FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Implementing a Multi-Approach Debugging of Industrial IoT Workflows

Andreia Cristina de Almeida Rodrigues

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Hugo Sereno Ferreira

Second Supervisor: João Pedro Dias

External Supervisor: José Pedro Silva

July 3, 2019

Implementing a Multi-Approach Debugging of Industrial IoT Workflows

Andreia Cristina de Almeida Rodrigues

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Filipe Figueiredo Correia, PhD

External Examiner: Prof. Ângelo Martins, PhD

Supervisor: Prof. Hugo Sereno Ferreira, PhD

July 3, 2019

Abstract

Industry 4.0 is revolutionizing the way factories compete in the market, aiming to achieve a higher level of operational efficiency and productivity by augmenting the level of automatization in factories.

Industrial Internet-of-Things (IIoT), the subset of the Internet of Things specific to industrial applications, results from the addition of sensing and actuating capabilities to industrial environments to improve the overall manufacturing processes. It focuses on making the manufacturing process more efficient, sustainable and autonomous, resulting in overall money and time savings. Supply chains become dynamic interconnected systems, boosting operational efficiency and reducing detection and troubleshooting time through the collection of data captured by the sensors and actuators integrated into the shop-floor machinery. This enables real-time monitoring of the production line which supports the advanced maintenance process, meant to ensure optimal performance of the manufacturing system at all times, avoiding machine downtime and detecting execution failures as soon as they happen with suitable debugging systems. These failures are hard to reproduce due to the nature of IIoT systems, with the non-determinism of concurrent processes, the time-sensitive nature of applications and partial failures that may occur. Therefore, the debugging system needs to be directly connected to the device when the exception or crash occurs, capturing information from these devices in real-time.

Some of these systems have highly-complex tasks and operations and need to be programmed accordingly. The use of visual programming (i.e. visual workflows) is common in these systems due to the abstraction they provide. The integration of workflows in the maintenance process simplifies the representation of operations and thus the understanding of the system's execution, allowing the common factory worker with close to none IT-knowledge to accurately identify and report system failures as soon as they happen. However, these programming environments have several deficiencies on what regards debugging capabilities, mostly due to the constraints that difficult the use of traditional mechanisms.

The work presented in this dissertation approaches these issues, delving into the design and implementation of a multi-strategy debugging mechanism into a commercial-grade Manufacturing Execution System. We intend to design, construct and test/evaluate a prototype to demonstrate the feasibility of both synchronized and snapshot remote debugging applied to IIoT automation workflows. The main goal is to achieve is a 0-downtime workflow remote debug approach, contemplating the design and implementation of a suitable debugging protocol and a supporting platform where workflows can be easily configurable, monitored and debugged.

To validate the approach, a *proof-of-concept* was developed and validated against different debugging scenarios. Such solution eases the maintenance process, providing the means for the detection of failures for both simulated and real-time production shop-floor execution systems, thus minimizing incorrect machinery behavior and machine inactivity due to failures, consequently reducing resource losses and improving production efficiency.

Resumo

A Indústria 4.0 está a revolucionar a maneira como as fábricas competem no mercado, tendo como objetivo alcançar um mais alto nível de eficiência operacional e de produtividade, através do aumento do nível de automatização nas fábricas.

A Internet das Coisas Industrial (IIoT), o subconjunto da Internet das Coisas específica para aplicações industriais, resulta da adição de capacidades de deteção e atuação a ambientes industriais para melhorar os processos gerais de fabrico. Este tipo de IoT concentra-se em tornar o processo de manufatura mais eficiente, sustentável e autónomo, resultando na poupança de tempo e dinheiro. As cadeias de fornecimento tornam-se sistemas dinâmicos e interconectados, aumentando a eficiência operacional e reduzindo o tempo de deteção e resolução de problemas através da recolha dos dados capturados pelos sensores e atuadores integrados nas máquinas em execução no chão de fábrica. Isso permite a monitoração da linha de produção, em tempo real, suportando o processo de manutenção avançada, garantindo o desempenho ideal do sistema de manufatura em todos os momentos, evitando paragens inesperadas de máquinas e detetando falhas de execução assim que estas ocorrem, com sistemas de debugging adequados. Estas falhas são difíceis de reproduzir devido à natureza dos sistemas IIoT, com o não determinismo de processos concorrentes, a sensibilidade à passagem do tempo e a falhas parciais que podem ocorrer. Por estas razões, um sistema de debugging deve estar diretamente ligado aos dispositivos IIoT quando as anomalias ou falhas ocorrem, capturando informações dos dispositivos em tempo real.

Alguns destes sistemas têm tarefas e operações altamente complexas e precisam ser programados adequadamente. O uso de programação visual (ou seja, representação visual dos fluxos de trabalho) é comum nesses sistemas devido à abstração que é fornecida. A integração de fluxos de trabalho, através de programação visual, no processo de manutenção simplifica a representação das operações e, assim, a compreensão da execução do sistema, permitindo que um trabalhador comum da fábrica, sem conhecimento específico de IT, consiga identificar e comunicar com precisão as falhas do sistema assim que elas ocorrem. No entanto, esses ambientes de programação têm algumas deficiências em relação aos recursos para fazer debugging, principalmente devido às restrições deste tipo de sistemas, que dificultam o uso de mecanismos tradicionais.

O trabalho apresentado nesta dissertação aborda essas questões, através do design e da implementação de um mecanismo de debugging multiestratégia num Manufacturing Execution System comercial. Pretendemos fazer o design, construir e testar/avaliar um protótipo para demonstrar a viabilidade de debug remoto, em modo sincronizado ou com snapshots, aplicada aos fluxos de trabalho de automação em sistemas IIoT. O principal objetivo é alcançar uma abordagem de debugging remoto de um fluxo de trabalho sem latência, contemplando a construção e a implementação de um protocolo de debugging adequado e de uma plataforma para suporte, na qual os fluxos de trabalho podem ser facilmente configurados e monitorados, permitindo o debug destes fluxos durante a sua execução.

Para validar a abordagem, foi desenvolvida e validada a prova de conceito elaborada, considerando diferentes cenários de debugging. Esta solução facilita o processo de manutenção,

fornecendo meios para a deteção de falhas em sistemas de execução simulados ou em tempo real aquando em produção, minimizando o comportamento incorreto e a inatividade das máquinas no chão de fábrica, devido à ocorrência de falhas, consequentemente reduzindo as perdas de recursos e melhorando a eficiência da produção.

Acknowledgements

First of all, I would like to thank my supervisor Hugo Sereno Ferreira and co-supervisor João Pedro Dias for the help and guidance given during the realization of this dissertation. Without their suggestions on how to improve my work, this project would not have achieved the present quality.

I would also like to thank my supervisors at Critical Manufacturing, José Pedro Silva for the suggestions on how to improve the development of this project and Micael Queiroz for the constant availability to provide input regarding the project's state and all the time spent helping me figure out why my code wasn't working.

Special thanks to my boyfriend Eduardo Leite for all the support and motivation, inspiring me every day to learn more and to get better at what I do.

Last but not least, I want to express my gratitude to my friends and family, for the support and trust, allowing me to make my own decisions which took me to where I am now.

Andreia Cristina de Almeida Rodrigues

"Things that matter most must never be at the mercy of things that matter least."

Johann Wolfgang von Goethe

Contents

List of Figures

List of Tables

Abbreviations

ACI	Automation Controller Instance
CPPS	Cyber-Physical Production Systems
CPS	Cyber-Physical Systems
IIoT	Industrial Internet of Things
IoMT	Internet of Manufacturing Things
IoT	Internet of Things
MES	Manufacturing Execution Systems
MOM	Manufacturing Operational Management
PLC	Programmable Logic Controller
SDLC	Software Development Life Cycle

Chapter 1

Introduction

Contents

This Chapter presents an overview of the scope of this project, as well as the problems it aims to solve. Section 1.1 presents contextualization of the main concepts related to this work. Sections 1.2 and 1.3 present, respectively, the motivation that drives this project and the goals trying to be achieved. To finalize, Section 1.4 presents the overall structure of this document.

1.1 Context

Ever since the beginning of industrialization, increasing efficiency in manufacturing systems has been a constant endeavor. Companies thrive to improve productivity and reduce costs without compromising quality, enhancing the efficiency of the manufacturing process.

The birth of Internet-of-Things followed by its application to industrial environments, *viz.* Industrial Internet of Things (IIoT), pushes the efforts of improving the manufacturing processes even further, towards the so-called Industry 4.0. It unifies the concept of objects, machines, assembly lines and whole factories leading to the "smart factory" vision with the implementation of cyber-physical systems (CPS), where the physical and virtual worlds converge. CPS are physical objects with embedded software and computing power. The manufacturing equipment will turn into cyber-physical production systems (CPPS) where the computational and communication resources of a device can be used directly for various control, supervisory, or monitoring functions. Factories become more intelligent, flexible, dynamic and autonomous, leading to the growth of

productivity, reduction of costs, increase of effectiveness and efficiency and improvement of the overall quality of the products [PGPM19, Lu17, AL15, IML+16a].

The use of networked sensors and intelligent devices to collect data about the manufacturing processes, boost operational efficiency and reduce detection and troubleshooting time, hence resulting in overall money and time savings. The computational elements that control the physical entities (the equipment), using sensors and actuators feedback in real-time, enable the detection and prevention of failures to avoid major losses. The software controlling these devices requires support for management, configuration, control and debug of operations, especially during the maintenance process [Orab, Zer].

Despite the adoption of Industry 4.0 and IIoT in the manufacturing industry revealing many positive consequences, the recent mass customization trend has contributed to the increase in the complexity of manufacturing systems [Sza19, BXW14]. This, along with the heterogeneous nature of the manufacturing environment, increases the need for maintenance of automated production systems, required to ensure the software's quality and reliability overtime [VHFR+15]. With a fast changing market and customers demanding a higher level of product customization, industrial software must be constantly updated and re-configured, imposing pressure over the need for software development and maintainability [MUK00].

The maintainability process benefits from the analysis and management of the information gathered by the sensors and actuators integrated within the machinery, enabling life cycle observation to ensure that the solution is operating according to the stated requirements [MBRB17, VHFR+15].

Ultimately, the increasing complexity of manufacturing systems, resulting from particularities as the heterogeneity, large-scale, highly-dynamic topologies, real-time needs, human-in-the-loop considerations and wide-range of application scenarios of IoT, along with coordination and collaboration between different CPS systems (systems of systems), is manifested in many ways, including non-linear hybrid systems with behaviors which are hard to predict and verify, composed of many interacting parts and properties, making software faults very hard to locate and debug given the amount of complexity involved. This imposes a need for continuous software adaptation, maintenance and life cycle observation of the production machinery, which can be greatly assisted with suitable debugging techniques, to ensure the software's quality and reliability [VHFR+15, MZ16, WTO15, DFF18].

1.2 Motivation

Both developing and debugging IoT systems is hard. These processes become even harder when we target manufacturing environments. The debugging needs of IIoT systems are still lacking in terms of tools, approaches, and methodologies that can be used for such. The problem grows when considering that, commonly, these systems are programmed using *visual workflows*, that typically lack tools for debugging. These workflows are an abstraction of the different tasks to be executed by the equipment, creating blocks that aggregate some of the steps to be performed, receiving

inputs from the previous tasks and delivering outputs to be used by the following [AGB93, SP07, DFF18].

In industrial systems, the increasing need for production efficiency and flexibility require active and real-time maintenance from technicians to reduce machine downtime. For expensive machines, usually, often downtime is more expensive than the actual repair in terms of lost production resource. Real-time monitoring, advanced maintenance, and debugging systems, through visual notations for the abstraction of the machinery workflow, can play a significant role in tackling this issue. Since IIoT systems are characterized by their time-sensitive, non-determinist processes, it's important to detect the software failures in real-time and remotely, through communication with the device where the exception or crash occurred [MZ16, MBC+17, SP07].

A Manufacturing Execution System (MES) is an information system that focuses on the digitization of shop-floor activities, controlling and optimizing the manufacturing production processes in near real-time, aiming to achieve and maintain a constant high performance in the highly competitive and rapidly changing manufacturing environment [WWB18, Mani].

This dissertation was proposed in the context of Critical Manufacturing's Manufacturing Execution System. It is a software platform that provides manufacturers with information for monitoring, controlling and assuring the correct execution of the production processes, gathering information along the process and providing tools for analyzing this information in order to optimize procedures and make the production process more efficient [Mani]. A module for the abstraction of automation workflows being carried out by the machines in production, as well as equipment integration, was introduced in this MES solution, along with local debugging capabilities for simulation of equipment execution through inputs and outputs provided by the user [Mand, Mang].

In order to prevent machinery downtime and to assure maximum production efficiency and effectiveness, real-time remote maintenance should be implemented as an extension of the currently available IoT module, providing remote debugging of operations' workflows being followed by the machines in the production line. This can be achieved by adapting the currently existent workflow engine and interface to enable receiving the workflow inputs and outputs directly, in real-time, from the equipment in execution, showing the debugging process in a user-friendly way to allow workers with close to none IT-knowledge to operate with this platform. This will provide manufacturers with a real insight into the machine's operations, extending the already existing monitoring and controlling capabilities of the production process.

1.3 Aims and Goals

The main goal of this work is, by studying the best practices for debugging different kinds of systems, to design and develop a debugging approach (and protocol) that can be used for IIoT and meet the requirements of a real-world Manufacturing Execution System. To validate both the approach and the underlying protocol, a *proof-of-concept* was implemented on top of the Critical Manufacturing's Manufacturing Execution System, allowing us to run a set of preliminary tests to verify its viability.

The requirements we wish to fulfill, simultaneously, with this solution are the following:

1. Provide a remote connection to a specific physical device running elsewhere without compromising its current execution;
2. Abstraction of the physical device's sequenced execution tasks through a workflow;
3. Ability to remotely debug the workflow being executed by the physical device in real-time.

These are expected to be answered through (a) the development of a prototype that aims at adapting a traditional remote debug session to the manufacturing environment, connecting the debugger with some physical equipment executing elsewhere through an intermediary, that will be listening to any changes that happen in the machinery, (b) the development of a remote debugging protocol which will have defined messages for informing or notifying the receiver of something that happened or must happen, related with each distinct action, generalizing the request messages as much as possible to simplify the protocol's usage, (c) the management of the debug sessions, ensuring that sessions debugging the same equipment are not interfering with each other, often checking if there are any data inconsistencies, using the developed protocol, (d) the creation of an abstraction of the execution tasks using a representation through workflows that is expected to provide a better understanding of the machines' execution, allowing to detect failures in real-time and to better identify what is causing the incorrect behavior, (e) handling all types of connection events and connection issues that may arise, always keeping the user updated on what is happening and immediately trying to take the debugging application back to a stable state if something unexpected happens.

The expected final result of the present work is a functional prototype of a tool, as the process's *proof-of-concept*, that fulfills the aforementioned functional requirements, capable of debugging both simulated and real-time production shop-floor execution systems, granting a better insight of the production machinery. It is expected that the clarity of the machines' operations provided by this prototype has a greater impact on reducing incorrect machinery behavior, detecting execution failures immediately as they happen and machine inactivity due to failures, consequently reducing resource losses and improving production efficiency.

1.4 Dissertation Structure

The current dissertation is structured as follows:

Chapter 2 provides a contextualization of the main concepts related to this dissertation.

Chapter 3 presents some of the current approaches on debugging, remote debugging, debugging of workflows and remote debugging protocols, while simultaneously identifying the shortcomings of the current approaches *w.r.t.* remote debugging of IIoT workflows.

Chapter 4 describes the current challenges of debugging IIoT systems, regarding the current solutions found, and presents the proposed solution, all the features to be implemented and how it will be evaluated.

Chapter 5 details the proposed solution, explaining the main software components, the interaction between the protocol messages and the techniques used to tackle some of the issues encountered to make the software as reliable as possible.

Chapter 6 presents the evaluation done to validate the *proof-of-concept* developed and all the simulated scenarios that were used.

Finally, Chapter 7 presents a brief summary of this dissertation, the main contributions and conclusions of this work, and the future goals for this project.

Chapter 2

Background

Contents

In order to understand the concepts covered in this document, this Chapter provides an overview of the main subjects related to the topics of the dissertation.

It starts with a presentation of the Industry 4.0 and Internet of Things concepts, their history, and definition and the relation with one another. It is followed by the definition of a Manufacturing Execution System and information found relevant regarding Critical Manufacturing's MES solution. Afterward, this Chapter presents the phases of the software development life cycle, as well as how the maintenance phase is performed and how is it adapted for the context of industrial systems. Lastly, it covers the description of the debugging process and how it can be applied in factories for the machinery maintenance process.

2.1 Industry 4.0

Ever since the beginning of industrialization and over the course of history, technological advancements and the appliance of new resources to improve industrial processes have led to paradigm shifts named "industrial revolutions" [LFK+14].

To this day, the industry has had three major revolutions (illustrated in Figure 2.1) that led to changes in the social structure of society, shaping the way people live, work and relate to one another [PGPM19].

The first industrial revolution started at the end of the 18th century and revolutionized the field of mechanization. It ensured the transition from manual to machine labor and it's connected to the use of water and steam power to mechanize production. The second big revolution occurred nearly a century later, and it was related to the intensive use of electrical energy to create mass production. The third revolution happened in the 20th century with the widespread of digitalization. It used both electronics and information technology to automate production [LFK+14, PGPM19, Lu17].

The fourth industrial revolution, known as "Industry 4.0", emerged in Western countries in the 21st century with the goal of achieving a higher level of operational efficiency and productivity by augmenting the level of automatization in factories [Lu17].

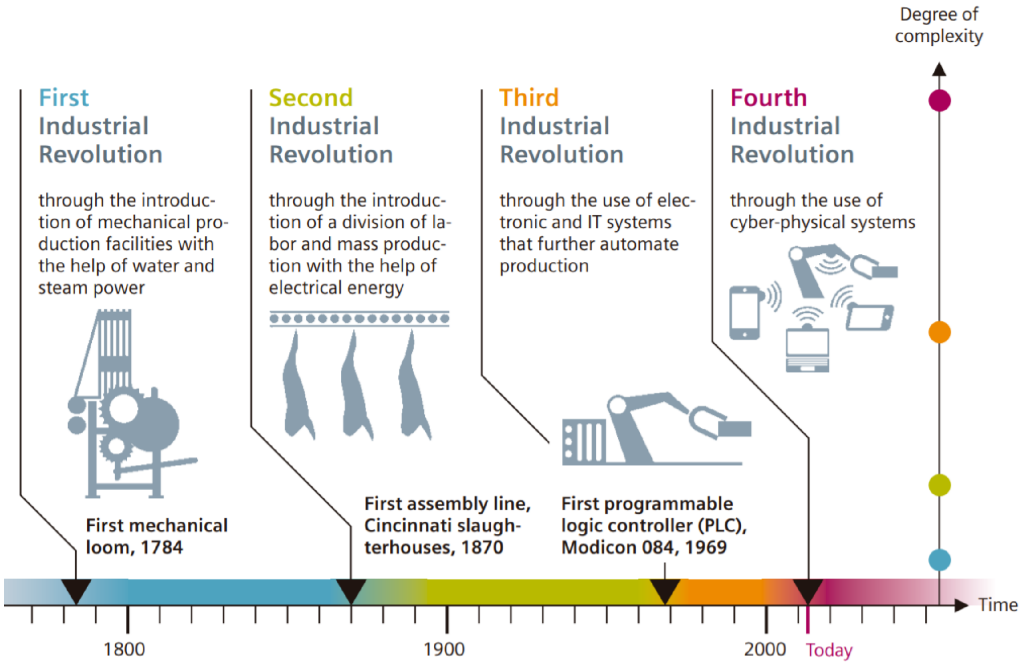

Figure 2.1: Timeline from the first industrial revolution to Industry 4.0 (Source: German Research Center for Artificial Intelligence (DFKI), 2011).

Industry 4.0 dictates the end of traditional centralized applications for production control and encompasses numerous technologies and associated paradigms, such as the Internet of Things (IoT), Enterprise Resource Planning (ERP), Radio Frequency Identification (RFID), Cloud Computing and Big Data [Lu17, AL15].

This concept describes primarily IT-driven changes in manufacturing systems [Lu17] and is defined as *"the integration of complex physical machinery and devices with networked sensors and software, used to predict, control and plan for better business and societal outcomes"* [II13].

It unifies the concept of objects, machines, assembly lines and whole factories [PGPM19] leading to the "smart factory" vision with the implementation of cyber-physical systems (which *"extend real-world physical objects by interconnecting them all together and providing their digital descriptions"* [SSH+18]), fulfilling the demanding requirements of production. Factories become more intelligent, flexible, dynamic and autonomous, leading to the growth of productivity, reduction of costs, an increase of effectiveness and efficiency and improvement of the overall quality of the products [Lu17].

As a result, Industry 4.0 will accelerate industry to achieve unprecedented levels of operational efficiencies and growth in productivity [TS16].

2.1.1 Internet of Things

The Internet-of-Things is an emerging paradigm where everyday objects can be equipped with identifying, sensing, networking and processing capabilities, allowing these objects to communicate with one another and with other devices and services over the Internet to accomplish some objective [WADX15]. It conceptualizes a world where there is a seamless integration of people and devices to converge the physical world with human-made virtual environments [BVD16], in which all things are wirelessly and seamlessly connected and can be controlled remotely and exchange data at any time [LXZ15].

As a complex cyber-physical system (or system of systems), IoT devices are equipped with embedded software and computer power through sensors, actuators, processors, and transceivers. Sensors and actuators are devices that help to interact with the physical environment. While sensors provide inputs about the device's current state (internal state and environment), an actuator performs actions to affect the environment or device in some way. The combination of these elements can enable objects to simultaneously be aware of their environment and interact with people, both goals of IoT. Manufacturing equipment will develop into cyber-physical production systems (CPPS), software enhanced machinery with autonomous computing power and embedded devices. CPPS know their state, their capacity, and their configuration options and are capable of taking decisions autonomously [II13, WADX15, SS17, AL15].

The IoT paradigm opens the doors to new innovations that will build new interactions among things and humans, enabling the attainment of smart cities, infrastructures, and services for enhancing the quality of life and usage of resources. Objects and devices in IoT will, therefore, *"be mobile, dynamic, and will generate massive amounts of frequently changing information"* derived from both sensors and actuators feedback [BVD16, WADX15].

IoT finds various applications in health care, fitness, education, entertainment, social life, energy conservation, environmental monitoring, home automation, and transport systems [SS17]. An IoT Analytics report [Scu] that analysed the top IoT enterprise-level segments pointed out that the most relevant segments are Smart City, Industrial IoT, Smart Building, Smart Car, Smart Energy/Grid, eHealth, Smart Supply Chain, Smart Agriculture, and Smart Retail, by this order of relevancy (illustrated in Figure 2.2). However, consumer-level IoT was not considered in this report (e.g. wearables and home automation). IoT enterprise applications usually fall under the

following three categories: (1) monitoring and actuating, (2) business process and data analysis, and (3) information gathering and collaborative consumption. The report also mentions that most IoT enterprise projects focus on cost-reduction, the main value driver for about 54% of the projects that were analyzed. The other main concerns were related to increasing revenue (about 35%) and to increase in overall safety (24%) [BVD16, Scu].

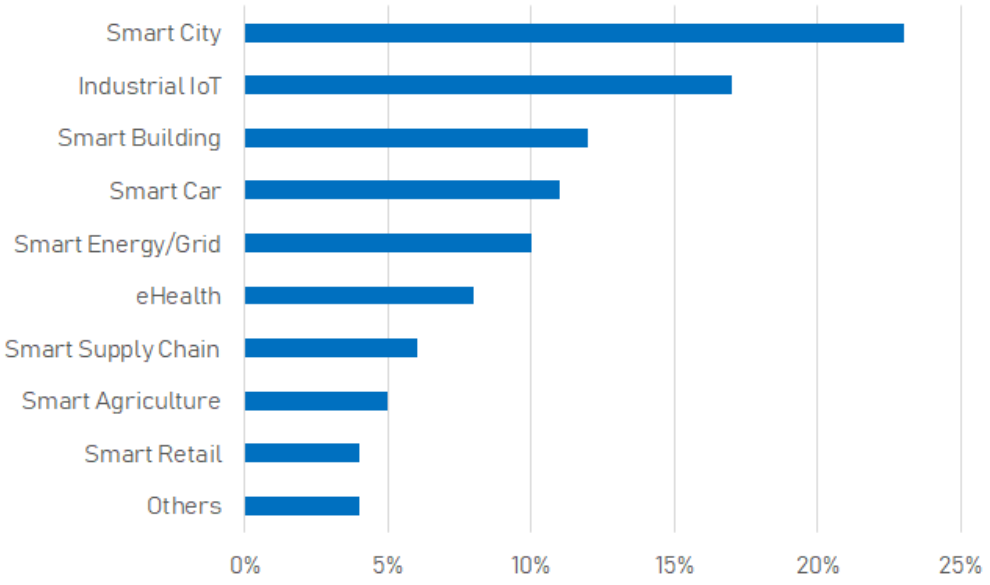

Figure 2.2: Statistics based upon 1600 enterprise IoT projects circa 2018, excluding consumer level IoT projects (Source: [Scu]).

The Internet of Things is *"the trend and direction for the new industrial revolution"*, Industry 4.0. By providing a connection between humans and machines, it's possible to obtain new information and knowledge, thus improving the efficiency and effectiveness of knowledge development and management in today's world [II13].

2.1.2 Industrial Internet Of Things

Industrial Internet of Things (IIoT), also referred to as Industrial Internet, is the subset of IoT specific to industrial applications. Smart and efficient manufacturing can be achieved with IIoT, connecting the shop-floor to production management. The fact that machines can perform specific tasks such as data acquisition and communication more accurately than humans has boosted the adoption of IIoT over the last few years, becoming a popular interest among big high-tech companies [BVD16, Azi19].

IIoT is about connecting all the industrial assets, including machines and control systems, with the information systems and the business processes. It focuses on the manufacturing stage of the product's life cycle, aiming for a quick and dynamic response to demand changes by equipping

high-tech products such as sensors and actuators, software and wireless connectivity in the production equipment [Lu17]. These generate large amounts of data that can be collected and analyzed with the goal of getting a better understanding of the manufacturing process and enable a more efficient and sustainable production system, reducing costs without compromising quality, by bringing transparency about the machines' operations, the materials used, the facility logistics, and even the human operators, feeding analytic solutions and leading to optimal industrial operations. The analysis of the data collected can aid the machinery maintenance process, as it will allow to understand and identify the top causes of failure and predict component failures to avoid unscheduled machine downtime, which usually leads to company resource losses [LGS17, SSH+18].

Industrial IoT has grown from a variety of technologies and their interconnections. It *"covers the domains of machine-to-machine (M2M) and industrial communication technologies with automation applications"*, as well as big data analysis and machine learning techniques [BVD16, SSH+18]. Some of the most important technologies are described below:

Cyber-physical Systems CPS are automated systems responsible for the connection of operations of the physical reality with computing and communication infrastructures. It comprises a set of interacting physical and digital components, which may be centralised or distributed, providing a combination of sensing, control, computation and networking functions, to influence the real world outcomes through physical processes [BHCW18, Lee08, RLSS10].

Cloud Computing A cloud computing system provides computing services (e.g. computing, storage) from cloud resource pools to particular manufacturing tasks. It provides high reliability, scalability, and interoperability, which can improve the efficiency of computing resource utilization in IIoT systems, keeping files in a cloud-based storage system rather than on local storage devices [SWYS11, Fra19].

Edge Computing Edge computing brings the computer data storage closer to the location it is needed. Unlike cloud computing, it leverages computing resources from network edge devices. As the computing services are located close to end-devices, these services are provided with much better latency performance [SWYS11].

Big Data Analytics Big data in IIoT refers to the huge volume, velocity and veracity of the data that is gathered from the industrial sensors, actuators and devices embedded in the manufacturing equipment. Enabling big data sharing and analytics is important in order to improve the efficiency of manufacturing process through the monitoring of large-scale industrial systems [SWYS11].

AI and Machine Learning Machine learning is a subset of artificial inteligence that can use huge amounts of data, of different types and sources, collected from the sensors and actuators embedded in the manufacturing equipment to find highly complex and non-linear patterns

and transform raw data into feature spaces, which are then applied for prediction, detection, classification, decision, regression, or forecasting of data related to manufacturing processes. For a manufacturing control system, it can enable self-awareness, self-diagnosis and self-healing, efficient management, effective resource utilization, and timely maintenance [SWYS11, WWIT16].

The architecture of an IIoT system must highlight the extensibility, scalability, modularity, and interoperability among heterogeneous devices using different technologies, providing an higher level of abstraction that helps identifying issues and challenges for different application scenarios. Several reference architecture frameworks have been conceived in the past, in different application contexts, for both IoT and IIoT. The Industrial Internet Consortium has recently released a reference architecture for IIoT systems, which has been widely accepted, focusing on various viewpoints (e.g. business, usage, functional and implementation views), providing models for each one. It is characterized by three-tiers: edge, platform and enterprise (illustrated in Figure 2.3) [SSH+18, Inf18, II19]:

Edge Tier Collects data from the edge nodes, which comprises assets, edge devices, sensors and control systems, interconnected by an independent local area network to communicate with the edge gateway, which in turn connects to larger networks of the platform tier, providing global coverage;

Platform Tier Integration platform which enables the integration of the assets, consolidating processes and analyzing data flows, providing management functions for devices and assets. It receives, processes and forwards control commands from the enterprise tier to the edge tier;

Enterprise Tier Implements domain-specific applications, decision support systems and provides end user interfaces. It receives data flows from both edge and platform tiers, also sending control commands to conduct them.

