucle de control MAPE-K, que ha permitido la monitorización, análisis, planificación y ejecución de las tareas en función de las condiciones actuales del entorno. Esta flexibilidad es fundamental en el contexto de la Industria 4.0 y 5.0, donde las líneas de producción no pueden depender de configuraciones estáticas y deben ser capaces de adaptarse rápidamente a cambios en la demanda o en los requisitos operativos.

El uso de RoboDK y el protocolo MQTT para la simulación de robots ha permitido visualizar cómo el sistema propuesto puede gestionar tareas colaborativas en tiempo real, incluyendo la asignación dinámica de robots a diferentes posiciones en una estación de trabajo. Este enfoque no solo simplifica el proceso de programación y ajuste de los robots, sino que también reduce la cantidad de código necesario, optimizando así el mantenimiento del sistema y minimizando los errores.

A pesar de los resultados positivos, el trabajo también ha revelado una serie de limitaciones y desafíos que deben ser abordados en investigaciones futuras. La seguridad sigue siendo uno de los principales obstáculos en la colaboración entre robots y humanos. Si bien los cobots han mejorado en este aspecto, es necesario seguir investigando y desarrollando mecanismos que permitan una interacción aún más segura y eficiente. Además, la integración de sistemas más complejos en entornos industriales reales conlleva desafíos técnicos adicionales, como la interoperabilidad entre diferentes tipos de robots y plataformas de software.

En cuanto a trabajos futuros, sería interesante profundizar en la extensión de la funcionalidad del sistema hacia una mayor autonomía en la toma de decisiones. Esto implicaría el uso de algoritmos de inteligencia artificial para optimizar las asignaciones de tareas en tiempo real y mejorar aún más la eficiencia del sistema. También sería valioso explorar el impacto de estas soluciones en escenarios industriales reales,

evaluando no solo su viabilidad técnica, sino también su rentabilidad económica y su impacto en la productividad. De esta forma, las reglas de adaptación seleccionadas podrían seguir una lógica orientada a ser las más rentables en condiciones específicas, ya sean económicas, de seguridad o de tiempo, entre otras.

En conclusión, este trabajo representa un paso adelante hacia la creación de sistemas de robótica colaborativa más adaptables y eficientes. La combinación de la arquitectura MAPE-K con tecnologías de simulación y comunicación ha demostrado ser una solución viable para mejorar la flexibilidad en los entornos de producción. Con los avances futuros en inteligencia artificial y seguridad, es probable que veamos una adopción aún mayor de estas tecnologías en la Industria 5.0, permitiendo una colaboración fluida y segura entre humanos y máquinas en una amplia variedad de tareas productivas.

Bibliografía

[1] J. Lee, G. Park and S. Ahn, “A Performance Evaluation of the Collaborative Robot System” 2021 21st International Conference on Control, Automation and Systems (ICCAS), Jeju, Korea, Republic of, 2021, pp. 1643-1648, doi: 10.23919/ICCAS52745.2021.9649859.

[2] admin. (2020, April 15). Historia de los Robots Colaborativos |Ripipsa| México. Ripipsa Cobots. https://ripipsacobots.com/historia-robots-colaborativos

[3] A. R. Riaz, A. Rauf, S. M. M. Gilani and M. B. Bashir, "Techniques and Methodologies of Self-Adaptive Software and its Architecture: A Survey,"2021 4th International Conference on Computing & Information Sciences (ICCIS), Karachi, Pakistan, 2021, pp. 1-5, doi: 10.1109/ICCIS54243.2021.9676190.

[4] ABB Group. Leading digital technologies for industry — ABB Group. (s. f.). ABB Group. https://global.abb/group/en

[5] Process Simulate software | Siemens Software. (s. f.). Siemens Digital Industries Software. https://plm.sw.siemens.com/en-US/tecnomatix/products/process-simulate-software/

[6] Robótica colaborativa en la industria del automóvil. (s. f.). Interempresas. https://www.interempresas.net/Sector-Automocion/Articulos/157970-Robotica-colaborativa-en-la-industria-del-automovil.html

[7] R. R, R. R. Sathya, S. V, J. L. N and G. S, “Enhancing Human Cobot Interaction with Mixed Reality: A Futuristic Review,” 2023 2nd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA), Coimbatore, India, 2023, pp. 1-6, doi: 10.1109/ICAECA56562.2023.10199911.

[8] A. Y. Zalozhnev and V. N. Ginz, “Industry 4.0: Underlying Technologies. Industry 5.0: Human-Computer Interaction as a Tech Bridge from Industry 4.0 to Industry 5.0,” 2023 9th International Conference on Web Research (ICWR), Tehran, Iran, Islamic Republic of, 2023, pp. 232-236, doi: 10.1109/ICWR57742.2023.10139166.

[9] (2005). An Architectural Blueprint for Autonomic Computing (). IBM .

[10] J. Fons, “Especificación de sistemas auto-adaptativos,” Mar. 2021.

[11] R. A. Atmoko and D. Yang, .ºnline Monitoring & Controlling Industrial Arm Robot Using MQTT Protocol,"2018 IEEE International Conference on Robotics, Biomimetics, and Intelligent Computational Systems (Robionetics), Bandung, Indo-

nesia, 2018, pp. 12-16, doi: 10.1109/ROBIONETICS.2018.8674672. keywords: Service robots;Monitoring;Protocols;Manipulators;Cloud computing;Internet of Things;Industrial ARM Robot;IIoT;MQTT,

[12] Papyrus. (s.f.). https://eclipse.dev/papyrus/

[13] "The Community for Open Collaboration and Innovation". Eclipse Foundation. https://www.eclipse.org/

[14] OSGi Working Group. https://www.osgi.org/

[15] Eclipse Mosquitto https://mosquitto.org/

[16] "Softblade GmbH - Home of MQTT.Fx®". Softblade.de https://www.softblade.de/

[17] Simulator for Industrial Robots and Offline Programming - RoboDK https://robodk.com/