Industrial IoT *"affects all the industrial value chain and is a requirement for smart manufacturing"* [SSH+18].

2.1.3 Summary

Industry 4.0 is an emerging concept that aims at a higher level of operational efficiency and productivity in factories by increasing the level of automation in manufacturing systems [Lu17].

The idea of a "smart factory" where objects, machines, assembly lines and whole factories are connected through a single network that can communicate in real-time to achieve a meaningful purpose is achieved with the implementation of IoT in the manufacturing process [WADX15], through the integration of embedded sensors, actuators, processors, and transceivers. The feedback received from these sensors and actuators generates massive amounts of data that allow the gaining of new information and knowledge about the factory's operations, leading to a better understanding of the manufacturing process [LGS17, SSH+18].

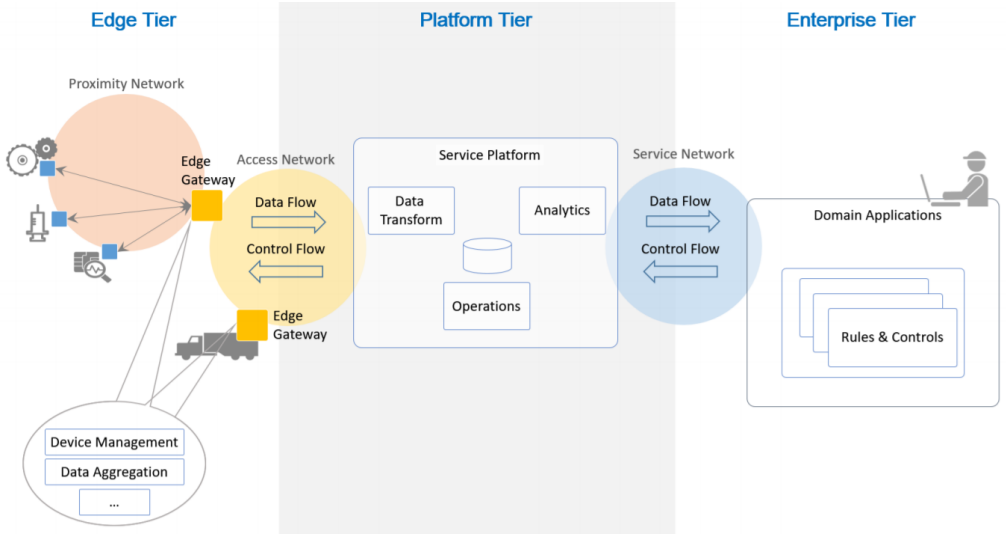

Figure 2.3: Three-tier IIoT system architecture proposed by the Industrial Internet Consortium (Source: [II19]).

The subset of the Internet of Things specific to industrial applications (IIoT) is, therefore, the trend and direction for the current industrial revolution, Industry 4.0, focusing on making the manufacturing process more efficient, sustainable and autonomous, resulting in overall money and time savings [Azi19, LGS17].

2.2 Manufacturing Execution Systems

A Manufacturing Execution System (MES) is *"an information system that drives the execution of manufacturing operations"* [Manf]. It is *"at the heart of industrial organizations' endeavors"* and focuses on the digitalization of shop-floor activities with the collection, analysis, and exchange of information captured in real-time during this process [WWB18]. This type of system aims to achieve and maintain constant high performance in the highly competitive and rapidly changing manufacturing environment [Mani].

2.2.1 Overview

Since the 90's, Manufacturing Execution Systems have been attracting more and more attention from manufacturers [QZ04]. According to MESA, or Manufacturing Enterprise Solutions Association, a manufacturing execution system is a dynamic information system that ensures the effective execution of the manufacturing operations through the gathering of real-time data, guiding, triggering and reporting on shop-floor activities as events occur. It *"manages production operations from the point of order release into manufacturing to the point of product delivery into finished goods"* [Manb].

By obtaining current and accurate data directly from the shop-floor, MES guides, initiates, responds to and reports on plant activities as they occur. The rapid response to changing conditions, coupled with a reduction on activities that don't add value to the manufacturing process, increases the effectiveness of these operations and processes. This system improves the return on operational assets, on-time delivery, inventory turns, gross margin and cash-flow performance [dUAP09].

Back in 1997, MESA defined the principal MES functionalities, as follows [Int97]:

Resource Allocation and Status Manages resources (e.g. machines, tools, materials, documents) and tracks all the operations they are involved in, in the production process;

Operations/Detail Scheduling Schedules all the activities to be performed, taking into account their duration of time, execution sequence, amount of resources available and attributes of specific production units, for optimizing the manufacturing process performance;

Dispatching Production Units Manages the flow of production units in the form of jobs, orders, batches, lots, and work orders, controlling these units and dispatching them to where they need to be;

Document Control Controls records and forms that must be maintained with the production unit (e.g. work instructions, recipes, drawings, standard operating procedures, part programs, batch records and engineering change notices) managing and distributing them when required. Also gathers certification statements of work and work conditions;

Data Collection/Acquisition Monitoring, gathering and organising data about processes, materials and operations;

Labor Management Tracking and management of personnel during a shift based on their qualifications, work patterns and business needs;

Quality Management Real-time analysis of data collected from the manufacturing process to ensure the product's quality and identify problems;

Process Management Production monitoring, directing the flow of work in the production process based on the planning and the current production activities;

Maintenance Management Tracks and directs the activities to maintain the equipment and tools, ensuring their availability for the manufacturing process;

Product Tracking and Genealogy Provides visibility of where work is at all times and its status information;

Performance Analysis Provides real-time reporting of manufacturing operations results, comparing them to past history and expected business results.

This way, by implementing these functionalities, MES systems are responsible for monitoring and assuring the correct execution of the production process on the shop-floor, monitoring

and controlling the material used for this process, gathering information along the way and providing the tools required for the analysis of the data obtained, in order to optimize production efficiency. The delivery and management of work instructions also fall under its responsibilities, as well as the providence of the tools necessary for solving problems that may come up during production [Mani].

Manufacturing execution systems focus on the vertical integration of manufacturing processes with business processes, working as a middle-layer, by bridging enterprise information systems (ERPs) with the shop-floor equipment, implementing manufacturing operational management (MOM) functions in the enterprise. It sets a connection between ERP, as a system of decision support, and the shop-floor, mostly concerned with automated control [RMK16, WWB18].

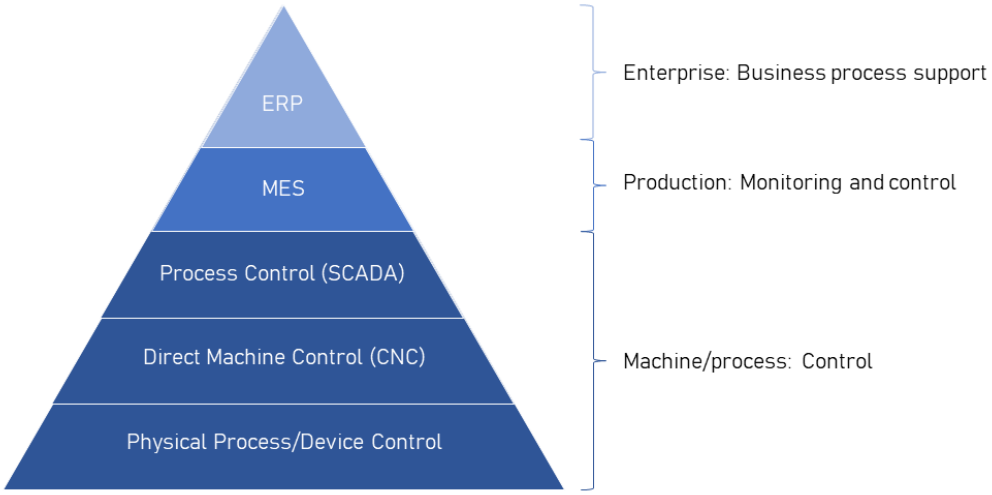

Figure 2.4: Automation pyramid (Adapted from: [WWB18]).

The automation pyramid classifies industrial systems into five distinct layers as it is illustrated in Figure 2.4. On top of the pyramid is ERP, which *"supports the execution of business processes, such as order fulfillment or inventory control"*. The bottom levels are related to automated control using real-time sensors and actuators and are specifically hardware oriented, e.g. sensors, programmable logic controllers (PLC), and supervisory control and data acquisition (SCADA) systems. MES is positioned at the middle level, bridging the gap between the upper and lower levels in the pyramid, complementing office-level information systems and extending their capabilities [WWB18, QZ04].

The analysis of the data collected throughout the product's life cycle is of growing importance for organizations. It is evident that the increasing amount of information collected from the equipment is due to the tremendous increase in the degree of automation on the shop floor. MES is responsible for the collection and analysis of this information, followed by the integration and presentation of the results in the industrial production, providing real-time, accurate, granular data which allows employees to have a better insight into the manufacturing processes, leading to predictable manufacturing processes, decreasing cost, increasing quality and meeting efficiency

requirements. Workers can optimize the decisions for controlling the manufacturing process by providing accurate and timely information through real-time interfaces for the automated equipment, enabling them to quickly react to any issues that may arise during the manufacturing process. The data gathered from the manufacturing equipment should be also analyzed in real-time to provide enterprise-wide optimal decisions such as validation of the equipment's setup and production schedule, monitoring of the process status and indication of the trend of product quality, and quicker reaction to dynamic customer demands [dUAP09].

The implementation of an MES system enables companies to improve productivity in their factories, improve customer satisfaction and provide an overall competitive advantage in a dynamic and competitive marketplace [WWB18, WWB18, QZ04].

MESA	**ISA-95**	**VDI**
Operations / detail scheduling	Detailed Production	Detailed planning and detailed scheduling control
Resource allocation and status	Scheduling	
Document control	Production data collection	Operating resources management
Dispatching production units	Production resource management	
Performance analysis		Material management
Labor management	Production definition management	Personnel management
Maintenance management	Product tracking	Data acquisition and processing
Process management	Production dispatching	Interface management
Quality management	Production execution	Performance analysis
Data collection and acquisition	Production performance analysis	Quality management
Product tracking and genealogy		Information management

Table 2.1: Set of functions assigned to Manufacturing Execution Systems according to ISA-95, MESA, and Verein Deutsche Ingenieure (VDI) at the manufacturing operational management (MOM) functions level (Source: [IML+16b]).

These manufacturing systems have been pivotal in the performance, quality and agility needed for the challenges created by a globalized manufacturing business. Future improvements and enhancements to these systems should focus on the following four main pillars [AL15]:

Decentralization For a smart product, or CPS, its computing power can be elsewhere as long as it is capable of identifying itself and connect to a physically centralized system, providing its position and state at anytime. MES is a single application but acts decentralized by using agents/objects to represent the shop-floor entities, that are autonomous and negotiate with each other to provide unique products;

Vertical Integration All services provided by the CPPS are exposed, allowing their participation in business processes for compliance or, more broadly, for processes related to quality, logistics, engineering or operations. MES systems shall be truly modular and interoperable, logically decentralized, making all functions and services available to be consumed by any smart shop-floor entity;

Connectivity and Mobile Most manufacturing execution systems already provide connectivity, but it is directed towards more sophisticated facilities with advanced equipment that allows a direct access to the information that is generated. This connectivity should be widespread in manufacturing facilities of different sophistication levels, providing more adaptable interfaces for accessing any type of equipment;

Cloud Computing and Advanced Analysis The "smart factory" vision requires achieving an holistic view of manufacturing operations through the integration of data from several different sources. Advanced analytics should be implemented for fully understanding the performance of the manufacturing processes, ensure the maximum quality of the produced products and optimize the supply chain, identifying inefficiencies and allowing corrective or preventive actions to be performed.

A study carried out by MESA from 2006 [Int06] (the most recent study by this association that was released to the public), providing insight of how MES systems have served manufacturing enterprises with many benefits, shows that companies that had these systems implemented had improved their operations in 31%, where 30% (vs 15% of companies that don't use MES systems) had improved their planned vs emergency maintenance work orders and 28% (vs 10%) had improved their production flexibility to accommodate dynamic customer demands. It also shows that 19% of these companies had more likely, on average, made gains regarding business and financial metrics, increasing productivity (per square foot) in about 37%, with 36% improving their cash-to-cash cycle time.

Manufacturing Execution Systems play a critical role in Industry 4.0, as it *"accommodates the Industrial Internet-of-Things (IIoT)-enabled production marketplace"*, playing a key role as an enabler of further innovation in manufacturing [Mani, WWB18].

2.2.2 Critical Manufacturing MES

Critical Manufacturing's MES solution is designed for complex discrete industries including solar, electronics, semiconductor and medical devices. It is a software platform with a deep set of modular applications (more than 30, shown in Figure 2.5), that are fully interoperable, that *"provides manufacturers in complex industries with maximum agility, visibility and reliability"* of manufacturing processes. It is a Web-based application, granting the ability to run on multi-platform and multi-form size devices [Manf, Mana].

It claims to ensure the following [Mane]:

- Low total cost of ownership through the use of standard integrated components and having a strong focus on simplicity and ease of use, deployment, and maintenance;
- Agility through a powerful, extensive framework, with rich functionality and ease of customization and integration that each customer can configure for their own needs;
- Versatility thought a generic data model and a rich, extensible, customizable application platform that runs on multi-platform and multi-form size devices;

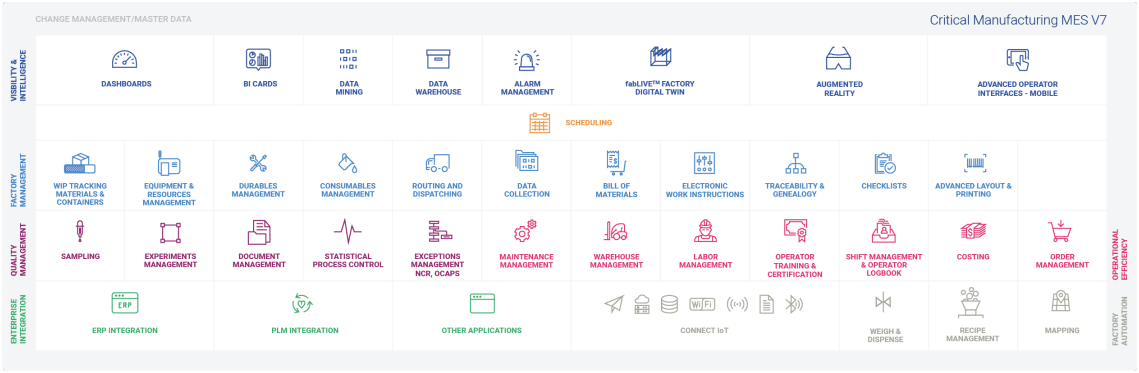

Figure 2.5: Critical Manufacturing MES V7 modules (Source: [Manc]).

- Optimal performance reliability through the use of standard, proven cost-effective hardware and software solutions, implementing an architecture formulated for high performance, scalability and smart database partitioning.

All three tiers of the infrastructure framework - Presentation, Business, and Analytics - are designed to be fully customizable and extensible, *"providing partitioning, modularity, and scalability of applications, being designed to work seamlessly together"* [Mane].

The recent advances in the integration of IIoT led to the development of one of the most recent solutions implemented in Critical Manufacturing MES called "Connect IoT", integrated into the "Factory Automation" module. It is a lightweight, low footprint solution that enables fast, easy integration of both standard equipment interfaces and new IoT devices into a distributed, decentralized architecture in which an autonomous production network can be built, reaching across all types of devices [Mand, Manj]. Through this module, production engineers and system integrators can connect their shop-floor equipment to Critical Manufacturing MES, dramatically reducing the time and effort for this very integration.

Beyond that, Connect IoT has a complete, single graphical overview of all automation workflows being carried out by the machines in production. It allows to *"easily drag and drop equipment and IoT devices into a model of the shop floor, creating a network of entities"* [Mang]. This solution provides ways to create and update complex logic with no code required using a user-friendly interface that allows workers with close to none IT-knowledge to work with this platform [Mand]. The system automatically abstracts workflows from specific drivers and allows re-usable, extensible workflow elements with complex logic to be created [Manj].

So far, this solution allows visual debugging of these workflows, simulating the equipment's execution through inputs and outputs provided by the user, as the IoT application to be debugged is running in the background of the same machine as the debugger. This is used for testing the equipment before deploying it in the production line [Manj].

Critical Manufacturing MES unifies the global enterprise, improving decision-making and increasing production efficiency. A comparison of modules/features provided by several market leading manufacturing execution systems can be seen in Table 2.2, where "x" means that the

MES system implements that module/feature. We can see that Critical Manufacturing MES solution is a quite complete solution through the variety of the modules it offers to manufacturers, delivering *"reliable access to detailed and timely operational information with full context and intelligence for fast, confident decisions and profitable action"* [Manc].

	Critical Manufacturing	Honeywell	iBASEt	Rockwell	LillyWorks
Data Collection	x	x	x	x	x
Dispatch Production Units	x		x		x
Document Control	x	x	x		x
Labor Management	x				x
Maintenance Management	x		x	x	x
Performance Management	x	x	x	x	x
Process Management	x	x	x	x	x
Product Genealogy	x	x	x		x
Quality Management	x	x	x	x	x
Resource Allocation	x	x		x	x
Scheduling	x	x		x	x
Lean Principles				x	
Real-Time Data	x	x	x	x	
Six Sigma			x		
Theory of Constrains	x				x
Automated Test & Inspection	x				
Web-Based Interface	x	x		x	
Historical Data	x	x	x	x	
Widely Available Platform	x	x		x	

Table 2.2: MES solutions comparison. "x" means that the manufacturing execution system implements that module/feature. A brief description of the features used for comparison can be found in Section 2.2.1, for the principal MES functionalities defined by MESA, and on Table 2.3, for the remaining (Adapted from [Sar03]).

The principal MES functionalities defined by MESA that were previously mentioned constituted the first eleven comparison measures used in Table 2.2. A brief description of the remaining eight features used for the MES solutions comparison can be found on Table 2.3.

Critical Manufacturing MES provides the backbone required to achieve the "smart factory" vision providing a way for companies to reach new levels of efficiency and innovation, in order to keep their business strong and competitive [Manh].

2.2.2.1 Architecture

To set up the environment necessary for the debug of a workflow in the "Connect IoT" MES module, it's necessary to define and relate the components presented in Table 2.4 in the Critical Manufacturing MES, which will be defined through the MES interface.

The Automation Driver, Controller, and Monitor processes will be run in a computer physically close to the physical device to debug. This three-component environment will be from now on called IoMT (Internet of Manufacturing Things) agent, for easier reference (see Fig. 2.7).

Feature	Description
Lean Principles	Implements the lean principles, meaning it aims to eliminate waste, the non-value-added components in any process, still providing customers with high quality products.
Real-time Data	Provides real-time information.
Six Sigma	Implements the six sigma principles, meaning it focus on developing and delivering near-perfect products and services, measuring and eliminating product defects.
Theory of Constrains	Implements a methodology for identifying the most important limiting factor (i.e. constraint) that stands in the way of achieving a goal and systematically improves that constraint until it is no longer the limiting factor.
Automated Test and Inspection	Connection to integrate inspection devices.
Web-Based Interface	Graphical user interfaces are internet-enabled.
Historical Data	The data that is acquired is stored in the MES database and mined for defects and performance analysis.
Widely Available Platform	Development platform is widely available.

Table 2.3: Features used for the MES solutions comparison (Adapted from [Sar03]).

The automation workflow graphs are defined in the Automation Controller workflow designer interface (Figure 2.6). These are composed of several tasks, linked through wires that connect outputs from one workflow task to the inputs of other tasks, enabling the passing of values between tasks. These links may have value converters associated, which may change the type of the variable (useful when the variable type doesn't match with what the input is expecting) or have some small business logic to apply to the variable before it gets to the next input. The workflow will be executed whenever a particular event occurs, activating it (e.g. upon initialization, when a certain device state changes or upon receiving an input from the previous task) and will be executed by the controller's workflow engine during runtime.

Component	Description
Automation Protocol	Definition of the communication protocol to be used between the Automation Driver and the physical device (machine), such as SECS/GEM, OPC, AMQP, MQTT, HTTPS, etc. The OPC UA protocol was chosen to be used for the testing of the developed prototype. It will be associated with the Automation Driver process.
Automation Driver	A low-level process that communicates physically with the device using the defined protocol, using it to interpret the messages received and report the events from the device to the Automation Controller associated with this driver. In the definition of this component, the protocol to be used, the properties/variables of the physical device we want to monitor, the events to be detected and the commands for the Automation Controller to execute will be designated.
Automation Controller	An event-driven process that executes workflows in response to events received from the Automation Driver or directly from the MES. It can communicate with more than one driver. The definition associates the Automation Driver(s) to the device/resource we want to monitor and defines the workflows to be executed in response to the received events. When a controller is initialized, it originates an Automation Controller Instance which will have a unique ID.
Automation Monitor	A process that determines which processes must be started or stopped, monitoring the health of these processes. It connects the Automation Controller and Driver(s), telling the driver processes where to find the controller it has been associated with so that they can begin communication.
Automation Manager	A process that will host one Automation Monitor and several Automation Controllers and Drivers as they were configured in the MES interface. It is responsible for spawning and controlling these processes.

Table 2.4: IoMT (Internet of Manufacturing Things) components.

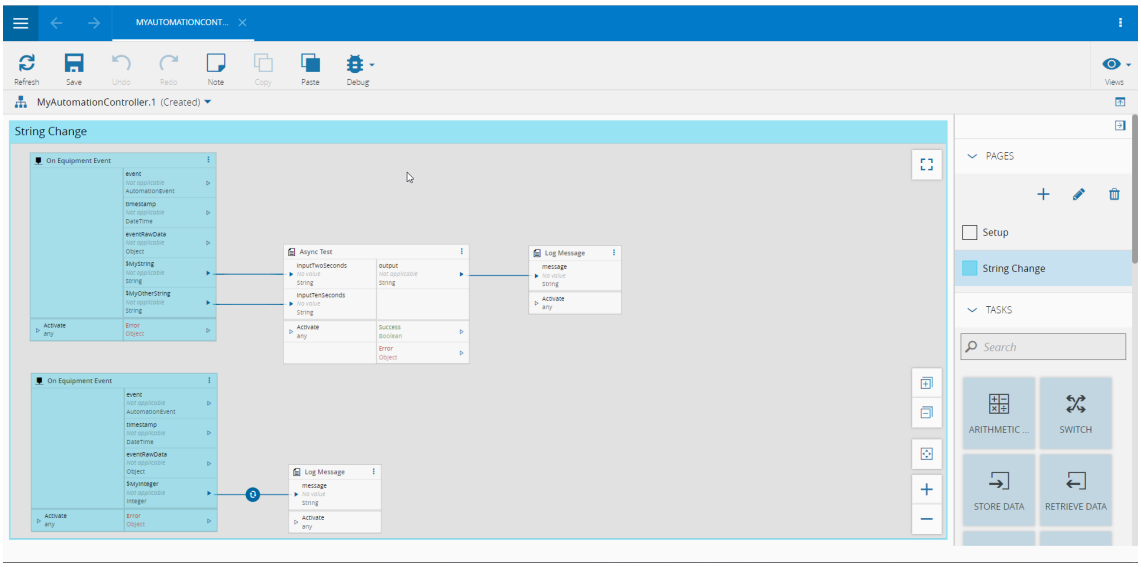

Figure 2.6: Automation Controller workflow designer interface.

2.2.3 Summary

Manufacturing Execution System is an information system that focuses on the digitization of shop-floor activities for the monitoring, documentation, and report of information regarding the transformation of raw materials into the desired final product in an integrated manner, resulting in the control and optimization of manufacturing production processes in near real-time [WWB18].

It aims to achieve and maintain a constant high performance in the highly competitive and rapidly changing manufacturing environment [Mani].

This type of systems focuses on the vertical integration of manufacturing processes with

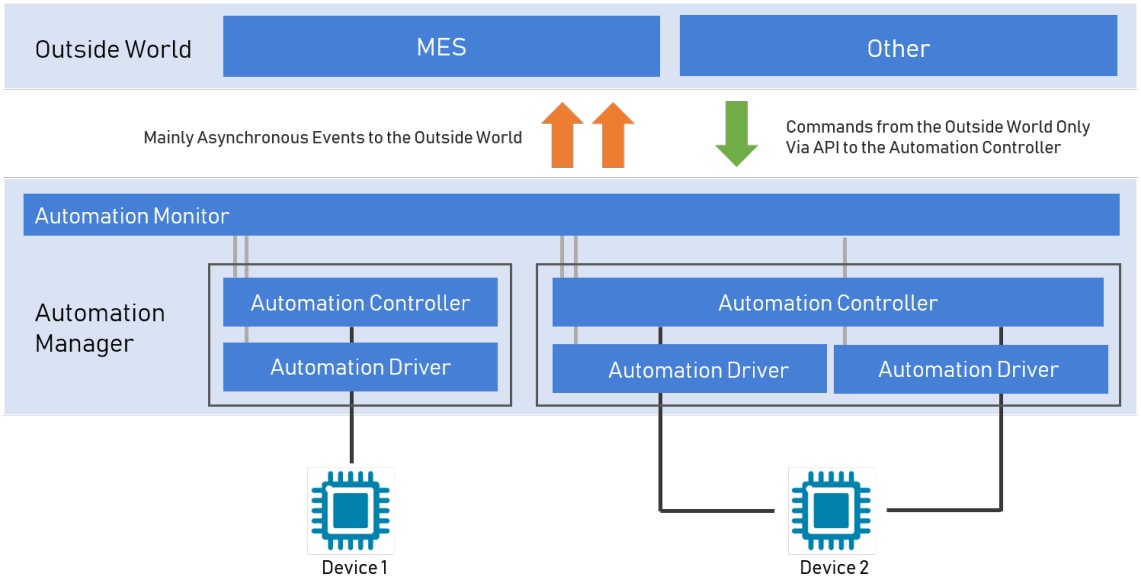

Figure 2.7: IoMT (Internet of Manufacturing Things) architecture.

business processes by bridging enterprise information systems (ERPs) with the shop-floor equipment [WWB18].

Critical Manufacturing has developed its own MES solution. This software platform has a deep set of modular, customizable and scalable applications which claims to provide manufacturers in complex industries with maximum agility, visibility, and reliability of the manufacturing process [Manf].

The recent advances in the integration of IIoT led to the development of a module implemented in Critical Manufacturing MES named "Connect IoT". This factory automation module allows the integration of IoT devices, machines, and interfaces, with the goal of dramatically reducing the time and effort required for this process. It also allows creating a graphical overview of the automation workflows being carried out by the machines in production and visually debug these through the simulation of the industrial equipment [Mand].

2.3 Software Development Life Cycle

The Software Development Life Cycle (SDLC) captures the (various) systematic approaches practiced for the development and maintenance of reliable, high-quality software systems [Ras14]. It is composed of a set of distinct (sometimes overlapping) work steps to be followed by software specialists, analysts and developers while planning, designing, implementing, testing, delivering and deploying software. Each work step is sequentially placed in order to use the outcome from the previously executed step. SDLC's main goal is of producing high-quality systems with an enhanced development process [KG15, WKYBWYF12].

Multiple software development models or methodologies, such as the *waterfall*, *incremental* and *agile* models, have hitherto been adopted, with varying degrees of success. Each has advantages and disadvantages, making of extreme importance to choose the one that better suits the type of project as it will impose a structure on the development of the software product [Ras14]. A systematic development process which is able to emphasize the understanding of the scope and complexity of the complete development process is essential for achieving a high degree of software integrity and robustness [WKYBWYF12].

The classic software life cycle model combines some version or derivative of the following sequential work phases for maximizing the chance of successful software delivery [WKYBWYF12] (Figure 2.8):

Requirement Gathering, Analysis, and Planning Involves a deep understanding of the project and gathering of the requirements to produce the final product. An estimation about the length of time each phase of development will take and what resources will be required will be performed, as well as the prediction of any problems that may arise during the software's life cycle [WKYBWYF12, GD15, KG15];

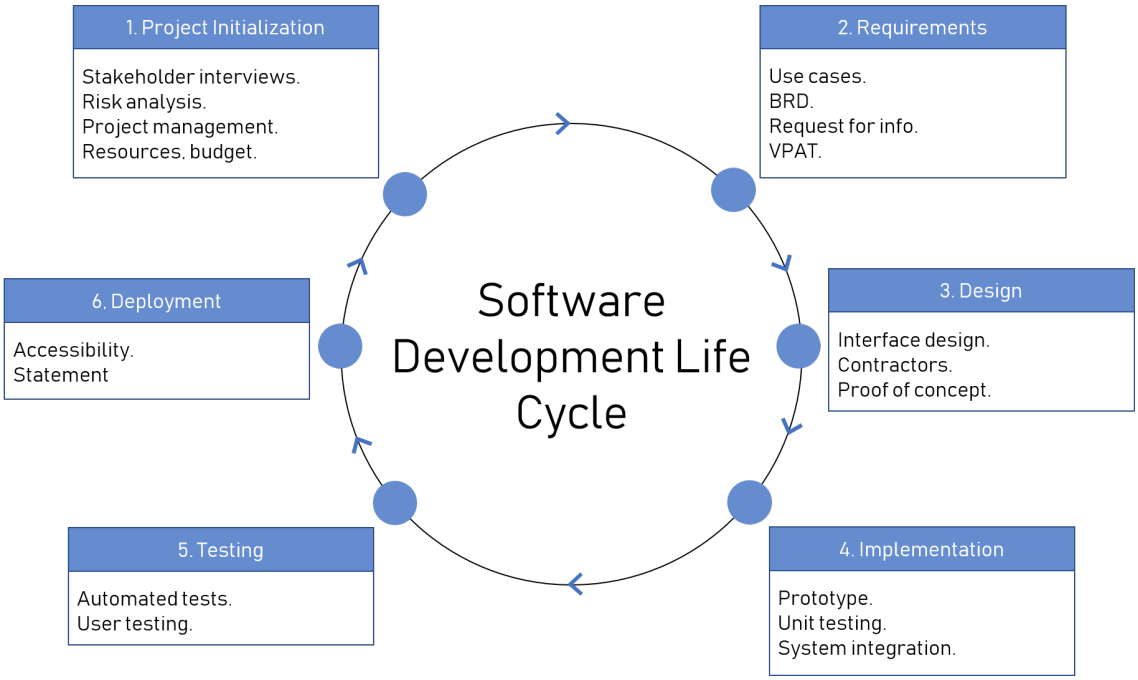

Figure 2.8: Software development life cycle phases (Source: [oM]).

Software Modeling and Design Design and architectural planning, producing a technical infrastructure of the software to be developed, shaped as diagrams and models. It has the primary goal of finding constraints and potential issues that the project may face as it evolves and provide a design/blueprint of the software system, following the requirements defined on the previous work step allowing the developers to start the implementation [WKYBWYF12, KG15];

Coding Development phase where the code is produced until all the previously defined goals relative to the software is attained. The work is divided into modules/units so that smaller tasks are distributed among the team. This is the work step that takes the longest in the software development life cycle and also the primary focus [WKYBWYF12, GD15, KG15];

Documentation On-going documentation is usually required and written as the work progresses, for future reference as well as for guiding further development [WKYBWYF12, KG15];

Testing Testing phase to identify defects and/or bugs in the developed software system. This work step is crucial and of extreme importance in the SDLC. It often overlaps with the development phase to ensure issues are addressed early on since the code developed needs to be tested properly to guarantee the security of the software created and to verify that it's solving the needs addressed and assembled during the requirements step [WKYBWYF12, GD15, KG15];

Deployment and Maintenance The deployment step comes after successfully testing and approving the release of the product. It consists of the deployment of the application built

on the client's environment. Software maintenance starts once the system is deployed and operational. Maintenance is used for future enhancements, new requirements that might be necessary to implement and fixes in the application code due to unexpected bugs or incorrect behavior [GD15, KG15].

The SDLC model selection and adoption process is crucial and maximizes the chance of a successful software delivery [WKYBWYF12].

2.3.1 Software Development Life Cycle For Industrial Systems

Smart and efficient manufacturing can be achieved with IIoT, connecting the shop-floor to production management [Azi19]. The increasing usage of physical entities equipped with sensors and actuators in the manufacturing industry, through the integration of IoT in factories, provides major feedback in real-time across a virtual network regarding the manufacturing process, enabling the fast detection and prevention of machinery failures [LFK+14]. Known as Programmable Logic Controllers (PLC), these computational entities are widely used in automation control and will autonomously perform many processes within cyber-physical systems [AA16].

PLCs are *"computer-based, solid-state, single processor devices"* that behave like an electric ladder diagram and are capable of controlling many types of industrial equipment and entire automated systems (illustrated in Figure 2.9) [AA16]. They are characterized by their cyclic data processing behavior which consists of reading all input values (provided by sensors), executing the PLC program with the received values and, when finalized, writing all output variables (which control the actuators), later restarting the cycle [VHFR+15].

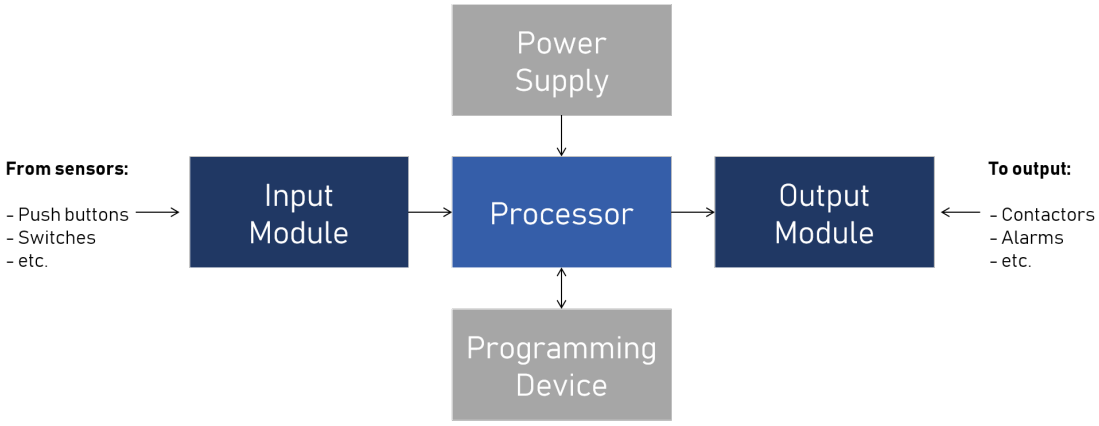

Figure 2.9: Analog input and output to a PLC (Adapted from: [WKYBWYF12]).

These controllers are highly efficient and reliable in applications which involve sequential control and synchronization of processes in the manufacturing industry. They are also cheaper than most other control elements [AA16], hence the reason why they are the standard industrial platform nowadays [VHFR+15]. These controllers are widely used in safety-critical processes, such as gas monitoring and ventilation control in mines, nuclear power plants, etc. However, as

the complexity of these controllers gets higher, the more difficult it is to ensure their reliability [AFMW17].

The implementation of logic and switching operations are the foundation for programming these devices, hence the reason why the term "logic" is applied. Input devices (containing sensors) and output devices (containing actuators) are controlled using logic operations and are connected to the PLC where the controller is monitoring the inputs and outputs according to the machine or process [AA16].

Diagrams, along with other graphical and model representations, have been playing a big role in software development since the 1940s, with the debut of modern digital computers. Initially used as paper-based aids to help programmers design and understand the structure of their programs, these diagrams started to be used directly in the design and coding of software, as a solution for the improvement of software development tools, due to the increasing complexity of hardware [Cox08].

A Visual Programming Language (VPL) can be defined as a language *"in which significant parts of the structure of a program are represented in a pictorial notation, which may include icons, connecting lines indicating relationships, motion, color, texture, shading, or any other non-textual device"*. By expressing the structure of programs and data pictorially, programmers can achieve a more concrete representation of these structures, making programs easier to build, debug, understand and reason about [Cox08].

PLCs have been, historically, programmed using VPLs, with schematic or ladder diagrams instead of usual computer languages [AA16]. This is done for abstracting the low-level concepts and details into high-level logic through the use of visual metaphors [BYF03]. Examples of these graphical languages are Ladder Diagrams (LD) (Figure 2.10), Function Block Diagrams (FBD), and Sequential Function Charts (SFC) [VHFR+15].

Using visual programming can be helpful for reductions in time and cost when developing software, as it has been stated by some researchers. It was also stated that improvements in productivity and reliability were achieved and were considerably noticeable. However, even though visual programming has several benefits, it also has some drawbacks when applied to general purposes (e.g. complex control structures and recursion). Visual notations are more effective when dealing directly with an application domain, which is the case of an IoT system [FTV02].

The SDLC model most commonly used for developing PLC-based control systems is the V-model (Figure 2.11), an extension of the waterfall model where validation, verification, and testing are prepared and usually performed in parallel with the requirement gathering, design, and implementation phases, respectively. It has the main advantage of including some validation before the development phase has begun. However, if errors are found during the testing phase, the test documents, along with requirement documents, will have to be updated, which can be quite expensive with the increase of the project's complexity [AFMW17].

IoT systems poses new challenges for software development beyond the traditional ones, due to the increase of complexity of developing software that targets distributed, heterogeneous and extremely dynamic systems [AKG+19].

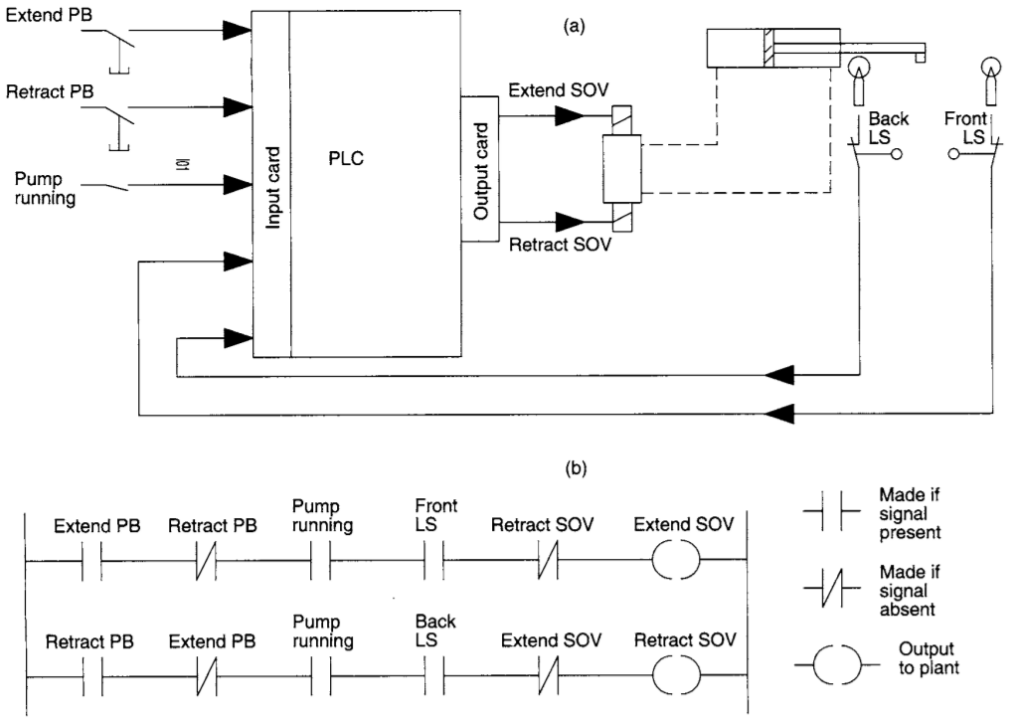

Figure 2.10: A simple PLC application: (a) a hydraulic cylinder controlled by a PLC; (b) the "Ladder Diagram" program used to control the cylinder (Source: [AA16]).

2.3.2 Software Maintenance

Software maintainability is *"the ease with which a software system can be modified to correct faults, improve performance or other attributes or adapt to a change of environment"* [61190]. It takes place once the software product has been delivered to the user [IYMD17] and can involve

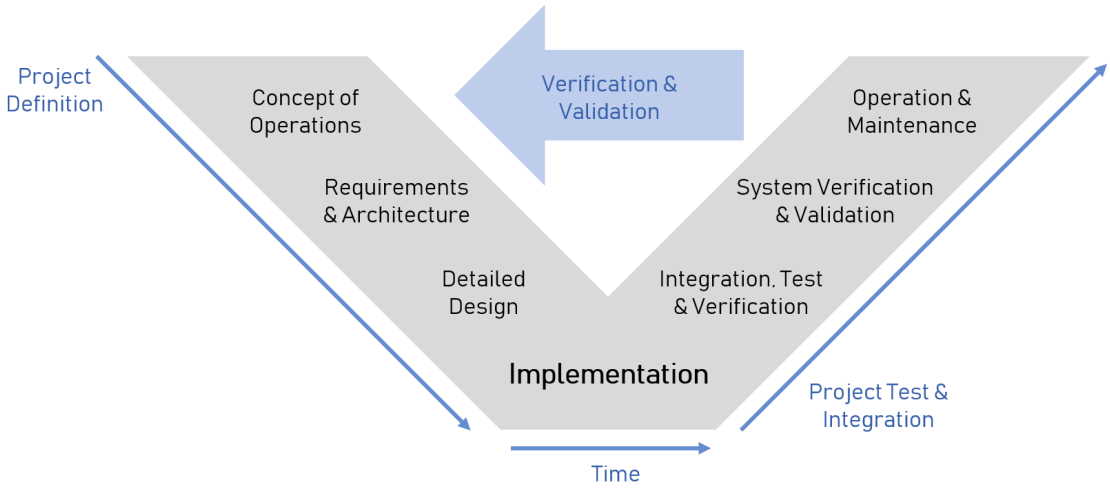

Figure 2.11: SDLC V-Model (Source: Wikipedia).

repair/modification of the software (bug fixes), implementation of new requirements (adding new components to the existing system), or adaptive maintenance of the environment where the software is operating (Figure 2.12) [VHFR+15].

Ensuring software quality has become a central issue because most organizations rely on software products to run their business operations efficiently and effectively, in order to remain competitive in business. Thus it is crucial to ensure the sustainable quality of a software product throughout its life cycle and, to ensure this, a good maintenance process is required [IYMD17].

The maintenance phase ensures that the delivered software product sustains a high level of quality and satisfies the client's requirements, aiming at adapting or perfecting the system towards this goal. This process is invoked both when there is a change in the software requirements and new features need to be implemented, or when failures are detected and need to be quickly fixed so that the software remains operational. Maintainability goes beyond corrective and preventive maintenance and is one of the software's quality factors, as a good maintenance process ensures a successful service in the long run [IYMD17, SM98, ALR00].

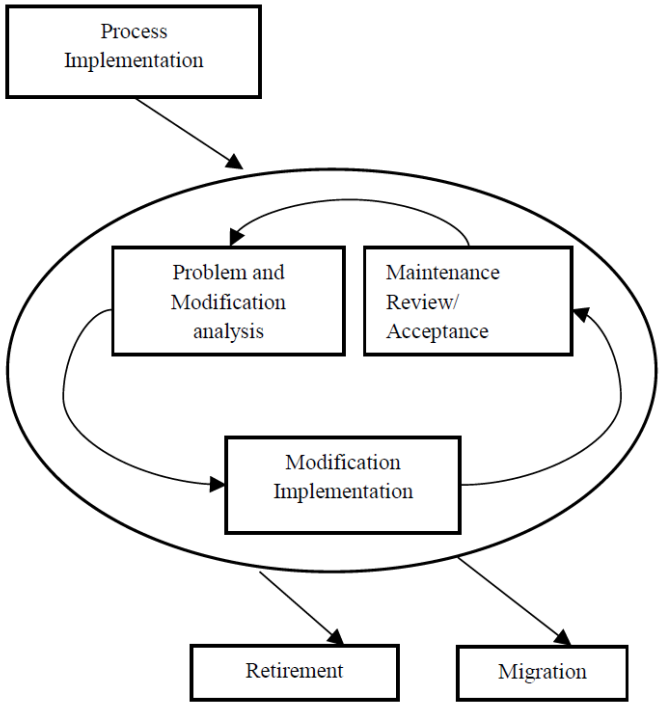

Figure 2.12: Example of the maintenance process. (Source: [IYMD17]).

This work step of SDLC is considered one of the most costly where the estimated cost is around 40% to 90% of the total budget given to software projects [Mom14], although previous studies have revealed that the need for intervention will decrease over time along the software's operation [IYMD17].

Based on Software Engineering Body of Knowledge (SWEBOK), maintenance types are comprised of corrective, adaptive, and perfective maintenance (Table 2.5). IEEE 14764 added a fourth

category of maintenance named preventive maintenance. Each of these maintenance types apply different processes and are responsible for activities that focus on keeping the system operational and valuable for the company [BF14].

Corrective and preventive maintenance are both used for the removal of faults during the operational life of a software system. Corrective maintenance's goal is to remove faults which have been detected and reported due to errors that happened as a result, whist preventive maintenance is aimed at uncovering and removing faults before they generate errors that may disturb the system's execution [ALR00].

	Correction	**Enhancement**
Proactive	Preventive	Perfective
Reactive	Corrective	Adaptive

Table 2.5: Software maintenance categories.

2.3.2.1 Software Dependability

Dependability is a measure of the system's attributes that are related to the degree of confidence one has that the software system will operate as expected. It aims at keeping the system operating as it was stated in the requirements, without failures (as shown in Figure 2.13), and well as keeping the maintainability of a system [ALR00].

It encompasses the following attributes [ALR00]:

Availability Readiness for correct service (software execution as stated in the software requirements);

Reliability Continuity of correct service (with no occurrence of failures);

Safety Absence of catastrophic consequences on the user(s) and in the environment;

Confidentiality Absence of unauthorized disclosure of information;

Integrity Absence of improper system state alterations;

Maintainability Ability to undergo repairs for service restoration (due to software failures) and modifications of features.

For a system to be highly dependable, the combined application of the following set techniques is required [ALR00]:

Fault Prevention Prevention of the occurrence and/or introduction of faults, usually employed during the design of the software;

Fault Tolerance Delivery of correct service in the presence of faults, intended to preserve the delivery of the intended service even in the presence of active faults, implemented through the detection of errors and subsequent system recovery;

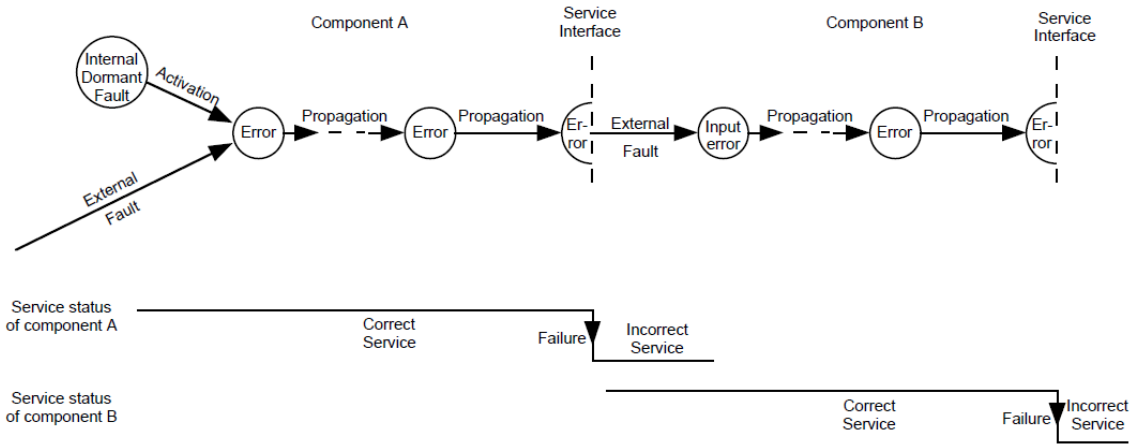

Figure 2.13: Example of error propagation (Source: [ALR00]).

Fault Removal Reduction of the number or severity of faults, employed during development and the operational life of a system, through verification of errors that are encountered and diagnose the causes that are leading to the system's fault though debugging, leading to the software's fault removal;

Fault Forecasting Estimation of the present number, the future incidence, and the likely consequences of faults.

The evolution of the dependability measurement over the software's life cycle depends on the system's stability, growth, and decrease of failure occurrences over time. These aspects are expected to be attained with the maintainability performed during the software's operation after deployment [ALR00].

2.3.3 Software Maintenance in Industrial Systems

The inherent nature of IoT systems is characterized by being highly heterogeneous, high-dimensional, dynamic and nonlinear, cross-domain and ultra-large-scale. These characteristics, amongst others, pose new challenges on how to design, develop, and maintain industrial IoT systems [AKG+19].

Factories require consistent professional maintenance of automated production systems. The complexity of these systems, both hardware and software, has been increasing, and they often require a life-span of more than 10 years since the software is deployed in the factory. For this reason, maintaining the developed software is strongly required to ensure the software's quality and reliability [VHFR+15].

Special requirements on the development and maintenance process are also needed, due to the need for rapid adjustment of production capacity and functionality, in response to new market conditions. With a fast changing market and customers demanding a higher level of product customization, industrial software must be constantly updated and re-configured to cope with changes to production processes or the introduction of new products. This imposes pressure over the need for software development and maintainability, demanding faster development processes while maintaining software quality [MUK00].

The analysis and management of the information gathered by the sensors and actuators integrated into the manufacturing machinery can also benefit the maintenance process, taking advantage of the implementation of cyber-physical systems (CPS) on factories. The information captured will allow monitoring the machine's health condition and correctness of operation, controlling if the solution is operating according to the client's requirements and detecting failures if they happen, enabling life cycle observation. The analysis of the captured information will allow maintaining the optimal performance that guarantees the best product quality [MBRB17, VHFR+15].

2.3.4 Summary

The Software Development Life Cycle (SDLC) captures the (various) systematic approaches practiced for the development and maintenance of reliable, high-quality software systems [Ras14]. It is composed of a set of distinct (sometimes overlapping) work steps for planning, designing, implementing, testing, delivering and deploying software. The selection and adoption of the right SDLC model for a project is crucial and maximizes the change of a successful software delivery [WKYBWYF12].

Programmable Logic Controllers are *"computer-based, solid-state, single processor devices"* capable of controlling many types of industrial equipment and entire automated systems [AA16]. They receive inputs from sensors and provide outputs for controlling the actuators. PLCs have been, historically, programmed using visual notations, which can often become quite complex [VHFR+15].

The SDLC model most commonly used in manufacturing software, suitable for developing PLC-based control systems, is the V-model [AFMW17].

Software maintenance is the step of SDLC that ensures that the software product delivered to the user still maintains a high level of quality and satisfies the client's requirements, and can either involve repair or modification actions, implementation of new requirements or adaptive maintenance to the environment [IYMD17].

In industrial systems, software maintainability is especially important. With a fast changing market and customers demanding a higher level of product customization, industrial software must be constantly updated and re-configured, imposing pressure over the need for software development and maintainability, demanding faster development processes while maintaining the software's quality [MUK00].

2.4 Debugging

Software debugging is the process of identifying errors or defects in software or in a computer system and solving them so that the program works correctly according to specification [SD17]. Any step of a program that performs unexpectedly is termed to be a fault. Debugging is an essential part of the software engineering process, being an arduous, time-consuming, costly task [SBAM17].

Man-made software should not be considered to be reliable, safe, secure, or always available unless it is thoroughly tested and verified, due to the unavoidable presence or occurrence of faults [ALR00]. Testing these systems is an important part of the software development life cycle. After, or even during, the development phase, testing and debugging should be taken seriously and given high priority. Without these steps, high-quality, reliable software cannot be provided in today's fast track based world [SBAM17].

A debugger is a tool that allows debugging a program, to see what is going on *inside* a program while it executes or what it was doing at the moment it crashed. Any application software will inevitably contain bugs during the development cycle. The distance from the defect causes to the failure may be long in both time and space so developers require a deep understanding of the software system and its environment to be able to follow the infection chain back to its root cause. They require access to powerful debugging tools for the correction of these software flaws, allowing them to work more efficiently and to better dig into the detailed operation of their application. Even though modern debuggers can aid software developers in gathering information about the system, they cannot relieve them of the selection of relevant information and the reasoning to find the software's issue [ALR00, PSTH17, LXWY09].

Software defects can have varying degrees of severity. This severity depends on the failure domain, controllability, and consistency of the failures encountered and the consequences in the environment. For example, a bug that formats output incorrectly is much less severe than one that corrupts critical data. The time required to solve a bug is also directly related to the complexity of the error. Complex defects like race conditions in multithreaded programs may take a substantial amount of time and effort to correct [ALR00].

As the total number of faults encountered increases, so does the cost of software development. Something that helps to decrease this cost is to detect and locate faults in the software in the early stages of development. This effort reduces the costs of maintenance and accelerates the development of the software (as less time will be wasted on testing) [SBAM17].

Different types of faults can be encountered such as system faults, business logic faults, functional faults, graphical user interface faults, behavior faults, security faults, etc. Software processing levels have different debugging approaches because, for example, parallel programs produce different types of errors than multithreaded programs, due to the non-deterministic nature of threads. Debugging is used in the fault removal technique used to improve the software system's dependability and, whatever debugging technique is chosen as the most suitable considering the nature of the software, aggregates the following work phases [SD17, SBAM17]:

Fault Detection Identification of erratic or unforeseen behavior;

Fault Localization Identification of the origin(s) of the problem;

Repair Correction of the problem by either replacing or modifying the existing code(s) or part(s) of the program identified in the previous phase.

Handling software faults through testing and debugging plays a vital role in the software engineering process as it helps to provide high-quality reliable software [SBAM17].

2.4.1 Remote Debugging

Traditionally, software applications are debugged in an environment in which the debugger is executing on the same computing device as the application being debugged. In some cases, resources such as memory, processing power, and network are consumed by the installation and execution of a development and debugging environment (e.g. via an integrated development environment (IDE)). This is a critical factor because such computing devices may have limited resources, for example, storage, processing, and communication. In these cases, using traditional debugging techniques may not be an option, and a remote debugger seems to be a reasonable solution [LJ15].

In the remote debugging, or cross-debugging, technique, the debugger runs on a host machine, while the program to be debugged is running on a specific hardware platform of a target machine (Figure 2.14). The application is launched using a remote debug module executing on the host machine, and it communicates with the target machine via a communication channel that can be set through a serial port, parallel port or network card interface, taking control of this machine to run the program [LXWY09, LJ15].

During a debugging session, debug commands are sent from the host machine to the target program and these commands are employed for debugging the application on the remote host device. The commands may include the launch of the application to be debugged, setting breakpoints for pausing the execution of the application at a certain execution point, resuming execution following a breakpoint, stepping through instructions of the application or terminating it. Debug commands may also *ask* for information regarding the state of the application such as the values stored in variables, parameters, registers and program counters of the application, stack traces for the call stack of the application, dumps of active memory following a crash or failure and such [LJ15]. This debugging technique is especially useful for debugging embedded software, and it requires cooperation between the host and target machines [WDDZ11].

2.4.2 Debugging on Industrial Systems

Nowadays, companies are competing fiercely to provide high-quality software at the lowest possible cost. Software maintenance plays a crucial role since it preserves the software quality after deployment, implementing new features and fixing errors that show up during execution. Debugging tools are used in this phase of the software development life cycle to help identify where the software faults are and what is the condition that leads to them [MZ16].

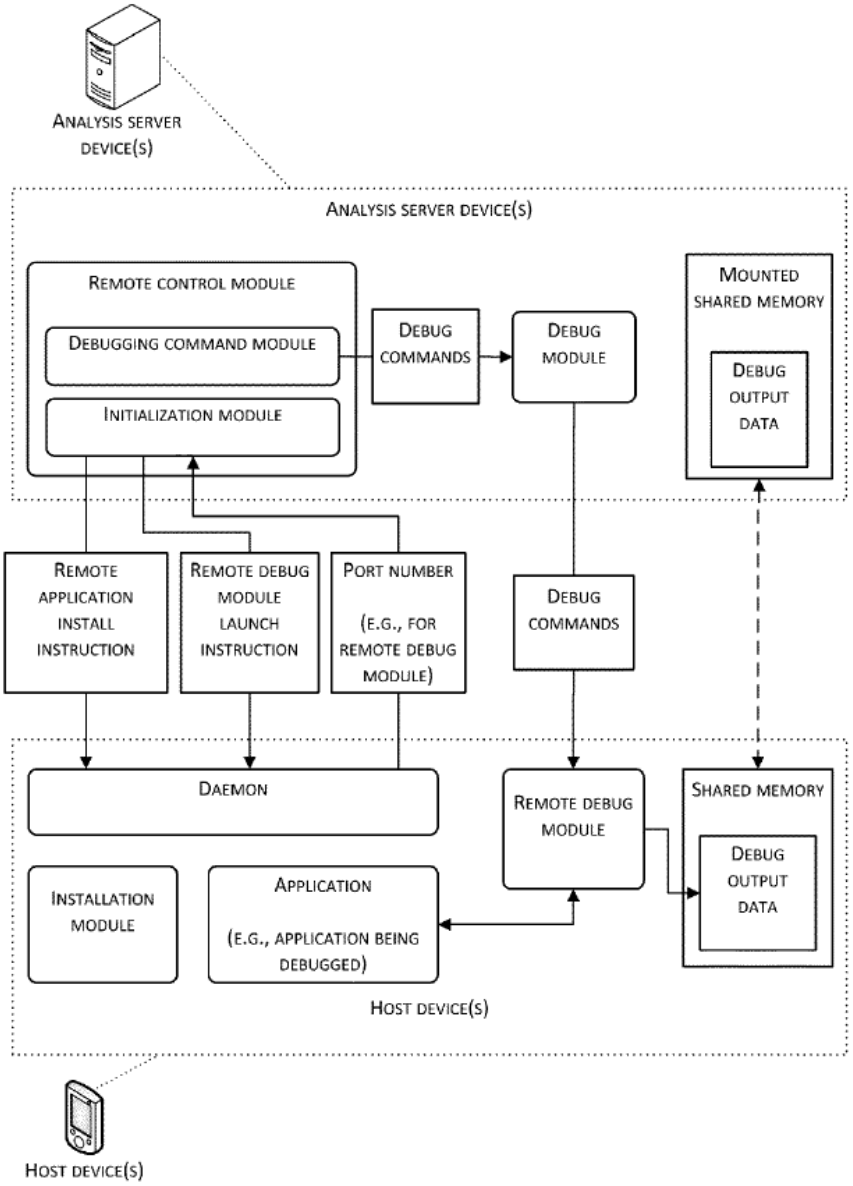

Figure 2.14: Example of an environment for remote debugging of an application (Adapted from: [LJ15]).

Additionally, in industrial systems, the increasing need for production efficiency and flexibility require active and real-time maintenance from skilled technicians in order to reduce machine downtime. For expensive machines, usually, often downtime is more expensive than the actual repair in terms of lost production resource. Thus, real-time monitoring, advanced maintenance, and debugging systems play a significant role in solving the problem remotely [MZ16].

Cyber-physical systems are similar to traditional distributed embedded systems, consisting of several interconnected devices with limited resource constraints. However, its main goal is to remain responsive to environmental changes and network commands. They *"are distributed applications that track, observe and analyze large collections of data from computerized entities"*. Debugging these distributed systems is hard because developers have to deal with the non-

determinism of concurrent processes, the time-sensitive nature of applications and partial failures that may occur. This makes the debugging task arduous since an error detected in one execution might not manifest itself in the next one [MBC+17].

Bugs in CPS systems are hard to reproduce and thus to fix, and for this reason, remote debugging techniques are helpful in the maintenance process because the debugger is connected to the device when an exception or crash occurs. Capturing information from these devices in real-time helps to identify and track where the causes of failure are situated [MBC+17].

The use of visual metaphors as abstraction mechanisms have been around the manufacturing and automation industry for a long time [AGB93]. In the manufacturing industrial environment, graphical input support and display interfaces can be used for better visualizing the control of the workflow of a specific machine. A typical workflow process is composed of a series of tasks and events, the order in which they must occur and the script to be executed as a result of each event [SP07].

Developing a workflow can be a complex process involving many system components. For this reason, running the workflow alongside a debugger in order to identify and fix issues is crucial [SBP+06].

Conventional debuggers are inadequate for the debug of workflows. The ideal environment, considering the context of a manufacturing system, is the availability for remote debug of workflows running on any shop-floor machine in production, through a user interface that allows a certain level of abstraction regarding the control of automation functions, either it is for controlling an industrial process programmable controller (PLC) or for programming the motion controller of a processing or production machine [SP07].

2.4.3 Summary

Software debugging is the process of identifying errors or defects in software or a computer system and solving them so that the program works correctly accordingly to specification [SD17]. Debugging is an essential part of the software engineering process, being an arduous, time-consuming, costly task. It is crucial since no software can be considered reliable unless it is thoroughly tested and verified, in order to make sure that there are no system anomalies [SBAM17, ALR00].

Debuggers allow the debugging of a program, to see what is going on *inside* a program while it executes and what the application was doing at the moment it crashed [LXWY09].

Nowadays, companies are competing fiercely to provide high-quality software at the lowest possible cost. Software maintenance plays a crucial role since it preserves the software quality after deployment. Debugging tools are used in this phase of the software development life cycle to help identify where the software faults are and what are the conditions that lead to them [MZ16].

Traditionally, software applications are debugged in an environment in which the debugger is executing on the same computing device as the application being debugged. Such computing devices may have limited resources, e.g. storage, processing, and communication, as it is the case of the ones commonly used in industrial systems. The debug of these systems has to be done through remote debugging. In this debugging technique, the debugger runs on a host machine,

while the program to be debugged is running on a specific hardware platform of a target machine. The application is launched using a remote debug module executing on the host machine, and communicates with the target machine via a communication channel, using it to send the debug commands [LXWY09, LJ15].

The use of visual metaphors as abstraction mechanisms have been around the manufacturing and automation industry for a long time [AGB93]. They can be used for better visualizing the control of the workflow of the tasks to be executed by a specific machine [SP07].

Developing a workflow can be a complex process involving many system components. For this reason, running the workflow alongside a debugger in order to identify and fix issues is crucial [SBP+06]. In the ideal environment, the availability for remote debug of workflows running on any shop-floor machine in production, through a user interface that allows a certain level of abstraction regarding the control of automation functions, should be provided [SP07].

2.5 Conclusions

From the background explored, several conclusions can be made.

Industry 4.0 is revolutionizing the way factories compete in the market, aiming to achieve a higher level of operational efficiency and productivity by augmenting the level of automatization in factories.

The integration of IoT in the manufacturing process was the big step into the current industrial revolution, with the integration of embedded sensors, actuators, processors, and transceivers. Manufacturing execution systems focus on the digitalization of shop-floor activities with the collection, analysis, and exchange of the information received in real-time from these devices, allowing better control and optimization of manufacturing production processes.

An adaptation of the software development life cycle for industries has certainly been a determining factor on the achievement of smart and efficient manufacturing. Because most organizations rely on software products to run their business operations, good maintenance has to be performed regularly and in real-time to ensure the sustainable quality of the software product through its life cycle.

Industrial software requires the need to be constantly updated and re-configured, imposing pressure over the need for maintenance of the software deployed in the industrial systems. The maintenance process of these systems takes advantage of the implementation of CPS on factories to monitor the machine's health condition and correctness of operation in the production line, in real-time, in order to maintain optimal performance of the manufacturing system at all times. For an expensive machine often downtime is more expensive than the actual repair in terms of lost production resource. This makes it even more important to detect execution failures as soon as they happen through real-time monitoring, advanced maintenance, and debugging systems.

Failures in CPS systems, are hard to reproduce due to the non-determinism of concurrent processes, the time-sensitive nature of applications and partial failures that may occur. Remote

debugging techniques are helpful in the maintenance process of these systems because the debugger is connected to the device when an exception or crash occurs, capturing information from these devices in real-time.

IIoT systems are usually programmed by the means of visual workflows, performing an abstraction of the different tasks to be executed by the equipment, representing the workflow of tasks in a simplistic way. Due to the nature of these systems, it's important to run the workflow alongside a debugger in order to identify and fix any occurring issues, helping the maintenance process of IIoT systems.

Chapter 3

Related Work

Contents

This Chapter aims to present some of the current approaches on debugging, remote debugging, debugging of workflows and remote debugging protocols, while simultaneously identifying the shortcomings of the current approaches *w.r.t.* remote debugging of IIoT workflows.

3.1 Remote Debugging

Remote debugging, or cross-debugging, is the process of debugging a program running on a different system from the debugger. The debugger runs on a host machine, whilst the program to be debugged is running on a specific hardware platform of a target machine. The host machine executes the debugger, which sets a communication channel for communicating with the target machine, controlling the program's execution and sending the debug commands [LXWY09, LJ15].

In this Section, we will look at some of the existing remote debugging solutions.

3.1.1 JPDA/JVM

Java Platform Debugger Architecture (JPDA) [Orab] is Java's debugging framework stack consisting of a mirror interface (JDI), a communication protocol (JDWP) and the debugging support running on the target virtual machine to debug (JVM TI). It splits the debugging process into the program which is being debugged and the user interface of the debugger application (JDI). The

debuggee application is running in the *back-end* while JDI runs in the *front-end*. They communicate using the JDWP protocol through a communication channel set between the two processes, thus supporting the remote debug process by running the debug process (running in the virtual machine) in a different machine from the debugger.

This framework is already included in IDEAs like IntelliJ IDEA, IBM Eclipse, Sun NetBeans, and many others [PBF+15].

3.1.2 TOD

TOD [PTP07] is an omniscient debugger solution that enables navigation backward in time within a program execution trace. It records the events that occur during the execution of the program to be debugged and lets the user conveniently navigate through the obtained execution trace and evaluate what may have possibly gone wrong during the execution.

Even though omniscient debugging isn't within the scope of this document, this particular solution relates through the event-driven architecture that TOD implements. It uses object-oriented instrumentation events for the organization and storing of execution traces, like the ones used to order the executing program to halt and return the control to the debugging environment through a breakpoint within the debugging of an application, or to write in a variable of the executing program [PBF+15, PTP07].

3.1.3 Visual Studio Remote Debugger

The Visual Studio Remote Debugger solution [Docb] provides, other than the regular debugging environment (which allows to *"break the execution of a program to examine the code, examine and edit variables, view registers, see the instructions created from the source code, and view the memory space used by the application"*), a snapshot debugger which targets live ASP.NET apps in Azure App Service. It takes a snapshot of the in-production apps when the executing code reaches an established point, through *snappoints* and *logpoints*, capturing the state of the program's execution at that particular moment. It allows seeing what went wrong with the executing application without impacting the traffic of said application, dramatically reducing the time it takes to solve issues that occur in production environments. Both this functionality and the regular debugger support debugging remote environments [Doca].

The debug of software deployed on a different computer, pre-proposes a dedicated debugging deployment. On the remote device or server where the application to debug is, rather than the Visual Studio machine, it's required to download and install the remote tools provided by Microsoft [Docc].

3.1.4 GDB

GDB (also known as GNU Debugger) [GDBb] is a portable debugger that offers extensive facilities for tracing and altering the execution of applications. These applications might be executing on the same machine as GDB (native), on another machine (remote), or on a simulator.

For the remote debugging functionalities, a dedicated process named gdb-server is used on the target machine in order to attach the running processes [PBF+15]. GDB uses a generic serial protocol which can be used with remote stubs (the code that runs on the remote system) to communicate with GDB, using serial communication or a TCP/IP connection [Fou].

3.1.5 Rivet

Rivet [Mic12] is an experimental framework solution developed by Microsoft which provides browser-agnostic remote debugging of client-side web applications. It allows developers to *"inspect and modify the state of live web pages that are running inside unmodified end-user web browsers"*, allowing to explore real application bugs in the context of the actual machine in which those bugs occur. The communication between the client and the server side is made using standard HTTP requests.

3.1.6 Vorlon

Volron [Volb] is an *"open source, extensible, platform-agnostic tool for remotely debugging and testing JavaScript applications"*. This framework can be used on both Web and Node.js applications (with some restrictions for Node.js, since it does not provide DOM elements). It allows to remotely load, inspect, test and debug JavaScript code running on any device with a web browser.

It creates a small web server that communicates with the remote devices using standard HTTP requests and hosts a visual dashboard to present the information to the user. It shows the list of connected available remote clients and a view to allow the user to go through the client's DOM elements and modify them in real-time. After the Vorlon script is injected into the client's application, it is possible to debug the deployed application remotely [Vola].

3.1.7 ELIoT

ELIoT [Siv14] (ErLang for the Internet of Things) is a development platform for smart devices connected through the network that *"provides an abstraction over the hardware on which the applications will be executed"*. It targets IoT smart-home applications and provides IoT-specific inter-process communication functionalities, supporting parallel and distributed programming and thus providing the ability to be used in naturally distributed systems like CPS. The main goal is to allow programmers to write concise code and easy to maintain, debug and test after deployment [SMC16].

It grants remote communication with the applications running on the IoT devices, having the ability to query sensors or sending a command to an actuator using CoAP (Constrained Application Protocol) services [Siv14, SMC16].

3.1.8 SLDSharp

SLDSharp [TI19] is a debugger for iteractive and/or real-time programs that keeps track of both the currently executing statement in a program and the changes in values of the expressions of interest, visualizing them in real-time. It allows the programmer to dig into the possible causes of a bug without having to suspend the program's execution or check a log file. The targets of this debugging method are functionality bugs that cause logical errors in the repeated execution process in the flow, which receives input events to obtain computation results, thus originating outputs.

The programmer selects the granularity of the real-time visualization of the executed parts of the program and the sections and conditions for its visualization, observing the values of the expressions of interest whilst debugging the program. To implement the debugger, it embeds code fragments into the program's original code to obtain the information needed for debugging [TI19].

3.1.9 Summary

Table 3.1 displays a brief comparison between the different remote debugging solutions presented in this Section, in regards to the features they offer and that are of interest for the context of this dissertation. "x" means that the solution supports the feature mentioned in that column.

Solution	Remote debugging?	Reverse debugging?	Snapshot debugging?	Web oriented?	IoT oriented?
JPDA	x				
TOD		x			
VS	x	x	x	x	
GDB	x	x			
Rivet	x			x	
Vorlon	x			x	
ELIoT	x				x
SLDSharp	x		x		

Table 3.1: Comparison between the different remote debugging solutions presented. "x" means that the solution supports that feature.

3.2 Workflow Debugging

Workflow management systems have surfaced as a solution for addressing the diversified nature of data and ease the execution of business processes, providing a better representation of the process tasks, relevant resources, and documentation of the execution process. These systems are *"widely used in real-world applications within a wide scope of subjects including data mining, scientific computing, and concurrent programming"*, where all referred applications consist of many sequential subroutines (which in some cases some can be aggregated), executed one after the other, each represented by a block for simplification [RWM+18].

In complex heavily-distributed scenarios, these subroutines may even be distributed across several computing nodes. Such tasks require that many subroutines are chained together in a predetermined sequence, forming a more complex workflow, and it's often difficult to monitor the execution of these tasks in real-time [RWM+18]. In this Section, we will look at some of the existing workflow debugging solutions.

3.2.1 Real-Time Workflow Monitor

RTWM [RWM+18] (Real-Time Workflow Monitor) provides *"a cloud service and a client library to monitor complex workflow systems in real-time"*. It claims to provide the progress of the execution of each task of the workflow in real-time, monitoring the execution order and fault occurrences of the system remotely, presenting it asynchronously through a user interface. The interface provides an infrastructure for setting up the workflow system, monitoring performance and debugging a chained set of tasks.

A web server runs the main application for debugging and receives events from the computing units through WebSocket connections, then providing visualization for monitoring on the web app. The UI Layer uses HTTP and WebSocket connections for communication with the device's lower layers in real-time. The initial configuration messages are communicated through an HTTP REST API and a WebSocket connectivity is set for the communication of events from the computing units. Additionally, the notifications of the occurrence of these events are pushed through the WebSocket handler to the client applications that are actively monitoring a workflow system [RWM+18].

3.2.2 Stampede

Stampede [GDS+11] (Synthesized Tools for Archiving, Monitoring Performance and Enhanced Debugging) *"intends to apply an offline workflow log analysis capability to address reliability and performance problems for large, complex scientific workflows"*. It works as an execution information capture and analysis system, streaming and storing information about the performance of workflows in real-time on a log file written during execution. It also supports distributed environments.

The components of this tool's architecture can be divided into a workflow execution engine, log collection components, and archival and analysis components. During the execution of a workflow, it writes a log file in near real-time containing the status of each job, inputs, and outputs of tasks, and the pre and post-execute scripts [GDS+11].

Higher-level analyzing tools can mine the generated logs in real time to determine current status, predict failures, and detect anomalous performance [RWM+18].

3.2.3 Node-RED

Node-RED [NRb] is a programming tool for creating flow-based programs that connect hardware devices, APIs and online services. It provides a browser-based graphical editor for building the

workflows, wiring the nodes that represent each task, each with a well-defined purpose that receives data as input, runs a function and passes the data onto the next node as it is defined in the network's flow.

These flow-based programs can be deployed to the runtime in a single click and functions programmable in JavaScript can be easily tied to each node of the flow [NRa].

This tool can be run locally, on a physical device or in the cloud, allowing to visually represent the flow of tasks with an abstraction level that provides the user with the ability to break down a problem into easier steps and look at what each produces, in order to understand what it is doing and detect any failure that may be happening on the program's execution without having to understand the individual lines of code within each node [NRb, NRa].

3.2.4 Zenodys

The Zenodys Platform [Zen] has a workflow builder that allows users to create their own workflows by wiring elements together, with a powerful browser drag-and-drop interface, in order to build any kind of application backend or decentralized application. It simplifies the creation of data management scenarios and to apply complex operations like AI, machine learning, demand response, predictive maintenance, etc. The workflow is executed in the computing engine developed by Zenodys and stored locally, remotely or in a decentralized system.

Then, the UI builder allows building a custom user interface directly inside Zenodys ecosystem, allowing the visualization, reporting, action triggering and other business and application functionalities. It allows visually debugging the created workflows and remote operations on the physical device the workflow is connected to (including remote debugging)[Zen].

3.2.5 Summary

Table 3.2 displays a brief comparison between the different workflow debugging solutions presented in this Section, in regards to the features they offer and that are of interest for the context of this dissertation. "x" means that the solution supports the feature mentioned in that column.

Solution	Workflow builder?	Remote debugging?	Web oriented?	IoT integration?
RTWM	x	x	x	
Stampede	x	x	x	
Node-RED	x		x	
Zenodys	x	x	x	x

Table 3.2: Comparison between the different workflow debugging solutions presented. "x" means that the solution supports that feature.

3.3 Remote Debugging Protocols

In this Section, we will look at some of the existing remote debugging protocols used for the communication between entities involved in a remote debug session, with some relevance to this project.

3.3.1 GDB Remote Serial Protocol

The GDB RSP [GDBa] (Remote Serial Protocol) is a high-level protocol that allows GDB to remotely connect through a serial port. On the target, it must be linked to the *debugging stub* file that implements the GDB remote serial protocol. After the communication setup process is finalized, the debugger can use the usual commands to examine and change data and to stop or continue the remote program's execution. This protocol supports a wide range of connection types and the different command types are distinguished by the message's first character.

3.3.2 Java Debug Wire Protocol

The JDWP [Oraa] (Java Debug Wire Protocol) is the protocol used for communication between a debugger and the JVM where the program is running. It does not specify the transportation of the messages, only details the format and the layout, and it accepts any transport mechanism that is suitable for the target debugger/target VM combination through a simple API.

In JDWP, when the transportation connection between the two machines is established, a handshake must occur between the two sides before any packets are sent. The packets of the messages exchanged are defined as *command* or *reply* packets. The first can be used by the VM to request information about the application's internal variables, to control the program's execution, or by the running application to notify the debugger of some event such as the reach of a breakpoint or an exception. Reply packets are sent in response to a command packet, providing information regarding the success or failure of the command to what it is replying, carrying some data if requested [Oraa].

3.3.3 JSON/HTTP Communication Protocol

A JSON/HTTP based communication protocol was designed by F. Pereira and L. Gomes [PG18] to allow CPS systems to support reliable communication between components of naturally distributed applications. It encompasses remote debugging and monitoring functionalities for the detection and quick resolution of system failures that lead to the incorrect behavior of applications, supporting step-by-step execution and breakpoint definition capabilities. The developed protocol employs publisher-subscriber and client-server patterns for establishing the interconnection between the remote components.

3.4 Conclusions

This Chapter presented some of the currently available solutions for debugging, remote debugging, debugging of workflows and remote debugging protocols. The solutions were here introduced as a result of an extensive search that was made using the standard literature platforms as well as commercial platforms since the subjects of this search were more practical rather than theoretical and not many solutions were documented in the platforms first mentioned.

Through an analysis of the solutions encountered, some alternatives for the debug of workflows and for remote debugging are offered, but none of them seems complete as a whole for these two combined and contextualized in the manufacturing industry. The maintenance process for the manufacturing production line should be aided with remote control and monitoring of the machines in execution in the shop-floor through a network and in real-time, enabling the detection of failures in the equipment as soon as they happen, thus avoiding unwanted monetary and resource losses. It should support the use of visual metaphors as an abstraction mechanism for the equipment's execution tasks for easier understanding of the cause that triggered the system failure since visual programming languages are a common way of simplifying the software's execution in manufacturing systems.

The communication protocols that were examined also don't suit the needs of the *proof-of-concept* to be developed, therefore requiring the design and implementation of a suitable protocol to connect the visualization of the workflow to be monitored to the physical device where it is running.

Chapter 4

Problem Statement

Contents

The goal of this Chapter is to describe what are the current challenges of debugging IIoT systems, regarding the current solutions for IoT remote debugging and workflow debugging presented in the previous Chapter. It presents the proposed solution for these challenges as well as the context in which it was implemented. Then, the relevant features of the solution are presented, along with the means for evaluating the solution's success.

4.1 Current Issues

Current solutions, as presented in Chapter 3, offer some alternatives for the debug of workflows and for remote debugging, but none of them seems complete as a whole for these two combined and contextualized in the manufacturing industry. To ensure the maintenance process for the manufacturing production line, remote control and monitoring of machines in execution through a network should be possible at any time to detect failures as soon as they happen and avoid any resource losses resulting from an unscheduled stop of a machine on the shop-floor.

To allow a better understanding of the information obtained through the real-time monitoring of the equipment's behavior, the use of visual metaphors as abstraction mechanisms seems suitable, since such approaches have been around the manufacturing and automation industry for a

long time [AGB93]. Visual programming is simply a formalization of the workflow of the tasks to be carried by the operational system. A typical workflow process is made up of a series of tasks and events, the order in which they must occur, and the script that is executed for each event [PGPM19, RWM+18]. A communication protocol is required to connect the visualization of the workflow to be monitored to the physical device where it is running. None of the solutions presented fully answer these demands simultaneously:

1. **Remote connection to a specific physical device running elsewhere without compromising its current execution -** the debugger connects to the physical device through an intermediate that translates the device's variable events according to the communication protocol that the equipment is using, notifying the debugger of the execution events, allowing bi-directional communication for the debugger to transmit the debug commands to the equipment;

2. **Abstraction of the physical device's sequenced execution tasks through a workflow -** the sequenced execution tasks of the physical equipment are translated into a workflow, where there is an abstraction of each task (encapsulated in a box and including some sort of internal business logic) receiving inputs, applying the business logic, and generating outputs which are then passed onto the next workflow task, until it reaches the final one, which has no output;

3. **Ability to debug the workflow being executed by the physical device in real-time (with or without interrupting its execution) -** with the remote connection to a specific physical device running elsewhere, whenever there is an event that triggers the initialization task of the equipment's workflow, it begins execution. The interface of the debugger allows the user to set breakpoints in the task's inputs or outputs, which will get activated whenever a variable passes in that specific point, showing the workflow's state in its entirety so that the user can evaluate what went wrong (if something went wrong) by analyzing the workflow's internal variables that show the state of the physical equipment it is debugging. It must have a debug mode which interrupts the device's execution when a breakpoint is reached for identifying exactly where the system failures start to arise, and a debug mode that never interrupts the device's execution, for debugging the machine whilst it is in execution in the factory's production line, just for value analysis, without interfering with its execution.

Thus, we consider that to fulfill the needs of IIoT systems maintenance, there should be a solution that addresses the challenges mentioned, providing a way to remotely debug IIoT systems with an abstraction of the shop-floor equipment's execution tasks through a workflow.

4.2 Case Study

The proposed solution was developed in the context of Critical Manufacturing's Manufacturing Execution System. It is a software platform with a deep set of modular, fully interoperable, appli-

cations which claims to *"provide manufacturers in complex industries with the maximum agility, visibility, and reliability"* of the manufacturing process [Manf].

The recent advances in the integration of IIoT led to the development of the "Connect IoT" module. This platform is a low-code solution that enables production engineers and system integrators to connect their shop-floor equipment to Critical Manufacturing MES. It has the goal to dramatically reduce the time and effort for this very integration, allowing to create a graphical overview of the automation workflows being carried out by the machines in production [Mand].

It has a single graphical view of automation workflows, providing ways to create and update complex logic via a user-friendly, no-code interface that allows workers with close to none IT-knowledge to fulfil their needs [Mand]. So far, Connect IoT allows to visually debug these workflows and simulate the equipment, with inputs and outputs provided by the user, as the IoT application is running on the background in the same machine as the debugger for the testing of this manufacturing environment before deploying it in the production line.

Machinery downtime should be avoided at all cost since it causes dreadful resource losses for companies [SS17]. To prevent this and to assure maximum production efficiency and effectiveness, real-time remote maintenance is required, which can be achieved through remote debugging of the operations' workflow being followed by the machines in the production line. We implement it as an extension of the currently available module in Critical Manufacturing MES, by receiving the workflow inputs and outputs directly from the equipment.

4.3 Desiderata

In the context of the development of the *proof-of-concept*, this Section describes the base requirements that we aim to tackle in this work. Most of the functionalities that are proposed already exist in current products with generalized applications, but not directed towards the manufacturing industry and its requirements. The development of these functionalities is essential for the validation of the solution proposed in this dissertation, in order to prove that IIoT systems greatly benefit from remote debugging functionalities.

The proposed solution pretends to support the following features:

DS1. There should be an adaptation of a debug session to the manufacturing environment, where there are events defined to be triggered by a certain change in the machine itself (though the integrated sensors and actuators) which will then have to be communicated to the debugger by an intermediary that will be listening to those changes;

DS2. The remote debugging protocol developed should be simplified and generalized so that the number of messages is reduced and can be used for the different debug modes. Each message should have the purpose of informing or notifying the receiver of something that happened or must happen and must be related to a distinct action;

DS3. The debugger should be aware of all the debug sessions it is a participant of, and there should be a frequent checkup for inconsistencies, using the debugging protocol, regarding

sessions that remain available, the debug mode type of each session, etc., according to the *availability rules* to be defined in Chapter 5, so that the machine's execution is not being interrupted when it shouldn't;

DS4. The solution should be able to handle all connection events and issues that may arise, always informing the user of that is happening through the visual interface, trying to reconnect with other entities as soon as it is possible;

DS5. Creating an abstraction of the execution tasks using a representation through workflows should provide a better understanding of the machine execution and allow to detect failures in real-time and what is causing the incorrect behavior, avoiding losses resulting from machinery downtime. This can be achieved through a simple interface that allows knowing what inputs were received and what outputs were generated by a task and the variable's evolution through the workflow path, presenting the debug information intuitively so that the general factory worker with close to none IT knowledge can operate with it;

DS6. The remote debugging method should provide a better insight of the production machinery, even if it is in execution (through the snapshot debug mode), in a real-life production line.

The aforementioned desiderata items are the base requirements for this project and is essential that the *proof-of-concept* developed in this dissertation answers these demands to prove that remote debugging can be applied to an IIoT workflow and improve the maintainability process of IIoT manufacturing systems.

4.4 Solution Proposal

This dissertation delves into the challenges of debugging IIoT systems, namely remote debugging strategies. A prototype to demonstrate the feasibility of both synchronized and snapshot remote debugging applied to an IIoT workflow was designed, constructed, and tested/evaluated. The main goal of this approach is to achieve is a 0-downtime workflow remote debug approach, contemplating the design and implementation of a suitable debugging protocol and a supporting platform where workflows can be easily configurable, monitored and debugged while developing or during production, choosing to interrupt or not the workflow's operations while debugging.

It will allow factory workers to debug a working machine over the network to access the feedback provided by the sensor devices and control the execution through the setup of breakpoints in the machine's workflow. This will provide access to the input and output values of each task, allowing to detect what may be causing incorrect behavior, resulting in an erroneous service in the production line. The proposed solution includes four debug modes:

Mock Debug All the workflow inputs are manually inserted into the UI of the debugger application. The IoT application runs in the same machine as the debugger just for simulating the execution of the real manufacturing environment;

Remote Synchronous Debug The workflow inputs are received directly from the manufacturing equipment and/or from MES. When the execution of the workflow stops on a breakpoint, the entire workflow stops and the developer can check and modify the state of the internal variables, before proceeding with the execution;

Remote Snapshot Debug Similar to the previous approach, but in this case the workflow developer can only check the current state of the workflow execution, not interrupting the source system;

Remote Profiler Debug Same as the Snapshot Debug mode approach, changing only in regards to the way the information is presented to the user.

By achieving a 0-downtime workflow remote debug approach, it is expected that the prototype developed provides a better debugging environment for both simulated and real-time production shop-floor execution systems, having a greater impact on reducing machine downtime during production and leading to the reduction of resource losses.

4.5 Methodology

The developed debugging functionalities will be tested through replication of common use case scenarios of a debugging system applied to a manufacturing system and the validation of the results obtained will be done by comparing them to the ones expected in those scenarios.

In order to evaluate the success of this work, most of the use cases will be directed towards the validation of the protocol implementation. The main goal is to guarantee that the interaction between entities and the different debugging modes is being done correctly considering the *availability rules* defined and that will be explained in Chapter 5. Every interaction will be covered, including invalid scenarios and the management of connection between entities, and thoroughly tested in order to validate the protocol developed.

It is expected that the main contributions of the developed work, focused on the implementation of IoT in the manufacturing industry, are related with the usage of real-time remote debugging during the maintenance phase, whilst in production, and how it can help minimize incorrect machinery behavior and, consequently, resource losses, also aiming at a higher production efficiency.

4.6 Conclusions

The solution here presented is an approach to remote debugging of IIoT systems, using visual metaphors as an abstraction mechanism for the equipment's execution tasks through workflows. This *proof-of-concept* aims at the demonstration of the feasibility of both synchronized and snapshot remote debugging applied to IIoT automation workflows. The solution will provide a total of four different debugging modes:

1. Debug of a simulation of the workflow's execution for testing before it is deployed in the production line;
2. Synchronous debugging of the shop-floor equipment that is not in active production;
3. Snapshot debugging of the shop-floor equipment in production;
4. Profiler debugging of the shop-floor equipment in production (captured just like the snapshot debugging mode but presenting the feedback received in a different way).

It aims at easing the maintenance process and providing the means for the detection of failures for both simulated and real-time production shop-floor execution systems. The debugging of the automation workflows being carried out by machines in production is done through a graphical user-friendly overview, where the workflows can be easily created, configured and monitored using the debug functionalities mentioned above.

It is expected that this solution, if successfully implemented, will provide a better debugging environment for both simulated and real-time production shop-floor execution systems, having a greater impact on reducing incorrect machinery behavior, as well as machine inactivity due to failures, consequently reducing resource losses and improving production efficiency.

Chapter 5

Solution Overview

Contents

This Chapter details the proposed solution, explaining its main software components and the techniques used to tackle the development problems.

Our approach focuses on updating the existing components of the MES software to support the new functionalities, more specifically, the Automation Controller process, mentioned in Subsection 2.2.2.1, and the Automation Controller view on MES where it is possible to build the workflows and debug them locally.

Because it is the Automation Controller process that will be running the workflow, receiving the events from the Automation Driver(s) connected to the device we want to debug, the communication between this component and the MES instance is essential to allow the remote debugging functionalities that this work aims to implement. For this, the modified components have to behave accordingly with the definition of the remote debugging protocol developed.

After the environment setup is done on MES (as mentioned in Subsection 2.2.2.1), in the Automation Controller workflow view of the MES interface, a dropdown list was added to allow the selection of the debug mode (Figure 5.1):

Mock Debug Debug of a simulation of the workflow's execution for testing before it is deployed in the production line;

Synchronous Remote Debug Synchronous debugging of the shop-floor equipment that is not in active production;

Snapshot Remote Debug Snapshot debugging of the shop-floor equipment in production;

Profiler Remote Debug Profiler debugging of the shop-floor equipment in production (captured just like the snapshot debugging mode but presenting the debug information in a different way).

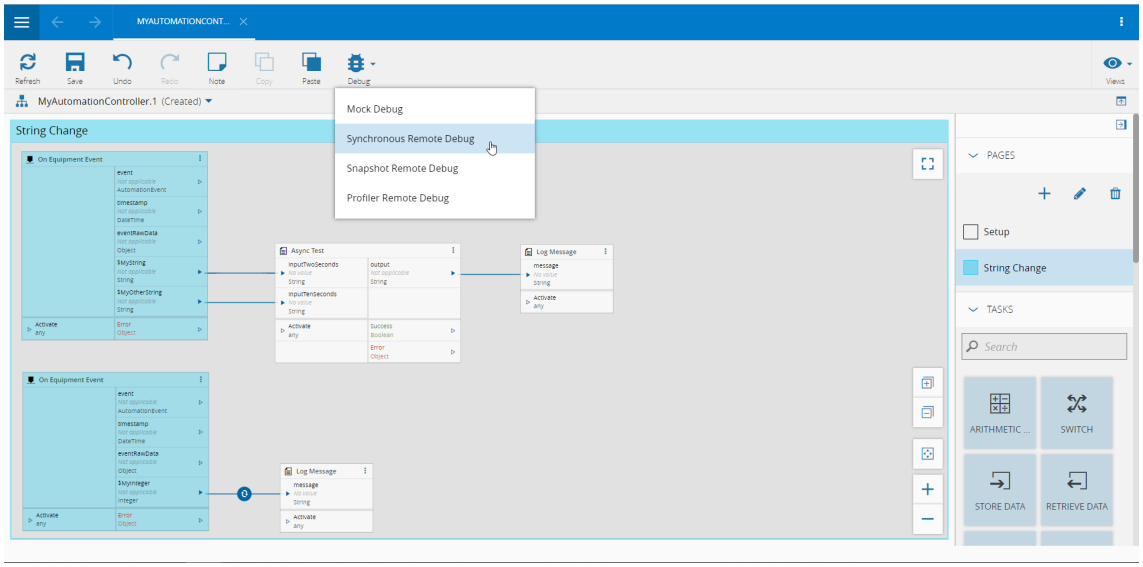

Figure 5.1: Dropdown list added to the MES interface for choosing the debug mode to debug the selected workflow.

The Mock Debug mode debugs an IoT application running in the same machine, as the debugger uses workflow inputs which are manually inserted into the UI of the debugger application. The workflow engine is running locally. This is useful for simulating the execution of a real manufacturing environment.

The remaining debug modes in the list interact with an IoT application running remotely, in physical devices executing in a real production factory environment, receiving the workflow inputs directly from the manufacturing equipment.

Because these modes require a connection with remote entities (the IoMT agents), a communication protocol was developed. To sent the messages between these entities, the message bus that was already implemented in the Critical Manufacturing MES, which uses WebSockets, was used. This way, all the messages will go directly to the message bus gateway, being diffused by all the MES instances and by the IoMT agents running the Automation Controller instances close to the physical machines. Four of the implemented methods in the message bus were used by this communication protocol: publish (sends a message asynchronously), sendRequest (sends a message synchronously and waits for the response), reply (replies to a sendRequest message), subscribe (subscribes to a message of a certain subject and registers the callback to be called when a message of this type is received).

In the remote debugging protocol, each message identifies the components it wants to be associated with as attachments to the message subject. An overview can be found in Table 5.1.

Message	Flow	Type	Description	Parameters	Reply
onCommunicationStarted	IoMT-MES	Async	Connection attempt from an IoMT agent upon initialization with all the MES instances available. It is sent to start/restart the communication between them and inform if the workflow instance selected on the debugger page is running in that server.	automationControllerInstanceId: string	onCheckWorkflowRunning-_(automationControllerInstanceId)
onCommunicationAttempt-_(automationControllerInstanceId)	MES-IoMT	Sync	Connection attempt from a MES instance upon selection of the Automation Controller instance to debug, repeated every 10 seconds to ensure it detects a change in the connection if it happens.	reply: function	reply
onCheckWorkflowRunning-_(automationControllerInstanceId)	MES-IoMT	Async / Sync	Communicates with the selected Automation Controller instance to know if the workflow instance selected on the debugger page is running in that server. The answer is received in the MES instance through the onCheckWorkflowRunningResponse_(automationControllerInstanceId)_(workflowId) message, in case of async request, or through a reply in the start debug function, in case of sync request (this will get a reply if it is running or get no reply otherwise).	workflowId: string, sessionId: string, reply?: function	onCheckWorkflowRunning-Response_(automationControllerInstanceId)_(workflowId), reply
onCheckWorkflowRunning-Response_(automationControllerInstanceId)_(workflowId)	IoMT-MES	Async	Checks if, according to the sessionIds related with the selected workflow instance, the workflow is available for debugging or not.	running: boolean, sessionsId: DebugSessionInfo[]	
onBreakpointChange_-(automationControllerInstanceId)_(workflowId)	MES-IoMT	Async	Adds or removes a breakpoint on the selected workflow instance.	sessionId: string, breakpoint: BreakpointDefinition	
onBreakpointToggle_-(automationControllerInstanceId)_(workflowId)	MES-IoMT	Async	Enables or disables a breakpoint on the selected workflow instance.	sessionId: string, breakpoint: BreakpointDefinition	
onStartDebug_(automationControllerInstanceId)	MES-IoMT	Async	Initializes a debug session in the selected Automation Controller instance, if available, generating a unique debug session Id and updating the workflow breakpoints associated with the session.	mesId: string, workflowId: string, debugMode: DebugMode, breakpoints: BreakpointDefinition[]	onDebugStarted_(mesId)_-(automationControllerInstanceId)_(workflowId)
onDebugStarted_(mesId)_-(automationControllerInstanceId)_(workflowId)	IoMT-MES	Async	Communicates the newly created debug session Id to the MES instance that asked for the start of that debug session.	sessionId: string	
onStopDebug_(automationControllerInstanceId)	MES-IoMT	Async	Stops an existing debug session.	sessionId: string	onDebugStopped_(automationControllerInstanceId)_-(sessionId)
onDebugStopped_(automationControllerInstanceId)_-(sessionId)	IoMT-MES	Async	Updates the session registry array.	sessionId: string, registry: DebugSessionInfoEntry[]	
onSessionRenewal_(automationControllerInstanceId)	MES-IoMT	Async	Renews an existing debug session. This is necessary to guarantee that the session hasn't expired and remains active over time.	sessionId: string	
onBeforeSetOutputs_(automationControllerInstanceId)_-(sessionId)	IoMT-MES	Async / Sync	Notifies the MES instance debugging a certain workflow that a variable on a task output with a breakpoint has been changed, changing the visual representation of the workflow in the interface.	workflowId: string, registryEntry: DebugSessionInfoEntry, reply?: function	reply
onAfterSetInputs_(automationControllerInstanceId)_-(sessionId)	IoMT-MES	Async / Sync	Notifies the MES instance debugging a certain workflow that a variable on a task input with a breakpoint has been changed, changing the visual representation of the workflow in the interface.	workflowId: string, registryEntry: DebugSessionInfoEntry, reply?: function	reply
onReceivedExecutionContext_(automationControllerInstanceId)_(workflowId)	MES-IoMT	Async	Saves the execution context of an snapshot debug session it wants to follow.	sessionId: string, executionContext: string	
onAvailableACIRequest	MES-IoMT	Async	Requests an update of all running Automation Controller instances of the current running state of the selected workflow in the debugger page.	workflowId: string	onAvailableACIRequest-Response_(workflowId)
onAvailableACIRequest- Response_(workflowId)	IoMT-MES	Async	Updates the currently available Automation Controller instances array with the information received regarding the automation controller instance.	automationControllerInstanceId: string, running: boolean	

Table 5.1: Remote debugging protocol developed.

5.1 Connection Preparation

When reaching the debugger page, both the debug mode and workflow to debug are already selected so, from there on, when remaining in this page, all the communication will be regarding that workflow and will have in consideration the debug mode that was selected. The only element to select that remains missing is the Automation Controller instance (ACI) we wish to connect with to begin the debug of the workflow. For that, the `onAvailableACIRequest` message is broadcasted to all the entities connected to the message bus, sending the ID of the selected workflow. The Automation Controller instances subscribed to this message will reply with the message `onAvailableACIRequestResponse` informing their unique ID and the execution state of the workflow in that controller instance. According to the response received, the MES instance will gather the IDs of the Automation Controller instances where the selected workflow is running. If there is more than one positive response, the IDs will be displayed in a dialog for the user to select which controller instance he wishes to debug (Figure 5.3). If only one controller instance replied within 5 seconds, the MES instance will assume it is the only one running the workflow and will automatically select it.

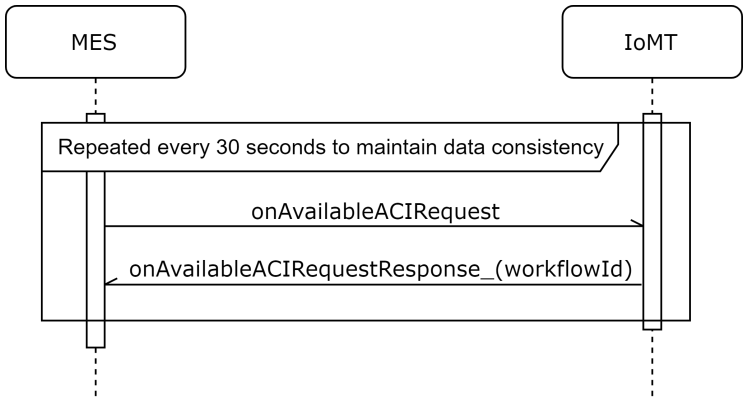

Figure 5.2: **Visual representation of the protocol messages interaction to query the Automation Controller instance availability.** The `onAvailableACIRequest` is broadcasted to all the entities listening, to which the Automation Controller instances will reply with the availability state of the workflow sent in the message's body (if it is running or not) with the `onAvailableACIRequestResponse` message. This interaction is repeated every 30 seconds to ensure data consistency and continuously update the list of available Automation Controller Instances to debug.

Once the Automation Controller instance selection is over, the communication between this instance and MES begins, ensuring the connection between these over time until the user stops the communication (for example, by leaving the debugger page). Once the controller instance selection is done, the MES instance sends an `onCheckWorkflowRunning` to know if the workflow is still running and if it is available for debugging in the selected mode. This message is more refined than the previous one used to know if the workflow is running because it will also receive the debug sessions (and respective mode type) associated with it, determining its availability status:

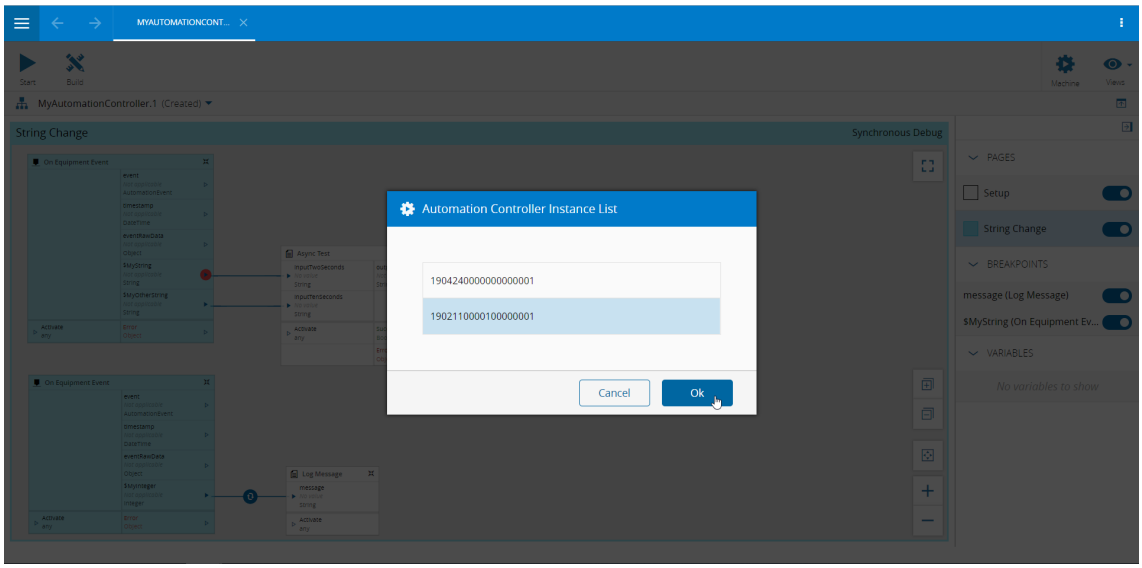

Figure 5.3: **Available Automation Controller instances list.** It is presented as a dialog for the user to select which one he wishes to debug.

- Only one synchronous debug session can be active at once, and no other debug sessions are allowed until that session has ended (synchronous, snapshot or profiler);
- There can be more than one snapshot session at the same time since they don't interfere with the machine's execution, but no synchronous session can start until all snapshot or profiler sessions have ended.

The workflow's availability status only depends on the debug sessions being performed in the Automation Controller instance that is selected.

The response, `onCheckWorkflowRunningResponse`, will determine, according to these rules and the chosen debug session, if the workflow is available for debugging or not, determining the state of the "Start Debug" button (enabled/disabled).

To keep the consistency of the information presented, both `onAvailableACIRequest` and `onCommunicationAttempt` requests are sent periodically, with the intervals of 30 and 10 seconds respectively. In case the `onCommunicationAttempt` message got a positive response, it will lead to the dispatch of the `onCheckWorkflowRunning` request, along with the message to renew the debug session state, if it has one, `onSessionRenewal`.

The session renewal state ensures that the sessions that unexpectedly disconnect over time are wiped by the IoMT agents periodically (every 30 seconds). The same is done regarding the available controller instances, which have a renew state that wipes the ones that unexpectedly stopped replying to the `onAvailableACIRequest` messages, every 30 seconds. If for some reason, the connection between the two entities (MES and IoMT) drops because the selected IoMT agent has disconnected, it will send an `onCommunicationStarted` message upon initialization in case it was connected to an MES instance to quickly re-establish the connection.

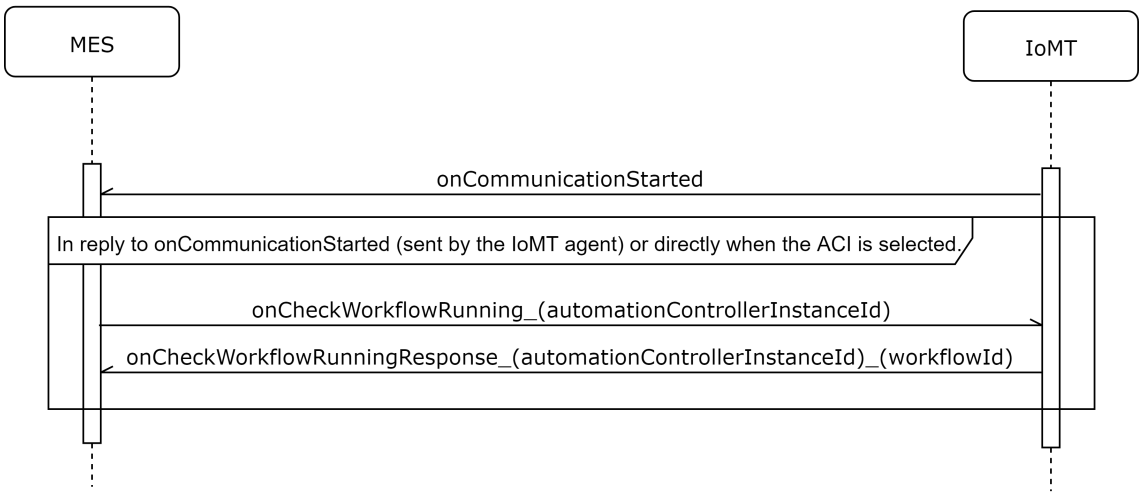

Figure 5.4: **Visual representation of the protocol messages interaction to initialize the communication between the selected Automation Controller instance and the MES instance.** The `onCheckWorkflowRunning` is sent in response to the `onCommunicationStarted` message (sent by the selected IoMT agent upon reconnection, in case the connection dropped), or automatically when the Automation Controller instance is selected, to query if the workflow remains running and is available for debugging according to the rules specified. The Automation Controller instance responds with the `onCheckWorkflowRunningResponse`. This interaction is absolutely necessary since it verifies that the new debug session won't interfere with an already existing session on that machine.

5.2 Debug Session

To start the debugging session, when the "Start Debug" button is clicked, the `onStartDebug` request is sent to the selected Automation Controller instance. It will re-check all the conditions previously mentioned to know if the workflow is available for debugging and if so, it will generate a unique session ID and update the workflow breakpoints associated with the debug session. The session ID will be communicated to the respective MES instance through the `onDebugStarted` message, using the unique MES instance identifier.

In the Automation Controller, during the equipment setup, the engine that is running the workflow will associate the Automation Driver to each task so that when a variable change happens, the driver will detect the changes and will emit an event subscribed by the respective task input/output (the subscription is done during the engine's workflow setup). When this event is detected by the task, it will trigger the `onBeforeSetOutputs` or the `onAfterSetInputs` hook respectively in case it was a task output or input. This process is common to all debug modes (remote or not).

In the case of the Mock Debug, the workflow engine is running next to the MES instance, receiving the task inputs directly from the user (Figure 5.8), and the hooks behave slightly differently. The functions associated with the hooks are directly injected in the local engine, detecting the modifications in the task's inputs or outputs, which will then trigger the breakpoint, if there is any, in the modified variable, stopping the workflow execution when a breakpoint is reached until the "Resume" button is pressed.

When working with a remote Automation Controller instance, when these hooks are triggered,

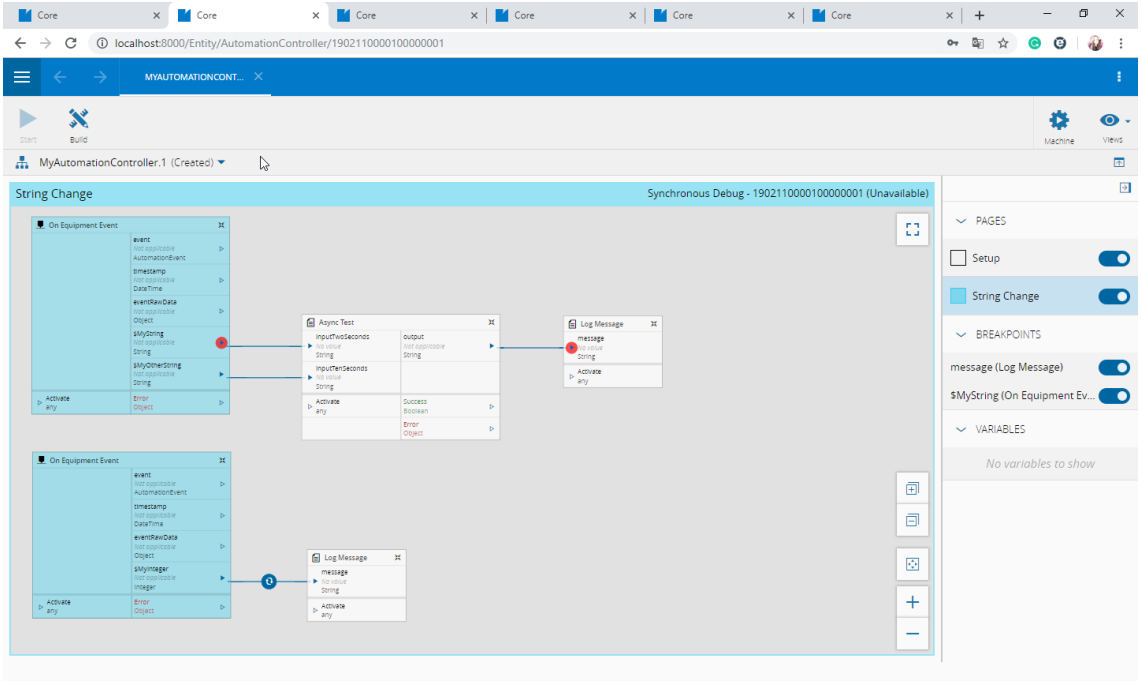

Figure 5.5: **Example of workflow unavailability.** The availability of a running workflow to debug is defined by the *availability rules* defined above. In this figure, a snapshot session was already active for this workflow in this Automation Controller instance, and for this reason, it is unavailable for starting a new synchronous session. This information is shown in the interface in the ribbon above the workflow and with the disabling of the "Start Debug" button.

it will identify the debug sessions associated with the workflow where the changes were detected and, for each, it will check if there are any breakpoints in the modified variable.

In the case of the Synchronous Debug, if there is a breakpoint in that variable, it will stop the engine's execution, send an `onAfterSetInputs` or `onBeforeSetOutputs` message, respectively in the case of an input or output, notifying the MES instance of the workflow state when the breakpoint was reached, the variable changes and the triggered breakpoint. The Automation Controller instance will then wait until it receives a reply to these messages.

When receiving the previously mentioned messages, the MES instance will trigger the respective breakpoint, update the workflow task instances to show the workflow state when the changes were detected and stop the engine's execution until the "Resume" button is pressed (Figure 5.9), which will then resume execution of both MES and Automation Controller instance workflow engines by replying to these messages.

Because when the engine is stopped the hooks will still get triggered when a variable is changed, in this mode, all the hooks triggered while waiting for the "Resume" command will be ignored by the engine, along with the propagation of these new variables in the workflow. This debug mode is only suitable for manufacturing systems that are not in production at the moment when the synchronous debugging session is started since it will alter the equipment's execution state.

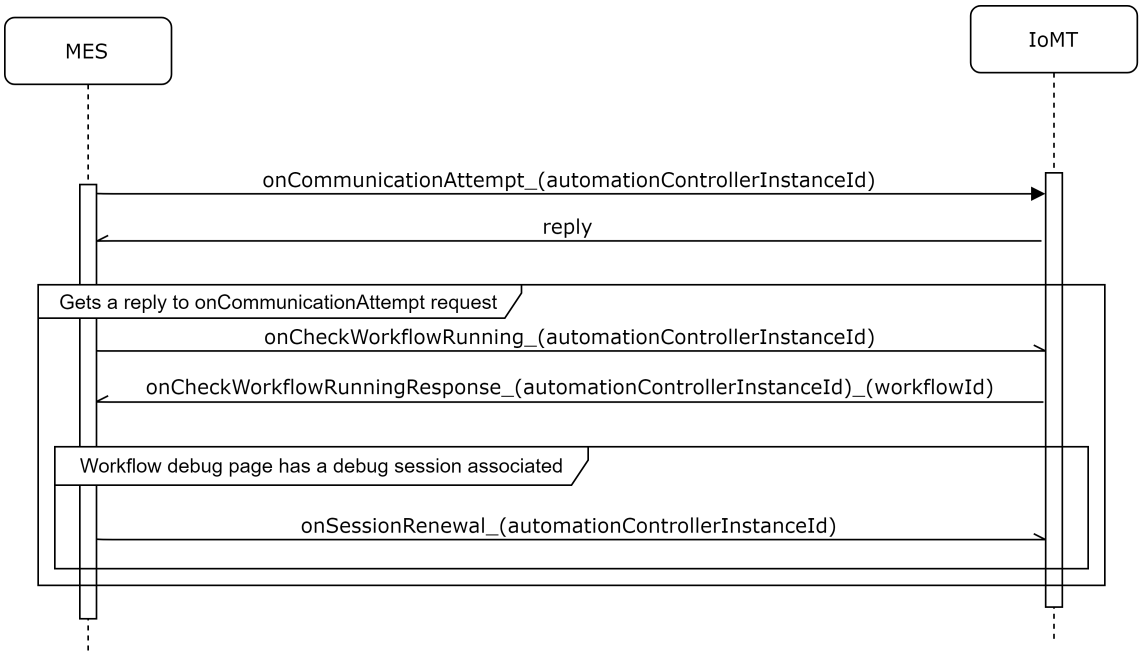

Figure 5.6: **Visual representation of the protocol messages interaction to ensure the connection between the Automation Controller instance and the MES instance, and to obtain the workflow's running and availability status.** The `onCommunicationStarted` message is sent every 10 seconds to maintain data consistency. If it doesn't get a reply, it means the connection with the Automation Controller instance dropped. Upon receiving a reply it will check the availability and running state of the workflow with the `onCheckWorkflowRunning` message and, if there is a debug session active at that moment, it will renew the session state on the Automation Controller instance with the `onSessionRenewal` message, notifying that it remains active. If any state changes are received from these requests it will be shown to the user through the MES interface (e.g. the workflow becomes unavailable and the debug session is terminated, the connection with the Automation Controller instance drops, etc).

In the case of the Snapshot or Profiler Debug, if there is a breakpoint in the modified variable, it will check the execution context where this change was detected, and update the registry of changes on that execution context.

This introduces the concept of a "flow" of the workflow, which begins when an equipment event is triggered and ends at the end of the workflow branch where there is no output link. Each flow will have its own execution context and will not interfere with the execution of another flow.

This was a mechanism made for dealing with asynchronous workflows where, for example, two variables will trigger the same output, but at different times. Imagining that two variables will change consecutively, and the concept of flows hasn't been implemented, and variable A will change prior to variable B but will take longer to reach the common output. The presented information to the user will be $varA \rightarrow varB \rightarrow outputB \rightarrow outputA$. Assuming that the output will be the result of calculations with the received variables but the user does not have insight of those calculations, it will assume the output with variable B was actually gotten using the variable A, since that one was changed first. With the introductions of flows, the user will be presented with only one workflow flow and the information of different execution contexts will not get

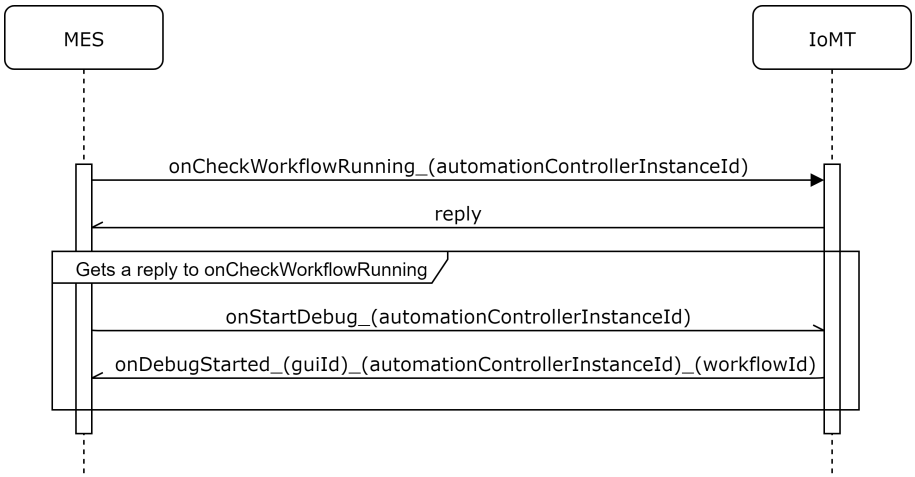

Figure 5.7: **Visual representation of the protocol messages interaction for starting a debug session.** Firstly, it is made sure that the workflow is running and available for debug through the `onCheckWorkflowRunning` request. If this gets a positive response, it requests the start of a new debug session with the `onStartDebug` message, to which the Automation Controller instance will generate a new unique session ID and will transmit it back to the MES instance using the `onDebugStarted` message.

mixed, making the workflow much easier to debug. Whenever an equipment event is triggered by a variable change, it creates a new execution context which will follow the evolution of that variable through the workflow (Figure 5.10).

The Snapshot and Profiler Debug modes work similarly, changing only in regards to the way they present the information to the user. In the case of the Snapshot Debug, we only want to follow the most recent detected execution context. One snapshot session debugs only one execution

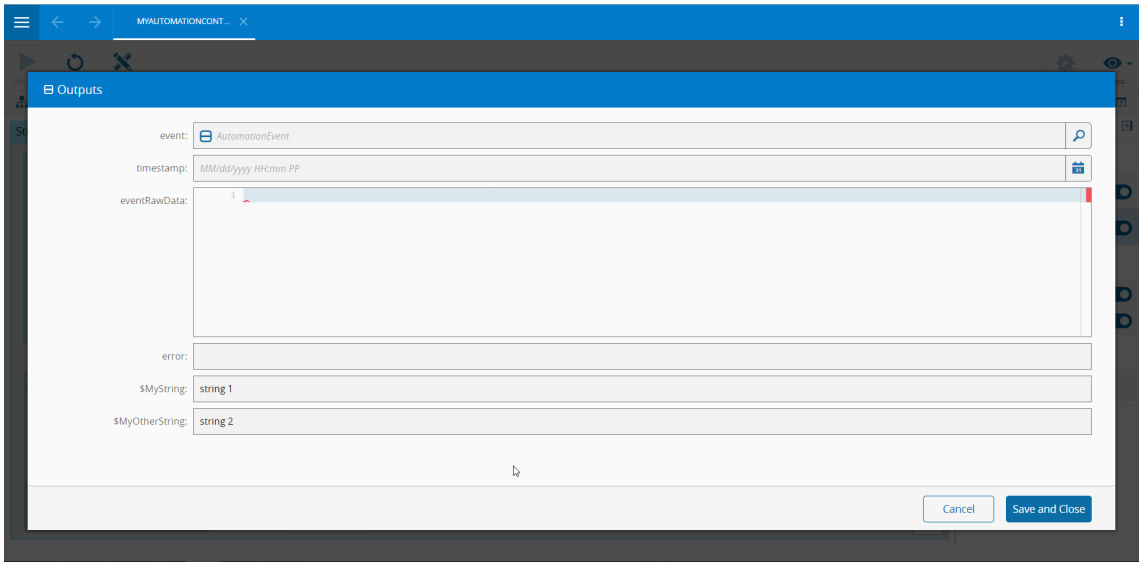

Figure 5.8: Example of the task inputs provided by the user for the Mock Debug.

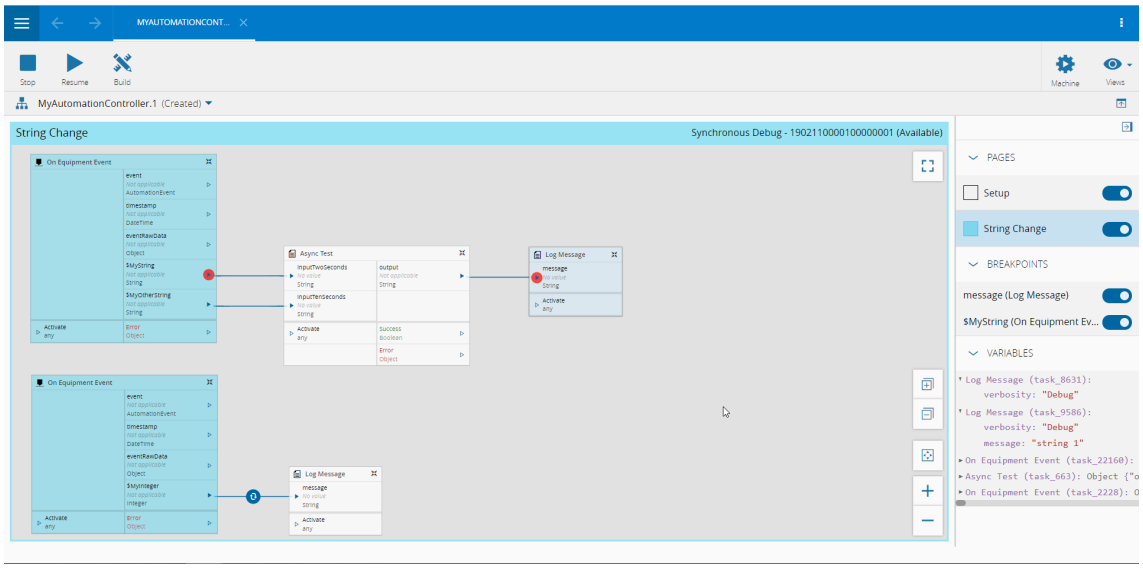

Figure 5.9: **Example of an active breakpoint and how it is shown to the user.** On the synchronous or snapshot debug modes, when a breakpoint is triggered, the workflow task instances are updated to show the workflow state when the changes were detected (on the right side of the interface) until the "Resume" button is pressed.

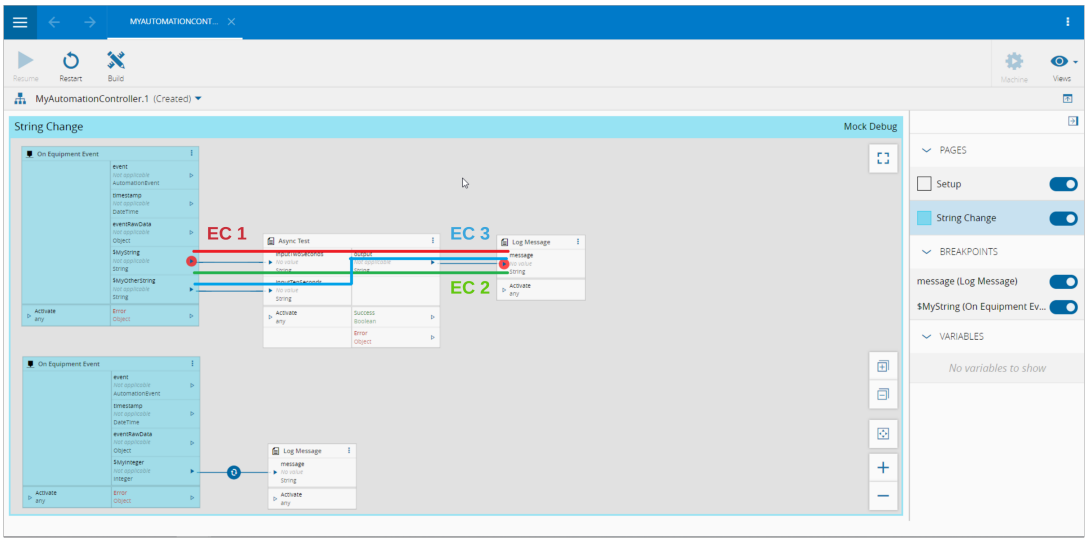

Figure 5.10: **Illustration of the different execution contexts that are created when a change is detected in the equipment.** A new execution context is created whenever an equipment event is triggered by a variable change, following the evolution of that variable through the workflow independently. If any of the variables *MyString* or *MyOtherString* is changed in the "OnEquipmentEvent" task of the workflow presented in this figure, it will create a new execution context, illustrated with different colors.

context to understand exactly what was the evolution of a variable through the workflow and the path it took. When a variable change is detected, if there is a breakpoint in this variable, the Automation Controller instance will send an `onAfterSetInputs` or `onBeforeSetOutputs`

message, respectively in the case of an input or output, notifying the MES instance of the workflow state when the breakpoint was reached, the variable changes and the triggered breakpoint. This will not affect the execution of the engine, and the workflow will keep triggering the event hooks, communicating the respective messages.

When receiving these messages, the MES instance will trigger the received breakpoint, update the workflow task instances to show the workflow state when the changes were detected and stop in that breakpoint until the"Resume" button is pressed. Because the engine's execution is not altered, when receiving a message while waiting at a breakpoint, it will stack the breakpoint promises. So when the "Resume" button is pressed, it will go to the next triggered breakpoint, if there is any, or wait for the next message.

Upon receiving the first variable change message, it will check the execution context in which this event was triggered and will send the `onReceivedExecutionContext` request to inform the Automation Controller instance that it wishes to follow that execution context. This will act as a filter in the event hooks to know which messages should be sent to the MES instances and ignore all variable changes with a different execution context as the one that was chosen for this debug session.

In the case of the Profiler Debug, it will collect the information of the different execution contexts passively, without notifying the user of the changes that are happening in the machine, until the session is stopped. This will allow the user to check all the events that happened while the debug session was active, chronologically. When the session is stopped, if the debug mode selected is the Profiler mode, the Automation Controller instance will reply with the `onDebugStopped` message which will sort the registry entries of all created execution contexts chronologically since the beginning of the debug session and send them to the MES instance. The MES interface will then change to allow a "Replay Mode" which will go through the received registry so that the user can carefully analyze the collected data. When the registry is discarded, a new profiler debug session may begin.

To stop any remote debug session, the `onStopDebug` message is sent when the "Stop Debug" button is clicked. After sending the necessary information to finalize the debug session to the MES instance, it will delete all the session information kept on the IoMT agent (chosen execution context, registry, breakpoints, etc). The information kept in the MES instance is erased differently according to the debug mode:

- **Synchronous:** done immediately after the session is stopped;
- **Snapshot:** done after the last breakpoint promise is resumed;
- **Profiler:** done when discarding the session registry.

Regarding the workflow breakpoints, they are associated with a debug session, allowing multiple snapshot sessions to have their breakpoints independently. The breakpoints are immediately set as they are displayed in the workflow debugger page when a debug session is started. The snapshot or profiler type sessions do not allow any breakpoint changes while a session is active (or while the session registry wasn't discarded in the case of the profiler mode) because this could cause some unexpected results when receiving the events of a breakpoint that was changed

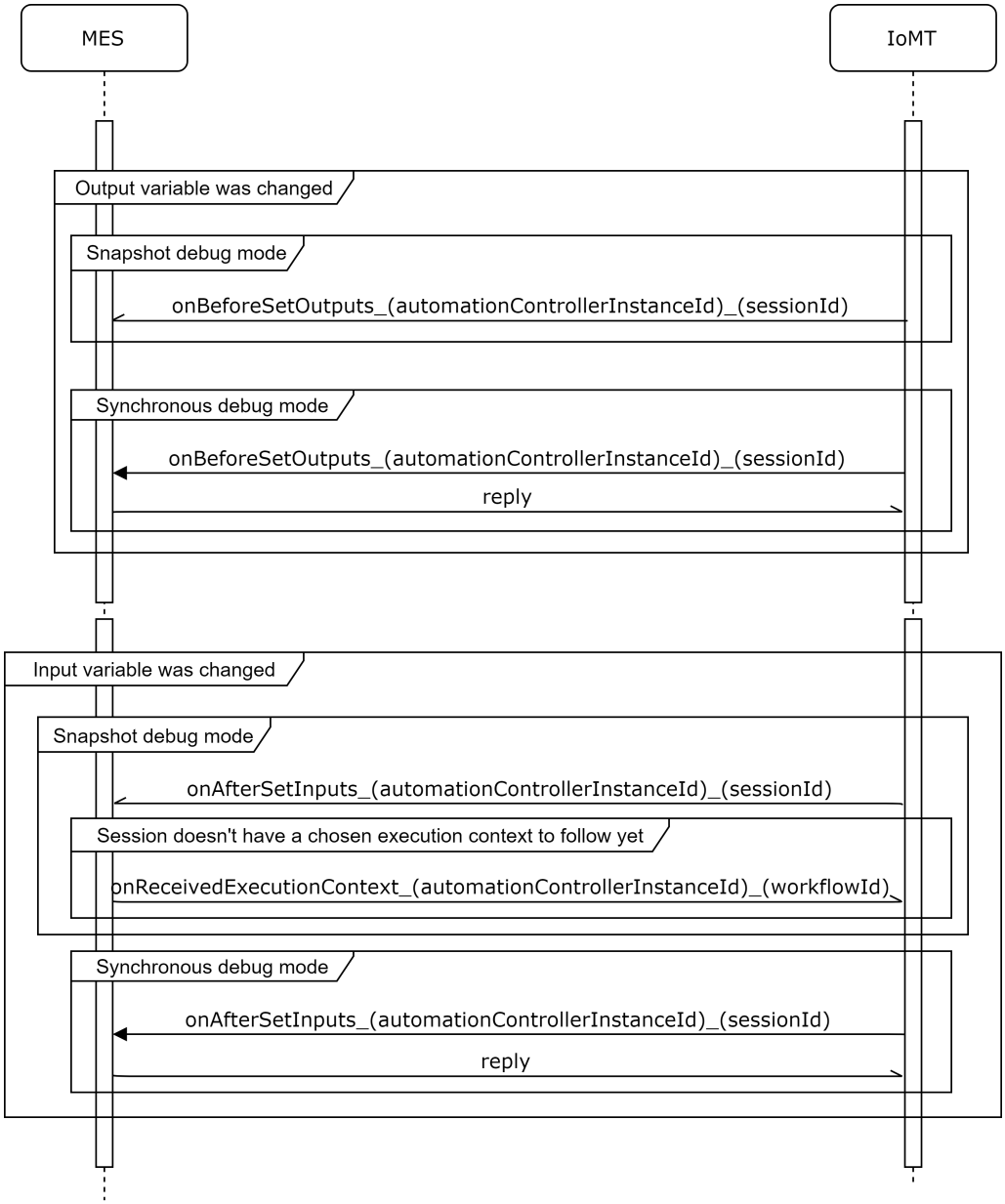

Figure 5.11: **Visual representation of the protocol messages interaction for notifying the MES instance of variable changes (with breakpoints) on the Automation Controller instance.** The `onBeforeSetOutputs` communicates the state of the workflow before the output was set (if the variable that changed was a task output) and `onAfterSetInputs` communicates the state of the workflow after the input was set (if the variable that changed was a task input). If the debug session these messages are related to is a synchronous debug mode session, then the messages will be sent synchronously, and the Automation Controller instance will be waiting to continue the execution until it get a reply from the MES instance (sent by pressing the "Resume" button, when waiting at an active breakpoint). If the debug session is a snapshot debug mode session, the messages will be sent asynchronously and the execution of the Automation Controller instance will continue unaltered.

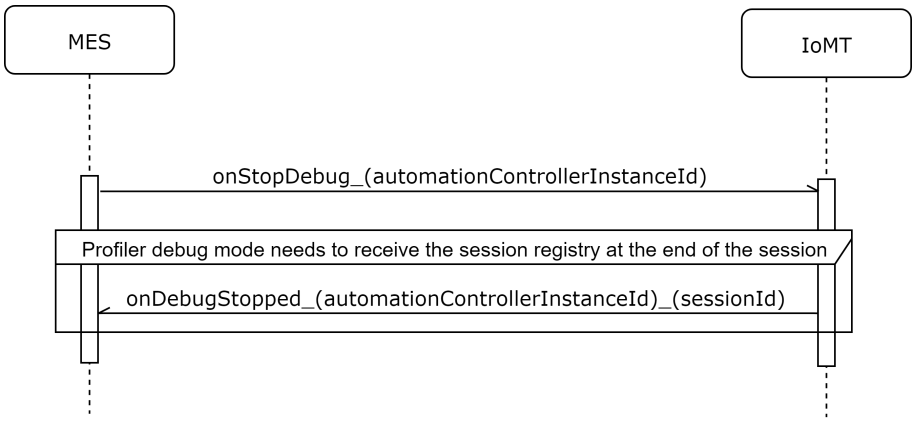

Figure 5.12: **Visual representation of the protocol messages interaction for stopping a debug session.** The `onStopDebug` message ends the session and removes all information related to it from the IoMT agent. Before erasing all the information, if the session was for the Profiler debug mode, the registry entries of that session will be communicated through the `onDebugStopped` message.

while the session was active. For the synchronous session, it is possible to change the breakpoints while the session is active. Breakpoints can be added or removed in the workflow display through the `onBreakpointChange` request or can be toggled to enabled or disabled through the `onBreakpointToggle` request.

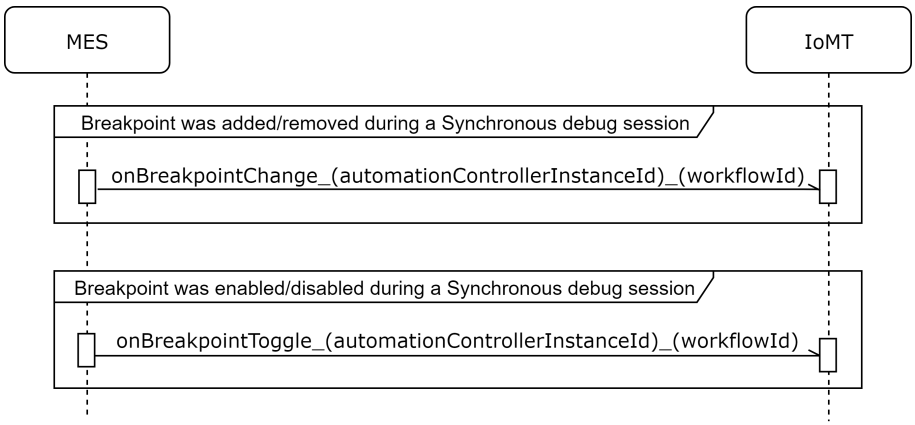

Figure 5.13: **Visual representation of the protocol messages interaction for changing the breakpoints during a synchronous debug session.** In order to change (add or remove) a breakpoint, the `onBreakpointChange` request is sent, and to change the state of a breakpoint without removing it (enable or disable), the `onBreakpointToggle` message is sent.

5.3 Conclusions

This Chapter presented an overview of the solution proposed for multi-approach debugging of industrial IoT workflows. It presented all the details of the remote debugging protocol developed to support this approach and how the different entities interact throughout the entire debugging

process, from the preparation of the debugging environment with the selection of the workflow, debug mode and physical machine to debug, to the actual debugging session.

The approach explained in this Chapter is considered innovative in the area of debugging of industrial IoT workflows with the implementation of a multi-strategy debugging mechanism into a commercial-grade Manufacturing Execution System. It is important to state that this approach is a concept that is meant to show the advantages of using real-time remote debugging in the industrial context, during the maintenance phase of the deployed software, whilst in production, for minimizing incorrect machinery behavior, as well as machine inactivity due to failures in order to reduce resource losses. Chapter 6 will present the conducted experiments for the validation of this solution and the results obtained.

Chapter 6

Evaluation

Contents

In contemplation of the solution presented in Chapter 5, it was proceeded to carry out an evaluation of the approach itself. This was done in order to evaluate the reliability, feasibility, and applicability of the designed approach under different scenarios, to understand how the implemented features work in the already existing environment, if they behave like it was defined in the solution overview, and if the debugging modes interact with one another like they are supposed to, using the developed protocol. Scenarios regarding the management of connection between the entities were also considered.

6.1 Experimental Setup

To test the debugger application easily, an OPC UA server was used for simulating real-time variable changes (Figure 6.1). These servers play a key role as a communication gateway, allowing OPC UA clients to access HMI or PLC data by subscribing to tags to receive real-time updates [Tec]. For this reason, the Automation Driver with the OPC UA protocol associated was the one used for testing the application.

The workflow used for simplification and to ease the understanding of the debugging functionalities developed can be seen in Figure 6.2 and consists of an event to be triggered when a variable in the equipment changes, named “OnEquipmentEvent”. The variables where named *MyString* and *MyOtherString* for simplification. These variables will go through an asynchronous task where one input (connected to the *MyString* variable) will take 2 seconds to be passed as an

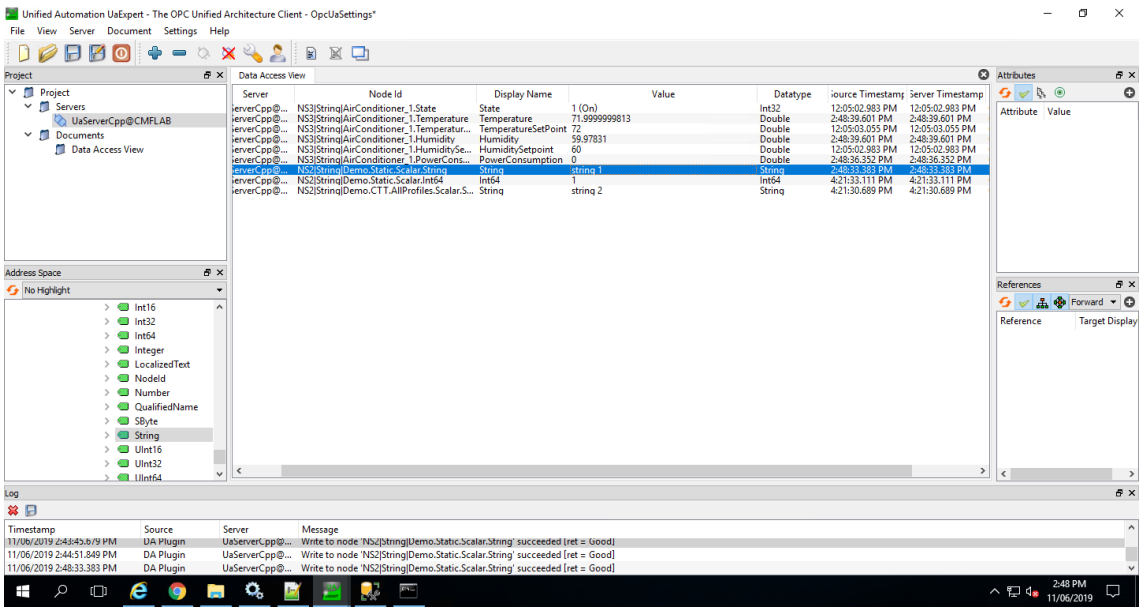

Figure 6.1: OPC UA server used for simulating variable changes in an equipment using the OPC UA protocol.

output and the other 10 seconds (connected to the *MyOtherString* variable), then being printed in the console through the "Log Message" task.

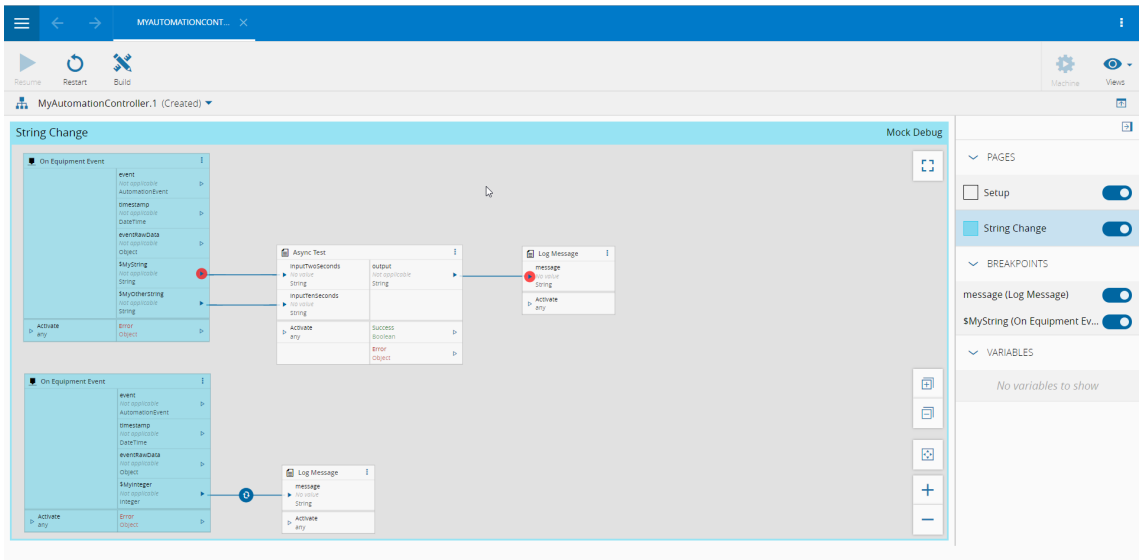

Figure 6.2: Workflow used for validating the functionalities implemented in the solution.

6.2 Simulated Scenarios

The simulated scenarios and respective results are presented in the following Sections. These were made in order to answer the requirements defined in Section 4 and to ensure that the solution developed did encompass all test cases and everything is working as it was defined in the proposal. All of the results obtained matched the expected results and can be observed in the different figures shown in this Section.

6.2.1 Scenario 1 - Available machines running the workflow

- **Scenario:** MES instance receives information about the availability of the selected workflow on an Automation Controller.
- **Expected Behaviour:** If available, the ID of the Automation Controller instance will show up in the Automation Controller list for selection.
- **Desiderata Items:** 4.3.4.
- **Notes:** This scenario can be observed in figure 6.3.

6.2.2 Scenario 2 - Handling disconnections

- **Scenario:** MES instance receives information that an Automation Controller previously available and running the selected workflow has disconnected.
- **Expected Behaviour:** ID of the Automation Controller that disconnected stops showing up on the Automation Controller list. It the Automation Controller instance has been selected, it will show up as "Unavailable" in the ribbon on top of the workflow in the MES instance interface.
- **Desiderata Items:** 4.3.4.
- **Notes:** This scenario can be seen in figure 6.4.

6.2.3 Scenario 3 - Successful connection with a machine

- **Scenario:** MES instance connects to the selected Automation Controller instance.
- **Expected Behaviour:** ID of the selected Automation Controller shows up in the ribbon on top of the workflow in the MES instance interface, along with the current availability of the workflow.
- **Desiderata Item:** 4.3.4.
- **Notes:** This scenario can be observed in figure 6.3.

6.2.4 Scenario 4 - Handling connection events (disconnection and re-connection) with a machine

- **Scenario:** MES instance can handle a disconnect, followed by a reconnect, of the selected Automation Controller instance.

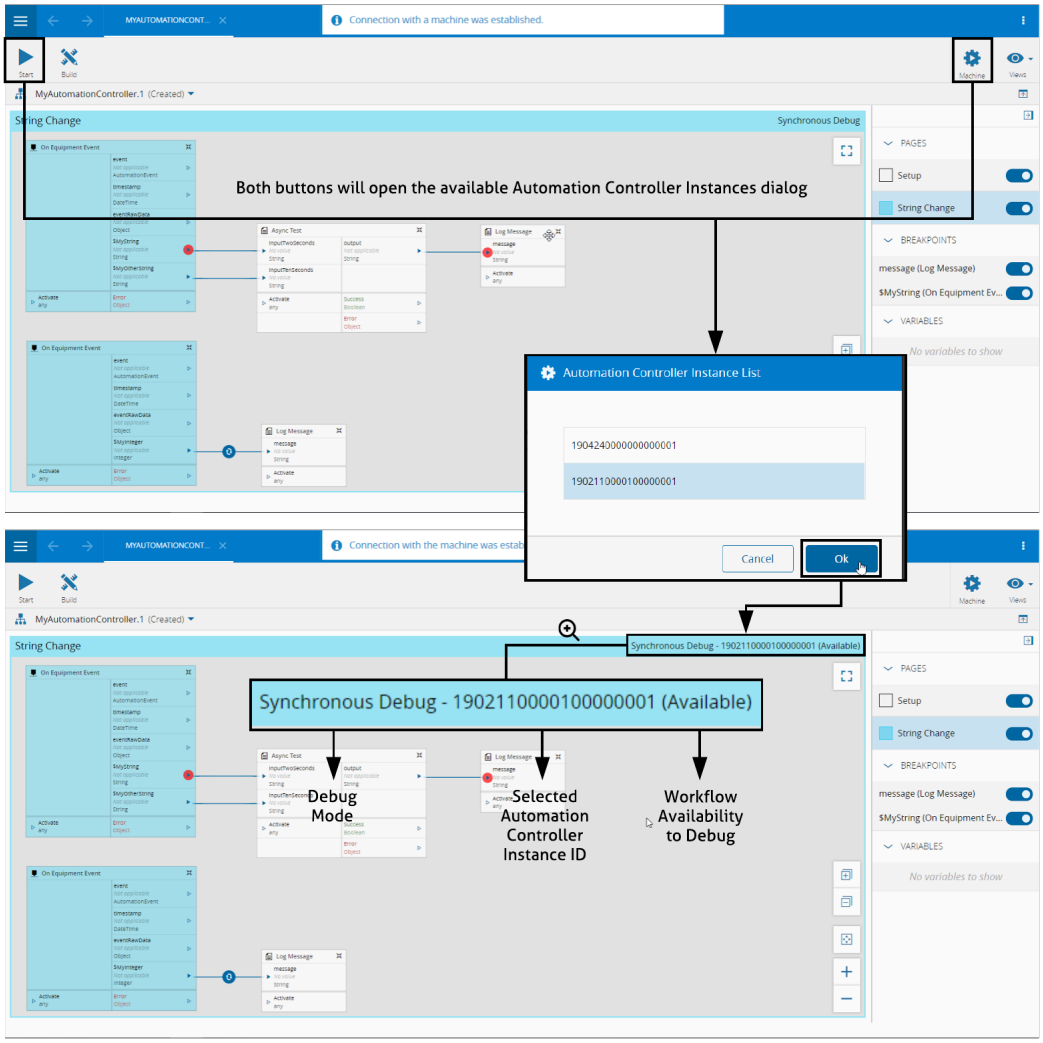

Figure 6.3: **Illustration of the simulated scenarios 6.2.1 and 6.2.3** Screenshots of the MES interface are presented to show the list of Automation Controller instances available for debugging and the selection process of an Automation Controller instance. When none is selected, no ID will show in the workflow top ribbon and both the "Start" and "Machines" buttons will open a dialog with the list for the selection. When it is selected, the top workflow ribbon shows the selected debug mode, Automation Controller instance ID it is actively connected with and the workflow debug availability. The "Machines" button allows to change the selected instance at any time, terminating the active debug session if there is one.

- **Expected Behaviour:** Connect/Disconnect messages will show up and the state of both the workflow availability and the "Start Debug" button will change accordingly. Will stop any active debug session upon disconnection.
- **Desiderata Items:** 4.3.4.
- **Notes:** This scenario can be seen in figure 6.4.

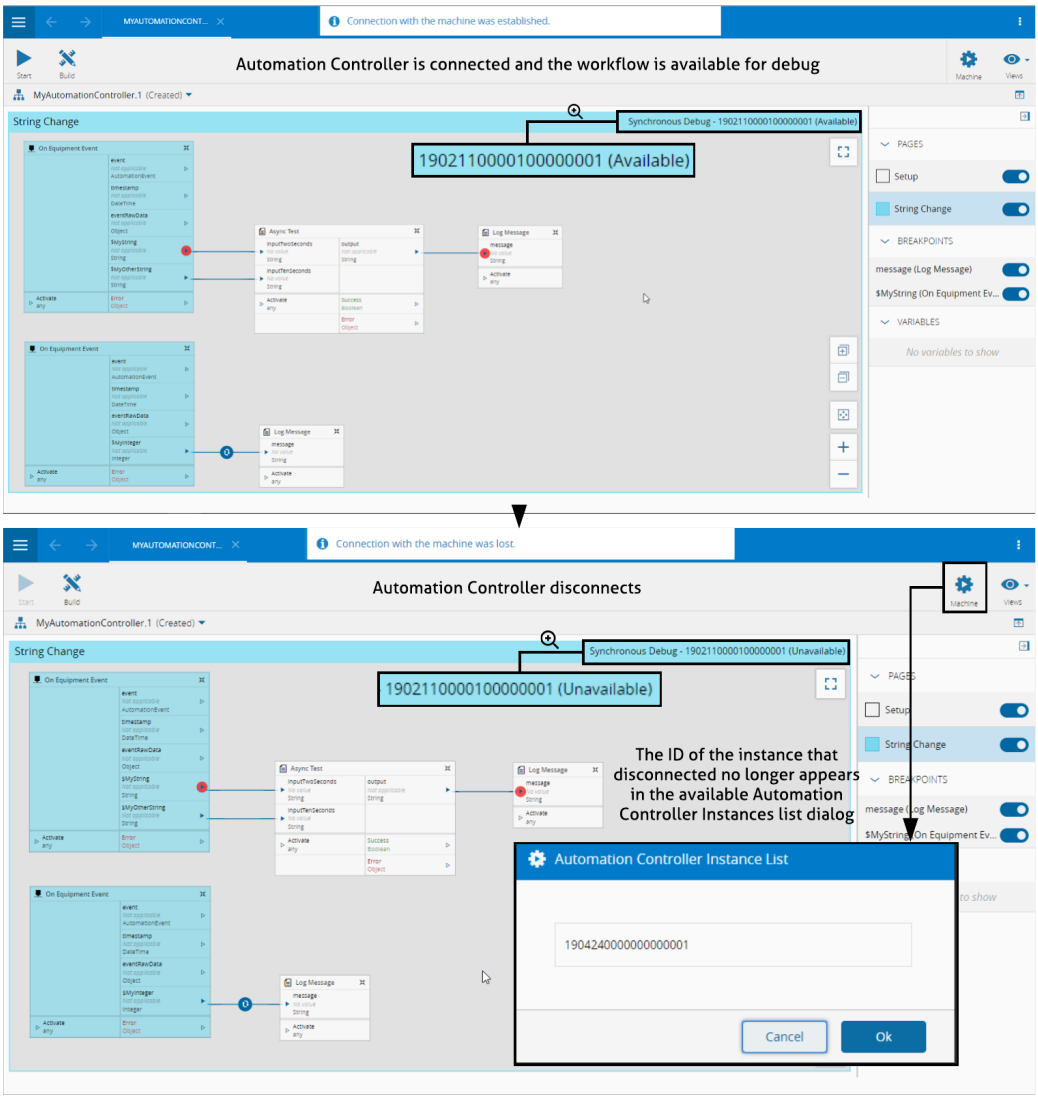

Figure 6.4: **Illustration of the simulated scenarios 6.2.2 and 6.2.4.** Screenshots of the MES interface are presented to show how it handles connection/disconnection events, showing the user what is happening through the interface and with information being shown on top of the page.

6.2.5 Scenario 5 - Synchronous session correctly receives the workflow state upon reaching a breakpoint

- **Scenario:** MES instance can begin a synchronous debug session and show the variable changes as they are received.
- **Expected Behaviour:** When there is an equipment variable change and a breakpoint is linked to that variable, the Automation Controller gets a notification and transmits it to the MES instance. This activates the breakpoint in the interface (the task gets a slight highlight) and shows the state of the workflow at the moment the breakpoint was triggered (in the right

panel of the interface), until the "Resume" button is pressed.

- **Desiderata Items:** 4.3.1, 4.3.2, 4.3.6.
- **Notes:** This scenario can be seen in figure 6.5.

6.2.6 Scenario 6 - Synchronous session ignores any other events whilst waiting for a "Resume" command

- **Scenario:** Automation Controller instance ignores any variable change when waiting for a response from the MES instance during a synchronous debug session.
- **Expected Behaviour:** Nothing happens in the MES instance whilst it is waiting at a breakpoint.
- **Desiderata Items:** 4.3.1, 4.3.2.
- **Notes:** This scenario can be seen in figure 6.5.

6.2.7 Scenario 7 - Snapshot session correctly receives the workflow state upon reaching a breakpoint and doesn't interfere with the workflow's execution

- **Scenario:** MES instance can begin a snapshot debug session and show the variable changes as they are received, stacking the breakpoints received if required.
- **Expected Behaviour:** Same as the synchronous debug, but if there are any changes whilst waiting for a "resume" response, the breakpoint promises stack and the next breakpoint will activate when the "Resume" button for the current breakpoint is pressed.
- **Desiderata Items:** 4.3.1, 4.3.2, 4.3.6.
- **Notes:** This scenario can be seen in figure 6.7.

6.2.8 Scenario 8 - Snapshot session holds to an execution context and ignores all events from other execution contexts

- **Scenario:** MES instance doesn't show any variable change during a snapshot debug session if it is not from the chosen execution context.
- **Expected Behaviour:** Variable changes from other execution contexts are ignored and nothing relative to those is shown in the MES instance.
- **Desiderata Items:** 4.3.6, 4.3.5.
- **Notes:** This scenario can be seen in figure 6.7.

6.2.9 Scenario 9 - Profiler session captures all events from all execution contexts and shows this registry in chronological order upon reaching the "replay mode"

- **Scenario:** MES instance can begin a profiler debug session and show the variable changes collected by the Automation Controller instance, chronologically, through "replay mode".

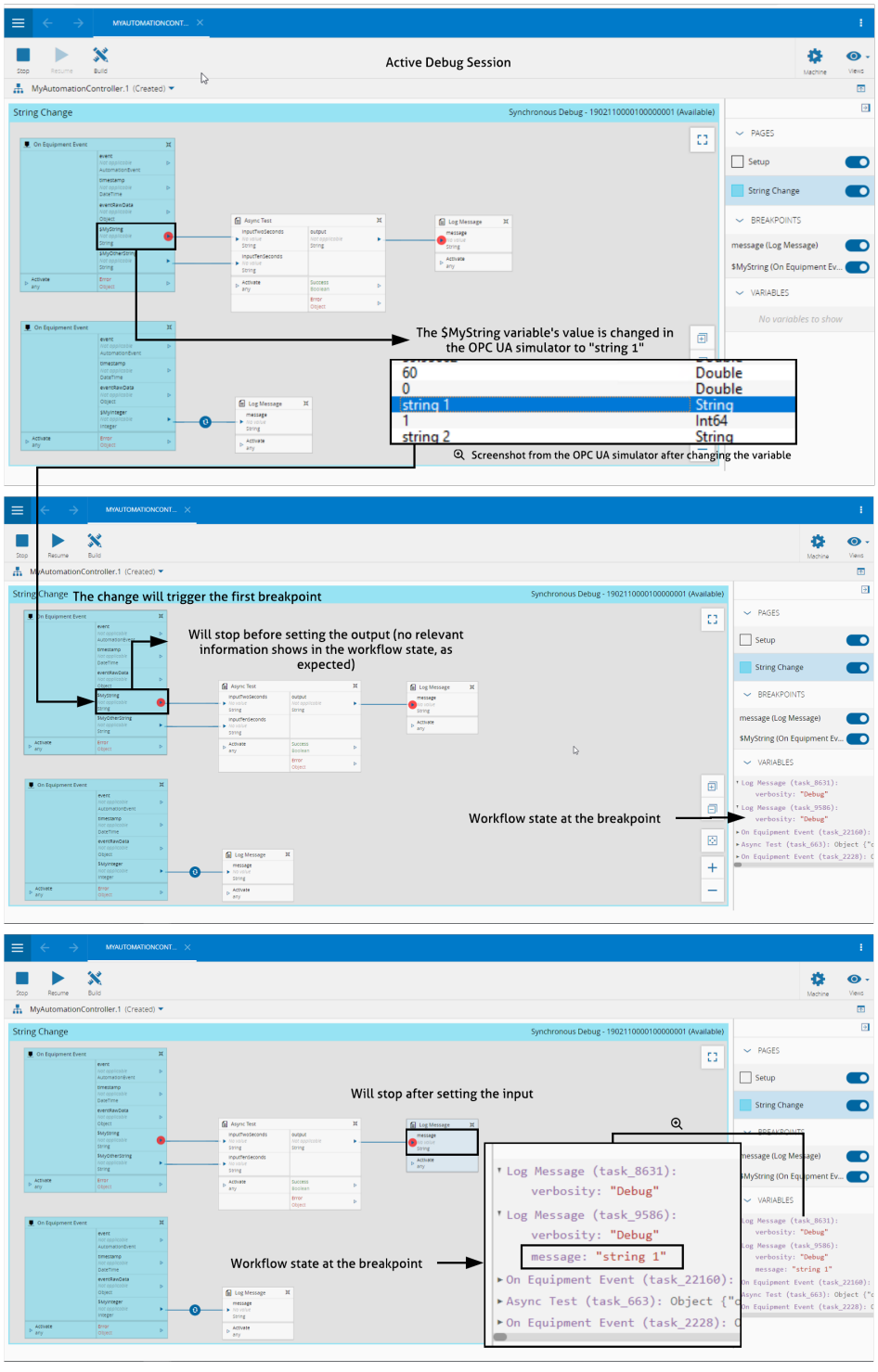

Figure 6.5: **Illustration of the simulated scenario 6.2.5.** Screenshots of the MES interface are presented to show how a synchronous debug session is presented to the user. Every time a variable connected to the "OnEquipmentEvent" is changed, it triggers a new flow and, if in its workflow path a breakpoint is activated, the workflow will stop, the workflow's state will be presented to the user and the Automation Controller instance will wait until the "Resume" button is pressed to continue the execution.

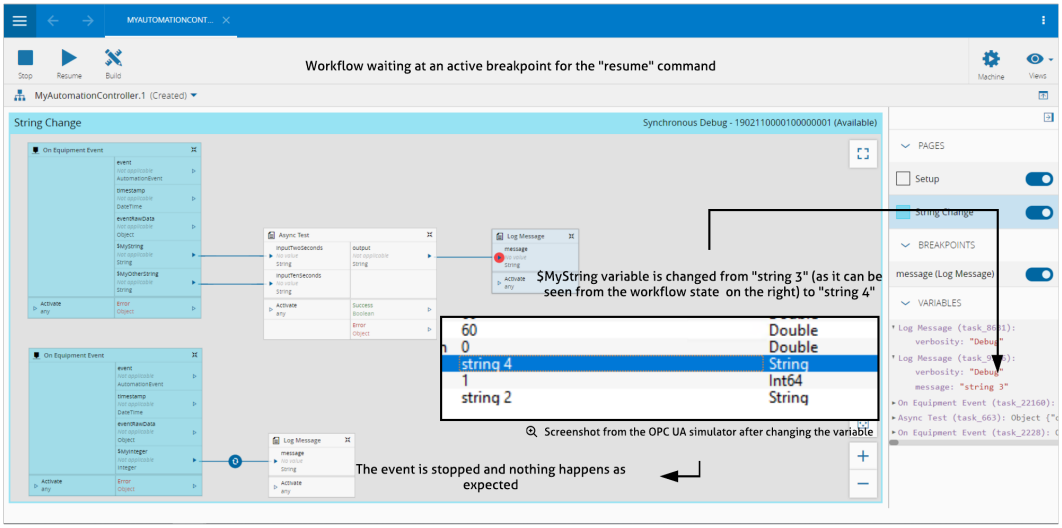

Figure 6.6: **Illustration of the simulated scenario 6.2.6.** Screenshots of the MES interface are presented to show how a synchronous debug session handles receiving any events whilst waiting for a *resume* command. The interface will remain the same and ignore any variable changes that happen whilst it is waiting for the command.

- **Expected Behaviour:** Collects variable changes from all execution contexts during the debug session. When stopped, sends the information to the MES instance where the user will see all changes that happened, chronologically, whilst debugging, using the breakpoint activation method used in the snapshot debugging.
- **Desiderata Items:** 4.3.1, 4.3.2, 4.3.6.
- **Notes:** This scenario can be seen in figure 6.8.

6.2.10 Scenario 10 - Workflow availability whilst involved in a synchronous debug session

- **Scenario:** MES instance doesn't let any other debug session start if there is already a synchronous debug session for the selected workflow and Automation Controller instance.
- **Expected Behaviour:** "Start Button" of all other debug sessions related to the same Automation Controller instance is disabled if there is already a synchronous debug session.
- **Desiderata Items:** 4.3.1, 4.3.3.
- **Notes:** This scenario can be seen in figure 6.9.

6.2.11 Scenario 11 - Workflow availability whilst involved in one or more snapshot or profiler debug sessions

- **Scenario:** MES instance doesn't let any synchronous debug session start if there is at least one snapshot or profiler debug session for the selected workflow and Automation Controller

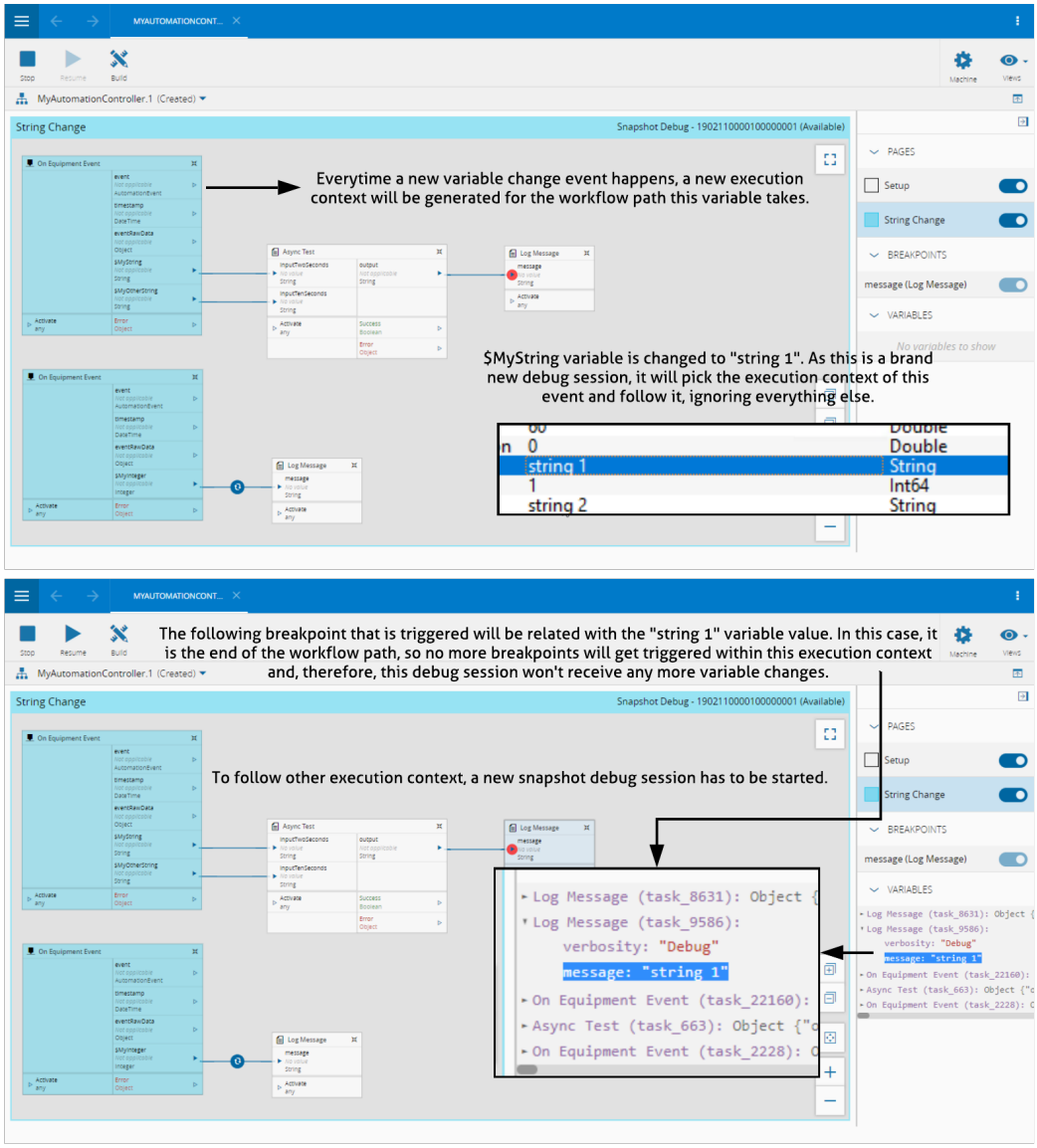

Figure 6.7: **Illustration of the simulated scenarios 6.2.7 and 6.2.8.** Screenshots of the MES interface are presented to show how a snapshot session is presented to the user. Every time a variable connected to the "OnEquipmentEvent" is changed, it generates a new execution context and the snapshot session picks a execution context and follows the variable's path along the workflow, not interfering with the workflow's execution. To follow another execution context, a new snapshot debug session has to be initialized.

instance.

- **Expected Behaviour:** "Start Button" of all synchronous debug sessions related to the same Automation Controller instance is disabled if there is at least one snapshot or profiler debug session.
- **Desiderata Items:** 4.3.1, 4.3.3.

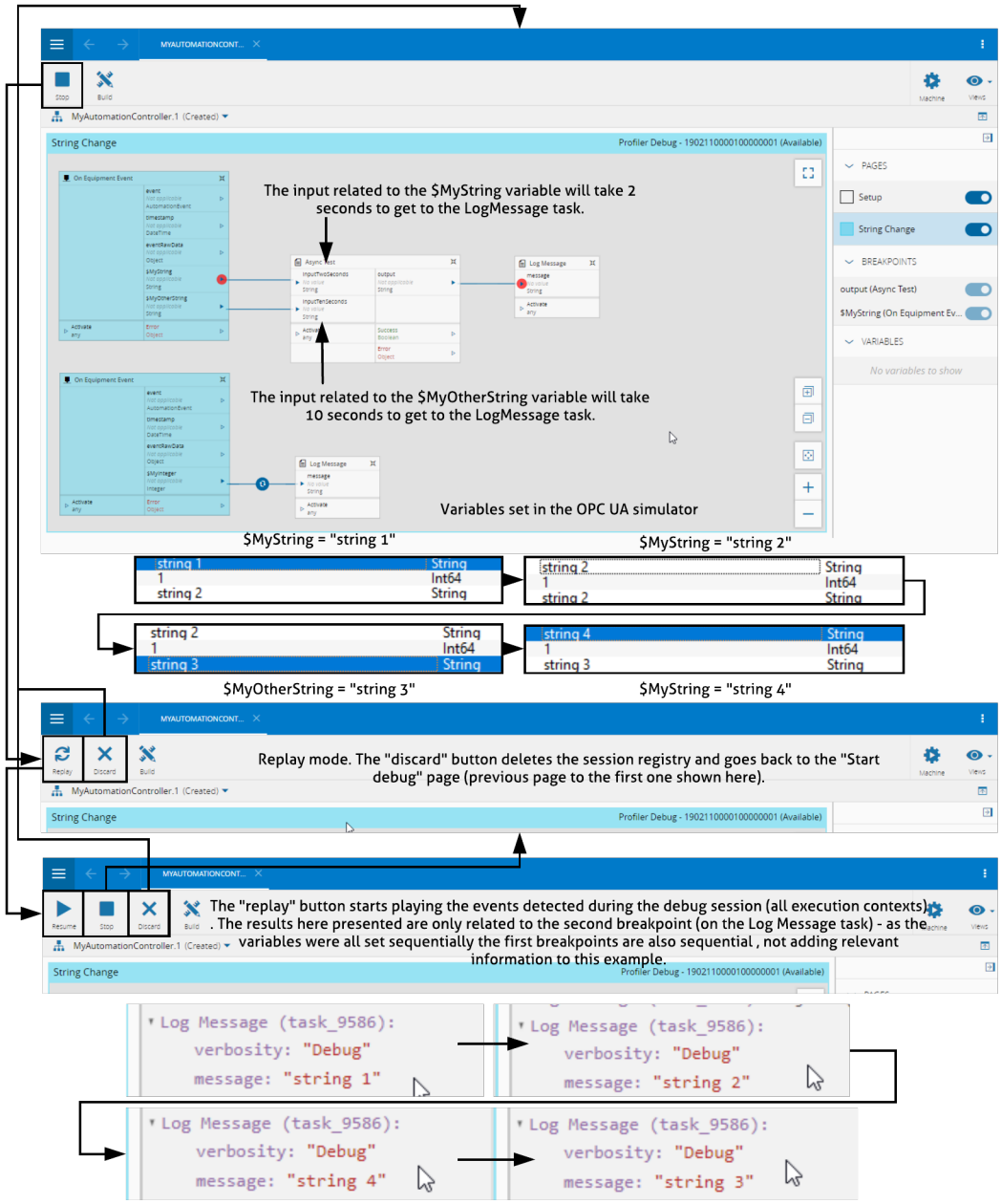

Figure 6.8: **Illustration of the simulated scenario 6.2.9.** Screenshots of the MES interface are presented to show how a profiler session is presented to the user. The registry of all breakpoints activated during the debug session is captured and ordered chronologically, presenting them to the user through the replay mode, after the session is finished.

- **Notes:** This scenario can be seen in figure 6.10.

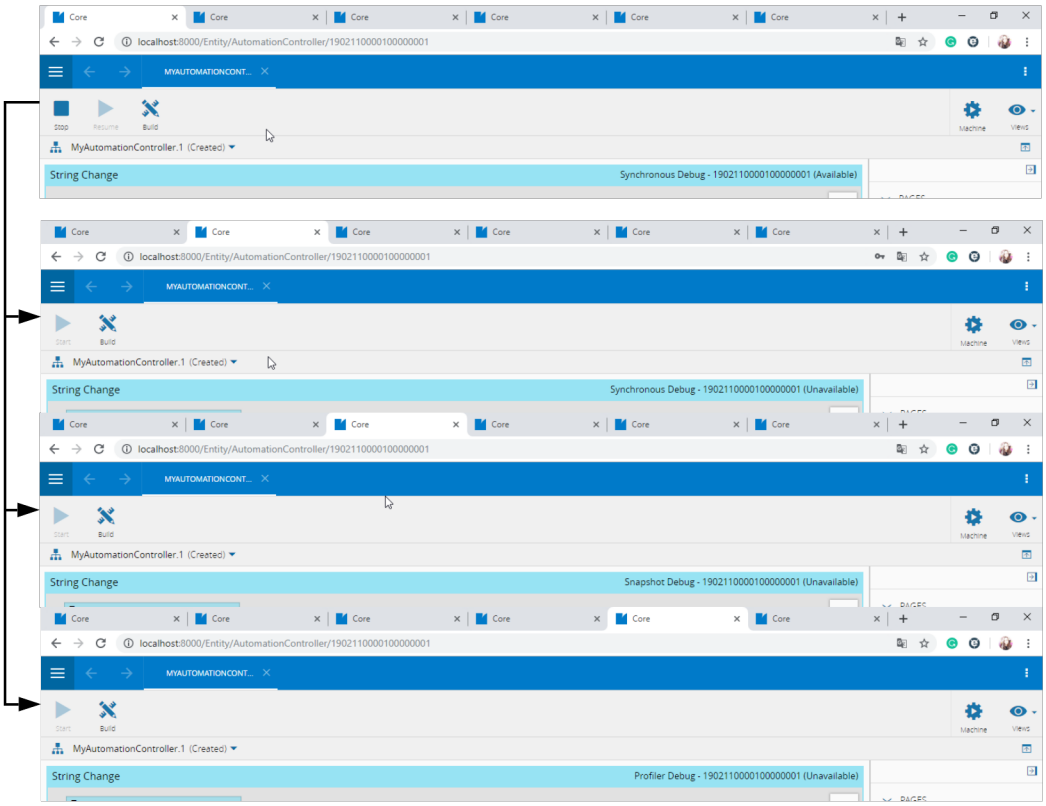

Figure 6.9: **Illustration of the simulated scenario 6.2.10.** Screenshots of the MES interface are presented to show how a synchronous debug session on a Automation Controller instance affects the availability of the other debug modes in that Automation Controller.

6.2.12 Scenario 12 - Synchronous debug session breakpoint changes

- **Scenario:** MES instance allows changing breakpoints during a synchronous session, and they are immediately updated in the Automation Controller instance.
- **Expected Behaviour:** Clicking in the workflow input/outputs adds/removes breakpoints. The debug session will act according to the modifications made (will stop on the newly added breakpoint or will continue execution upon reaching the variable where the breakpoint was removed).
- **Desiderata Items:** 4.3.1, 4.3.2, 4.3.6.

6.2.13 Scenario 13 - Synchronous debug session breakpoint state changes

- **Scenario:** MES instance allows changing the breakpoints active state during a synchronous session, and they are immediately updated in the Automation Controller instance.

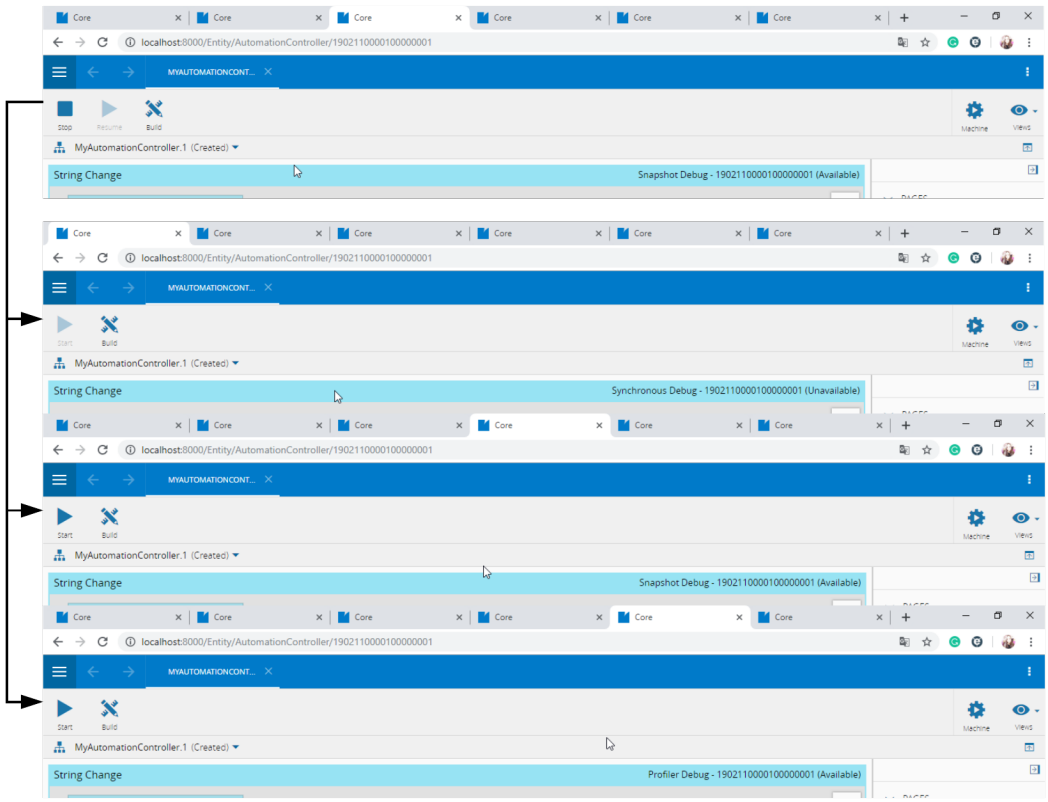

Figure 6.10: **Illustration of the simulated scenario 6.2.11**. Screenshots of the MES interface are presented to show how a snapshot or profiler debug session on a Automation Controller instance affects the availability of the other debug modes in that Automation Controller.

- **Expected Behaviour:** Clicking in the workflow input/outputs enables/disables breakpoints. The debug session will act according to the modifications made (will stop on enabled breakpoints or will continue execution upon reaching disabled breakpoints).
- **Desiderata Items:** 4.3.1, 4.3.2, 4.3.6.

6.2.14 Scenario 14 - Snapshot and profiler debug session breakpoint changes disabled

- **Scenario:** MES instance doesn't allow changing breakpoints during a snapshot/profiler debugging session.
- **Expected Behaviour:** Clicking on a breakpoint whilst in an snapshot or profiler session won't do anything and a warning message will be displayed.
- **Desiderata Items:** 4.3.1, 4.3.2, 4.3.6.
- **Notes:** This scenario can be seen in figure 6.11.

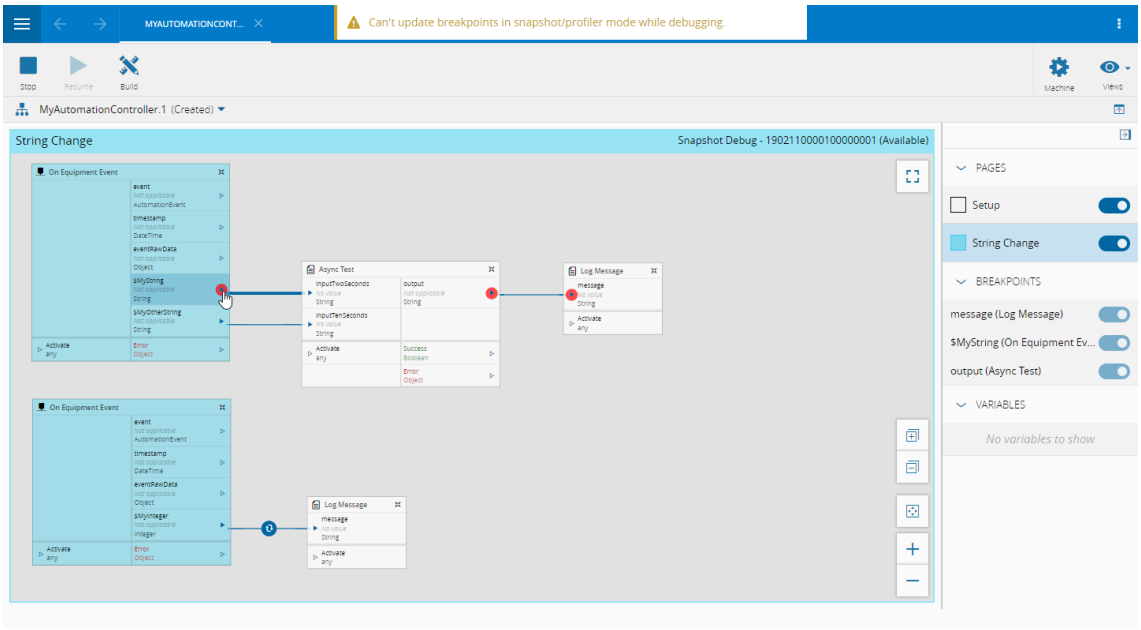

Figure 6.11: **Illustration of the simulated scenario 6.2.14.** Screenshots of the MES interface are presented to show that breakpoint changes are not allowed during a snapshot or profiler debug session and how it is presented to the user through a warning.

6.2.15 Scenario 15 - Debug sessions of the same workflow happening in different machines don't compromise the workflow availability of each other

- **Scenario:** MES instance debugs the same workflow running on different machines (Automation Controller instances) without compromising its availability status. The availability of a workflow is determined for each Automation Controller instance and the debug sessions of the different instances don't interfere with each other.
- **Expected Behaviour:** Since each machine is an individual Automation Controller instance and the workflow executions are independent, it is possible to perform multiple debug sessions for the same workflow (that would normally interfere with the workflow's availability status) but running on different Automation Controller instances. The workflow's availability status is individual for each Automation Controller instance and dependents only on the debug sessions being performed in each.
- **Desiderata Items:** 4.3.3.

6.3 Conclusions

The simulated scenarios demonstrate that the *proof-of-concept* developed is a rather complete solution in terms of debugging IIoT automation systems.

The wide variety of scenarios made for the verification that all the communication interactions between the manufacturing execution system and the equipment(s) it was communicating with

were being performed as expected (Scenarios 6.2.1, 6.2.2, 6.2.3 and 6.2.4), also taking into consideration faulty scenarios. The most critical faulty scenario is the one in which the Automation Controller instance disconnects during a debugging session (Scenario 6.2.4), with the software system maintaining the user informed of every action that has happened through the user interface and reconnecting with the Automation Controller instance as soon as it is available again, notifying the user of the new state of availability upon reconnection.

All scenarios related to the debugging sessions were validated (Scenarios 6.2.5, 6.2.6, 6.2.7, 6.2.8 and 6.2.9) with success, as the prototype behaved according to the expected results in each scenario and all the actions performed corresponded to the requirements of this *proof-of-concept*. The system proved itself to be very capable and able to perform all the debug sessions according to what was stated in Chapter 5.

Scenarios related to the breakpoint management were also simulated (Scenarios 6.2.12, 6.2.13 and 6.2.14), taking into account the selected debug mode, testing if a change in the user interface regarding addition, removal or state changes of breakpoints would affect the next events to be triggered, or not, in the variable where the breakpoint was placed/removed or the state was enabled/disabled.

Lastly, scenarios regarding the workflow's availability state were evaluated to see if it followed the *availability rules* defined in Chapter 5. Regarding the same selected Automation Controller instance:

- Scenario 6.2.10 validated that the workflow was unavailable for incoming debug sessions (being displayed and behaving according to what was stated in the *availability rules*) when a synchronous debug session was already active in that equipment;
- Scenario 6.2.11 validated that the workflow was unavailable for incoming synchronous debug sessions (being displayed and behaving according to what was stated in the *availability rules*) when one or more snapshot or profiler debug sessions were already active in that equipment.

Regarding different Automation Controller instances, Scenario 6.2.15 validated that debugging a workflow running on different equipment, does not affect the workflow's availability for the selected Automation Controller instance.

Judging by the results of the tested scenarios, it can be assumed that the solution was properly implemented and it was possible to design a multi-approach debugging of industrial IoT workflows. The protocol developed to support the solution prototype was able to fulfill the desired objectives, enabling all the functionalities that were described in the solution proposal. This solution ensures the coherence of data at all times and the continuous connection between entities, when required, taking faulty scenarios into account and being able to deal with them, always returning to a stable state.

Chapter 7

Conclusions and Future Work

Contents

This final Chapter of the dissertation presents the main contributions and conclusions of this work, ending with a description of the future work that is planned.

7.1 Summary

From the background and related work explored in this dissertation, regarding Industry 4.0, manufacturing execution systems, software development life cycle and debugging, several conclusions can be made. These conclusions essentially defined the process and approach that was taken to develop this *proof-of-concept*.

Industry 4.0 is revolutionizing the way factories compete in the market, aiming to achieve a higher level of operational efficiency and productivity by augmenting the level of automatization in factories. The integration of IoT in the manufacturing process with embedded sensors and actuators enable the digitalization of shop-floor activities with the collection, analysis, and exchange of the information received in real-time from these devices.

Because of the recent implementation of CPS in factories, an adaptation of the software development life cycle for industries has been crucial for the achievement of smart and efficient manufacturing. Industrial software requires the need to be constantly updated and re-configured, imposing pressure over the need for maintenance of the software deployed in the industrial systems. Real-time monitoring of the production line should also be considered during the maintenance process to allow optimal performance of the manufacturing system at all times, avoiding

machine downtime and detecting execution failures as soon as they happen with suitable debugging systems.

Due to the failures of CPS systems being hard to reproduce due to the non-determinism of concurrent processes, the time-sensitive nature of applications and partial failures that may occur, remote debugging can aid the maintenance process since the debugger is connected to the device when the exception or crash occurs and captures information from the device in real-time. To help understand, identify and fix occurring issues in these type of systems, a remote debugger should be run alongside an abstraction of the different tasks to be executed by the equipment, using visual workflows, to represent the different actions to be performed by the machines in a simplistic way, allowing the common factory worker with close to none IT-knowledge to accurately identify and report system failures as soon as they happen.

An extensive search was done to find different solutions that were already researched and validated in the fields of debugging, remote debugging, debugging of workflows and remote debugging protocols. None of the solutions that were examined seemed complete for the remote debugging of workflows and was adapted for the manufacturing industry. The remote debugging protocols found did also not cover what was necessary for the fulfillment of the defined requirements.

To allow a better understanding of the information obtained through the real-time monitoring of equipment feedback, using visual metaphors as abstraction mechanisms, the demands presented in the following list have to be simultaneously answered:

1. Remote connection to a specific physical device running elsewhere without compromising its current execution;
2. Abstraction of the physical device's sequenced execution tasks through a workflow;
3. Ability to debug the workflow being executed by the physical device in real-time (with or without interrupting its execution).

With these requirements in mind, a functional prototype to demonstrate the feasibility of both synchronized and snapshot remote debugging applied to an IIoT workflow was designed, constructed, and tested/evaluated. A debugging protocol was defined to support all the features that were intended to be implemented in the multi-approach debugging solution proposed, along with how the different system entities will interact throughout the entire debugging process, from the preparation of the debugging environment with the selection of the workflow, debug mode and physical machine to debug, to the actual debugging session. The remote debugging protocol messages and their interaction can be found in Table 5.1. The *proof-of-concept* that was developed is intended to show the advantages of using real-time remote debugging in the industrial context, during the maintenance phase of the deployed software, whilst in production, for minimizing incorrect machinery behavior, as well as machine inactivity due to failures in order to reduce resource losses.

This prototype was developed in the context of Critical Manufacturing's Manufacturing Execution System. A MES is an information system that focuses on the digitalization of shop-floor

activities through the collection, analysis, and exchange of the information captured in real-time during the manufacturing process. Due to the increase in the level of automation in manufacturing systems in recent years, it is understandable why remote debugging applied to IIoT workflows is a desired feature and should be integrated into an already available MES system.

The solution was thoroughly tested using a list of simulated scenarios that verified that every functionality that was developed was correctly implemented and that there weren't any data inconsistencies and the different entities were communicating as intended. By the results obtained, it can be said that the solution was properly implemented and it was possible to design a multi-approach debugging of industrial IoT workflows, fulfilling all the desired objectives and enabling all the functionalities that were described in the solution proposal, also taking faulty scenarios into account.

7.2 Main Contributions

The main contributions of this work are focused on the possibility to use an abstraction of the production machines' execution tasks through workflows to remotely debug them in real-time with real data provided by the sensors and actuators of said machine and how that can help to quickly identify and fix any occurring issues, thus minimizing incorrect machinery behavior and machinery downtime, helping the maintenance process of IIoT systems:

1. Synchronous, snapshot and profiler remote debugging functionalities that support the manufacturing environment's needs;

2. A simplified generalized remote debugging protocol that serves the particularities of each debugging mode implemented (synchronous, snapshot and profiler), taking into consideration all elements that might interfere with their correct execution;

3. An intuitive, user-friendly interface for presenting the debug information, allowing the general factory worker with close to none IT knowledge to operate with it by creating an abstraction of the machine's execution tasks by using a representation through workflows. The simplified interface provides a better understanding of the machine execution and allows to detect in real-time what can be causing incorrect behavior.

Another contribution to be aware of was the development of a scientific article. The article summarized the state of art of this dissertation and the solution proposed, along with how it was implemented and the simulated scenarios that were used for its validation.

7.3 Future Work

As for the future work to improve the current solution, testing it in a real-life environment is required. The impact of its usage in the production environment should be measured to know if there is a better insight of the production machinery, as expected, and incorrect behavior is being

detected more quickly than without this debug solution. This evaluation shall be made to perfect the existing solution and improve its usability and efficiency.

Another improvement that should be done is that when using the Synchronous Debug mode, it should not only be possible to see the values of the variables, but also to change them in real-time while a task's execution is stopped by a breakpoint, affecting the execution of the target device being debugged.

Lastly, another improvement that should be implemented would be to be able to choose the target machine/resource to debug directly through the target's IP address. The target Automation Controller identification should also be more detailed (in the *proof-of-concept* developed, the only identification used for these entities was the randomized ID it is given).

References

[61190] IEC 61131-3. Programmable logic controllers – part 3: Programming languages. In *IEC Standard 61131-3*, 1990.

[AA16] Ephrem Ryan Alphonsus and Mohammad Omar Abdullah. A review on the applications of programmable logic controllers (plcs). *Renewable and Sustainable Energy Reviews*, 60:1185–1205, 2016.

[AFMW17] A. Almohammad, J. F. Ferreira, A. Mendes, and P. White. Reqcap: Hierarchical requirements modeling and test generation for industrial control systems. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 351–358, 2017.

[AGB93] William A. Gruver and Jack Boudreaux. *Intelligent Manufacturing: Programming Environments for CIM*. Springer Verlag, 01 1993.

[AKG+19] M. Aly, F. Khomh, Y. Guéhéneuc, H. Washizaki, and S. Yacout. Is fragmentation a threat to the success of the internet of things? *IEEE Internet of Things Journal*, 6(1):472–487, Feb 2019.

[AL15] Francisco Almada-Lobo. The industry 4.0 revolution and the future of manufacturing execution systems (mes). *Journal of Innovation Management*, 3(4):16–21, 2015.

[ALR00] A. Avizienis, J. C. Laprie, and B. Randell. Fundamental concepts of dependability. In *3rd IEEE Information Survivability Workshop (ISW-2000), Boston, Massachusetts, USA*, pages 7–12, 10 2000.

[Azi19] A. Azizi. Modern manufacturing. In *SpringerBriefs in Applied Sciences and Technology*, pages 7–17. Springer, Singapore, 2019.

[BF14] Pierre Bourque and Richard E. Fairley, editors. *SWEBOK: Guide to the Software Engineering Body of Knowledge*. IEEE Computer Society, version 3.0 edition, 2014.

[BHCW18] Hugh Boyes, Bil Hallaq, Joe Cunningham, and Tim Watson. The industrial internet of things (iiot): An analysis framework. *Computers in Industry*, 101:1 – 12, 2018.

[BVD16] Rajkumar Buyya and Amir Vahid Dastjerdi. *Internet of Things: Principles and Paradigms*, pages 3–23. Morgan Kaufmann, 2016.

[BXW14] Z. Bi, L. D. Xu, and C. Wang. Internet of things for enterprise systems of modern manufacturing. *IEEE Transactions on Industrial Informatics*, 10(2):1537–1546, May 2014.

[BYF03] Mohammed Bani Younis and Georg Frey. Formalization of existing plc programs: A survey. 08 2003.

[Cox08] Philip T. Cox. *Visual Programming Languages*, pages 1–10. American Cancer Society, 2008.

[DFF18] J. P. Dias, J. P. Faria, and H. S. Ferreira. A reactive and model-based approach for developing internet-of-things systems. In *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, pages 276–281, Sep. 2018.

[Doca] Microsoft Docs. Debug apps using visual studio - visual studio. Accessed: 2019-01-31.

[Docb] Microsoft Docs. Debugging your apps - visual studio. Accessed: 2019-01-31.

[Docc] Microsoft Docs. Remote debugging - visual studio. Accessed: 2019-01-31.

[dUAP09] B. Saenz de Ugarte, A. Artiba, and R. Pellerin. Manufacturing execution system – a literature review. *Production Planning & Control*, 20(6):525–539, 2009.

[Fou] Free Software Foundation. Debugging with gdb: Remote debugging. Accessed: 2019-01-31.

[Fra19] Jake Frankenfield. Cloud computing definition, May 2019. Accessed: 2019-06-26.

[FTV02] Filomena Ferrucci, Genoveffa Tortora, and Giuliana Vitiello. *Visual Programming*. American Cancer Society, 2002.

[GD15] M. Geogy and A. Dharani. Prominence of each phase in software development life cycle contributes to the overall quality of a product. In *2015 International Conference on Soft-Computing and Networks Security (ICSNS)*, pages 1–2, 2015.

[GDBa] GDB. Debugging with gdb - the gdb remote serial protocol. Accessed: 2019-02-5.

[GDBb] GDB. Gdb: The gnu project debugger. Accessed: 2019-01-31.

[GDS+11] Dan Gunter, Ewa Deelman, Taghrid Samak, Christopher Brooks, Monte Goode, Gideon Juve, Gaurang Mehta, Priscilla Moraes, Fabio Silva, Martin Swany, and Karan Vahi. Online workflow management and performance analysis with stampede. In *2011 7th International Conference on Network and Service Management, CNSM 2011*, pages 1–10, 01 2011.

[II13] Consortium II. Fact sheet, 2013.

[II19] Consortium II. The industrial internet of things volume g1: Reference architecture, 2019. Accessed: 2019-06-27.

[IML+16a] S. Iarovyi, W. M. Mohammed, A. Lobov, B. R. Ferrer, and J. L. M. Lastra. Cyber–physical systems for open-knowledge-driven manufacturing execution systems. *Proceedings of the IEEE*, 104(5):1142–1154, May 2016.

[IML+16b] S. Iarovyi, W. M. Mohammed, A. Lobov, B. R. Ferrer, and J. L. M. Lastra. Cyber–physical systems for open-knowledge-driven manufacturing execution systems. *Proceedings of the IEEE*, 104(5):1142–1154, 2016.

[Inf18] Infosys. Interoperability between iic architecture & industry 4.0 reference architecture for industrial assets, 2018. Accessed: 2019-06-27.

[Int97] MESA Internacional. Mes functionalities & mrp to mes - data flow possibilities, March 1997. Accessed: 2019-06-20.

[Int06] MESA Internacional. Metrics that matter: Uncovering kpis that justify operational improvements, October 2006. Accessed: 2019-06-20.

[IYMD17] K. S. K. Ibrahim, J. H. Yahaya, Z. Mansor, and A. Deraman. Towards the quality factor of software maintenance process: A review. In *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, volume 9, pages 115–118, 2017.

[KG15] S. K. Khatri and A. Garg. Evolving a risk-free, requirement centric and goal oriented software development life cycle model. In *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, pages 1–6, 2015.

[Lee08] E. A. Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369, May 2008.

[LFK+14] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business and Information Systems Engineering*, 6(4):239–242, 2014.

[LGS17] P. Lade, R. Ghosh, and S. Srinivasan. Manufacturing analytics and industrial internet of things. *IEEE Intelligent Systems*, 32(3):74–79, 2017.

[LJ15] M. Lachwani and Srinivasan J. Remote application debugging, October 2015. US 9,170,922 B1.

[Lu17] Yang Lu. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, 6:1–10, 2017.

[LXWY09] H. Li, Y. Xu, F. Wu, and C Yin. Research of "stub" remote debugging technique. In *2009 4th International Conference on Computer Science Education*, pages 990–994, July 2009.

[LXZ15] Shancang Li, Li Da Xu, and Shanshan Zhao. The internet of things: a survey. *Information Systems Frontiers*, 17(2):243–259, 2015.

[Mana] Critical Manufacturing. Brochure mes v6 generic. Accessed: 2019-02-1.

[Manb] Critical Manufacturing. Critical manufacturing - a brief history of manufacturing execution systems. Accessed: 2019-01-02.

[Manc] Critical Manufacturing. Critical manufacturing - complete modular solution. Accessed: 2019-06-21.

[Mand] Critical Manufacturing. Critical manufacturing - connect iot. Accessed: 2019-02-1.

[Mane] Critical Manufacturing. Critical manufacturing - critical manufacturing mes | infrastructure framework for manufacturing equipment integration, data analysis and business intelligence. Accessed: 2019-02-1.

[Manf] Critical Manufacturing. Critical manufacturing - critical manufacturing mes | integrated manufacturing execution system. Accessed: 2019-02-1.

[Mang] Critical Manufacturing. Critical manufacturing - integration and automation. Accessed: 2019-02-1.

[Manh] Critical Manufacturing. Critical manufacturing - press releases - the future of manufacturing at hannover messe. Accessed: 2019-06-18.

[Mani] Critical Manufacturing. Critical manufacturing - what is mes? Accessed: 2019-01-02.

[Manj] Critical Manufacturing. Critical manufacturing releases v6, new version of its industry 4.0-ready mes: Connect with the future. Accessed: 2019-05-29.

[MBC+17] M. Marra, E. G. Boix, S. Costiou, M. Kerboeuf, A. Plantec, G. Polito, and S. Ducasse. Debugging cyber-physical systems with pharo an experience report. In *IWST 2017 - Proceedings of the 12th International Workshop on Smalltalk Technologies, in conjunction with the 25th International Smalltalk Joint Conference*, 2017.

[MBRB17] Elaheh Maleki, Farouk Belkadi, Mathieu Ritou, and Alain Bernard. A tailored ontology supporting sensor implementation for the maintenance of industrial machines. *Sensors*, 17(9), 2017.

[Mic12] James Mickens. Rivet: Browser-agnostic remote debugging for web applications. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 30–30. USENIX Association, 2012.

[Mom14] H. Momeni. Aspect-oriented software maintainability assessment using adaptive neuro fuzzy inference system (anfis). In *Journal of Mathematics and Computer Science*, volume 12, pages 243–252, 2014.

[MUK00] M. G. Mehrabi, A. G. Ulsoy, and Y. Koren. Reconfigurable manufacturing systems: Key to future manufacturing. *Journal of Intelligent Manufacturing*, 11(4):403–419, Aug 2000.

[MZ16] Pieter J. Mosterman and Justyna Zander. Cyber-physical systems challenges: a needs analysis for collaborating embedded software systems. *Software & Systems Modeling*, 15(1):5–16, Feb 2016.

REFERENCES

[NRa] Node-Red. Node-red. Accessed: 2019-02-5.

[NRb] Node-Red. Node-red: About. Accessed: 2019-02-5.

[oM] University of Melbourne. Software development lifecycle. Accessed: 2019-01-18.

[Oraa] Oracle. Java debug wire protocol. Accessed: 2019-02-5.

[Orab] Oracle. Java platformer debugger architecture. Accessed: 2019-01-30.

[PBF+15] N. Papoulias, N. Bouraqadi, L. Fabresse, Ducasse S., and Denker M. Mercury: Properties and design of a remote debugging solution using reflection. *Journal of Object Technology*, 14(2):1:1–36, May 2015.

[PG18] F. Pereira and L. Gomes. A json/http communication protocol to support the development of distributed cyber-physical systems. In *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, pages 23–30, July 2018.

[PGPM19] U. A. Pozdnyakova, V. V. Golikov, I. A. Peters, and I. A. Morozova. Genesis of the revolutionary transition to industry 4.0 in the 21st century and overview of previous industrial revolutions. In *Studies in Systems, Decision and Control*, volume 169, pages 11–19. Springer International Publishing, 2019.

[PSTH17] Michael Perscheid, Benjamin Siegmund, Marcel Taeumel, and Robert Hirschfeld. Studying the advancement in debugging practice of professional software developers. *Software Quality Journal*, 25(1):83–110, 2017.

[PTP07] Guillaume Pothier, Éric Tanter, and José Piquer. Scalable omniscient debugging. In *Proceedings of the 22Nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications*, OOPSLA '07, pages 535–552. ACM, 2007.

[QZ04] R. G. Qiu and Mengchu Zhou. Mighty mess - state-of-the-art and future manufacturing execution systems. *IEEE Robotics Automation Magazine*, 11(1):19–25, March 2004.

[Ras14] Vanshika Rastogi. Software development life cycle models - comparison, consequences. In *International Journal of Computer Science and Information Technologies (IJCSIT), Vol. 6*, pages 168–172, 2014.

[RLSS10] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic. Cyber-physical systems: The next computing revolution. In *Design Automation Conference*, pages 731–736, June 2010.

[RMK16] Vasja Roblek, Maja Meško, and Alojz Krapež. A complex view of industry 4.0. *SAGE Open*, 6(2):2158244016653987, 2016.

[RWM+18] D. Rathnayake, A. Wickramarachchi, V. Mallawaarachchi, D. Meedeniya, and I. Perera. A realtime monitoring platform for workflow subroutines. In *2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer)*, pages 41–47, 2018.

[Sar03] Roland B. Sargeant. *Functional Specifiction of a Manufacturing Execution System*. PhD thesis, Massachusetts Institute of Technology, 2003.

[SBAM17] S. Soomro, M. R. Belgaum, Z. Alansari, and M. H. Miraz. Fault localization models in debugging. In *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*, pages 57–62, Dec 2017.

[SBP+06] A. J. Sagar, A. G. Bhandarkar, D. Pilarinos, D. K. Shukla, M. Mehta, S. Chub, and V. Kalra. Workflow debugger, December 2006. US 2006/0288332 A1.

[Scu] Padraig Scully. The top 10 iot segments in 2018 – based on 1,600 real iot projects. Accessed: 2019-06-18.

[SD17] S. Srivastva and S. Dhir. Debugging approaches on various software processing levels. In *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, volume 2, pages 302–306, 2017.

[Siv14] A. Sivieri. Eliot: A programming framework for the internet of things, 2014. PhD thesis, politecnico di milano.

[SM98] M. J. C. Sousa and H. M. Moreira. A survey on the software maintenance process. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pages 265–274, Nov 1998.

[SMC16] A. Sivieri, L. Mottola, and G. Cugola. Building internet of things software with eliot. *Computer Communications*, 89-90:141 – 153, 2016. Internet of Things: Research challenges and Solutions.

[SP07] R. Schmitt and Wagner P. Method for debugging flowchart programs for industrial controllers, November 2007. US 7,302.676 B2.

[SS17] Pallavi Sethi and Smruti R. Sarangi. Internet of things: Architectures, protocols, and applications. *Journal of Electrical and Computer Engineering*, 2017:25, 2017.

[SSH+18] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund. Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics*, 14(11):4724–4734, 2018.

[SWYS11] Jianhua Shi, Jiafu Wan, Hehua Yan, and Hui Suo. A survey of cyber physical systems. *Proc. of the Int. Conf. on Wireless Communications and Signal Processing*, 11 2011.

[Sza19] Andrea Szalavetz. Industry 4.0 and capability development in manufacturing subsidiaries. *Technological Forecasting and Social Change*, 145:384 – 395, 2019.

[Tec] Tecon. Opc ua server. Accessed: 2019-05-02.

[TI19] H. Tanno and H. Iwasaki. Suspend-less debugging for interactive and/or realtime programs. *12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, April 2019.

[TS16] Lane Thames and Dirk Schaefer. Software-defined cloud manufacturing for industry 4.0. *Procedia CIRP*, 52:12 – 17, 2016. The Sixth International Conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV2016).

[VHFR+15] B. Vogel-Heuser, J. Fischer, S. Rösch, S. Feldmann, and S. Ulewicz. Challenges for maintenance of plc-software and its related hardware for automated production systems: Selected industrial case studies. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 362–371, 2015.

[Vola] Volron.js. documentation. Accessed: 2019-01-31.

[Volb] Volron.js. Volron.js. Accessed: 2019-01-31.

[WADX15] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. The internet of things—a survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274, 2015.

[WDDZ11] C. Wenyu, H. Dongpu, T. Dongcheng, and H. Zongbo. A model of remote debugger supporting multiple types of connection. In *2011 International Conference on Electronics, Communications and Control (ICECC)*, pages 642–645, Sep. 2011.

[WKYBWYF12] Loo Wooi Khong, Leau Yu Beng, Tham Wai Yip, and Tan Fun. Software development life cycle agile vs traditional approaches. In *2012 International Conference on Information and Network Technology (ICINT 2012)*, 02 2012.

[WTO15] Lihui Wang, Martin Törngren, and Mauro Onori. Current status and advancement of cyber-physical systems in manufacturing. *Journal of Manufacturing Systems*, 37:517 – 527, 2015.

[WWB18] S. Waschull, J. C. Wortmann, and J. A. C. Bokhorst. Manufacturing execution systems: The next level of automated control or of shop-floor support? In Ilkyeong Moon, Gyu M. Lee, Jinwoo Park, Dimitris Kiritsis, and Gregor von Cieminski, editors, *Advances in Production Management Systems. Smart Manufacturing for Industry 4.0*, pages 386–393. Springer International Publishing, 2018.

[WWIT16] Thorsten Wuest, D Weimer, Chris Irgens, and Klaus-Dieter Thoben. Machine learning in manufacturing: Advantages, challenges, and applications. *Production & Manufacturing Research*, 4:23–45, 06 2016.

[Zen] Zenodys. Zenodys native computing engine. Accessed: 2019-06-09.

[Zer] ZeroTurnaround. Jrebel - jrebel 2018.x documentation. Accessed: 2019-01-30.